# Isabelle/HOL — Higher-Order Logic

December 17, 2025

## Contents


**1 Loading the code generator and related modules 33**

**2 The basis of Higher-Order Logic 34**

2.1 Primitive logic . . . 34
2.1.1 Core syntax . . . 34
2.1.2 Defined connectives and quantifiers . . . 35
2.1.3 Additional concrete syntax . . . 35
2.1.4 Axioms and basic definitions . . . 37
2.2 Fundamental rules . . . 38
2.2.1 Equality . . . 38
2.2.2 Congruence rules for application . . . 39
2.2.3 Equality of booleans – iff . . . 39
2.2.4 True (1) . . . 39
2.2.5 Universal quantifier (1) . . . 40
2.2.6 False . . . 40
2.2.7 Negation . . . 40
2.2.8 Implication . . . 40
2.2.9 Disjunction (1) . . . 41
2.2.10 Derivation of *iffI* . . . 41
2.2.11 True (2) . . . 41
2.2.12 Universal quantifier (2) . . . 42
2.2.13 Existential quantifier . . . 42
2.2.14 Conjunction . . . 42
2.2.15 Disjunction (2) . . . 42
2.2.16 Classical logic . . . 42
2.2.17 Unique existence . . . 43
2.2.18 Classical intro rules for disjunction and existential quantifiers . . . 44
2.2.19 Intuitionistic Reasoning . . . 45
2.2.20 Atomizing meta-level connectives . . . 45
2.2.21 Atomizing elimination rules . . . 46

2.3 Package setup . . . . . . . . . . 46
2.3.1 Sledgehammer setup . . . . . . . . . . 46
2.3.2 Classical Reasoner setup . . . . . . . . . . 46
2.3.3 THE: definite description operator . . . . . . . . . . 48
2.3.4 Simplifier . . . . . . . . . . 48
2.3.5 Generic cases and induction . . . . . . . . . . 56
2.3.6 Coherent logic . . . . . . . . . . 58
2.3.7 Reorienting equalities . . . . . . . . . . 58
2.4 Other simple lemmas and lemma duplicates . . . . . . . . . . 58
2.5 Basic ML bindings . . . . . . . . . . 58

**3 *NO-MATCH* simproc 60**
3.1 Code generator setup . . . . . . . . . . 60
3.1.1 Generic code generator preprocessor setup . . . . . . . . . . 60
3.1.2 Generic code generator foundation . . . . . . . . . . 61
3.1.3 Equality . . . . . . . . . . 62
3.1.4 Generic code generator target languages . . . . . . . . . . 62
3.1.5 Evaluation and normalization by evaluation . . . . . . . . . . 64
3.2 Counterexample Search Units . . . . . . . . . . 64
3.2.1 Quickcheck . . . . . . . . . . 64
3.2.2 Nitpick setup . . . . . . . . . . 64
3.3 Preprocessing for the predicate compiler . . . . . . . . . . 64
3.4 Legacy tactics and ML bindings . . . . . . . . . . 64

**4 Abstract orderings 64**
4.1 Abstract ordering . . . . . . . . . . 65
4.2 Syntactic orders . . . . . . . . . . 67
4.3 Quasi orders . . . . . . . . . . 68
4.4 Partial orders . . . . . . . . . . 69
4.5 Linear (total) orders . . . . . . . . . . 71
4.6 Reasoning tools setup . . . . . . . . . . 73
4.7 Bounded quantifiers . . . . . . . . . . 74
4.8 Transitivity reasoning . . . . . . . . . . 76
4.9 min and max – fundamental . . . . . . . . . . 80
4.10 (Unique) top and bottom elements . . . . . . . . . . 81
4.11 Dense orders . . . . . . . . . . 82
4.12 Wellorders . . . . . . . . . . 83
4.13 Order on *bool* . . . . . . . . . . 84
4.14 Order on - $\Rightarrow$ - . . . . . . . . . . 85
4.15 Order on unary and binary predicates . . . . . . . . . . 87
4.16 Name duplicates . . . . . . . . . . 87

**5 Groups, also combined with orderings** **88**
5.1 Dynamic facts . . . 88
5.2 Abstract structures . . . 89
5.3 Generic operations . . . 90
5.4 Semigroups and Monoids . . . 91
5.5 Groups . . . 95
5.6 (Partially) Ordered Groups . . . 98
5.7 Support for reasoning about signs . . . 100
5.8 Canonically ordered monoids . . . 108
5.9 Tools setup . . . 110

**6 Abstract lattices** **111**
6.1 Abstract semilattice . . . 111
6.2 Syntactic infimum and supremum operations . . . 113
6.3 Concrete lattices . . . 113
6.3.1 Intro and elim rules . . . 114
6.3.2 Equational laws . . . 115
6.3.3 Strict order . . . 117
6.4 Distributive lattices . . . 118
6.5 Bounded lattices . . . 118
6.6 $min/max$ as special case of lattice . . . 120
6.7 Uniqueness of inf and sup . . . 121
6.8 Lattice on - $\Rightarrow$ - . . . 122

**7 Boolean Algebras** **123**
7.1 Abstract boolean algebra . . . 123
7.1.1 Complement . . . 123
7.1.2 Conjunction . . . 124
7.1.3 Disjunction . . . 124
7.1.4 De Morgan's Laws . . . 125
7.2 Symmetric Difference . . . 125
7.3 Type classes . . . 126
7.4 Lattice on *bool* . . . 128
7.5 Lattice on unary and binary predicates . . . 129
7.6 Simproc setup . . . 130

**8 Set theory for higher-order logic** **131**
8.1 Sets as predicates . . . 131
8.2 Subsets and bounded quantifiers . . . 134
8.3 Basic operations . . . 138
8.3.1 Subsets . . . 138
8.3.2 Equality . . . 139
8.3.3 The empty set . . . 140
8.3.4 The universal set – UNIV . . . 140

8.3.5 The Powerset operator – Pow . . . . . . . . . . . . . . . 141
8.3.6 Set complement . . . . . . . . . . . . . . . . . . . . . . . 141
8.3.7 Binary intersection . . . . . . . . . . . . . . . . . . . . . 142
8.3.8 Binary union . . . . . . . . . . . . . . . . . . . . . . . . 142
8.3.9 Set difference . . . . . . . . . . . . . . . . . . . . . . . . 143
8.3.10 Augmenting a set – *insert* . . . . . . . . . . . . . . . . . 143
8.3.11 Singletons, using insert . . . . . . . . . . . . . . . . . . . 144
8.3.12 Image of a set under a function . . . . . . . . . . . . . . 145
8.3.13 Some rules with *if* . . . . . . . . . . . . . . . . . . . . . 148
8.4 Further operations and lemmas . . . . . . . . . . . . . . . . . 149
8.4.1 The "proper subset" relation . . . . . . . . . . . . . . . 149
8.4.2 Derived rules involving subsets. . . . . . . . . . . . . . . 150
8.4.3 Equalities involving union, intersection, inclusion, etc. 151
8.4.4 Monotonicity of various operations . . . . . . . . . . . 161
8.4.5 Inverse image of a function . . . . . . . . . . . . . . . . 162
8.4.6 Singleton sets . . . . . . . . . . . . . . . . . . . . . . . . 164
8.4.7 Getting the contents of a singleton set . . . . . . . . . 164
8.4.8 Monad operation . . . . . . . . . . . . . . . . . . . . . . 164
8.4.9 Operations for execution . . . . . . . . . . . . . . . . . . 165

**9 HOL type definitions 168**

**10 Notions about functions 169**
10.1 The Identity Function *id* . . . . . . . . . . . . . . . . . . . . 169
10.2 The Composition Operator $f \circ g$ . . . . . . . . . . . . . . . . . 170
10.3 The Forward Composition Operator *fcomp* . . . . . . . . . . . 171
10.4 Mapping functions . . . . . . . . . . . . . . . . . . . . . . . . 171
10.5 Injectivity and Bijectivity . . . . . . . . . . . . . . . . . . . . 172
10.5.1 Inj/surj/bij of Algebraic Operations . . . . . . . . . . 180
10.6 Function Updating . . . . . . . . . . . . . . . . . . . . . . . . 182
10.7 *override-on* . . . . . . . . . . . . . . . . . . . . . . . . . . . 183
10.8 Inversion of injective functions . . . . . . . . . . . . . . . . . 184
10.9 Monotonicity . . . . . . . . . . . . . . . . . . . . . . . . . . . 185
10.9.1 Specializations For *ord* Type Class And More . . . . . 186
10.9.2 Least value operator . . . . . . . . . . . . . . . . . . . 191
10.10Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 192
10.10.1 Proof tools . . . . . . . . . . . . . . . . . . . . . . . . 192
10.10.2 Functorial structure of types . . . . . . . . . . . . . . 192

**11 Complete lattices 192**
11.1 Syntactic infimum and supremum operations . . . . . . . . . 192
11.2 Abstract complete lattices . . . . . . . . . . . . . . . . . . . . 194
11.3 Complete lattice on *bool* . . . . . . . . . . . . . . . . . . . . . 202
11.4 Complete lattice on - ⇒ - . . . . . . . . . . . . . . . . . . . . . 203

11.5 Complete lattice on unary and binary predicates . . . . . . . 204
11.6 Complete lattice on - *set* . . . . . . . . . . . . . . . . . . . . . 205
11.6.1 Inter . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206
11.6.2 Intersections of families . . . . . . . . . . . . . . . . . 207
11.6.3 Union . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209
11.6.4 Unions of families . . . . . . . . . . . . . . . . . . . . 210
11.6.5 Distributive laws . . . . . . . . . . . . . . . . . . . . . 213
11.7 Injections and bijections . . . . . . . . . . . . . . . . . . . . . 214
11.7.1 Complement . . . . . . . . . . . . . . . . . . . . . . . . 215
11.7.2 Miniscoping and maxiscoping . . . . . . . . . . . . . . 215

**12 Wrapping Existing Freely Generated Type's Constructors 217**

**13 Knaster-Tarski Fixpoint Theorem and inductive definitions 217**
13.1 Least fixed points . . . . . . . . . . . . . . . . . . . . . . . . . 218
13.2 General induction rules for least fixed points . . . . . . . . . . 218
13.3 Greatest fixed points . . . . . . . . . . . . . . . . . . . . . . . 219
13.4 Coinduction rules for greatest fixed points . . . . . . . . . . . 220
13.5 Even Stronger Coinduction Rule, by Martin Coen . . . . . . . 221
13.6 Rules for fixed point calculus . . . . . . . . . . . . . . . . . . 221
13.7 Inductive predicates and sets . . . . . . . . . . . . . . . . . . 222
13.8 The Schroeder-Bernstein Theorem . . . . . . . . . . . . . . . 222
13.9 Inductive datatypes and primitive recursion . . . . . . . . . . 223

**14 Cartesian products 223**
14.1 *bool* is a datatype . . . . . . . . . . . . . . . . . . . . . . . . 223
14.2 The *unit* type . . . . . . . . . . . . . . . . . . . . . . . . . . . 224
14.3 The product type . . . . . . . . . . . . . . . . . . . . . . . . . 227
14.3.1 Type definition . . . . . . . . . . . . . . . . . . . . . . . 227
14.3.2 Tuple syntax . . . . . . . . . . . . . . . . . . . . . . . . 228
14.3.3 Code generator setup . . . . . . . . . . . . . . . . . . . 229
14.3.4 Fundamental operations and properties . . . . . . . . 229
14.3.5 Derived operations . . . . . . . . . . . . . . . . . . . . 234
14.4 Code generator setup for paired and tripled bounded set comprehension . . . . . . . . . . . . . . . . . . . . . . . . . . . 243
14.5 Simproc for rewriting a set comprehension into a pointfree expression . . . . . . . . . . . . . . . . . . . . . . . . . . . 244
14.6 Lemmas about disjointness . . . . . . . . . . . . . . . . . . . 244
14.7 Inductively defined sets . . . . . . . . . . . . . . . . . . . . . 244
14.8 Legacy theorem bindings and duplicates . . . . . . . . . . . . 244

**15 The Disjoint Sum of Two Types** **244**
15.1 Construction of the sum type and its basic abstract operations 244
15.2 Projections . . . 247
15.3 The Disjoint Sum of Sets . . . 247

**16 Rings** **248**
16.1 Semirings and rings . . . 248
16.2 Abstract divisibility . . . 250
16.3 Towards integral domains . . . 255
16.4 (Partial) Division . . . 258
16.5 Quotient and remainder in integral domains . . . 271
16.6 Interlude: basic tool support for algebraic and arithmetic calculations . . . 272
16.7 Ordered semirings and rings . . . 272
16.8 Dioids . . . 288

**17 Natural numbers** **288**
17.1 Type *ind* . . . 288
17.2 Type nat . . . 288
17.3 Arithmetic operators . . . 291
17.3.1 Addition . . . 292
17.3.2 Difference . . . 293
17.3.3 Multiplication . . . 293
17.4 Orders on *nat* . . . 294
17.4.1 Operation definition . . . 294
17.4.2 Introduction properties . . . 295
17.4.3 Elimination properties . . . 296
17.4.4 Inductive (?) properties . . . 296
17.4.5 Monotonicity of Addition . . . 300
17.4.6 *min* and *max* . . . 301
17.4.7 Additional theorems about ($\leq$) . . . 303
17.4.8 More results about difference . . . 307
17.4.9 Monotonicity of multiplication . . . 308
17.5 Natural operation of natural numbers on functions . . . 310
17.6 Kleene iteration . . . 312
17.7 Embedding of the naturals into any *semiring-1*: *of-nat* . . . 313
17.8 The set of natural numbers . . . 316
17.9 Further arithmetic facts concerning the natural numbers . . . 317
17.9.1 Greatest operator . . . 322
17.10 Monotonicity of *funpow* . . . 322
17.11 Kleene's fixed point theorem for continuous functions . . . 323
17.12 The divides relation on *nat* . . . 323
17.13 Aliasses . . . 325
17.14 Size of a datatype value . . . 326

17.15 Code module namespace . . . . . . . . . . . . . . . . . . . . . 326

**18 Fields** **326**

18.1 Division rings . . . . . . . . . . . . . . . . . . . . . . . . . . . 326

18.2 Fields . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 331

18.3 Ordered fields . . . . . . . . . . . . . . . . . . . . . . . . . . . 334

**19 Relations – as sets of pairs, and binary predicates** **346**

19.1 Fundamental . . . . . . . . . . . . . . . . . . . . . . . . . . . 347

19.1.1 Relations as sets of pairs . . . . . . . . . . . . . . . . . . 347

19.1.2 Conversions between set and predicate relations . . . . 347

19.2 Properties of relations . . . . . . . . . . . . . . . . . . . . . . 349

19.2.1 Reflexivity . . . . . . . . . . . . . . . . . . . . . . . . . 349

19.2.2 Irreflexivity . . . . . . . . . . . . . . . . . . . . . . . . . 351

19.2.3 Asymmetry . . . . . . . . . . . . . . . . . . . . . . . . . 353

19.2.4 Symmetry . . . . . . . . . . . . . . . . . . . . . . . . . . 355

19.2.5 Antisymmetry . . . . . . . . . . . . . . . . . . . . . . . . 357

19.2.6 Transitivity . . . . . . . . . . . . . . . . . . . . . . . . . 360

19.2.7 Totality . . . . . . . . . . . . . . . . . . . . . . . . . . . 363

19.2.8 Left uniqueness . . . . . . . . . . . . . . . . . . . . . . . 365

19.2.9 Right uniqueness . . . . . . . . . . . . . . . . . . . . . . 366

19.3 Relation operations . . . . . . . . . . . . . . . . . . . . . . . . 367

19.3.1 The identity relation . . . . . . . . . . . . . . . . . . . . 367

19.3.2 Diagonal: identity over a set . . . . . . . . . . . . . . . 368

19.3.3 Composition . . . . . . . . . . . . . . . . . . . . . . . . . 369

19.3.4 Converse . . . . . . . . . . . . . . . . . . . . . . . . . . . 371

19.3.5 Domain, range and field . . . . . . . . . . . . . . . . . . 374

19.3.6 Image of a set under a relation . . . . . . . . . . . . . . 377

19.3.7 Inverse image . . . . . . . . . . . . . . . . . . . . . . . . 379

19.3.8 Powerset . . . . . . . . . . . . . . . . . . . . . . . . . . . 380

**20 Finite sets** **380**

20.1 Predicate for finite sets . . . . . . . . . . . . . . . . . . . . . . 380

20.1.1 Choice principles . . . . . . . . . . . . . . . . . . . . . . 381

20.1.2 Finite sets are the images of initial segments of natural numbers . . . . . . . . . . . . . . . . . . . . . . . . . . 381

20.2 Finiteness and common set operations . . . . . . . . . . . . . 381

20.3 Further induction rules on finite sets . . . . . . . . . . . . . . 386

20.4 Class *finite* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 387

20.5 A basic fold functional for finite sets . . . . . . . . . . . . . . 388

20.5.1 From *fold-graph* to *fold* . . . . . . . . . . . . . . . . . . 389

20.5.2 Liftings to *comp-fun-commute-on* etc. . . . . . . . . . . 392

20.5.3 *UNIV* as carrier set . . . . . . . . . . . . . . . . . . . . 392

20.5.4 Expressing set operations via *fold* . . . . . . . . . . . . 393

20.5.5 Expressing relation operations via *fold* . . . . . . . . . 396
20.6 Locales as mini-packages for fold operations . . . . . . . . . . 397
20.6.1 The natural case . . . . . . . . . . . . . . . . . . . . . . 397
20.6.2 With idempotency . . . . . . . . . . . . . . . . . . . . . 397
20.6.3 *UNIV* as the carrier set . . . . . . . . . . . . . . . . . . 398
20.7 Finite cardinality . . . . . . . . . . . . . . . . . . . . . . . . . 399
20.7.1 Cardinality of image . . . . . . . . . . . . . . . . . . . . 404
20.7.2 Pigeonhole Principles . . . . . . . . . . . . . . . . . . . 406
20.7.3 Cardinality of sums . . . . . . . . . . . . . . . . . . . . 406
20.8 Minimal and maximal elements of finite sets . . . . . . . . . . 406
20.8.1 Finite orders . . . . . . . . . . . . . . . . . . . . . . . . 407
20.8.2 Relating injectivity and surjectivity . . . . . . . . . . 408
20.9 Infinite Sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . 408
20.10 The finite powerset operator . . . . . . . . . . . . . . . . . . 410

**21 Reflexive and Transitive closure of a relation 410**
21.1 Reflexive closure . . . . . . . . . . . . . . . . . . . . . . . . . 412
21.2 Reflexive-transitive closure . . . . . . . . . . . . . . . . . . . 413
21.3 Transitive closure . . . . . . . . . . . . . . . . . . . . . . . . 417
21.4 Symmetric closure . . . . . . . . . . . . . . . . . . . . . . . . 424
21.5 The power operation on relations . . . . . . . . . . . . . . . . 424
21.6 Bounded transitive closure . . . . . . . . . . . . . . . . . . . 430
21.7 Acyclic relations . . . . . . . . . . . . . . . . . . . . . . . . . 430
21.8 Setup of transitivity reasoner . . . . . . . . . . . . . . . . . . 431

**22 Well-founded Recursion 431**
22.1 Basic Definitions . . . . . . . . . . . . . . . . . . . . . . . . . 432
22.2 Equivalence of Definitions . . . . . . . . . . . . . . . . . . . . 432
22.3 Induction Principles . . . . . . . . . . . . . . . . . . . . . . . 432
22.4 Introduction Rules . . . . . . . . . . . . . . . . . . . . . . . . 433
22.5 Ordering Properties . . . . . . . . . . . . . . . . . . . . . . . 433
22.6 Basic Results . . . . . . . . . . . . . . . . . . . . . . . . . . . 434
22.6.1 Minimal-element characterization of well-foundedness 435
22.6.2 Finite characterization of well-foundedness . . . . . . 435
22.6.3 Antimonotonicity . . . . . . . . . . . . . . . . . . . . . . 435
22.6.4 Equivalence between *wfp-on* and *wfp* . . . . . . . . . . 436
22.6.5 Well-foundedness of transitive closure . . . . . . . . . 436
22.6.6 Well-foundedness of image . . . . . . . . . . . . . . . . 438
22.7 Well-Foundedness Results for Unions . . . . . . . . . . . . . . 438
22.8 Well-Foundedness of Composition . . . . . . . . . . . . . . . . 439
22.9 Acyclic relations . . . . . . . . . . . . . . . . . . . . . . . . . 439
22.9.1 Wellfoundedness of finite acyclic relations . . . . . . . 440
22.10 *nat* is well-founded . . . . . . . . . . . . . . . . . . . . . . . 440
22.11 Accessible Part . . . . . . . . . . . . . . . . . . . . . . . . . . 441

22.12 Tools for building wellfounded relations . . . . . . . . . . . . . 442
22.12.1 Conversion to a known well-founded relation . . . . . 443
22.12.2 Measure functions into *nat* . . . . . . . . . . . . . . . . 443
22.12.3 Lexicographic combinations . . . . . . . . . . . . . . . 444
22.12.4 Bounded increase must terminate . . . . . . . . . . . . 446
22.13 Code Generation Setup . . . . . . . . . . . . . . . . . . . . . 446

**23 Well-Founded Recursion Combinator 447**
23.0.1 Well-founded recursion via genuine fixpoints . . . . . . 448
23.1 Wellfoundedness of *same-fst* . . . . . . . . . . . . . . . . . . 448

**24 Orders as Relations 448**
24.1 Orders on a set . . . . . . . . . . . . . . . . . . . . . . . . . 448
24.2 Orders on the field . . . . . . . . . . . . . . . . . . . . . . . 450
24.3 Relations given by a predicate and the field . . . . . . . . . . 451
24.4 Orders on a type . . . . . . . . . . . . . . . . . . . . . . . . 451
24.5 Order-like relations . . . . . . . . . . . . . . . . . . . . . . . 452
24.5.1 Auxiliaries . . . . . . . . . . . . . . . . . . . . . . . . 452
24.5.2 The upper and lower bounds operators . . . . . . . . . 452
24.6 Variations on Well-Founded Relations . . . . . . . . . . . . . 455
24.6.1 Characterizations of well-foundedness . . . . . . . . . 455
24.6.2 Characterizations of well-foundedness . . . . . . . . . 455

**25 Hilbert's Epsilon-Operator and the Axiom of Choice 456**
25.1 Hilbert's epsilon . . . . . . . . . . . . . . . . . . . . . . . . 456
25.2 Hilbert's Epsilon-operator . . . . . . . . . . . . . . . . . . . 456
25.3 Axiom of Choice, Proved Using the Description Operator . . 457
25.4 Getting an element of a nonempty set . . . . . . . . . . . . . 458
25.5 Function Inverse . . . . . . . . . . . . . . . . . . . . . . . . 458
25.6 Other Consequences of Hilbert's Epsilon . . . . . . . . . . . 463
25.7 An aside: bounded accessible part . . . . . . . . . . . . . . . 464
25.8 More on injections, bijections, and inverses . . . . . . . . . . 464
25.9 Specification package – Hilbertized version . . . . . . . . . . 466
25.10 Complete Distributive Lattices – Properties depending on Hilbert Choice . . . . . . . . . . . . . . . . . . . . . . . . . 466

**26 Zorn's Lemma and the Well-ordering Theorem 468**
26.1 Zorn's Lemma for the Subset Relation . . . . . . . . . . . . . 468
26.1.1 Results that do not require an order . . . . . . . . . . 468
26.1.2 Hausdorff's Maximum Principle . . . . . . . . . . . . . 471
26.1.3 Results for the proper subset relation . . . . . . . . . 471
26.1.4 Zorn's lemma . . . . . . . . . . . . . . . . . . . . . . . 472
26.2 Zorn's Lemma for Partial Orders . . . . . . . . . . . . . . . . 472
26.3 Other variants of Zorn's Lemma . . . . . . . . . . . . . . . . 473

26.4 The Well Ordering Theorem . . . . . . . . . . . . . . . . . . . 474

**27 Well-Order Relations as Needed by Bounded Natural Functors 476**

27.1 Auxiliaries . . . . . . . . . . . . . . . . . . . . . . . . . . . 476
27.2 Well-founded induction and recursion adapted to non-strict well-order relations . . . . . . . . . . . . . . . . . . . . . . . . . 477
27.3 The notions of maximum, minimum, supremum, successor and order filter . . . . . . . . . . . . . . . . . . . . . . . . . 478
27.3.1 Properties of max2 . . . . . . . . . . . . . . . . . . . . 478
27.3.2 Existence and uniqueness for isMinim and well-definedness of minim . . . . . . . . . . . . . . . . . . . . . . . . . 479
27.3.3 Properties of minim . . . . . . . . . . . . . . . . . . . 479
27.3.4 Properties of successor . . . . . . . . . . . . . . . . . . 480
27.3.5 Properties of order filters . . . . . . . . . . . . . . . . 480
27.3.6 Other properties . . . . . . . . . . . . . . . . . . . . . 481

**28 Well-Order Embeddings as Needed by Bounded Natural Functors 481**

28.1 Auxiliaries . . . . . . . . . . . . . . . . . . . . . . . . . . . 482
28.2 (Well-order) embeddings, strict embeddings, isomorphisms and order-compatible functions . . . . . . . . . . . . . . . . . 482
28.3 Given any two well-orders, one can be embedded in the other 484
28.4 Uniqueness of embeddings . . . . . . . . . . . . . . . . . . . 485
28.5 More properties of embeddings, strict embeddings and isomorphisms . . . . . . . . . . . . . . . . . . . . . . . . . . . 486

**29 Constructions on Wellorders as Needed by Bounded Natural Functors 488**

29.1 Restriction to a set . . . . . . . . . . . . . . . . . . . . . . . 489
29.2 Order filters versus restrictions and embeddings . . . . . . . . 490
29.3 The strict inclusion on proper ofilters is well-founded . . . . . 491
29.4 Ordering the well-orders by existence of embeddings . . . . . 491
29.5 $<o$ is well-founded . . . . . . . . . . . . . . . . . . . . . . 496
29.6 Copy via direct images . . . . . . . . . . . . . . . . . . . . . 497
29.7 Bounded square . . . . . . . . . . . . . . . . . . . . . . . . . 498

**30 Cardinal-Order Relations as Needed by Bounded Natural Functors 501**

30.1 Cardinal orders . . . . . . . . . . . . . . . . . . . . . . . . . 502
30.2 Cardinal of a set . . . . . . . . . . . . . . . . . . . . . . . . 503
30.3 Cardinals versus set operations on arbitrary sets . . . . . . . 505
30.4 Cardinals versus set operations involving infinite sets . . . . . 510
30.5 The cardinal $\omega$ and the finite cardinals . . . . . . . . . . . . 512

30.5.1 First as well-orders . . . . . . . . . . . . . . . . . . . . 512
30.5.2 Then as cardinals . . . . . . . . . . . . . . . . . . . . . 513
30.6 The successor of a cardinal . . . . . . . . . . . . . . . . . . . . 514
30.7 Regular cardinals . . . . . . . . . . . . . . . . . . . . . . . . . 517
30.8 Others . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518
30.9 Regular vs. stable cardinals . . . . . . . . . . . . . . . . . . . . 518

**31 Cardinal Arithmetic as Needed by Bounded Natural Functors 520**
31.1 Zero . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 520
31.2 (In)finite cardinals . . . . . . . . . . . . . . . . . . . . . . . . 521
31.3 Binary sum . . . . . . . . . . . . . . . . . . . . . . . . . . . 522
31.4 One . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 523
31.5 Two . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 524
31.6 Family sum . . . . . . . . . . . . . . . . . . . . . . . . . . . 524
31.7 Product . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 524
31.8 Exponentiation . . . . . . . . . . . . . . . . . . . . . . . . . . 526

**32 Function Definition Base 530**

**33 Definition of Bounded Natural Functors 531**

**34 Composition of Bounded Natural Functors 536**

**35 Registration of Basic Types as Bounded Natural Functors 540**

**36 Shared Fixpoint Operations on Bounded Natural Functors 542**

**37 Least Fixpoint (Datatype) Operation on Bounded Natural Functors 548**

**38 Equivalence Relations in Higher-Order Set Theory 551**
38.1 Equivalence relations – set version . . . . . . . . . . . . . . . . 551
38.2 Equivalence classes . . . . . . . . . . . . . . . . . . . . . . . . 552
38.3 Quotients . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 553
38.4 Refinement of one equivalence relation WRT another . . . . . 554
38.5 Defining unary operations upon equivalence classes . . . . . . 554
38.6 Defining binary operations upon equivalence classes . . . . . 555
38.7 Quotients and finiteness . . . . . . . . . . . . . . . . . . . . . 557
38.8 Kernel of a Function . . . . . . . . . . . . . . . . . . . . . . . 557
38.9 Projection . . . . . . . . . . . . . . . . . . . . . . . . . . . . 557
38.10 Equivalence relations – predicate version . . . . . . . . . . . . 558
38.11 Equivalence closure . . . . . . . . . . . . . . . . . . . . . . . . 560

**39 MESON Proof Method** **564**
39.1 Negation Normal Form . . . . . . . . . . . . . . . . . . . . . . 564
39.2 Pulling out the existential quantifiers . . . . . . . . . . . . . . . 564
39.3 Lemmas for Forward Proof . . . . . . . . . . . . . . . . . . . . 565
39.4 Clausification helper . . . . . . . . . . . . . . . . . . . . . . . . 566
39.5 Skolemization helpers . . . . . . . . . . . . . . . . . . . . . . . 567
39.6 Meson package . . . . . . . . . . . . . . . . . . . . . . . . . . . 567

**40 Automatic Theorem Provers (ATPs)** **567**
40.1 ATP problems and proofs . . . . . . . . . . . . . . . . . . . . . 567
40.2 Higher-order reasoning helpers . . . . . . . . . . . . . . . . . . 567
40.3 Basic connection between ATPs and HOL . . . . . . . . . . . . 570

**41 Metis Proof Method** **570**
41.1 Literal selection and lambda-lifting helpers . . . . . . . . . . . 570
41.2 Metis package . . . . . . . . . . . . . . . . . . . . . . . . . . . . 571

**42 Generic theorem transfer using relations** **571**
42.1 Relator for function space . . . . . . . . . . . . . . . . . . . . . 571
42.2 Transfer method . . . . . . . . . . . . . . . . . . . . . . . . . . 572
42.3 Predicates on relations, i.e. "class constraints" . . . . . . . . 573
42.4 Properties of relators . . . . . . . . . . . . . . . . . . . . . . . . 576
42.5 Transfer rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . 577
42.6 *of-bool* and *of-nat* . . . . . . . . . . . . . . . . . . . . . . . . . 582

**43 Lifting package** **582**
43.1 Function map . . . . . . . . . . . . . . . . . . . . . . . . . . . . 582
43.2 Quotient Predicate . . . . . . . . . . . . . . . . . . . . . . . . . 583
43.3 Quotient composition . . . . . . . . . . . . . . . . . . . . . . . . 586
43.4 Respects predicate . . . . . . . . . . . . . . . . . . . . . . . . . 586
43.5 Domains . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 591
43.6 ML setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 592

**44 Definition of Quotient Types** **592**
44.1 Quotient Predicate . . . . . . . . . . . . . . . . . . . . . . . . . 593
44.2 lemmas for regularisation of ball and bex . . . . . . . . . . . . 596
44.3 Bounded abstraction . . . . . . . . . . . . . . . . . . . . . . . . 597
44.4 *Bex1-rel* quantifier . . . . . . . . . . . . . . . . . . . . . . . . . 598
44.5 Various respects and preserve lemmas . . . . . . . . . . . . . . 599
44.6 Quotient composition . . . . . . . . . . . . . . . . . . . . . . . . 601
44.7 Quotient3 to Quotient . . . . . . . . . . . . . . . . . . . . . . . 601
44.8 ML setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 602
44.9 Methods / Interface . . . . . . . . . . . . . . . . . . . . . . . . . 603

**45 Lifting of BNFs** **603**

**46 Binary Numerals 606**
46.1 The *num* type . . . 606
46.2 Numeral operations . . . 607
46.3 Binary numerals . . . 610
46.4 Class-specific numeral rules . . . 611
46.4.1 Structures with addition: class *numeral* . . . 611
46.4.2 Structures with negation: class *neg-numeral* . . . 611
46.4.3 Structures with multiplication: class *semiring-numeral* 614
46.4.4 Structures with a zero: class *semiring-1* . . . 615
46.4.5 Equality: class *semiring-char-0* . . . 615
46.4.6 Comparisons: class *linordered-nonzero-semiring* . . . 616
46.4.7 Multiplication and negation: class *ring-1* . . . 618
46.4.8 Equality using *iszero* for rings with non-zero characteristic . . . 619
46.4.9 Equality and negation: class *ring-char-0* . . . 620
46.4.10 Structures with negation and order: class *linordered-idom* 621
46.4.11 Natural numbers . . . 624
46.5 Particular lemmas concerning $2::'a$ . . . 627
46.6 Numeral equations as default simplification rules . . . 627
46.6.1 Special Simplification for Constants . . . 628
46.6.2 Optional Simplification Rules Involving Constants . . 629
46.7 Setting up simprocs . . . 630
46.7.1 Simplification of arithmetic operations on integer constants . . . 631
46.7.2 Simplification of arithmetic when nested to the right . 632
46.8 Code module namespace . . . 632
46.9 Printing of evaluated natural numbers as numerals . . . 632
46.10 More on auxiliary conversion . . . 633

**47 Exponentiation 633**
47.1 Powers for Arbitrary Monoids . . . 633
47.2 Exponentiation on ordered types . . . 640
47.3 Miscellaneous rules . . . 647
47.4 Exponentiation for the Natural Numbers . . . 648
47.4.1 Cardinality of the Powerset . . . 649
47.5 Code generator tweak . . . 649

**48 Big sum and product over finite (non-empty) sets 649**
48.1 Generic monoid operation over a set . . . 649
48.1.1 Standard sum or product indexed by a finite set . . . 650
48.1.2 HOL Light variant: sum/product indexed by the non-neutral subset . . . 656
48.2 Generalized summation over a set . . . 657
48.2.1 Properties in more restricted classes of structures . . . 658

48.2.2 Cardinality as special case of *sum* . . . . . . . . . . . 663
48.2.3 Cardinality of products . . . . . . . . . . . . . . . . . 665
48.3 Generalized product over a set . . . . . . . . . . . . . . . . . 665
48.3.1 Properties in more restricted classes of structures . . . 666

**49 Chain-complete partial orders and their fixpoints 671**
49.1 Chains . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 671
49.2 Chain-complete partial orders . . . . . . . . . . . . . . . . . . 671
49.3 Transfinite iteration of a function . . . . . . . . . . . . . . . . 672
49.4 Fixpoint combinator . . . . . . . . . . . . . . . . . . . . . . . 672
49.5 Fixpoint induction . . . . . . . . . . . . . . . . . . . . . . . . 673

**50 Datatype option 674**
50.0.1 Operations . . . . . . . . . . . . . . . . . . . . . . . . 676
50.1 Transfer rules for the Transfer package . . . . . . . . . . . . . 681
50.1.1 Interaction with finite sets . . . . . . . . . . . . . . . . 681
50.1.2 Code generator setup . . . . . . . . . . . . . . . . . . 681

**51 Partial Function Definitions 682**
51.1 Axiomatic setup . . . . . . . . . . . . . . . . . . . . . . . . . 683
51.2 Flat interpretation: tailrec and option . . . . . . . . . . . . . 685

**52 Reconstructing external resolution proofs for propositional logic 687**

**53 Function Definitions and Termination Proofs 687**
53.1 Definitions with default value . . . . . . . . . . . . . . . . . . 688
53.2 Measure functions . . . . . . . . . . . . . . . . . . . . . . . . 689
53.3 Congruence rules . . . . . . . . . . . . . . . . . . . . . . . . . 689
53.4 Simp rules for termination proofs . . . . . . . . . . . . . . . . 689
53.5 Decomposition . . . . . . . . . . . . . . . . . . . . . . . . . . 690
53.6 Reduction pairs . . . . . . . . . . . . . . . . . . . . . . . . . . 690
53.7 Concrete orders for SCNP termination proofs . . . . . . . . . 690
53.8 Yet more induction principles on the natural numbers . . . . 692
53.9 Tool setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 692

**54 The Integers as Equivalence Classes over Pairs of Natural Numbers 692**
54.1 Definition of integers as a quotient type . . . . . . . . . . . . 692
54.2 Integers form a commutative ring . . . . . . . . . . . . . . . . 692
54.3 Integers are totally ordered . . . . . . . . . . . . . . . . . . . 693
54.4 Ordering properties of arithmetic operations . . . . . . . . . . 694
54.5 Embedding of the Integers into any *ring-1*: *of-int* . . . . . . . 695
54.6 Magnitude of an Integer, as a Natural Number: *nat* . . . . . 700
54.7 Lemmas about the Function *of-nat* and Orderings . . . . . . 703

54.8 Cases and induction . . . . . . . . . . . . . . . . . . . . . . . 704
54.8.1 Binary comparisons . . . . . . . . . . . . . . . . . . . 705
54.8.2 Comparisons, for Ordered Rings . . . . . . . . . . . . 705
54.9 The Set of Integers . . . . . . . . . . . . . . . . . . . . . . . . 705
54.10 *sum* and *prod* . . . . . . . . . . . . . . . . . . . . . . . . . 708
54.11 Setting up simplification procedures . . . . . . . . . . . . . . 708
54.12 More Inequality Reasoning . . . . . . . . . . . . . . . . . . . 708
54.13 The functions *nat* and *int* . . . . . . . . . . . . . . . . . . . 709
54.14 Induction principles for int . . . . . . . . . . . . . . . . . . . 711
54.15 Intermediate value theorems . . . . . . . . . . . . . . . . . . . 712
54.16 Products and 1, by T. M. Rasmussen . . . . . . . . . . . . . . 712
54.17 The divides relation . . . . . . . . . . . . . . . . . . . . . . . 713
54.18 Powers with integer exponents . . . . . . . . . . . . . . . . . 715
54.19 Finiteness of intervals . . . . . . . . . . . . . . . . . . . . . . 720
54.20 Configuration of the code generator . . . . . . . . . . . . . . . 720
54.21 Duplicates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 724

**55 Big infimum (minimum) and supremum (maximum) over finite (non-empty) sets 724**
55.1 Generic lattice operations over a set . . . . . . . . . . . . . . . 724
55.1.1 Without neutral element . . . . . . . . . . . . . . . . . . 724
55.1.2 With neutral element . . . . . . . . . . . . . . . . . . . 726
55.2 Lattice operations on finite sets . . . . . . . . . . . . . . . . . . 728
55.3 Infimum and Supremum over non-empty sets . . . . . . . . . 729
55.4 Minimum and Maximum over non-empty sets . . . . . . . . . 730
55.5 An aside: code generation for *LEAST* and *GREATEST* . . . 736
55.6 Arg Min . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 737
55.7 Arg Max . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 739

**56 Division in euclidean (semi)rings 740**
56.1 Euclidean (semi)rings with explicit division and remainder . . 740
56.2 Euclidean (semi)rings with cancel rules . . . . . . . . . . . . . 742
56.3 Uniquely determined division . . . . . . . . . . . . . . . . . . 747
56.4 Division on *nat* . . . . . . . . . . . . . . . . . . . . . . . . . . 748
56.5 Division on *int* . . . . . . . . . . . . . . . . . . . . . . . . . . 755
56.5.1 Basic instantiation . . . . . . . . . . . . . . . . . . . . . 755
56.5.2 Algebraic foundations . . . . . . . . . . . . . . . . . . . 756
56.5.3 Basic conversions . . . . . . . . . . . . . . . . . . . . . . 757
56.5.4 Euclidean division . . . . . . . . . . . . . . . . . . . . . 758
56.5.5 Trivial reduction steps . . . . . . . . . . . . . . . . . . . 759
56.5.6 More uniqueness rules . . . . . . . . . . . . . . . . . . . 759
56.5.7 Laws for unary minus . . . . . . . . . . . . . . . . . . . 760
56.5.8 Borders . . . . . . . . . . . . . . . . . . . . . . . . . . . 760
56.5.9 Splitting Rules for div and mod . . . . . . . . . . . . . . 761

56.5.10 Algebraic rewrites . . . . . . . . . . . . . . . . . . . . 761
56.5.11 Distributive laws for conversions. . . . . . . . . . . . . 762
56.5.12 Monotonicity in the First Argument (Dividend) . . . . 762
56.5.13 Monotonicity in the Second Argument (Divisor) . . . 762
56.5.14 Quotients of Signs . . . . . . . . . . . . . . . . . . . . 763
56.5.15 Further properties . . . . . . . . . . . . . . . . . . . . 764
56.5.16 Computing *div* and *mod* by shifting . . . . . . . . . . 765
56.6 Code generation . . . . . . . . . . . . . . . . . . . . . . . . 765

**57 Parity in rings and semirings 766**
57.1 Ring structures with parity and *even*/*odd* predicates . . . . . 766
57.2 Instance for *nat* . . . . . . . . . . . . . . . . . . . . . . . . 769
57.3 Parity and powers . . . . . . . . . . . . . . . . . . . . . . . 771
57.4 Instance for *int* . . . . . . . . . . . . . . . . . . . . . . . . 773
57.5 Special case: euclidean rings structurally containing the natural numbers . . . . . . . . . . . . . . . . . . . . . . . . . . 774
57.6 Generic symbolic computations . . . . . . . . . . . . . . . . . 777
57.6.1 Computation by simplification . . . . . . . . . . . . . . 782
57.7 Computing congruences modulo $2 \^{} q$ . . . . . . . . . . . . . 782

**58 Combination and Cancellation Simprocs for Numeral Expressions 783**

**59 Semiring normalization 785**

**60 Groebner bases 787**
60.1 Groebner Bases . . . . . . . . . . . . . . . . . . . . . . . . . 787

**61 Set intervals 789**
61.1 Various equivalences . . . . . . . . . . . . . . . . . . . . . . 791
61.2 Logical Equivalences for Set Inclusion and Equality . . . . . . 792
61.3 Two-sided intervals . . . . . . . . . . . . . . . . . . . . . . . 793
61.3.1 Emptyness, singletons, subset . . . . . . . . . . . . . . . 794
61.4 Infinite intervals . . . . . . . . . . . . . . . . . . . . . . . . 800
61.4.1 Intersection . . . . . . . . . . . . . . . . . . . . . . . . 801
61.5 Intervals of natural numbers . . . . . . . . . . . . . . . . . . 803
61.5.1 The Constant *lessThan* . . . . . . . . . . . . . . . . . . 803
61.5.2 The Constant *greaterThan* . . . . . . . . . . . . . . . . 803
61.5.3 The Constant *atLeast* . . . . . . . . . . . . . . . . . . 804
61.5.4 The Constant *atMost* . . . . . . . . . . . . . . . . . . 804
61.5.5 The Constant *atLeastLessThan* . . . . . . . . . . . . . . 804
61.5.6 The Constant *atLeastAtMost* . . . . . . . . . . . . . . . 805
61.5.7 Intervals of nats with *Suc* . . . . . . . . . . . . . . . . 805
61.5.8 Intervals and numerals . . . . . . . . . . . . . . . . . . 806

61.5.9 Image . . . 806
61.5.10 Finiteness . . . 810
61.5.11 Proving Inclusions and Equalities between Unions . . . 811
61.5.12 Cardinality . . . 811
61.6 Intervals of integers . . . 813
61.6.1 Finiteness . . . 813
61.6.2 Cardinality . . . 813
61.7 Lemmas useful with the summation operator sum . . . 814
61.7.1 Disjoint Unions . . . 815
61.7.2 Disjoint Intersections . . . 815
61.7.3 Some Differences . . . 816
61.7.4 Some Subset Conditions . . . 816
61.8 Generic big monoid operation over intervals . . . 816
61.9 Summation indexed over intervals . . . 819
61.9.1 Shifting bounds . . . 823
61.9.2 Telescoping sums . . . 824
61.9.3 The formula for geometric sums . . . 825
61.9.4 Geometric progressions . . . 826
61.9.5 The formulae for arithmetic sums . . . 826
61.9.6 Division remainder . . . 828
61.10 Products indexed over intervals . . . 828
61.10.1 Telescoping products . . . 829
61.11 Efficient folding over intervals . . . 829

**62 Decision Procedure for Presburger Arithmetic 830**
62.1 The $-\infty$ and $+\infty$ Properties . . . 830
62.2 The A and B sets . . . 831
62.3 Cooper's Theorem $-\infty$ and $+\infty$ Version . . . 833
62.3.1 First some trivial facts about periodic sets or predicates 833
62.3.2 The $-\infty$ Version . . . 833
62.3.3 The $+\infty$ Version . . . 833
62.4 Nice facts about division by $4::'a$ . . . 836

**63 Bindings to Satisfiability Modulo Theories (SMT) solvers based on SMT-LIB 2 837**
63.1 A skolemization tactic and proof method . . . 837
63.2 Triggers for quantifier instantiation . . . 837
63.3 Higher-order encoding . . . 838
63.4 Normalization . . . 838
63.5 Integer division and modulo for Z3 . . . 839
63.6 Extra theorems for veriT reconstruction . . . 840
63.7 Setup . . . 847
63.8 Configuration . . . 848
63.9 General configuration options . . . 848

63.10 Certificates . . . 849
63.11 Tracing . . . 849
63.12 Schematic rules for Z3 proof reconstruction . . . 849

**64 Sledgehammer: Isabelle–ATP Linkup 852**

**65 Setup for Lifting/Transfer for the set type 852**
65.1 Relator and predicator properties . . . 852
65.2 Quotient theorem for the Lifting package . . . 854
65.3 Transfer rules for the Transfer package . . . 854
65.3.1 Unconditional transfer rules . . . 854
65.3.2 Rules requiring bi-unique, bi-total or right-total relations . . . 855

**66 The datatype of finite lists 858**
66.1 Basic list processing functions . . . 859
66.1.1 List comprehension . . . 867
66.1.2 [] and (#) . . . 868
66.1.3 *length* . . . 868
66.1.4 @ – append . . . 870
66.1.5 *map* . . . 872
66.1.6 *rev* . . . 874
66.1.7 *set* . . . 876
66.1.8 *concat* . . . 878
66.1.9 *filter* . . . 879
66.1.10 List partitioning . . . 881
66.1.11 (!) . . . 882
66.1.12 *list-update* . . . 883
66.1.13 *last* and *butlast* . . . 885
66.1.14 *take* and *drop* . . . 887
66.1.15 *takeWhile* and *dropWhile* . . . 892
66.1.16 *zip* . . . 895
66.1.17 *list-all2* . . . 899
66.1.18 *List.product* and *product-lists* . . . 901
66.1.19 *fold* with natural argument order . . . 902
66.1.20 Fold variants: *foldr* and *foldl* . . . 905
66.1.21 *upt* . . . 906
66.1.22 *upto*: interval-list on *int* . . . 909
66.1.23 *successively* . . . 910
66.1.24 *distinct* and *remdups* and *remdups-adj* . . . 911
66.2 *distinct-adj* . . . 918
66.2.1 *insert* . . . 919
66.2.2 *List.union* . . . 919
66.2.3 *find* . . . 919

66.2.4 *count-list* . . . . . . . . . . . . . . . . . . . . . . . 920
66.2.5 *List.extract* . . . . . . . . . . . . . . . . . . . . . . . 921
66.2.6 *remove1* . . . . . . . . . . . . . . . . . . . . . . . . 921
66.2.7 *removeAll* . . . . . . . . . . . . . . . . . . . . . . . 922
66.2.8 *minus-list-mset* . . . . . . . . . . . . . . . . . . . . . 924
66.2.9 *minus-list-set* . . . . . . . . . . . . . . . . . . . . . 925
66.2.10 *inter-list-set* . . . . . . . . . . . . . . . . . . . . . 926
66.2.11 *replicate* . . . . . . . . . . . . . . . . . . . . . . . 926
66.2.12 *enumerate* . . . . . . . . . . . . . . . . . . . . . . . 929
66.2.13 *rotate1* and *rotate* . . . . . . . . . . . . . . . . . . . 930
66.2.14 *nths* — a generalization of (!) to sets . . . . . . . . . . 932
66.2.15 *subseqs* and *List.n-lists* . . . . . . . . . . . . . . . . . 934
66.2.16 *splice* . . . . . . . . . . . . . . . . . . . . . . . . . 934
66.2.17 *shuffles* . . . . . . . . . . . . . . . . . . . . . . . . 935
66.2.18 Transpose . . . . . . . . . . . . . . . . . . . . . . . . 936
66.2.19 *min* and *arg-min* . . . . . . . . . . . . . . . . . . . . 937
66.2.20 (In)finiteness . . . . . . . . . . . . . . . . . . . . . . 937
66.3 Sorting . . . . . . . . . . . . . . . . . . . . . . . . . . . 939
66.3.1 *sorted-wrt* . . . . . . . . . . . . . . . . . . . . . . . 939
66.3.2 *sorted* . . . . . . . . . . . . . . . . . . . . . . . . . 940
66.3.3 Sorting functions . . . . . . . . . . . . . . . . . . . . . 943
66.3.4 *transpose* on sorted lists . . . . . . . . . . . . . . . . . 947
66.3.5 *sorted-key-list-of-set* . . . . . . . . . . . . . . . . . . 948
66.3.6 *lists*: the list-forming operator over sets . . . . . . . . 953
66.3.7 Inductive definition for membership . . . . . . . . . . . 954
66.3.8 Lists as Cartesian products . . . . . . . . . . . . . . . 955
66.3.9 Transitive Closure on Lists . . . . . . . . . . . . . . . 955
66.4 Relations on Lists . . . . . . . . . . . . . . . . . . . . . . . 956
66.4.1 Length Lexicographic Ordering . . . . . . . . . . . . . 956
66.4.2 Lexicographic Ordering . . . . . . . . . . . . . . . . . 958
66.4.3 Lexicographic combination of measure functions . . . 963
66.4.4 Lifting Relations to Lists: one element . . . . . . . . . 964
66.4.5 Lifting Relations to Lists: all elements . . . . . . . . . 966
66.5 Size function . . . . . . . . . . . . . . . . . . . . . . . . . 968
66.6 Monad operation . . . . . . . . . . . . . . . . . . . . . . . 968
66.7 Code generation . . . . . . . . . . . . . . . . . . . . . . . 969
66.7.1 Counterparts for set-related operations . . . . . . . . . 969
66.7.2 Special implementations . . . . . . . . . . . . . . . . . 975
66.7.3 Operations for optimization and efficiency . . . . . . . 976
66.7.4 Implementation of sets by lists . . . . . . . . . . . . . 976
66.7.5 Pretty lists . . . . . . . . . . . . . . . . . . . . . . . . 979
66.7.6 Use convenient predefined operations . . . . . . . . . . 979
66.8 Setup for Lifting/Transfer . . . . . . . . . . . . . . . . . . . 980
66.8.1 Transfer rules for the Transfer package . . . . . . . . . 980

66.9 Misc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 984

**67 Sum and product over lists 984**

67.1 List summation . . . . . . . . . . . . . . . . . . . . . . . . . . 986
67.2 Horner sums . . . . . . . . . . . . . . . . . . . . . . . . . . . . 991
67.3 Further facts about *List.n-lists* . . . . . . . . . . . . . . . . . 993
67.4 Tools setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . 993
67.5 List product . . . . . . . . . . . . . . . . . . . . . . . . . . . . 993

**68 Bit operations in suitable algebraic structures 995**

68.1 Abstract bit structures . . . . . . . . . . . . . . . . . . . . . . 995
68.2 Bit operations . . . . . . . . . . . . . . . . . . . . . . . . . . . 1000
68.3 Common algebraic structure . . . . . . . . . . . . . . . . . . . . 1014
68.4 Instance *int* . . . . . . . . . . . . . . . . . . . . . . . . . . . 1016
68.5 Instance *nat* . . . . . . . . . . . . . . . . . . . . . . . . . . . 1024
68.6 Symbolic computations on numeral expressions . . . . . . . . 1029
68.7 Symbolic computations for code generation . . . . . . . . . . 1038
68.8 More properties . . . . . . . . . . . . . . . . . . . . . . . . . . 1044
68.9 Bit concatenation . . . . . . . . . . . . . . . . . . . . . . . . . 1044
68.10Taking bits with sign propagation . . . . . . . . . . . . . . . . 1045
68.11Key ideas of bit operations . . . . . . . . . . . . . . . . . . . . 1050
68.12Lemma duplicates and other . . . . . . . . . . . . . . . . . . . 1051

**69 Numeric types for code generation onto target language numerals only 1054**

69.1 Type of target language integers . . . . . . . . . . . . . . . . . 1055
69.2 Code theorems for target language integers . . . . . . . . . . 1062
69.3 Serializer setup for target language integers . . . . . . . . . . 1071
69.4 Type of target language naturals . . . . . . . . . . . . . . . . . 1077
69.5 Inductive representation of target language naturals . . . . . 1082
69.6 Code refinement for target language naturals . . . . . . . . . 1083

**70 A HOL random engine 1086**

70.1 Auxiliary functions . . . . . . . . . . . . . . . . . . . . . . . . 1086
70.2 Random seeds . . . . . . . . . . . . . . . . . . . . . . . . . . . 1086
70.3 Base selectors . . . . . . . . . . . . . . . . . . . . . . . . . . . 1087
70.4 *ML* interface . . . . . . . . . . . . . . . . . . . . . . . . . . . 1088

**71 Maps 1088**

71.1 *empty* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1091
71.2 *map-upd* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1091
71.3 *map-of* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1091
71.4 *map-option* related . . . . . . . . . . . . . . . . . . . . . . . . 1093
71.5 *map-comp* related . . . . . . . . . . . . . . . . . . . . . . . . 1093

71.6 ++ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1094
71.7 *restrict-map* . . . . . . . . . . . . . . . . . . . . . . . . . . 1095
71.8 *map-upds* . . . . . . . . . . . . . . . . . . . . . . . . . . . 1096
71.9 *dom* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1097
71.10 *ran* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1099
71.11 *graph* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1100
71.12 *map-le* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1101
71.13 Various . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1102

**72 Finite types as explicit enumerations** **1103**
72.1 Class *enum* . . . . . . . . . . . . . . . . . . . . . . . . . . . 1103
72.2 Implementations using *enum* . . . . . . . . . . . . . . . . . . . 1104
72.2.1 Unbounded operations and quantifiers . . . . . . . . . 1104
72.2.2 An executable choice operator . . . . . . . . . . . . . 1104
72.2.3 Equality and order on functions . . . . . . . . . . . . 1105
72.2.4 Operations on relations . . . . . . . . . . . . . . . . . 1105
72.2.5 Bounded accessible part . . . . . . . . . . . . . . . . . 1106
72.3 Default instances for *enum* . . . . . . . . . . . . . . . . . . . 1107
72.4 Small finite types . . . . . . . . . . . . . . . . . . . . . . . . . 1110
72.5 Closing up . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1121

**73 Character and string types** **1121**
73.1 Strings as list of bytes . . . . . . . . . . . . . . . . . . . . . . 1121
73.1.1 Bytes as datatype . . . . . . . . . . . . . . . . . . . . 1121
73.2 Strings as dedicated type for target language code generation 1127
73.2.1 Logical specification . . . . . . . . . . . . . . . . . . . 1127
73.2.2 Syntactic representation . . . . . . . . . . . . . . . . . 1128
73.2.3 Operations . . . . . . . . . . . . . . . . . . . . . . . . 1129
73.2.4 Executable conversions . . . . . . . . . . . . . . . . . 1131
73.2.5 Technical code generation setup . . . . . . . . . . . . 1132
73.2.6 Code generation utility . . . . . . . . . . . . . . . . . 1136
73.2.7 Finally . . . . . . . . . . . . . . . . . . . . . . . . . . 1136

**74 Reflecting Pure types into HOL** **1136**

**75 Predicates as enumerations** **1137**
75.1 The type of predicate enumerations (a monad) . . . . . . . . 1137
75.2 Emptiness check and definite choice . . . . . . . . . . . . . . 1141
75.3 Derived operations . . . . . . . . . . . . . . . . . . . . . . . . 1142
75.4 Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . 1144

**76 Lazy sequences** **1149**
76.1 Type of lazy sequences . . . . . . . . . . . . . . . . . . . . . . 1149
76.2 Code setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 1152
76.3 Generator Sequences . . . . . . . . . . . . . . . . . . . . . . . 1153
76.3.1 General lazy sequence operation . . . . . . . . . . . . 1153
76.3.2 Small lazy typeclasses . . . . . . . . . . . . . . . . . . 1153
76.4 With Hit Bound Value . . . . . . . . . . . . . . . . . . . . . . 1154

**77 Depth-Limited Sequences with failure element** **1155**
77.1 Depth-Limited Sequence . . . . . . . . . . . . . . . . . . . . . 1155
77.2 Positive Depth-Limited Sequence . . . . . . . . . . . . . . . . 1157
77.3 Negative Depth-Limited Sequence . . . . . . . . . . . . . . . . 1158
77.4 Negation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1158

**78 Term evaluation using the generic code generator** **1159**
78.1 Term representation . . . . . . . . . . . . . . . . . . . . . . . 1159
78.1.1 Terms and class *term-of* . . . . . . . . . . . . . . . . . 1159
78.1.2 Syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . 1160
78.2 Tools setup and evaluation . . . . . . . . . . . . . . . . . . . . 1160
78.3 Dedicated *term-of* instances . . . . . . . . . . . . . . . . . . . 1161
78.4 Generic reification . . . . . . . . . . . . . . . . . . . . . . . . 1162
78.5 Diagnostic . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1162

**79 A simple counterexample generator performing random testing** **1162**
79.1 Catching Match exceptions . . . . . . . . . . . . . . . . . . . . 1162
79.2 The *random* class . . . . . . . . . . . . . . . . . . . . . . . . 1163
79.3 Fundamental and numeric types . . . . . . . . . . . . . . . . . 1163
79.4 Complex generators . . . . . . . . . . . . . . . . . . . . . . . . 1166
79.5 Deriving random generators for datatypes . . . . . . . . . . . 1168
79.6 Code setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1168

**80 The Random-Predicate Monad** **1168**

**81 Various kind of sequences inside the random monad** **1170**

**82 A simple counterexample generator performing exhaustive testing** **1173**
82.1 Basic operations for exhaustive generators . . . . . . . . . . . 1173
82.2 Exhaustive generator type classes . . . . . . . . . . . . . . . . 1173
82.2.1 A smarter enumeration scheme for functions over finite datatypes . . . . . . . . . . . . . . . . . . . . . . . . 1179
82.3 Bounded universal quantifiers . . . . . . . . . . . . . . . . . . 1187
82.4 Fast exhaustive combinators . . . . . . . . . . . . . . . . . . . 1187
82.5 Continuation passing style functions as plus monad . . . . . . 1187

82.6 Defining generators for any first-order data type . . . . . . . 1189
82.7 Defining generators for abstract types . . . . . . . . . . . . . 1189

**83 A compiler for predicates defined by introduction rules 1190**
83.1 Set membership as a generator predicate . . . . . . . . . . . . 1190

**84 Counterexample generator performing narrowing-based testing 1191**
84.1 Counterexample generator . . . . . . . . . . . . . . . . . . . . 1191
84.1.1 Code generation setup . . . . . . . . . . . . . . . . . . 1191
84.1.2 Narrowing's deep representation of types and terms . 1192
84.1.3 From narrowing's deep representation of terms to *HOL.Code-Evaluation*'s terms . . . . . . . . . . . . . . . . . . . . . . . . . . . 1192
84.1.4 Auxilary functions for Narrowing . . . . . . . . . . . . 1192
84.1.5 Narrowing's basic operations . . . . . . . . . . . . . . 1193
84.1.6 Narrowing generator type class . . . . . . . . . . . . . 1194
84.1.7 class *is-testable* . . . . . . . . . . . . . . . . . . . . . 1194
84.1.8 Defining a simple datatype to represent functions in an incomplete and redundant way . . . . . . . . . . . 1194
84.1.9 Setting up the counterexample generator . . . . . . . 1195
84.2 Narrowing for sets . . . . . . . . . . . . . . . . . . . . . . . . 1195
84.3 Narrowing for integers . . . . . . . . . . . . . . . . . . . . . . 1196
84.4 The *find-unused-assms* command . . . . . . . . . . . . . . . . 1197
84.5 Closing up . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1197

**85 Program extraction for HOL 1198**
85.1 Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1198
85.2 Type of extracted program . . . . . . . . . . . . . . . . . . . . 1198
85.3 Realizability . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1199
85.4 Computational content of basic inference rules . . . . . . . . 1201

**86 Extensible records with structural subtyping 1206**
86.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1206
86.2 Operators and lemmas for types isomorphic to tuples . . . . . 1207
86.3 Logical infrastructure for records . . . . . . . . . . . . . . . . 1208
86.4 Concrete record syntax . . . . . . . . . . . . . . . . . . . . . . 1213
86.5 Record package . . . . . . . . . . . . . . . . . . . . . . . . . . 1214

**87 Greatest common divisor and least common multiple 1214**
87.1 Abstract bounded quasi semilattices as common foundation . 1214
87.2 Abstract GCD and LCM . . . . . . . . . . . . . . . . . . . . . 1216
87.3 An aside: GCD and LCM on finite sets for incomplete gcd rings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1226
87.4 Coprimality . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1229

87.5 GCD and LCM for multiplicative normalisation functions . . 1232
87.6 GCD and LCM on *nat* and *int* . . . . . . . . . . . . . . . . . 1233
87.7 Bezout's theorem . . . . . . . . . . . . . . . . . . . . . . . . 1240
87.8 LCM properties on *nat* and *int* . . . . . . . . . . . . . . . . . 1241
87.9 The complete divisibility lattice on *nat* and *int* . . . . . . . . 1243
87.9.1 Setwise GCD and LCM for integers . . . . . . . . . . 1245
87.10 GCD and LCM on *integer* . . . . . . . . . . . . . . . . . . . . 1246
87.11 Characteristic of a semiring . . . . . . . . . . . . . . . . . . . 1248

**88 Nitpick: Yet Another Counterexample Generator for Isabelle/HOL 1250**

**89 Greatest Fixpoint (Codatatype) Operation on Bounded Natural Functors 1254**
89.1 Equivalence relations, quotients, and Hilbert's choice . . . . . 1259

**90 Filters on predicates 1260**
90.1 Filters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1260
90.1.1 Eventually . . . . . . . . . . . . . . . . . . . . . . . . 1260
90.2 Frequently as dual to eventually . . . . . . . . . . . . . . . . 1262
90.2.1 Finer-than relation . . . . . . . . . . . . . . . . . . . . 1265
90.2.2 Map function for filters . . . . . . . . . . . . . . . . . . 1268
90.2.3 Contravariant map function for filters . . . . . . . . . 1269
90.2.4 Standard filters . . . . . . . . . . . . . . . . . . . . . . 1270
90.2.5 Order filters . . . . . . . . . . . . . . . . . . . . . . . . 1271
90.3 Sequentially . . . . . . . . . . . . . . . . . . . . . . . . . . . 1273
90.4 Increasing finite subsets . . . . . . . . . . . . . . . . . . . . . 1274
90.5 The cofinite filter . . . . . . . . . . . . . . . . . . . . . . . . 1275
90.5.1 Product of filters . . . . . . . . . . . . . . . . . . . . . 1275
90.6 Limits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1278
90.7 Limits to *at-top* and *at-bot* . . . . . . . . . . . . . . . . . . . 1281
90.8 Setup *'a filter* for lifting and transfer . . . . . . . . . . . . . . 1282

**91 Conditionally-complete Lattices 1286**
91.1 Preorders . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1287
91.2 Lattices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1290
91.3 Conditionally complete lattices . . . . . . . . . . . . . . . . . 1291
91.4 Complete lattices . . . . . . . . . . . . . . . . . . . . . . . . 1296
91.5 Instances . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1297

**92 Factorial Function, Rising Factorials 1301**
92.1 Factorial Function . . . . . . . . . . . . . . . . . . . . . . . . 1302
92.2 Pochhammer's symbol: generalized rising factorial . . . . . . 1304
92.3 Misc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1307

**93 Binomial Coefficients, Binomial Theorem, Inclusion-exclusion Principle 1308**
93.1 Binomial coefficients . . . . . . . . . . . . . . . . . . . . . . . . 1308
93.2 The binomial theorem (courtesy of Tobias Nipkow): . . . . . 1309
93.3 Generalized binomial coefficients . . . . . . . . . . . . . . . . . 1311
93.4 Summation on the upper index . . . . . . . . . . . . . . . . . . 1315
93.5 More on Binomial Coefficients . . . . . . . . . . . . . . . . . . . 1317
93.6 Inclusion-exclusion principle . . . . . . . . . . . . . . . . . . . . 1318
93.7 Versions for unrestrictedly additive functions . . . . . . . . . 1319
93.8 General "Moebius inversion" inclusion-exclusion principle . . . 1320
93.9 Executable code . . . . . . . . . . . . . . . . . . . . . . . . . . 1321

**94 Main HOL 1321**
94.1 Namespace cleanup . . . . . . . . . . . . . . . . . . . . . . . . . 1321
94.2 Syntax cleanup . . . . . . . . . . . . . . . . . . . . . . . . . . . 1322
94.3 Lattice syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1322

**95 Archimedean Fields, Floor and Ceiling Functions 1323**
95.1 Class of Archimedean fields . . . . . . . . . . . . . . . . . . . . 1323
95.2 Existence and uniqueness of floor function . . . . . . . . . . . 1324
95.3 Floor function . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1325
95.4 Ceiling function . . . . . . . . . . . . . . . . . . . . . . . . . . . 1328
95.4.1 Ceiling with numerals. . . . . . . . . . . . . . . . . . . . . 1329
95.4.2 Addition and subtraction of integers. . . . . . . . . . . . 1331
95.5 Negation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1332
95.6 Natural numbers . . . . . . . . . . . . . . . . . . . . . . . . . . . 1332
95.7 Frac Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1332
95.8 Fractional part arithmetic . . . . . . . . . . . . . . . . . . . . . 1333
95.9 Rounding to the nearest integer . . . . . . . . . . . . . . . . . 1334

**96 Rational numbers 1336**
96.1 Rational numbers as quotient . . . . . . . . . . . . . . . . . . . 1336
96.1.1 Construction of the type of rational numbers . . . . . 1336
96.1.2 Representation and basic operations . . . . . . . . . . 1336
96.1.3 Function *normalize* . . . . . . . . . . . . . . . . . . . . . 1339
96.1.4 Various . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1341
96.1.5 The ordered field of rational numbers . . . . . . . . . 1341
96.1.6 Rationals are an Archimedean field . . . . . . . . . . . 1343
96.2 Linear arithmetic setup . . . . . . . . . . . . . . . . . . . . . . . 1343
96.3 Embedding from Rationals to other Fields . . . . . . . . . . . 1344
96.4 The Set of Rational Numbers . . . . . . . . . . . . . . . . . . . 1346
96.5 Implementation of rational numbers as pairs of integers . . . 1348
96.6 Setup for Nitpick . . . . . . . . . . . . . . . . . . . . . . . . . . 1353
96.7 Float syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1353

96.8 Hiding implementation details . . . . . . . . . . . . . . . . . . 1353

**97 Development of the Reals using Cauchy Sequences 1353**
97.1 Preliminary lemmas . . . . . . . . . . . . . . . . . . . . . . . . 1354
97.2 Sequences that converge to zero . . . . . . . . . . . . . . . . . 1354
97.3 Cauchy sequences . . . . . . . . . . . . . . . . . . . . . . . . . 1355
97.4 Equivalence relation on Cauchy sequences . . . . . . . . . . . 1356
97.5 The field of real numbers . . . . . . . . . . . . . . . . . . . . 1357
97.6 Positive reals . . . . . . . . . . . . . . . . . . . . . . . . . . . 1358
97.7 Completeness . . . . . . . . . . . . . . . . . . . . . . . . . . . 1360
97.8 Supremum of a set of reals . . . . . . . . . . . . . . . . . . . 1361
97.9 Hiding implementation details . . . . . . . . . . . . . . . . . . 1361
97.10Embedding numbers into the Reals . . . . . . . . . . . . . . . 1361
97.11Embedding the Naturals into the Reals . . . . . . . . . . . . 1362
97.12The Archimedean Property of the Reals . . . . . . . . . . . . 1363
97.13Rationals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1364
97.14Density of the Rational Reals in the Reals . . . . . . . . . . . 1364
97.15Numerals and Arithmetic . . . . . . . . . . . . . . . . . . . . 1364
97.16Simprules combining $x + y$ and *0* . . . . . . . . . . . . . . 1364
97.17Lemmas about powers . . . . . . . . . . . . . . . . . . . . . . 1365
97.18Density of the Reals . . . . . . . . . . . . . . . . . . . . . . . 1365
97.19Archimedean properties and useful consequences . . . . . . . 1366
97.20Floor and Ceiling Functions from the Reals to the Integers . 1367
97.21Exponentiation with floor . . . . . . . . . . . . . . . . . . . . 1369
97.22Implementation of rational real numbers . . . . . . . . . . . . 1370
97.23Setup for Nitpick . . . . . . . . . . . . . . . . . . . . . . . . . 1373
97.24Setup for SMT . . . . . . . . . . . . . . . . . . . . . . . . . . 1373
97.25Setup for Argo . . . . . . . . . . . . . . . . . . . . . . . . . . 1374

**98 Topological Spaces 1374**
98.1 Topological space . . . . . . . . . . . . . . . . . . . . . . . . . 1374
98.2 Hausdorff and other separation properties . . . . . . . . . . . 1377
98.3 Generators for topologies . . . . . . . . . . . . . . . . . . . . . 1378
98.4 Order topologies . . . . . . . . . . . . . . . . . . . . . . . . . . 1379
98.5 Setup some topologies . . . . . . . . . . . . . . . . . . . . . . . 1380
98.5.1 Boolean is an order topology . . . . . . . . . . . . . . . 1380
98.5.2 Topological filters . . . . . . . . . . . . . . . . . . . . . 1381
98.5.3 Tendsto . . . . . . . . . . . . . . . . . . . . . . . . . . . 1386
98.5.4 Rules about *Lim* . . . . . . . . . . . . . . . . . . . . . 1390
98.6 Limits on sequences . . . . . . . . . . . . . . . . . . . . . . . . 1391
98.7 Monotone sequences and subsequences . . . . . . . . . . . . . 1392
98.7.1 Definition of subsequence. . . . . . . . . . . . . . . . . . 1392
98.7.2 Subsequence (alternative definition, (e.g. Hoskins) . . 1394
98.7.3 Increasing and Decreasing Series . . . . . . . . . . . . . 1397

98.8 First countable topologies . . . . . . . . . . . . . . . . . . . . 1398
98.9 Function limit at a point . . . . . . . . . . . . . . . . . . . . . . 1399
98.9.1 Relation of *LIM* and *LIMSEQ* . . . . . . . . . . . . . 1401
98.10 Continuity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1402
98.10.1 Continuity on a set . . . . . . . . . . . . . . . . . . . . 1402
98.10.2 Continuity at a point . . . . . . . . . . . . . . . . . . . 1405
98.10.3 Open-cover compactness . . . . . . . . . . . . . . . . . 1409
98.11 Finite intersection property . . . . . . . . . . . . . . . . . . . 1410
98.12 Connectedness . . . . . . . . . . . . . . . . . . . . . . . . . . 1411

**99 Linear Continuum Topologies** **1414**
99.1 Intermediate Value Theorem . . . . . . . . . . . . . . . . . . . 1415
99.2 Uniform spaces . . . . . . . . . . . . . . . . . . . . . . . . . . 1417
99.2.1 Totally bounded sets . . . . . . . . . . . . . . . . . . . . 1418
99.2.2 Cauchy filter . . . . . . . . . . . . . . . . . . . . . . . . 1418
99.2.3 Uniformly continuous functions . . . . . . . . . . . . . 1419

**100 Product Topology** **1420**
100.1 Product is a topological space . . . . . . . . . . . . . . . . . . 1420
100.1.1 Continuity of operations . . . . . . . . . . . . . . . . . 1421
100.1.2 Connectedness of products . . . . . . . . . . . . . . . . 1423
100.1.3 Separation axioms . . . . . . . . . . . . . . . . . . . . . 1423
100.2 A generic notion of the convex, affine, conic hull, or closed "hull". . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1424

**101 Modules** **1425**
101.1 Locale for additive functions . . . . . . . . . . . . . . . . . . . 1425

**102 Subspace** **1427**

**103 Span: subspace generated by a set** **1428**

**104 Dependent and independent sets** **1431**

**105 Representation of a vector on a specific basis** **1432**

**106 Vector Spaces** **1437**

**107 Vector Spaces and Algebras over the Reals** **1451**
107.1 Real vector spaces . . . . . . . . . . . . . . . . . . . . . . . . 1451
107.2 Embedding of the Reals into any *real-algebra-1*: *of-real* . . . 1455
107.3 The Set of Real Numbers . . . . . . . . . . . . . . . . . . . . . 1458
107.4 Ordered real vector spaces . . . . . . . . . . . . . . . . . . . . 1460
107.5 Real normed vector spaces . . . . . . . . . . . . . . . . . . . . 1463
107.6 Metric spaces . . . . . . . . . . . . . . . . . . . . . . . . . . . 1470

107.7 Class instances for real numbers . . . . . . . . . . . . . . . . . 1471
107.8 Extra type constraints . . . . . . . . . . . . . . . . . . . . . . . . . 1472
107.9 Sign function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1473
107.10 Bounded Linear and Bilinear Operators . . . . . . . . . . . . 1474
107.11 Filters and Limits on Metric Space . . . . . . . . . . . . . . . . 1478
107.11.1 Limits of Sequences . . . . . . . . . . . . . . . . . . . . 1480
107.11.2 Limits of Functions . . . . . . . . . . . . . . . . . . . . . 1480
107.12 Complete metric spaces . . . . . . . . . . . . . . . . . . . . . . 1481
107.13 Cauchy sequences . . . . . . . . . . . . . . . . . . . . . . . . . . 1481
107.13.1 Cauchy Sequences are Convergent . . . . . . . . . . . 1483
107.14 The set of real numbers is a complete metric space . . . . . . 1483

**108 Limits on Real Vector Spaces** **1484**
108.1 Filter going to infinity norm . . . . . . . . . . . . . . . . . . . . 1484
108.1.1 Boundedness . . . . . . . . . . . . . . . . . . . . . . . . 1485
108.1.2 Bounded Sequences . . . . . . . . . . . . . . . . . . . . 1486
108.1.3 A Few More Equivalence Theorems for Boundedness . 1487
108.1.4 Upper Bounds and Lubs of Bounded Sequences . . . . 1487
108.1.5 Polynomal function extremal theorem, from HOL Light 1488
108.2 Convergence to Zero . . . . . . . . . . . . . . . . . . . . . . . . . 1488
108.2.1 Distance and norms . . . . . . . . . . . . . . . . . . . . 1490
108.3 Topological Monoid . . . . . . . . . . . . . . . . . . . . . . . . . 1491
108.3.1 Topological group . . . . . . . . . . . . . . . . . . . . . 1492
108.3.2 Linear operators and multiplication . . . . . . . . . . 1494
108.3.3 Inverse and division . . . . . . . . . . . . . . . . . . . . 1500
108.4 Relate *at*, *at-left* and *at-right* . . . . . . . . . . . . . . . . . . 1505
108.5 Floor and Ceiling . . . . . . . . . . . . . . . . . . . . . . . . . . . 1513
108.6 Limits of Sequences . . . . . . . . . . . . . . . . . . . . . . . . . 1514
108.7 Convergence on sequences . . . . . . . . . . . . . . . . . . . . . 1517
108.8 More about *filterlim* (thanks to Wenda Li) . . . . . . . . . . . 1520
108.9 Power Sequences . . . . . . . . . . . . . . . . . . . . . . . . . . . 1522
108.10 Limits of Functions . . . . . . . . . . . . . . . . . . . . . . . . . 1523
108.11 Continuity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1524
108.12 Uniform Continuity . . . . . . . . . . . . . . . . . . . . . . . . . 1526
108.13 Nested Intervals and Bisection – Needed for Compactness . . 1526
108.14 Boundedness of continuous functions . . . . . . . . . . . . . . 1527

**109 Infinite Series** **1529**
109.1 Definition of infinite summability . . . . . . . . . . . . . . . . . 1529
109.2 Infinite summability on topological monoids . . . . . . . . . . 1530
109.3 Infinite summability on ordered, topological monoids . . . . . 1532
109.4 Infinite summability on topological monoids . . . . . . . . . . 1534
109.5 Infinite summability on real normed vector spaces . . . . . . 1535
109.6 Infinite summability on real normed algebras . . . . . . . . . 1537

109.7 Infinite summability on real normed fields . . . . . . . . . . . 1538
109.8 Telescoping . . . . . . . . . . . . . . . . . . . . . . . . . . . 1539
109.9 Infinite summability on Banach spaces . . . . . . . . . . . . . 1540
109.10 The Ratio Test . . . . . . . . . . . . . . . . . . . . . . . . . 1541
109.11 Cauchy Product Formula . . . . . . . . . . . . . . . . . . . . 1542
109.12 Series on *reals* . . . . . . . . . . . . . . . . . . . . . . . . 1542

**110 Differentiation** **1544**
110.1 Frechet derivative . . . . . . . . . . . . . . . . . . . . . . . . 1545
110.1.1 Limit transformation for derivatives . . . . . . . . . . 1548
110.2 Continuity . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1549
110.3 Composition . . . . . . . . . . . . . . . . . . . . . . . . . . . 1549
110.4 Uniqueness . . . . . . . . . . . . . . . . . . . . . . . . . . . 1551
110.5 Differentiability predicate . . . . . . . . . . . . . . . . . . . . 1552
110.6 Vector derivative . . . . . . . . . . . . . . . . . . . . . . . . 1555
110.7 Derivatives . . . . . . . . . . . . . . . . . . . . . . . . . . . 1557
110.8 Local extrema . . . . . . . . . . . . . . . . . . . . . . . . . . 1563
110.9 Rolle's Theorem . . . . . . . . . . . . . . . . . . . . . . . . . 1565
110.10 Mean Value Theorem . . . . . . . . . . . . . . . . . . . . . . 1566
110.10.1 A function is constant if its derivative is 0 over an interval. . . . . . . . . . . . . . . . . . . . . . . . . . 1566
110.10.2 A function with positive derivative is increasing . . . . 1567
110.11 Generalized Mean Value Theorem . . . . . . . . . . . . . . . 1570
110.12 L'Hopitals rule . . . . . . . . . . . . . . . . . . . . . . . . . 1570

**111 Nth Roots of Real Numbers** **1574**
111.1 Existence of Nth Root . . . . . . . . . . . . . . . . . . . . . . 1574
111.2 Nth Root . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1574
111.3 Square Root . . . . . . . . . . . . . . . . . . . . . . . . . . . 1578
111.4 Square Root of Sum of Squares . . . . . . . . . . . . . . . . . 1582

**112 Power Series, Transcendental Functions etc.** **1584**
112.1 Properties of Power Series . . . . . . . . . . . . . . . . . . . 1585
112.2 Alternating series test / Leibniz formula . . . . . . . . . . . . 1586
112.3 Term-by-Term Differentiability of Power Series . . . . . . . . 1587
112.4 The Derivative of a Power Series Has the Same Radius of Convergence . . . . . . . . . . . . . . . . . . . . . . . . . . . 1588
112.5 Derivability of power series . . . . . . . . . . . . . . . . . . . 1590
112.6 Exponential Function . . . . . . . . . . . . . . . . . . . . . . 1591
112.6.1 Properties of the Exponential Function . . . . . . . . 1592
112.6.2 Properties of the Exponential Function on Reals . . . 1593
112.7 Natural Logarithm . . . . . . . . . . . . . . . . . . . . . . . . 1595
112.7.1 A couple of simple bounds . . . . . . . . . . . . . . . . 1604
112.8 The general logarithm . . . . . . . . . . . . . . . . . . . . . . 1604

112.9 Sine and Cosine . . . . . . . . . . . . . . . . . . . . . . . . . . 1619
112.10 Properties of Sine and Cosine . . . . . . . . . . . . . . . . . . . . 1622
112.11 Deriving the Addition Formulas . . . . . . . . . . . . . . . . . . . 1622
112.12 The Constant Pi . . . . . . . . . . . . . . . . . . . . . . . . . . 1625
112.13 More Corollaries about Sine and Cosine . . . . . . . . . . . . . 1633
112.14 Tangent . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1635
112.15 Cotangent . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1639
112.16 Inverse Trigonometric Functions . . . . . . . . . . . . . . . . . 1641
112.17 Prove Totality of the Trigonometric Functions . . . . . . . . . 1647
112.18 Machin's formula . . . . . . . . . . . . . . . . . . . . . . . . . 1650
112.19 Introducing the inverse tangent power series . . . . . . . . . . 1651
112.20 Existence of Polar Coordinates . . . . . . . . . . . . . . . . . . 1652
112.21 Basics about polynomial functions: products, extremal behaviour and root counts . . . . . . . . . . . . . . . . . . . . . . 1652
112.22 Hyperbolic functions . . . . . . . . . . . . . . . . . . . . . . . 1654
112.22.1 More specific properties of the real functions . . . . . 1658
112.22.2 Limits . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1661
112.22.3 Properties of the inverse hyperbolic functions . . . . . 1661
112.23 Simprocs for root and power literals . . . . . . . . . . . . . . 1667

**113 Complex Numbers: Rectangular and Polar Representations 1668**
113.1 Addition and Subtraction . . . . . . . . . . . . . . . . . . . . . 1668
113.2 Multiplication and Division . . . . . . . . . . . . . . . . . . . . 1669
113.3 Scalar Multiplication . . . . . . . . . . . . . . . . . . . . . . . . 1670
113.4 Numerals, Arithmetic, and Embedding from R . . . . . . . . . 1670
113.5 The Complex Number $i$ . . . . . . . . . . . . . . . . . . . . . . 1672
113.6 Vector Norm . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1673
113.7 Absolute value . . . . . . . . . . . . . . . . . . . . . . . . . . . 1675
113.8 Completeness of the Complexes . . . . . . . . . . . . . . . . . . 1676
113.9 Complex Conjugation . . . . . . . . . . . . . . . . . . . . . . . 1677
113.10 Basic Lemmas . . . . . . . . . . . . . . . . . . . . . . . . . . . 1680
113.11 Polar Form for Complex Numbers . . . . . . . . . . . . . . . . 1683
113.11.1 $\cos\theta + i\sin\theta$ . . . . . . . . . . . . . . . . . . . . . . 1683
113.11.2 $r(\cos\theta + i\sin\theta)$ . . . . . . . . . . . . . . . . . . . . 1684
113.11.3 Complex exponential . . . . . . . . . . . . . . . . . . . 1685
113.11.4 Complex argument . . . . . . . . . . . . . . . . . . . . 1687
113.12 Complex n-th roots . . . . . . . . . . . . . . . . . . . . . . . . 1688
113.13 Square root of complex numbers . . . . . . . . . . . . . . . . . 1688

**114 MacLaurin and Taylor Series 1691**
114.1 Maclaurin's Theorem with Lagrange Form of Remainder . . . 1691
114.2 More Convenient "Bidirectional" Version. . . . . . . . . . . . . 1693
114.3 Version for Exponential Function . . . . . . . . . . . . . . . . . 1694
114.4 Version for Sine Function . . . . . . . . . . . . . . . . . . . . . 1694

114.5 Maclaurin Expansion for Cosine Function . . . . . . . . . . . 1695

**115 Taylor series** **1696**

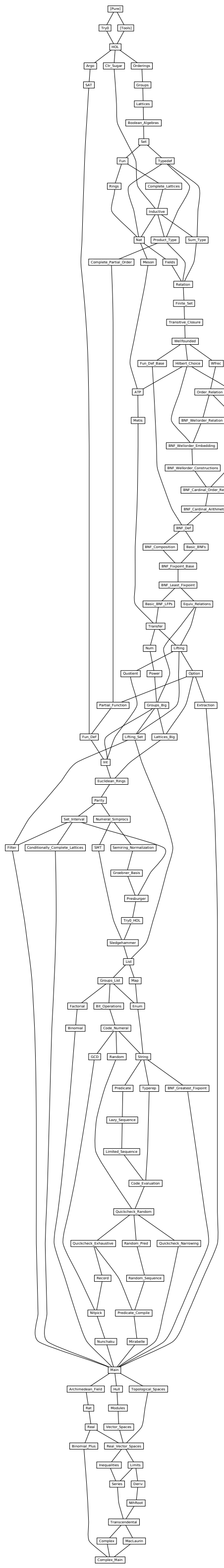
**116 More facts about binomial coefficients** **1696**

116.1 More facts about binomial coefficients . . . . . . . . . . . . . 1697

116.2 Results about binomials and integers, thanks to Alexander Maletzky . . . . . . . . . . . . . . . . . . . . . . . . . . . 1698

116.3 Sums . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1699

**117 Comprehensive Complex Theory** **1700**



# 1 Loading the code generator and related modules

**theory** *Code-Generator*
**imports** *Pure*
**keywords**
  *print-codeproc code-thms code-deps* :: *diag* **and**
  *export-code code-identifier code-printing code-reserved*
    *code-monad code-reflect* :: *thy-decl* **and**
  *checking* **and**
  *datatypes functions module-name file file-prefix*
    *constant type-constructor type-class class-relation class-instance code-module*
    :: *quasi-command*
**begin**

⟨*ML*⟩

**code-datatype** $TYPE('a::\{\})$

**definition** *holds* :: *prop* **where**
  $holds \equiv ((\lambda x{::}prop.\ x) \equiv (\lambda x.\ x))$

**lemma** *holds*: *PROP holds*
  ⟨*proof*⟩

**code-datatype** *holds*

**lemma** *implies-code* [*code*]:
  $(PROP\ P \Longrightarrow PROP\ holds) \equiv PROP\ holds$
  $(PROP\ holds \Longrightarrow PROP\ P) \equiv PROP\ P$
⟨*proof*⟩

⟨*ML*⟩

**hide-const** (**open**) *holds*

**end**

**theory** *Try0*
  **imports** *Pure*
  **keywords** *try0* :: *diag*
**begin**

⟨*ML*⟩

**end**

# 2 The basis of Higher-Order Logic

**theory** *HOL*
**imports** *Pure Try0 Tools.Code-Generator*
**keywords**
  *try solve-direct quickcheck print-coercions print-claset*
    *print-induct-rules* :: *diag* **and**
  *quickcheck-params* :: *thy-decl*
**abbrevs** *?<* = $\exists_{\leq 1}$
**begin**

⟨*ML*⟩

## 2.1 Primitive logic

The definition of the logic is based on Mike Gordon's technical report [2] that describes the first implementation of HOL. However, there are a number of differences. In particular, we start with the definite description operator and introduce Hilbert's $\varepsilon$ operator only much later. Moreover, axiom $(P \longrightarrow Q) \longrightarrow (Q \longrightarrow P) \longrightarrow (P = Q)$ is derived from the other axioms. The fact that this axiom is derivable was first noticed by Bruno Barras (for Mike Gordon's line of HOL systems) and later independently by Alexander Maletzky (for Isabelle/HOL).

### 2.1.1 Core syntax

⟨*ML*⟩
**default-sort** *type*
⟨*ML*⟩

**axiomatization where** *fun-arity*: *OFCLASS*($'a \Rightarrow 'b$, *type-class*)
**instance** *fun* :: (*type*, *type*) *type* ⟨*proof*⟩

**axiomatization where** *itself-arity*: *OFCLASS*(*'a itself*, *type-class*)
**instance** *itself* :: (*type*) *type* ⟨*proof*⟩

**typedecl** *bool*

**judgment** *Trueprop* :: *bool* ⇒ *prop* (‹(‹*notation=judgment*›-)› *5*)

**axiomatization** *implies* :: [*bool*, *bool*] ⇒ *bool* (**infixr** ‹⟶› *25*)
  **and** *eq* :: [*'a*, *'a*] ⇒ *bool*
  **and** *The* :: (*'a* ⇒ *bool*) ⇒ *'a*

**notation** (*input*)
  *eq* (**infixl** ‹=› *50*)
**notation** (**output**)
  *eq* (**infix** ‹=› *50*)

The input syntax for *eq* is more permissive than the output syntax because of the large amount of material that relies on infixl.

### 2.1.2 Defined connectives and quantifiers

**definition** *True* :: *bool*
  **where** *True* ≡ ((λ*x*::*bool*. *x*) = (λ*x*. *x*))

**definition** *All* :: (*'a* ⇒ *bool*) ⇒ *bool* (**binder** ‹∀› *10*)
  **where** *All P* ≡ (*P* = (λ*x*. *True*))

**definition** *Ex* :: (*'a* ⇒ *bool*) ⇒ *bool* (**binder** ‹∃› *10*)
  **where** *Ex P* ≡ ∀ *Q*. (∀ *x*. *P x* ⟶ *Q*) ⟶ *Q*

**definition** *False* :: *bool*
  **where** *False* ≡ (∀ *P*. *P*)

**definition** *Not* :: *bool* ⇒ *bool* (‹(‹*open-block notation*=‹*prefix* ¬››¬ -)› [*40*] *40*)
  **where** *not-def*: ¬ *P* ≡ *P* ⟶ *False*

**definition** *conj* :: [*bool*, *bool*] ⇒ *bool* (**infixr** ‹∧› *35*)
  **where** *and-def*: *P* ∧ *Q* ≡ ∀ *R*. (*P* ⟶ *Q* ⟶ *R*) ⟶ *R*

**definition** *disj* :: [*bool*, *bool*] ⇒ *bool* (**infixr** ‹∨› *30*)
  **where** *or-def*: *P* ∨ *Q* ≡ ∀ *R*. (*P* ⟶ *R*) ⟶ (*Q* ⟶ *R*) ⟶ *R*

**definition** *Uniq* :: (*'a* ⇒ *bool*) ⇒ *bool*
  **where** *Uniq P* ≡ (∀ *x y*. *P x* ⟶ *P y* ⟶ *y* = *x*)

**definition** *Ex1* :: (*'a* ⇒ *bool*) ⇒ *bool*
  **where** *Ex1 P* ≡ ∃ *x*. *P x* ∧ (∀ *y*. *P y* ⟶ *y* = *x*)

### 2.1.3 Additional concrete syntax

**syntax** (*ASCII*) *-Uniq* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=4 notation*=‹*binder* ?<››?< -./ -)› [*0*, *10*] *10*)
**syntax** *-Uniq* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=2 notation*=‹*binder* $\exists_{\leq 1}$››$\exists_{\leq 1}$ -./ -)› [*0*, *10*] *10*)

**syntax-consts** *-Uniq* ⇌ *Uniq*

**translations** $\exists_{\leq 1}$*x*. *P* ⇌ *CONST Uniq* (λ*x*. *P*)

⟨*ML*⟩

**syntax** (*ASCII*)
  *-Ex1* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=3 notation*=‹*binder EX*!››*EX*! -./ -)› [*0*, *10*] *10*)

**syntax** (*input*)
*-Ex1* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=3 notation=*‹*binder ?!*››*?! -./ -*)› [*0*, *10*] *10*)
**syntax** *-Ex1* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=3 notation=*‹*binder* ∃!››∃!*-./ -*)› [*0*, *10*] *10*)

**syntax-consts** *-Ex1* ⇌ *Ex1*

**translations** ∃!*x*. *P* ⇌ *CONST Ex1* (λ*x*. *P*)

⟨*ML*⟩

**syntax**
*-Not-Ex* :: *idts* ⇒ *bool* ⇒ *bool* (‹(‹*indent=3 notation=*‹*binder* ∄››∄*-./ -*)› [*0*, *10*] *10*)
*-Not-Ex1* :: *pttrn* ⇒ *bool* ⇒ *bool* (‹(‹*indent=3 notation=*‹*binder* ∄!››∄!*-./ -*)› [*0*, *10*] *10*)
**syntax-consts**
*-Not-Ex* ⇌ *Ex* **and**
*-Not-Ex1* ⇌ *Ex1*
**translations**
∄*x*. *P* ⇌ ¬ (∃*x*. *P*)
∄!*x*. *P* ⇌ ¬ (∃!*x*. *P*)

**abbreviation** *not-equal* :: [*'a*, *'a*] ⇒ *bool* (**infix** ‹≠› *50*)
**where** *x* ≠ *y* ≡ ¬ (*x* = *y*)

**notation** (*ASCII*)
*Not* (‹(‹*open-block notation=*‹*prefix* ~››~ *-*)› [*40*] *40*) **and**
*conj* (**infixr** ‹&› *35*) **and**
*disj* (**infixr** ‹|› *30*) **and**
*implies* (**infixr** ‹-->› *25*) **and**
*not-equal* (**infix** ‹~=› *50*)

**abbreviation** (*iff*)
*iff* :: [*bool*, *bool*] ⇒ *bool* (**infixr** ‹⟷› *25*)
**where** *A* ⟷ *B* ≡ *A* = *B*

**syntax** *-The* :: [*pttrn*, *bool*] ⇒ *'a* (‹(‹*indent=3 notation=*‹*binder THE*››*THE -./ -*)› [*0*, *10*] *10*)
**syntax-consts** *-The* ⇌ *The*
**translations** *THE x*. *P* ⇌ *CONST The* (λ*x*. *P*)
⟨*ML*⟩

**nonterminal** *case-syn* **and** *cases-syn*
**syntax**
*-case-syntax* :: [*'a*, *cases-syn*] ⇒ *'b* (‹(‹*notation=*‹*mixfix case expression*››*case -*

*of/ -)› 10)*
  *-case1* :: [*'a*, *'b*] ⇒ *case-syn*
  (‹(‹*indent=2 notation=*‹*mixfix case clause*››(‹*open-block notation=*‹*pattern case*››-)
⇒/ -)› *10*)
  :: *case-syn* ⇒ *cases-syn* (‹-›)
  *-case2* :: [*case-syn*, *cases-syn*] ⇒ *cases-syn* (‹-/ | -›)
**syntax** (*ASCII*)
  *-case1* :: [*'a*, *'b*] ⇒ *case-syn*
  (‹(‹*indent=2 notation=*‹*mixfix case clause*››(‹*open-block notation=*‹*pattern case*››-)
=>/ -)› *10*)

**notation** (*ASCII*)
  *All* (**binder** ‹*ALL* › *10*) **and**
  *Ex* (**binder** ‹*EX* › *10*)

**notation** (*input*)
  *All* (**binder** ‹! › *10*) **and**
  *Ex* (**binder** ‹*?* › *10*)

### 2.1.4 Axioms and basic definitions

**axiomatization where**
  *refl*: $t = (t{::}'a)$ **and**
  *subst*: $s = t \Longrightarrow P\ s \Longrightarrow P\ t$ **and**
  *ext*: $(\bigwedge x{::}'a.\ (f\ x\ {::}'b) = g\ x) \Longrightarrow (\lambda x.\ f\ x) = (\lambda x.\ g\ x)$
  — Extensionality is built into the meta-logic, and this rule expresses a related property. It is an eta-expanded version of the traditional rule, and similar to the ABS rule of HOL **and**

  *the-eq-trivial*: $(THE\ x.\ x = a) = (a{::}'a)$

**axiomatization where**
  *impI*: $(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$ **and**
  *mp*: $[\![P \longrightarrow Q;\ P]\!] \Longrightarrow Q$ **and**

  *True-or-False*: $(P = True) \vee (P = False)$

**definition** *If* :: *bool* ⇒ *'a* ⇒ *'a* ⇒ *'a* (‹(‹*notation=*‹*mixfix if expression*››*if* (-)/ *then* (-)/ *else* (-))› [*0*, *0*, *10*] *10*)
  **where** $If\ P\ x\ y \equiv (THE\ z{::}'a.\ (P = True \longrightarrow z = x) \wedge (P = False \longrightarrow z = y))$

**definition** *Let* :: *'a* ⇒ (*'a* ⇒ *'b*) ⇒ *'b*
  **where** $Let\ s\ f \equiv f\ s$

**nonterminal** *letbinds* **and** *letbind*

**open-bundle** *let-syntax*
**begin**

**syntax**
*-bind* :: [*pttrn*, *'a*] ⇒ *letbind* (‹(‹*indent=2 notation=*‹*mixfix let binding*››- =/ -)› *10*)
:: *letbind* ⇒ *letbinds* (‹-›)
*-binds* :: [*letbind*, *letbinds*] ⇒ *letbinds* (‹-;/ -›)
*-Let* :: [*letbinds*, *'a*] ⇒ *'a* (‹(‹*notation=*‹*mixfix let expression*››*let* (-)/ *in* (-))› [*0*, *10*] *10*)
**syntax-consts**
*-bind -binds -Let* ⇌ *Let*
**translations**
*-Let* (*-binds b bs*) *e* ⇌ *-Let b* (*-Let bs e*)
*let x* = *a in e* ⇌ *CONST Let a* (*λx. e*)

**end**

**axiomatization** *undefined* :: *'a*

**class** *default* = **fixes** *default* :: *'a*

## 2.2 Fundamental rules

### 2.2.1 Equality

**lemma** *sym*: $s = t \Longrightarrow t = s$
⟨*proof*⟩

**lemma** *ssubst*: $t = s \Longrightarrow P\ s \Longrightarrow P\ t$
⟨*proof*⟩

**lemma** *trans*: $[\![r = s;\ s = t]\!] \Longrightarrow r = t$
⟨*proof*⟩

**lemma** *trans-sym* [*Pure.elim?*]: $r = s \Longrightarrow t = s \Longrightarrow r = t$
⟨*proof*⟩

**lemma** *meta-eq-to-obj-eq*:
**assumes** $A \equiv B$
**shows** $A = B$
⟨*proof*⟩

Useful with *erule* for proving equalities from known equalities.

**lemma** *box-equals*: $[\![a = b;\ a = c;\ b = d]\!] \Longrightarrow c = d$
⟨*proof*⟩

For calculational reasoning:

**lemma** *forw-subst*: $a = b \Longrightarrow P\ b \Longrightarrow P\ a$
⟨*proof*⟩

**lemma** *back-subst*: $P\ a \Longrightarrow a = b \Longrightarrow P\ b$
⟨*proof*⟩

### 2.2.2 Congruence rules for application

Similar to *AP-THM* in Gordon's HOL.

**lemma** *fun-cong*: $(f :: 'a \Rightarrow 'b) = g \Longrightarrow f\ x = g\ x$
⟨*proof*⟩

Similar to *AP-TERM* in Gordon's HOL and FOL's *subst-context*.

**lemma** *arg-cong*: $x = y \Longrightarrow f\ x = f\ y$
⟨*proof*⟩

**lemma** *arg-cong2*: $[\![a = b;\ c = d]\!] \Longrightarrow f\ a\ c = f\ b\ d$
⟨*proof*⟩

**lemma** *arg-cong3*: $[\![x = x';\ y = y';\ z = z']\!] \Longrightarrow f\ x\ y\ z = f\ x'\ y'\ z'$
⟨*proof*⟩

**lemma** *arg-cong4*: $[\![w = w';\ x = x';\ y = y';\ z = z']\!] \Longrightarrow f\ w\ x\ y\ z = f\ w'\ x'\ y'\ z'$
⟨*proof*⟩

**lemma** *cong*: $[\![f = g;\ (x::'a) = y]\!] \Longrightarrow f\ x = g\ y$
⟨*proof*⟩

⟨*ML*⟩

### 2.2.3 Equality of booleans – iff

**lemma** *iffD2*: $[\![P = Q;\ Q]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *rev-iffD2*: $[\![Q;\ P = Q]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *iffD1*: $Q = P \Longrightarrow Q \Longrightarrow P$
⟨*proof*⟩

**lemma** *rev-iffD1*: $Q \Longrightarrow Q = P \Longrightarrow P$
⟨*proof*⟩

**lemma** *iffE*:
**assumes** *major*: $P = Q$
**and** *minor*: $[\![P \longrightarrow Q;\ Q \longrightarrow P]\!] \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

### 2.2.4 True (1)

**lemma** *TrueI*: *True*
⟨*proof*⟩

**lemma** *eqTrueE*: $P = True \Longrightarrow P$
⟨*proof*⟩

### 2.2.5 Universal quantifier (1)

**lemma** *spec*: $\forall x{::}'a.\ P\ x \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *allE*:
**assumes** *major*: $\forall x.\ P\ x$ **and** *minor*: $P\ x \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

**lemma** *all-dupE*:
**assumes** *major*: $\forall x.\ P\ x$ **and** *minor*: $[\![P\ x; \forall x.\ P\ x]\!] \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

### 2.2.6 False

Depends upon *spec*; it is impossible to do propositional logic before quantifiers!

**lemma** *FalseE*: $False \Longrightarrow P$
⟨*proof*⟩

**lemma** *False-neq-True*: $False = True \Longrightarrow P$
⟨*proof*⟩

### 2.2.7 Negation

**lemma** *notI*:
**assumes** $P \Longrightarrow False$
**shows** $\neg\ P$
⟨*proof*⟩

**lemma** *False-not-True*: $False \neq True$
⟨*proof*⟩

**lemma** *True-not-False*: $True \neq False$
⟨*proof*⟩

**lemma** *notE*: $[\![\neg\ P; P]\!] \Longrightarrow R$
⟨*proof*⟩

### 2.2.8 Implication

**lemma** *impE*:
**assumes** $P \longrightarrow Q\ P\ Q \Longrightarrow R$
**shows** $R$

⟨*proof*⟩

Reduces $Q$ to $P \longrightarrow Q$, allowing substitution in $P$.

**lemma** *rev-mp*: $[\![P;\ P \longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

**lemma** *contrapos-nn*:
**assumes** *major*: $\neg\ Q$
**and** *minor*: $P \Longrightarrow Q$
**shows** $\neg\ P$
⟨*proof*⟩

Not used at all, but we already have the other 3 combinations.

**lemma** *contrapos-pn*:
**assumes** *major*: $Q$
**and** *minor*: $P \Longrightarrow \neg\ Q$
**shows** $\neg\ P$
⟨*proof*⟩

**lemma** *not-sym*: $t \neq s \Longrightarrow s \neq t$
⟨*proof*⟩

**lemma** *eq-neq-eq-imp-neq*: $[\![x = a;\ a \neq b;\ b = y]\!] \Longrightarrow x \neq y$
⟨*proof*⟩

### 2.2.9 Disjunction (1)

**lemma** *disjE*:
**assumes** *major*: $P \vee Q$
**and** *minorP*: $P \Longrightarrow R$
**and** *minorQ*: $Q \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

### 2.2.10 Derivation of *iffI*

In an intuitionistic version of HOL *iffI* needs to be an axiom.

**lemma** *iffI*:
**assumes** $P \Longrightarrow Q$ **and** $Q \Longrightarrow P$
**shows** $P = Q$
⟨*proof*⟩

### 2.2.11 True (2)

**lemma** *eqTrueI*: $P \Longrightarrow P = True$
⟨*proof*⟩

### 2.2.12 Universal quantifier (2)

**lemma** *allI*:
  **assumes** $\bigwedge x{::}'a.\ P\ x$
  **shows** $\forall\, x.\ P\ x$
  ⟨*proof*⟩

### 2.2.13 Existential quantifier

**lemma** *exI*: $P\ x \Longrightarrow \exists\, x{::}'a.\ P\ x$
  ⟨*proof*⟩

**lemma** *exE*:
  **assumes** *major*: $\exists\, x{::}'a.\ P\ x$
    **and** *minor*: $\bigwedge x.\ P\ x \Longrightarrow Q$
  **shows** $Q$
  ⟨*proof*⟩

### 2.2.14 Conjunction

**lemma** *conjI*: $[\![P;\ Q]\!] \Longrightarrow P \wedge Q$
  ⟨*proof*⟩

**lemma** *conjunct1*: $[\![P \wedge Q]\!] \Longrightarrow P$
  ⟨*proof*⟩

**lemma** *conjunct2*: $[\![P \wedge Q]\!] \Longrightarrow Q$
  ⟨*proof*⟩

**lemma** *conjE*:
  **assumes** *major*: $P \wedge Q$
    **and** *minor*: $[\![P;\ Q]\!] \Longrightarrow R$
  **shows** $R$
⟨*proof*⟩

**lemma** *context-conjI*:
  **assumes** $P\ P \Longrightarrow Q$
  **shows** $P \wedge Q$
  ⟨*proof*⟩

### 2.2.15 Disjunction (2)

**lemma** *disjI1*: $P \Longrightarrow P \vee Q$
  ⟨*proof*⟩

**lemma** *disjI2*: $Q \Longrightarrow P \vee Q$
  ⟨*proof*⟩

### 2.2.16 Classical logic

**lemma** *classical*:

**assumes** $\neg P \Longrightarrow P$
**shows** $P$
$\langle proof \rangle$

**lemmas** *ccontr* = *FalseE* [*THEN classical*]

*notE* with premises exchanged; it discharges $\neg R$ so that it can be used to make elimination rules.

**lemma** *rev-notE*:
**assumes** *premp*: $P$
**and** *premnot*: $\neg R \Longrightarrow \neg P$
**shows** $R$
$\langle proof \rangle$

Double negation law.

**lemma** *notnotD*: $\neg\neg P \Longrightarrow P$
$\langle proof \rangle$

**lemma** *contrapos-pp*:
**assumes** *p1*: $Q$
**and** *p2*: $\neg P \Longrightarrow \neg Q$
**shows** $P$
$\langle proof \rangle$

### 2.2.17 Unique existence

**lemma** *Uniq-I* [*intro?*]:
**assumes** $\bigwedge x\ y.\ [\![P\ x;\ P\ y]\!] \Longrightarrow y = x$
**shows** *Uniq* $P$
$\langle proof \rangle$

**lemma** *Uniq-D* [*dest?*]: $[\![Uniq\ P;\ P\ a;\ P\ b]\!] \Longrightarrow a{=}b$
$\langle proof \rangle$

**lemma** *ex1I*:
**assumes** $P\ a\ \bigwedge x.\ P\ x \Longrightarrow x = a$
**shows** $\exists!x.\ P\ x$
$\langle proof \rangle$

Sometimes easier to use: the premises have no shared variables. Safe!

**lemma** *ex-ex1I*:
**assumes** *ex-prem*: $\exists x.\ P\ x$
**and** *eq*: $\bigwedge x\ y.\ [\![P\ x;\ P\ y]\!] \Longrightarrow x = y$
**shows** $\exists!x.\ P\ x$
$\langle proof \rangle$

**lemma** *ex1E*:
**assumes** *major*: $\exists!x.\ P\ x$ **and** *minor*: $\bigwedge x.\ [\![P\ x;\ \forall y.\ P\ y \longrightarrow y = x]\!] \Longrightarrow R$
**shows** $R$

⟨*proof*⟩

**lemma** *ex1-implies-ex*: $\exists!x.\ P\ x \Longrightarrow \exists x.\ P\ x$
⟨*proof*⟩

### 2.2.18 Classical intro rules for disjunction and existential quantifiers

**lemma** *disjCI*:
**assumes** $\neg\ Q \Longrightarrow P$
**shows** $P \vee Q$
⟨*proof*⟩

**lemma** *excluded-middle*: $\neg\ P \vee P$
⟨*proof*⟩

case distinction as a natural deduction rule. Note that $\neg\ P$ is the second case, not the first.

**lemma** *case-split* [*case-names True False*]:
**assumes** $P \Longrightarrow Q\ \neg\ P \Longrightarrow Q$
**shows** $Q$
⟨*proof*⟩

Classical implies ($\longrightarrow$) elimination.

**lemma** *impCE*:
**assumes** *major*: $P \longrightarrow Q$
**and** *minor*: $\neg\ P \Longrightarrow R\ Q \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

This version of $\longrightarrow$ elimination works on $Q$ before $P$. It works best for those cases in which $P$ holds "almost everywhere". Can't install as default: would break old proofs.

**lemma** *impCE'*:
**assumes** *major*: $P \longrightarrow Q$
**and** *minor*: $Q \Longrightarrow R\ \neg\ P \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

The analogous introduction rule for conjunction, above, is even constructive

**lemma** *context-disjE*:
**assumes** *major*: $P \vee Q$ **and** *minor*: $P \Longrightarrow R\ \neg P \Longrightarrow Q \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

Classical $\longleftrightarrow$ elimination.

**lemma** *iffCE*:
**assumes** *major*: $P = Q$

**and** *minor*: $[\![P;\ Q]\!] \Longrightarrow R$ $[\![\neg\ P;\ \neg\ Q]\!] \Longrightarrow R$
**shows** $R$
$\langle proof \rangle$

**lemma** *exCI*:
**assumes** $\forall\, x.\ \neg\ P\ x \Longrightarrow P\ a$
**shows** $\exists\, x.\ P\ x$
$\langle proof \rangle$

### 2.2.19 Intuitionistic Reasoning

**lemma** *impE'*:
**assumes** *1*: $P \longrightarrow Q$
**and** *2*: $Q \Longrightarrow R$
**and** *3*: $P \longrightarrow Q \Longrightarrow P$
**shows** $R$
$\langle proof \rangle$

**lemma** *allE'*:
**assumes** *1*: $\forall\, x.\ P\ x$
**and** *2*: $P\ x \Longrightarrow \forall\, x.\ P\ x \Longrightarrow Q$
**shows** $Q$
$\langle proof \rangle$

**lemma** *notE'*:
**assumes** *1*: $\neg\ P$
**and** *2*: $\neg\ P \Longrightarrow P$
**shows** $R$
$\langle proof \rangle$

**lemma** *TrueE*: $True \Longrightarrow P \Longrightarrow P$ $\langle proof \rangle$
**lemma** *notFalseE*: $\neg\ False \Longrightarrow P \Longrightarrow P$ $\langle proof \rangle$

**lemmas** [*Pure.elim!*] = *disjE iffE FalseE conjE exE TrueE notFalseE*
**and** [*Pure.intro!*] = *iffI conjI impI TrueI notI allI refl*
**and** [*Pure.elim 2*] = *allE notE' impE'*
**and** [*Pure.intro*] = *exI disjI2 disjI1*

**lemmas** [*trans*] = *trans*
**and** [*sym*] = *sym not-sym*
**and** [*Pure.elim?*] = *iffD1 iffD2 impE*

### 2.2.20 Atomizing meta-level connectives

**axiomatization where**
*eq-reflection*: $x = y \Longrightarrow x \equiv y$ — admissible axiom

**lemma** *atomize-all* [*atomize*]: $(\bigwedge x.\ P\ x) \equiv Trueprop\ (\forall\, x.\ P\ x)$
$\langle proof \rangle$

**lemma** *atomize-imp* [*atomize*]: $(A \Longrightarrow B) \equiv Trueprop\ (A \longrightarrow B)$
⟨*proof*⟩

**lemma** *atomize-not*: $(A \Longrightarrow False) \equiv Trueprop\ (\neg\ A)$
⟨*proof*⟩

**lemma** *atomize-eq* [*atomize*, *code*]: $(x \equiv y) \equiv Trueprop\ (x = y)$
⟨*proof*⟩

**lemma** *atomize-conj* [*atomize*]: $(A\ \&\&\&\ B) \equiv Trueprop\ (A \wedge B)$
⟨*proof*⟩

**lemmas** [*symmetric*, *rulify*] = *atomize-all atomize-imp*
**and** [*symmetric*, *defn*] = *atomize-all atomize-imp atomize-eq*

### 2.2.21 Atomizing elimination rules

**lemma** *atomize-exL*[*atomize-elim*]: $(\bigwedge x.\ P\ x \Longrightarrow Q) \equiv ((\exists\, x.\ P\ x) \Longrightarrow Q)$
⟨*proof*⟩

**lemma** *atomize-conjL*[*atomize-elim*]: $(A \Longrightarrow B \Longrightarrow C) \equiv (A \wedge B \Longrightarrow C)$
⟨*proof*⟩

**lemma** *atomize-disjL*[*atomize-elim*]: $((A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C) \equiv ((A \vee B \Longrightarrow C) \Longrightarrow C)$
⟨*proof*⟩

**lemma** *atomize-elimL*[*atomize-elim*]: $(\bigwedge B.\ (A \Longrightarrow B) \Longrightarrow B) \equiv Trueprop\ A$ ⟨*proof*⟩

## 2.3 Package setup

⟨*ML*⟩

### 2.3.1 Sledgehammer setup

Theorems blacklisted to Sledgehammer. These theorems typically produce clauses that are prolific (match too many equality or membership literals) and relate to seldom-used facts. Some duplicate other rules.

**named-theorems** *no-atp theorems that should be filtered out by Sledgehammer*

### 2.3.2 Classical Reasoner setup

**lemma** *imp-elim*: $P \longrightarrow Q \Longrightarrow (\neg\ R \Longrightarrow P) \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *swap*: $\neg\ P \Longrightarrow (\neg\ R \Longrightarrow P) \Longrightarrow R$
⟨*proof*⟩

**lemma** *thin-refl*: $[\![x = x;\ PROP\ W]\!] \Longrightarrow PROP\ W$ ⟨*proof*⟩

$\langle ML \rangle$

**declare** *iffI* [*intro*!]
  **and** *notI* [*intro*!]
  **and** *impI* [*intro*!]
  **and** *disjCI* [*intro*!]
  **and** *conjI* [*intro*!]
  **and** *TrueI* [*intro*!]
  **and** *refl* [*intro*!]

**declare** *iffCE* [*elim*!]
  **and** *FalseE* [*elim*!]
  **and** *impCE* [*elim*!]
  **and** *disjE* [*elim*!]
  **and** *conjE* [*elim*!]

**declare** *ex-ex1I* [*intro*!]
  **and** *allI* [*intro*!]
  **and** *exI* [*intro*]

**declare** *exE* [*elim*!]
  *allE* [*elim*]

$\langle ML \rangle$

**lemma** *contrapos-np*: $\neg\ Q \Longrightarrow (\neg\ P \Longrightarrow Q) \Longrightarrow P$
  $\langle proof \rangle$

**declare** *ex-ex1I* [*rule del*, *intro*! *2*]
  **and** *ex1I* [*intro*]

**declare** *ext* [*intro*]

**lemmas** [*intro?*] = *ext*
  **and** [*elim?*] = *ex1-implies-ex*

Better than *ex1E* for classical reasoner: needs no quantifier duplication!

**lemma** *alt-ex1E* [*elim*!]:
  **assumes** *major*: $\exists!x.\ P\ x$
    **and** *minor*: $\bigwedge x.\ [\![P\ x;\ \forall\ y\ y'.\ P\ y \wedge P\ y' \longrightarrow y = y']\!] \Longrightarrow R$
  **shows** $R$
$\langle proof \rangle$

And again using Uniq

**lemma** *alt-ex1E'*:
  **assumes** $\exists!x.\ P\ x\ \bigwedge x.\ [\![P\ x;\ \exists_{\leq 1}x.\ P\ x]\!] \Longrightarrow R$
  **shows** $R$
  $\langle proof \rangle$

**lemma** *ex1-iff-ex-Uniq*: $(\exists!x.\ P\ x) \longleftrightarrow (\exists\,x.\ P\ x) \wedge (\exists_{\leq 1}x.\ P\ x)$
⟨*proof*⟩

⟨*ML*⟩

### 2.3.3 THE: definite description operator

**lemma** *the-equality* [*intro*]:
**assumes** $P\ a$
**and** $\bigwedge x.\ P\ x \Longrightarrow x = a$
**shows** $(THE\ x.\ P\ x) = a$
⟨*proof*⟩

**lemma** *theI*:
**assumes** $P\ a$
**and** $\bigwedge x.\ P\ x \Longrightarrow x = a$
**shows** $P\ (THE\ x.\ P\ x)$
⟨*proof*⟩

**lemma** *theI'*: $\exists!x.\ P\ x \Longrightarrow P\ (THE\ x.\ P\ x)$
⟨*proof*⟩

Easier to apply than *theI*: only one occurrence of $P$.

**lemma** *theI2*:
**assumes** $P\ a\ \bigwedge x.\ P\ x \Longrightarrow x = a\ \bigwedge x.\ P\ x \Longrightarrow Q\ x$
**shows** $Q\ (THE\ x.\ P\ x)$
⟨*proof*⟩

**lemma** *the1I2*:
**assumes** $\exists!x.\ P\ x\ \bigwedge x.\ P\ x \Longrightarrow Q\ x$
**shows** $Q\ (THE\ x.\ P\ x)$
⟨*proof*⟩

**lemma** *the1-equality* [*elim?*]: $[\![\exists!x.\ P\ x;\ P\ a]\!] \Longrightarrow (THE\ x.\ P\ x) = a$
⟨*proof*⟩

**lemma** *the1-equality'*: $[\![\exists_{\leq 1}x.\ P\ x;\ P\ a]\!] \Longrightarrow (THE\ x.\ P\ x) = a$
⟨*proof*⟩

**lemma** *the-sym-eq-trivial*: $(THE\ y.\ x = y) = x$
⟨*proof*⟩

### 2.3.4 Simplifier

**lemma** *eta-contract-eq*: $(\lambda s.\ f\ s) = f$ ⟨*proof*⟩

**lemma** *subst-all*:
‹$(\bigwedge x.\ x = a \Longrightarrow PROP\ P\ x) \equiv PROP\ P\ a$›

$\langle(\bigwedge x.\ a = x \Longrightarrow \mathit{PROP}\ P\ x) \equiv \mathit{PROP}\ P\ a\rangle$
$\langle \mathit{proof} \rangle$

**lemma** *simp-thms*:
**shows** *not-not*: $(\neg\ \neg\ P) = P$
**and** *Not-eq-iff*: $((\neg\ P) = (\neg\ Q)) = (P = Q)$
**and**
$(P \neq Q) = (P = (\neg\ Q))$
$(P \vee \neg P) = \mathit{True} \quad (\neg\ P \vee P) = \mathit{True}$
$(x = x) = \mathit{True}$
**and** *not-True-eq-False* [*code*]: $(\neg\ \mathit{True}) = \mathit{False}$
**and** *not-False-eq-True* [*code*]: $(\neg\ \mathit{False}) = \mathit{True}$
**and**
$(\neg\ P) \neq P \quad P \neq (\neg\ P)$
$(\mathit{True} = P) = P$
**and** *eq-True*: $(P = \mathit{True}) = P$
**and** $(\mathit{False} = P) = (\neg\ P)$
**and** *eq-False*: $(P = \mathit{False}) = (\neg\ P)$
**and**
$(\mathit{True} \longrightarrow P) = P \quad (\mathit{False} \longrightarrow P) = \mathit{True}$
$(P \longrightarrow \mathit{True}) = \mathit{True} \quad (P \longrightarrow P) = \mathit{True}$
$(P \longrightarrow \mathit{False}) = (\neg\ P) \quad (P \longrightarrow \neg\ P) = (\neg\ P)$
$(P \wedge \mathit{True}) = P \quad (\mathit{True} \wedge P) = P$
$(P \wedge \mathit{False}) = \mathit{False} \quad (\mathit{False} \wedge P) = \mathit{False}$
$(P \wedge P) = P \quad (P \wedge (P \wedge Q)) = (P \wedge Q)$
$(P \wedge \neg\ P) = \mathit{False} \quad (\neg\ P \wedge P) = \mathit{False}$
$(P \vee \mathit{True}) = \mathit{True} \quad (\mathit{True} \vee P) = \mathit{True}$
$(P \vee \mathit{False}) = P \quad (\mathit{False} \vee P) = P$
$(P \vee P) = P \quad (P \vee (P \vee Q)) = (P \vee Q)$ **and**
$(\forall x.\ P) = P \quad (\exists x.\ P) = P \quad \exists x.\ x = t \quad \exists x.\ t = x$
**and**
$\bigwedge P.\ (\exists x.\ x = t \wedge P\ x) = P\ t$
$\bigwedge P.\ (\exists x.\ t = x \wedge P\ x) = P\ t$
$\bigwedge P.\ (\forall x.\ x = t \longrightarrow P\ x) = P\ t$
$\bigwedge P.\ (\forall x.\ t = x \longrightarrow P\ x) = P\ t$
$(\forall x.\ x \neq t) = \mathit{False} \quad (\forall x.\ t \neq x) = \mathit{False}$
$\langle \mathit{proof} \rangle$

**lemma** *disj-absorb*: $A \vee A \longleftrightarrow A$
$\langle \mathit{proof} \rangle$

**lemma** *disj-left-absorb*: $A \vee (A \vee B) \longleftrightarrow A \vee B$
$\langle \mathit{proof} \rangle$

**lemma** *conj-absorb*: $A \wedge A \longleftrightarrow A$
$\langle \mathit{proof} \rangle$

**lemma** *conj-left-absorb*: $A \wedge (A \wedge B) \longleftrightarrow A \wedge B$
$\langle \mathit{proof} \rangle$

**lemma** *eq-ac*:
**shows** *eq-commute*: $a = b \longleftrightarrow b = a$
**and** *iff-left-commute*: $(P \longleftrightarrow (Q \longleftrightarrow R)) \longleftrightarrow (Q \longleftrightarrow (P \longleftrightarrow R))$
**and** *iff-assoc*: $((P \longleftrightarrow Q) \longleftrightarrow R) \longleftrightarrow (P \longleftrightarrow (Q \longleftrightarrow R))$
⟨*proof*⟩

**lemma** *neq-commute*: $a \neq b \longleftrightarrow b \neq a$ ⟨*proof*⟩

**lemma** *conj-comms*:
**shows** *conj-commute*: $P \land Q \longleftrightarrow Q \land P$
**and** *conj-left-commute*: $P \land (Q \land R) \longleftrightarrow Q \land (P \land R)$ ⟨*proof*⟩
**lemma** *conj-assoc*: $(P \land Q) \land R \longleftrightarrow P \land (Q \land R)$ ⟨*proof*⟩

**lemmas** *conj-ac* = *conj-commute conj-left-commute conj-assoc*

**lemma** *disj-comms*:
**shows** *disj-commute*: $P \lor Q \longleftrightarrow Q \lor P$
**and** *disj-left-commute*: $P \lor (Q \lor R) \longleftrightarrow Q \lor (P \lor R)$ ⟨*proof*⟩
**lemma** *disj-assoc*: $(P \lor Q) \lor R \longleftrightarrow P \lor (Q \lor R)$ ⟨*proof*⟩

**lemmas** *disj-ac* = *disj-commute disj-left-commute disj-assoc*

**lemma** *conj-disj-distribL*: $P \land (Q \lor R) \longleftrightarrow P \land Q \lor P \land R$ ⟨*proof*⟩
**lemma** *conj-disj-distribR*: $(P \lor Q) \land R \longleftrightarrow P \land R \lor Q \land R$ ⟨*proof*⟩

**lemma** *disj-conj-distribL*: $P \lor (Q \land R) \longleftrightarrow (P \lor Q) \land (P \lor R)$ ⟨*proof*⟩
**lemma** *disj-conj-distribR*: $(P \land Q) \lor R \longleftrightarrow (P \lor R) \land (Q \lor R)$ ⟨*proof*⟩

**lemma** *imp-conjR*: $(P \longrightarrow (Q \land R)) = ((P \longrightarrow Q) \land (P \longrightarrow R))$ ⟨*proof*⟩
**lemma** *imp-conjL*: $((P \land Q) \longrightarrow R) = (P \longrightarrow (Q \longrightarrow R))$ ⟨*proof*⟩
**lemma** *imp-disjL*: $((P \lor Q) \longrightarrow R) = ((P \longrightarrow R) \land (Q \longrightarrow R))$ ⟨*proof*⟩

These two are specialized, but *imp-disj-not1* is useful in *Auth/Yahalom*.

**lemma** *imp-disj-not1*: $(P \longrightarrow Q \lor R) \longleftrightarrow (\neg Q \longrightarrow P \longrightarrow R)$ ⟨*proof*⟩
**lemma** *imp-disj-not2*: $(P \longrightarrow Q \lor R) \longleftrightarrow (\neg R \longrightarrow P \longrightarrow Q)$ ⟨*proof*⟩

**lemma** *imp-disj1*: $((P \longrightarrow Q) \lor R) \longleftrightarrow (P \longrightarrow Q \lor R)$ ⟨*proof*⟩
**lemma** *imp-disj2*: $(Q \lor (P \longrightarrow R)) \longleftrightarrow (P \longrightarrow Q \lor R)$ ⟨*proof*⟩

**lemma** *imp-cong*: $(P = P') \Longrightarrow (P' \Longrightarrow (Q = Q')) \Longrightarrow ((P \longrightarrow Q) \longleftrightarrow (P' \longrightarrow Q'))$
⟨*proof*⟩

**lemma** *de-Morgan-disj*: $\neg (P \lor Q) \longleftrightarrow \neg P \land \neg Q$ ⟨*proof*⟩
**lemma** *de-Morgan-conj*: $\neg (P \land Q) \longleftrightarrow \neg P \lor \neg Q$ ⟨*proof*⟩
**lemma** *not-imp*: $\neg (P \longrightarrow Q) \longleftrightarrow P \land \neg Q$ ⟨*proof*⟩
**lemma** *not-iff*: $P \neq Q \longleftrightarrow (P \longleftrightarrow \neg Q)$ ⟨*proof*⟩
**lemma** *disj-not1*: $\neg P \lor Q \longleftrightarrow (P \longrightarrow Q)$ ⟨*proof*⟩

**lemma** *disj-not2*: $P \lor \neg\ Q \longleftrightarrow (Q \longrightarrow P)$ ⟨*proof*⟩
**lemma** *imp-conv-disj*: $(P \longrightarrow Q) \longleftrightarrow (\neg\ P) \lor Q$ ⟨*proof*⟩
**lemma** *disj-imp*: $P \lor Q \longleftrightarrow \neg\ P \longrightarrow Q$ ⟨*proof*⟩

**lemma** *iff-conv-conj-imp*: $(P \longleftrightarrow Q) \longleftrightarrow (P \longrightarrow Q) \land (Q \longrightarrow P)$ ⟨*proof*⟩

**lemma** *cases-simp*: $(P \longrightarrow Q) \land (\neg\ P \longrightarrow Q) \longleftrightarrow Q$
— Avoids duplication of subgoals after *if-split*, when the true and false
— cases boil down to the same thing.
⟨*proof*⟩

**lemma** *not-all*: $\neg\ (\forall x.\ P\ x) \longleftrightarrow (\exists x.\ \neg\ P\ x)$ ⟨*proof*⟩
**lemma** *imp-all*: $((\forall x.\ P\ x) \longrightarrow Q) \longleftrightarrow (\exists x.\ P\ x \longrightarrow Q)$ ⟨*proof*⟩
**lemma** *not-ex*: $\neg\ (\exists x.\ P\ x) \longleftrightarrow (\forall x.\ \neg\ P\ x)$ ⟨*proof*⟩
**lemma** *imp-ex*: $((\exists x.\ P\ x) \longrightarrow Q) \longleftrightarrow (\forall x.\ P\ x \longrightarrow Q)$ ⟨*proof*⟩
**lemma** *all-not-ex*: $(\forall x.\ P\ x) \longleftrightarrow \neg\ (\exists x.\ \neg\ P\ x)$ ⟨*proof*⟩

**declare** *All-def* [*no-atp*]

**lemma** *ex-disj-distrib*: $(\exists x.\ P\ x \lor Q\ x) \longleftrightarrow (\exists x.\ P\ x) \lor (\exists x.\ Q\ x)$ ⟨*proof*⟩
**lemma** *all-conj-distrib*: $(\forall x.\ P\ x \land Q\ x) \longleftrightarrow (\forall x.\ P\ x) \land (\forall x.\ Q\ x)$ ⟨*proof*⟩
**lemma** *all-imp-conj-distrib*: $(\forall x.\ P\ x \longrightarrow Q\ x \land R\ x) \longleftrightarrow (\forall x.\ P\ x \longrightarrow Q\ x) \land (\forall x.\ P\ x \longrightarrow R\ x)$
⟨*proof*⟩

The $\land$ congruence rule: not included by default! May slow rewrite proofs down by as much as 50%

**lemma** *conj-cong*: $P = P' \Longrightarrow (P' \Longrightarrow Q = Q') \Longrightarrow (P \land Q) = (P' \land Q')$
⟨*proof*⟩

**lemma** *rev-conj-cong*: $Q = Q' \Longrightarrow (Q' \Longrightarrow P = P') \Longrightarrow (P \land Q) = (P' \land Q')$
⟨*proof*⟩

The | congruence rule: not included by default!

**lemma** *disj-cong*: $P = P' \Longrightarrow (\neg\ P' \Longrightarrow Q = Q') \Longrightarrow (P \lor Q) = (P' \lor Q')$
⟨*proof*⟩

if-then-else rules

**lemma** *if-True* [*code*]: (*if True then x else y*) $= x$
⟨*proof*⟩

**lemma** *if-False* [*code*]: (*if False then x else y*) $= y$
⟨*proof*⟩

**lemma** *if-P*: $P \Longrightarrow$ (*if P then x else y*) $= x$
⟨*proof*⟩

**lemma** *if-not-P*: $\neg\ P \Longrightarrow (\mathit{if}\ P\ \mathit{then}\ x\ \mathit{else}\ y) = y$
⟨*proof*⟩

**lemma** *if-split*: $P\ (\mathit{if}\ Q\ \mathit{then}\ x\ \mathit{else}\ y) = ((Q \longrightarrow P\ x) \wedge (\neg\ Q \longrightarrow P\ y))$
⟨*proof*⟩

**lemma** *if-split-asm*: $P\ (\mathit{if}\ Q\ \mathit{then}\ x\ \mathit{else}\ y) = (\neg\ ((Q \wedge \neg\ P\ x) \vee (\neg\ Q \wedge \neg\ P\ y)))$
⟨*proof*⟩

**lemmas** *if-splits* [*no-atp*] = *if-split if-split-asm*

**lemma** *if-cancel*: $(\mathit{if}\ c\ \mathit{then}\ x\ \mathit{else}\ x) = x$
⟨*proof*⟩

**lemma** *if-eq-cancel*: $(\mathit{if}\ x = y\ \mathit{then}\ y\ \mathit{else}\ x) = x$
⟨*proof*⟩

**lemma** *if-bool-eq-conj*: $(\mathit{if}\ P\ \mathit{then}\ Q\ \mathit{else}\ R) = ((P \longrightarrow Q) \wedge (\neg\ P \longrightarrow R))$
— This form is useful for expanding *if*s on the RIGHT of the $\Longrightarrow$ symbol.
⟨*proof*⟩

**lemma** *if-bool-eq-disj*: $(\mathit{if}\ P\ \mathit{then}\ Q\ \mathit{else}\ R) = ((P \wedge Q) \vee (\neg\ P \wedge R))$
— And this form is useful for expanding *if*s on the LEFT.
⟨*proof*⟩

**lemma** *Eq-TrueI*: $P \Longrightarrow P \equiv \mathit{True}$ ⟨*proof*⟩
**lemma** *Eq-FalseI*: $\neg\ P \Longrightarrow P \equiv \mathit{False}$ ⟨*proof*⟩

## let rules for simproc

**lemma** *Let-folded*: $f\ x \equiv g\ x \Longrightarrow \mathit{Let}\ x\ f \equiv \mathit{Let}\ x\ g$
⟨*proof*⟩

**lemma** *Let-unfold*: $f\ x \equiv g \Longrightarrow \mathit{Let}\ x\ f \equiv g$
⟨*proof*⟩

The following copy of the implication operator is useful for fine-tuning congruence rules. It instructs the simplifier to simplify its premise.

**definition** *simp-implies* :: *prop* ⇒ *prop* ⇒ *prop* (**infixr** ‹=*simp*=>› *1*)
**where** *simp-implies* ≡ (⟹)

**lemma** *simp-impliesI*:
**assumes** *PQ*: (*PROP P* ⟹ *PROP Q*)
**shows** *PROP P =simp=> PROP Q*
⟨*proof*⟩

**lemma** *simp-impliesE*:
**assumes** *PQ*: *PROP P =simp=> PROP Q*
**and** *P*: *PROP P*

**and** $QR$: $PROP\ Q \Longrightarrow PROP\ R$
**shows** $PROP\ R$
⟨*proof*⟩

**lemma** *simp-implies-cong*:
**assumes** $PP'$ :$PROP\ P \equiv PROP\ P'$
**and** $P'QQ'$: $PROP\ P' \Longrightarrow (PROP\ Q \equiv PROP\ Q')$
**shows** $(PROP\ P =simp=> PROP\ Q) \equiv (PROP\ P' =simp=> PROP\ Q')$
⟨*proof*⟩

**lemma** *uncurry*:
**assumes** $P \longrightarrow Q \longrightarrow R$
**shows** $P \wedge Q \longrightarrow R$
⟨*proof*⟩

**lemma** *iff-allI*:
**assumes** $\bigwedge x.\ P\ x = Q\ x$
**shows** $(\forall x.\ P\ x) = (\forall x.\ Q\ x)$
⟨*proof*⟩

**lemma** *iff-exI*:
**assumes** $\bigwedge x.\ P\ x = Q\ x$
**shows** $(\exists x.\ P\ x) = (\exists x.\ Q\ x)$
⟨*proof*⟩

**lemma** *all-comm*: $(\forall x\ y.\ P\ x\ y) = (\forall y\ x.\ P\ x\ y)$
⟨*proof*⟩

**lemma** *ex-comm*: $(\exists x\ y.\ P\ x\ y) = (\exists y\ x.\ P\ x\ y)$
⟨*proof*⟩

⟨*ML*⟩

Simproc for proving $(y = x) \equiv False$ from premise $\neg\ (x = y)$:

⟨*ML*⟩

**lemma** *True-implies-equals*: $(True \Longrightarrow PROP\ P) \equiv PROP\ P$
⟨*proof*⟩

**lemma** *implies-True-equals*: $(PROP\ P \Longrightarrow True) \equiv Trueprop\ True$
⟨*proof*⟩

**lemma** *False-implies-equals*: $(False \Longrightarrow P) \equiv Trueprop\ True$
⟨*proof*⟩

**lemma** *implies-False-swap*:
$(False \Longrightarrow PROP\ P \Longrightarrow PROP\ Q) \equiv (PROP\ P \Longrightarrow False \Longrightarrow PROP\ Q)$
⟨*proof*⟩

⟨*ML*⟩

**lemma** *ex-simps*:
$\bigwedge P\ Q.\ (\exists x.\ P\ x \wedge Q) = ((\exists x.\ P\ x) \wedge Q)$
$\bigwedge P\ Q.\ (\exists x.\ P \wedge Q\ x) = (P \wedge (\exists x.\ Q\ x))$
$\bigwedge P\ Q.\ (\exists x.\ P\ x \vee Q) = ((\exists x.\ P\ x) \vee Q)$
$\bigwedge P\ Q.\ (\exists x.\ P \vee Q\ x) = (P \vee (\exists x.\ Q\ x))$
$\bigwedge P\ Q.\ (\exists x.\ P\ x \longrightarrow Q) = ((\forall x.\ P\ x) \longrightarrow Q)$
$\bigwedge P\ Q.\ (\exists x.\ P \longrightarrow Q\ x) = (P \longrightarrow (\exists x.\ Q\ x))$
— Miniscoping: pushing in existential quantifiers.
⟨*proof*⟩

**lemma** *all-simps*:
$\bigwedge P\ Q.\ (\forall x.\ P\ x \wedge Q) = ((\forall x.\ P\ x) \wedge Q)$
$\bigwedge P\ Q.\ (\forall x.\ P \wedge Q\ x) = (P \wedge (\forall x.\ Q\ x))$
$\bigwedge P\ Q.\ (\forall x.\ P\ x \vee Q) = ((\forall x.\ P\ x) \vee Q)$
$\bigwedge P\ Q.\ (\forall x.\ P \vee Q\ x) = (P \vee (\forall x.\ Q\ x))$
$\bigwedge P\ Q.\ (\forall x.\ P\ x \longrightarrow Q) = ((\exists x.\ P\ x) \longrightarrow Q)$
$\bigwedge P\ Q.\ (\forall x.\ P \longrightarrow Q\ x) = (P \longrightarrow (\forall x.\ Q\ x))$
— Miniscoping: pushing in universal quantifiers.
⟨*proof*⟩

**lemmas** [*simp*] =
*triv-forall-equality* — prunes params
*True-implies-equals implies-True-equals* — prune *True* in asms
*False-implies-equals* — prune *False* in asms
*if-True*
*if-False*
*if-cancel*
*if-eq-cancel*
*imp-disjL* — In general it seems wrong to add distributive laws by default: they might cause exponential blow-up. But *imp-disjL* has been in for a while and cannot be removed without affecting existing proofs. Moreover, rewriting by $(P \vee Q \longrightarrow R) = ((P \longrightarrow R) \wedge (Q \longrightarrow R))$ might be justified on the grounds that it allows simplification of $R$ in the two cases.
*conj-assoc*
*disj-assoc*
*de-Morgan-conj*
*de-Morgan-disj*
*imp-disj1*
*imp-disj2*
*not-imp*
*disj-not1*
*not-all*
*not-ex*
*cases-simp*
*the-eq-trivial*

*the-sym-eq-trivial*
*ex-simps*
*all-simps*
*simp-thms*
*subst-all*

**lemmas** [*cong*] = *imp-cong simp-implies-cong*
**lemmas** [*split*] = *if-split*

⟨*ML*⟩

Simplifies $x$ assuming $c$ and $y$ assuming $\neg\ c$.

**lemma** *if-cong*:
**assumes** $b = c$
**and** $c \Longrightarrow x = u$
**and** $\neg\ c \Longrightarrow y = v$
**shows** $(\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y) = (\mathit{if}\ c\ \mathit{then}\ u\ \mathit{else}\ v)$
⟨*proof*⟩

Prevents simplification of $x$ and $y$: faster and allows the execution of functional programs.

**lemma** *if-weak-cong* [*cong*]:
**assumes** $b = c$
**shows** $(\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y) = (\mathit{if}\ c\ \mathit{then}\ x\ \mathit{else}\ y)$
⟨*proof*⟩

Prevents simplification of t: much faster

**lemma** *let-weak-cong*:
**assumes** $a = b$
**shows** $(\mathit{let}\ x = a\ \mathit{in}\ t\ x) = (\mathit{let}\ x = b\ \mathit{in}\ t\ x)$
⟨*proof*⟩

To tidy up the result of a simproc. Only the RHS will be simplified.

**lemma** *eq-cong2*:
**assumes** $u = u'$
**shows** $(t \equiv u) \equiv (t \equiv u')$
⟨*proof*⟩

**lemma** *if-distrib*: $f\ (\mathit{if}\ c\ \mathit{then}\ x\ \mathit{else}\ y) = (\mathit{if}\ c\ \mathit{then}\ f\ x\ \mathit{else}\ f\ y)$
⟨*proof*⟩

**lemma** *if-distribR*: $(\mathit{if}\ b\ \mathit{then}\ f\ \mathit{else}\ g)\ x = (\mathit{if}\ b\ \mathit{then}\ f\ x\ \mathit{else}\ g\ x)$
⟨*proof*⟩

**lemma** *all-if-distrib*: $(\forall x.\ \mathit{if}\ x = a\ \mathit{then}\ P\ x\ \mathit{else}\ Q\ x) \longleftrightarrow P\ a \wedge (\forall x.\ x{\neq}a \longrightarrow Q\ x)$
⟨*proof*⟩

**lemma** *ex-if-distrib*: $(\exists\, x.\ \mathit{if}\ x = a\ \mathit{then}\ P\ x\ \mathit{else}\ Q\ x) \longleftrightarrow P\ a \vee (\exists\, x.\ x{\neq}a \wedge Q\ x)$
⟨*proof*⟩

**lemma** *if-if-eq-conj*: $(\mathit{if}\ P\ \mathit{then}\ \mathit{if}\ Q\ \mathit{then}\ x\ \mathit{else}\ y\ \mathit{else}\ y) = (\mathit{if}\ P \wedge Q\ \mathit{then}\ x\ \mathit{else}\ y)$
⟨*proof*⟩

As a simplification rule, it replaces all function equalities by first-order equalities.

**lemma** *fun-eq-iff*: $f = g \longleftrightarrow (\forall\, x.\ f\ x = g\ x)$
⟨*proof*⟩

### 2.3.5 Generic cases and induction

Rule projections:

⟨*ML*⟩

**context**
**begin**

**qualified definition** *induct-forall* $P \equiv \forall\, x.\ P\ x$
**qualified definition** *induct-implies* $A\ B \equiv A \longrightarrow B$
**qualified definition** *induct-equal* $x\ y \equiv x = y$
**qualified definition** *induct-conj* $A\ B \equiv A \wedge B$
**qualified definition** *induct-true* $\equiv$ *True*
**qualified definition** *induct-false* $\equiv$ *False*

**lemma** *induct-forall-eq*: $(\bigwedge x.\ P\ x) \equiv \mathit{Trueprop}\ (\mathit{induct\text{-}forall}\ (\lambda x.\ P\ x))$
⟨*proof*⟩

**lemma** *induct-implies-eq*: $(A \Longrightarrow B) \equiv \mathit{Trueprop}\ (\mathit{induct\text{-}implies}\ A\ B)$
⟨*proof*⟩

**lemma** *induct-equal-eq*: $(x \equiv y) \equiv \mathit{Trueprop}\ (\mathit{induct\text{-}equal}\ x\ y)$
⟨*proof*⟩

**lemma** *induct-conj-eq*: $(A\ \&\&\&\ B) \equiv \mathit{Trueprop}\ (\mathit{induct\text{-}conj}\ A\ B)$
⟨*proof*⟩

**lemmas** *induct-atomize'* = *induct-forall-eq induct-implies-eq induct-conj-eq*
**lemmas** *induct-atomize* = *induct-atomize' induct-equal-eq*
**lemmas** *induct-rulify'* [*symmetric*] = *induct-atomize'*
**lemmas** *induct-rulify* [*symmetric*] = *induct-atomize*
**lemmas** *induct-rulify-fallback* =
*induct-forall-def induct-implies-def induct-equal-def induct-conj-def*
*induct-true-def induct-false-def*

**lemma** *induct-forall-conj*: $\mathit{induct\text{-}forall}\ (\lambda x.\ \mathit{induct\text{-}conj}\ (A\ x)\ (B\ x)) =$

*induct-conj* (*induct-forall* $A$) (*induct-forall* $B$)
⟨*proof*⟩

**lemma** *induct-implies-conj*: *induct-implies* $C$ (*induct-conj* $A$ $B$) $=$
*induct-conj* (*induct-implies* $C$ $A$) (*induct-implies* $C$ $B$)
⟨*proof*⟩

**lemma** *induct-conj-curry*: (*induct-conj* $A$ $B \Longrightarrow$ *PROP* $C$) $\equiv$ ($A \Longrightarrow B \Longrightarrow$ *PROP* $C$)
⟨*proof*⟩

**lemmas** *induct-conj* = *induct-forall-conj induct-implies-conj induct-conj-curry*

**lemma** *induct-trueI*: *induct-true*
⟨*proof*⟩

Method setup.

⟨*ML*⟩

Pre-simplification of induction and cases rules

**lemma** [*induct-simp*]: ($\bigwedge x$. *induct-equal* $x$ $t \Longrightarrow$ *PROP* $P$ $x$) $\equiv$ *PROP* $P$ $t$
⟨*proof*⟩

**lemma** [*induct-simp*]: ($\bigwedge x$. *induct-equal* $t$ $x \Longrightarrow$ *PROP* $P$ $x$) $\equiv$ *PROP* $P$ $t$
⟨*proof*⟩

**lemma** [*induct-simp*]: (*induct-false* $\Longrightarrow P$) $\equiv$ *Trueprop induct-true*
⟨*proof*⟩

**lemma** [*induct-simp*]: (*induct-true* $\Longrightarrow$ *PROP* $P$) $\equiv$ *PROP* $P$
⟨*proof*⟩

**lemma** [*induct-simp*]: (*PROP* $P \Longrightarrow$ *induct-true*) $\equiv$ *Trueprop induct-true*
⟨*proof*⟩

**lemma** [*induct-simp*]: ($\bigwedge x::'a::\{\}$. *induct-true*) $\equiv$ *Trueprop induct-true*
⟨*proof*⟩

**lemma** [*induct-simp*]: *induct-implies induct-true* $P \equiv P$
⟨*proof*⟩

**lemma** [*induct-simp*]: $x = x \longleftrightarrow$ *True*
⟨*proof*⟩

**end**

⟨*ML*⟩

### 2.3.6 Coherent logic

⟨*ML*⟩

### 2.3.7 Reorienting equalities

⟨*ML*⟩

## 2.4 Other simple lemmas and lemma duplicates

**lemma** *eq-iff-swap*: $(x = y \longleftrightarrow P) \Longrightarrow (y = x \longleftrightarrow P)$
⟨*proof*⟩

**lemma** *all-cong1*: $(\bigwedge x.\ P\ x = P'\ x) \Longrightarrow (\forall x.\ P\ x) = (\forall x.\ P'\ x)$
⟨*proof*⟩

**lemma** *ex-cong1*: $(\bigwedge x.\ P\ x = P'\ x) \Longrightarrow (\exists x.\ P\ x) = (\exists x.\ P'\ x)$
⟨*proof*⟩

**lemma** *all-cong*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x = P'\ x) \Longrightarrow (\forall x.\ Q\ x \longrightarrow P\ x) = (\forall x.\ Q\ x \longrightarrow P'\ x)$
⟨*proof*⟩

**lemma** *ex-cong*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x = P'\ x) \Longrightarrow (\exists x.\ Q\ x \wedge P\ x) = (\exists x.\ Q\ x \wedge P'\ x)$
⟨*proof*⟩

**lemma** *ex1-eq* [*iff*]: $\exists!x.\ x = t$ $\exists!x.\ t = x$
⟨*proof*⟩

**lemma** *choice-eq*: $(\forall x.\ \exists!y.\ P\ x\ y) = (\exists!f.\ \forall x.\ P\ x\ (f\ x))$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemmas** *eq-sym-conv* = *eq-commute*

**lemma** *nnf-simps*:
$(\neg\ (P \wedge Q)) = (\neg\ P \vee \neg\ Q)$
$(\neg\ (P \vee Q)) = (\neg\ P \wedge \neg\ Q)$
$(P \longrightarrow Q) = (\neg\ P \vee Q)$
$(P = Q) = ((P \wedge Q) \vee (\neg\ P \wedge \neg\ Q))$
$(\neg\ (P = Q)) = ((P \wedge \neg\ Q) \vee (\neg\ P \wedge Q))$
$(\neg\ \neg\ P) = P$
⟨*proof*⟩

## 2.5 Basic ML bindings

⟨*ML*⟩

**locale** *cnf*
**begin**

**lemma** *clause2raw-notE*: $[\![P;\ \neg P]\!] \Longrightarrow$ *False* $\langle proof \rangle$
**lemma** *clause2raw-not-disj*: $[\![\neg\ P;\ \neg\ Q]\!] \Longrightarrow \neg\ (P \vee Q)$ $\langle proof \rangle$
**lemma** *clause2raw-not-not*: $P \Longrightarrow \neg\neg\ P$ $\langle proof \rangle$

**lemma** *iff-refl*: $(P{::}bool) = P$ $\langle proof \rangle$
**lemma** *iff-trans*: $[|\ (P{::}bool) = Q;\ Q = R\ |] ==> P = R$ $\langle proof \rangle$
**lemma** *conj-cong*: $[|\ P = P';\ Q = Q'\ |] ==> (P \wedge Q) = (P' \wedge Q')$ $\langle proof \rangle$
**lemma** *disj-cong*: $[|\ P = P';\ Q = Q'\ |] ==> (P \vee Q) = (P' \vee Q')$ $\langle proof \rangle$

**lemma** *make-nnf-imp*: $[|\ (\neg P) = P';\ Q = Q'\ |] ==> (P \longrightarrow Q) = (P' \vee Q')$ $\langle proof \rangle$
**lemma** *make-nnf-iff*: $[|\ P = P';\ (\neg P) = NP;\ Q = Q';\ (\neg Q) = NQ\ |] ==> (P = Q) = ((P' \vee NQ) \wedge (NP \vee Q'))$ $\langle proof \rangle$
**lemma** *make-nnf-not-false*: $(\neg False) = True$ $\langle proof \rangle$
**lemma** *make-nnf-not-true*: $(\neg True) = False$ $\langle proof \rangle$
**lemma** *make-nnf-not-conj*: $[|\ (\neg P) = P';\ (\neg Q) = Q'\ |] ==> (\neg(P \wedge Q)) = (P' \vee Q')$ $\langle proof \rangle$
**lemma** *make-nnf-not-disj*: $[|\ (\neg P) = P';\ (\neg Q) = Q'\ |] ==> (\neg(P \vee Q)) = (P' \wedge Q')$ $\langle proof \rangle$
**lemma** *make-nnf-not-imp*: $[|\ P = P';\ (\neg Q) = Q'\ |] ==> (\neg(P \longrightarrow Q)) = (P' \wedge Q')$ $\langle proof \rangle$
**lemma** *make-nnf-not-iff*: $[|\ P = P';\ (\neg P) = NP;\ Q = Q';\ (\neg Q) = NQ\ |] ==> (\neg(P = Q)) = ((P' \vee Q') \wedge (NP \vee NQ))$ $\langle proof \rangle$
**lemma** *make-nnf-not-not*: $P = P' ==> (\neg\neg P) = P'$ $\langle proof \rangle$

**lemma** *simp-TF-conj-True-l*: $[|\ P = True;\ Q = Q'\ |] ==> (P \wedge Q) = Q'$ $\langle proof \rangle$
**lemma** *simp-TF-conj-True-r*: $[|\ P = P';\ Q = True\ |] ==> (P \wedge Q) = P'$ $\langle proof \rangle$
**lemma** *simp-TF-conj-False-l*: $P = False ==> (P \wedge Q) = False$ $\langle proof \rangle$
**lemma** *simp-TF-conj-False-r*: $Q = False ==> (P \wedge Q) = False$ $\langle proof \rangle$
**lemma** *simp-TF-disj-True-l*: $P = True ==> (P \vee Q) = True$ $\langle proof \rangle$
**lemma** *simp-TF-disj-True-r*: $Q = True ==> (P \vee Q) = True$ $\langle proof \rangle$
**lemma** *simp-TF-disj-False-l*: $[|\ P = False;\ Q = Q'\ |] ==> (P \vee Q) = Q'$ $\langle proof \rangle$
**lemma** *simp-TF-disj-False-r*: $[|\ P = P';\ Q = False\ |] ==> (P \vee Q) = P'$ $\langle proof \rangle$

**lemma** *make-cnf-disj-conj-l*: $[|\ (P \vee R) = PR;\ (Q \vee R) = QR\ |] ==> ((P \wedge Q) \vee R) = (PR \wedge QR)$ $\langle proof \rangle$
**lemma** *make-cnf-disj-conj-r*: $[|\ (P \vee Q) = PQ;\ (P \vee R) = PR\ |] ==> (P \vee (Q \wedge R)) = (PQ \wedge PR)$ $\langle proof \rangle$

**lemma** *make-cnfx-disj-ex-l*: $((\exists\,(x{::}bool).\ P\ x) \vee Q) = (\exists\,x.\ P\ x \vee Q)$ $\langle proof \rangle$
**lemma** *make-cnfx-disj-ex-r*: $(P \vee (\exists\,(x{::}bool).\ Q\ x)) = (\exists\,x.\ P \vee Q\ x)$ $\langle proof \rangle$
**lemma** *make-cnfx-newlit*: $(P \vee Q) = (\exists\,x.\ (P \vee x) \wedge (Q \vee \neg x))$ $\langle proof \rangle$
**lemma** *make-cnfx-ex-cong*: $(\forall\,(x{::}bool).\ P\ x = Q\ x) \Longrightarrow (\exists\,x.\ P\ x) = (\exists\,x.\ Q\ x)$ $\langle proof \rangle$

**lemma** *weakening-thm*: $[|\ P;\ Q\ |] ==> Q$ $\langle proof \rangle$

**lemma** *cnftac-eq-imp*: $[|\ P = Q;\ P\ |] ==> Q$ $\langle proof \rangle$

**end**

⟨*ML*⟩

# 3 *NO-MATCH* simproc

The simplification procedure can be used to avoid simplification of terms of a certain form.

**definition** $NO\text{-}MATCH :: 'a \Rightarrow 'b \Rightarrow bool$
  **where** $NO\text{-}MATCH\ pat\ val \equiv True$

**lemma** $NO\text{-}MATCH\text{-}cong[cong]$: $NO\text{-}MATCH\ pat\ val = NO\text{-}MATCH\ pat\ val$
  ⟨*proof*⟩

**declare** [[*coercion-args NO-MATCH* − −]]

⟨*ML*⟩

This setup ensures that a rewrite rule of the form $NO\text{-}MATCH\ pat\ val \Longrightarrow t$ is only applied, if the pattern *pat* does not match the value *val*.

Tagging a premise of a simp rule with ASSUMPTION forces the simplifier not to simplify the argument and to solve it by an assumption.

**definition** $ASSUMPTION :: bool \Rightarrow bool$
  **where** $ASSUMPTION\ A \equiv A$

**lemma** $ASSUMPTION\text{-}cong[cong]$: $ASSUMPTION\ A = ASSUMPTION\ A$
  ⟨*proof*⟩

**lemma** $ASSUMPTION\text{-}I$: $A \Longrightarrow ASSUMPTION\ A$
  ⟨*proof*⟩

**lemma** $ASSUMPTION\text{-}D$: $ASSUMPTION\ A \Longrightarrow A$
  ⟨*proof*⟩

⟨*ML*⟩

## 3.1 Code generator setup

### 3.1.1 Generic code generator preprocessor setup

**lemma** *conj-left-cong*: $P \longleftrightarrow Q \Longrightarrow P \wedge R \longleftrightarrow Q \wedge R$
  ⟨*proof*⟩

**lemma** *disj-left-cong*: $P \longleftrightarrow Q \Longrightarrow P \vee R \longleftrightarrow Q \vee R$
  ⟨*proof*⟩

⟨*ML*⟩

### 3.1.2 Generic code generator foundation

Datatype *bool*

**code-datatype** *True False*

**lemma** [*code*]:
$P \wedge True \longleftrightarrow P$
$P \wedge False \longleftrightarrow False$
$True \wedge P \longleftrightarrow P$
$False \wedge P \longleftrightarrow False$
⟨*proof*⟩

**lemma** [*code*]:
$P \vee True \longleftrightarrow True$
$P \vee False \longleftrightarrow P$
$True \vee P \longleftrightarrow True$
$False \vee P \longleftrightarrow P$
⟨*proof*⟩

**lemma** [*code*]:
$(P \longrightarrow True) \longleftrightarrow True$
$(P \longrightarrow False) \longleftrightarrow \neg P$
$(True \longrightarrow P) \longleftrightarrow P$
$(False \longrightarrow P) \longleftrightarrow True$
⟨*proof*⟩

More about *prop*

**lemma** [*code nbe*]:
$(P \Longrightarrow R) \equiv Trueprop\ (P \longrightarrow R)$
$(PROP\ Q \Longrightarrow True) \equiv Trueprop\ True$
$(True \Longrightarrow PROP\ Q) \equiv PROP\ Q$
⟨*proof*⟩

**lemma** *Trueprop-code* [*code*]: $Trueprop\ True \equiv Code\text{-}Generator.holds$
⟨*proof*⟩

**declare** *Trueprop-code* [*symmetric*, *code-post*]

Cases

**lemma** *Let-case-cert*:
**assumes** $CASE \equiv (\lambda x.\ Let\ x\ f)$
**shows** $CASE\ x \equiv f\ x$
⟨*proof*⟩

⟨*ML*⟩

**declare** [[*code abort*: *undefined*]]

### 3.1.3 Equality

**lemma** [*code nbe*]:
‹$x = x \longleftrightarrow True$›
⟨*proof*⟩

**class** *equal* =
**fixes** $equal :: 'a \Rightarrow 'a \Rightarrow bool$
**assumes** *equal-eq*: $equal\ x\ y \longleftrightarrow x = y$
**begin**

**lemma** *eq-equal* [*code*]: $(=) \equiv equal$
⟨*proof*⟩

**lemma** *equal* [*code-post*]: $equal = (=)$
⟨*proof*⟩

**lemma** *equal-refl*: $equal\ x\ x \longleftrightarrow True$
⟨*proof*⟩

**end**

⟨*ML*⟩

**instantiation** *itself* :: (*type*) *equal*
**begin**

**definition** *equal-itself* :: $'a\ itself \Rightarrow 'a\ itself \Rightarrow bool$
**where** *equal-itself* $x\ y \longleftrightarrow x = y$

**instance**
⟨*proof*⟩

**end**

**lemma** *equal-itself-code* [*code*]: $equal\ TYPE('a)\ TYPE('a) \longleftrightarrow True$
⟨*proof*⟩

⟨*ML*⟩

**lemma** *equal-alias-cert*: $OFCLASS('a,\ equal\text{-}class) \equiv (((=) :: 'a \Rightarrow 'a \Rightarrow bool) \equiv equal)$
(**is** *?ofclass* ≡ *?equal*)
⟨*proof*⟩

⟨*ML*⟩

### 3.1.4 Generic code generator target languages

type *bool*

**code-printing**
**type-constructor** *bool* ⇀
(*SML*) *bool* **and** (*OCaml*) *bool* **and** (*Haskell*) *Bool* **and** (*Scala*) *Boolean*
| **constant** *True* ⇀
(*SML*) *true* **and** (*OCaml*) *true* **and** (*Haskell*) *True* **and** (*Scala*) *true*
| **constant** *False* ⇀
(*SML*) *false* **and** (*OCaml*) *false* **and** (*Haskell*) *False* **and** (*Scala*) *false*

**code-reserved**
(*SML*) *bool true false*
**and** (*OCaml*) *bool*
**and** (*Scala*) *Boolean*

**code-printing**
**constant** *Not* ⇀
(*SML*) *not* **and** (*OCaml*) *not* **and** (*Haskell*) *not* **and** (*Scala*) '! -
| **constant** *HOL.conj* ⇀
(*SML*) **infixl** *1 andalso* **and** (*OCaml*) **infixl** *3* && **and** (*Haskell*) **infixr** *3* && **and** (*Scala*) **infixl** *3* &&
| **constant** *HOL.disj* ⇀
(*SML*) **infixl** *0 orelse* **and** (*OCaml*) **infixl** *2* || **and** (*Haskell*) **infixl** *2* || **and** (*Scala*) **infixl** *1* ||
| **constant** *HOL.implies* ⇀
(*SML*) !(*if* (-)/ *then* (-)/ *else true*)
**and** (*OCaml*) !(*if* (-)/ *then* (-)/ *else true*)
**and** (*Haskell*) !(*if* (-)/ *then* (-)/ *else True*)
**and** (*Scala*) !((-) *match* {/ *case true* => (-)/ *case false* => *true*/ })
| **constant** *If* ⇀
(*SML*) !(*if* (-)/ *then* (-)/ *else* (-))
**and** (*OCaml*) !(*if* (-)/ *then* (-)/ *else* (-))
**and** (*Haskell*) !(*if* (-)/ *then* (-)/ *else* (-))
**and** (*Scala*) !((-) *match* {/ *case true* => (-)/ *case false* => (-)/ })

**code-reserved**
(*SML*) *not*
**and** (*OCaml*) *not*

**code-identifier**
**code-module** *Pure* ⇀
(*SML*) *HOL* **and** (*OCaml*) *HOL* **and** (*Haskell*) *HOL* **and** (*Scala*) *HOL*

Using built-in Haskell equality.

**code-printing**
**type-class** *equal* ⇀ (*Haskell*) *Eq*
| **constant** *HOL.equal* ⇀ (*Haskell*) **infix** *4* ==
| **constant** *HOL.eq* ⇀ (*Haskell*) **infix** *4* ==

*undefined*

**code-printing**

**constant** *undefined* ⇀
(*SML*) !(*raise*/ *Fail*/ *undefined*)
**and** (*OCaml*) *failwith*/ *undefined*
**and** (*Haskell*) *error*/ *undefined*
**and** (*Scala*) !*sys.error*(*undefined*)

### 3.1.5 Evaluation and normalization by evaluation

⟨*ML*⟩

## 3.2 Counterexample Search Units

### 3.2.1 Quickcheck

**quickcheck-params** [*size* = *5*, *iterations* = *50*]

### 3.2.2 Nitpick setup

**named-theorems** *nitpick-unfold alternative definitions of constants as needed by Nitpick*
**and** *nitpick-simp equational specification of constants as needed by Nitpick*
**and** *nitpick-psimp partial equational specification of constants as needed by Nitpick*
**and** *nitpick-choice-spec choice specification of constants as needed by Nitpick*

**declare** *if-bool-eq-conj* [*nitpick-unfold*, *no-atp*]
**and** *if-bool-eq-disj* [*no-atp*]

## 3.3 Preprocessing for the predicate compiler

**named-theorems** *code-pred-def alternative definitions of constants for the Predicate Compiler*
**and** *code-pred-inline inlining definitions for the Predicate Compiler*
**and** *code-pred-simp simplification rules for the optimisations in the Predicate Compiler*

## 3.4 Legacy tactics and ML bindings

⟨*ML*⟩

**hide-const** (**open**) *eq equal*

**end**

# 4 Abstract orderings

**theory** *Orderings*
**imports** *HOL*
**keywords** *print-orders* :: *diag*
**begin**

## 4.1 Abstract ordering

**locale** *partial-preordering* =
  **fixes** *less-eq* :: ‹$'a \Rightarrow 'a \Rightarrow bool$› (**infix** ‹≤› *50*)
  **assumes** *refl*: ‹$a \leq a$› — not *iff*: makes problems due to multiple (dual) interpretations
    **and** *trans*: ‹$a \leq b \Longrightarrow b \leq c \Longrightarrow a \leq c$›

**locale** *preordering* = *partial-preordering* +
  **fixes** *less* :: ‹$'a \Rightarrow 'a \Rightarrow bool$› (**infix** ‹<› *50*)
  **assumes** *strict-iff-not*: ‹$a < b \longleftrightarrow a \leq b \wedge \neg\, b \leq a$›
**begin**

**lemma** *strict-implies-order*:
  ‹$a < b \Longrightarrow a \leq b$›
  ⟨*proof*⟩

**lemma** *irrefl*: — not *iff*: makes problems due to multiple (dual) interpretations
  ‹$\neg\, a < a$›
  ⟨*proof*⟩

**lemma** *asym*:
  ‹$a < b \Longrightarrow b < a \Longrightarrow False$›
  ⟨*proof*⟩

**lemma** *strict-trans1*:
  ‹$a \leq b \Longrightarrow b < c \Longrightarrow a < c$›
  ⟨*proof*⟩

**lemma** *strict-trans2*:
  ‹$a < b \Longrightarrow b \leq c \Longrightarrow a < c$›
  ⟨*proof*⟩

**lemma** *strict-trans*:
  ‹$a < b \Longrightarrow b < c \Longrightarrow a < c$›
  ⟨*proof*⟩

**end**

**lemma** *preordering-strictI*: — Alternative introduction rule with bias towards strict order
  **fixes** *less-eq* (**infix** ‹≤› *50*)
    **and** *less* (**infix** ‹<› *50*)
  **assumes** *less-eq-less*: ‹$\bigwedge a\ b.\ a \leq b \longleftrightarrow a < b \vee a = b$›
    **assumes** *asym*: ‹$\bigwedge a\ b.\ a < b \Longrightarrow \neg\, b < a$›
  **assumes** *irrefl*: ‹$\bigwedge a.\ \neg\, a < a$›
  **assumes** *trans*: ‹$\bigwedge a\ b\ c.\ a < b \Longrightarrow b < c \Longrightarrow a < c$›
  **shows** ‹*preordering* (≤) (<)›
⟨*proof*⟩

**lemma** *preordering-dualI*:
  **fixes** *less-eq* (**infix** ‹$\leq$› *50*)
    **and** *less* (**infix** ‹$<$› *50*)
  **assumes** ‹*preordering* $(\lambda a\ b.\ b \leq a)$ $(\lambda a\ b.\ b < a)$›
  **shows** ‹*preordering* $(\leq)$ $(<)$›
⟨*proof*⟩

**locale** *ordering* = *partial-preordering* +
  **fixes** *less* :: ‹$'a \Rightarrow 'a \Rightarrow$ *bool*› (**infix** ‹$<$› *50*)
  **assumes** *strict-iff-order*: ‹$a < b \longleftrightarrow a \leq b \wedge a \neq b$›
  **assumes** *antisym*: ‹$a \leq b \Longrightarrow b \leq a \Longrightarrow a = b$›
**begin**

**sublocale** *preordering* ‹$(\leq)$› ‹$(<)$›
⟨*proof*⟩

**lemma** *strict-implies-not-eq*:
  ‹$a < b \Longrightarrow a \neq b$›
  ⟨*proof*⟩

**lemma** *not-eq-order-implies-strict*:
  ‹$a \neq b \Longrightarrow a \leq b \Longrightarrow a < b$›
  ⟨*proof*⟩

**lemma** *order-iff-strict*:
  ‹$a \leq b \longleftrightarrow a < b \vee a = b$›
  ⟨*proof*⟩

**lemma** *eq-iff*: ‹$a = b \longleftrightarrow a \leq b \wedge b \leq a$›
  ⟨*proof*⟩

**end**

**lemma** *ordering-strictI*: — Alternative introduction rule with bias towards strict order
  **fixes** *less-eq* (**infix** ‹$\leq$› *50*)
    **and** *less* (**infix** ‹$<$› *50*)
  **assumes** *less-eq-less*: ‹$\bigwedge a\ b.\ a \leq b \longleftrightarrow a < b \vee a = b$›
    **assumes** *asym*: ‹$\bigwedge a\ b.\ a < b \Longrightarrow \neg\ b < a$›
  **assumes** *irrefl*: ‹$\bigwedge a.\ \neg\ a < a$›
  **assumes** *trans*: ‹$\bigwedge a\ b\ c.\ a < b \Longrightarrow b < c \Longrightarrow a < c$›
  **shows** ‹*ordering* $(\leq)$ $(<)$›
⟨*proof*⟩

**lemma** *ordering-dualI*:
  **fixes** *less-eq* (**infix** ‹$\leq$› *50*)
    **and** *less* (**infix** ‹$<$› *50*)
  **assumes** ‹*ordering* $(\lambda a\ b.\ b \leq a)$ $(\lambda a\ b.\ b < a)$›
  **shows** ‹*ordering* $(\leq)$ $(<)$›

⟨*proof*⟩

**locale** *ordering-top* = *ordering* +
  **fixes** *top* :: ‹'a› (‹⊤›)
  **assumes** *extremum* [*simp*]: ‹a ≤ ⊤›
**begin**

**lemma** *extremum-uniqueI*:
  ‹⊤ ≤ a ⟹ a = ⊤›
  ⟨*proof*⟩

**lemma** *extremum-unique*:
  ‹⊤ ≤ a ⟷ a = ⊤›
  ⟨*proof*⟩

**lemma** *extremum-strict* [*simp*]:
  ‹¬ (⊤ < a)›
  ⟨*proof*⟩

**lemma** *not-eq-extremum*:
  ‹a ≠ ⊤ ⟷ a < ⊤›
  ⟨*proof*⟩

**end**

## 4.2 Syntactic orders

**class** *ord* =
  **fixes** *less-eq* :: 'a ⇒ 'a ⇒ *bool*
    **and** *less* :: 'a ⇒ 'a ⇒ *bool*
**begin**

**notation**
  *less-eq* (‹'(≤')›) **and**
  *less-eq* (‹(‹notation=‹infix ≤››-/ ≤ -)› [51, 51] 50) **and**
  *less* (‹'(<')›) **and**
  *less* (‹(‹notation=‹infix <››-/ < -)› [51, 51] 50)

**abbreviation** (*input*)
  *greater-eq* (**infix** ‹≥› 50)
  **where** x ≥ y ≡ y ≤ x

**abbreviation** (*input*)
  *greater* (**infix** ‹>› 50)
  **where** x > y ≡ y < x

**notation** (*ASCII*)
  *less-eq* (‹'(<=')›) **and**
  *less-eq* (‹(‹notation=‹infix <=››-/ <= -)› [51, 51] 50)

**notation** (*input*)
*greater-eq* (**infix** ‹>=› *50*)

**end**

## 4.3 Quasi orders

**class** *preorder* = *ord* +
**assumes** *less-le-not-le*: $x < y \longleftrightarrow x \leq y \wedge \neg (y \leq x)$
**and** *order-refl* [*iff*]: $x \leq x$
**and** *order-trans*: $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
**begin**

**sublocale** *order*: *preordering less-eq less* + *dual-order*: *preordering greater-eq greater*
⟨*proof*⟩

Reflexivity.

**lemma** *eq-refl*: $x = y \Longrightarrow x \leq y$
— This form is useful with the classical reasoner.
⟨*proof*⟩

**lemma** *less-irrefl* [*iff*]: $\neg\ x < x$
⟨*proof*⟩

**lemma** *less-imp-le*: $x < y \Longrightarrow x \leq y$
⟨*proof*⟩

Asymmetry.

**lemma** *less-not-sym*: $x < y \Longrightarrow \neg (y < x)$
⟨*proof*⟩

**lemma** *less-asym*: $x < y \Longrightarrow (\neg P \Longrightarrow y < x) \Longrightarrow P$
⟨*proof*⟩

Transitivity.

**lemma** *less-trans*: $x < y \Longrightarrow y < z \Longrightarrow x < z$
⟨*proof*⟩

**lemma** *le-less-trans*: $x \leq y \Longrightarrow y < z \Longrightarrow x < z$
⟨*proof*⟩

**lemma** *less-le-trans*: $x < y \Longrightarrow y \leq z \Longrightarrow x < z$
⟨*proof*⟩

Useful for simplification, but too risky to include by default.

**lemma** *less-imp-not-less*: $x < y \Longrightarrow (\neg\ y < x) \longleftrightarrow True$
⟨*proof*⟩

**lemma** *less-imp-triv*: $x < y \Longrightarrow (y < x \longrightarrow P) \longleftrightarrow True$
⟨*proof*⟩

Transitivity rules for calculational reasoning

**lemma** *less-asym'*: $a < b \Longrightarrow b < a \Longrightarrow P$
⟨*proof*⟩

Dual order

**lemma** *dual-preorder*:
‹*class.preorder* $(\geq)$ $(>)$›
⟨*proof*⟩

**end**

**lemma** *preordering-preorderI*:
‹*class.preorder* $(\leq)$ $(<)$› **if** ‹*preordering* $(\leq)$ $(<)$›
**for** *less-eq* (**infix** ‹$\leq$› *50*) **and** *less* (**infix** ‹$<$› *50*)
⟨*proof*⟩

## 4.4 Partial orders

**class** *order* = *preorder* +
**assumes** *order-antisym*: $x \leq y \Longrightarrow y \leq x \Longrightarrow x = y$
**begin**

**lemma** *less-le*: $x < y \longleftrightarrow x \leq y \wedge x \neq y$
⟨*proof*⟩

**sublocale** *order*: *ordering less-eq less* + *dual-order*: *ordering greater-eq greater*
⟨*proof*⟩

Reflexivity.

**lemma** *le-less*: $x \leq y \longleftrightarrow x < y \vee x = y$
— NOT suitable for iff, since it can cause PROOF FAILED.
⟨*proof*⟩

**lemma** *le-imp-less-or-eq*: $x \leq y \Longrightarrow x < y \vee x = y$
⟨*proof*⟩

Useful for simplification, but too risky to include by default.

**lemma** *less-imp-not-eq*: $x < y \Longrightarrow (x = y) \longleftrightarrow False$
⟨*proof*⟩

**lemma** *less-imp-not-eq2*: $x < y \Longrightarrow (y = x) \longleftrightarrow False$
⟨*proof*⟩

Transitivity rules for calculational reasoning

**lemma** *neq-le-trans*: $a \neq b \Longrightarrow a \leq b \Longrightarrow a < b$

⟨*proof*⟩

**lemma** *le-neq-trans*: $a \leq b \Longrightarrow a \neq b \Longrightarrow a < b$
⟨*proof*⟩

Asymmetry.

**lemma** *order-eq-iff*: $x = y \longleftrightarrow x \leq y \wedge y \leq x$
⟨*proof*⟩

**lemma** *antisym-conv*: $y \leq x \Longrightarrow x \leq y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *less-imp-neq*: $x < y \Longrightarrow x \neq y$
⟨*proof*⟩

**lemma** *antisym-conv1*: $\neg\, x < y \Longrightarrow x \leq y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *antisym-conv2*: $x \leq y \Longrightarrow \neg\, x < y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *leD*: $y \leq x \Longrightarrow \neg\, x < y$
⟨*proof*⟩

Least value operator

**definition** (**in** *ord*)
*Least* :: $('a \Rightarrow bool) \Rightarrow 'a$ (**binder** ‹*LEAST* › *10*) **where**
$Least\ P = (THE\ x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow x \leq y))$

**lemma** *Least-equality*:
**assumes** $P\ x$
**and** $\bigwedge y.\ P\ y \Longrightarrow x \leq y$
**shows** $Least\ P = x$
⟨*proof*⟩

**lemma** *LeastI2-order*:
**assumes** $P\ x$
**and** $\bigwedge y.\ P\ y \Longrightarrow x \leq y$
**and** $\bigwedge x.\ P\ x \Longrightarrow \forall\, y.\ P\ y \longrightarrow x \leq y \Longrightarrow Q\ x$
**shows** $Q\ (Least\ P)$
⟨*proof*⟩

**lemma** *Least-ex1*:
**assumes** $\exists!x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow x \leq y)$
**shows** *Least1I*: $P\ (Least\ P)$ **and** *Least1-le*: $P\ z \Longrightarrow Least\ P \leq z$
⟨*proof*⟩

Greatest value operator

**definition** *Greatest* :: $('a \Rightarrow bool) \Rightarrow 'a$ (**binder** ‹*GREATEST* › *10*) **where**

$Greatest\ P = (THE\ x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow x \geq y))$

**lemma** *GreatestI2-order*:
$\llbracket\ P\ x;$
$\bigwedge y.\ P\ y \Longrightarrow x \geq y;$
$\bigwedge x.\ \llbracket\ P\ x;\ \forall\, y.\ P\ y \longrightarrow x \geq y\ \rrbracket \Longrightarrow Q\ x\ \rrbracket$
$\Longrightarrow Q\ (Greatest\ P)$
$\langle proof \rangle$

**lemma** *Greatest-equality*:
$\llbracket\ P\ x;\ \ \bigwedge y.\ P\ y \Longrightarrow x \geq y\ \rrbracket \Longrightarrow Greatest\ P = x$
$\langle proof \rangle$

**end**

**lemma** *ordering-orderI*:
**fixes** *less-eq* (**infix** ‹≤› *50*)
**and** *less* (**infix** ‹<› *50*)
**assumes** *ordering less-eq less*
**shows** *class.order less-eq less*
$\langle proof \rangle$

**lemma** *order-strictI*:
**fixes** *less* (**infix** ‹<› *50*)
**and** *less-eq* (**infix** ‹≤› *50*)
**assumes** $\bigwedge a\ b.\ a \leq b \longleftrightarrow a < b \vee a = b$
**assumes** $\bigwedge a\ b.\ a < b \Longrightarrow \neg\ b < a$
**assumes** $\bigwedge a.\ \neg\ a < a$
**assumes** $\bigwedge a\ b\ c.\ a < b \Longrightarrow b < c \Longrightarrow a < c$
**shows** *class.order less-eq less*
$\langle proof \rangle$

**context** *order*
**begin**

Dual order

**lemma** *dual-order*:
*class.order* $(\geq)$ $(>)$
$\langle proof \rangle$

**end**

## 4.5 Linear (total) orders

**class** *linorder* = *order* +
**assumes** *linear*: $x \leq y \vee y \leq x$
**begin**

**lemma** *less-linear*: $x < y \vee x = y \vee y < x$

⟨*proof*⟩

**lemma** *le-less-linear*: $x \leq y \vee y < x$
⟨*proof*⟩

**lemma** *le-cases* [*case-names le ge*]:
$(x \leq y \Longrightarrow P) \Longrightarrow (y \leq x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** (**in** *linorder*) *le-cases3*:
$[\![ [\![x \leq y;\ y \leq z]\!] \Longrightarrow P;\ [\![y \leq x;\ x \leq z]\!] \Longrightarrow P;\ [\![x \leq z;\ z \leq y]\!] \Longrightarrow P;$
$[\![z \leq y;\ y \leq x]\!] \Longrightarrow P;\ [\![y \leq z;\ z \leq x]\!] \Longrightarrow P;\ [\![z \leq x;\ x \leq y]\!] \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *linorder-cases* [*case-names less equal greater*]:
$(x < y \Longrightarrow P) \Longrightarrow (x = y \Longrightarrow P) \Longrightarrow (y < x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *linorder-wlog*[*case-names le sym*]:
$(\bigwedge a\ b.\ a \leq b \Longrightarrow P\ a\ b) \Longrightarrow (\bigwedge a\ b.\ P\ b\ a \Longrightarrow P\ a\ b) \Longrightarrow P\ a\ b$
⟨*proof*⟩

**lemma** *not-less*: $\neg\ x < y \longleftrightarrow y \leq x$
⟨*proof*⟩

**lemma** *not-less-iff-gr-or-eq*: $\neg(x < y) \longleftrightarrow (x > y \vee x = y)$
⟨*proof*⟩

**lemma** *not-le*: $\neg\ x \leq y \longleftrightarrow y < x$
⟨*proof*⟩

**lemma** *neq-iff*: $x \neq y \longleftrightarrow x < y \vee y < x$
⟨*proof*⟩

**lemma** *neqE*: $x \neq y \Longrightarrow (x < y \Longrightarrow R) \Longrightarrow (y < x \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *antisym-conv3*: $\neg\ y < x \Longrightarrow \neg\ x < y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *leI*: $\neg\ x < y \Longrightarrow y \leq x$
⟨*proof*⟩

**lemma** *not-le-imp-less*: $\neg\ y \leq x \Longrightarrow x < y$
⟨*proof*⟩

**lemma** *linorder-less-wlog*[*case-names less refl sym*]:
$[\![\bigwedge a\ b.\ a < b \Longrightarrow P\ a\ b;\ \ \bigwedge a.\ P\ a\ a;\ \ \bigwedge a\ b.\ P\ b\ a \Longrightarrow P\ a\ b]\!] \Longrightarrow P\ a\ b$
⟨*proof*⟩

Dual order

**lemma** *dual-linorder*:
  *class.linorder* $(\geq)$ $(>)$
⟨*proof*⟩

**end**

Alternative introduction rule with bias towards strict order

**lemma** *linorder-strictI*:
  **fixes** *less-eq* (**infix** ‹≤› *50*)
    **and** *less* (**infix** ‹<› *50*)
  **assumes** *class.order less-eq less*
  **assumes** *trichotomy*: $\bigwedge a\ b.\ a < b \vee a = b \vee b < a$
  **shows** *class.linorder less-eq less*
⟨*proof*⟩

## 4.6 Reasoning tools setup

⟨*ML*⟩

The method *order* allows one to use the order tactic located in `../Provers/order_tac.ML` in a standalone fashion.

The tactic rearranges the goal to prove *False*, then retrieves order literals of partial and linear orders (i.e. $x = y$, $x \leq y$, $x < y$, and their negated versions) from the premises and finally tries to derive a contradiction. Its main use case is as a solver to *simp* (see below), where it e.g. solves premises of conditional rewrite rules.

The tactic has two configuration attributes that control its behaviour:

- *order-trace* toggles tracing for the solver.
- *order-split-limit* limits the number of order literals of the form $\neg\ x < y$ that are passed to the tactic. This is helpful since these literals lead to case splitting and thus exponential runtime. This only applies to partial orders.

  We setup the solver for HOL with the structure `HOL_Order_Tac` here but the prover is agnostic to the object logic. It is possible to register orders with the prover using the functions `HOL_Order_Tac.declare_order` and `HOL_Order_Tac.declare_linorder`, which we do below for the type classes *order* and *linorder*. If possible, one should instantiate these type classes instead of registering new orders with the solver. One can also interpret the type class locales *order* and *linorder*. An example can be seen in `Library/Sublist.thy`, which contains e.g. the prefix order on lists.

  The diagnostic command **print-orders** shows all orders known to the tactic in the current context.

Declarations to set up transitivity reasoner of partial and linear orders.

**context** *order*
**begin**

**lemma** *nless-le*: $(\neg\ a < b) \longleftrightarrow (\neg\ a \leq b) \vee a = b$
⟨*proof*⟩

⟨*ML*⟩

**end**

**context** *linorder*
**begin**

**lemma** *nle-le*: $(\neg\ a \leq b) \longleftrightarrow b \leq a \wedge b \neq a$
⟨*proof*⟩

⟨*ML*⟩

**end**

⟨*ML*⟩

## 4.7 Bounded quantifiers

**syntax** (*ASCII*)
-All-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ALL››ALL -<-./ -)› [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder EX››EX -<-./ -)› [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ALL››ALL -<=-./ -)› [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder EX››EX -<=-./ -)› [0, 0, 10] 10)

-All-greater :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ALL››ALL ->-./ -)› [0, 0, 10] 10)
-Ex-greater :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder EX››EX ->-./ -)› [0, 0, 10] 10)
-All-greater-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ALL››ALL ->=-./ -)› [0, 0, 10] 10)
-Ex-greater-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder EX››EX ->=-./ -)› [0, 0, 10] 10)

-All-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ALL››ALL -~=-./ -)› [0, 0, 10] 10)
-Ex-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder EX››EX -~=-./ -)› [0, 0, 10] 10)

**syntax**
-All-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∀››∀ -<-./ -)› [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∃››∃ -<-./ -)› [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∀››∀ -≤-./ -)› [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∃››∃ -≤-./ -)› [0, 0, 10] 10)

-All-greater :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∀››∀ ->-./ -)› [0, 0, 10] 10)
-Ex-greater :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∃››∃ ->-./ -)› [0, 0, 10] 10)
-All-greater-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∀››∀ -≥-./ -)› [0, 0, 10] 10)
-Ex-greater-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∃››∃ -≥-./ -)› [0, 0, 10] 10)

-All-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∀››∀ -≠-./ -)› [0, 0, 10] 10)
-Ex-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ∃››∃ -≠-./ -)› [0, 0, 10] 10)

**syntax** (input)
-All-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder !››! -<-./ -)› [0, 0, 10] 10)
-Ex-less :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ?››? -<-./ -)› [0, 0, 10] 10)
-All-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder !››! -<=-./ -)› [0, 0, 10] 10)
-Ex-less-eq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ?››? -<=-./ -)› [0, 0, 10] 10)
-All-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder !››! -~=-./ -)› [0, 0, 10] 10)
-Ex-neq :: [idt, 'a, bool] => bool (‹(‹indent=3 notation=‹binder ?››? -~=-./ -)› [0, 0, 10] 10)

**syntax-consts**
-All-less -All-less-eq -All-greater -All-greater-eq -All-neq ⇌ All **and**
-Ex-less -Ex-less-eq -Ex-greater -Ex-greater-eq -Ex-neq ⇌ Ex

**translations**
$\forall x{<}y.\ P \rightharpoonup \forall x.\ x < y \longrightarrow P$
$\exists x{<}y.\ P \rightharpoonup \exists x.\ x < y \wedge P$
$\forall x{\le}y.\ P \rightharpoonup \forall x.\ x \le y \longrightarrow P$
$\exists x{\le}y.\ P \rightharpoonup \exists x.\ x \le y \wedge P$
$\forall x{>}y.\ P \rightharpoonup \forall x.\ x > y \longrightarrow P$
$\exists x{>}y.\ P \rightharpoonup \exists x.\ x > y \wedge P$

$\forall x{\geq}y.\ P \rightharpoonup \forall x.\ x \geq y \longrightarrow P$
$\exists x{\geq}y.\ P \rightharpoonup \exists x.\ x \geq y \wedge P$
$\forall x{\neq}y.\ P \rightharpoonup \forall x.\ x \neq y \longrightarrow P$
$\exists x{\neq}y.\ P \rightharpoonup \exists x.\ x \neq y \wedge P$

⟨*ML*⟩

## 4.8 Transitivity reasoning

**context** *ord*
**begin**

**lemma** *ord-le-eq-trans*: $a \leq b \Longrightarrow b = c \Longrightarrow a \leq c$
⟨*proof*⟩

**lemma** *ord-eq-le-trans*: $a = b \Longrightarrow b \leq c \Longrightarrow a \leq c$
⟨*proof*⟩

**lemma** *ord-less-eq-trans*: $a < b \Longrightarrow b = c \Longrightarrow a < c$
⟨*proof*⟩

**lemma** *ord-eq-less-trans*: $a = b \Longrightarrow b < c \Longrightarrow a < c$
⟨*proof*⟩

**end**

**lemma** *order-less-subst2*: $(a::'a::order) < b \Longrightarrow f\ b < (c::'c::order) \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow f\ a < c$
⟨*proof*⟩

**lemma** *order-less-subst1*: $(a::'a::order) < f\ b \Longrightarrow (b::'b::order) < c \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow a < f\ c$
⟨*proof*⟩

**lemma** *order-le-less-subst2*: $(a::'a::order) <= b \Longrightarrow f\ b < (c::'c::order) \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow f\ a < c$
⟨*proof*⟩

**lemma** *order-le-less-subst1*: $(a::'a::order) <= f\ b \Longrightarrow (b::'b::order) < c \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow a < f\ c$
⟨*proof*⟩

**lemma** *order-less-le-subst2*: $(a::'a::order) < b \Longrightarrow f\ b <= (c::'c::order) \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow f\ a < c$
⟨*proof*⟩

**lemma** *order-less-le-subst1*: $(a::'a::order) < f\ b \Longrightarrow (b::'b::order) <= c \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow a < f\ c$
⟨*proof*⟩

**lemma** *order-subst1*: $(a::'a::order) <= f\ b \Longrightarrow (b::'b::order) <= c \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow a <= f\ c$
$\langle proof \rangle$

**lemma** *order-subst2*: $(a::'a::order) <= b \Longrightarrow f\ b <= (c::'c::order) \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow f\ a <= c$
$\langle proof \rangle$

**lemma** *ord-le-eq-subst*: $a <= b \Longrightarrow f\ b = c \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow f\ a <= c$
$\langle proof \rangle$

**lemma** *ord-eq-le-subst*: $a = f\ b \Longrightarrow b <= c \Longrightarrow$
$(!!x\ y.\ x <= y \Longrightarrow f\ x <= f\ y) \Longrightarrow a <= f\ c$
$\langle proof \rangle$

**lemma** *ord-less-eq-subst*: $a < b \Longrightarrow f\ b = c \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow f\ a < c$
$\langle proof \rangle$

**lemma** *ord-eq-less-subst*: $a = f\ b \Longrightarrow b < c \Longrightarrow$
$(!!x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow a < f\ c$
$\langle proof \rangle$

Note that this list of rules is in reverse order of priorities.

**lemmas** [*trans*] =
*order-less-subst2*
*order-less-subst1*
*order-le-less-subst2*
*order-le-less-subst1*
*order-less-le-subst2*
*order-less-le-subst1*
*order-subst2*
*order-subst1*
*ord-le-eq-subst*
*ord-eq-le-subst*
*ord-less-eq-subst*
*ord-eq-less-subst*
*forw-subst*
*back-subst*
*rev-mp*
*mp*

**lemmas** (**in** *order*) [*trans*] =
*neq-le-trans*
*le-neq-trans*

**lemmas** (**in** *preorder*) [*trans*] =

*less-trans*
*less-asym'*
*le-less-trans*
*less-le-trans*
*order-trans*

**lemmas** (**in** *order*) [*trans*] =
*order.antisym*

**lemmas** (**in** *ord*) [*trans*] =
*ord-le-eq-trans*
*ord-eq-le-trans*
*ord-less-eq-trans*
*ord-eq-less-trans*

**lemmas** [*trans*] =
*trans*

**lemmas** *order-trans-rules* =
*order-less-subst2*
*order-less-subst1*
*order-le-less-subst2*
*order-le-less-subst1*
*order-less-le-subst2*
*order-less-le-subst1*
*order-subst2*
*order-subst1*
*ord-le-eq-subst*
*ord-eq-le-subst*
*ord-less-eq-subst*
*ord-eq-less-subst*
*forw-subst*
*back-subst*
*rev-mp*
*mp*
*neq-le-trans*
*le-neq-trans*
*less-trans*
*less-asym'*
*le-less-trans*
*less-le-trans*
*order-trans*
*order.antisym*
*ord-le-eq-trans*
*ord-eq-le-trans*
*ord-less-eq-trans*
*ord-eq-less-trans*
*trans*

These support proving chains of decreasing inequalities a $\geq$ b $\geq$ c ... in Isar proofs.

**lemma** *xt1* [*no-atp*]:
$a = b \Longrightarrow b > c \Longrightarrow a > c$
$a > b \Longrightarrow b = c \Longrightarrow a > c$
$a = b \Longrightarrow b \geq c \Longrightarrow a \geq c$
$a \geq b \Longrightarrow b = c \Longrightarrow a \geq c$
$(x::'a::order) \geq y \Longrightarrow y \geq x \Longrightarrow x = y$
$(x::'a::order) \geq y \Longrightarrow y \geq z \Longrightarrow x \geq z$
$(x::'a::order) > y \Longrightarrow y \geq z \Longrightarrow x > z$
$(x::'a::order) \geq y \Longrightarrow y > z \Longrightarrow x > z$
$(a::'a::order) > b \Longrightarrow b > a \Longrightarrow P$
$(x::'a::order) > y \Longrightarrow y > z \Longrightarrow x > z$
$(a::'a::order) \geq b \Longrightarrow a \neq b \Longrightarrow a > b$
$(a::'a::order) \neq b \Longrightarrow a \geq b \Longrightarrow a > b$
$a = f\ b \Longrightarrow b > c \Longrightarrow (\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y) \Longrightarrow a > f\ c$
$a > b \Longrightarrow f\ b = c \Longrightarrow (\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y) \Longrightarrow f\ a > c$
$a = f\ b \Longrightarrow b \geq c \Longrightarrow (\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y) \Longrightarrow a \geq f\ c$
$a \geq b \Longrightarrow f\ b = c \Longrightarrow (\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y) \Longrightarrow f\ a \geq c$
⟨*proof*⟩

**lemma** *xt2* [*no-atp*]:
**assumes** $(a::'a::order) \geq f\ b$
**and** $b \geq c$
**and** $\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y$
**shows** $a \geq f\ c$
⟨*proof*⟩

**lemma** *xt3* [*no-atp*]:
**assumes** $(a::'a::order) \geq b$
**and** $(f\ b::'b::order) \geq c$
**and** $\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y$
**shows** $f\ a \geq c$
⟨*proof*⟩

**lemma** *xt4* [*no-atp*]:
**assumes** $(a::'a::order) > f\ b$
**and** $(b::'b::order) \geq c$
**and** $\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y$
**shows** $a > f\ c$
⟨*proof*⟩

**lemma** *xt5* [*no-atp*]:
**assumes** $(a::'a::order) > b$
**and** $(f\ b::'b::order) \geq c$
**and** $\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y$
**shows** $f\ a > c$
⟨*proof*⟩

**lemma** *xt6* [*no-atp*]:

**assumes** $(a::'a::order) \geq f\ b$
**and** $b > c$
**and** $\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y$
**shows** $a > f\ c$
⟨*proof*⟩

**lemma** *xt7* [*no-atp*]:
**assumes** $(a::'a::order) \geq b$
**and** $(f\ b::'b::order) > c$
**and** $\bigwedge x\ y.\ x \geq y \Longrightarrow f\ x \geq f\ y$
**shows** $f\ a > c$
⟨*proof*⟩

**lemma** *xt8* [*no-atp*]:
**assumes** $(a::'a::order) > f\ b$
**and** $(b::'b::order) > c$
**and** $\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y$
**shows** $a > f\ c$
⟨*proof*⟩

**lemma** *xt9* [*no-atp*]:
**assumes** $(a::'a::order) > b$
**and** $(f\ b::'b::order) > c$
**and** $\bigwedge x\ y.\ x > y \Longrightarrow f\ x > f\ y$
**shows** $f\ a > c$
⟨*proof*⟩

**lemmas** *xtrans* = *xt1 xt2 xt3 xt4 xt5 xt6 xt7 xt8 xt9*

## 4.9 min and max – fundamental

**definition** (**in** *ord*) *min* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
$min\ a\ b = (if\ a \leq b\ then\ a\ else\ b)$

**definition** (**in** *ord*) *max* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
$max\ a\ b = (if\ a \leq b\ then\ b\ else\ a)$

**lemma** *min-absorb1*: $x \leq y \Longrightarrow min\ x\ y = x$
⟨*proof*⟩

**lemma** *max-absorb2*: $x \leq y \Longrightarrow max\ x\ y = y$
⟨*proof*⟩

**lemma** *min-absorb2*: $(y::'a::order) \leq x \Longrightarrow min\ x\ y = y$
⟨*proof*⟩

**lemma** *max-absorb1*: $(y::'a::order) \leq x \Longrightarrow max\ x\ y = x$
⟨*proof*⟩

**lemma** *max-min-same* [*simp*]:
  **fixes** $x\ y :: 'a :: linorder$
  **shows** $max\ x\ (min\ x\ y) = x$ $max\ (min\ x\ y)\ x = x$ $max\ (min\ x\ y)\ y = y$ $max\ y\ (min\ x\ y) = y$
⟨*proof*⟩

## 4.10 (Unique) top and bottom elements

**class** *bot* =
  **fixes** $bot :: 'a\ (\langle\bot\rangle)$

**class** *order-bot* = *order* + *bot* +
  **assumes** *bot-least*: $\bot \leq a$
**begin**

**sublocale** *bot*: *ordering-top greater-eq greater bot*
  ⟨*proof*⟩

**lemma** *le-bot*:
  $a \leq \bot \Longrightarrow a = \bot$
  ⟨*proof*⟩

**lemma** *bot-unique*:
  $a \leq \bot \longleftrightarrow a = \bot$
  ⟨*proof*⟩

**lemma** *not-less-bot*:
  $\neg\ a < \bot$
  ⟨*proof*⟩

**lemma** *bot-less*:
  $a \neq \bot \longleftrightarrow \bot < a$
  ⟨*proof*⟩

**lemma** *max-bot*[*simp*]: $max\ bot\ x = x$
⟨*proof*⟩

**lemma** *max-bot2*[*simp*]: $max\ x\ bot = x$
⟨*proof*⟩

**lemma** *min-bot*[*simp*]: $min\ bot\ x = bot$
⟨*proof*⟩

**lemma** *min-bot2*[*simp*]: $min\ x\ bot = bot$
⟨*proof*⟩

**end**

**class** *top* =

**fixes** *top* :: $'a$ (‹⊤›)

**class** *order-top* = *order* + *top* +
**assumes** *top-greatest*: $a \leq \top$
**begin**

**sublocale** *top*: *ordering-top less-eq less top*
⟨*proof*⟩

**lemma** *top-le*:
$\top \leq a \Longrightarrow a = \top$
⟨*proof*⟩

**lemma** *top-unique*:
$\top \leq a \longleftrightarrow a = \top$
⟨*proof*⟩

**lemma** *not-top-less*:
$\neg \top < a$
⟨*proof*⟩

**lemma** *less-top*:
$a \neq \top \longleftrightarrow a < \top$
⟨*proof*⟩

**lemma** *max-top*[*simp*]: *max top x* = *top*
⟨*proof*⟩

**lemma** *max-top2*[*simp*]: *max x top* = *top*
⟨*proof*⟩

**lemma** *min-top*[*simp*]: *min top x* = *x*
⟨*proof*⟩

**lemma** *min-top2*[*simp*]: *min x top* = *x*
⟨*proof*⟩

**end**

## 4.11 Dense orders

**class** *dense-order* = *order* +
**assumes** *dense*: $x < y \Longrightarrow (\exists z.\ x < z \wedge z < y)$

**class** *dense-linorder* = *linorder* + *dense-order*
**begin**

**lemma** *dense-le*:
**fixes** *y z* :: $'a$

**assumes** $\bigwedge x.\ x < y \Longrightarrow x \leq z$
**shows** $y \leq z$
$\langle proof \rangle$

**lemma** *dense-le-bounded*:
**fixes** $x\ y\ z :: 'a$
**assumes** $x < y$
**assumes** $*$: $\bigwedge w.\ [\![\ x < w\ ;\ w < y\ ]\!] \Longrightarrow w \leq z$
**shows** $y \leq z$
$\langle proof \rangle$

**lemma** *dense-ge*:
**fixes** $y\ z :: 'a$
**assumes** $\bigwedge x.\ z < x \Longrightarrow y \leq x$
**shows** $y \leq z$
$\langle proof \rangle$

**lemma** *dense-ge-bounded*:
**fixes** $x\ y\ z :: 'a$
**assumes** $z < x$
**assumes** $*$: $\bigwedge w.\ [\![\ z < w\ ;\ w < x\ ]\!] \Longrightarrow y \leq w$
**shows** $y \leq z$
$\langle proof \rangle$

**end**

**class** *no-top* = *order* +
**assumes** *gt-ex*: $\exists\, y.\ x < y$

**class** *no-bot* = *order* +
**assumes** *lt-ex*: $\exists\, y.\ y < x$

**class** *unbounded-dense-linorder* = *dense-linorder* + *no-top* + *no-bot*

**class** *unbounded-dense-order* = *dense-order* + *no-top* + *no-bot*

**instance** *unbounded-dense-linorder* $\subseteq$ *unbounded-dense-order* $\langle proof \rangle$

## 4.12 Wellorders

**class** *wellorder* = *linorder* +
**assumes** *less-induct* [*case-names less*]: $(\bigwedge x.\ (\bigwedge y.\ y < x \Longrightarrow P\ y) \Longrightarrow P\ x) \Longrightarrow P\ a$
**begin**

**lemma** *wellorder-Least-lemma*:
**fixes** $k :: 'a$
**assumes** $P\ k$
**shows** *LeastI*: $P\ (LEAST\ x.\ P\ x)$ **and** *Least-le*: $(LEAST\ x.\ P\ x) \leq k$

⟨*proof*⟩
**lemma** *LeastI-ex*: $\exists\, x.\ P\ x \Longrightarrow P\ (\mathit{Least}\ P)$
⟨*proof*⟩

**lemma** *LeastI2*:
$P\ a \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (\mathit{Least}\ P)$
⟨*proof*⟩

**lemma** *LeastI2-ex*:
$\exists\, a.\ P\ a \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (\mathit{Least}\ P)$
⟨*proof*⟩

**lemma** *LeastI2-wellorder*:
**assumes** $P\ a$
**and** $\bigwedge a.\ [\![\ P\ a;\ \forall\, b.\ P\ b \longrightarrow a \le b\ ]\!] \Longrightarrow Q\ a$
**shows** $Q\ (\mathit{Least}\ P)$
⟨*proof*⟩

**lemma** *LeastI2-wellorder-ex*:
**assumes** $\exists\, x.\ P\ x$
**and** $\bigwedge a.\ [\![\ P\ a;\ \forall\, b.\ P\ b \longrightarrow a \le b\ ]\!] \Longrightarrow Q\ a$
**shows** $Q\ (\mathit{Least}\ P)$
⟨*proof*⟩

**lemma** *not-less-Least*: $k < (\mathit{LEAST}\ x.\ P\ x) \Longrightarrow \neg\ P\ k$
⟨*proof*⟩

**lemma** *exists-least-iff*: $(\exists\, n.\ P\ n) \longleftrightarrow (\exists\, n.\ P\ n \wedge (\forall\, m < n.\ \neg\ P\ m))$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *exists-least-iff'*:
**shows** $(\exists\, n.\ P\ n) \longleftrightarrow P\ (\mathit{Least}\ P) \wedge (\forall\, m < (\mathit{Least}\ P).\ \neg\ P\ m)$
⟨*proof*⟩

**end**

## 4.13 Order on *bool*

**instantiation** *bool* :: {*order-bot*, *order-top*, *linorder*}
**begin**

**definition**
*le-bool-def* [*simp*]: $P \le Q \longleftrightarrow P \longrightarrow Q$

**definition**
[*simp*]: $(P{::}\mathit{bool}) < Q \longleftrightarrow \neg\ P \wedge Q$

**definition**

$[simp]$: $\bot \longleftrightarrow False$

**definition**
$[simp]$: $\top \longleftrightarrow True$

**instance** $\langle proof \rangle$

**end**

**lemma** *le-boolI*: $(P \Longrightarrow Q) \Longrightarrow P \leq Q$
$\langle proof \rangle$

**lemma** *le-boolI'*: $P \longrightarrow Q \Longrightarrow P \leq Q$
$\langle proof \rangle$

**lemma** *le-boolE*: $P \leq Q \Longrightarrow P \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

**lemma** *le-boolD*: $P \leq Q \Longrightarrow P \longrightarrow Q$
$\langle proof \rangle$

**lemma** *bot-boolE*: $\bot \Longrightarrow P$
$\langle proof \rangle$

**lemma** *top-boolI*: $\top$
$\langle proof \rangle$

**lemma** $[code]$:
$False \leq b \longleftrightarrow True$
$True \leq b \longleftrightarrow b$
$False < b \longleftrightarrow b$
$True < b \longleftrightarrow False$
$\langle proof \rangle$

## 4.14 Order on - ⇒ -

**instantiation** *fun* :: (*type*, *ord*) *ord*
**begin**

**definition**
*le-fun-def*: $f \leq g \longleftrightarrow (\forall x.\ f\ x \leq g\ x)$

**definition**
$(f{::}'a \Rightarrow 'b) < g \longleftrightarrow f \leq g \wedge \neg (g \leq f)$

**instance** $\langle proof \rangle$

**end**

**instance** *fun* :: (*type*, *preorder*) *preorder* ⟨*proof*⟩

**instance** *fun* :: (*type*, *order*) *order* ⟨*proof*⟩

**instantiation** *fun* :: (*type*, *bot*) *bot*
**begin**

**definition**
$\bot = (\lambda x.\ \bot)$

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *order-bot*) *order-bot*
**begin**

**lemma** *bot-apply* [*simp*, *code*]:
$\bot\ x = \bot$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *top*) *top*
**begin**

**definition**
[*no-atp*]: $\top = (\lambda x.\ \top)$

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *order-top*) *order-top*
**begin**

**lemma** *top-apply* [*simp*, *code*]:
$\top\ x = \top$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**lemma** *le-funI*: $(\bigwedge x.\ f\ x \le g\ x) \Longrightarrow f \le g$
⟨*proof*⟩

**lemma** *le-funE*: $f \leq g \Longrightarrow (f\ x \leq g\ x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *le-funD*: $f \leq g \Longrightarrow f\ x \leq g\ x$
⟨*proof*⟩

## 4.15 Order on unary and binary predicates

**lemma** *predicate1I*:
**assumes** *PQ*: $\bigwedge x.\ P\ x \Longrightarrow Q\ x$
**shows** $P \leq Q$
⟨*proof*⟩

**lemma** *predicate1D*:
$P \leq Q \Longrightarrow P\ x \Longrightarrow Q\ x$
⟨*proof*⟩

**lemma** *rev-predicate1D*:
$P\ x \Longrightarrow P \leq Q \Longrightarrow Q\ x$
⟨*proof*⟩

**lemma** *predicate2I*:
**assumes** *PQ*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow Q\ x\ y$
**shows** $P \leq Q$
⟨*proof*⟩

**lemma** *predicate2D*:
$P \leq Q \Longrightarrow P\ x\ y \Longrightarrow Q\ x\ y$
⟨*proof*⟩

**lemma** *rev-predicate2D*:
$P\ x\ y \Longrightarrow P \leq Q \Longrightarrow Q\ x\ y$
⟨*proof*⟩

**lemma** *bot1E* [*no-atp*]: $\bot\ x \Longrightarrow P$
⟨*proof*⟩

**lemma** *bot2E*: $\bot\ x\ y \Longrightarrow P$
⟨*proof*⟩

**lemma** *top1I*: $\top\ x$
⟨*proof*⟩

**lemma** *top2I*: $\top\ x\ y$
⟨*proof*⟩

## 4.16 Name duplicates

**lemmas** *antisym* = *order.antisym*
**lemmas** *eq-iff* = *order.eq-iff*

**lemmas** *order-eq-refl* = *preorder-class.eq-refl*
**lemmas** *order-less-irrefl* = *preorder-class.less-irrefl*
**lemmas** *order-less-imp-le* = *preorder-class.less-imp-le*
**lemmas** *order-less-not-sym* = *preorder-class.less-not-sym*
**lemmas** *order-less-asym* = *preorder-class.less-asym*
**lemmas** *order-less-trans* = *preorder-class.less-trans*
**lemmas** *order-le-less-trans* = *preorder-class.le-less-trans*
**lemmas** *order-less-le-trans* = *preorder-class.less-le-trans*
**lemmas** *order-less-imp-not-less* = *preorder-class.less-imp-not-less*
**lemmas** *order-less-imp-triv* = *preorder-class.less-imp-triv*
**lemmas** *order-less-asym′* = *preorder-class.less-asym′*

**lemmas** *order-less-le* = *order-class.less-le*
**lemmas** *order-le-less* = *order-class.le-less*
**lemmas** *order-le-imp-less-or-eq* = *order-class.le-imp-less-or-eq*
**lemmas** *order-less-imp-not-eq* = *order-class.less-imp-not-eq*
**lemmas** *order-less-imp-not-eq2* = *order-class.less-imp-not-eq2*
**lemmas** *order-neq-le-trans* = *order-class.neq-le-trans*
**lemmas** *order-le-neq-trans* = *order-class.le-neq-trans*
**lemmas** *order-eq-iff* = *order-class.order.eq-iff*
**lemmas** *order-antisym-conv* = *order-class.antisym-conv*

**lemmas** *linorder-linear* = *linorder-class.linear*
**lemmas** *linorder-less-linear* = *linorder-class.less-linear*
**lemmas** *linorder-le-less-linear* = *linorder-class.le-less-linear*
**lemmas** *linorder-le-cases* = *linorder-class.le-cases*
**lemmas** *linorder-not-less* = *linorder-class.not-less*
**lemmas** *linorder-not-le* = *linorder-class.not-le*
**lemmas** *linorder-neq-iff* = *linorder-class.neq-iff*
**lemmas** *linorder-neqE* = *linorder-class.neqE*

**end**

# 5 Groups, also combined with orderings

**theory** *Groups*
  **imports** *Orderings*
**begin**

## 5.1 Dynamic facts

**named-theorems** *ac-simps associativity and commutativity simplification rules*
  **and** *algebra-simps algebra simplification rules for rings*
  **and** *algebra-split-simps algebra simplification rules for rings, with potential goal splitting*
  **and** *field-simps algebra simplification rules for fields*
  **and** *field-split-simps algebra simplification rules for fields, with potential goal splitting*

The rewrites accumulated in *algebra-simps* deal with the classical algebraic structures of groups, rings and family. They simplify terms by multiplying everything out (in case of a ring) and bringing sums and products into a canonical form (by ordered rewriting). As a result it decides group and ring equalities but also helps with inequalities.

Of course it also works for fields, but it knows nothing about multiplicative inverses or division. This is catered for by *field-simps*.

Facts in *field-simps* multiply with denominators in (in)equations if they can be proved to be non-zero (for equations) or positive/negative (for inequalities). Can be too aggressive and is therefore separate from the more benign *algebra-simps*.

Collections *algebra-split-simps* and *field-split-simps* correspond to *algebra-simps* and *field-simps* but contain more aggresive rules that may lead to goal splitting.

## 5.2 Abstract structures

These locales provide basic structures for interpretation into bigger structures; extensions require careful thinking, otherwise undesired effects may occur due to interpretation.

**locale** *semigroup* =
  **fixes** $f :: {}'a \Rightarrow {}'a \Rightarrow {}'a$ (**infixl** ‹∗› *70*)
  **assumes** *assoc* [*ac-simps*]: $a * b * c = a * (b * c)$

**locale** *abel-semigroup* = *semigroup* +
  **assumes** *commute* [*ac-simps*]: $a * b = b * a$
**begin**

**lemma** *left-commute* [*ac-simps*]: $b * (a * c) = a * (b * c)$
⟨*proof*⟩

**end**

**locale** *monoid* = *semigroup* +
  **fixes** $z :: {}'a$ (‹**1**›)
  **assumes** *left-neutral* [*simp*]: $\mathbf{1} * a = a$
  **assumes** *right-neutral* [*simp*]: $a * \mathbf{1} = a$

**locale** *comm-monoid* = *abel-semigroup* +
  **fixes** $z :: {}'a$ (‹**1**›)
  **assumes** *comm-neutral*: $a * \mathbf{1} = a$
**begin**

**sublocale** *monoid*
  ⟨*proof*⟩

**end**

**locale** *group* = *semigroup* +
  **fixes** $z$ :: $'a$ (‹**1**›)
  **fixes** *inverse* :: $'a \Rightarrow 'a$
  **assumes** *group-left-neutral*: $\mathbf{1} * a = a$
  **assumes** *left-inverse* [*simp*]: *inverse* $a * a = \mathbf{1}$
**begin**

**lemma** *left-cancel*: $a * b = a * c \longleftrightarrow b = c$
⟨*proof*⟩

**sublocale** *monoid*
⟨*proof*⟩

**lemma** *inverse-unique*:
  **assumes** $a * b = \mathbf{1}$
  **shows** *inverse* $a = b$
⟨*proof*⟩

**lemma** *inverse-neutral* [*simp*]: *inverse* $\mathbf{1} = \mathbf{1}$
  ⟨*proof*⟩

**lemma** *inverse-inverse* [*simp*]: *inverse* (*inverse* $a$) = $a$
  ⟨*proof*⟩

**lemma** *right-inverse* [*simp*]: $a *$ *inverse* $a = \mathbf{1}$
⟨*proof*⟩

**lemma** *inverse-distrib-swap*: *inverse* $(a * b)$ = *inverse* $b *$ *inverse* $a$
⟨*proof*⟩

**lemma** *right-cancel*: $b * a = c * a \longleftrightarrow b = c$
⟨*proof*⟩

**end**

## 5.3 Generic operations

**class** *zero* =
  **fixes** *zero* :: $'a$ (‹*0*›)

**class** *one* =
  **fixes** *one* :: $'a$ (‹*1*›)

**hide-const** (**open**) *zero one*

**lemma** *Let-0* [*simp*]: *Let 0 f = f 0*
  ⟨*proof*⟩

**lemma** *Let-1* [*simp*]: *Let 1 f* = *f 1*
⟨*proof*⟩

⟨*ML*⟩

**class** *plus* =
**fixes** *plus* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ‹+› *65*)

**class** *minus* =
**fixes** *minus* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ‹−› *65*)

**class** *uminus* =
**fixes** *uminus* :: *'a* ⇒ *'a* (‹(‹*open-block notation*=‹*prefix* −››− -)› [*81*] *80*)

**class** *times* =
**fixes** *times* :: *'a* ⇒ *'a* ⇒ *'a* (**infixl** ‹∗› *70*)

**bundle** *uminus-syntax*
**begin**
**notation** *uminus* (‹(‹*open-block notation*=‹*prefix* −››− -)› [*81*] *80*)
**end**

## 5.4 Semigroups and Monoids

**class** *semigroup-add* = *plus* +
**assumes** *add-assoc*: (*a* + *b*) + *c* = *a* + (*b* + *c*)
**begin**

**sublocale** *add*: *semigroup plus*
⟨*proof*⟩

**declare** *add.assoc* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]

**end**

**hide-fact** *add-assoc*

**class** *ab-semigroup-add* = *semigroup-add* +
**assumes** *add-commute*: *a* + *b* = *b* + *a*
**begin**

**sublocale** *add*: *abel-semigroup plus*
⟨*proof*⟩

**declare** *add.commute* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]
*add.left-commute* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]

**lemmas** *add-ac* = *add.assoc add.commute add.left-commute*

**end**

**hide-fact** *add-commute*

**lemmas** *add-ac* = *add.assoc add.commute add.left-commute*

**class** *semigroup-mult* = *times* +
  **assumes** *mult-assoc*: $(a * b) * c = a * (b * c)$
**begin**

**sublocale** *mult*: *semigroup times*
  ⟨*proof*⟩

**declare** *mult.assoc* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]

**end**

**hide-fact** *mult-assoc*

**class** *ab-semigroup-mult* = *semigroup-mult* +
  **assumes** *mult-commute*: $a * b = b * a$
**begin**

**sublocale** *mult*: *abel-semigroup times*
  ⟨*proof*⟩

**declare** *mult.commute* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]
  *mult.left-commute* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]

**lemmas** *mult-ac* = *mult.assoc mult.commute mult.left-commute*

**end**

**hide-fact** *mult-commute*

**lemmas** *mult-ac* = *mult.assoc mult.commute mult.left-commute*

**class** *monoid-add* = *zero* + *semigroup-add* +
  **assumes** *add-0-left*: $0 + a = a$
    **and** *add-0-right*: $a + 0 = a$
**begin**

**sublocale** *add*: *monoid plus 0*
  ⟨*proof*⟩

**end**

**lemma** *zero-reorient*: $0 = x \longleftrightarrow x = 0$

⟨*proof*⟩

**class** *comm-monoid-add* = *zero* + *ab-semigroup-add* +
**assumes** *add-0*: $0 + a = a$
**begin**

**subclass** *monoid-add*
⟨*proof*⟩

**sublocale** *add*: *comm-monoid plus 0*
⟨*proof*⟩

**end**

**class** *monoid-mult* = *one* + *semigroup-mult* +
**assumes** *mult-1-left*: $1 * a = a$
**and** *mult-1-right*: $a * 1 = a$
**begin**

**sublocale** *mult*: *monoid times 1*
⟨*proof*⟩

**end**

**lemma** *one-reorient*: $1 = x \longleftrightarrow x = 1$
⟨*proof*⟩

**class** *comm-monoid-mult* = *one* + *ab-semigroup-mult* +
**assumes** *mult-1*: $1 * a = a$
**begin**

**subclass** *monoid-mult*
⟨*proof*⟩

**sublocale** *mult*: *comm-monoid times 1*
⟨*proof*⟩

**end**

**class** *cancel-semigroup-add* = *semigroup-add* +
**assumes** *add-left-imp-eq*: $a + b = a + c \Longrightarrow b = c$
**assumes** *add-right-imp-eq*: $b + a = c + a \Longrightarrow b = c$
**begin**

**lemma** *add-left-cancel* [*simp*]: $a + b = a + c \longleftrightarrow b = c$
⟨*proof*⟩

**lemma** *add-right-cancel* [*simp*]: $b + a = c + a \longleftrightarrow b = c$
⟨*proof*⟩

**end**

**class** *cancel-ab-semigroup-add* = *ab-semigroup-add* + *minus* +
**assumes** *add-diff-cancel-left'* [*simp*]: $(a + b) - a = b$
**assumes** *diff-diff-add* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a - b - c = a - (b + c)$
**begin**

**lemma** *add-diff-cancel-right'* [*simp*]: $(a + b) - b = a$
⟨*proof*⟩

**subclass** *cancel-semigroup-add*
⟨*proof*⟩

**lemma** *add-diff-cancel-left* [*simp*]: $(c + a) - (c + b) = a - b$
⟨*proof*⟩

**lemma** *add-diff-cancel-right* [*simp*]: $(a + c) - (b + c) = a - b$
⟨*proof*⟩

**lemma** *diff-right-commute*: $a - c - b = a - b - c$
⟨*proof*⟩

**end**

**class** *cancel-comm-monoid-add* = *cancel-ab-semigroup-add* + *comm-monoid-add*
**begin**

**lemma** *diff-zero* [*simp*]: $a - 0 = a$
⟨*proof*⟩

**lemma** *diff-cancel* [*simp*]: $a - a = 0$
⟨*proof*⟩

**lemma** *add-implies-diff*:
**assumes** $c + b = a$
**shows** $c = a - b$
⟨*proof*⟩

**lemma** *add-cancel-right-right* [*simp*]: $a = a + b \longleftrightarrow b = 0$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *add-cancel-right-left* [*simp*]: $a = b + a \longleftrightarrow b = 0$
⟨*proof*⟩

**lemma** *add-cancel-left-right* [*simp*]: $a + b = a \longleftrightarrow b = 0$
⟨*proof*⟩

**lemma** *add-cancel-left-left* [*simp*]: $b + a = a \longleftrightarrow b = 0$
⟨*proof*⟩

**end**

**class** *comm-monoid-diff* = *cancel-comm-monoid-add* +
**assumes** *zero-diff* [*simp*]: $0 - a = 0$
**begin**

**lemma** *diff-add-zero* [*simp*]: $a - (a + b) = 0$
⟨*proof*⟩

**end**

## 5.5 Groups

**class** *group-add* = *minus* + *uminus* + *monoid-add* +
**assumes** *left-minus*: $- a + a = 0$
**assumes** *add-uminus-conv-diff* [*simp*]: $a + (- b) = a - b$
**begin**

**lemma** *diff-conv-add-uminus*: $a - b = a + (- b)$
⟨*proof*⟩

**sublocale** *add*: *group plus 0 uminus*
⟨*proof*⟩

**lemma** *minus-unique*: $a + b = 0 \Longrightarrow - a = b$
⟨*proof*⟩

**lemma** *minus-zero*: $- 0 = 0$
⟨*proof*⟩

**lemma** *minus-minus*: $- (- a) = a$
⟨*proof*⟩

**lemma** *right-minus*: $a + - a = 0$
⟨*proof*⟩

**lemma** *diff-self* [*simp*]: $a - a = 0$
⟨*proof*⟩

**subclass** *cancel-semigroup-add*
⟨*proof*⟩

**lemma** *minus-add-cancel* [*simp*]: $- a + (a + b) = b$
⟨*proof*⟩

**lemma** *add-minus-cancel* [*simp*]: $a + (- a + b) = b$
⟨*proof*⟩

**lemma** *diff-add-cancel* [*simp*]: $a - b + b = a$
⟨*proof*⟩

**lemma** *add-diff-cancel* [*simp*]: $a + b - b = a$
⟨*proof*⟩

**lemma** *minus-add*: $- (a + b) = - b + - a$
⟨*proof*⟩

**lemma** *right-minus-eq* [*simp*]: $a - b = 0 \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *eq-iff-diff-eq-0*: $a = b \longleftrightarrow a - b = 0$
⟨*proof*⟩

**lemma** *diff-0* [*simp*]: $0 - a = - a$
⟨*proof*⟩

**lemma** *diff-0-right* [*simp*]: $a - 0 = a$
⟨*proof*⟩

**lemma** *diff-minus-eq-add* [*simp*]: $a - - b = a + b$
⟨*proof*⟩

**lemma** *neg-equal-iff-equal* [*simp*]: $- a = - b \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *neg-equal-0-iff-equal* [*simp*]: $- a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *neg-0-equal-iff-equal* [*simp*]: $0 = - a \longleftrightarrow 0 = a$
⟨*proof*⟩

The next two equations can make the simplifier loop!

**lemma** *equation-minus-iff*: $a = - b \longleftrightarrow b = - a$
⟨*proof*⟩

**lemma** *minus-equation-iff*: $- a = b \longleftrightarrow - b = a$
⟨*proof*⟩

**lemma** *eq-neg-iff-add-eq-0*: $a = - b \longleftrightarrow a + b = 0$
⟨*proof*⟩

**lemma** *add-eq-0-iff2*: $a + b = 0 \longleftrightarrow a = - b$
⟨*proof*⟩

**lemma** *neg-eq-iff-add-eq-0*: $- a = b \longleftrightarrow a + b = 0$
⟨*proof*⟩

**lemma** *add-eq-0-iff*: $a + b = 0 \longleftrightarrow b = - a$
⟨*proof*⟩

**lemma** *minus-diff-eq* [*simp*]: $- (a - b) = b - a$
⟨*proof*⟩

**lemma** *add-diff-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a + (b - c) = (a + b) - c$
⟨*proof*⟩

**lemma** *diff-add-eq-diff-diff-swap*: $a - (b + c) = a - c - b$
⟨*proof*⟩

**lemma** *diff-eq-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a - b = c \longleftrightarrow a = c + b$
⟨*proof*⟩

**lemma** *eq-diff-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a = c - b \longleftrightarrow a + b = c$
⟨*proof*⟩

**lemma** *diff-diff-eq2* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a - (b - c) = (a + c) - b$
⟨*proof*⟩

**lemma** *diff-eq-diff-eq*: $a - b = c - d \Longrightarrow a = b \longleftrightarrow c = d$
⟨*proof*⟩

**end**

**class** *ab-group-add* = *minus* + *uminus* + *comm-monoid-add* +
**assumes** *ab-left-minus*: $- a + a = 0$
**assumes** *ab-diff-conv-add-uminus*: $a - b = a + (- b)$
**begin**

**subclass** *group-add*
⟨*proof*⟩

**subclass** *cancel-comm-monoid-add*
⟨*proof*⟩

**lemma** *uminus-add-conv-diff* [*simp*]: $- a + b = b - a$
⟨*proof*⟩

**lemma** *minus-add-distrib* [*simp*]: $- (a + b) = - a + - b$
⟨*proof*⟩

**lemma** *diff-add-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$(a - b) + c = (a + c) - b$
⟨*proof*⟩

**lemma** *minus-diff-commute*:
$- b - a = - a - b$
⟨*proof*⟩

**end**

## 5.6 (Partially) Ordered Groups

The theory of partially ordered groups is taken from the books:

- *Lattice Theory* by Garret Birkhoff, American Mathematical Society, 1979
- *Partially Ordered Algebraic Systems*, Pergamon Press, 1963

Most of the used notions can also be looked up in

- http://www.mathworld.com by Eric Weisstein et. al.
- *Algebra I* by van der Waerden, Springer

**class** *ordered-ab-semigroup-add* = *order* + *ab-semigroup-add* +
**assumes** *add-left-mono*: $a \leq b \Longrightarrow c + a \leq c + b$
**begin**

**lemma** *add-right-mono*: $a \leq b \Longrightarrow a + c \leq b + c$
⟨*proof*⟩

non-strict, in both arguments

**lemma** *add-mono*: $a \leq b \Longrightarrow c \leq d \Longrightarrow a + c \leq b + d$
⟨*proof*⟩

**end**

Strict monotonicity in both arguments

**class** *strict-ordered-ab-semigroup-add* = *ordered-ab-semigroup-add* +
**assumes** *add-strict-mono*: $a < b \Longrightarrow c < d \Longrightarrow a + c < b + d$

**class** *ordered-cancel-ab-semigroup-add* =
*ordered-ab-semigroup-add* + *cancel-ab-semigroup-add*
**begin**

**lemma** *add-strict-left-mono*: $a < b \Longrightarrow c + a < c + b$

⟨*proof*⟩

**lemma** *add-strict-right-mono*: $a < b \Longrightarrow a + c < b + c$
⟨*proof*⟩

**subclass** *strict-ordered-ab-semigroup-add*
⟨*proof*⟩

**lemma** *add-less-le-mono*: $a < b \Longrightarrow c \leq d \Longrightarrow a + c < b + d$
⟨*proof*⟩

**lemma** *add-le-less-mono*: $a \leq b \Longrightarrow c < d \Longrightarrow a + c < b + d$
⟨*proof*⟩

**end**

**class** *ordered-ab-semigroup-add-imp-le* = *ordered-cancel-ab-semigroup-add* +
**assumes** *add-le-imp-le-left*: $c + a \leq c + b \Longrightarrow a \leq b$
**begin**

**lemma** *add-less-imp-less-left*:
**assumes** *less*: $c + a < c + b$
**shows** $a < b$
⟨*proof*⟩

**lemma** *add-less-imp-less-right*: $a + c < b + c \Longrightarrow a < b$
⟨*proof*⟩

**lemma** *add-less-cancel-left* [*simp*]: $c + a < c + b \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *add-less-cancel-right* [*simp*]: $a + c < b + c \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *add-le-cancel-left* [*simp*]: $c + a \leq c + b \longleftrightarrow a \leq b$
⟨*proof*⟩

**lemma** *add-le-cancel-right* [*simp*]: $a + c \leq b + c \longleftrightarrow a \leq b$
⟨*proof*⟩

**lemma** *add-le-imp-le-right*: $a + c \leq b + c \Longrightarrow a \leq b$
⟨*proof*⟩

**lemma** *max-add-distrib-left*: $max\ x\ y + z = max\ (x + z)\ (y + z)$
⟨*proof*⟩

**lemma** *min-add-distrib-left*: $min\ x\ y + z = min\ (x + z)\ (y + z)$
⟨*proof*⟩

**lemma** *max-add-distrib-right*: $x + max\ y\ z = max\ (x + y)\ (x + z)$
⟨*proof*⟩

**lemma** *min-add-distrib-right*: $x + min\ y\ z = min\ (x + y)\ (x + z)$
⟨*proof*⟩

**end**

## 5.7 Support for reasoning about signs

**class** *ordered-comm-monoid-add* = *comm-monoid-add* + *ordered-ab-semigroup-add*
**begin**

**lemma** *add-nonneg-nonneg* [*simp*]: $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a + b$
⟨*proof*⟩

**lemma** *add-nonpos-nonpos*: $a \leq 0 \Longrightarrow b \leq 0 \Longrightarrow a + b \leq 0$
⟨*proof*⟩

**lemma** *add-nonneg-eq-0-iff*: $0 \leq x \Longrightarrow 0 \leq y \Longrightarrow x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *add-nonpos-eq-0-iff*: $x \leq 0 \Longrightarrow y \leq 0 \Longrightarrow x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *add-increasing*: $0 \leq a \Longrightarrow b \leq c \Longrightarrow b \leq a + c$
⟨*proof*⟩

**lemma** *add-increasing2*: $0 \leq c \Longrightarrow b \leq a \Longrightarrow b \leq a + c$
⟨*proof*⟩

**lemma** *add-decreasing*: $a \leq 0 \Longrightarrow c \leq b \Longrightarrow a + c \leq b$
⟨*proof*⟩

**lemma** *add-decreasing2*: $c \leq 0 \Longrightarrow a \leq b \Longrightarrow a + c \leq b$
⟨*proof*⟩

**lemma** *add-pos-nonneg*: $0 < a \Longrightarrow 0 \leq b \Longrightarrow 0 < a + b$
⟨*proof*⟩

**lemma** *add-pos-pos*: $0 < a \Longrightarrow 0 < b \Longrightarrow 0 < a + b$
⟨*proof*⟩

**lemma** *add-nonneg-pos*: $0 \leq a \Longrightarrow 0 < b \Longrightarrow 0 < a + b$
⟨*proof*⟩

**lemma** *add-neg-nonpos*: $a < 0 \Longrightarrow b \leq 0 \Longrightarrow a + b < 0$
⟨*proof*⟩

**lemma** *add-neg-neg*: $a < 0 \Longrightarrow b < 0 \Longrightarrow a + b < 0$
⟨*proof*⟩

**lemma** *add-nonpos-neg*: $a \leq 0 \Longrightarrow b < 0 \Longrightarrow a + b < 0$
⟨*proof*⟩

**lemmas** *add-sign-intros* =
*add-pos-nonneg add-pos-pos add-nonneg-pos add-nonneg-nonneg*
*add-neg-nonpos add-neg-neg add-nonpos-neg add-nonpos-nonpos*

**end**

**class** *strict-ordered-comm-monoid-add* = *comm-monoid-add* + *strict-ordered-ab-semigroup-add*
**begin**

**lemma** *pos-add-strict*: $0 < a \Longrightarrow b < c \Longrightarrow b < a + c$
⟨*proof*⟩

**end**

**class** *ordered-cancel-comm-monoid-add* = *ordered-comm-monoid-add* + *cancel-ab-semigroup-add*
**begin**

**subclass** *ordered-cancel-ab-semigroup-add* ⟨*proof*⟩
**subclass** *strict-ordered-comm-monoid-add* ⟨*proof*⟩

**lemma** *add-strict-increasing*: $0 < a \Longrightarrow b \leq c \Longrightarrow b < a + c$
⟨*proof*⟩

**lemma** *add-strict-increasing2*: $0 \leq a \Longrightarrow b < c \Longrightarrow b < a + c$
⟨*proof*⟩

**end**

**class** *ordered-ab-semigroup-monoid-add-imp-le* = *monoid-add* + *ordered-ab-semigroup-add-imp-le*
**begin**

**lemma** *add-less-same-cancel1* [*simp*]: $b + a < b \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *add-less-same-cancel2* [*simp*]: $a + b < b \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *less-add-same-cancel1* [*simp*]: $a < a + b \longleftrightarrow 0 < b$
⟨*proof*⟩

**lemma** *less-add-same-cancel2* [*simp*]: $a < b + a \longleftrightarrow 0 < b$
⟨*proof*⟩

**lemma** *add-le-same-cancel1* [*simp*]: $b + a \leq b \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *add-le-same-cancel2* [*simp*]: $a + b \leq b \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *le-add-same-cancel1* [*simp*]: $a \leq a + b \longleftrightarrow 0 \leq b$
⟨*proof*⟩

**lemma** *le-add-same-cancel2* [*simp*]: $a \leq b + a \longleftrightarrow 0 \leq b$
⟨*proof*⟩

**subclass** *cancel-comm-monoid-add*
⟨*proof*⟩

**subclass** *ordered-cancel-comm-monoid-add*
⟨*proof*⟩

**end**

**class** *ordered-ab-group-add* = *ab-group-add* + *ordered-ab-semigroup-add*
**begin**

**subclass** *ordered-cancel-ab-semigroup-add* ⟨*proof*⟩

**subclass** *ordered-ab-semigroup-monoid-add-imp-le*
⟨*proof*⟩

**lemma** *max-diff-distrib-left*: $max\ x\ y - z = max\ (x - z)\ (y - z)$
⟨*proof*⟩

**lemma** *min-diff-distrib-left*: $min\ x\ y - z = min\ (x - z)\ (y - z)$
⟨*proof*⟩

**lemma** *le-imp-neg-le*:
**assumes** $a \leq b$
**shows** $- b \leq - a$
⟨*proof*⟩

**lemma** *neg-le-iff-le* [*simp*]: $- b \leq - a \longleftrightarrow a \leq b$
⟨*proof*⟩

**lemma** *neg-le-0-iff-le* [*simp*]: $- a \leq 0 \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *neg-0-le-iff-le* [*simp*]: $0 \leq - a \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *neg-less-iff-less* [*simp*]: $- b < - a \longleftrightarrow a < b$

⟨*proof*⟩

**lemma** *neg-less-0-iff-less* [*simp*]: $- a < 0 \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *neg-0-less-iff-less* [*simp*]: $0 < - a \longleftrightarrow a < 0$
⟨*proof*⟩

The next several equations can make the simplifier loop!

**lemma** *less-minus-iff*: $a < - b \longleftrightarrow b < - a$
⟨*proof*⟩

**lemma** *minus-less-iff*: $- a < b \longleftrightarrow - b < a$
⟨*proof*⟩

**lemma** *le-minus-iff*: $a \leq - b \longleftrightarrow b \leq - a$
⟨*proof*⟩

**lemma** *minus-le-iff*: $- a \leq b \longleftrightarrow - b \leq a$
⟨*proof*⟩

**lemma** *diff-less-0-iff-less* [*simp*]: $a - b < 0 \longleftrightarrow a < b$
⟨*proof*⟩

**lemmas** *less-iff-diff-less-0* = *diff-less-0-iff-less* [*symmetric*]

**lemma** *diff-less-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a - b < c \longleftrightarrow a < c + b$
⟨*proof*⟩

**lemma** *less-diff-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a < c - b \longleftrightarrow a + b < c$
⟨*proof*⟩

**lemma** *diff-gt-0-iff-gt* [*simp*]: $a - b > 0 \longleftrightarrow a > b$
⟨*proof*⟩

**lemma** *diff-le-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a - b \leq c \longleftrightarrow a \leq c + b$
⟨*proof*⟩

**lemma** *le-diff-eq* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*]:
$a \leq c - b \longleftrightarrow a + b \leq c$
⟨*proof*⟩

**lemma** *diff-le-0-iff-le* [*simp*]: $a - b \leq 0 \longleftrightarrow a \leq b$
⟨*proof*⟩

**lemmas** *le-iff-diff-le-0* = *diff-le-0-iff-le* [*symmetric*]

**lemma** *diff-ge-0-iff-ge* [*simp*]: $a - b \geq 0 \longleftrightarrow a \geq b$
⟨*proof*⟩

**lemma** *diff-eq-diff-less*: $a - b = c - d \Longrightarrow a < b \longleftrightarrow c < d$
⟨*proof*⟩

**lemma** *diff-eq-diff-less-eq*: $a - b = c - d \Longrightarrow a \leq b \longleftrightarrow c \leq d$
⟨*proof*⟩

**lemma** *diff-mono*: $a \leq b \Longrightarrow d \leq c \Longrightarrow a - c \leq b - d$
⟨*proof*⟩

**lemma** *diff-left-mono*: $b \leq a \Longrightarrow c - a \leq c - b$
⟨*proof*⟩

**lemma** *diff-right-mono*: $a \leq b \Longrightarrow a - c \leq b - c$
⟨*proof*⟩

**lemma** *diff-strict-mono*: $a < b \Longrightarrow d < c \Longrightarrow a - c < b - d$
⟨*proof*⟩

**lemma** *diff-strict-left-mono*: $b < a \Longrightarrow c - a < c - b$
⟨*proof*⟩

**lemma** *diff-strict-right-mono*: $a < b \Longrightarrow a - c < b - c$
⟨*proof*⟩

**end**

**locale** *group-cancel*
**begin**

**lemma** *add1*: $(A::'a::\mathit{comm\text{-}monoid\text{-}add}) \equiv k + a \Longrightarrow A + b \equiv k + (a + b)$
⟨*proof*⟩

**lemma** *add2*: $(B::'a::\mathit{comm\text{-}monoid\text{-}add}) \equiv k + b \Longrightarrow a + B \equiv k + (a + b)$
⟨*proof*⟩

**lemma** *sub1*: $(A::'a::\mathit{ab\text{-}group\text{-}add}) \equiv k + a \Longrightarrow A - b \equiv k + (a - b)$
⟨*proof*⟩

**lemma** *sub2*: $(B::'a::\mathit{ab\text{-}group\text{-}add}) \equiv k + b \Longrightarrow a - B \equiv - k + (a - b)$
⟨*proof*⟩

**lemma** *neg1*: $(A::'a::\mathit{ab\text{-}group\text{-}add}) \equiv k + a \Longrightarrow - A \equiv - k + - a$
⟨*proof*⟩

**lemma** *rule0*: $(a::'a::\mathit{comm\text{-}monoid\text{-}add}) \equiv a + 0$

⟨*proof*⟩

**end**

⟨*ML*⟩

**class** *linordered-ab-semigroup-add* =
*linorder* + *ordered-ab-semigroup-add*

**class** *linordered-cancel-ab-semigroup-add* =
*linorder* + *ordered-cancel-ab-semigroup-add*
**begin**

**subclass** *linordered-ab-semigroup-add* ⟨*proof*⟩

**subclass** *ordered-ab-semigroup-add-imp-le*
⟨*proof*⟩

**end**

**class** *linordered-ab-group-add* = *linorder* + *ordered-ab-group-add*
**begin**

**subclass** *linordered-cancel-ab-semigroup-add* ⟨*proof*⟩

**lemma** *equal-neg-zero* [*simp*]: $a = -\ a \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *neg-equal-zero* [*simp*]: $-\ a = a \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *neg-less-eq-nonneg* [*simp*]: $-\ a \leq a \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *neg-less-pos* [*simp*]: $-\ a < a \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *less-eq-neg-nonpos* [*simp*]: $a \leq -\ a \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *less-neg-neg* [*simp*]: $a < -\ a \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *double-zero* [*simp*]: $a + a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *double-zero-sym* [*simp*]: $0 = a + a \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *zero-less-double-add-iff-zero-less-single-add* [*simp*]: $0 < a + a \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *zero-le-double-add-iff-zero-le-single-add* [*simp*]: $0 \leq a + a \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *double-add-less-zero-iff-single-add-less-zero* [*simp*]: $a + a < 0 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *double-add-le-zero-iff-single-add-le-zero* [*simp*]: $a + a \leq 0 \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *minus-max-eq-min*: $-\ max\ x\ y = min\ (-\ x)\ (-\ y)$
⟨*proof*⟩

**lemma** *minus-min-eq-max*: $-\ min\ x\ y = max\ (-\ x)\ (-\ y)$
⟨*proof*⟩

**end**

**class** *abs* =
**fixes** *abs* :: $'a \Rightarrow 'a$ (‹(‹open-block notation=‹mixfix abs››|-|)›)

**bundle** *abs-syntax*
**begin**
**notation** *abs* (‹(‹open-block notation=‹mixfix abs››|-|)›)
**end**

**class** *sgn* =
**fixes** *sgn* :: $'a \Rightarrow 'a$

**class** *ordered-ab-group-add-abs* = *ordered-ab-group-add* + *abs* +
**assumes** *abs-ge-zero* [*simp*]: $|a| \geq 0$
**and** *abs-ge-self*: $a \leq |a|$
**and** *abs-leI*: $a \leq b \Longrightarrow -\ a \leq b \Longrightarrow |a| \leq b$
**and** *abs-minus-cancel* [*simp*]: $|-a| = |a|$
**and** *abs-triangle-ineq*: $|a + b| \leq |a| + |b|$
**begin**

**lemma** *abs-minus-le-zero*: $-\ |a| \leq 0$
⟨*proof*⟩

**lemma** *abs-of-nonneg* [*simp*]:
**assumes** *nonneg*: $0 \leq a$
**shows** $|a| = a$
⟨*proof*⟩

**lemma** *abs-idempotent* [*simp*]: $||a|| = |a|$
⟨*proof*⟩

**lemma** *abs-eq-0* [*simp*]: $|a| = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *abs-zero* [*simp*]: $|0| = 0$
⟨*proof*⟩

**lemma** *abs-0-eq* [*simp*]: $0 = |a| \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *abs-le-zero-iff* [*simp*]: $|a| \leq 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *abs-le-self-iff* [*simp*]: $|a| \leq a \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *zero-less-abs-iff* [*simp*]: $0 < |a| \longleftrightarrow a \neq 0$
⟨*proof*⟩

**lemma** *abs-not-less-zero* [*simp*]: $\neg\ |a| < 0$
⟨*proof*⟩

**lemma** *abs-ge-minus-self*: $-\ a \leq |a|$
⟨*proof*⟩

**lemma** *abs-minus-commute*: $|a - b| = |b - a|$
⟨*proof*⟩

**lemma** *abs-of-pos*: $0 < a \Longrightarrow |a| = a$
⟨*proof*⟩

**lemma** *abs-of-nonpos* [*simp*]:
**assumes** $a \leq 0$
**shows** $|a| = -\ a$
⟨*proof*⟩

**lemma** *abs-of-neg*: $a < 0 \Longrightarrow |a| = -\ a$
⟨*proof*⟩

**lemma** *abs-le-D1*: $|a| \leq b \Longrightarrow a \leq b$
⟨*proof*⟩

**lemma** *abs-le-D2*: $|a| \leq b \Longrightarrow -\ a \leq b$
⟨*proof*⟩

**lemma** *abs-le-iff*: $|a| \leq b \longleftrightarrow a \leq b \wedge -\ a \leq b$
⟨*proof*⟩

**lemma** *abs-triangle-ineq2*: $|a| - |b| \leq |a - b|$

⟨*proof*⟩

**lemma** *abs-triangle-ineq2-sym*: $|a| - |b| \leq |b - a|$
⟨*proof*⟩

**lemma** *abs-triangle-ineq3*: $||a| - |b|| \leq |a - b|$
⟨*proof*⟩

**lemma** *abs-triangle-ineq4*: $|a - b| \leq |a| + |b|$
⟨*proof*⟩

**lemma** *abs-diff-triangle-ineq*: $|a + b - (c + d)| \leq |a - c| + |b - d|$
⟨*proof*⟩

**lemma** *abs-add-abs* [*simp*]: $||a| + |b|| = |a| + |b|$
(**is** *?L* = *?R*)
⟨*proof*⟩

**end**

**lemma** *dense-eq0-I*:
**fixes** $x$::$'a$::{*dense-linorder*,*ordered-ab-group-add-abs*}
**assumes** $\bigwedge e.\ 0 < e \Longrightarrow |x| \leq e$
**shows** $x = 0$
⟨*proof*⟩

**hide-fact** (**open**) *ab-diff-conv-add-uminus add-0 mult-1 ab-left-minus*

**lemmas** *add-0* = *add-0-left*
**lemmas** *mult-1* = *mult-1-left*
**lemmas** *ab-left-minus* = *left-minus*
**lemmas** *diff-diff-eq* = *diff-diff-add*

## 5.8 Canonically ordered monoids

Canonically ordered monoids are never groups.

**class** *canonically-ordered-monoid-add* = *comm-monoid-add* + *order* +
**assumes** *le-iff-add*: $a \leq b \longleftrightarrow (\exists\, c.\ b = a + c)$
**begin**

**lemma** *zero-le*[*simp*]: $0 \leq x$
⟨*proof*⟩

**lemma** *le-zero-eq*[*simp*]: $n \leq 0 \longleftrightarrow n = 0$
⟨*proof*⟩

**lemma** *not-less-zero*[*simp*]: $\neg\ n < 0$
⟨*proof*⟩

**lemma** *zero-less-iff-neq-zero*: $0 < n \longleftrightarrow n \neq 0$
⟨*proof*⟩

This theorem is useful with *blast*

**lemma** *gr-zeroI*: $(n = 0 \Longrightarrow False) \Longrightarrow 0 < n$
⟨*proof*⟩

**lemma** *not-gr-zero*[*simp*]: $\neg\ 0 < n \longleftrightarrow n = 0$
⟨*proof*⟩

**subclass** *ordered-comm-monoid-add*
⟨*proof*⟩

**lemma** *gr-implies-not-zero*: $m < n \Longrightarrow n \neq 0$
⟨*proof*⟩

**lemma** *add-eq-0-iff-both-eq-0*[*simp*]: $x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *zero-eq-add-iff-both-eq-0*[*simp*]: $0 = x + y \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *less-eqE*:
**assumes** ‹$a \leq b$›
**obtains** $c$ **where** ‹$b = a + c$›
⟨*proof*⟩

**lemma** *lessE*:
**assumes** ‹$a < b$›
**obtains** $c$ **where** ‹$b = a + c$› **and** ‹$c \neq 0$›
⟨*proof*⟩

**lemmas** *zero-order* = *zero-le le-zero-eq not-less-zero zero-less-iff-neq-zero not-gr-zero*
— This should be attributed with [*iff*], but then *blast* fails in *Set*.

**end**

**class** *ordered-cancel-comm-monoid-diff* =
*canonically-ordered-monoid-add* + *comm-monoid-diff* + *ordered-ab-semigroup-add-imp-le*
**begin**

**context**
**fixes** $a\ b :: 'a$
**assumes** *le*: $a \leq b$
**begin**

**lemma** *add-diff-inverse*: $a + (b - a) = b$
⟨*proof*⟩

**lemma** *add-diff-assoc*: $c + (b - a) = c + b - a$
⟨*proof*⟩

**lemma** *add-diff-assoc2*: $b - a + c = b + c - a$
⟨*proof*⟩

**lemma** *diff-add-assoc*: $c + b - a = c + (b - a)$
⟨*proof*⟩

**lemma** *diff-add-assoc2*: $b + c - a = b - a + c$
⟨*proof*⟩

**lemma** *diff-diff-right*: $c - (b - a) = c + a - b$
⟨*proof*⟩

**lemma** *diff-add*: $b - a + a = b$
⟨*proof*⟩

**lemma** *le-add-diff*: $c \leq b + c - a$
⟨*proof*⟩

**lemma** *le-imp-diff-is-add*: $a \leq b \Longrightarrow b - a = c \longleftrightarrow b = c + a$
⟨*proof*⟩

**lemma** *le-diff-conv2*: $c \leq b - a \longleftrightarrow c + a \leq b$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**end**

**end**

## 5.9 Tools setup

**lemma** *add-mono-thms-linordered-semiring*:
**fixes** $i\ j\ k$ :: $'a$::*ordered-ab-semigroup-add*
**shows** $i \leq j \wedge k \leq l \Longrightarrow i + k \leq j + l$
**and** $i = j \wedge k \leq l \Longrightarrow i + k \leq j + l$
**and** $i \leq j \wedge k = l \Longrightarrow i + k \leq j + l$
**and** $i = j \wedge k = l \Longrightarrow i + k = j + l$
⟨*proof*⟩

**lemma** *add-mono-thms-linordered-field*:
**fixes** $i\ j\ k$ :: $'a$::*ordered-cancel-ab-semigroup-add*
**shows** $i < j \wedge k = l \Longrightarrow i + k < j + l$
**and** $i = j \wedge k < l \Longrightarrow i + k < j + l$
**and** $i < j \wedge k \leq l \Longrightarrow i + k < j + l$
**and** $i \leq j \wedge k < l \Longrightarrow i + k < j + l$
**and** $i < j \wedge k < l \Longrightarrow i + k < j + l$

⟨*proof*⟩

**code-identifier**
**code-module** *Groups* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 6 Abstract lattices

**theory** *Lattices*
**imports** *Groups*
**begin**

## 6.1 Abstract semilattice

These locales provide a basic structure for interpretation into bigger structures; extensions require careful thinking, otherwise undesired effects may occur due to interpretation.

**locale** *semilattice* = *abel-semigroup* +
**assumes** *idem* [*simp*]: $a * a = a$
**begin**

**lemma** *left-idem* [*simp*]: $a * (a * b) = a * b$
⟨*proof*⟩

**lemma** *right-idem* [*simp*]: $(a * b) * b = a * b$
⟨*proof*⟩

**end**

**locale** *semilattice-neutr* = *semilattice* + *comm-monoid*

**locale** *semilattice-order* = *semilattice* +
**fixes** *less-eq* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** ‹≤› *50*)
**and** *less* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** ‹<› *50*)
**assumes** *order-iff*: $a \leq b \longleftrightarrow a = a * b$
**and** *strict-order-iff*: $a < b \longleftrightarrow a = a * b \wedge a \neq b$
**begin**

**lemma** *orderI*: $a = a * b \Longrightarrow a \leq b$
⟨*proof*⟩

**lemma** *orderE*:
**assumes** $a \leq b$
**obtains** $a = a * b$
⟨*proof*⟩

**sublocale** *ordering less-eq less*

⟨*proof*⟩

**lemma** *cobounded1* [*simp*]: $a * b \leq a$
⟨*proof*⟩

**lemma** *cobounded2* [*simp*]: $a * b \leq b$
⟨*proof*⟩

**lemma** *boundedI*:
**assumes** $a \leq b$ **and** $a \leq c$
**shows** $a \leq b * c$
⟨*proof*⟩

**lemma** *boundedE*:
**assumes** $a \leq b * c$
**obtains** $a \leq b$ **and** $a \leq c$
⟨*proof*⟩

**lemma** *bounded-iff* [*simp*]: $a \leq b * c \longleftrightarrow a \leq b \wedge a \leq c$
⟨*proof*⟩

**lemma** *strict-boundedE*:
**assumes** $a < b * c$
**obtains** $a < b$ **and** $a < c$
⟨*proof*⟩

**lemma** *coboundedI1*: $a \leq c \Longrightarrow a * b \leq c$
⟨*proof*⟩

**lemma** *coboundedI2*: $b \leq c \Longrightarrow a * b \leq c$
⟨*proof*⟩

**lemma** *strict-coboundedI1*: $a < c \Longrightarrow a * b < c$
⟨*proof*⟩

**lemma** *strict-coboundedI2*: $b < c \Longrightarrow a * b < c$
⟨*proof*⟩

**lemma** *mono*: $a \leq c \Longrightarrow b \leq d \Longrightarrow a * b \leq c * d$
⟨*proof*⟩

**lemma** *absorb1*: $a \leq b \Longrightarrow a * b = a$
⟨*proof*⟩

**lemma** *absorb2*: $b \leq a \Longrightarrow a * b = b$
⟨*proof*⟩

**lemma** *absorb3*: $a < b \Longrightarrow a * b = a$
⟨*proof*⟩

**lemma** *absorb4*: $b < a \Longrightarrow a * b = b$
⟨*proof*⟩

**lemma** *absorb-iff1*: $a \leq b \longleftrightarrow a * b = a$
⟨*proof*⟩

**lemma** *absorb-iff2*: $b \leq a \longleftrightarrow a * b = b$
⟨*proof*⟩

**end**

**locale** *semilattice-neutr-order* = *semilattice-neutr* + *semilattice-order*
**begin**

**sublocale** *ordering-top less-eq less* **1**
⟨*proof*⟩

**lemma** *eq-neutr-iff* [*simp*]: ‹$a * b = \mathbf{1} \longleftrightarrow a = \mathbf{1} \wedge b = \mathbf{1}$›
⟨*proof*⟩

**lemma** *neutr-eq-iff* [*simp*]: ‹$\mathbf{1} = a * b \longleftrightarrow a = \mathbf{1} \wedge b = \mathbf{1}$›
⟨*proof*⟩

**end**

Interpretations for boolean operators

**interpretation** *conj*: *semilattice-neutr* ‹($\wedge$)› *True*
⟨*proof*⟩

**interpretation** *disj*: *semilattice-neutr* ‹($\vee$)› *False*
⟨*proof*⟩

**declare** *conj-assoc* [*ac-simps del*] *disj-assoc* [*ac-simps del*] — already simp by default

## 6.2 Syntactic infimum and supremum operations

**class** *inf* =
**fixes** *inf* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹$\sqcap$› *70*)

**class** *sup* =
**fixes** *sup* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹$\sqcup$› *65*)

## 6.3 Concrete lattices

**class** *semilattice-inf* = *order* + *inf* +
**assumes** *inf-le1* [*simp*]: $x \sqcap y \leq x$
**and** *inf-le2* [*simp*]: $x \sqcap y \leq y$

**and** *inf-greatest*: $x \le y \Longrightarrow x \le z \Longrightarrow x \le y \sqcap z$

**class** *semilattice-sup* = *order* + *sup* +
**assumes** *sup-ge1* [*simp*]: $x \le x \sqcup y$
**and** *sup-ge2* [*simp*]: $y \le x \sqcup y$
**and** *sup-least*: $y \le x \Longrightarrow z \le x \Longrightarrow y \sqcup z \le x$
**begin**

Dual lattice.

**lemma** *dual-semilattice*: *class.semilattice-inf sup greater-eq greater*
⟨*proof*⟩

**end**

**class** *lattice* = *semilattice-inf* + *semilattice-sup*

### 6.3.1 Intro and elim rules

**context** *semilattice-inf*
**begin**

**lemma** *le-infI1*: $a \le x \Longrightarrow a \sqcap b \le x$
⟨*proof*⟩

**lemma** *le-infI2*: $b \le x \Longrightarrow a \sqcap b \le x$
⟨*proof*⟩

**lemma** *le-infI*: $x \le a \Longrightarrow x \le b \Longrightarrow x \le a \sqcap b$
⟨*proof*⟩

**lemma** *le-infE*: $x \le a \sqcap b \Longrightarrow (x \le a \Longrightarrow x \le b \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *le-inf-iff*: $x \le y \sqcap z \longleftrightarrow x \le y \wedge x \le z$
⟨*proof*⟩

**lemma** *le-iff-inf*: $x \le y \longleftrightarrow x \sqcap y = x$
⟨*proof*⟩

**lemma** *inf-mono*: $a \le c \Longrightarrow b \le d \Longrightarrow a \sqcap b \le c \sqcap d$
⟨*proof*⟩

**end**

**context** *semilattice-sup*
**begin**

**lemma** *le-supI1*: $x \le a \Longrightarrow x \le a \sqcup b$
⟨*proof*⟩

**lemma** *le-supI2*: $x \leq b \Longrightarrow x \leq a \sqcup b$
⟨*proof*⟩

**lemma** *le-supI*: $a \leq x \Longrightarrow b \leq x \Longrightarrow a \sqcup b \leq x$
⟨*proof*⟩

**lemma** *le-supE*: $a \sqcup b \leq x \Longrightarrow (a \leq x \Longrightarrow b \leq x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *le-sup-iff*: $x \sqcup y \leq z \longleftrightarrow x \leq z \wedge y \leq z$
⟨*proof*⟩

**lemma** *le-iff-sup*: $x \leq y \longleftrightarrow x \sqcup y = y$
⟨*proof*⟩

**lemma** *sup-mono*: $a \leq c \Longrightarrow b \leq d \Longrightarrow a \sqcup b \leq c \sqcup d$
⟨*proof*⟩

**end**

### 6.3.2 Equational laws

**context** *semilattice-inf*
**begin**

**sublocale** *inf*: *semilattice inf*
⟨*proof*⟩

**sublocale** *inf*: *semilattice-order inf less-eq less*
⟨*proof*⟩

**lemma** *inf-assoc*: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
⟨*proof*⟩

**lemma** *inf-commute*: $(x \sqcap y) = (y \sqcap x)$
⟨*proof*⟩

**lemma** *inf-left-commute*: $x \sqcap (y \sqcap z) = y \sqcap (x \sqcap z)$
⟨*proof*⟩

**lemma** *inf-idem*: $x \sqcap x = x$
⟨*proof*⟩

**lemma** *inf-left-idem*: $x \sqcap (x \sqcap y) = x \sqcap y$
⟨*proof*⟩

**lemma** *inf-right-idem*: $(x \sqcap y) \sqcap y = x \sqcap y$
⟨*proof*⟩

**lemma** *inf-absorb1*: $x \leq y \Longrightarrow x \sqcap y = x$
⟨*proof*⟩

**lemma** *inf-absorb2*: $y \leq x \Longrightarrow x \sqcap y = y$
⟨*proof*⟩

**lemmas** *inf-aci* = *inf-commute inf-assoc inf-left-commute inf-left-idem*

**end**

**context** *semilattice-sup*
**begin**

**sublocale** *sup*: *semilattice sup*
⟨*proof*⟩

**sublocale** *sup*: *semilattice-order sup greater-eq greater*
⟨*proof*⟩

**lemma** *sup-assoc*: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
⟨*proof*⟩

**lemma** *sup-commute*: $(x \sqcup y) = (y \sqcup x)$
⟨*proof*⟩

**lemma** *sup-left-commute*: $x \sqcup (y \sqcup z) = y \sqcup (x \sqcup z)$
⟨*proof*⟩

**lemma** *sup-idem*: $x \sqcup x = x$
⟨*proof*⟩

**lemma** *sup-left-idem* [*simp*]: $x \sqcup (x \sqcup y) = x \sqcup y$
⟨*proof*⟩

**lemma** *sup-absorb1*: $y \leq x \Longrightarrow x \sqcup y = x$
⟨*proof*⟩

**lemma** *sup-absorb2*: $x \leq y \Longrightarrow x \sqcup y = y$
⟨*proof*⟩

**lemmas** *sup-aci* = *sup-commute sup-assoc sup-left-commute sup-left-idem*

**end**

**context** *lattice*
**begin**

**lemma** *dual-lattice*: *class.lattice sup* $(\geq)$ $(>)$ *inf*

⟨*proof*⟩

**lemma** *inf-sup-absorb* [*simp*]: $x \sqcap (x \sqcup y) = x$
⟨*proof*⟩

**lemma** *sup-inf-absorb* [*simp*]: $x \sqcup (x \sqcap y) = x$
⟨*proof*⟩

**lemmas** *inf-sup-aci* = *inf-aci sup-aci*

**lemmas** *inf-sup-ord* = *inf-le1 inf-le2 sup-ge1 sup-ge2*

Towards distributivity.

**lemma** *distrib-sup-le*: $x \sqcup (y \sqcap z) \leq (x \sqcup y) \sqcap (x \sqcup z)$
⟨*proof*⟩

**lemma** *distrib-inf-le*: $(x \sqcap y) \sqcup (x \sqcap z) \leq x \sqcap (y \sqcup z)$
⟨*proof*⟩

If you have one of them, you have them all.

**lemma** *distrib-imp1*:
**assumes** *distrib*: $\bigwedge x\ y\ z.\ x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
**shows** $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$
⟨*proof*⟩

**lemma** *distrib-imp2*:
**assumes** *distrib*: $\bigwedge x\ y\ z.\ x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$
**shows** $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
⟨*proof*⟩

**end**

### 6.3.3 Strict order

**context** *semilattice-inf*
**begin**

**lemma** *less-infI1*: $a < x \Longrightarrow a \sqcap b < x$
⟨*proof*⟩

**lemma** *less-infI2*: $b < x \Longrightarrow a \sqcap b < x$
⟨*proof*⟩

**end**

**context** *semilattice-sup*
**begin**

**lemma** *less-supI1*: $x < a \Longrightarrow x < a \sqcup b$

⟨*proof*⟩

**lemma** *less-supI2*: $x < b \Longrightarrow x < a \sqcup b$
⟨*proof*⟩

**end**

## 6.4 Distributive lattices

**class** *distrib-lattice* = *lattice* +
**assumes** *sup-inf-distrib1*: $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

**context** *distrib-lattice*
**begin**

**lemma** *sup-inf-distrib2*: $(y \sqcap z) \sqcup x = (y \sqcup x) \sqcap (z \sqcup x)$
⟨*proof*⟩

**lemma** *inf-sup-distrib1*: $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
⟨*proof*⟩

**lemma** *inf-sup-distrib2*: $(y \sqcup z) \sqcap x = (y \sqcap x) \sqcup (z \sqcap x)$
⟨*proof*⟩

**lemma** *dual-distrib-lattice*: *class.distrib-lattice sup* $(\geq)$ $(>)$ *inf*
⟨*proof*⟩

**lemmas** *sup-inf-distrib* = *sup-inf-distrib1 sup-inf-distrib2*

**lemmas** *inf-sup-distrib* = *inf-sup-distrib1 inf-sup-distrib2*

**lemmas** *distrib* = *sup-inf-distrib1 sup-inf-distrib2 inf-sup-distrib1 inf-sup-distrib2*

**end**

## 6.5 Bounded lattices

**class** *bounded-semilattice-inf-top* = *semilattice-inf* + *order-top*
**begin**

**sublocale** *inf-top*: *semilattice-neutr inf top*
+ *inf-top*: *semilattice-neutr-order inf top less-eq less*
⟨*proof*⟩

**lemma** *inf-top-left*: $\top \sqcap x = x$
⟨*proof*⟩

**lemma** *inf-top-right*: $x \sqcap \top = x$
⟨*proof*⟩

**lemma** *inf-eq-top-iff*: $x \sqcap y = \top \longleftrightarrow x = \top \wedge y = \top$
⟨*proof*⟩

**lemma** *top-eq-inf-iff*: $\top = x \sqcap y \longleftrightarrow x = \top \wedge y = \top$
⟨*proof*⟩

**end**

**class** *bounded-semilattice-sup-bot* = *semilattice-sup* + *order-bot*
**begin**

**sublocale** *sup-bot*: *semilattice-neutr sup bot*
+ *sup-bot*: *semilattice-neutr-order sup bot greater-eq greater*
⟨*proof*⟩

**lemma** *sup-bot-left*: $\bot \sqcup x = x$
⟨*proof*⟩

**lemma** *sup-bot-right*: $x \sqcup \bot = x$
⟨*proof*⟩

**lemma** *sup-eq-bot-iff*: $x \sqcup y = \bot \longleftrightarrow x = \bot \wedge y = \bot$
⟨*proof*⟩

**lemma** *bot-eq-sup-iff*: $\bot = x \sqcup y \longleftrightarrow x = \bot \wedge y = \bot$
⟨*proof*⟩

**end**

**class** *bounded-lattice-bot* = *lattice* + *order-bot*
**begin**

**subclass** *bounded-semilattice-sup-bot* ⟨*proof*⟩

**lemma** *inf-bot-left* [*simp*]: $\bot \sqcap x = \bot$
⟨*proof*⟩

**lemma** *inf-bot-right* [*simp*]: $x \sqcap \bot = \bot$
⟨*proof*⟩

**end**

**class** *bounded-lattice-top* = *lattice* + *order-top*
**begin**

**subclass** *bounded-semilattice-inf-top* ⟨*proof*⟩

**lemma** *sup-top-left* [*simp*]: $\top \sqcup x = \top$
⟨*proof*⟩

**lemma** *sup-top-right* [*simp*]: $x \sqcup \top = \top$
⟨*proof*⟩

**end**

**class** *bounded-lattice* = *lattice* + *order-bot* + *order-top*
**begin**

**subclass** *bounded-lattice-bot* ⟨*proof*⟩
**subclass** *bounded-lattice-top* ⟨*proof*⟩

**lemma** *dual-bounded-lattice*: *class.bounded-lattice sup greater-eq greater inf* $\top$ $\bot$
⟨*proof*⟩

**end**

## 6.6 *min/max* as special case of lattice

**context** *linorder*
**begin**

**sublocale** *min*: *semilattice-order min less-eq less*
+ *max*: *semilattice-order max greater-eq greater*
⟨*proof*⟩

**declare** *min.absorb1* [*simp*] *min.absorb2* [*simp*]
*min.absorb3* [*simp*] *min.absorb4* [*simp*]
*max.absorb1* [*simp*] *max.absorb2* [*simp*]
*max.absorb3* [*simp*] *max.absorb4* [*simp*]

**lemma** *min-le-iff-disj*: $min\ x\ y \leq z \longleftrightarrow x \leq z \lor y \leq z$
⟨*proof*⟩

**lemma** *le-max-iff-disj*: $z \leq max\ x\ y \longleftrightarrow z \leq x \lor z \leq y$
⟨*proof*⟩

**lemma** *min-less-iff-disj*: $min\ x\ y < z \longleftrightarrow x < z \lor y < z$
⟨*proof*⟩

**lemma** *less-max-iff-disj*: $z < max\ x\ y \longleftrightarrow z < x \lor z < y$
⟨*proof*⟩

**lemma** *min-less-iff-conj* [*simp*]: $z < min\ x\ y \longleftrightarrow z < x \land z < y$
⟨*proof*⟩

**lemma** *max-less-iff-conj* [*simp*]: $max\ x\ y < z \longleftrightarrow x < z \land y < z$
⟨*proof*⟩

**lemma** *min-max-distrib1*: $min\ (max\ b\ c)\ a = max\ (min\ b\ a)\ (min\ c\ a)$
⟨*proof*⟩

**lemma** *min-max-distrib2*: $min\ a\ (max\ b\ c) = max\ (min\ a\ b)\ (min\ a\ c)$
⟨*proof*⟩

**lemma** *max-min-distrib1*: $max\ (min\ b\ c)\ a = min\ (max\ b\ a)\ (max\ c\ a)$
⟨*proof*⟩

**lemma** *max-min-distrib2*: $max\ a\ (min\ b\ c) = min\ (max\ a\ b)\ (max\ a\ c)$
⟨*proof*⟩

**lemmas** *min-max-distribs* = *min-max-distrib1 min-max-distrib2 max-min-distrib1 max-min-distrib2*

**lemma** *split-min* [*no-atp*]: $P\ (min\ i\ j) \longleftrightarrow (i \le j \longrightarrow P\ i) \land (\neg\ i \le j \longrightarrow P\ j)$
⟨*proof*⟩

**lemma** *split-max* [*no-atp*]: $P\ (max\ i\ j) \longleftrightarrow (i \le j \longrightarrow P\ j) \land (\neg\ i \le j \longrightarrow P\ i)$
⟨*proof*⟩

**lemma** *split-min-lin* [*no-atp*]:
‹$P\ (min\ a\ b) \longleftrightarrow (b = a \longrightarrow P\ a) \land (a < b \longrightarrow P\ a) \land (b < a \longrightarrow P\ b)$›
⟨*proof*⟩

**lemma** *split-max-lin* [*no-atp*]:
‹$P\ (max\ a\ b) \longleftrightarrow (b = a \longrightarrow P\ a) \land (a < b \longrightarrow P\ b) \land (b < a \longrightarrow P\ a)$›
⟨*proof*⟩

**end**

**lemma** *inf-min*: $inf = (min :: 'a::\{semilattice\text{-}inf,linorder\} \Rightarrow 'a \Rightarrow 'a)$
⟨*proof*⟩

**lemma** *sup-max*: $sup = (max :: 'a::\{semilattice\text{-}sup,linorder\} \Rightarrow 'a \Rightarrow 'a)$
⟨*proof*⟩

## 6.7 Uniqueness of inf and sup

**lemma** (**in** *semilattice-inf*) *inf-unique*:
**fixes** $f$ (**infixl** ‹△› *70*)
**assumes** *le1*: $\bigwedge x\ y.\ x \vartriangle y \le x$
**and** *le2*: $\bigwedge x\ y.\ x \vartriangle y \le y$
**and** *greatest*: $\bigwedge x\ y\ z.\ x \le y \Longrightarrow x \le z \Longrightarrow x \le y \vartriangle z$
**shows** $x \sqcap y = x \vartriangle y$
⟨*proof*⟩

**lemma** (**in** *semilattice-sup*) *sup-unique*:
**fixes** $f$ (**infixl** ‹∇› *70*)

**assumes** *ge1* [*simp*]: $\bigwedge x\ y.\ x \le x \nabla y$
**and** *ge2*: $\bigwedge x\ y.\ y \le x \nabla y$
**and** *least*: $\bigwedge x\ y\ z.\ y \le x \Longrightarrow z \le x \Longrightarrow y \nabla z \le x$
**shows** $x \sqcup y = x \nabla y$
⟨*proof*⟩

## 6.8 Lattice on - ⇒ -

**instantiation** *fun* :: (*type*, *semilattice-sup*) *semilattice-sup*
**begin**

**definition** $f \sqcup g = (\lambda x.\ f\ x \sqcup g\ x)$

**lemma** *sup-apply* [*simp*, *code*]: $(f \sqcup g)\ x = f\ x \sqcup g\ x$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *semilattice-inf*) *semilattice-inf*
**begin**

**definition** $f \sqcap g = (\lambda x.\ f\ x \sqcap g\ x)$

**lemma** *inf-apply* [*simp*, *code*]: $(f \sqcap g)\ x = f\ x \sqcap g\ x$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instance** *fun* :: (*type*, *lattice*) *lattice* ⟨*proof*⟩

**instance** *fun* :: (*type*, *distrib-lattice*) *distrib-lattice*
⟨*proof*⟩

**instance** *fun* :: (*type*, *bounded-lattice*) *bounded-lattice* ⟨*proof*⟩

**instantiation** *fun* :: (*type*, *uminus*) *uminus*
**begin**

**definition** *fun-Compl-def*: $-\ A = (\lambda x.\ -\ A\ x)$

**lemma** *uminus-apply* [*simp*, *code*]: $(-\ A)\ x = -\ (A\ x)$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *minus*) *minus*
**begin**

**definition** *fun-diff-def*: $A - B = (\lambda x.\ A\ x - B\ x)$

**lemma** *minus-apply* [*simp*, *code*]: $(A - B)\ x = A\ x - B\ x$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**end**

# 7 Boolean Algebras

**theory** *Boolean-Algebras*
**imports** *Lattices*
**begin**

## 7.1 Abstract boolean algebra

**locale** *abstract-boolean-algebra* = *conj*: *abel-semigroup* ‹($\sqcap$)› + *disj*: *abel-semigroup* ‹($\sqcup$)›
**for** *conj* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹$\sqcap$› *70*)
**and** *disj* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹$\sqcup$› *65*) +
**fixes** *compl* :: ‹$'a \Rightarrow 'a$› (‹(‹*open-block notation*=‹*prefix* −››− -)› [*81*] *80*)
**and** *zero* :: ‹$'a$› (‹**0**›)
**and** *one* :: ‹$'a$› (‹**1**›)
**assumes** *conj-disj-distrib*: ‹$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$›
**and** *disj-conj-distrib*: ‹$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$›
**and** *conj-one-right*: ‹$x \sqcap \mathbf{1} = x$›
**and** *disj-zero-right*: ‹$x \sqcup \mathbf{0} = x$›
**and** *conj-cancel-right* [*simp*]: ‹$x \sqcap - x = \mathbf{0}$›
**and** *disj-cancel-right* [*simp*]: ‹$x \sqcup - x = \mathbf{1}$›
**begin**

**sublocale** *conj*: *semilattice-neutr* ‹($\sqcap$)› ‹**1**›
⟨*proof*⟩

**sublocale** *disj*: *semilattice-neutr* ‹($\sqcup$)› ‹**0**›
⟨*proof*⟩

### 7.1.1 Complement

**lemma** *complement-unique*:

**assumes** *1*: $a \sqcap x = \mathbf{0}$
**assumes** *2*: $a \sqcup x = \mathbf{1}$
**assumes** *3*: $a \sqcap y = \mathbf{0}$
**assumes** *4*: $a \sqcup y = \mathbf{1}$
**shows** $x = y$
⟨*proof*⟩

**lemma** *compl-unique*: $x \sqcap y = \mathbf{0} \Longrightarrow x \sqcup y = \mathbf{1} \Longrightarrow -\ x = y$
⟨*proof*⟩

**lemma** *double-compl* [*simp*]: $-\ (-\ x) = x$
⟨*proof*⟩

**lemma** *compl-eq-compl-iff* [*simp*]:
‹$-\ x = -\ y \longleftrightarrow x = y$› (**is** ‹$?P \longleftrightarrow ?Q$›)
⟨*proof*⟩

### 7.1.2 Conjunction

**lemma** *conj-zero-right* [*simp*]: $x \sqcap \mathbf{0} = \mathbf{0}$
⟨*proof*⟩

**lemma** *compl-one* [*simp*]: $-\ \mathbf{1} = \mathbf{0}$
⟨*proof*⟩

**lemma** *conj-zero-left* [*simp*]: $\mathbf{0} \sqcap x = \mathbf{0}$
⟨*proof*⟩

**lemma** *conj-cancel-left* [*simp*]: $-\ x \sqcap x = \mathbf{0}$
⟨*proof*⟩

**lemma** *conj-disj-distrib2*: $(y \sqcup z) \sqcap x = (y \sqcap x) \sqcup (z \sqcap x)$
⟨*proof*⟩

**lemmas** *conj-disj-distribs* = *conj-disj-distrib conj-disj-distrib2*

### 7.1.3 Disjunction

**context**
**begin**

**interpretation** *dual*: *abstract-boolean-algebra* ‹$(\sqcup)$› ‹$(\sqcap)$› *compl* ‹$\mathbf{1}$› ‹$\mathbf{0}$›
⟨*proof*⟩

**lemma** *disj-one-right* [*simp*]: $x \sqcup \mathbf{1} = \mathbf{1}$
⟨*proof*⟩

**lemma** *compl-zero* [*simp*]: $-\ \mathbf{0} = \mathbf{1}$
⟨*proof*⟩

**lemma** *disj-one-left* [*simp*]: $\mathbf{1} \sqcup x = \mathbf{1}$
⟨*proof*⟩

**lemma** *disj-cancel-left* [*simp*]: $- x \sqcup x = \mathbf{1}$
⟨*proof*⟩

**lemma** *disj-conj-distrib2*: $(y \sqcap z) \sqcup x = (y \sqcup x) \sqcap (z \sqcup x)$
⟨*proof*⟩

**lemmas** *disj-conj-distribs* = *disj-conj-distrib disj-conj-distrib2*

**end**

### 7.1.4 De Morgan's Laws

**lemma** *de-Morgan-conj* [*simp*]: $- (x \sqcap y) = - x \sqcup - y$
⟨*proof*⟩

**context**
**begin**

**interpretation** *dual*: *abstract-boolean-algebra* ‹($\sqcup$)› ‹($\sqcap$)› *compl* ‹$\mathbf{1}$› ‹$\mathbf{0}$›
⟨*proof*⟩

**lemma** *de-Morgan-disj* [*simp*]: $- (x \sqcup y) = - x \sqcap - y$
⟨*proof*⟩

**end**

**end**

## 7.2 Symmetric Difference

**locale** *abstract-boolean-algebra-sym-diff* = *abstract-boolean-algebra* +
**fixes** *xor* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹$\ominus$› *65*)
**assumes** *xor-def* : ‹$x \ominus y = (x \sqcap - y) \sqcup (- x \sqcap y)$›
**begin**

**sublocale** *xor*: *comm-monoid xor* ‹$\mathbf{0}$›
⟨*proof*⟩

**lemma** *xor-def2*:
‹$x \ominus y = (x \sqcup y) \sqcap (- x \sqcup - y)$›
⟨*proof*⟩

**lemma** *xor-one-right* [*simp*]: $x \ominus \mathbf{1} = - x$
⟨*proof*⟩

**lemma** *xor-one-left* [*simp*]: $\mathbf{1} \ominus x = - x$
⟨*proof*⟩

**lemma** *xor-self* [*simp*]: $x \ominus x = \mathbf{0}$
⟨*proof*⟩

**lemma** *xor-left-self* [*simp*]: $x \ominus (x \ominus y) = y$
⟨*proof*⟩

**lemma** *xor-compl-left* [*simp*]: $- x \ominus y = - (x \ominus y)$
⟨*proof*⟩

**lemma** *xor-compl-right* [*simp*]: $x \ominus - y = - (x \ominus y)$
⟨*proof*⟩

**lemma** *xor-cancel-right* [*simp*]: $x \ominus - x = \mathbf{1}$
⟨*proof*⟩

**lemma** *xor-cancel-left* [*simp*]: $- x \ominus x = \mathbf{1}$
⟨*proof*⟩

**lemma** *conj-xor-distrib*: $x \sqcap (y \ominus z) = (x \sqcap y) \ominus (x \sqcap z)$
⟨*proof*⟩

**lemma** *conj-xor-distrib2*: $(y \ominus z) \sqcap x = (y \sqcap x) \ominus (z \sqcap x)$
⟨*proof*⟩

**lemmas** *conj-xor-distribs* = *conj-xor-distrib conj-xor-distrib2*

**end**

## 7.3 Type classes

**class** *boolean-algebra* = *distrib-lattice* + *bounded-lattice* + *minus* + *uminus* +
**assumes** *inf-compl-bot*: ‹$x \sqcap - x = \bot$›
**and** *sup-compl-top*: ‹$x \sqcup - x = \top$›
**assumes** *diff-eq*: ‹$x - y = x \sqcap - y$›
**begin**

**sublocale** *boolean-algebra*: *abstract-boolean-algebra* ‹($\sqcap$)› ‹($\sqcup$)› *uminus* $\bot$ $\top$
⟨*proof*⟩

**lemma** *compl-inf-bot*: $- x \sqcap x = \bot$
⟨*proof*⟩

**lemma** *compl-sup-top*: $- x \sqcup x = \top$
⟨*proof*⟩

**lemma** *compl-unique*:
**assumes** $x \sqcap y = \bot$
**and** $x \sqcup y = \top$

**shows** $- x = y$
⟨*proof*⟩

**lemma** *double-compl*: $- (- x) = x$
⟨*proof*⟩

**lemma** *compl-eq-compl-iff*: $- x = - y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *compl-bot-eq*: $- \bot = \top$
⟨*proof*⟩

**lemma** *compl-top-eq*: $- \top = \bot$
⟨*proof*⟩

**lemma** *compl-inf*: $- (x \sqcap y) = - x \sqcup - y$
⟨*proof*⟩

**lemma** *compl-sup*: $- (x \sqcup y) = - x \sqcap - y$
⟨*proof*⟩

**lemma** *compl-mono*:
**assumes** $x \leq y$
**shows** $- y \leq - x$
⟨*proof*⟩

**lemma** *compl-le-compl-iff* [*simp*]: $- x \leq - y \longleftrightarrow y \leq x$
⟨*proof*⟩

**lemma** *compl-le-swap1*:
**assumes** $y \leq - x$
**shows** $x \leq -y$
⟨*proof*⟩

**lemma** *compl-le-swap2*:
**assumes** $- y \leq x$
**shows** $- x \leq y$
⟨*proof*⟩

**lemma** *compl-less-compl-iff* [*simp*]: $- x < - y \longleftrightarrow y < x$
⟨*proof*⟩

**lemma** *compl-less-swap1*:
**assumes** $y < - x$
**shows** $x < - y$
⟨*proof*⟩

**lemma** *compl-less-swap2*:
**assumes** $- y < x$

**shows** $- x < y$
⟨*proof*⟩

**lemma** *sup-cancel-left1*: ‹$x \sqcup a \sqcup (- x \sqcup b) = \top$›
⟨*proof*⟩

**lemma** *sup-cancel-left2*: ‹$- x \sqcup a \sqcup (x \sqcup b) = \top$›
⟨*proof*⟩

**lemma** *inf-cancel-left1*: ‹$x \sqcap a \sqcap (- x \sqcap b) = \bot$›
⟨*proof*⟩

**lemma** *inf-cancel-left2*: ‹$- x \sqcap a \sqcap (x \sqcap b) = \bot$›
⟨*proof*⟩

**lemma** *sup-compl-top-left1* [*simp*]: ‹$- x \sqcup (x \sqcup y) = \top$›
⟨*proof*⟩

**lemma** *sup-compl-top-left2* [*simp*]: ‹$x \sqcup (- x \sqcup y) = \top$›
⟨*proof*⟩

**lemma** *inf-compl-bot-left1* [*simp*]: ‹$- x \sqcap (x \sqcap y) = \bot$›
⟨*proof*⟩

**lemma** *inf-compl-bot-left2* [*simp*]: ‹$x \sqcap (- x \sqcap y) = \bot$›
⟨*proof*⟩

**lemma** *inf-compl-bot-right* [*simp*]: ‹$x \sqcap (y \sqcap - x) = \bot$›
⟨*proof*⟩

**end**

## 7.4 Lattice on *bool*

**instantiation** *bool* :: *boolean-algebra*
**begin**

**definition** *bool-Compl-def* [*simp*]: $uminus = Not$

**definition** *bool-diff-def* [*simp*]: $A - B \longleftrightarrow A \wedge \neg B$

**definition** [*simp*]: $P \sqcap Q \longleftrightarrow P \wedge Q$

**definition** [*simp*]: $P \sqcup Q \longleftrightarrow P \vee Q$

**instance** ⟨*proof*⟩

**end**

**lemma** *sup-boolI1*: $P \Longrightarrow P \sqcup Q$
⟨*proof*⟩

**lemma** *sup-boolI2*: $Q \Longrightarrow P \sqcup Q$
⟨*proof*⟩

**lemma** *sup-boolE*: $P \sqcup Q \Longrightarrow (P \Longrightarrow R) \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**instance** *fun* :: (*type*, *boolean-algebra*) *boolean-algebra*
⟨*proof*⟩

## 7.5 Lattice on unary and binary predicates

**lemma** *inf1I*: $A\ x \Longrightarrow B\ x \Longrightarrow (A \sqcap B)\ x$
⟨*proof*⟩

**lemma** *inf2I*: $A\ x\ y \Longrightarrow B\ x\ y \Longrightarrow (A \sqcap B)\ x\ y$
⟨*proof*⟩

**lemma** *inf1E*: $(A \sqcap B)\ x \Longrightarrow (A\ x \Longrightarrow B\ x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *inf2E*: $(A \sqcap B)\ x\ y \Longrightarrow (A\ x\ y \Longrightarrow B\ x\ y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *inf1D1*: $(A \sqcap B)\ x \Longrightarrow A\ x$
⟨*proof*⟩

**lemma** *inf2D1*: $(A \sqcap B)\ x\ y \Longrightarrow A\ x\ y$
⟨*proof*⟩

**lemma** *inf1D2*: $(A \sqcap B)\ x \Longrightarrow B\ x$
⟨*proof*⟩

**lemma** *inf2D2*: $(A \sqcap B)\ x\ y \Longrightarrow B\ x\ y$
⟨*proof*⟩

**lemma** *sup1I1*: $A\ x \Longrightarrow (A \sqcup B)\ x$
⟨*proof*⟩

**lemma** *sup2I1*: $A\ x\ y \Longrightarrow (A \sqcup B)\ x\ y$
⟨*proof*⟩

**lemma** *sup1I2*: $B\ x \Longrightarrow (A \sqcup B)\ x$
⟨*proof*⟩

**lemma** *sup2I2*: $B\ x\ y \Longrightarrow (A \sqcup B)\ x\ y$
⟨*proof*⟩

**lemma** *sup1E*: $(A \sqcup B)\ x \Longrightarrow (A\ x \Longrightarrow P) \Longrightarrow (B\ x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *sup2E*: $(A \sqcup B)\ x\ y \Longrightarrow (A\ x\ y \Longrightarrow P) \Longrightarrow (B\ x\ y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

Classical introduction rule: no commitment to $A$ vs $B$.

**lemma** *sup1CI*: $(\neg\ B\ x \Longrightarrow A\ x) \Longrightarrow (A \sqcup B)\ x$
⟨*proof*⟩

**lemma** *sup2CI*: $(\neg\ B\ x\ y \Longrightarrow A\ x\ y) \Longrightarrow (A \sqcup B)\ x\ y$
⟨*proof*⟩

## 7.6 Simproc setup

**locale** *boolean-algebra-cancel*
**begin**

**lemma** *sup1*: $(A{::}'a{::}\mathit{semilattice\text{-}sup}) \equiv \mathit{sup}\ k\ a \Longrightarrow \mathit{sup}\ A\ b \equiv \mathit{sup}\ k\ (\mathit{sup}\ a\ b)$
⟨*proof*⟩

**lemma** *sup2*: $(B{::}'a{::}\mathit{semilattice\text{-}sup}) \equiv \mathit{sup}\ k\ b \Longrightarrow \mathit{sup}\ a\ B \equiv \mathit{sup}\ k\ (\mathit{sup}\ a\ b)$
⟨*proof*⟩

**lemma** *sup0*: $(a{::}'a{::}\mathit{bounded\text{-}semilattice\text{-}sup\text{-}bot}) \equiv \mathit{sup}\ a\ \mathit{bot}$
⟨*proof*⟩

**lemma** *inf1*: $(A{::}'a{::}\mathit{semilattice\text{-}inf}) \equiv \mathit{inf}\ k\ a \Longrightarrow \mathit{inf}\ A\ b \equiv \mathit{inf}\ k\ (\mathit{inf}\ a\ b)$
⟨*proof*⟩

**lemma** *inf2*: $(B{::}'a{::}\mathit{semilattice\text{-}inf}) \equiv \mathit{inf}\ k\ b \Longrightarrow \mathit{inf}\ a\ B \equiv \mathit{inf}\ k\ (\mathit{inf}\ a\ b)$
⟨*proof*⟩

**lemma** *inf0*: $(a{::}'a{::}\mathit{bounded\text{-}semilattice\text{-}inf\text{-}top}) \equiv \mathit{inf}\ a\ \mathit{top}$
⟨*proof*⟩

**end**

⟨*ML*⟩

**context** *boolean-algebra*
**begin**

**lemma** *shunt1*: $(x \sqcap y \le z) \longleftrightarrow (x \le -y \sqcup z)$
⟨*proof*⟩

**lemma** *shunt2*: $(x \sqcap -y \le z) \longleftrightarrow (x \le y \sqcup z)$

⟨*proof*⟩

**lemma** *inf-shunt*: $(x \sqcap y = \bot) \longleftrightarrow (x \leq - y)$
⟨*proof*⟩

**lemma** *sup-shunt*: $(x \sqcup y = \top) \longleftrightarrow (- x \leq y)$
⟨*proof*⟩

**lemma** *diff-shunt-var*[*simp*]: $(x - y = \bot) \longleftrightarrow (x \leq y)$
⟨*proof*⟩

**lemma** *diff-shunt*[*simp*]: $(\bot = x - y) \longleftrightarrow (x \leq y)$
⟨*proof*⟩

**lemma** *sup-neg-inf*:
‹$p \leq q \sqcup r \longleftrightarrow p \sqcap -q \leq r$› (**is** ‹$?P \longleftrightarrow ?Q$›)
⟨*proof*⟩

**end**

**end**

# 8 Set theory for higher-order logic

**theory** *Set*
**imports** *Lattices Boolean-Algebras*
**begin**

## 8.1 Sets as predicates

**typedecl** *'a set*

**axiomatization** *Collect* :: $('a \Rightarrow bool) \Rightarrow 'a\ set$ — comprehension
**and** *member* :: $'a \Rightarrow 'a\ set \Rightarrow bool$ — membership
**where** *mem-Collect-eq* [*iff*, *code-unfold*]: *member a* (*Collect P*) = *P a*
**and** *Collect-mem-eq* [*simp*, *code-unfold*]: *Collect* ($\lambda x.$ *member x A*) = *A*

**notation**
*member* (‹'(∈')›) **and**
*member* (‹(‹*notation*=‹*infix* ∈››-/ ∈ -)› [*51*, *51*] *50*)

**abbreviation** *not-member*
**where** *not-member x A* ≡ ¬ ($x \in A$) — non-membership
**notation**
*not-member* (‹'(∉')›) **and**
*not-member* (‹(‹*notation*=‹*infix* ∉››-/ ∉ -)› [*51*, *51*] *50*)

**open-bundle** *member-ASCII-syntax*
**begin**

**notation** (*ASCII*)
*member* (‹'(:')›) **and**
*member* (‹(‹*notation*=‹*infix* :››-/ : -)› [*51*, *51*] *50*) **and**
*not-member* (‹'(~:')›) **and**
*not-member* (‹(‹*notation*=‹*infix* ~:››-/ ~: -)› [*51*, *51*] *50*)
**end**

Set comprehensions

**syntax**
*-Coll* :: *pttrn* ⇒ *bool* ⇒ *'a set* (‹(‹*indent*=*1* *notation*=‹*mixfix set comprehension*››{-./ -})›)
**syntax-consts**
*-Coll* ⇌ *Collect*
**translations**
$\{x.\ P\}$ ⇌ *CONST Collect* $(\lambda x.\ P)$

**syntax** (*ASCII*)
*-Collect* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *'a set* (‹(‹*indent*=*1* *notation*=‹*mixfix set comprehension*››{(-/: -)./ -})›)
**syntax**
*-Collect* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *'a set* (‹(‹*indent*=*1* *notation*=‹*mixfix set comprehension*››{(-/ ∈ -)./ -})›)
**translations**
$\{p{:}A.\ P\}$ ⇀ *CONST Collect* $(\lambda p.\ p \in A \wedge P)$

⟨*ML*⟩

**lemma** *CollectI*: $P\ a \Longrightarrow a \in \{x.\ P\ x\}$
⟨*proof*⟩

**lemma** *CollectD*: $a \in \{x.\ P\ x\} \Longrightarrow P\ a$
⟨*proof*⟩

**lemma** *Collect-cong*: $(\bigwedge x.\ P\ x = Q\ x) \Longrightarrow \{x.\ P\ x\} = \{x.\ Q\ x\}$
⟨*proof*⟩

Simproc for pulling $x = t$ in $\{x.\ \ldots \wedge x = t \wedge \ldots\}$ to the front (and similarly for $t = x$):

⟨*ML*⟩

**lemmas** *CollectE* = *CollectD* [*elim-format*]

**lemma** *set-eqI*:
**assumes** $\bigwedge x.\ x \in A \longleftrightarrow x \in B$
**shows** $A = B$
⟨*proof*⟩

**lemma** *set-eq-iff*: $A = B \longleftrightarrow (\forall x.\ x \in A \longleftrightarrow x \in B)$
⟨*proof*⟩

**lemma** *Collect-eqI*:
  **assumes** $\bigwedge x.\ P\ x = Q\ x$
  **shows** *Collect* $P =$ *Collect* $Q$
  ⟨*proof*⟩

Lifting of predicate class instances

**instantiation** *set* :: (*type*) *boolean-algebra*
**begin**

**definition** *less-eq-set*
  **where** $A \leq B \longleftrightarrow (\lambda x.\ member\ x\ A) \leq (\lambda x.\ member\ x\ B)$

**definition** *less-set*
  **where** $A < B \longleftrightarrow (\lambda x.\ member\ x\ A) < (\lambda x.\ member\ x\ B)$

**definition** *inf-set*
  **where** $A \sqcap B = Collect\ ((\lambda x.\ member\ x\ A) \sqcap (\lambda x.\ member\ x\ B))$

**definition** *sup-set*
  **where** $A \sqcup B = Collect\ ((\lambda x.\ member\ x\ A) \sqcup (\lambda x.\ member\ x\ B))$

**definition** *bot-set*
  **where** $\bot = Collect\ \bot$

**definition** *top-set*
  **where** $\top = Collect\ \top$

**definition** *uminus-set*
  **where** $-\ A = Collect\ (-\ (\lambda x.\ member\ x\ A))$

**definition** *minus-set*
  **where** $A - B = Collect\ ((\lambda x.\ member\ x\ A) - (\lambda x.\ member\ x\ B))$

**instance**
  ⟨*proof*⟩

**end**

Set enumerations

**abbreviation** *empty* :: *'a set* (‹{}›)
  **where** $\{\} \equiv bot$

**definition** *insert* :: $'a \Rightarrow 'a\ set \Rightarrow 'a\ set$
  **where** *insert-compr*: $insert\ a\ B = \{x.\ x = a \lor x \in B\}$

**open-bundle** *set-enumeration-syntax*
**begin**

**syntax**
*-Finset* :: *args* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set enumeration*››{-})›)
**syntax-consts**
*-Finset* ⇌ *insert*
**translations**
$\{x, xs\}$ ⇌ *CONST insert* $x$ $\{xs\}$
$\{x\}$ ⇌ *CONST insert* $x$ $\{\}$

**end**

## 8.2 Subsets and bounded quantifiers

**abbreviation** *subset* :: *'a set* ⇒ *'a set* ⇒ *bool*
**where** *subset* ≡ *less*

**abbreviation** *subset-eq* :: *'a set* ⇒ *'a set* ⇒ *bool*
**where** *subset-eq* ≡ *less-eq*

**notation**
*subset* (‹'(⊂')›) **and**
*subset* (‹(‹*notation=*‹*infix* ⊂››-/ ⊂ -)› [*51*, *51*] *50*) **and**
*subset-eq* (‹'(⊆')›) **and**
*subset-eq* (‹(‹*notation=*‹*infix* ⊆››-/ ⊆ -)› [*51*, *51*] *50*)

**abbreviation** (*input*)
*supset* :: *'a set* ⇒ *'a set* ⇒ *bool* **where**
*supset* ≡ *greater*

**abbreviation** (*input*)
*supset-eq* :: *'a set* ⇒ *'a set* ⇒ *bool* **where**
*supset-eq* ≡ *greater-eq*

**notation**
*supset* (‹'(⊃')›) **and**
*supset* (‹(‹*notation=*‹*infix* ⊃››-/ ⊃ -)› [*51*, *51*] *50*) **and**
*supset-eq* (‹'(⊇')›) **and**
*supset-eq* (‹(‹*notation=*‹*infix* ⊇››-/ ⊇ -)› [*51*, *51*] *50*)

**notation** (*ASCII* **output**)
*subset* (‹'(<')›) **and**
*subset* (‹(‹*notation=*‹*infix* <››-/ < -)› [*51*, *51*] *50*) **and**
*subset-eq* (‹'(<=')›) **and**
*subset-eq* (‹(‹*notation=*‹*infix* <=››-/ <= -)› [*51*, *51*] *50*)

**definition** *Ball* :: *'a set* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*
**where** *Ball A P* ⟷ $(\forall x.\ x \in A \longrightarrow P\ x)$ — bounded universal quantifiers

**definition** *Bex* :: *'a set* ⇒ (*'a* ⇒ *bool*) ⇒ *bool*
**where** *Bex A P* ⟷ $(\exists x.\ x \in A \land P\ x)$ — bounded existential quantifiers

**syntax** (*ASCII*)
*-Ball* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ALL :››ALL (-/:-)./ -)› [0, 0, 10] 10)
*-Bex* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder EX :››EX (-/:-)./ -)› [0, 0, 10] 10)
*-Bex1* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder EX! :››EX! (-/:-)./ -)› [0, 0, 10] 10)
*-Bleast* :: *id* ⇒ *'a set* ⇒ *bool* ⇒ *'a* (‹(‹indent=3 notation=‹binder LEAST :››LEAST (-/:-)./ -)› [0, 0, 10] 10)

**syntax** (*input*)
*-Ball* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ! :››! (-/:-)./ -)› [0, 0, 10] 10)
*-Bex* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ? :››? (-/:-)./ -)› [0, 0, 10] 10)
*-Bex1* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ?! :››?! (-/:-)./ -)› [0, 0, 10] 10)

**syntax**
*-Ball* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ∀››∀(-/∈-)./ -)› [0, 0, 10] 10)
*-Bex* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ∃››∃(-/∈-)./ -)› [0, 0, 10] 10)
*-Bex1* :: *pttrn* ⇒ *'a set* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder ∃!››∃!(-/∈-)./ -)› [0, 0, 10] 10)
*-Bleast* :: *id* ⇒ *'a set* ⇒ *bool* ⇒ *'a* (‹(‹indent=3 notation=‹binder LEAST››LEAST(-/∈-)./ -)› [0, 0, 10] 10)

**syntax-consts**
*-Ball* ⇌ *Ball* **and**
*-Bex* ⇌ *Bex* **and**
*-Bex1* ⇌ *Ex1* **and**
*-Bleast* ⇌ *Least*

**translations**
∀*x*∈*A*. *P* ⇌ *CONST Ball A* (λ*x*. *P*)
∃*x*∈*A*. *P* ⇌ *CONST Bex A* (λ*x*. *P*)
∃!*x*∈*A*. *P* ⇀ ∃!*x*. *x* ∈ *A* ∧ *P*
*LEAST x*:*A*. *P* ⇀ *LEAST x*. *x* ∈ *A* ∧ *P*

**syntax** (*ASCII* **output**)
*-setlessAll* :: [*idt*, *'a*, *bool*] ⇒ *bool* (‹(‹indent=3 notation=‹binder ALL››ALL -<-./ -)› [0, 0, 10] 10)
*-setlessEx* :: [*idt*, *'a*, *bool*] ⇒ *bool* (‹(‹indent=3 notation=‹binder EX››EX -<-./ -)› [0, 0, 10] 10)
*-setleAll* :: [*idt*, *'a*, *bool*] ⇒ *bool* (‹(‹indent=3 notation=‹binder ALL››ALL -<=-./ -)› [0, 0, 10] 10)
*-setleEx* :: [*idt*, *'a*, *bool*] ⇒ *bool* (‹(‹indent=3 notation=‹binder EX››EX -<=-./

-)› [0, 0, 10] 10)
  -setleEx1 :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder EX!››EX! -<=-./ -)› [0, 0, 10] 10)

**syntax**
  -setlessAll :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder ∀››∀ -⊂-./ -)› [0, 0, 10] 10)
  -setlessEx :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder ∃››∃ -⊂-./ -)› [0, 0, 10] 10)
  -setleAll :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder ∀››∀ -⊆-./ -)› [0, 0, 10] 10)
  -setleEx :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder ∃››∃ -⊆-./ -)› [0, 0, 10] 10)
  -setleEx1 :: [idt, 'a, bool] ⇒ bool (‹(‹indent=3 notation=‹binder ∃!››∃!-⊆-./ -)› [0, 0, 10] 10)

**syntax-consts**
  -setlessAll -setleAll ⇌ All **and**
  -setlessEx -setleEx ⇌ Ex **and**
  -setleEx1 ⇌ Ex1

**translations**
  $\forall A{\subset}B.\ P \rightharpoonup \forall A.\ A \subset B \longrightarrow P$
  $\exists A{\subset}B.\ P \rightharpoonup \exists A.\ A \subset B \wedge P$
  $\forall A{\subseteq}B.\ P \rightharpoonup \forall A.\ A \subseteq B \longrightarrow P$
  $\exists A{\subseteq}B.\ P \rightharpoonup \exists A.\ A \subseteq B \wedge P$
  $\exists!A{\subseteq}B.\ P \rightharpoonup \exists!A.\ A \subseteq B \wedge P$

⟨ML⟩

Translate between $\{e \mid x1\ldots xn.\ P\}$ and $\{u.\ \exists\, x1\ldots xn.\ u = e \wedge P\}$; $\{y.\ \exists\, x1\ldots xn.\ y = e \wedge P\}$ is only translated if $[0..n] \subseteq bvs\ e$.

**syntax**
  -Setcompr :: 'a ⇒ idts ⇒ bool ⇒ 'a set
    (‹(‹indent=1 notation=‹mixfix set comprehension››{- |/-./ -})›)
**syntax-consts**
  -Setcompr ⇌ Collect

⟨ML⟩

**lemma** ballI [intro!]: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \Longrightarrow \forall x{\in}A.\ P\ x$
  ⟨proof⟩

**lemmas** strip = impI allI ballI

**lemma** bspec [dest?]: $\forall x{\in}A.\ P\ x \Longrightarrow x \in A \Longrightarrow P\ x$
  ⟨proof⟩

Gives better instantiation for bound:

⟨*ML*⟩

**lemma** *ballE* [*elim*]: $\forall x{\in}A.\ P\ x \Longrightarrow (P\ x \Longrightarrow Q) \Longrightarrow (x \notin A \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *bexI* [*intro*]: $P\ x \Longrightarrow x \in A \Longrightarrow \exists x{\in}A.\ P\ x$
— Normally the best argument order: $P\ x$ constrains the choice of $x \in A$.
⟨*proof*⟩

**lemma** *rev-bexI* [*intro?*]: $x \in A \Longrightarrow P\ x \Longrightarrow \exists x{\in}A.\ P\ x$
— The best argument order when there is only one $x \in A$.
⟨*proof*⟩

**lemma** *bexCI*: $(\forall x{\in}A.\ \neg\ P\ x \Longrightarrow P\ a) \Longrightarrow a \in A \Longrightarrow \exists x{\in}A.\ P\ x$
⟨*proof*⟩

**lemma** *bexE* [*elim!*]: $\exists x{\in}A.\ P\ x \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow P\ x \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *ball-triv* [*simp*]: $(\forall x{\in}A.\ P) \longleftrightarrow ((\exists x.\ x \in A) \longrightarrow P)$
— trivial rewrite rule.
⟨*proof*⟩

**lemma** *bex-triv* [*simp*]: $(\exists x{\in}A.\ P) \longleftrightarrow ((\exists x.\ x \in A) \wedge P)$
— Dual form for existentials.
⟨*proof*⟩

**lemma** *bex-triv-one-point1* [*simp*]: $(\exists x{\in}A.\ x = a) \longleftrightarrow a \in A$
⟨*proof*⟩

**lemma** *bex-triv-one-point2* [*simp*]: $(\exists x{\in}A.\ a = x) \longleftrightarrow a \in A$
⟨*proof*⟩

**lemma** *bex-one-point1* [*simp*]: $(\exists x{\in}A.\ x = a \wedge P\ x) \longleftrightarrow a \in A \wedge P\ a$
⟨*proof*⟩

**lemma** *bex-one-point2* [*simp*]: $(\exists x{\in}A.\ a = x \wedge P\ x) \longleftrightarrow a \in A \wedge P\ a$
⟨*proof*⟩

**lemma** *ball-one-point1* [*simp*]: $(\forall x{\in}A.\ x = a \longrightarrow P\ x) \longleftrightarrow (a \in A \longrightarrow P\ a)$
⟨*proof*⟩

**lemma** *ball-one-point2* [*simp*]: $(\forall x{\in}A.\ a = x \longrightarrow P\ x) \longleftrightarrow (a \in A \longrightarrow P\ a)$
⟨*proof*⟩

**lemma** *ball-conj-distrib*: $(\forall x{\in}A.\ P\ x \wedge Q\ x) \longleftrightarrow (\forall x{\in}A.\ P\ x) \wedge (\forall x{\in}A.\ Q\ x)$
⟨*proof*⟩

**lemma** *bex-disj-distrib*: $(\exists x{\in}A.\ P\ x \vee Q\ x) \longleftrightarrow (\exists x{\in}A.\ P\ x) \vee (\exists x{\in}A.\ Q\ x)$

$\langle proof \rangle$

Congruence rules

**lemma** *ball-cong*:
$[\![ A = B; \bigwedge x. x \in B \Longrightarrow P\ x \longleftrightarrow Q\ x ]\!] \Longrightarrow$
$(\forall x\in A. P\ x) \longleftrightarrow (\forall x\in B. Q\ x)$
$\langle proof \rangle$

**lemma** *ball-cong-simp* [*cong*]:
$[\![ A = B; \bigwedge x. x \in B =simp=> P\ x \longleftrightarrow Q\ x ]\!] \Longrightarrow$
$(\forall x\in A. P\ x) \longleftrightarrow (\forall x\in B. Q\ x)$
$\langle proof \rangle$

**lemma** *bex-cong*:
$[\![ A = B; \bigwedge x. x \in B \Longrightarrow P\ x \longleftrightarrow Q\ x ]\!] \Longrightarrow$
$(\exists x\in A. P\ x) \longleftrightarrow (\exists x\in B. Q\ x)$
$\langle proof \rangle$

**lemma** *bex-cong-simp* [*cong*]:
$[\![ A = B; \bigwedge x. x \in B =simp=> P\ x \longleftrightarrow Q\ x ]\!] \Longrightarrow$
$(\exists x\in A. P\ x) \longleftrightarrow (\exists x\in B. Q\ x)$
$\langle proof \rangle$

**lemma** *bex1-def*: $(\exists!x\in X. P\ x) \longleftrightarrow (\exists x\in X. P\ x) \wedge (\forall x\in X. \forall y\in X. P\ x \longrightarrow P\ y \longrightarrow x = y)$
$\langle proof \rangle$

## 8.3 Basic operations

### 8.3.1 Subsets

**lemma** *subsetI* [*intro!*]: $(\bigwedge x. x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
$\langle proof \rangle$

Map the type $'a\ set \Rightarrow anything$ to just $'a$; for overloading constants whose first argument has type $'a\ set$.

**lemma** *subsetD* [*elim*, *intro?*]: $A \subseteq B \Longrightarrow c \in A \Longrightarrow c \in B$
$\langle proof \rangle$

**lemma** *rev-subsetD* [*intro?*,*no-atp*]: $c \in A \Longrightarrow A \subseteq B \Longrightarrow c \in B$
— The same, with reversed premises for use with *erule* – cf. $[\![?P; ?P \longrightarrow ?Q]\!] \Longrightarrow ?Q$.
$\langle proof \rangle$

**lemma** *subsetCE* [*elim*,*no-atp*]: $A \subseteq B \Longrightarrow (c \notin A \Longrightarrow P) \Longrightarrow (c \in B \Longrightarrow P) \Longrightarrow P$
— Classical elimination rule.
$\langle proof \rangle$

**lemma** *subset-eq*: $A \subseteq B \longleftrightarrow (\forall x{\in}A.\ x \in B)$
⟨*proof*⟩

**lemma** *contra-subsetD* [*no-atp*]: $A \subseteq B \Longrightarrow c \notin B \Longrightarrow c \notin A$
⟨*proof*⟩

**lemma** *subset-refl*: $A \subseteq A$
⟨*proof*⟩

**lemma** *subset-trans*: $A \subseteq B \Longrightarrow B \subseteq C \Longrightarrow A \subseteq C$
⟨*proof*⟩

**lemma** *subset-not-subset-eq* [*code*]: $A \subset B \longleftrightarrow A \subseteq B \wedge \neg B \subseteq A$
⟨*proof*⟩

**lemma** *eq-mem-trans*: $a = b \Longrightarrow b \in A \Longrightarrow a \in A$
⟨*proof*⟩

**lemmas** *basic-trans-rules* [*trans*] =
*order-trans-rules rev-subsetD subsetD eq-mem-trans*

### 8.3.2 Equality

**lemma** *subset-antisym* [*intro*!]: $A \subseteq B \Longrightarrow B \subseteq A \Longrightarrow A = B$
— Anti-symmetry of the subset relation.
⟨*proof*⟩

Equality rules from ZF set theory – are they appropriate here?

**lemma** *equalityD1*: $A = B \Longrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *equalityD2*: $A = B \Longrightarrow B \subseteq A$
⟨*proof*⟩

Be careful when adding this to the claset as *subset-empty* is in the simpset: $A = \{\}$ goes to $\{\} \subseteq A$ and $A \subseteq \{\}$ and then back to $A = \{\}$!

**lemma** *equalityE*: $A = B \Longrightarrow (A \subseteq B \Longrightarrow B \subseteq A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *equalityCE* [*elim*]: $A = B \Longrightarrow (c \in A \Longrightarrow c \in B \Longrightarrow P) \Longrightarrow (c \notin A \Longrightarrow c \notin B \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *eqset-imp-iff*: $A = B \Longrightarrow x \in A \longleftrightarrow x \in B$
⟨*proof*⟩

**lemma** *eqelem-imp-iff*: $x = y \Longrightarrow x \in A \longleftrightarrow y \in A$
⟨*proof*⟩

### 8.3.3 The empty set

**lemma** *empty-def*: $\{\} = \{x.\ False\}$
  ⟨*proof*⟩

**lemma** *empty-iff* [*simp*]: $c \in \{\} \longleftrightarrow False$
  ⟨*proof*⟩

**lemma** *emptyE* [*elim*!]: $a \in \{\} \Longrightarrow P$
  ⟨*proof*⟩

**lemma** *empty-subsetI* [*iff*]: $\{\} \subseteq A$
  — One effect is to delete the ASSUMPTION $\{\} \subseteq A$
  ⟨*proof*⟩

**lemma** *equals0I*: $(\bigwedge y.\ y \in A \Longrightarrow False) \Longrightarrow A = \{\}$
  ⟨*proof*⟩

**lemma** *equals0D*: $A = \{\} \Longrightarrow a \notin A$
  — Use for reasoning about disjointness: $A \cap B = \{\}$
  ⟨*proof*⟩

**lemma** *ball-empty* [*simp*]: $Ball\ \{\}\ P \longleftrightarrow True$
  ⟨*proof*⟩

**lemma** *bex-empty* [*simp*]: $Bex\ \{\}\ P \longleftrightarrow False$
  ⟨*proof*⟩

### 8.3.4 The universal set – UNIV

**abbreviation** *UNIV* :: *'a set*
  **where** $UNIV \equiv top$

**lemma** *UNIV-def*: $UNIV = \{x.\ True\}$
  ⟨*proof*⟩

**lemma** *UNIV-I* [*simp*]: $x \in UNIV$
  ⟨*proof*⟩

**declare** *UNIV-I* [*intro*] — unsafe makes it less likely to cause problems

**lemma** *UNIV-witness* [*intro?*]: $\exists x.\ x \in UNIV$
  ⟨*proof*⟩

**lemma** *subset-UNIV*: $A \subseteq UNIV$
  ⟨*proof*⟩

Eta-contracting these two rules (to remove $P$) causes them to be ignored because of their interaction with congruence rules.

**lemma** *ball-UNIV* [*simp*]: $Ball\ UNIV\ P \longleftrightarrow All\ P$
⟨*proof*⟩

**lemma** *bex-UNIV* [*simp*]: $Bex\ UNIV\ P \longleftrightarrow Ex\ P$
⟨*proof*⟩

**lemma** *UNIV-eq-I*: $(\bigwedge x.\ x \in A) \Longrightarrow UNIV = A$
⟨*proof*⟩

**lemma** *UNIV-not-empty* [*iff*]: $UNIV \neq \{\}$
⟨*proof*⟩

**lemma** *empty-not-UNIV*[*simp*]: $\{\} \neq UNIV$
⟨*proof*⟩

### 8.3.5 The Powerset operator – Pow

**definition** $Pow :: 'a\ set \Rightarrow 'a\ set\ set$
**where** *Pow-def*: $Pow\ A = \{B.\ B \subseteq A\}$

**lemma** *Pow-iff* [*iff*]: $A \in Pow\ B \longleftrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *PowI*: $A \subseteq B \Longrightarrow A \in Pow\ B$
⟨*proof*⟩

**lemma** *PowD*: $A \in Pow\ B \Longrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *Pow-bottom*: $\{\} \in Pow\ B$
⟨*proof*⟩

**lemma** *Pow-top*: $A \in Pow\ A$
⟨*proof*⟩

**lemma** *Pow-not-empty*: $Pow\ A \neq \{\}$
⟨*proof*⟩

### 8.3.6 Set complement

**lemma** *Compl-iff* [*simp*]: $c \in -\ A \longleftrightarrow c \notin A$
⟨*proof*⟩

**lemma** *ComplI* [*intro*!]: $(c \in A \Longrightarrow False) \Longrightarrow c \in -\ A$
⟨*proof*⟩

This form, with negated conclusion, works well with the Classical prover. Negated assumptions behave like formulae on the right side of the notional turnstile ...

**lemma** *ComplD* [*dest*!]: $c \in - A \Longrightarrow c \notin A$
⟨*proof*⟩

**lemmas** *ComplE* = *ComplD* [*elim-format*]

**lemma** *Compl-eq*: $- A = \{x. \neg x \in A\}$
⟨*proof*⟩

### 8.3.7 Binary intersection

**abbreviation** *inter* :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ (**infixl** ‹∩› *70*)
**where** $(\cap) \equiv inf$

**notation** (*ASCII*)
*inter* (**infixl** ‹*Int*› *70*)

**lemma** *Int-def*: $A \cap B = \{x. x \in A \wedge x \in B\}$
⟨*proof*⟩

**lemma** *Int-iff* [*simp*]: $c \in A \cap B \longleftrightarrow c \in A \wedge c \in B$
⟨*proof*⟩

**lemma** *IntI* [*intro*!]: $c \in A \Longrightarrow c \in B \Longrightarrow c \in A \cap B$
⟨*proof*⟩

**lemma** *IntD1*: $c \in A \cap B \Longrightarrow c \in A$
⟨*proof*⟩

**lemma** *IntD2*: $c \in A \cap B \Longrightarrow c \in B$
⟨*proof*⟩

**lemma** *IntE* [*elim*!]: $c \in A \cap B \Longrightarrow (c \in A \Longrightarrow c \in B \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

### 8.3.8 Binary union

**abbreviation** *union* :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ (**infixl** ‹∪› *65*)
**where** $union \equiv sup$

**notation** (*ASCII*)
*union* (**infixl** ‹*Un*› *65*)

**lemma** *Un-def*: $A \cup B = \{x. x \in A \vee x \in B\}$
⟨*proof*⟩

**lemma** *Un-iff* [*simp*]: $c \in A \cup B \longleftrightarrow c \in A \vee c \in B$
⟨*proof*⟩

**lemma** *UnI1* [*elim?*]: $c \in A \Longrightarrow c \in A \cup B$
⟨*proof*⟩

**lemma** *UnI2* [*elim?*]: $c \in B \Longrightarrow c \in A \cup B$
⟨*proof*⟩

Classical introduction rule: no commitment to $A$ vs. $B$.

**lemma** *UnCI* [*intro*!]: $(c \notin B \Longrightarrow c \in A) \Longrightarrow c \in A \cup B$
⟨*proof*⟩

**lemma** *UnE* [*elim*!]: $c \in A \cup B \Longrightarrow (c \in A \Longrightarrow P) \Longrightarrow (c \in B \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *insert-def*: $insert\ a\ B = \{x.\ x = a\} \cup B$
⟨*proof*⟩

### 8.3.9 Set difference

**lemma** *Diff-iff* [*simp*]: $c \in A - B \longleftrightarrow c \in A \wedge c \notin B$
⟨*proof*⟩

**lemma** *DiffI* [*intro*!]: $c \in A \Longrightarrow c \notin B \Longrightarrow c \in A - B$
⟨*proof*⟩

**lemma** *DiffD1*: $c \in A - B \Longrightarrow c \in A$
⟨*proof*⟩

**lemma** *DiffD2*: $c \in A - B \Longrightarrow c \in B \Longrightarrow P$
⟨*proof*⟩

**lemma** *DiffE* [*elim*!]: $c \in A - B \Longrightarrow (c \in A \Longrightarrow c \notin B \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *set-diff-eq*: $A - B = \{x.\ x \in A \wedge x \notin B\}$
⟨*proof*⟩

**lemma** *Compl-eq-Diff-UNIV*: $-\ A = (UNIV - A)$
⟨*proof*⟩

**abbreviation** *sym-diff* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set$ **where**
$sym\text{-}diff\ A\ B \equiv ((A - B) \cup (B-A))$

### 8.3.10 Augmenting a set – *insert*

**lemma** *insert-iff* [*simp*]: $a \in insert\ b\ A \longleftrightarrow a = b \vee a \in A$
⟨*proof*⟩

**lemma** *insertI1*: $a \in insert\ a\ B$
⟨*proof*⟩

**lemma** *insertI2*: $a \in B \Longrightarrow a \in insert\ b\ B$

⟨*proof*⟩

**lemma** *insertE* [*elim*!]: $a \in insert\ b\ A \Longrightarrow (a = b \Longrightarrow P) \Longrightarrow (a \in A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *insertCI* [*intro*!]: $(a \notin B \Longrightarrow a = b) \Longrightarrow a \in insert\ b\ B$
— Classical introduction rule.
⟨*proof*⟩

**lemma** *subset-insert-iff*: $A \subseteq insert\ x\ B \longleftrightarrow (if\ x \in A\ then\ A - \{x\} \subseteq B\ else\ A \subseteq B)$
⟨*proof*⟩

**lemma** *set-insert*:
**assumes** $x \in A$
**obtains** $B$ **where** $A = insert\ x\ B$ **and** $x \notin B$
⟨*proof*⟩

**lemma** *insert-ident*: $x \notin A \Longrightarrow x \notin B \Longrightarrow insert\ x\ A = insert\ x\ B \longleftrightarrow A = B$
⟨*proof*⟩

**lemma** *insert-eq-iff*:
**assumes** $a \notin A$ $b \notin B$
**shows** $insert\ a\ A = insert\ b\ B \longleftrightarrow$
$(if\ a = b\ then\ A = B\ else\ \exists C.\ A = insert\ b\ C \land b \notin C \land B = insert\ a\ C \land a \notin C)$
(**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

**lemma** *insert-UNIV*[*simp*]: $insert\ x\ UNIV = UNIV$
⟨*proof*⟩

### 8.3.11 Singletons, using insert

**lemma** *singletonI* [*intro*!]: $a \in \{a\}$
— Redundant? But unlike *insertCI*, it proves the subgoal immediately!
⟨*proof*⟩

**lemma** *singletonD* [*dest*!]: $b \in \{a\} \Longrightarrow b = a$
⟨*proof*⟩

**lemmas** *singletonE* = *singletonD* [*elim-format*]

**lemma** *singleton-iff*: $b \in \{a\} \longleftrightarrow b = a$
⟨*proof*⟩

**lemma** *singleton-inject* [*dest*!]: $\{a\} = \{b\} \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *singleton-insert-inj-eq* [*iff*]: $\{b\} = insert\ a\ A \longleftrightarrow a = b \wedge A \subseteq \{b\}$
⟨*proof*⟩

**lemma** *singleton-insert-inj-eq'* [*iff*]: $insert\ a\ A = \{b\} \longleftrightarrow a = b \wedge A \subseteq \{b\}$
⟨*proof*⟩

**lemma** *subset-singletonD*: $A \subseteq \{x\} \Longrightarrow A = \{\} \vee A = \{x\}$
⟨*proof*⟩

**lemma** *subset-singleton-iff*: $X \subseteq \{a\} \longleftrightarrow X = \{\} \vee X = \{a\}$
⟨*proof*⟩

**lemma** *subset-singleton-iff-Uniq*: $(\exists\, a.\ A \subseteq \{a\}) \longleftrightarrow (\exists_{\leq 1} x.\ x \in A)$
⟨*proof*⟩

**lemma** *singleton-conv* [*simp*]: $\{x.\ x = a\} = \{a\}$
⟨*proof*⟩

**lemma** *singleton-conv2* [*simp*]: $\{x.\ a = x\} = \{a\}$
⟨*proof*⟩

**lemma** *Diff-single-insert*: $A - \{x\} \subseteq B \Longrightarrow A \subseteq insert\ x\ B$
⟨*proof*⟩

**lemma** *subset-Diff-insert*: $A \subseteq B - insert\ x\ C \longleftrightarrow A \subseteq B - C \wedge x \notin A$
⟨*proof*⟩

**lemma** *doubleton-eq-iff*: $\{a, b\} = \{c, d\} \longleftrightarrow a = c \wedge b = d \vee a = d \wedge b = c$
⟨*proof*⟩

**lemma** *Un-singleton-iff*: $A \cup B = \{x\} \longleftrightarrow A = \{\} \wedge B = \{x\} \vee A = \{x\} \wedge B = \{\} \vee A = \{x\} \wedge B = \{x\}$
⟨*proof*⟩

**lemma** *singleton-Un-iff*: $\{x\} = A \cup B \longleftrightarrow A = \{\} \wedge B = \{x\} \vee A = \{x\} \wedge B = \{\} \vee A = \{x\} \wedge B = \{x\}$
⟨*proof*⟩

### 8.3.12 Image of a set under a function

Frequently $b$ does not have the syntactic form of $f\ x$.

**definition** *image* :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set$ (**infixr** ‹'› *90*)
**where** $f\ `\ A = \{y.\ \exists x{\in}A.\ y = f\ x\}$

**lemma** *image-eqI* [*simp*, *intro*]: $b = f\ x \Longrightarrow x \in A \Longrightarrow b \in f\ `\ A$
⟨*proof*⟩

**lemma** *imageI*: $x \in A \Longrightarrow f\ x \in f\ `\ A$

⟨*proof*⟩

**lemma** *rev-image-eqI*: $x \in A \Longrightarrow b = f\,x \Longrightarrow b \in f\ `\ A$
— This version's more effective when we already have the required $x$.
⟨*proof*⟩

**lemma** *imageE* [*elim*!]:
**assumes** $b \in (\lambda x.\ f\,x)\ `\ A$ — The eta-expansion gives variable-name preservation.
**obtains** $x$ **where** $b = f\,x$ **and** $x \in A$
⟨*proof*⟩

**lemma** *Compr-image-eq*: $\{x \in f\ `\ A.\ P\ x\} = f\ `\ \{x \in A.\ P\ (f\,x)\}$
⟨*proof*⟩

**lemma** *image-Un*: $f\ `\ (A \cup B) = f\ `\ A \cup f\ `\ B$
⟨*proof*⟩

**lemma** *image-iff*: $z \in f\ `\ A \longleftrightarrow (\exists\,x{\in}A.\ z = f\,x)$
⟨*proof*⟩

**lemma** *image-subsetI*: $(\bigwedge x.\ x \in A \Longrightarrow f\,x \in B) \Longrightarrow f\ `\ A \subseteq B$
— Replaces the three steps *subsetI*, *imageE*, *hypsubst*, but breaks too many existing proofs.
⟨*proof*⟩

**lemma** *image-subset-iff*: $f\ `\ A \subseteq B \longleftrightarrow (\forall\,x{\in}A.\ f\,x \in B)$
— This rewrite rule would confuse users if made default.
⟨*proof*⟩

**lemma** *subset-imageE*:
**assumes** $B \subseteq f\ `\ A$
**obtains** $C$ **where** $C \subseteq A$ **and** $B = f\ `\ C$
⟨*proof*⟩

**lemma** *subset-image-iff*: $B \subseteq f\ `\ A \longleftrightarrow (\exists\,AA{\subseteq}A.\ B = f\ `\ AA)$
⟨*proof*⟩

**lemma** *image-ident* [*simp*]: $(\lambda x.\ x)\ `\ Y = Y$
⟨*proof*⟩

**lemma** *image-empty* [*simp*]: $f\ `\ \{\} = \{\}$
⟨*proof*⟩

**lemma** *image-insert* [*simp*]: $f\ `\ \mathit{insert}\ a\ B = \mathit{insert}\ (f\ a)\ (f\ `\ B)$
⟨*proof*⟩

**lemma** *image-constant*: $x \in A \Longrightarrow (\lambda x.\ c)\ `\ A = \{c\}$
⟨*proof*⟩

**lemma** *image-constant-conv*: $(\lambda x.\ c)$ ‘ $A = (if\ A = \{\}\ then\ \{\}\ else\ \{c\})$
⟨*proof*⟩

**lemma** *image-image*: $f$ ‘ $(g$ ‘ $A) = (\lambda x.\ f\ (g\ x))$ ‘ $A$
⟨*proof*⟩

**lemma** *insert-image* [*simp*]: $x \in A \Longrightarrow insert\ (f\ x)\ (f$ ‘ $A) = f$ ‘ $A$
⟨*proof*⟩

**lemma** *image-is-empty* [*iff*]: $f$ ‘ $A = \{\} \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *empty-is-image* [*iff*]: $\{\} = f$ ‘ $A \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *image-Collect*: $f$ ‘ $\{x.\ P\ x\} = \{f\ x \mid x.\ P\ x\}$
— NOT suitable as a default simp rule: the RHS isn't simpler than the LHS, with its implicit quantifier and conjunction. Also image enjoys better equational properties than does the RHS.
⟨*proof*⟩

**lemma** *if-image-distrib* [*simp*]:
$(\lambda x.\ if\ P\ x\ then\ f\ x\ else\ g\ x)$ ‘ $S = f$ ‘ $(S \cap \{x.\ P\ x\}) \cup g$ ‘ $(S \cap \{x.\ \neg\ P\ x\})$
⟨*proof*⟩

**lemma** *image-cong*:
$f$ ‘ $M = g$ ‘ $N$ **if** $M = N$ $\bigwedge x.\ x \in N \Longrightarrow f\ x = g\ x$
⟨*proof*⟩

**lemma** *image-cong-simp* [*cong*]:
$f$ ‘ $M = g$ ‘ $N$ **if** $M = N$ $\bigwedge x.\ x \in N$ =*simp*=> $f\ x = g\ x$
⟨*proof*⟩

**lemma** *image-Int-subset*: $f$ ‘ $(A \cap B) \subseteq f$ ‘ $A \cap f$ ‘ $B$
⟨*proof*⟩

**lemma** *image-diff-subset*: $f$ ‘ $A - f$ ‘ $B \subseteq f$ ‘ $(A - B)$
⟨*proof*⟩

**lemma** *Setcompr-eq-image*: $\{f\ x \mid x.\ x \in A\} = f$ ‘ $A$
⟨*proof*⟩

**lemma** *setcompr-eq-image*: $\{f\ x \mid x.\ P\ x\} = f$ ‘ $\{x.\ P\ x\}$
⟨*proof*⟩

**lemma** *ball-imageD*: $\forall x \in f$ ‘ $A.\ P\ x \Longrightarrow \forall x \in A.\ P\ (f\ x)$
⟨*proof*⟩

**lemma** *bex-imageD*: $\exists x \in f$ ‘ $A.\ P\ x \Longrightarrow \exists x \in A.\ P\ (f\ x)$

⟨*proof*⟩

**lemma** *image-add-0* [*simp*]: $(+)$ $(0::'a::comm\text{-}monoid\text{-}add)$ ‘ $S = S$
⟨*proof*⟩

**theorem** *Cantors-theorem*: $\nexists f.\ f$ ‘ $A = Pow\ A$
⟨*proof*⟩

Range of a function – just an abbreviation for image!

**abbreviation** *range* :: $('a \Rightarrow 'b) \Rightarrow 'b\ set$ — of function
**where** *range f* $\equiv f$ ‘ *UNIV*

**lemma** *range-eqI*: $b = f\ x \Longrightarrow b \in range\ f$
⟨*proof*⟩

**lemma** *rangeI*: $f\ x \in range\ f$
⟨*proof*⟩

**lemma** *rangeE* [*elim?*]: $b \in range\ (\lambda x.\ f\ x) \Longrightarrow (\bigwedge x.\ b = f\ x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *range-subsetD*: $range\ f \subseteq B \Longrightarrow f\ i \in B$
⟨*proof*⟩

**lemma** *full-SetCompr-eq*: $\{u.\ \exists x.\ u = f\ x\} = range\ f$
⟨*proof*⟩

**lemma** *range-composition*: $range\ (\lambda x.\ f\ (g\ x)) = f$ ‘ $range\ g$
⟨*proof*⟩

**lemma** *range-constant* [*simp*]: $range\ (\lambda\text{-}.\ x) = \{x\}$
⟨*proof*⟩

**lemma** *range-eq-singletonD*: $range\ f = \{a\} \Longrightarrow f\ x = a$
⟨*proof*⟩

### 8.3.13 Some rules with *if*

Elimination of $\{x.\ \ldots \wedge x = t \wedge \ldots\}$.

**lemma** *Collect-conv-if*: $\{x.\ x = a \wedge P\ x\} = (if\ P\ a\ then\ \{a\}\ else\ \{\})$
⟨*proof*⟩

**lemma** *Collect-conv-if2*: $\{x.\ a = x \wedge P\ x\} = (if\ P\ a\ then\ \{a\}\ else\ \{\})$
⟨*proof*⟩

Rewrite rules for boolean case-splitting: faster than *if-split* [*split*].

**lemma** *if-split-eq1*: $(if\ Q\ then\ x\ else\ y) = b \longleftrightarrow (Q \longrightarrow x = b) \wedge (\neg\ Q \longrightarrow y = b)$

$\langle proof \rangle$

**lemma** *if-split-eq2*: $a = (\textit{if } Q \textit{ then } x \textit{ else } y) \longleftrightarrow (Q \longrightarrow a = x) \wedge (\neg\ Q \longrightarrow a = y)$
$\langle proof \rangle$

Split ifs on either side of the membership relation. Not for [*simp*] – can cause goals to blow up!

**lemma** *if-split-mem1*: $(\textit{if } Q \textit{ then } x \textit{ else } y) \in b \longleftrightarrow (Q \longrightarrow x \in b) \wedge (\neg\ Q \longrightarrow y \in b)$
$\langle proof \rangle$

**lemma** *if-split-mem2*: $(a \in (\textit{if } Q \textit{ then } x \textit{ else } y)) \longleftrightarrow (Q \longrightarrow a \in x) \wedge (\neg\ Q \longrightarrow a \in y)$
$\langle proof \rangle$

**lemmas** *split-ifs* = *if-bool-eq-conj if-split-eq1 if-split-eq2 if-split-mem1 if-split-mem2*

## 8.4 Further operations and lemmas

### 8.4.1 The "proper subset" relation

**lemma** *psubsetI* [*intro*!]: $A \subseteq B \Longrightarrow A \neq B \Longrightarrow A \subset B$
$\langle proof \rangle$

**lemma** *psubsetE* [*elim*!]: $A \subset B \Longrightarrow (A \subseteq B \Longrightarrow \neg\ B \subseteq A \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

**lemma** *psubset-insert-iff*:
$A \subset \textit{insert } x\ B \longleftrightarrow (\textit{if } x \in B \textit{ then } A \subset B \textit{ else if } x \in A \textit{ then } A - \{x\} \subset B \textit{ else } A \subseteq B)$
$\langle proof \rangle$

**lemma** *psubset-eq*: $A \subset B \longleftrightarrow A \subseteq B \wedge A \neq B$
$\langle proof \rangle$

**lemma** *psubset-imp-subset*: $A \subset B \Longrightarrow A \subseteq B$
$\langle proof \rangle$

**lemma** *psubset-trans*: $A \subset B \Longrightarrow B \subset C \Longrightarrow A \subset C$
$\langle proof \rangle$

**lemma** *psubsetD*: $A \subset B \Longrightarrow c \in A \Longrightarrow c \in B$
$\langle proof \rangle$

**lemma** *psubset-subset-trans*: $A \subset B \Longrightarrow B \subseteq C \Longrightarrow A \subset C$
$\langle proof \rangle$

**lemma** *subset-psubset-trans*: $A \subseteq B \Longrightarrow B \subset C \Longrightarrow A \subset C$
$\langle proof \rangle$

**lemma** *psubset-imp-ex-mem*: $A \subset B \Longrightarrow \exists\, b.\ b \in B - A$
⟨*proof*⟩

**lemma** *atomize-ball*: $(\bigwedge x.\ x \in A \Longrightarrow P\ x) \equiv \mathit{Trueprop}\ (\forall\, x{\in}A.\ P\ x)$
⟨*proof*⟩

**lemmas** [*symmetric*, *rulify*] = *atomize-ball*
**and** [*symmetric*, *defn*] = *atomize-ball*

**lemma** *image-Pow-mono*: $f\ `\ A \subseteq B \Longrightarrow \mathit{image}\ f\ `\ \mathit{Pow}\ A \subseteq \mathit{Pow}\ B$
⟨*proof*⟩

**lemma** *image-Pow-surj*: $f\ `\ A = B \Longrightarrow \mathit{image}\ f\ `\ \mathit{Pow}\ A = \mathit{Pow}\ B$
⟨*proof*⟩

### 8.4.2 Derived rules involving subsets.

*insert*.

**lemma** *subset-insertI*: $B \subseteq \mathit{insert}\ a\ B$
⟨*proof*⟩

**lemma** *subset-insertI2*: $A \subseteq B \Longrightarrow A \subseteq \mathit{insert}\ b\ B$
⟨*proof*⟩

**lemma** *subset-insert*: $x \notin A \Longrightarrow A \subseteq \mathit{insert}\ x\ B \longleftrightarrow A \subseteq B$
⟨*proof*⟩

Finite Union – the least upper bound of two sets.

**lemma** *Un-upper1*: $A \subseteq A \cup B$
⟨*proof*⟩

**lemma** *Un-upper2*: $B \subseteq A \cup B$
⟨*proof*⟩

**lemma** *Un-least*: $A \subseteq C \Longrightarrow B \subseteq C \Longrightarrow A \cup B \subseteq C$
⟨*proof*⟩

Finite Intersection – the greatest lower bound of two sets.

**lemma** *Int-lower1*: $A \cap B \subseteq A$
⟨*proof*⟩

**lemma** *Int-lower2*: $A \cap B \subseteq B$
⟨*proof*⟩

**lemma** *Int-greatest*: $C \subseteq A \Longrightarrow C \subseteq B \Longrightarrow C \subseteq A \cap B$
⟨*proof*⟩

Set difference.

**lemma** *Diff-subset*[*simp*]: $A - B \subseteq A$
⟨*proof*⟩

**lemma** *Diff-subset-conv*: $A - B \subseteq C \longleftrightarrow A \subseteq B \cup C$
⟨*proof*⟩

### 8.4.3 Equalities involving union, intersection, inclusion, etc.

{}.

**lemma** *Collect-const* [*simp*]: $\{s.\ P\} = (\mathit{if}\ P\ \mathit{then}\ \mathit{UNIV}\ \mathit{else}\ \{\})$
— supersedes *Collect-False-empty*
⟨*proof*⟩

**lemma** *subset-empty* [*simp*]: $A \subseteq \{\} \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *not-psubset-empty* [*iff*]: $\neg\ (A < \{\})$
⟨*proof*⟩

**lemma** *Collect-subset* [*simp*]: $\{x{\in}A.\ P\ x\} \subseteq A$ ⟨*proof*⟩

**lemma** *Collect-empty-eq* [*simp*]: $\mathit{Collect}\ P = \{\} \longleftrightarrow (\forall x.\ \neg\ P\ x)$
⟨*proof*⟩

**lemma** *empty-Collect-eq* [*simp*]: $\{\} = \mathit{Collect}\ P \longleftrightarrow (\forall x.\ \neg\ P\ x)$
⟨*proof*⟩

**lemma** *Collect-neg-eq*: $\{x.\ \neg\ P\ x\} = -\ \{x.\ P\ x\}$
⟨*proof*⟩

**lemma** *Collect-disj-eq*: $\{x.\ P\ x \vee Q\ x\} = \{x.\ P\ x\} \cup \{x.\ Q\ x\}$
⟨*proof*⟩

**lemma** *Collect-imp-eq*: $\{x.\ P\ x \longrightarrow Q\ x\} = -\ \{x.\ P\ x\} \cup \{x.\ Q\ x\}$
⟨*proof*⟩

**lemma** *Collect-conj-eq*: $\{x.\ P\ x \wedge Q\ x\} = \{x.\ P\ x\} \cap \{x.\ Q\ x\}$
⟨*proof*⟩

**lemma** *Collect-conj-eq2*: $\{x \in A.\ P\ x \wedge Q\ x\} = \{x \in A.\ P\ x\} \cap \{x \in A.\ Q\ x\}$
⟨*proof*⟩

**lemma** *Collect-mono-iff*: $\mathit{Collect}\ P \subseteq \mathit{Collect}\ Q \longleftrightarrow (\forall x.\ P\ x \longrightarrow Q\ x)$
⟨*proof*⟩

*insert*.

**lemma** *insert-is-Un*: $\mathit{insert}\ a\ A = \{a\} \cup A$

— NOT SUITABLE FOR REWRITING since $\{a\} \equiv insert\ a\ \{\}$
⟨*proof*⟩

**lemma** *insert-not-empty* [*simp*]: $insert\ a\ A \neq \{\}$
**and** *empty-not-insert* [*simp*]: $\{\} \neq insert\ a\ A$
⟨*proof*⟩

**lemma** *insert-absorb*: $a \in A \Longrightarrow insert\ a\ A = A$
— [*simp*] causes recursive calls when there are nested inserts
— with *quadratic* running time
⟨*proof*⟩

**lemma** *insert-absorb2* [*simp*]: $insert\ x\ (insert\ x\ A) = insert\ x\ A$
⟨*proof*⟩

**lemma** *insert-commute*: $insert\ x\ (insert\ y\ A) = insert\ y\ (insert\ x\ A)$
⟨*proof*⟩

**lemma** *insert-subset* [*simp*]: $insert\ x\ A \subseteq B \longleftrightarrow x \in B \wedge A \subseteq B$
⟨*proof*⟩

**lemma** *mk-disjoint-insert*: $a \in A \Longrightarrow \exists B.\ A = insert\ a\ B \wedge a \notin B$
— use new $B$ rather than $A - \{a\}$ to avoid infinite unfolding
⟨*proof*⟩

**lemma** *insert-Collect*: $insert\ a\ (Collect\ P) = \{u.\ u \neq a \longrightarrow P\ u\}$
⟨*proof*⟩

**lemma** *insert-inter-insert* [*simp*]: $insert\ a\ A \cap insert\ a\ B = insert\ a\ (A \cap B)$
⟨*proof*⟩

**lemma** *insert-disjoint* [*simp*]:
$insert\ a\ A \cap B = \{\} \longleftrightarrow a \notin B \wedge A \cap B = \{\}$
$\{\} = insert\ a\ A \cap B \longleftrightarrow a \notin B \wedge \{\} = A \cap B$
⟨*proof*⟩

**lemma** *disjoint-insert* [*simp*]:
$B \cap insert\ a\ A = \{\} \longleftrightarrow a \notin B \wedge B \cap A = \{\}$
$\{\} = A \cap insert\ b\ B \longleftrightarrow b \notin A \wedge \{\} = A \cap B$
⟨*proof*⟩

*Int*

**lemma** *Int-absorb*: $A \cap A = A$
⟨*proof*⟩

**lemma** *Int-left-absorb*: $A \cap (A \cap B) = A \cap B$
⟨*proof*⟩

**lemma** *Int-commute*: $A \cap B = B \cap A$

⟨*proof*⟩

**lemma** *Int-left-commute*: $A \cap (B \cap C) = B \cap (A \cap C)$
⟨*proof*⟩

**lemma** *Int-assoc*: $(A \cap B) \cap C = A \cap (B \cap C)$
⟨*proof*⟩

**lemmas** *Int-ac* = *Int-assoc Int-left-absorb Int-commute Int-left-commute*
— Intersection is an AC-operator

**lemma** *Int-absorb1*: $B \subseteq A \Longrightarrow A \cap B = B$
⟨*proof*⟩

**lemma** *Int-absorb2*: $A \subseteq B \Longrightarrow A \cap B = A$
⟨*proof*⟩

**lemma** *Int-empty-left*: $\{\} \cap B = \{\}$
⟨*proof*⟩

**lemma** *Int-empty-right*: $A \cap \{\} = \{\}$
⟨*proof*⟩

**lemma** *disjoint-eq-subset-Compl*: $A \cap B = \{\} \longleftrightarrow A \subseteq - B$
⟨*proof*⟩

**lemma** *disjoint-iff*: $A \cap B = \{\} \longleftrightarrow (\forall x.\ x{\in}A \longrightarrow x \notin B)$
⟨*proof*⟩

**lemma** *disjoint-iff-not-equal*: $A \cap B = \{\} \longleftrightarrow (\forall x{\in}A.\ \forall y{\in}B.\ x \neq y)$
⟨*proof*⟩

**lemma** *Int-UNIV-left*: $UNIV \cap B = B$
⟨*proof*⟩

**lemma** *Int-UNIV-right*: $A \cap UNIV = A$
⟨*proof*⟩

**lemma** *Int-Un-distrib*: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
⟨*proof*⟩

**lemma** *Int-Un-distrib2*: $(B \cup C) \cap A = (B \cap A) \cup (C \cap A)$
⟨*proof*⟩

**lemma** *Int-UNIV*: $A \cap B = UNIV \longleftrightarrow A = UNIV \wedge B = UNIV$
⟨*proof*⟩

**lemma** *Int-subset-iff*: $C \subseteq A \cap B \longleftrightarrow C \subseteq A \wedge C \subseteq B$
⟨*proof*⟩

**lemma** *Int-Collect*: $x \in A \cap \{x.\ P\ x\} \longleftrightarrow x \in A \wedge P\ x$
⟨*proof*⟩

*Un.*

**lemma** *Un-absorb*: $A \cup A = A$
⟨*proof*⟩

**lemma** *Un-left-absorb*: $A \cup (A \cup B) = A \cup B$
⟨*proof*⟩

**lemma** *Un-commute*: $A \cup B = B \cup A$
⟨*proof*⟩

**lemma** *Un-left-commute*: $A \cup (B \cup C) = B \cup (A \cup C)$
⟨*proof*⟩

**lemma** *Un-assoc*: $(A \cup B) \cup C = A \cup (B \cup C)$
⟨*proof*⟩

**lemmas** *Un-ac* = *Un-assoc Un-left-absorb Un-commute Un-left-commute*
— Union is an AC-operator

**lemma** *Un-absorb1*: $A \subseteq B \Longrightarrow A \cup B = B$
⟨*proof*⟩

**lemma** *Un-absorb2*: $B \subseteq A \Longrightarrow A \cup B = A$
⟨*proof*⟩

**lemma** *Un-empty-left*: $\{\} \cup B = B$
⟨*proof*⟩

**lemma** *Un-empty-right*: $A \cup \{\} = A$
⟨*proof*⟩

**lemma** *Un-UNIV-left*: $UNIV \cup B = UNIV$
⟨*proof*⟩

**lemma** *Un-UNIV-right*: $A \cup UNIV = UNIV$
⟨*proof*⟩

**lemma** *Un-insert-left* [*simp*]: $(insert\ a\ B) \cup C = insert\ a\ (B \cup C)$
⟨*proof*⟩

**lemma** *Un-insert-right* [*simp*]: $A \cup (insert\ a\ B) = insert\ a\ (A \cup B)$
⟨*proof*⟩

**lemma** *Int-insert-left*: $(insert\ a\ B) \cap C = (if\ a \in C\ then\ insert\ a\ (B \cap C)\ else\ B \cap C)$

⟨*proof*⟩

**lemma** *Int-insert-left-if0* [*simp*]: $a \notin C \Longrightarrow (insert\ a\ B) \cap C = B \cap C$
⟨*proof*⟩

**lemma** *Int-insert-left-if1* [*simp*]: $a \in C \Longrightarrow (insert\ a\ B) \cap C = insert\ a\ (B \cap C)$
⟨*proof*⟩

**lemma** *Int-insert-right*: $A \cap (insert\ a\ B) = (if\ a \in A\ then\ insert\ a\ (A \cap B)\ else\ A \cap B)$
⟨*proof*⟩

**lemma** *Int-insert-right-if0* [*simp*]: $a \notin A \Longrightarrow A \cap (insert\ a\ B) = A \cap B$
⟨*proof*⟩

**lemma** *Int-insert-right-if1* [*simp*]: $a \in A \Longrightarrow A \cap (insert\ a\ B) = insert\ a\ (A \cap B)$
⟨*proof*⟩

**lemma** *Un-Int-distrib*: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
⟨*proof*⟩

**lemma** *Un-Int-distrib2*: $(B \cap C) \cup A = (B \cup A) \cap (C \cup A)$
⟨*proof*⟩

**lemma** *Un-Int-crazy*: $(A \cap B) \cup (B \cap C) \cup (C \cap A) = (A \cup B) \cap (B \cup C) \cap (C \cup A)$
⟨*proof*⟩

**lemma** *subset-Un-eq*: $A \subseteq B \longleftrightarrow A \cup B = B$
⟨*proof*⟩

**lemma** *Un-empty* [*iff*]: $A \cup B = \{\} \longleftrightarrow A = \{\} \wedge B = \{\}$
⟨*proof*⟩

**lemma** *Un-subset-iff*: $A \cup B \subseteq C \longleftrightarrow A \subseteq C \wedge B \subseteq C$
⟨*proof*⟩

**lemma** *Un-Diff-Int*: $(A - B) \cup (A \cap B) = A$
⟨*proof*⟩

**lemma** *Diff-Int2*: $A \cap C - B \cap C = A \cap C - B$
⟨*proof*⟩

**lemma** *subset-UnE*:
**assumes** $C \subseteq A \cup B$
**obtains** $A'\ B'$ **where** $A' \subseteq A\ B' \subseteq B\ C = A' \cup B'$
⟨*proof*⟩

**lemma** *Un-Int-eq* [*simp*]: $(S \cup T) \cap S = S$ $(S \cup T) \cap T = T$ $S \cap (S \cup T) = S$ $T \cap (S \cup T) = T$
⟨*proof*⟩

**lemma** *Int-Un-eq* [*simp*]: $(S \cap T) \cup S = S$ $(S \cap T) \cup T = T$ $S \cup (S \cap T) = S$ $T \cup (S \cap T) = T$
⟨*proof*⟩

Set complement

**lemma** *Compl-disjoint* [*simp*]: $A \cap - A = \{\}$
⟨*proof*⟩

**lemma** *Compl-disjoint2* [*simp*]: $- A \cap A = \{\}$
⟨*proof*⟩

**lemma** *Compl-partition*: $A \cup - A = UNIV$
⟨*proof*⟩

**lemma** *Compl-partition2*: $- A \cup A = UNIV$
⟨*proof*⟩

**lemma** *double-complement*: $- (-A) = A$ **for** $A :: 'a\ set$
⟨*proof*⟩

**lemma** *Compl-Un*: $- (A \cup B) = (- A) \cap (- B)$
⟨*proof*⟩

**lemma** *Compl-Int*: $- (A \cap B) = (- A) \cup (- B)$
⟨*proof*⟩

**lemma** *subset-Compl-self-eq*: $A \subseteq - A \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *Un-Int-assoc-eq*: $(A \cap B) \cup C = A \cap (B \cup C) \longleftrightarrow C \subseteq A$
— Halmos, Naive Set Theory, page 16.
⟨*proof*⟩

**lemma** *Compl-UNIV-eq*: $- UNIV = \{\}$
⟨*proof*⟩

**lemma** *Compl-empty-eq*: $- \{\} = UNIV$
⟨*proof*⟩

**lemma** *Compl-subset-Compl-iff* [*iff*]: $- A \subseteq - B \longleftrightarrow B \subseteq A$
⟨*proof*⟩

**lemma** *Compl-eq-Compl-iff* [*iff*]: $- A = - B \longleftrightarrow A = B$
**for** $A\ B :: 'a\ set$
⟨*proof*⟩

**lemma** *Compl-insert*: $- \mathit{insert}\ x\ A = (- A) - \{x\}$
⟨*proof*⟩

Bounded quantifiers.

The following are not added to the default simpset because (a) they duplicate the body and (b) there are no similar rules for *Int*.

**lemma** *ball-Un*: $(\forall x \in A \cup B.\ P\ x) \longleftrightarrow (\forall x{\in}A.\ P\ x) \wedge (\forall x{\in}B.\ P\ x)$
⟨*proof*⟩

**lemma** *bex-Un*: $(\exists x \in A \cup B.\ P\ x) \longleftrightarrow (\exists x{\in}A.\ P\ x) \vee (\exists x{\in}B.\ P\ x)$
⟨*proof*⟩

Set difference.

**lemma** *Diff-eq*: $A - B = A \cap (- B)$
⟨*proof*⟩

**lemma** *Diff-eq-empty-iff*: $A - B = \{\} \longleftrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *Diff-cancel* [*simp*]: $A - A = \{\}$
⟨*proof*⟩

**lemma** *Diff-idemp* [*simp*]: $(A - B) - B = A - B$
**for** $A\ B :: {}'a\ set$
⟨*proof*⟩

**lemma** *Diff-triv*: $A \cap B = \{\} \Longrightarrow A - B = A$
⟨*proof*⟩

**lemma** *empty-Diff* [*simp*]: $\{\} - A = \{\}$
⟨*proof*⟩

**lemma** *Diff-empty* [*simp*]: $A - \{\} = A$
⟨*proof*⟩

**lemma** *Diff-UNIV* [*simp*]: $A - \mathit{UNIV} = \{\}$
⟨*proof*⟩

**lemma** *Diff-insert0* [*simp*]: $x \notin A \Longrightarrow A - \mathit{insert}\ x\ B = A - B$
⟨*proof*⟩

**lemma** *Diff-insert*: $A - \mathit{insert}\ a\ B = A - B - \{a\}$
— NOT SUITABLE FOR REWRITING since $\{a\} \equiv \mathit{insert}\ a\ 0$
⟨*proof*⟩

**lemma** *Diff-insert2*: $A - \mathit{insert}\ a\ B = A - \{a\} - B$
— NOT SUITABLE FOR REWRITING since $\{a\} \equiv \mathit{insert}\ a\ 0$

⟨*proof*⟩

**lemma** *insert-Diff-if*: $insert\ x\ A - B = (if\ x \in B\ then\ A - B\ else\ insert\ x\ (A - B))$
⟨*proof*⟩

**lemma** *insert-Diff1* [*simp*]: $x \in B \Longrightarrow insert\ x\ A - B = A - B$
⟨*proof*⟩

**lemma** *insert-Diff-single*[*simp*]: $insert\ a\ (A - \{a\}) = insert\ a\ A$
⟨*proof*⟩

**lemma** *insert-Diff*: $a \in A \Longrightarrow insert\ a\ (A - \{a\}) = A$
⟨*proof*⟩

**lemma** *Diff-insert-absorb*: $x \notin A \Longrightarrow (insert\ x\ A) - \{x\} = A$
⟨*proof*⟩

**lemma** *Diff-disjoint* [*simp*]: $A \cap (B - A) = \{\}$
⟨*proof*⟩

**lemma** *Diff-partition*: $A \subseteq B \Longrightarrow A \cup (B - A) = B$
⟨*proof*⟩

**lemma** *double-diff*: $A \subseteq B \Longrightarrow B \subseteq C \Longrightarrow B - (C - A) = A$
⟨*proof*⟩

**lemma** *Un-Diff-cancel* [*simp*]: $A \cup (B - A) = A \cup B$
⟨*proof*⟩

**lemma** *Un-Diff-cancel2* [*simp*]: $(B - A) \cup A = B \cup A$
⟨*proof*⟩

**lemma** *Diff-Un*: $A - (B \cup C) = (A - B) \cap (A - C)$
⟨*proof*⟩

**lemma** *Diff-Int*: $A - (B \cap C) = (A - B) \cup (A - C)$
⟨*proof*⟩

**lemma** *Diff-Diff-Int*: $A - (A - B) = A \cap B$
⟨*proof*⟩

**lemma** *Un-Diff*: $(A \cup B) - C = (A - C) \cup (B - C)$
⟨*proof*⟩

**lemma** *Int-Diff*: $(A \cap B) - C = A \cap (B - C)$
⟨*proof*⟩

**lemma** *Diff-Int-distrib*: $C \cap (A - B) = (C \cap A) - (C \cap B)$

⟨*proof*⟩

**lemma** *Diff-Int-distrib2*: $(A - B) \cap C = (A \cap C) - (B \cap C)$
⟨*proof*⟩

**lemma** *Diff-Compl* [*simp*]: $A - (- B) = A \cap B$
⟨*proof*⟩

**lemma** *Compl-Diff-eq* [*simp*]: $- (A - B) = - A \cup B$
⟨*proof*⟩

**lemma** *subset-Compl-singleton* [*simp*]: $A \subseteq - \{b\} \longleftrightarrow b \notin A$
⟨*proof*⟩

Quantification over type *bool*.

**lemma** *bool-induct*: $P\ True \Longrightarrow P\ False \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *all-bool-eq*: $(\forall b.\ P\ b) \longleftrightarrow P\ True \wedge P\ False$
⟨*proof*⟩

**lemma** *bool-contrapos*: $P\ x \Longrightarrow \neg\ P\ False \Longrightarrow P\ True$
⟨*proof*⟩

**lemma** *ex-bool-eq*: $(\exists b.\ P\ b) \longleftrightarrow P\ True \vee P\ False$
⟨*proof*⟩

**lemma** *UNIV-bool*: $UNIV = \{False,\ True\}$
⟨*proof*⟩

*Pow*

**lemma** *Pow-empty* [*simp*]: $Pow\ \{\} = \{\{\}\}$
⟨*proof*⟩

**lemma** *Pow-singleton-iff* [*simp*]: $Pow\ X = \{Y\} \longleftrightarrow X = \{\} \wedge Y = \{\}$
⟨*proof*⟩

**lemma** *Pow-insert*: $Pow\ (insert\ a\ A) = Pow\ A \cup (insert\ a\ `\ Pow\ A)$
⟨*proof*⟩

**lemma** *Pow-Compl*: $Pow\ (- A) = \{- B \mid B.\ A \in Pow\ B\}$
⟨*proof*⟩

**lemma** *Pow-UNIV* [*simp*]: $Pow\ UNIV = UNIV$
⟨*proof*⟩

**lemma** *Un-Pow-subset*: $Pow\ A \cup Pow\ B \subseteq Pow\ (A \cup B)$
⟨*proof*⟩

**lemma** *Pow-Int-eq* [*simp*]: $Pow\ (A \cap B) = Pow\ A \cap Pow\ B$
⟨*proof*⟩

Miscellany.

**lemma** *Int-Diff-disjoint*: $A \cap B \cap (A - B) = \{\}$
⟨*proof*⟩

**lemma** *Int-Diff-Un*: $A \cap B \cup (A - B) = A$
⟨*proof*⟩

**lemma** *set-eq-subset*: $A = B \longleftrightarrow A \subseteq B \wedge B \subseteq A$
⟨*proof*⟩

**lemma** *subset-iff*: $A \subseteq B \longleftrightarrow (\forall\, t.\ t \in A \longrightarrow t \in B)$
⟨*proof*⟩

**lemma** *subset-iff-psubset-eq*: $A \subseteq B \longleftrightarrow A \subset B \vee A = B$
⟨*proof*⟩

**lemma** *all-not-in-conv* [*simp*]: $(\forall\, x.\ x \notin A) \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *ex-in-conv*: $(\exists\, x.\ x \in A) \longleftrightarrow A \neq \{\}$
⟨*proof*⟩

**lemma** *ball-simps* [*simp*, *no-atp*]:
$\bigwedge A\ P\ Q.\ (\forall\, x{\in}A.\ P\ x \vee Q) \longleftrightarrow ((\forall\, x{\in}A.\ P\ x) \vee Q)$
$\bigwedge A\ P\ Q.\ (\forall\, x{\in}A.\ P \vee Q\ x) \longleftrightarrow (P \vee (\forall\, x{\in}A.\ Q\ x))$
$\bigwedge A\ P\ Q.\ (\forall\, x{\in}A.\ P \longrightarrow Q\ x) \longleftrightarrow (P \longrightarrow (\forall\, x{\in}A.\ Q\ x))$
$\bigwedge A\ P\ Q.\ (\forall\, x{\in}A.\ P\ x \longrightarrow Q) \longleftrightarrow ((\exists\, x{\in}A.\ P\ x) \longrightarrow Q)$
$\bigwedge P.\ (\forall\, x{\in}\{\}.\ P\ x) \longleftrightarrow True$
$\bigwedge P.\ (\forall\, x{\in}UNIV.\ P\ x) \longleftrightarrow (\forall\, x.\ P\ x)$
$\bigwedge a\ B\ P.\ (\forall\, x{\in}insert\ a\ B.\ P\ x) \longleftrightarrow (P\ a \wedge (\forall\, x{\in}B.\ P\ x))$
$\bigwedge P\ Q.\ (\forall\, x{\in}Collect\ Q.\ P\ x) \longleftrightarrow (\forall\, x.\ Q\ x \longrightarrow P\ x)$
$\bigwedge A\ P\ f.\ (\forall\, x{\in}f`A.\ P\ x) \longleftrightarrow (\forall\, x{\in}A.\ P\ (f\ x))$
$\bigwedge A\ P.\ (\neg\ (\forall\, x{\in}A.\ P\ x)) \longleftrightarrow (\exists\, x{\in}A.\ \neg\ P\ x)$
⟨*proof*⟩

**lemma** *bex-simps* [*simp*, *no-atp*]:
$\bigwedge A\ P\ Q.\ (\exists\, x{\in}A.\ P\ x \wedge Q) \longleftrightarrow ((\exists\, x{\in}A.\ P\ x) \wedge Q)$
$\bigwedge A\ P\ Q.\ (\exists\, x{\in}A.\ P \wedge Q\ x) \longleftrightarrow (P \wedge (\exists\, x{\in}A.\ Q\ x))$
$\bigwedge P.\ (\exists\, x{\in}\{\}.\ P\ x) \longleftrightarrow False$
$\bigwedge P.\ (\exists\, x{\in}UNIV.\ P\ x) \longleftrightarrow (\exists\, x.\ P\ x)$
$\bigwedge a\ B\ P.\ (\exists\, x{\in}insert\ a\ B.\ P\ x) \longleftrightarrow (P\ a \vee (\exists\, x{\in}B.\ P\ x))$
$\bigwedge P\ Q.\ (\exists\, x{\in}Collect\ Q.\ P\ x) \longleftrightarrow (\exists\, x.\ Q\ x \wedge P\ x)$
$\bigwedge A\ P\ f.\ (\exists\, x{\in}f`A.\ P\ x) \longleftrightarrow (\exists\, x{\in}A.\ P\ (f\ x))$
$\bigwedge A\ P.\ (\neg(\exists\, x{\in}A.\ P\ x)) \longleftrightarrow (\forall\, x{\in}A.\ \neg\ P\ x)$
⟨*proof*⟩

**lemma** *ex-image-cong-iff* [*simp*, *no-atp*]:
$(\exists x.\ x{\in}f`A) \longleftrightarrow A \neq \{\}$ $(\exists x.\ x{\in}f`A \wedge P\ x) \longleftrightarrow (\exists x{\in}A.\ P\ (f\ x))$
⟨*proof*⟩

### 8.4.4 Monotonicity of various operations

**lemma** *image-mono*: $A \subseteq B \Longrightarrow f\ `\ A \subseteq f\ `\ B$
⟨*proof*⟩

**lemma** *Pow-mono*: $A \subseteq B \Longrightarrow Pow\ A \subseteq Pow\ B$
⟨*proof*⟩

**lemma** *insert-mono*: $C \subseteq D \Longrightarrow insert\ a\ C \subseteq insert\ a\ D$
⟨*proof*⟩

**lemma** *Un-mono*: $A \subseteq C \Longrightarrow B \subseteq D \Longrightarrow A \cup B \subseteq C \cup D$
⟨*proof*⟩

**lemma** *Int-mono*: $A \subseteq C \Longrightarrow B \subseteq D \Longrightarrow A \cap B \subseteq C \cap D$
⟨*proof*⟩

**lemma** *Diff-mono*: $A \subseteq C \Longrightarrow D \subseteq B \Longrightarrow A - B \subseteq C - D$
⟨*proof*⟩

**lemma** *Compl-anti-mono*: $A \subseteq B \Longrightarrow -\ B \subseteq -\ A$
⟨*proof*⟩

Monotonicity of implications.

**lemma** *in-mono*: $A \subseteq B \Longrightarrow x \in A \longrightarrow x \in B$
⟨*proof*⟩

**lemma** *conj-mono*: $P1 \longrightarrow Q1 \Longrightarrow P2 \longrightarrow Q2 \Longrightarrow (P1 \wedge P2) \longrightarrow (Q1 \wedge Q2)$
⟨*proof*⟩

**lemma** *disj-mono*: $P1 \longrightarrow Q1 \Longrightarrow P2 \longrightarrow Q2 \Longrightarrow (P1 \vee P2) \longrightarrow (Q1 \vee Q2)$
⟨*proof*⟩

**lemma** *imp-mono*: $Q1 \longrightarrow P1 \Longrightarrow P2 \longrightarrow Q2 \Longrightarrow (P1 \longrightarrow P2) \longrightarrow (Q1 \longrightarrow Q2)$
⟨*proof*⟩

**lemma** *imp-refl*: $P \longrightarrow P$ ⟨*proof*⟩

**lemma** *not-mono*: $Q \longrightarrow P \Longrightarrow \neg\ P \longrightarrow \neg\ Q$
⟨*proof*⟩

**lemma** *ex-mono*: $(\bigwedge x.\ P\ x \longrightarrow Q\ x) \Longrightarrow (\exists x.\ P\ x) \longrightarrow (\exists x.\ Q\ x)$
⟨*proof*⟩

**lemma** *all-mono*: $(\bigwedge x.\ P\ x \longrightarrow Q\ x) \Longrightarrow (\forall\, x.\ P\ x) \longrightarrow (\forall\, x.\ Q\ x)$
⟨*proof*⟩

**lemma** *Collect-mono*: $(\bigwedge x.\ P\ x \longrightarrow Q\ x) \Longrightarrow \mathit{Collect}\ P \subseteq \mathit{Collect}\ Q$
⟨*proof*⟩

**lemma** *Int-Collect-mono*: $A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow P\ x \longrightarrow Q\ x) \Longrightarrow A \cap \mathit{Collect}\ P \subseteq B \cap \mathit{Collect}\ Q$
⟨*proof*⟩

**lemmas** *basic-monos* =
*subset-refl imp-refl disj-mono conj-mono ex-mono Collect-mono in-mono*

**lemma** *eq-to-mono*: $a = b \Longrightarrow c = d \Longrightarrow b \longrightarrow d \Longrightarrow a \longrightarrow c$
⟨*proof*⟩

### 8.4.5 Inverse image of a function

**definition** *vimage* :: $('a \Rightarrow 'b) \Rightarrow 'b\ set \Rightarrow 'a\ set$ (**infixr** ‹−`› *90*)
**where** $f\ -`\ B \equiv \{x.\ f\ x \in B\}$

**lemma** *vimage-eq* [*simp*]: $a \in f\ -`\ B \longleftrightarrow f\ a \in B$
⟨*proof*⟩

**lemma** *vimage-singleton-eq*: $a \in f\ -`\ \{b\} \longleftrightarrow f\ a = b$
⟨*proof*⟩

**lemma** *vimageI* [*intro*]: $f\ a = b \Longrightarrow b \in B \Longrightarrow a \in f\ -`\ B$
⟨*proof*⟩

**lemma** *vimageI2*: $f\ a \in A \Longrightarrow a \in f\ -`\ A$
⟨*proof*⟩

**lemma** *vimageE* [*elim*!]: $a \in f\ -`\ B \Longrightarrow (\bigwedge x.\ f\ a = x \Longrightarrow x \in B \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *vimageD*: $a \in f\ -`\ A \Longrightarrow f\ a \in A$
⟨*proof*⟩

**lemma** *vimage-empty* [*simp*]: $f\ -`\ \{\} = \{\}$
⟨*proof*⟩

**lemma** *vimage-Compl*: $f\ -`\ (-\ A) = -\ (f\ -`\ A)$
⟨*proof*⟩

**lemma** *vimage-Un* [*simp*]: $f\ -`\ (A \cup B) = (f\ -`\ A) \cup (f\ -`\ B)$
⟨*proof*⟩

**lemma** *vimage-Int* [*simp*]: $f -` (A \cap B) = (f -` A) \cap (f -` B)$
⟨*proof*⟩

**lemma** *vimage-Collect-eq* [*simp*]: $f -` Collect\ P = \{y.\ P\ (f\ y)\}$
⟨*proof*⟩

**lemma** *vimage-Collect*: $(\bigwedge x.\ P\ (f\ x) = Q\ x) \Longrightarrow f -` (Collect\ P) = Collect\ Q$
⟨*proof*⟩

**lemma** *vimage-insert*: $f -` (insert\ a\ B) = (f -` \{a\}) \cup (f -` B)$
— NOT suitable for rewriting because of the recurrence of $\{a\}$.
⟨*proof*⟩

**lemma** *vimage-Diff*: $f -` (A - B) = (f -` A) - (f -` B)$
⟨*proof*⟩

**lemma** *vimage-UNIV* [*simp*]: $f -` UNIV = UNIV$
⟨*proof*⟩

**lemma** *vimage-mono*: $A \subseteq B \Longrightarrow f -` A \subseteq f -` B$
— monotonicity
⟨*proof*⟩

**lemma** *vimage-image-eq*: $f -` (f ` A) = \{y.\ \exists x \in A.\ f\ x = f\ y\}$
⟨*proof*⟩

**lemma** *image-vimage-subset*: $f ` (f -` A) \subseteq A$
⟨*proof*⟩

**lemma** *image-vimage-eq* [*simp*]: $f ` (f -` A) = A \cap range\ f$
⟨*proof*⟩

**lemma** *image-subset-iff-subset-vimage*: $f ` A \subseteq B \longleftrightarrow A \subseteq f -` B$
⟨*proof*⟩

**lemma** *subset-vimage-iff*: $A \subseteq f -` B \longleftrightarrow (\forall x \in A.\ f\ x \in B)$
⟨*proof*⟩

**lemma** *vimage-const* [*simp*]: $((\lambda x.\ c) -` A) = (if\ c \in A\ then\ UNIV\ else\ \{\})$
⟨*proof*⟩

**lemma** *vimage-if* [*simp*]: $((\lambda x.\ if\ x \in B\ then\ c\ else\ d) -` A) =$
$(if\ c \in A\ then\ (if\ d \in A\ then\ UNIV\ else\ B)$
$else\ if\ d \in A\ then\ -\ B\ else\ \{\})$
⟨*proof*⟩

**lemma** *vimage-inter-cong*: $(\bigwedge\ w.\ w \in S \Longrightarrow f\ w = g\ w) \Longrightarrow f -` y \cap S = g -`$
$y \cap S$
⟨*proof*⟩

**lemma** *vimage-ident* [*simp*]: $(\lambda x.\ x) -` Y = Y$
⟨*proof*⟩

### 8.4.6 Singleton sets

**definition** *is-singleton* :: $'a\ set \Rightarrow bool$
**where** *is-singleton* $A \longleftrightarrow (\exists x.\ A = \{x\})$

**lemma** *is-singletonI* [*simp*, *intro*!]: *is-singleton* $\{x\}$
⟨*proof*⟩

**lemma** *is-singletonI'*: $A \neq \{\} \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x = y) \Longrightarrow$ *is-singleton* $A$
⟨*proof*⟩

**lemma** *is-singletonE*: *is-singleton* $A \Longrightarrow (\bigwedge x.\ A = \{x\} \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *is-singleton-iff-ex1*:
‹*is-singleton* $A \longleftrightarrow (\exists!x.\ x \in A)$›
⟨*proof*⟩

### 8.4.7 Getting the contents of a singleton set

**definition** *the-elem* :: $'a\ set \Rightarrow 'a$
**where** *the-elem* $X = (THE\ x.\ X = \{x\})$

**lemma** *the-elem-eq* [*simp*]: *the-elem* $\{x\} = x$
⟨*proof*⟩

**lemma** *is-singleton-the-elem*: *is-singleton* $A \longleftrightarrow A = \{$*the-elem* $A\}$
⟨*proof*⟩

**lemma** *the-elem-image-unique*:
**assumes** $A \neq \{\}$
**and** $*$: $\bigwedge y.\ y \in A \Longrightarrow f\ y = a$
**shows** *the-elem* $(f\ `\ A) = a$
⟨*proof*⟩

### 8.4.8 Monad operation

**definition** *bind* :: $'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow 'b\ set$
**where** *bind* $A\ f = \{x.\ \exists B \in f`A.\ x \in B\}$

**hide-const** (**open**) *bind*

**lemma** *bind-bind*: $Set.bind\ (Set.bind\ A\ B)\ C = Set.bind\ A\ (\lambda x.\ Set.bind\ (B\ x)\ C)$
**for** $A :: 'a\ set$

⟨*proof*⟩

**lemma** *empty-bind* [*simp*]: $Set.bind\ \{\}\ f = \{\}$
⟨*proof*⟩

**lemma** *nonempty-bind-const*: $A \neq \{\} \Longrightarrow Set.bind\ A\ (\lambda\text{-}.\ B) = B$
⟨*proof*⟩

**lemma** *bind-const*: $Set.bind\ A\ (\lambda\text{-}.\ B) = (\mathit{if}\ A = \{\}\ \mathit{then}\ \{\}\ \mathit{else}\ B)$
⟨*proof*⟩

**lemma** *bind-singleton-conv-image*: $Set.bind\ A\ (\lambda x.\ \{f\ x\}) = f\ `\ A$
⟨*proof*⟩

### 8.4.9 Operations for execution

Use those operations only for generating executable / efficient code. Otherwise use the RHSs directly.

**context**
**begin**

**qualified definition** *is-empty* :: $'a\ set \Rightarrow bool$ — only for code generation
**where** *is-empty-iff* [*code-abbrev*, *simp*]: $is\text{-}empty\ A \longleftrightarrow A = \{\}$

**qualified definition** *remove* :: $'a \Rightarrow 'a\ set \Rightarrow 'a\ set$ — only for code generation
**where** *remove-eq* [*code-abbrev*, *simp*]: $remove\ x\ A = A - \{x\}$

**qualified definition** *filter* :: $('a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a\ set$ — only for code generation
**where** *filter-eq* [*code-abbrev*, *simp*]: $filter\ P\ A = \{a \in A.\ P\ a\}$

**qualified definition** *can-select* :: $('a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$ — only for code generation
**where** *can-select-iff* [*code-abbrev*, *simp*]: $can\text{-}select\ P\ A = (\exists!x{\in}A.\ P\ x)$

**qualified lemma** *can-select-iff-is-singleton*:
‹$Set.can\text{-}select\ P\ A \longleftrightarrow is\text{-}singleton\ (Set.filter\ P\ A)$›
⟨*proof*⟩

**end**

**instantiation** *set* :: (*equal*) *equal*
**begin**

**definition** $HOL.equal\ A\ B \longleftrightarrow A \subseteq B \wedge B \subseteq A$

**instance**
⟨*proof*⟩

**end**

Misc

**definition** *pairwise* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
**where** $pairwise\ R\ S \longleftrightarrow (\forall x \in S.\ \forall y \in S.\ x \neq y \longrightarrow R\ x\ y)$

**lemma** *pairwise-alt*: $pairwise\ R\ S \longleftrightarrow (\forall x{\in}S.\ \forall y{\in}S{-}\{x\}.\ R\ x\ y)$
⟨*proof*⟩

**lemma** *pairwise-trivial* [*simp*]: $pairwise\ (\lambda i\ j.\ j \neq i)\ I$
⟨*proof*⟩

**lemma** *pairwiseI* [*intro?*]:
$pairwise\ R\ S$ **if** $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow x \neq y \Longrightarrow R\ x\ y$
⟨*proof*⟩

**lemma** *pairwiseD*:
$R\ x\ y$ **and** $R\ y\ x$
**if** $pairwise\ R\ S\ x \in S$ **and** $y \in S$ **and** $x \neq y$
⟨*proof*⟩

**lemma** *pairwise-empty* [*simp*]: $pairwise\ P\ \{\}$
⟨*proof*⟩

**lemma** *pairwise-singleton* [*simp*]: $pairwise\ P\ \{A\}$
⟨*proof*⟩

**lemma** *pairwise-insert*:
$pairwise\ r\ (insert\ x\ s) \longleftrightarrow (\forall y.\ y \in s \wedge y \neq x \longrightarrow r\ x\ y \wedge r\ y\ x) \wedge pairwise\ r\ s$
⟨*proof*⟩

**lemma** *pairwise-subset*: $pairwise\ P\ S \Longrightarrow T \subseteq S \Longrightarrow pairwise\ P\ T$
⟨*proof*⟩

**lemma** *pairwise-mono*: $[\![pairwise\ P\ A;\ \bigwedge x\ y.\ P\ x\ y \Longrightarrow Q\ x\ y;\ B \subseteq A]\!] \Longrightarrow pairwise\ Q\ B$
⟨*proof*⟩

**lemma** *pairwise-imageI*:
$pairwise\ P\ (f\ `\ A)$
**if** $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \Longrightarrow f\ x \neq f\ y \Longrightarrow P\ (f\ x)\ (f\ y)$
⟨*proof*⟩

**lemma** *pairwise-image*: $pairwise\ r\ (f\ `\ s) \longleftrightarrow pairwise\ (\lambda x\ y.\ (f\ x \neq f\ y) \longrightarrow r\ (f\ x)\ (f\ y))\ s$
⟨*proof*⟩

**definition** *disjnt* :: $'a\ set \Rightarrow 'a\ set \Rightarrow bool$
**where** $disjnt\ A\ B \longleftrightarrow A \cap B = \{\}$

**lemma** *disjnt-self-iff-empty* [*simp*]: *disjnt* $S$ $S \longleftrightarrow S = \{\}$
⟨*proof*⟩

**lemma** *disjnt-commute*: *disjnt* $A$ $B$ = *disjnt* $B$ $A$
⟨*proof*⟩

**lemma** *disjnt-iff*: *disjnt* $A$ $B \longleftrightarrow (\forall x. \neg (x \in A \wedge x \in B))$
⟨*proof*⟩

**lemma** *disjnt-sym*: *disjnt* $A$ $B \Longrightarrow$ *disjnt* $B$ $A$
⟨*proof*⟩

**lemma** *disjnt-empty1* [*simp*]: *disjnt* $\{\}$ $A$ **and** *disjnt-empty2* [*simp*]: *disjnt* $A$ $\{\}$
⟨*proof*⟩

**lemma** *disjnt-insert1* [*simp*]: *disjnt* (*insert* $a$ $X$) $Y \longleftrightarrow a \notin Y \wedge$ *disjnt* $X$ $Y$
⟨*proof*⟩

**lemma** *disjnt-insert2* [*simp*]: *disjnt* $Y$ (*insert* $a$ $X$) $\longleftrightarrow a \notin Y \wedge$ *disjnt* $Y$ $X$
⟨*proof*⟩

**lemma** *disjnt-subset1* : ⟦*disjnt* $X$ $Y$; $Z \subseteq X$⟧ $\Longrightarrow$ *disjnt* $Z$ $Y$
⟨*proof*⟩

**lemma** *disjnt-subset2* : ⟦*disjnt* $X$ $Y$; $Z \subseteq Y$⟧ $\Longrightarrow$ *disjnt* $X$ $Z$
⟨*proof*⟩

**lemma** *disjnt-Un1* [*simp*]: *disjnt* $(A \cup B)$ $C \longleftrightarrow$ *disjnt* $A$ $C \wedge$ *disjnt* $B$ $C$
⟨*proof*⟩

**lemma** *disjnt-Un2* [*simp*]: *disjnt* $C$ $(A \cup B) \longleftrightarrow$ *disjnt* $C$ $A \wedge$ *disjnt* $C$ $B$
⟨*proof*⟩

**lemma** *disjnt-Diff1*: *disjnt* $(X - Y)$ $(U - V)$ **and** *disjnt-Diff2*: *disjnt* $(U - V)$ $(X - Y)$
**if** $X \subseteq V$
⟨*proof*⟩

**lemma** *disjoint-image-subset*: ⟦*pairwise disjnt* $\mathcal{A}$; $\bigwedge X.\ X \in \mathcal{A} \Longrightarrow f\ X \subseteq X$⟧ $\Longrightarrow$
*pairwise disjnt* ($f$ ‘$\mathcal{A}$)
⟨*proof*⟩

**lemma** *pairwise-disjnt-iff*: *pairwise disjnt* $\mathcal{A} \longleftrightarrow (\forall x. \exists_{\leq 1}\ X.\ X \in \mathcal{A} \wedge x \in X)$
⟨*proof*⟩

**lemma** *disjnt-insert*:
‹*disjnt* (*insert* $x$ $M$) $N$› **if** ‹$x \notin N$› ‹*disjnt* $M$ $N$›
⟨*proof*⟩

**lemma** *Int-emptyI*: $(\bigwedge x.\ x \in A \Longrightarrow x \in B \Longrightarrow \mathit{False}) \Longrightarrow A \cap B = \{\}$
⟨*proof*⟩

**lemma** *in-image-insert-iff*:
**assumes** $\bigwedge C.\ C \in B \Longrightarrow x \notin C$
**shows** $A \in \mathit{insert}\ x\ `\ B \longleftrightarrow x \in A \wedge A - \{x\} \in B$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**hide-const** (**open**) *member not-member*

**lemmas** *equalityI* = *subset-antisym*
**lemmas** *set-mp* = *subsetD*
**lemmas** *set-rev-mp* = *rev-subsetD*

**end**

# 9 HOL type definitions

**theory** *Typedef*
**imports** *Set*
**keywords**
*typedef* :: *thy-goal-defn* **and**
*morphisms* :: *quasi-command*
**begin**

**locale** *type-definition* =
**fixes** *Rep* **and** *Abs* **and** *A*
**assumes** *Rep*: $\mathit{Rep}\ x \in A$
**and** *Rep-inverse*: $\mathit{Abs}\ (\mathit{Rep}\ x) = x$
**and** *Abs-inverse*: $y \in A \Longrightarrow \mathit{Rep}\ (\mathit{Abs}\ y) = y$
— This will be axiomatized for each typedef!
**begin**

**lemma** *Rep-inject*: $\mathit{Rep}\ x = \mathit{Rep}\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *Abs-inject*:
**assumes** $x \in A$ **and** $y \in A$
**shows** $\mathit{Abs}\ x = \mathit{Abs}\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *Rep-cases* [*cases set*]:
**assumes** $y \in A$
**and** *hyp*: $\bigwedge x.\ y = \mathit{Rep}\ x \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *Abs-cases* [*cases type*]:
**assumes** *r*: $\bigwedge y.\ x = \mathit{Abs}\ y \Longrightarrow y \in A \Longrightarrow P$

**shows** $P$
⟨*proof*⟩

**lemma** *Rep-induct* [*induct set*]:
**assumes** $y$: $y \in A$
**and** *hyp*: $\bigwedge x.\ P\ (Rep\ x)$
**shows** $P\ y$
⟨*proof*⟩

**lemma** *Abs-induct* [*induct type*]:
**assumes** $r$: $\bigwedge y.\ y \in A \Longrightarrow P\ (Abs\ y)$
**shows** $P\ x$
⟨*proof*⟩

**lemma** *Rep-range*: $range\ Rep = A$
⟨*proof*⟩

**lemma** *Abs-image*: $Abs\ `\ A = UNIV$
⟨*proof*⟩

**end**

⟨*ML*⟩

**end**

# 10 Notions about functions

**theory** *Fun*
**imports** *Set*
**keywords** *functor* :: *thy-goal-defn*
**begin**

**lemma** *apply-inverse*: $f\ x = u \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow g\ (f\ x) = x) \Longrightarrow P\ x \Longrightarrow x = g\ u$
⟨*proof*⟩

Uniqueness, so NOT the axiom of choice.

**lemma** *uniq-choice*: $\forall x.\ \exists!y.\ Q\ x\ y \Longrightarrow \exists f.\ \forall x.\ Q\ x\ (f\ x)$
⟨*proof*⟩

**lemma** *b-uniq-choice*: $\forall x{\in}S.\ \exists!y.\ Q\ x\ y \Longrightarrow \exists f.\ \forall x{\in}S.\ Q\ x\ (f\ x)$
⟨*proof*⟩

## 10.1 The Identity Function *id*

**definition** *id* :: $'a \Rightarrow 'a$
**where** $id = (\lambda x.\ x)$

**lemma** *id-apply* [*simp*]: $id\ x = x$
⟨*proof*⟩

**lemma** *image-id* [*simp*]: $image\ id = id$
⟨*proof*⟩

**lemma** *vimage-id* [*simp*]: $vimage\ id = id$
⟨*proof*⟩

**lemma** *eq-id-iff*: $(\forall x.\ f\ x = x) \longleftrightarrow f = id$
⟨*proof*⟩

**code-printing**
**constant** *id* ⇀ (*Haskell*) *id*

## 10.2 The Composition Operator $f \circ g$

**definition** $comp :: ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'c$ (**infixl** ‹∘› *55*)
**where** $f \circ g = (\lambda x.\ f\ (g\ x))$

**notation** (*ASCII*)
*comp* (**infixl** ‹*o*› *55*)

**lemma** *comp-apply* [*simp*]: $(f \circ g)\ x = f\ (g\ x)$
⟨*proof*⟩

**lemma** *comp-assoc*: $(f \circ g) \circ h = f \circ (g \circ h)$
⟨*proof*⟩

**lemma** *id-comp* [*simp*]: $id \circ g = g$
⟨*proof*⟩

**lemma** *comp-id* [*simp*]: $f \circ id = f$
⟨*proof*⟩

**lemma** *comp-eq-dest*: $a \circ b = c \circ d \Longrightarrow a\ (b\ v) = c\ (d\ v)$
⟨*proof*⟩

**lemma** *comp-eq-elim*: $a \circ b = c \circ d \Longrightarrow ((\bigwedge v.\ a\ (b\ v) = c\ (d\ v)) \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *comp-eq-dest-lhs*: $a \circ b = c \Longrightarrow a\ (b\ v) = c\ v$
⟨*proof*⟩

**lemma** *comp-eq-id-dest*: $a \circ b = id \circ c \Longrightarrow a\ (b\ v) = c\ v$
⟨*proof*⟩

**lemma** *image-comp*: $f\ `\ (g\ `\ r) = (f \circ g)\ `\ r$
⟨*proof*⟩

**lemma** *vimage-comp*: $f -` (g -` x) = (g \circ f) -` x$
⟨*proof*⟩

**lemma** *image-eq-imp-comp*: $f ` A = g ` B \Longrightarrow (h \circ f) ` A = (h \circ g) ` B$
⟨*proof*⟩

**lemma** *image-bind*: $f ` (Set.bind\ A\ g) = Set.bind\ A\ ((`)\ f \circ g)$
⟨*proof*⟩

**lemma** *bind-image*: $Set.bind\ (f ` A)\ g = Set.bind\ A\ (g \circ f)$
⟨*proof*⟩

**lemma** (**in** *group-add*) *minus-comp-minus* [*simp*]: *uminus* ∘ *uminus* = *id*
⟨*proof*⟩

**lemma** (**in** *boolean-algebra*) *minus-comp-minus* [*simp*]: *uminus* ∘ *uminus* = *id*
⟨*proof*⟩

**code-printing**
**constant** *comp* ⇀ (*SML*) **infixl** *5 o* **and** (*Haskell*) **infixr** *9* .

## 10.3 The Forward Composition Operator *fcomp*

**definition** *fcomp* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'c$ (**infixl** ‹∘>› *60*)
**where** $f \circ\!> g = (\lambda x.\ g\ (f\ x))$

**lemma** *fcomp-apply* [*simp*]: $(f \circ\!> g)\ x = g\ (f\ x)$
⟨*proof*⟩

**lemma** *fcomp-assoc*: $(f \circ\!> g) \circ\!> h = f \circ\!> (g \circ\!> h)$
⟨*proof*⟩

**lemma** *id-fcomp* [*simp*]: $id \circ\!> g = g$
⟨*proof*⟩

**lemma** *fcomp-id* [*simp*]: $f \circ\!> id = f$
⟨*proof*⟩

**lemma** *fcomp-comp*: *fcomp f g* = *comp g f*
⟨*proof*⟩

**code-printing**
**constant** *fcomp* ⇀ (*Eval*) **infixl** *1* #>

**no-notation** *fcomp* (**infixl** ‹∘>› *60*)

## 10.4 Mapping functions

**definition** *map-fun* :: $('c \Rightarrow 'a) \Rightarrow ('b \Rightarrow 'd) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'c \Rightarrow 'd$

**where** $map\text{-}fun\ f\ g\ h = g \circ h \circ f$

**lemma** *map-fun-apply* [*simp*]: $map\text{-}fun\ f\ g\ h\ x = g\ (h\ (f\ x))$
⟨*proof*⟩

## 10.5 Injectivity and Bijectivity

**definition** $inj\text{-}on :: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow bool$ — injective
**where** $inj\text{-}on\ f\ A \longleftrightarrow (\forall\, x{\in}A.\ \forall\, y{\in}A.\ f\ x = f\ y \longrightarrow x = y)$

**definition** $bij\text{-}betw :: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow bool$ — bijective
**where** $bij\text{-}betw\ f\ A\ B \longleftrightarrow inj\text{-}on\ f\ A \wedge f\ `\ A = B$

A common special case: functions injective, surjective or bijective over the entire domain type.

**abbreviation** $inj :: ('a \Rightarrow 'b) \Rightarrow bool$
**where** $inj\ f \equiv inj\text{-}on\ f\ UNIV$

**abbreviation** $surj :: ('a \Rightarrow 'b) \Rightarrow bool$
**where** $surj\ f \equiv range\ f = UNIV$

**translations** — The negated case:
$\neg\ CONST\ surj\ f \leftharpoondown CONST\ range\ f \neq CONST\ UNIV$

**abbreviation** $bij :: ('a \Rightarrow 'b) \Rightarrow bool$
**where** $bij\ f \equiv bij\text{-}betw\ f\ UNIV\ UNIV$

**lemma** *inj-def*: $inj\ f \longleftrightarrow (\forall\, x\ y.\ f\ x = f\ y \longrightarrow x = y)$
⟨*proof*⟩

**lemma** *injI*: $(\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y) \Longrightarrow inj\ f$
⟨*proof*⟩

**theorem** *range-ex1-eq*: $inj\ f \Longrightarrow b \in range\ f \longleftrightarrow (\exists!x.\ b = f\ x)$
⟨*proof*⟩

**lemma** *injD*: $inj\ f \Longrightarrow f\ x = f\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-on-eq-iff*: $inj\text{-}on\ f\ A \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow f\ x = f\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-on-cong*: $(\bigwedge a.\ a \in A \Longrightarrow f\ a = g\ a) \Longrightarrow inj\text{-}on\ f\ A \longleftrightarrow inj\text{-}on\ g\ A$
⟨*proof*⟩

**lemma** *image-strict-mono*: $inj\text{-}on\ f\ B \Longrightarrow A \subset B \Longrightarrow f\ `\ A \subset f\ `\ B$
⟨*proof*⟩

**lemma** *inj-compose*: $inj\ f \Longrightarrow inj\ g \Longrightarrow inj\ (f \circ g)$

⟨*proof*⟩

**lemma** *inj-fun*: $inj\ f \Longrightarrow inj\ (\lambda x\ y.\ f\ x)$
⟨*proof*⟩

**lemma** *inj-eq*: $inj\ f \Longrightarrow f\ x = f\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-on-iff-Uniq*: $inj\text{-}on\ f\ A \longleftrightarrow (\forall\, x{\in}A.\ \exists_{\leq 1} y.\ y{\in}A \wedge f\ x = f\ y)$
⟨*proof*⟩

**lemma** *inj-on-id*[*simp*]: $inj\text{-}on\ id\ A$
⟨*proof*⟩

**lemma** *inj-on-id2*[*simp*]: $inj\text{-}on\ (\lambda x.\ x)\ A$
⟨*proof*⟩

**lemma** *inj-on-Int*: $inj\text{-}on\ f\ A \vee inj\text{-}on\ f\ B \Longrightarrow inj\text{-}on\ f\ (A \cap B)$
⟨*proof*⟩

**lemma** *surj-id*: $surj\ id$
⟨*proof*⟩

**lemma** *bij-id*[*simp*]: $bij\ id$
⟨*proof*⟩

**lemma** *bij-uminus*: $bij\ (uminus :: 'a \Rightarrow 'a{::}group\text{-}add)$
⟨*proof*⟩

**lemma** *bij-betwE*: $bij\text{-}betw\ f\ A\ B \Longrightarrow \forall\, a{\in}A.\ f\ a \in B$
⟨*proof*⟩

**lemma** *inj-onI* [*intro?*]: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow f\ x = f\ y \Longrightarrow x = y) \Longrightarrow inj\text{-}on\ f\ A$
⟨*proof*⟩

For those frequent proofs by contradiction

**lemma** *inj-onCI*: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow f\ x = f\ y \Longrightarrow x \neq y \Longrightarrow False) \Longrightarrow inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *inj-on-inverseI*: $(\bigwedge x.\ x \in A \Longrightarrow g\ (f\ x) = x) \Longrightarrow inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *inj-onD*: $inj\text{-}on\ f\ A \Longrightarrow f\ x = f\ y \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-on-subset*:
$[\![\ inj\text{-}on\ f\ A;\ B \subseteq A\ ]\!] \Longrightarrow inj\text{-}on\ f\ B$

⟨*proof*⟩

**lemma** *comp-inj-on*: *inj-on f A* ⟹ *inj-on g* (*f* ‘ *A*) ⟹ *inj-on* (*g* ∘ *f*) *A*
⟨*proof*⟩

**lemma** *inj-on-imageI*: *inj-on* (*g* ∘ *f*) *A* ⟹ *inj-on g* (*f* ‘ *A*)
⟨*proof*⟩

**lemma** *inj-on-image-iff*:
∀ *x*∈*A*. ∀ *y*∈*A*. *g* (*f x*) = *g* (*f y*) ⟷ *g x* = *g y* ⟹ *inj-on f A* ⟹ *inj-on g* (*f* ‘ *A*) ⟷ *inj-on g A*
⟨*proof*⟩

**lemma** *inj-on-contraD*: *inj-on f A* ⟹ *x* ≠ *y* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *f x* ≠ *f y*
⟨*proof*⟩

**lemma** *inj-singleton* [*simp*]: *inj-on* (λ*x*. {*x*}) *A*
⟨*proof*⟩

**lemma** *inj-on-empty*[*iff*]: *inj-on f* {}
⟨*proof*⟩

**lemma** *inj-on-Un*: *inj-on f* (*A* ∪ *B*) ⟷ *inj-on f A* ∧ *inj-on f B* ∧ *f* ‘ (*A* − *B*) ∩ *f* ‘ (*B* − *A*) = {}
⟨*proof*⟩

**lemma** *inj-on-insert* [*iff*]: *inj-on f* (*insert a A*) ⟷ *inj-on f A* ∧ *f a* ∉ *f* ‘ (*A* − {*a*})
⟨*proof*⟩

**lemma** *inj-on-diff*: *inj-on f A* ⟹ *inj-on f* (*A* − *B*)
⟨*proof*⟩

**lemma** *comp-inj-on-iff*: *inj-on f A* ⟹ *inj-on f′* (*f* ‘ *A*) ⟷ *inj-on* (*f′* ∘ *f*) *A*
⟨*proof*⟩

**lemma** *inj-on-imageI2*: *inj-on* (*f′* ∘ *f*) *A* ⟹ *inj-on f A*
⟨*proof*⟩

**lemma** *inj-img-insertE*:
**assumes** *inj-on f A*
**assumes** *x* ∉ *B*
**and** *insert x B* = *f* ‘ *A*
**obtains** *x′ A′* **where** *x′* ∉ *A′* **and** *A* = *insert x′ A′* **and** *x* = *f x′* **and** *B* = *f* ‘ *A′*
⟨*proof*⟩

**lemma** *linorder-inj-onI*:
**fixes** *A* :: ′*a*::*order set*

**assumes** *ne*: $\bigwedge x\ y.\ [\![x < y;\ x{\in}A;\ y{\in}A]\!] \Longrightarrow f\ x \neq f\ y$ **and** *lin*: $\bigwedge x\ y.\ [\![x{\in}A;\ y{\in}A]\!] \Longrightarrow x{\leq}y \vee y{\leq}x$
**shows** *inj-on f A*
⟨*proof*⟩

**lemma** *linorder-inj-onI'*:
**fixes** $A :: 'a :: linorder\ set$
**assumes** $\bigwedge i\ j.\ i \in A \Longrightarrow j \in A \Longrightarrow i < j \Longrightarrow f\ i \neq f\ j$
**shows** *inj-on f A*
⟨*proof*⟩

**lemma** *linorder-injI*:
**assumes** $\bigwedge x\ y{::}'a{::}linorder.\ x < y \Longrightarrow f\ x \neq f\ y$
**shows** *inj f*
— Courtesy of Stephan Merz
⟨*proof*⟩

**lemma** *inj-on-image-Pow*: *inj-on f A* $\Longrightarrow$ *inj-on* (*image f*) (*Pow A*)
⟨*proof*⟩

**lemma** *inj-on-vimage-image*: *inj-on* $(\lambda b.\ f -` \{b\})$ $(f\ `\ A)$
⟨*proof*⟩

**lemma** *bij-betw-image-Pow*: *bij-betw f A B* $\Longrightarrow$ *bij-betw* (*image f*) (*Pow A*) (*Pow B*)
⟨*proof*⟩

**lemma** *surj-def*: *surj f* $\longleftrightarrow (\forall y.\ \exists x.\ y = f\ x)$
⟨*proof*⟩

**lemma** *surjI*:
**assumes** $\bigwedge x.\ g\ (f\ x) = x$
**shows** *surj g*
⟨*proof*⟩

**lemma** *surjD*: *surj f* $\Longrightarrow \exists x.\ y = f\ x$
⟨*proof*⟩

**lemma** *surjE*: *surj f* $\Longrightarrow (\bigwedge x.\ y = f\ x \Longrightarrow C) \Longrightarrow C$
⟨*proof*⟩

**lemma** *comp-surj*: *surj f* $\Longrightarrow$ *surj g* $\Longrightarrow$ *surj* $(g \circ f)$
⟨*proof*⟩

**lemma** *bij-betw-imageI*: *inj-on f A* $\Longrightarrow f\ `\ A = B \Longrightarrow$ *bij-betw f A B*
⟨*proof*⟩

**lemma** *bij-betw-imp-surj-on*: *bij-betw f A B* $\Longrightarrow f\ `\ A = B$
⟨*proof*⟩

**lemma** *bij-betw-imp-surj*: $bij\text{-}betw\ f\ A\ UNIV \Longrightarrow surj\ f$
⟨*proof*⟩

**lemma** *bij-betw-empty1*: $bij\text{-}betw\ f\ \{\}\ A \Longrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *bij-betw-empty2*: $bij\text{-}betw\ f\ A\ \{\} \Longrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *inj-on-imp-bij-betw*: $inj\text{-}on\ f\ A \Longrightarrow bij\text{-}betw\ f\ A\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *bij-betw-DiffI*:
**assumes** $bij\text{-}betw\ f\ A\ B\ bij\text{-}betw\ f\ C\ D\ C \subseteq A\ D \subseteq B$
**shows** $bij\text{-}betw\ f\ (A - C)\ (B - D)$
⟨*proof*⟩

**lemma** *bij-betw-singleton-iff* [*simp*]: $bij\text{-}betw\ f\ \{x\}\ \{y\} \longleftrightarrow f\ x = y$
⟨*proof*⟩

**lemma** *bij-betw-singletonI* [*intro*]: $f\ x = y \Longrightarrow bij\text{-}betw\ f\ \{x\}\ \{y\}$
⟨*proof*⟩

**lemma** *bij-betw-imp-empty-iff*: $bij\text{-}betw\ f\ A\ B \Longrightarrow A = \{\} \longleftrightarrow B = \{\}$
⟨*proof*⟩

**lemma** *bij-betw-imp-Ex-iff*: $bij\text{-}betw\ f\ \{x.\ P\ x\}\ \{x.\ Q\ x\} \Longrightarrow (\exists\, x.\ P\ x) \longleftrightarrow (\exists\, x.\ Q\ x)$
⟨*proof*⟩

**lemma** *bij-betw-imp-Bex-iff*: $bij\text{-}betw\ f\ \{x{\in}A.\ P\ x\}\ \{x{\in}B.\ Q\ x\} \Longrightarrow (\exists\, x{\in}A.\ P\ x) \longleftrightarrow (\exists\, x{\in}B.\ Q\ x)$
⟨*proof*⟩

**lemma** *bij-betw-apply*: $[\![bij\text{-}betw\ f\ A\ B;\ a \in A]\!] \Longrightarrow f\ a \in B$
⟨*proof*⟩

**lemma** *bij-def*: $bij\ f \longleftrightarrow inj\ f \wedge surj\ f$
⟨*proof*⟩

**lemma** *bijI*: $inj\ f \Longrightarrow surj\ f \Longrightarrow bij\ f$
⟨*proof*⟩

**lemma** *bij-is-inj*: $bij\ f \Longrightarrow inj\ f$
⟨*proof*⟩

**lemma** *bij-is-surj*: $bij\ f \Longrightarrow surj\ f$
⟨*proof*⟩

**lemma** *bij-betw-imp-inj-on*: *bij-betw f A B* $\Longrightarrow$ *inj-on f A*
⟨*proof*⟩

**lemma** *bij-betw-trans*: *bij-betw f A B* $\Longrightarrow$ *bij-betw g B C* $\Longrightarrow$ *bij-betw* $(g \circ f)$ *A C*
⟨*proof*⟩

**lemma** *bij-comp*: *bij f* $\Longrightarrow$ *bij g* $\Longrightarrow$ *bij* $(g \circ f)$
⟨*proof*⟩

**lemma** *bij-betw-comp-iff*: *bij-betw f A* $A'$ $\Longrightarrow$ *bij-betw* $f'$ $A'$ $A''$ $\longleftrightarrow$ *bij-betw* $(f' \circ f)$ *A* $A''$
⟨*proof*⟩

**lemma** *bij-betw-Collect*:
**assumes** *bij-betw f A B* $\bigwedge x.\ x \in A \Longrightarrow Q\ (f\ x) \longleftrightarrow P\ x$
**shows** *bij-betw f* $\{x{\in}A.\ P\ x\}$ $\{y{\in}B.\ Q\ y\}$
⟨*proof*⟩

**lemma** *bij-betw-comp-iff2*:
**assumes** *bij*: *bij-betw* $f'$ $A'$ $A''$
**and** *img*: $f$ ‘ $A \leq A'$
**shows** *bij-betw f A* $A'$ $\longleftrightarrow$ *bij-betw* $(f' \circ f)$ *A* $A''$ (**is** *?L* $\longleftrightarrow$ *?R*)
⟨*proof*⟩

**lemma** *bij-betw-inv*:
**assumes** *bij-betw f A B*
**shows** $\exists\, g.$ *bij-betw g B A*
⟨*proof*⟩

**lemma** *bij-betw-cong*: $(\bigwedge a.\ a \in A \Longrightarrow f\ a = g\ a) \Longrightarrow$ *bij-betw f A* $A'$ = *bij-betw g A* $A'$
⟨*proof*⟩

**lemma** *bij-betw-id*[*intro*, *simp*]: *bij-betw id A A*
⟨*proof*⟩

**lemma** *bij-betw-id-iff*: *bij-betw id A B* $\longleftrightarrow$ $A = B$
⟨*proof*⟩

**lemma** *bij-betw-combine*:
*bij-betw f A B* $\Longrightarrow$ *bij-betw f C D* $\Longrightarrow$ $B \cap D = \{\}$ $\Longrightarrow$ *bij-betw f* $(A \cup C)$ $(B \cup D)$
⟨*proof*⟩

**lemma** *bij-betw-subset*: *bij-betw f A* $A'$ $\Longrightarrow$ $B \subseteq A$ $\Longrightarrow$ $f$ ‘ $B = B'$ $\Longrightarrow$ *bij-betw f B* $B'$
⟨*proof*⟩

**lemma** *bij-betw-ball*: $bij\text{-}betw\ f\ A\ B \Longrightarrow (\forall\, b \in B.\ phi\ b) = (\forall\, a \in A.\ phi\ (f\ a))$
⟨*proof*⟩

**lemma** *bij-pointE*:
**assumes** *bij f*
**obtains** $x$ **where** $y = f\ x$ **and** $\bigwedge x'.\ y = f\ x' \Longrightarrow x' = x$
⟨*proof*⟩

**lemma** *bij-iff*:
‹$bij\ f \longleftrightarrow (\forall\, x.\ \exists !y.\ f\ y = x)$› (**is** ‹$?P \longleftrightarrow ?Q$›)
⟨*proof*⟩

**lemma** *bij-betw-partition*:
‹$bij\text{-}betw\ f\ A\ B$›
**if** ‹$bij\text{-}betw\ f\ (A \cup C)\ (B \cup D)$› ‹$bij\text{-}betw\ f\ C\ D$› ‹$A \cap C = \{\}$› ‹$B \cap D = \{\}$›
⟨*proof*⟩

**lemma** *surj-image-vimage-eq*: $surj\ f \Longrightarrow f\ `\ (f\ -`\ A) = A$
⟨*proof*⟩

**lemma** *surj-vimage-empty*:
**assumes** *surj f*
**shows** $f\ -`\ A = \{\} \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *inj-vimage-image-eq*: $inj\ f \Longrightarrow f\ -`\ (f\ `\ A) = A$
⟨*proof*⟩

**lemma** *vimage-subsetD*: $surj\ f \Longrightarrow f\ -`\ B \subseteq A \Longrightarrow B \subseteq f\ `\ A$
⟨*proof*⟩

**lemma** *vimage-subsetI*: $inj\ f \Longrightarrow B \subseteq f\ `\ A \Longrightarrow f\ -`\ B \subseteq A$
⟨*proof*⟩

**lemma** *vimage-subset-eq*: $bij\ f \Longrightarrow f\ -`\ B \subseteq A \longleftrightarrow B \subseteq f\ `\ A$
⟨*proof*⟩

**lemma** *inj-on-image-eq-iff*: $inj\text{-}on\ f\ C \Longrightarrow A \subseteq C \Longrightarrow B \subseteq C \Longrightarrow f\ `\ A = f\ `\ B \longleftrightarrow A = B$
⟨*proof*⟩

**lemma** *inj-on-Un-image-eq-iff*: $inj\text{-}on\ f\ (A \cup B) \Longrightarrow f\ `\ A = f\ `\ B \longleftrightarrow A = B$
⟨*proof*⟩

**lemma** *inj-on-image-Int*: $inj\text{-}on\ f\ C \Longrightarrow A \subseteq C \Longrightarrow B \subseteq C \Longrightarrow f\ `\ (A \cap B) = f\ `\ A \cap f\ `\ B$
⟨*proof*⟩

**lemma** *inj-on-image-set-diff*: $inj\text{-}on\ f\ C \Longrightarrow A - B \subseteq C \Longrightarrow B \subseteq C \Longrightarrow f\ `\ (A$

$- B) = f\ `\ A - f\ `\ B$
⟨*proof*⟩

**lemma** *image-Int*: $inj\ f \Longrightarrow f\ `\ (A \cap B) = f\ `\ A \cap f\ `\ B$
⟨*proof*⟩

**lemma** *image-set-diff*: $inj\ f \Longrightarrow f\ `\ (A - B) = f\ `\ A - f\ `\ B$
⟨*proof*⟩

**lemma** *inj-on-image-mem-iff*: $inj\text{-}on\ f\ B \Longrightarrow a \in B \Longrightarrow A \subseteq B \Longrightarrow f\ a \in f\ `\ A \longleftrightarrow a \in A$
⟨*proof*⟩

**lemma** *inj-image-mem-iff*: $inj\ f \Longrightarrow f\ a \in f\ `\ A \longleftrightarrow a \in A$
⟨*proof*⟩

**lemma** *inj-image-subset-iff*: $inj\ f \Longrightarrow f\ `\ A \subseteq f\ `\ B \longleftrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *inj-image-eq-iff*: $inj\ f \Longrightarrow f\ `\ A = f\ `\ B \longleftrightarrow A = B$
⟨*proof*⟩

**lemma** *surj-Compl-image-subset*: $surj\ f \Longrightarrow - (f\ `\ A) \subseteq f\ `\ (- A)$
⟨*proof*⟩

**lemma** *inj-image-Compl-subset*: $inj\ f \Longrightarrow f\ `\ (- A) \subseteq - (f\ `\ A)$
⟨*proof*⟩

**lemma** *bij-image-Compl-eq*: $bij\ f \Longrightarrow f\ `\ (- A) = - (f\ `\ A)$
⟨*proof*⟩

**lemma** *inj-vimage-singleton*: $inj\ f \Longrightarrow f\ -`\ \{a\} \subseteq \{THE\ x.\ f\ x = a\}$
— The inverse image of a singleton under an injective function is included in a singleton.
⟨*proof*⟩

**lemma** *inj-on-vimage-singleton*: $inj\text{-}on\ f\ A \Longrightarrow f\ -`\ \{a\} \cap A \subseteq \{THE\ x.\ x \in A \wedge f\ x = a\}$
⟨*proof*⟩

**lemma** *bij-betw-byWitness*:
 **assumes** *left*: $\forall\, a \in A.\ f'\ (f\ a) = a$
  **and** *right*: $\forall\, a' \in A'.\ f\ (f'\ a') = a'$
  **and** $f\ `\ A \subseteq A'$
  **and** *img2*: $f'\ `\ A' \subseteq A$
 **shows** $bij\text{-}betw\ f\ A\ A'$
⟨*proof*⟩

**corollary** *notIn-Un-bij-betw*:

**assumes** $b \notin A$
**and** $f\ b \notin A'$
**and** *bij-betw* $f\ A\ A'$
**shows** *bij-betw* $f\ (A \cup \{b\})\ (A' \cup \{f\ b\})$
⟨*proof*⟩

**lemma** *notIn-Un-bij-betw3*:
**assumes** $b \notin A$
**and** $f\ b \notin A'$
**shows** *bij-betw* $f\ A\ A' =$ *bij-betw* $f\ (A \cup \{b\})\ (A' \cup \{f\ b\})$
⟨*proof*⟩

**lemma** *inj-on-disjoint-Un*:
**assumes** *inj-on* $f\ A$ **and** *inj-on* $g\ B$
**and** $f$ ' $A \cap g$ ' $B = \{\}$
**shows** *inj-on* $(\lambda x.\ if\ x \in A\ then\ f\ x\ else\ g\ x)\ (A \cup B)$
⟨*proof*⟩

**lemma** *bij-betw-disjoint-Un*:
**assumes** *bij-betw* $f\ A\ C$ **and** *bij-betw* $g\ B\ D$
**and** $A \cap B = \{\}$
**and** $C \cap D = \{\}$
**shows** *bij-betw* $(\lambda x.\ if\ x \in A\ then\ f\ x\ else\ g\ x)\ (A \cup B)\ (C \cup D)$
⟨*proof*⟩

**lemma** *involuntory-imp-bij*:
‹*bij f*› **if** ‹$\bigwedge x.\ f\ (f\ x) = x$›
⟨*proof*⟩

### 10.5.1 Inj/surj/bij of Algebraic Operations

**context** *cancel-semigroup-add*
**begin**

**lemma** *inj-on-add* [*simp*]:
*inj-on* $((+)\ a)\ A$
⟨*proof*⟩

**lemma** *inj-on-add'* [*simp*]:
*inj-on* $(\lambda b.\ b + a)\ A$
⟨*proof*⟩

**lemma** *bij-betw-add* [*simp*]:
*bij-betw* $((+)\ a)\ A\ B \longleftrightarrow (+)\ a$ ' $A = B$
⟨*proof*⟩

**end**

**context** *group-add*

**begin**

**lemma** *diff-left-imp-eq*: $a - b = a - c \Longrightarrow b = c$
⟨*proof*⟩

**lemma** *inj-uminus*[*simp*, *intro*]: *inj-on uminus A*
⟨*proof*⟩

**lemma** *surj-uminus*[*simp*]: *surj uminus*
⟨*proof*⟩

**lemma** *surj-plus* [*simp*]:
*surj* ((+) *a*)
⟨*proof*⟩

**lemma** *surj-plus-right* [*simp*]:
*surj* ($\lambda b.\ b+a$)
⟨*proof*⟩

**lemma** *inj-on-diff-left* [*simp*]:
‹*inj-on* ((−) *a*) *A*›
⟨*proof*⟩

**lemma** *inj-on-diff-right* [*simp*]:
‹*inj-on* ($\lambda b.\ b - a$) *A*›
⟨*proof*⟩

**lemma** *surj-diff* [*simp*]:
*surj* ((−) *a*)
⟨*proof*⟩

**lemma** *surj-diff-right* [*simp*]:
*surj* ($\lambda x.\ x - a$)
⟨*proof*⟩

**lemma shows** *bij-plus*: *bij* ((+) *a*) **and** *bij-plus-right*: *bij* ($\lambda x.\ x + a$)
**and** *bij-uminus*: *bij uminus*
**and** *bij-diff*: *bij* ((−) *a*) **and** *bij-diff-right*: *bij* ($\lambda x.\ x - a$)
⟨*proof*⟩

**lemma** *translation-subtract-Compl*:
($\lambda x.\ x - a$) ‘ (− *t*) = − (($\lambda x.\ x - a$) ‘ *t*)
⟨*proof*⟩

**lemma** *translation-diff*:
(+) *a* ‘ ($s - t$) = ((+) *a* ‘ *s*) − ((+) *a* ‘ *t*)
⟨*proof*⟩

**lemma** *translation-subtract-diff*:

$(\lambda x.\ x - a)$ ‘ $(s - t) = ((\lambda x.\ x - a)$ ‘ $s) - ((\lambda x.\ x - a)$ ‘ $t)$
⟨*proof*⟩

**lemma** *translation-Int*:
$(+)\ a$ ‘ $(s \cap t) = ((+)\ a$ ‘ $s) \cap ((+)\ a$ ‘ $t)$
⟨*proof*⟩

**lemma** *translation-subtract-Int*:
$(\lambda x.\ x - a)$ ‘ $(s \cap t) = ((\lambda x.\ x - a)$ ‘ $s) \cap ((\lambda x.\ x - a)$ ‘ $t)$
⟨*proof*⟩

**lemma** *translation-Compl*:
$(+)\ a$ ‘ $(- t) = - ((+)\ a$ ‘ $t)$
⟨*proof*⟩

**end**

## 10.6 Function Updating

**definition** *fun-upd* :: $('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow ('a \Rightarrow 'b)$
**where** *fun-upd f a b* = $(\lambda x.$ *if* $x = a$ *then* $b$ *else* $f\ x)$

**nonterminal** *updbinds* **and** *updbind*

**open-bundle** *update-syntax*
**begin**

**syntax**
*-updbind* :: $'a \Rightarrow 'a \Rightarrow$ *updbind* (‹(‹*indent=2 notation=*‹*mixfix update*››-
:=/ -)›)
:: *updbind* $\Rightarrow$ *updbinds* (‹-›)
*-updbinds*:: *updbind* $\Rightarrow$ *updbinds* $\Rightarrow$ *updbinds* (‹-,/ -›)
*-Update* :: $'a \Rightarrow$ *updbinds* $\Rightarrow 'a$
(‹(‹*open-block notation=*‹*mixfix function update*››-/'((2-)')›) [*1000*, *0*] *900*)
**syntax-consts**
*-Update* ⇌ *fun-upd*
**translations**
*-Update f* (*-updbinds b bs*) ⇌ *-Update* (*-Update f b*) *bs*
$f(x{:=}y)$ ⇌ *CONST fun-upd f x y*

**end**

**lemma** *fun-upd-idem-iff*: $f(x{:=}y) = f \longleftrightarrow f\ x = y$
⟨*proof*⟩

**lemma** *fun-upd-idem*: $f\ x = y \Longrightarrow f(x := y) = f$
⟨*proof*⟩

**lemma** *fun-upd-triv* [*iff*]: $f(x := f\ x) = f$
⟨*proof*⟩

**lemma** *fun-upd-apply* [*simp*]: $(f(x := y))\ z = (\mathit{if}\ z = x\ \mathit{then}\ y\ \mathit{else}\ f\ z)$
⟨*proof*⟩

**lemma** *fun-upd-same*: $(f(x := y))\ x = y$
⟨*proof*⟩

**lemma** *fun-upd-other*: $z \neq x \Longrightarrow (f(x := y))\ z = f\ z$
⟨*proof*⟩

**lemma** *fun-upd-upd* [*simp*]: $f(x := y, x := z) = f(x := z)$
⟨*proof*⟩

**lemma** *fun-upd-twist*: $a \neq c \Longrightarrow (m(a := b))(c := d) = (m(c := d))(a := b)$
⟨*proof*⟩

**lemma** *inj-on-fun-updI*: $\mathit{inj\text{-}on}\ f\ A \Longrightarrow y \notin f\ `\ A \Longrightarrow \mathit{inj\text{-}on}\ (f(x := y))\ A$
⟨*proof*⟩

**lemma** *fun-upd-image*: $f(x := y)\ `\ A = (\mathit{if}\ x \in A\ \mathit{then}\ \mathit{insert}\ y\ (f\ `\ (A - \{x\}))\ \mathit{else}\ f\ `\ A)$
⟨*proof*⟩

**lemma** *fun-upd-comp*: $f \circ (g(x := y)) = (f \circ g)(x := f\ y)$
⟨*proof*⟩

**lemma** *fun-upd-eqD*: $f(x := y) = g(x := z) \Longrightarrow y = z$
⟨*proof*⟩

## 10.7 *override-on*

**definition** *override-on* :: $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b$
**where** $\mathit{override\text{-}on}\ f\ g\ A = (\lambda a.\ \mathit{if}\ a \in A\ \mathit{then}\ g\ a\ \mathit{else}\ f\ a)$

**lemma** *override-on-emptyset*[*simp*]: $\mathit{override\text{-}on}\ f\ g\ \{\} = f$
⟨*proof*⟩

**lemma** *override-on-apply-notin*[*simp*]: $a \notin A \Longrightarrow (\mathit{override\text{-}on}\ f\ g\ A)\ a = f\ a$
⟨*proof*⟩

**lemma** *override-on-apply-in*[*simp*]: $a \in A \Longrightarrow (\mathit{override\text{-}on}\ f\ g\ A)\ a = g\ a$
⟨*proof*⟩

**lemma** *override-on-insert*: $\mathit{override\text{-}on}\ f\ g\ (\mathit{insert}\ x\ X) = (\mathit{override\text{-}on}\ f\ g\ X)(x{:=}g\ x)$

⟨*proof*⟩

**lemma** *override-on-insert'*: $override\text{-}on\ f\ g\ (insert\ x\ X) = (override\text{-}on\ (f(x{:=}g\ x))\ g\ X)$
⟨*proof*⟩

## 10.8 Inversion of injective functions

**definition** *the-inv-into* :: $'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a)$
**where** $the\text{-}inv\text{-}into\ A\ f = (\lambda x.\ THE\ y.\ y \in A \wedge f\ y = x)$

**lemma** *the-inv-into-f-f*: $inj\text{-}on\ f\ A \Longrightarrow x \in A \Longrightarrow the\text{-}inv\text{-}into\ A\ f\ (f\ x) = x$
⟨*proof*⟩

**lemma** *f-the-inv-into-f*: $inj\text{-}on\ f\ A \Longrightarrow y \in f\ `\ A \Longrightarrow f\ (the\text{-}inv\text{-}into\ A\ f\ y) = y$
⟨*proof*⟩

**lemma** *f-the-inv-into-f-bij-betw*:
$bij\text{-}betw\ f\ A\ B \Longrightarrow (bij\text{-}betw\ f\ A\ B \Longrightarrow x \in B) \Longrightarrow f\ (the\text{-}inv\text{-}into\ A\ f\ x) = x$
⟨*proof*⟩

**lemma** *the-inv-into-into*: $inj\text{-}on\ f\ A \Longrightarrow x \in f\ `\ A \Longrightarrow A \subseteq B \Longrightarrow the\text{-}inv\text{-}into\ A\ f\ x \in B$
⟨*proof*⟩

**lemma** *the-inv-into-onto* [*simp*]: $inj\text{-}on\ f\ A \Longrightarrow the\text{-}inv\text{-}into\ A\ f\ `\ (f\ `\ A) = A$
⟨*proof*⟩

**lemma** *the-inv-into-f-eq*: $inj\text{-}on\ f\ A \Longrightarrow f\ x = y \Longrightarrow x \in A \Longrightarrow the\text{-}inv\text{-}into\ A\ f\ y = x$
⟨*proof*⟩

**lemma** *the-inv-into-comp*:
$inj\text{-}on\ f\ (g\ `\ A) \Longrightarrow inj\text{-}on\ g\ A \Longrightarrow x \in f\ `\ g\ `\ A \Longrightarrow$
$the\text{-}inv\text{-}into\ A\ (f \circ g)\ x = (the\text{-}inv\text{-}into\ A\ g \circ the\text{-}inv\text{-}into\ (g\ `\ A)\ f)\ x$
⟨*proof*⟩

**lemma** *inj-on-the-inv-into*: $inj\text{-}on\ f\ A \Longrightarrow inj\text{-}on\ (the\text{-}inv\text{-}into\ A\ f)\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *bij-betw-the-inv-into*: $bij\text{-}betw\ f\ A\ B \Longrightarrow bij\text{-}betw\ (the\text{-}inv\text{-}into\ A\ f)\ B\ A$
⟨*proof*⟩

**lemma** *bij-betw-iff-bijections*:
$bij\text{-}betw\ f\ A\ B \longleftrightarrow (\exists\ g.\ (\forall\ x \in A.\ f\ x \in B \wedge g(f\ x) = x) \wedge (\forall\ y \in B.\ g\ y \in A \wedge f(g\ y) = y))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

**abbreviation** *the-inv* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a)$
**where** *the-inv f* $\equiv$ *the-inv-into UNIV f*

**lemma** *the-inv-f-f*: *the-inv f* $(f\ x) = x$ **if** *inj f*
$\langle proof \rangle$

## 10.9 Monotonicity

**definition** *monotone-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
**where** *monotone-on A orda ordb f* $\longleftrightarrow (\forall x \in A.\ \forall y \in A.\ orda\ x\ y \longrightarrow ordb\ (f\ x)\ (f\ y))$

**abbreviation** *monotone* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
**where** *monotone* $\equiv$ *monotone-on UNIV*

**lemma** *monotone-def*[*no-atp*]: *monotone orda ordb f* $\longleftrightarrow (\forall x\ y.\ orda\ x\ y \longrightarrow ordb\ (f\ x)\ (f\ y))$
$\langle proof \rangle$

Lemma *monotone-def* is provided for backward compatibility.

**lemma** *monotone-onI*:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow orda\ x\ y \Longrightarrow ordb\ (f\ x)\ (f\ y)) \Longrightarrow monotone\text{-}on\ A\ orda\ ordb\ f$
$\langle proof \rangle$

**lemma** *monotoneI*[*intro?*]: $(\bigwedge x\ y.\ orda\ x\ y \Longrightarrow ordb\ (f\ x)\ (f\ y)) \Longrightarrow monotone\ orda\ ordb\ f$
$\langle proof \rangle$

**lemma** *monotone-onD*:
$monotone\text{-}on\ A\ orda\ ordb\ f \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow orda\ x\ y \Longrightarrow ordb\ (f\ x)\ (f\ y)$
$\langle proof \rangle$

**lemma** *monotoneD*[*dest?*]: $monotone\ orda\ ordb\ f \Longrightarrow orda\ x\ y \Longrightarrow ordb\ (f\ x)\ (f\ y)$
$\langle proof \rangle$

**lemma** *monotone-on-subset*: $monotone\text{-}on\ A\ orda\ ordb\ f \Longrightarrow B \subseteq A \Longrightarrow monotone\text{-}on\ B\ orda\ ordb\ f$
$\langle proof \rangle$

**lemma** *monotone-on-empty*[*simp*]: *monotone-on* $\{\}$ *orda ordb f*
$\langle proof \rangle$

**lemma** *monotone-on-o*:
**assumes**
*mono-f*: *monotone-on A orda ordb f* **and**

  *mono-g*: *monotone-on B ordc orda g* **and**
  $g$ ‘ $B \subseteq A$
 **shows** *monotone-on B ordc ordb* $(f \circ g)$
⟨*proof*⟩

### 10.9.1 Specializations For *ord* Type Class And More

**context** *ord* **begin**

**abbreviation** *mono-on* :: $'a$ *set* $\Rightarrow$ $('a \Rightarrow 'b$ :: *ord*$)$ $\Rightarrow$ *bool*
 **where** *mono-on* $A \equiv$ *monotone-on* $A$ $(\leq)$ $(\leq)$

**abbreviation** *strict-mono-on* :: $'a$ *set* $\Rightarrow$ $('a \Rightarrow 'b$ :: *ord*$)$ $\Rightarrow$ *bool*
 **where** *strict-mono-on* $A \equiv$ *monotone-on* $A$ $(<)$ $(<)$

**abbreviation** *antimono-on* :: $'a$ *set* $\Rightarrow$ $('a \Rightarrow 'b$ :: *ord*$)$ $\Rightarrow$ *bool*
 **where** *antimono-on* $A \equiv$ *monotone-on* $A$ $(\leq)$ $(\lambda x\ y.\ y \leq x)$

**abbreviation** *strict-antimono-on* :: $'a$ *set* $\Rightarrow$ $('a \Rightarrow 'b$ :: *ord*$)$ $\Rightarrow$ *bool*
 **where** *strict-antimono-on* $A \equiv$ *monotone-on* $A$ $(<)$ $(\lambda x\ y.\ y < x)$

**lemma** *mono-on-def*[*no-atp*]: *mono-on* $A\ f \longleftrightarrow (\forall r\ s.\ r \in A \land s \in A \land r \leq s \longrightarrow f\ r \leq f\ s)$
 ⟨*proof*⟩

**lemma** *strict-mono-on-def*[*no-atp*]:
 *strict-mono-on* $A\ f \longleftrightarrow (\forall r\ s.\ r \in A \land s \in A \land r < s \longrightarrow f\ r < f\ s)$
 ⟨*proof*⟩

Lemmas *mono-on-def* and *strict-mono-on-def* are provided for backward compatibility.

**lemma** *mono-onI*:
 $(\bigwedge r\ s.\ r \in A \Longrightarrow s \in A \Longrightarrow r \leq s \Longrightarrow f\ r \leq f\ s) \Longrightarrow$ *mono-on* $A\ f$
 ⟨*proof*⟩

**lemma** *strict-mono-onI*:
 $(\bigwedge r\ s.\ r \in A \Longrightarrow s \in A \Longrightarrow r < s \Longrightarrow f\ r < f\ s) \Longrightarrow$ *strict-mono-on* $A\ f$
 ⟨*proof*⟩

**lemma** *mono-onD*: $[\![$*mono-on* $A\ f;\ r \in A;\ s \in A;\ r \leq s]\!] \Longrightarrow f\ r \leq f\ s$
 ⟨*proof*⟩

**lemma** *strict-mono-onD*: $[\![$*strict-mono-on* $A\ f;\ r \in A;\ s \in A;\ r < s]\!] \Longrightarrow f\ r < f\ s$
 ⟨*proof*⟩

**lemma** *mono-on-subset*: *mono-on* $A\ f \Longrightarrow B \subseteq A \Longrightarrow$ *mono-on* $B\ f$
 ⟨*proof*⟩

**end**

**context** *order* **begin**

**abbreviation** *mono* :: $('a \Rightarrow 'b::order) \Rightarrow bool$
**where** *mono* $\equiv$ *mono-on UNIV*

**abbreviation** *strict-mono* :: $('a \Rightarrow 'b::order) \Rightarrow bool$
**where** *strict-mono* $\equiv$ *strict-mono-on UNIV*

**abbreviation** *antimono* :: $('a \Rightarrow 'b::order) \Rightarrow bool$
**where** *antimono* $\equiv$ *monotone* $(\leq)$ $(\lambda x\ y.\ y \leq x)$

**lemma** *mono-def*[*no-atp*]: *mono f* $\longleftrightarrow$ $(\forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$
⟨*proof*⟩

**lemma** *strict-mono-def*[*no-atp*]: *strict-mono f* $\longleftrightarrow$ $(\forall x\ y.\ x < y \longrightarrow f\ x < f\ y)$
⟨*proof*⟩

**lemma** *antimono-def*[*no-atp*]: *antimono f* $\longleftrightarrow$ $(\forall x\ y.\ x \leq y \longrightarrow f\ x \geq f\ y)$
⟨*proof*⟩

Lemmas *mono-def*, *strict-mono-def*, and *antimono-def* are provided for backward compatibility.

**lemma** *monoI* [*intro?*]: $(\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \leq f\ y) \Longrightarrow$ *mono f*
⟨*proof*⟩

**lemma** *strict-monoI* [*intro?*]: $(\bigwedge x\ y.\ x < y \Longrightarrow f\ x < f\ y) \Longrightarrow$ *strict-mono f*
⟨*proof*⟩

**lemma** *antimonoI* [*intro?*]: $(\bigwedge x\ y.\ x \leq y \Longrightarrow f\ x \geq f\ y) \Longrightarrow$ *antimono f*
⟨*proof*⟩

**lemma** *monoD* [*dest?*]: *mono f* $\Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$
⟨*proof*⟩

**lemma** *strict-monoD* [*dest?*]: *strict-mono f* $\Longrightarrow x < y \Longrightarrow f\ x < f\ y$
⟨*proof*⟩

**lemma** *antimonoD* [*dest?*]: *antimono f* $\Longrightarrow x \leq y \Longrightarrow f\ x \geq f\ y$
⟨*proof*⟩

**lemma** *monoE*:
**assumes** *mono f*
**assumes** $x \leq y$
**obtains** $f\ x \leq f\ y$
⟨*proof*⟩

**lemma** *antimonoE*:
**fixes** $f :: 'a \Rightarrow 'b::order$

**assumes** *antimono f*
**assumes** $x \leq y$
**obtains** $f\,x \geq f\,y$
⟨*proof*⟩

**end**

**lemma** *mono-imp-mono-on*: *mono f* $\Longrightarrow$ *mono-on A f*
⟨*proof*⟩

**lemma** *strict-mono-on-imp-mono-on*: *strict-mono-on A f* $\Longrightarrow$ *mono-on A f*
**for** $f$ :: *'a::order* $\Rightarrow$ *'b::preorder*
⟨*proof*⟩

**lemma** *strict-mono-mono* [*dest?*]:
*strict-mono f* $\Longrightarrow$ *mono f*
⟨*proof*⟩

**lemma** *mono-on-ident*: *mono-on S* ($\lambda x.\ x$)
⟨*proof*⟩

**lemma** *mono-on-id*: *mono-on S id*
⟨*proof*⟩

**lemma** *strict-mono-on-ident*: *strict-mono-on S* ($\lambda x.\ x$)
⟨*proof*⟩

**lemma** *strict-mono-on-id*: *strict-mono-on S id*
⟨*proof*⟩

**lemma** *mono-on-const*:
**fixes** $a$ :: *'b::preorder* **shows** *mono-on S* ($\lambda x.\ a$)
⟨*proof*⟩

**lemma** *antimono-on-const*:
**fixes** $a$ :: *'b::preorder* **shows** *antimono-on S* ($\lambda x.\ a$)
⟨*proof*⟩

**context** *linorder* **begin**

**lemma** *mono-on-strict-invE*:
**fixes** $f$ :: *'a* $\Rightarrow$ *'b::preorder*
**assumes** *mono-on S f*
**assumes** $x \in S$ $y \in S$
**assumes** $f\,x < f\,y$
**obtains** $x < y$
⟨*proof*⟩

**corollary** *mono-on-invE*:
**fixes** $f :: 'a \Rightarrow 'b::preorder$
**assumes** *mono-on* $S\ f$
**assumes** $x \in S\ y \in S$
**assumes** $f\ x < f\ y$
**obtains** $x \leq y$
$\langle proof \rangle$

**lemma** *strict-mono-on-eq*:
**assumes** *strict-mono-on* $S\ (f::'a \Rightarrow 'b::preorder)$
**assumes** $x \in S\ y \in S$
**shows** $f\ x = f\ y \longleftrightarrow x = y$
$\langle proof \rangle$

**lemma** *strict-mono-on-less-eq*:
**assumes** *strict-mono-on* $S\ (f::'a \Rightarrow 'b::preorder)$
**assumes** $x \in S\ y \in S$
**shows** $f\ x \leq f\ y \longleftrightarrow x \leq y$
$\langle proof \rangle$

**lemma** *strict-mono-on-less*:
**assumes** *strict-mono-on* $S\ (f::'a \Rightarrow \text{-}::preorder)$
**assumes** $x \in S\ y \in S$
**shows** $f\ x < f\ y \longleftrightarrow x < y$
$\langle proof \rangle$

**lemmas** *mono-invE* = *mono-on-invE*[*OF - UNIV-I UNIV-I*]
**lemmas** *mono-strict-invE* = *mono-on-strict-invE*[*OF - UNIV-I UNIV-I*]
**lemmas** *strict-mono-eq* = *strict-mono-on-eq*[*OF - UNIV-I UNIV-I*]
**lemmas** *strict-mono-less-eq* = *strict-mono-on-less-eq*[*OF - UNIV-I UNIV-I*]
**lemmas** *strict-mono-less* = *strict-mono-on-less*[*OF - UNIV-I UNIV-I*]

**end**

**lemma** *strict-mono-inv*:
**fixes** $f :: ('a::linorder) \Rightarrow ('b::linorder)$
**assumes** *strict-mono* $f$ **and** *surj* $f$ **and** *inv*: $\bigwedge x.\ g\ (f\ x) = x$
**shows** *strict-mono* $g$
$\langle proof \rangle$

**lemma** *strict-mono-on-imp-inj-on*:
**fixes** $f :: 'a::linorder \Rightarrow 'b::preorder$
**assumes** *strict-mono-on* $A\ f$
**shows** *inj-on* $f\ A$
$\langle proof \rangle$

**lemma** *strict-mono-on-leD*:
**fixes** $f :: 'a::order \Rightarrow 'b::preorder$
**assumes** *strict-mono-on* $A\ f\ x \in A\ y \in A\ x \leq y$

**shows** $f\ x \leq f\ y$
$\langle proof \rangle$

**lemma** *strict-mono-on-eqD*:
**fixes** $f$ :: $'c$::*linorder* $\Rightarrow$ $'d$::*preorder*
**assumes** *strict-mono-on* $A\ f\ f\ x = f\ y\ x \in A\ y \in A$
**shows** $y = x$
$\langle proof \rangle$

**lemma** *mono-imp-strict-mono*:
**fixes** $f$ :: $'a$::*order* $\Rightarrow$ $'b$::*order*
**shows** $[\![$*mono-on* $S\ f$; *inj-on* $f\ S]\!] \Longrightarrow$ *strict-mono-on* $S\ f$
$\langle proof \rangle$

**lemma** *strict-mono-iff-mono*:
**fixes** $f$ :: $'a$::*linorder* $\Rightarrow$ $'b$::*order*
**shows** *strict-mono-on* $S\ f \longleftrightarrow$ *mono-on* $S\ f \land$ *inj-on* $f\ S$
$\langle proof \rangle$

**lemma** *antimono-imp-strict-antimono*:
**fixes** $f$ :: $'a$::*order* $\Rightarrow$ $'b$::*order*
**shows** $[\![$*antimono-on* $S\ f$; *inj-on* $f\ S]\!] \Longrightarrow$ *strict-antimono-on* $S\ f$
$\langle proof \rangle$

**lemma** *strict-antimono-iff-antimono*:
**fixes** $f$ :: $'a$::*linorder* $\Rightarrow$ $'b$::*order*
**shows** *strict-antimono-on* $S\ f \longleftrightarrow$ *antimono-on* $S\ f \land$ *inj-on* $f\ S$
$\langle proof \rangle$

**lemma** *mono-compose*: *mono* $Q \Longrightarrow$ *mono* $(\lambda i\ x.\ Q\ i\ (f\ x))$
$\langle proof \rangle$

**lemma** *mono-add*:
**fixes** $a$ :: $'a$::*ordered-ab-semigroup-add*
**shows** *mono* $((+)\ a)$
$\langle proof \rangle$

**lemma** (**in** *semilattice-inf*) *mono-inf*: *mono* $f \Longrightarrow f\ (A \sqcap B) \leq f\ A \sqcap f\ B$
**for** $f$ :: $'a \Rightarrow\ 'b$::*semilattice-inf*
$\langle proof \rangle$

**lemma** (**in** *semilattice-sup*) *mono-sup*: *mono* $f \Longrightarrow f\ A \sqcup f\ B \leq f\ (A \sqcup B)$
**for** $f$ :: $'a \Rightarrow\ 'b$::*semilattice-sup*
$\langle proof \rangle$

**lemma** *monotone-on-sup-fun*:
**fixes** $f\ g$ :: - $\Rightarrow$ -:: *semilattice-sup*
**shows** *monotone-on* $A\ P\ (\leq)\ f \Longrightarrow$ *monotone-on* $A\ P\ (\leq)\ g \Longrightarrow$ *monotone-on* $A\ P\ (\leq)\ (f \sqcup g)$

⟨*proof*⟩

**lemma** *monotone-on-inf-fun*:
**fixes** $f\ g$ :: - ⇒ -:: *semilattice-inf*
**shows** *monotone-on* $A\ P\ (\leq)\ f \Longrightarrow$ *monotone-on* $A\ P\ (\leq)\ g \Longrightarrow$ *monotone-on* $A\ P\ (\leq)\ (f \sqcap g)$
⟨*proof*⟩

**lemma** *antimonotone-on-sup-fun*:
**fixes** $f\ g$ :: - ⇒ -:: *semilattice-sup*
**shows** *monotone-on* $A\ P\ (\geq)\ f \Longrightarrow$ *monotone-on* $A\ P\ (\geq)\ g \Longrightarrow$ *monotone-on* $A\ P\ (\geq)\ (f \sqcup g)$
⟨*proof*⟩

**lemma** *antimonotone-on-inf-fun*:
**fixes** $f\ g$ :: - ⇒ -:: *semilattice-inf*
**shows** *monotone-on* $A\ P\ (\geq)\ f \Longrightarrow$ *monotone-on* $A\ P\ (\geq)\ g \Longrightarrow$ *monotone-on* $A\ P\ (\geq)\ (f \sqcap g)$
⟨*proof*⟩

**lemma** (**in** *linorder*) *min-of-mono*: *mono* $f \Longrightarrow$ *min* $(f\ m)\ (f\ n) = f$ (*min* $m\ n$)
⟨*proof*⟩

**lemma** (**in** *linorder*) *max-of-mono*: *mono* $f \Longrightarrow$ *max* $(f\ m)\ (f\ n) = f$ (*max* $m\ n$)
⟨*proof*⟩

**lemma** (**in** *linorder*)
*max-of-antimono*: *antimono* $f \Longrightarrow$ *max* $(f\ x)\ (f\ y) = f$ (*min* $x\ y$) **and**
*min-of-antimono*: *antimono* $f \Longrightarrow$ *min* $(f\ x)\ (f\ y) = f$ (*max* $x\ y$)
⟨*proof*⟩

**lemma** (**in** *linorder*) *strict-mono-imp-inj-on*: *strict-mono* $f \Longrightarrow$ *inj-on* $f\ A$
⟨*proof*⟩

**lemma** *mono-Int*: *mono* $f \Longrightarrow f\ (A \cap B) \subseteq f\ A \cap f\ B$
⟨*proof*⟩

**lemma** *mono-Un*: *mono* $f \Longrightarrow f\ A \cup f\ B \subseteq f\ (A \cup B)$
⟨*proof*⟩

### 10.9.2 Least value operator

**lemma** *Least-mono*: *mono* $f \Longrightarrow \exists\, x{\in}S.\ \forall\, y{\in}S.\ x \leq y \Longrightarrow (LEAST\ y.\ y \in f\ `\ S) = f\ (LEAST\ x.\ x \in S)$
**for** $f$ :: $'a$::*order* ⇒ $'b$::*order*
— Courtesy of Stephan Merz
⟨*proof*⟩

## 10.10 Setup

### 10.10.1 Proof tools

Simplify terms of the form $f(\ldots,x{:=}y,\ldots,x{:=}z,\ldots)$ to $f(\ldots,x{:=}z,\ldots)$

⟨*ML*⟩

### 10.10.2 Functorial structure of types

⟨*ML*⟩

**functor** *map-fun*: *map-fun*
  ⟨*proof*⟩

**functor** *vimage*
  ⟨*proof*⟩

Legacy theorem names

**lemmas** *o-def* = *comp-def*
**lemmas** *o-apply* = *comp-apply*
**lemmas** *o-assoc* = *comp-assoc* [*symmetric*]
**lemmas** *id-o* = *id-comp*
**lemmas** *o-id* = *comp-id*
**lemmas** *o-eq-dest* = *comp-eq-dest*
**lemmas** *o-eq-elim* = *comp-eq-elim*
**lemmas** *o-eq-dest-lhs* = *comp-eq-dest-lhs*
**lemmas** *o-eq-id-dest* = *comp-eq-id-dest*

**end**

# 11 Complete lattices

**theory** *Complete-Lattices*
  **imports** *Fun*
**begin**

## 11.1 Syntactic infimum and supremum operations

**class** *Inf* =
  **fixes** *Inf* :: *'a set* ⇒ *'a* (‹(‹*open-block notation=*‹*prefix* ⨅››⨅ -)› [*900*] *900*)

**class** *Sup* =
  **fixes** *Sup* :: *'a set* ⇒ *'a* (‹(‹*open-block notation=*‹*prefix* ⨆››⨆ -)› [*900*] *900*)

**syntax**
  *-INF1* :: *pttrns* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder INF*››*INF* -./ -)› [*0*, *10*] *10*)
  *-INF* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder INF*››*INF* -∈-./ -)› [*0*, *0*, *10*] *10*)

*-SUP1* :: *pttrns* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder SUP*››*SUP* -./ -)› [*0*, *10*] *10*)
*-SUP* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder SUP*››*SUP* -∈-./ -)› [*0*, *0*, *10*] *10*)

**syntax**
*-INF1* :: *pttrns* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder* ⨅››⨅ -./ -)› [*0*, *10*] *10*)
*-INF* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder* ⨅››⨅ -∈-./ -)› [*0*, *0*, *10*] *10*)
*-SUP1* :: *pttrns* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder* ⨆››⨆ -./ -)› [*0*, *10*] *10*)
*-SUP* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder* ⨆››⨆ -∈-./ -)› [*0*, *0*, *10*] *10*)

**syntax-consts**
*-INF1 -INF* ⇌ *Inf* **and**
*-SUP1 -SUP* ⇌ *Sup*

**translations**
⨅*x y. f* ⇌ ⨅*x.* ⨅*y. f*
⨅*x. f* ⇌ ⨅(*CONST range* (λ*x. f*))
⨅*x*∈*A. f* ⇌ *CONST Inf* ((λ*x. f*) ‘ *A*)
⨆*x y. f* ⇌ ⨆*x.* ⨆*y. f*
⨆*x. f* ⇌ ⨆(*CONST range* (λ*x. f*))
⨆*x*∈*A. f* ⇌ *CONST Sup* ((λ*x. f*) ‘ *A*)

**context** *Inf*
**begin**

**lemma** *INF-image*: ⨅ (*g* ‘ *f* ‘ *A*) = ⨅ ((*g* ∘ *f*) ‘ *A*)
⟨*proof*⟩

**lemma** *INF-identity-eq* [*simp*]: (⨅*x*∈*A. x*) = ⨅*A*
⟨*proof*⟩

**lemma** *INF-id-eq* [*simp*]: ⨅(*id* ‘ *A*) = ⨅*A*
⟨*proof*⟩

**lemma** *INF-cong*: *A* = *B* ⟹ (⋀*x. x* ∈ *B* ⟹ *C x* = *D x*) ⟹ ⨅(*C* ‘ *A*) = ⨅(*D* ‘ *B*)
⟨*proof*⟩

**lemma** *INF-cong-simp*:
*A* = *B* ⟹ (⋀*x. x* ∈ *B =simp=> C x* = *D x*) ⟹ ⨅(*C* ‘ *A*) = ⨅(*D* ‘ *B*)
⟨*proof*⟩

**end**

**context** *Sup*
**begin**

**lemma** *SUP-image*: $\bigsqcup\ (g\ `\ f\ `\ A) = \bigsqcup\ ((g \circ f)\ `\ A)$
⟨*proof*⟩

**lemma** *SUP-identity-eq* [*simp*]: $(\bigsqcup x{\in}A.\ x) = \bigsqcup A$
⟨*proof*⟩

**lemma** *SUP-id-eq* [*simp*]: $\bigsqcup(id\ `\ A) = \bigsqcup A$
⟨*proof*⟩

**lemma** *SUP-cong*: $A = B \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow C\ x = D\ x) \Longrightarrow \bigsqcup(C\ `\ A) = \bigsqcup(D\ `\ B)$
⟨*proof*⟩

**lemma** *SUP-cong-simp*:
$A = B \Longrightarrow (\bigwedge x.\ x \in B =simp=> C\ x = D\ x) \Longrightarrow \bigsqcup(C\ `\ A) = \bigsqcup(D\ `\ B)$
⟨*proof*⟩

**end**

## 11.2 Abstract complete lattices

A complete lattice always has a bottom and a top, so we include them into the following type class, along with assumptions that define bottom and top in terms of infimum and supremum.

**class** *complete-lattice* = *lattice* + *Inf* + *Sup* + *bot* + *top* +
**assumes** *Inf-lower*: $x \in A \Longrightarrow \bigsqcap A \le x$
**and** *Inf-greatest*: $(\bigwedge x.\ x \in A \Longrightarrow z \le x) \Longrightarrow z \le \bigsqcap A$
**and** *Sup-upper*: $x \in A \Longrightarrow x \le \bigsqcup A$
**and** *Sup-least*: $(\bigwedge x.\ x \in A \Longrightarrow x \le z) \Longrightarrow \bigsqcup A \le z$
**and** *Inf-empty* [*simp*]: $\bigsqcap\{\} = \top$
**and** *Sup-empty* [*simp*]: $\bigsqcup\{\} = \bot$
**begin**

**subclass** *bounded-lattice*
⟨*proof*⟩

**lemma** *dual-complete-lattice*: *class.complete-lattice Sup Inf sup* $(\ge)$ $(>)$ *inf* $\top$ $\bot$
⟨*proof*⟩

**end**

**context** *complete-lattice*
**begin**

**lemma** *Sup-eqI*:

$(\bigwedge y.\ y \in A \Longrightarrow y \leq x) \Longrightarrow (\bigwedge y.\ (\bigwedge z.\ z \in A \Longrightarrow z \leq y) \Longrightarrow x \leq y) \Longrightarrow \bigsqcup A = x$
$\langle proof \rangle$

**lemma** *Inf-eqI*:
$(\bigwedge i.\ i \in A \Longrightarrow x \leq i) \Longrightarrow (\bigwedge y.\ (\bigwedge i.\ i \in A \Longrightarrow y \leq i) \Longrightarrow y \leq x) \Longrightarrow \bigsqcap A = x$
$\langle proof \rangle$

**lemma** *SUP-eqI*:
$(\bigwedge i.\ i \in A \Longrightarrow f\ i \leq x) \Longrightarrow (\bigwedge y.\ (\bigwedge i.\ i \in A \Longrightarrow f\ i \leq y) \Longrightarrow x \leq y) \Longrightarrow (\bigsqcup i{\in}A.\ f\ i) = x$
$\langle proof \rangle$

**lemma** *INF-eqI*:
$(\bigwedge i.\ i \in A \Longrightarrow x \leq f\ i) \Longrightarrow (\bigwedge y.\ (\bigwedge i.\ i \in A \Longrightarrow f\ i \geq y) \Longrightarrow x \geq y) \Longrightarrow (\bigsqcap i{\in}A.\ f\ i) = x$
$\langle proof \rangle$

**lemma** *INF-lower*: $i \in A \Longrightarrow (\bigsqcap i{\in}A.\ f\ i) \leq f\ i$
$\langle proof \rangle$

**lemma** *INF-greatest*: $(\bigwedge i.\ i \in A \Longrightarrow u \leq f\ i) \Longrightarrow u \leq (\bigsqcap i{\in}A.\ f\ i)$
$\langle proof \rangle$

**lemma** *SUP-upper*: $i \in A \Longrightarrow f\ i \leq (\bigsqcup i{\in}A.\ f\ i)$
$\langle proof \rangle$

**lemma** *SUP-least*: $(\bigwedge i.\ i \in A \Longrightarrow f\ i \leq u) \Longrightarrow (\bigsqcup i{\in}A.\ f\ i) \leq u$
$\langle proof \rangle$

**lemma** *Inf-lower2*: $u \in A \Longrightarrow u \leq v \Longrightarrow \bigsqcap A \leq v$
$\langle proof \rangle$

**lemma** *INF-lower2*: $i \in A \Longrightarrow f\ i \leq u \Longrightarrow (\bigsqcap i{\in}A.\ f\ i) \leq u$
$\langle proof \rangle$

**lemma** *Sup-upper2*: $u \in A \Longrightarrow v \leq u \Longrightarrow v \leq \bigsqcup A$
$\langle proof \rangle$

**lemma** *SUP-upper2*: $i \in A \Longrightarrow u \leq f\ i \Longrightarrow u \leq (\bigsqcup i{\in}A.\ f\ i)$
$\langle proof \rangle$

**lemma** *le-Inf-iff*: $b \leq \bigsqcap A \longleftrightarrow (\forall\, a{\in}A.\ b \leq a)$
$\langle proof \rangle$

**lemma** *le-INF-iff*: $u \leq (\bigsqcap i{\in}A.\ f\ i) \longleftrightarrow (\forall\, i{\in}A.\ u \leq f\ i)$
$\langle proof \rangle$

**lemma** *Sup-le-iff*: $\bigsqcup A \leq b \longleftrightarrow (\forall\, a{\in}A.\ a \leq b)$

⟨*proof*⟩

**lemma** *SUP-le-iff*: $(\bigsqcup i{\in}A.\ f\ i) \le u \longleftrightarrow (\forall\, i{\in}A.\ f\ i \le u)$
⟨*proof*⟩

**lemma** *Inf-insert* [*simp*]: $\bigsqcap(insert\ a\ A) = a \sqcap \bigsqcap A$
⟨*proof*⟩

**lemma** *INF-insert*: $(\bigsqcap x{\in}insert\ a\ A.\ f\ x) = f\ a \sqcap \bigsqcap(f\ `\ A)$
⟨*proof*⟩

**lemma** *Sup-insert* [*simp*]: $\bigsqcup(insert\ a\ A) = a \sqcup \bigsqcup A$
⟨*proof*⟩

**lemma** *SUP-insert*: $(\bigsqcup x{\in}insert\ a\ A.\ f\ x) = f\ a \sqcup \bigsqcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *INF-empty*: $(\bigsqcap x{\in}\{\}.\ f\ x) = \top$
⟨*proof*⟩

**lemma** *SUP-empty*: $(\bigsqcup x{\in}\{\}.\ f\ x) = \bot$
⟨*proof*⟩

**lemma** *Inf-UNIV* [*simp*]: $\bigsqcap UNIV = \bot$
⟨*proof*⟩

**lemma** *Sup-UNIV* [*simp*]: $\bigsqcup UNIV = \top$
⟨*proof*⟩

**lemma** *Inf-eq-Sup*: $\bigsqcap A = \bigsqcup\{b.\ \forall\, a \in A.\ b \le a\}$
⟨*proof*⟩

**lemma** *Sup-eq-Inf*: $\bigsqcup A = \bigsqcap\{b.\ \forall\, a \in A.\ a \le b\}$
⟨*proof*⟩

**lemma** *Inf-superset-mono*: $B \subseteq A \Longrightarrow \bigsqcap A \le \bigsqcap B$
⟨*proof*⟩

**lemma** *Sup-subset-mono*: $A \subseteq B \Longrightarrow \bigsqcup A \le \bigsqcup B$
⟨*proof*⟩

**lemma** *Inf-mono*:
**assumes** $\bigwedge b.\ b \in B \Longrightarrow \exists\, a{\in}A.\ a \le b$
**shows** $\bigsqcap A \le \bigsqcap B$
⟨*proof*⟩

**lemma** *INF-mono*: $(\bigwedge m.\ m \in B \Longrightarrow \exists\, n{\in}A.\ f\ n \le g\ m) \Longrightarrow (\bigsqcap n{\in}A.\ f\ n) \le (\bigsqcap n{\in}B.\ g\ n)$
⟨*proof*⟩

**lemma** *INF-mono'*: $(\bigwedge x.\ f\ x \le g\ x) \Longrightarrow (\bigsqcap x{\in}A.\ f\ x) \le (\bigsqcap x{\in}A.\ g\ x)$
⟨*proof*⟩

**lemma** *Sup-mono*:
**assumes** $\bigwedge a.\ a \in A \Longrightarrow \exists\, b{\in}B.\ a \le b$
**shows** $\bigsqcup A \le \bigsqcup B$
⟨*proof*⟩

**lemma** *SUP-mono*: $(\bigwedge n.\ n \in A \Longrightarrow \exists\, m{\in}B.\ f\ n \le g\ m) \Longrightarrow (\bigsqcup n{\in}A.\ f\ n) \le (\bigsqcup n{\in}B.\ g\ n)$
⟨*proof*⟩

**lemma** *SUP-mono'*: $(\bigwedge x.\ f\ x \le g\ x) \Longrightarrow (\bigsqcup x{\in}A.\ f\ x) \le (\bigsqcup x{\in}A.\ g\ x)$
⟨*proof*⟩

**lemma** *INF-superset-mono*: $B \subseteq A \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow f\ x \le g\ x) \Longrightarrow (\bigsqcap x{\in}A.\ f\ x) \le (\bigsqcap x{\in}B.\ g\ x)$
— The last inclusion is POSITIVE!
⟨*proof*⟩

**lemma** *SUP-subset-mono*: $A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\ x \le g\ x) \Longrightarrow (\bigsqcup x{\in}A.\ f\ x) \le (\bigsqcup x{\in}B.\ g\ x)$
⟨*proof*⟩

**lemma** *Inf-less-eq*:
**assumes** $\bigwedge v.\ v \in A \Longrightarrow v \le u$
**and** $A \neq \{\}$
**shows** $\bigsqcap A \le u$
⟨*proof*⟩

**lemma** *less-eq-Sup*:
**assumes** $\bigwedge v.\ v \in A \Longrightarrow u \le v$
**and** $A \neq \{\}$
**shows** $u \le \bigsqcup A$
⟨*proof*⟩

**lemma** *INF-eq*:
**assumes** $\bigwedge i.\ i \in A \Longrightarrow \exists j{\in}B.\ f\ i \ge g\ j$
**and** $\bigwedge j.\ j \in B \Longrightarrow \exists\, i{\in}A.\ g\ j \ge f\ i$
**shows** $\bigsqcap (f\ `\ A) = \bigsqcap (g\ `\ B)$
⟨*proof*⟩

**lemma** *SUP-eq*:
**assumes** $\bigwedge i.\ i \in A \Longrightarrow \exists j{\in}B.\ f\ i \le g\ j$
**and** $\bigwedge j.\ j \in B \Longrightarrow \exists\, i{\in}A.\ g\ j \le f\ i$
**shows** $\bigsqcup (f\ `\ A) = \bigsqcup (g\ `\ B)$
⟨*proof*⟩

**lemma** *less-eq-Inf-inter*: $\bigsqcap A \sqcup \bigsqcap B \leq \bigsqcap(A \cap B)$
⟨*proof*⟩

**lemma** *Sup-inter-less-eq*: $\bigsqcup(A \cap B) \leq \bigsqcup A \sqcap \bigsqcup B$
⟨*proof*⟩

**lemma** *Inf-union-distrib*: $\bigsqcap(A \cup B) = \bigsqcap A \sqcap \bigsqcap B$
⟨*proof*⟩

**lemma** *INF-union*: $(\bigsqcap i \in A \cup B.\ M\ i) = (\bigsqcap i \in A.\ M\ i) \sqcap (\bigsqcap i{\in}B.\ M\ i)$
⟨*proof*⟩

**lemma** *Sup-union-distrib*: $\bigsqcup(A \cup B) = \bigsqcup A \sqcup \bigsqcup B$
⟨*proof*⟩

**lemma** *SUP-union*: $(\bigsqcup i \in A \cup B.\ M\ i) = (\bigsqcup i \in A.\ M\ i) \sqcup (\bigsqcup i{\in}B.\ M\ i)$
⟨*proof*⟩

**lemma** *INF-inf-distrib*: $(\bigsqcap a{\in}A.\ f\ a) \sqcap (\bigsqcap a{\in}A.\ g\ a) = (\bigsqcap a{\in}A.\ f\ a \sqcap g\ a)$
⟨*proof*⟩

**lemma** *SUP-sup-distrib*: $(\bigsqcup a{\in}A.\ f\ a) \sqcup (\bigsqcup a{\in}A.\ g\ a) = (\bigsqcup a{\in}A.\ f\ a \sqcup g\ a)$
(**is** $?L = ?R$)
⟨*proof*⟩

**lemma** *Inf-top-conv* [*simp*]:
$\bigsqcap A = \top \longleftrightarrow (\forall x{\in}A.\ x = \top)$
$\top = \bigsqcap A \longleftrightarrow (\forall x{\in}A.\ x = \top)$
⟨*proof*⟩

**lemma** *INF-top-conv* [*simp*]:
$(\bigsqcap x{\in}A.\ B\ x) = \top \longleftrightarrow (\forall x{\in}A.\ B\ x = \top)$
$\top = (\bigsqcap x{\in}A.\ B\ x) \longleftrightarrow (\forall x{\in}A.\ B\ x = \top)$
⟨*proof*⟩

**lemma** *Sup-bot-conv* [*simp*]:
$\bigsqcup A = \bot \longleftrightarrow (\forall x{\in}A.\ x = \bot)$
$\bot = \bigsqcup A \longleftrightarrow (\forall x{\in}A.\ x = \bot)$
⟨*proof*⟩

**lemma** *SUP-bot-conv* [*simp*]:
$(\bigsqcup x{\in}A.\ B\ x) = \bot \longleftrightarrow (\forall x{\in}A.\ B\ x = \bot)$
$\bot = (\bigsqcup x{\in}A.\ B\ x) \longleftrightarrow (\forall x{\in}A.\ B\ x = \bot)$
⟨*proof*⟩

**lemma** *INF-constant*: $(\bigsqcap y{\in}A.\ c) = (\mathit{if}\ A = \{\}\ \mathit{then}\ \top\ \mathit{else}\ c)$
⟨*proof*⟩

**lemma** *SUP-constant*: $(\bigsqcup y{\in}A.\ c) = (\mathit{if}\ A = \{\}\ \mathit{then}\ \bot\ \mathit{else}\ c)$

⟨*proof*⟩

**lemma** *INF-const* [*simp*]: $A \neq \{\} \Longrightarrow (\bigsqcap i{\in}A.\ f) = f$
⟨*proof*⟩

**lemma** *SUP-const* [*simp*]: $A \neq \{\} \Longrightarrow (\bigsqcup i{\in}A.\ f) = f$
⟨*proof*⟩

**lemma** *INF-top* [*simp*]: $(\bigsqcap x{\in}A.\ \top) = \top$
⟨*proof*⟩

**lemma** *SUP-bot* [*simp*]: $(\bigsqcup x{\in}A.\ \bot) = \bot$
⟨*proof*⟩

**lemma** *INF-commute*: $(\bigsqcap i{\in}A.\ \bigsqcap j{\in}B.\ f\ i\ j) = (\bigsqcap j{\in}B.\ \bigsqcap i{\in}A.\ f\ i\ j)$
⟨*proof*⟩

**lemma** *SUP-commute*: $(\bigsqcup i{\in}A.\ \bigsqcup j{\in}B.\ f\ i\ j) = (\bigsqcup j{\in}B.\ \bigsqcup i{\in}A.\ f\ i\ j)$
⟨*proof*⟩

**lemma** *INF-absorb*:
**assumes** $k \in I$
**shows** $A\ k \sqcap (\bigsqcap i{\in}I.\ A\ i) = (\bigsqcap i{\in}I.\ A\ i)$
⟨*proof*⟩

**lemma** *SUP-absorb*:
**assumes** $k \in I$
**shows** $A\ k \sqcup (\bigsqcup i{\in}I.\ A\ i) = (\bigsqcup i{\in}I.\ A\ i)$
⟨*proof*⟩

**lemma** *INF-inf-const1*: $I \neq \{\} \Longrightarrow (\bigsqcap i{\in}I.\ \mathit{inf}\ x\ (f\ i)) = \mathit{inf}\ x\ (\bigsqcap i{\in}I.\ f\ i)$
⟨*proof*⟩

**lemma** *INF-inf-const2*: $I \neq \{\} \Longrightarrow (\bigsqcap i{\in}I.\ \mathit{inf}\ (f\ i)\ x) = \mathit{inf}\ (\bigsqcap i{\in}I.\ f\ i)\ x$
⟨*proof*⟩

**lemma** *less-INF-D*:
**assumes** $y < (\bigsqcap i{\in}A.\ f\ i)\ i \in A$
**shows** $y < f\ i$
⟨*proof*⟩

**lemma** *SUP-lessD*:
**assumes** $(\bigsqcup i{\in}A.\ f\ i) < y\ i \in A$
**shows** $f\ i < y$
⟨*proof*⟩

**lemma** *INF-UNIV-bool-expand*: $(\bigsqcap b.\ A\ b) = A\ \mathit{True} \sqcap A\ \mathit{False}$
⟨*proof*⟩

**lemma** *SUP-UNIV-bool-expand*: $(\bigsqcup b.\ A\ b) = A\ \mathit{True} \sqcup A\ \mathit{False}$
⟨*proof*⟩

**lemma** *Inf-le-Sup*: $A \neq \{\} \Longrightarrow \mathit{Inf}\ A \leq \mathit{Sup}\ A$
⟨*proof*⟩

**lemma** *INF-le-SUP*: $A \neq \{\} \Longrightarrow \bigsqcap(f\ `\ A) \leq \bigsqcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *INF-eq-const*: $I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i = x) \Longrightarrow \bigsqcap(f\ `\ I) = x$
⟨*proof*⟩

**lemma** *SUP-eq-const*: $I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i = x) \Longrightarrow \bigsqcup(f\ `\ I) = x$
⟨*proof*⟩

**lemma** *INF-eq-iff*: $I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i \leq c) \Longrightarrow \bigsqcap(f\ `\ I) = c \longleftrightarrow (\forall\, i{\in}I.\ f\ i = c)$
⟨*proof*⟩

**lemma** *SUP-eq-iff*: $I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow c \leq f\ i) \Longrightarrow \bigsqcup(f\ `\ I) = c \longleftrightarrow (\forall\, i{\in}I.\ f\ i = c)$
⟨*proof*⟩

**end**

**context** *complete-lattice*
**begin**
**lemma** *Sup-Inf-le*: $\mathit{Sup}\ (\mathit{Inf}\ `\ \{f\ `\ A \mid f\ .\ (\forall\ \ Y \in A\ .\ f\ Y \in Y)\}) \leq \mathit{Inf}\ (\mathit{Sup}\ `\ A)$
⟨*proof*⟩
**end**

**class** *complete-distrib-lattice* = *complete-lattice* +
**assumes** *Inf-Sup-le*: $\mathit{Inf}\ (\mathit{Sup}\ `\ A) \leq \mathit{Sup}\ (\mathit{Inf}\ `\ \{f\ `\ A \mid f\ .\ (\forall\ \ Y \in A\ .\ f\ Y \in Y)\})$
**begin**

**lemma** *Inf-Sup*: $\mathit{Inf}\ (\mathit{Sup}\ `\ A) = \mathit{Sup}\ (\mathit{Inf}\ `\ \{f\ `\ A \mid f\ .\ (\forall\ \ Y \in A\ .\ f\ Y \in Y)\})$
⟨*proof*⟩

**subclass** *distrib-lattice*
⟨*proof*⟩
**end**

**context** *complete-lattice*
**begin**
**context**
**fixes** $f :: 'a \Rightarrow 'b{::}\mathit{complete\text{-}lattice}$
**assumes** *mono f*
**begin**

**lemma** *mono-Inf*: $f\ (\bigsqcap A) \leq (\bigsqcap x{\in}A.\ f\ x)$
⟨*proof*⟩

**lemma** *mono-Sup*: $(\bigsqcup x{\in}A.\ f\ x) \leq f\ (\bigsqcup A)$
⟨*proof*⟩

**lemma** *mono-INF*: $f\ (\bigsqcap i{\in}I.\ A\ i) \leq (\bigsqcap x{\in}I.\ f\ (A\ x))$
⟨*proof*⟩

**lemma** *mono-SUP*: $(\bigsqcup x{\in}I.\ f\ (A\ x)) \leq f\ (\bigsqcup i{\in}I.\ A\ i)$
⟨*proof*⟩

**end**

**end**

**class** *complete-boolean-algebra* = *boolean-algebra* + *complete-distrib-lattice*
**begin**

**lemma** *uminus-Inf*: $-\ (\bigsqcap A) = \bigsqcup (uminus\ `\ A)$
⟨*proof*⟩

**lemma** *uminus-INF*: $-\ (\bigsqcap x{\in}A.\ B\ x) = (\bigsqcup x{\in}A.\ -\ B\ x)$
⟨*proof*⟩

**lemma** *uminus-Sup*: $-\ (\bigsqcup A) = \bigsqcap (uminus\ `\ A)$
⟨*proof*⟩

**lemma** *uminus-SUP*: $-\ (\bigsqcup x{\in}A.\ B\ x) = (\bigsqcap x{\in}A.\ -\ B\ x)$
⟨*proof*⟩

**end**

**class** *complete-linorder* = *linorder* + *complete-lattice*
**begin**

**lemma** *dual-complete-linorder*:
*class.complete-linorder Sup Inf sup* $(\geq)\ (>)$ *inf* $\top\ \bot$
⟨*proof*⟩

**lemma** *complete-linorder-inf-min*: *inf* = *min*
⟨*proof*⟩

**lemma** *complete-linorder-sup-max*: *sup* = *max*
⟨*proof*⟩

**lemma** *Inf-less-iff*: $\bigsqcap S < a \longleftrightarrow (\exists x{\in}S.\ x < a)$
⟨*proof*⟩

**lemma** *INF-less-iff*: $(\prod i{\in}A.\ f\ i) < a \longleftrightarrow (\exists\, x{\in}A.\ f\ x < a)$
⟨*proof*⟩

**lemma** *less-Sup-iff*: $a < \bigsqcup S \longleftrightarrow (\exists\, x{\in}S.\ a < x)$
⟨*proof*⟩

**lemma** *less-SUP-iff*: $a < (\bigsqcup i{\in}A.\ f\ i) \longleftrightarrow (\exists\, x{\in}A.\ a < f\ x)$
⟨*proof*⟩

**lemma** *Sup-eq-top-iff* [*simp*]: $\bigsqcup A = \top \longleftrightarrow (\forall\, x{<}\top.\ \exists\, i{\in}A.\ x < i)$
⟨*proof*⟩

**lemma** *SUP-eq-top-iff* [*simp*]: $(\bigsqcup i{\in}A.\ f\ i) = \top \longleftrightarrow (\forall\, x{<}\top.\ \exists\, i{\in}A.\ x < f\ i)$
⟨*proof*⟩

**lemma** *Inf-eq-bot-iff* [*simp*]: $\prod A = \bot \longleftrightarrow (\forall\, x{>}\bot.\ \exists\, i{\in}A.\ i < x)$
⟨*proof*⟩

**lemma** *INF-eq-bot-iff* [*simp*]: $(\prod i{\in}A.\ f\ i) = \bot \longleftrightarrow (\forall\, x{>}\bot.\ \exists\, i{\in}A.\ f\ i < x)$
⟨*proof*⟩

**lemma** *Inf-le-iff*: $\prod A \le x \longleftrightarrow (\forall\, y{>}x.\ \exists\, a{\in}A.\ y > a)$
⟨*proof*⟩

**lemma** *INF-le-iff*: $\prod (f\ `\ A) \le x \longleftrightarrow (\forall\, y{>}x.\ \exists\, i{\in}A.\ y > f\ i)$
⟨*proof*⟩

**lemma** *le-Sup-iff*: $x \le \bigsqcup A \longleftrightarrow (\forall\, y{<}x.\ \exists\, a{\in}A.\ y < a)$
⟨*proof*⟩

**lemma** *le-SUP-iff*: $x \le \bigsqcup (f\ `\ A) \longleftrightarrow (\forall\, y{<}x.\ \exists\, i{\in}A.\ y < f\ i)$
⟨*proof*⟩

**end**

## 11.3 Complete lattice on *bool*

**instantiation** *bool* :: *complete-lattice*
**begin**

**definition** [*simp*, *code*]: $\prod A \longleftrightarrow \mathit{False} \notin A$

**definition** [*simp*, *code*]: $\bigsqcup A \longleftrightarrow \mathit{True} \in A$

**instance**
⟨*proof*⟩

**end**

**lemma** *not-False-in-image-Ball* [*simp*]: $False \notin P \text{ ` } A \longleftrightarrow Ball\ A\ P$
⟨*proof*⟩

**lemma** *True-in-image-Bex* [*simp*]: $True \in P \text{ ` } A \longleftrightarrow Bex\ A\ P$
⟨*proof*⟩

**lemma** *INF-bool-eq* [*simp*]: $(\lambda A\ f.\ \bigsqcap(f \text{ ` } A)) = Ball$
⟨*proof*⟩

**lemma** *SUP-bool-eq* [*simp*]: $(\lambda A\ f.\ \bigsqcup(f \text{ ` } A)) = Bex$
⟨*proof*⟩

**instance** *bool* :: *complete-boolean-algebra*
⟨*proof*⟩

## 11.4 Complete lattice on - ⇒ -

**instantiation** *fun* :: (*type*, *Inf*) *Inf*
**begin**

**definition** $\bigsqcap A = (\lambda x.\ \bigsqcap f{\in}A.\ f\ x)$

**lemma** *Inf-apply* [*simp*, *code*]: $(\bigsqcap A)\ x = (\bigsqcap f{\in}A.\ f\ x)$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *Sup*) *Sup*
**begin**

**definition** $\bigsqcup A = (\lambda x.\ \bigsqcup f{\in}A.\ f\ x)$

**lemma** *Sup-apply* [*simp*, *code*]: $(\bigsqcup A)\ x = (\bigsqcup f{\in}A.\ f\ x)$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: (*type*, *complete-lattice*) *complete-lattice*
**begin**

**instance**
⟨*proof*⟩

**end**

**lemma** *INF-apply* [*simp*]: $(\bigsqcap y{\in}A.\ f\ y)\ x = (\bigsqcap y{\in}A.\ f\ y\ x)$
⟨*proof*⟩

**lemma** *SUP-apply* [*simp*]: $(\bigsqcup y{\in}A.\ f\ y)\ x = (\bigsqcup y{\in}A.\ f\ y\ x)$
⟨*proof*⟩

## 11.5 Complete lattice on unary and binary predicates

**lemma** *Inf1-I*: $(\bigwedge P.\ P \in A \Longrightarrow P\ a) \Longrightarrow (\bigsqcap A)\ a$
⟨*proof*⟩

**lemma** *INF1-I*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x\ b) \Longrightarrow (\bigsqcap x{\in}A.\ B\ x)\ b$
⟨*proof*⟩

**lemma** *INF2-I*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x\ b\ c) \Longrightarrow (\bigsqcap x{\in}A.\ B\ x)\ b\ c$
⟨*proof*⟩

**lemma** *Inf2-I*: $(\bigwedge r.\ r \in A \Longrightarrow r\ a\ b) \Longrightarrow (\bigsqcap A)\ a\ b$
⟨*proof*⟩

**lemma** *Inf1-D*: $(\bigsqcap A)\ a \Longrightarrow P \in A \Longrightarrow P\ a$
⟨*proof*⟩

**lemma** *INF1-D*: $(\bigsqcap x{\in}A.\ B\ x)\ b \Longrightarrow a \in A \Longrightarrow B\ a\ b$
⟨*proof*⟩

**lemma** *Inf2-D*: $(\bigsqcap A)\ a\ b \Longrightarrow r \in A \Longrightarrow r\ a\ b$
⟨*proof*⟩

**lemma** *INF2-D*: $(\bigsqcap x{\in}A.\ B\ x)\ b\ c \Longrightarrow a \in A \Longrightarrow B\ a\ b\ c$
⟨*proof*⟩

**lemma** *Inf1-E*:
**assumes** $(\bigsqcap A)\ a$
**obtains** $P\ a \mid P \notin A$
⟨*proof*⟩

**lemma** *INF1-E*:
**assumes** $(\bigsqcap x{\in}A.\ B\ x)\ b$
**obtains** $B\ a\ b \mid a \notin A$
⟨*proof*⟩

**lemma** *Inf2-E*:
**assumes** $(\bigsqcap A)\ a\ b$
**obtains** $r\ a\ b \mid r \notin A$
⟨*proof*⟩

**lemma** *INF2-E*:

**assumes** $(\bigsqcap x{\in}A.\ B\ x)\ b\ c$
**obtains** $B\ a\ b\ c \mid a \notin A$
⟨*proof*⟩

**lemma** *Sup1-I*: $P \in A \Longrightarrow P\ a \Longrightarrow (\bigsqcup A)\ a$
⟨*proof*⟩

**lemma** *SUP1-I*: $a \in A \Longrightarrow B\ a\ b \Longrightarrow (\bigsqcup x{\in}A.\ B\ x)\ b$
⟨*proof*⟩

**lemma** *Sup2-I*: $r \in A \Longrightarrow r\ a\ b \Longrightarrow (\bigsqcup A)\ a\ b$
⟨*proof*⟩

**lemma** *SUP2-I*: $a \in A \Longrightarrow B\ a\ b\ c \Longrightarrow (\bigsqcup x{\in}A.\ B\ x)\ b\ c$
⟨*proof*⟩

**lemma** *Sup1-E*:
**assumes** $(\bigsqcup A)\ a$
**obtains** $P$ **where** $P \in A$ **and** $P\ a$
⟨*proof*⟩

**lemma** *SUP1-E*:
**assumes** $(\bigsqcup x{\in}A.\ B\ x)\ b$
**obtains** $x$ **where** $x \in A$ **and** $B\ x\ b$
⟨*proof*⟩

**lemma** *Sup2-E*:
**assumes** $(\bigsqcup A)\ a\ b$
**obtains** $r$ **where** $r \in A\ r\ a\ b$
⟨*proof*⟩

**lemma** *SUP2-E*:
**assumes** $(\bigsqcup x{\in}A.\ B\ x)\ b\ c$
**obtains** $x$ **where** $x \in A\ B\ x\ b\ c$
⟨*proof*⟩

## 11.6 Complete lattice on - *set*

**instantiation** *set* :: (*type*) *complete-lattice*
**begin**

**definition** $\bigsqcap A = \{x.\ \bigsqcap((\lambda B.\ x \in B)\ \text{`}\ A)\}$

**definition** $\bigsqcup A = \{x.\ \bigsqcup((\lambda B.\ x \in B)\ \text{`}\ A)\}$

**instance**
⟨*proof*⟩

**end**

### 11.6.1 Inter

**abbreviation** *Inter* :: $'a\ set\ set \Rightarrow 'a\ set$ (‹$\bigcap$›)
**where** $\bigcap S \equiv \prod S$

**lemma** *Inter-eq*: $\bigcap A = \{x.\ \forall B \in A.\ x \in B\}$
⟨*proof*⟩

**lemma** *Inter-iff* [*simp*]: $A \in \bigcap C \longleftrightarrow (\forall X{\in}C.\ A \in X)$
⟨*proof*⟩

**lemma** *InterI* [*intro*!]: $(\bigwedge X.\ X \in C \Longrightarrow A \in X) \Longrightarrow A \in \bigcap C$
⟨*proof*⟩

A "destruct" rule – every $X$ in $C$ contains $A$ as an element, but $A \in X$ can hold when $X \in C$ does not! This rule is analogous to *spec*.

**lemma** *InterD* [*elim*, *Pure.elim*]: $A \in \bigcap C \Longrightarrow X \in C \Longrightarrow A \in X$
⟨*proof*⟩

**lemma** *InterE* [*elim*]: $A \in \bigcap C \Longrightarrow (X \notin C \Longrightarrow R) \Longrightarrow (A \in X \Longrightarrow R) \Longrightarrow R$
— "Classical" elimination rule – does not require proving $X \in C$.
⟨*proof*⟩

**lemma** *Inter-lower*: $B \in A \Longrightarrow \bigcap A \subseteq B$
⟨*proof*⟩

**lemma** *Inter-subset*: $(\bigwedge X.\ X \in A \Longrightarrow X \subseteq B) \Longrightarrow A \neq \{\} \Longrightarrow \bigcap A \subseteq B$
⟨*proof*⟩

**lemma** *Inter-greatest*: $(\bigwedge X.\ X \in A \Longrightarrow C \subseteq X) \Longrightarrow C \subseteq \bigcap A$
⟨*proof*⟩

**lemma** *Inter-empty*: $\bigcap \{\} = UNIV$
⟨*proof*⟩

**lemma** *Inter-UNIV*: $\bigcap UNIV = \{\}$
⟨*proof*⟩

**lemma** *Inter-insert*: $\bigcap (insert\ a\ B) = a \cap \bigcap B$
⟨*proof*⟩

**lemma** *Inter-Un-subset*: $\bigcap A \cup \bigcap B \subseteq \bigcap (A \cap B)$
⟨*proof*⟩

**lemma** *Inter-Un-distrib*: $\bigcap (A \cup B) = \bigcap A \cap \bigcap B$
⟨*proof*⟩

**lemma** *Inter-UNIV-conv* [*simp*]:
$\bigcap A = UNIV \longleftrightarrow (\forall x{\in}A.\ x = UNIV)$

$UNIV = \bigcap A \longleftrightarrow (\forall x{\in}A.\ x = UNIV)$
⟨*proof*⟩

**lemma** *Inter-anti-mono*: $B \subseteq A \Longrightarrow \bigcap A \subseteq \bigcap B$
⟨*proof*⟩

### 11.6.2 Intersections of families

**syntax** (*ASCII*)
*-INTER1* :: *pttrns* ⇒ ′*b set* ⇒ ′*b set* (‹(‹*indent=3 notation=*‹*binder INT*››*INT* -./ -)› [*0*, *10*] *10*)
*-INTER* :: *pttrn* ⇒ ′*a set* ⇒ ′*b set* ⇒ ′*b set* (‹(‹*indent=3 notation=*‹*binder INT*››*INT* -:-./ -)› [*0*, *0*, *10*] *10*)

**syntax**
*-INTER1* :: *pttrns* ⇒ ′*b set* ⇒ ′*b set* (‹(‹*indent=3 notation=*‹*binder* ⋂››⋂ -./ -)› [*0*, *10*] *10*)
*-INTER* :: *pttrn* ⇒ ′*a set* ⇒ ′*b set* ⇒ ′*b set* (‹(‹*indent=3 notation=*‹*binder* ⋂››⋂ -∈-./ -)› [*0*, *0*, *10*] *10*)

**syntax** (*latex* **output**)
*-INTER1* :: *pttrns* ⇒ ′*b set* ⇒ ′*b set* (‹(*3*⋂(‹*unbreakable*›$_{-}$)/ -)› [*0*, *10*] *10*)
*-INTER* :: *pttrn* ⇒ ′*a set* ⇒ ′*b set* ⇒ ′*b set* (‹(*3*⋂(‹*unbreakable*›$_{-\in-}$)/ -)› [*0*, *0*, *10*] *10*)

**syntax-consts**
*-INTER1 -INTER* ⇌ *Inter*

**translations**
$\bigcap x\ y.\ f \rightleftharpoons \bigcap x.\ \bigcap y.\ f$
$\bigcap x.\ f \rightleftharpoons \bigcap(CONST\ range\ (\lambda x.\ f))$
$\bigcap x{\in}A.\ f \rightleftharpoons CONST\ Inter\ ((\lambda x.\ f)\ `\ A)$

**lemma** *INTER-eq*: $(\bigcap x{\in}A.\ B\ x) = \{y.\ \forall x{\in}A.\ y \in B\ x\}$
⟨*proof*⟩

**lemma** *INT-iff* [*simp*]: $b \in (\bigcap x{\in}A.\ B\ x) \longleftrightarrow (\forall x{\in}A.\ b \in B\ x)$
⟨*proof*⟩

**lemma** *INT-I* [*intro*!]: $(\bigwedge x.\ x \in A \Longrightarrow b \in B\ x) \Longrightarrow b \in (\bigcap x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *INT-D* [*elim*, *Pure.elim*]: $b \in (\bigcap x{\in}A.\ B\ x) \Longrightarrow a \in A \Longrightarrow b \in B\ a$
⟨*proof*⟩

**lemma** *INT-E* [*elim*]: $b \in (\bigcap x{\in}A.\ B\ x) \Longrightarrow (b \in B\ a \Longrightarrow R) \Longrightarrow (a \notin A \Longrightarrow R) \Longrightarrow R$
— "Classical" elimination – by the Excluded Middle on $a \in A$.

⟨*proof*⟩

**lemma** *Collect-ball-eq*: $\{x.\ \forall\, y{\in}A.\ P\ x\ y\} = (\bigcap y{\in}A.\ \{x.\ P\ x\ y\})$
⟨*proof*⟩

**lemma** *Collect-all-eq*: $\{x.\ \forall\, y.\ P\ x\ y\} = (\bigcap y.\ \{x.\ P\ x\ y\})$
⟨*proof*⟩

**lemma** *INT-lower*: $a \in A \Longrightarrow (\bigcap x{\in}A.\ B\ x) \subseteq B\ a$
⟨*proof*⟩

**lemma** *INT-greatest*: $(\bigwedge x.\ x \in A \Longrightarrow C \subseteq B\ x) \Longrightarrow C \subseteq (\bigcap x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *INT-empty*: $(\bigcap x{\in}\{\}.\ B\ x) = \mathit{UNIV}$
⟨*proof*⟩

**lemma** *INT-absorb*: $k \in I \Longrightarrow A\ k \cap (\bigcap i{\in}I.\ A\ i) = (\bigcap i{\in}I.\ A\ i)$
⟨*proof*⟩

**lemma** *INT-subset-iff*: $B \subseteq (\bigcap i{\in}I.\ A\ i) \longleftrightarrow (\forall\, i{\in}I.\ B \subseteq A\ i)$
⟨*proof*⟩

**lemma** *INT-insert* [*simp*]: $(\bigcap x \in \mathit{insert}\ a\ A.\ B\ x) = B\ a \cap \bigcap\ (B\ `\ A)$
⟨*proof*⟩

**lemma** *INT-Un*: $(\bigcap i \in A \cup B.\ M\ i) = (\bigcap i \in A.\ M\ i) \cap (\bigcap i{\in}B.\ M\ i)$
⟨*proof*⟩

**lemma** *INT-insert-distrib*: $u \in A \Longrightarrow (\bigcap x{\in}A.\ \mathit{insert}\ a\ (B\ x)) = \mathit{insert}\ a\ (\bigcap x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *INT-constant* [*simp*]: $(\bigcap y{\in}A.\ c) = (\mathit{if}\ A = \{\}\ \mathit{then}\ \mathit{UNIV}\ \mathit{else}\ c)$
⟨*proof*⟩

**lemma** *INTER-UNIV-conv*:
$(\mathit{UNIV} = (\bigcap x{\in}A.\ B\ x)) = (\forall\, x{\in}A.\ B\ x = \mathit{UNIV})$
$((\bigcap x{\in}A.\ B\ x) = \mathit{UNIV}) = (\forall\, x{\in}A.\ B\ x = \mathit{UNIV})$
⟨*proof*⟩

**lemma** *INT-bool-eq*: $(\bigcap b.\ A\ b) = A\ \mathit{True} \cap A\ \mathit{False}$
⟨*proof*⟩

**lemma** *INT-anti-mono*: $A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\, x \subseteq g\ x) \Longrightarrow (\bigcap x{\in}B.\ f\, x) \subseteq (\bigcap x{\in}A.\ g\ x)$
— The last inclusion is POSITIVE!
⟨*proof*⟩

**lemma** *Pow-INT-eq*: $Pow\ (\bigcap x{\in}A.\ B\ x) = (\bigcap x{\in}A.\ Pow\ (B\ x))$
⟨*proof*⟩

**lemma** *vimage-INT*: $f -`\ (\bigcap x{\in}A.\ B\ x) = (\bigcap x{\in}A.\ f -`\ B\ x)$
⟨*proof*⟩

### 11.6.3 Union

**abbreviation** *Union* :: $'a\ set\ set \Rightarrow 'a\ set$ (‹$\bigcup$›)
**where** $\bigcup S \equiv \bigsqcup S$

**lemma** *Union-eq*: $\bigcup A = \{x.\ \exists B \in A.\ x \in B\}$
⟨*proof*⟩

**lemma** *Union-iff* [*simp*]: $A \in \bigcup C \longleftrightarrow (\exists X{\in}C.\ A{\in}X)$
⟨*proof*⟩

**lemma** *UnionI* [*intro*]: $X \in C \Longrightarrow A \in X \Longrightarrow A \in \bigcup C$
— The order of the premises presupposes that $C$ is rigid; $A$ may be flexible.
⟨*proof*⟩

**lemma** *UnionE* [*elim*!]: $A \in \bigcup C \Longrightarrow (\bigwedge X.\ A \in X \Longrightarrow X \in C \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *Union-upper*: $B \in A \Longrightarrow B \subseteq \bigcup A$
⟨*proof*⟩

**lemma** *Union-least*: $(\bigwedge X.\ X \in A \Longrightarrow X \subseteq C) \Longrightarrow \bigcup A \subseteq C$
⟨*proof*⟩

**lemma** *Union-empty*: $\bigcup\{\} = \{\}$
⟨*proof*⟩

**lemma** *Union-UNIV*: $\bigcup UNIV = UNIV$
⟨*proof*⟩

**lemma** *Union-insert*: $\bigcup(insert\ a\ B) = a \cup \bigcup B$
⟨*proof*⟩

**lemma** *Union-Un-distrib* [*simp*]: $\bigcup(A \cup B) = \bigcup A \cup \bigcup B$
⟨*proof*⟩

**lemma** *Union-Int-subset*: $\bigcup(A \cap B) \subseteq \bigcup A \cap \bigcup B$
⟨*proof*⟩

**lemma** *Union-empty-conv*: $(\bigcup A = \{\}) \longleftrightarrow (\forall x{\in}A.\ x = \{\})$
⟨*proof*⟩

**lemma** *empty-Union-conv*: $(\{\} = \bigcup A) \longleftrightarrow (\forall x{\in}A.\ x = \{\})$

⟨*proof*⟩

**lemma** *subset-Pow-Union*: $A \subseteq Pow\ (\bigcup A)$
⟨*proof*⟩

**lemma** *Union-Pow-eq* [*simp*]: $\bigcup (Pow\ A) = A$
⟨*proof*⟩

**lemma** *Union-mono*: $A \subseteq B \Longrightarrow \bigcup A \subseteq \bigcup B$
⟨*proof*⟩

**lemma** *Union-subsetI*: $(\bigwedge x.\ x \in A \Longrightarrow \exists y.\ y \in B \wedge x \subseteq y) \Longrightarrow \bigcup A \subseteq \bigcup B$
⟨*proof*⟩

**lemma** *disjnt-inj-on-iff*:
$[\![inj\text{-}on\ f\ (\bigcup \mathcal{A});\ X \in \mathcal{A};\ Y \in \mathcal{A}]\!] \Longrightarrow disjnt\ (f\ `\ X)\ (f\ `\ Y) \longleftrightarrow disjnt\ X\ Y$
⟨*proof*⟩

**lemma** *disjnt-Union1* [*simp*]: $disjnt\ (\bigcup \mathcal{A})\ B \longleftrightarrow (\forall A \in \mathcal{A}.\ disjnt\ A\ B)$
⟨*proof*⟩

**lemma** *disjnt-Union2* [*simp*]: $disjnt\ B\ (\bigcup \mathcal{A}) \longleftrightarrow (\forall A \in \mathcal{A}.\ disjnt\ B\ A)$
⟨*proof*⟩

### 11.6.4 Unions of families

**syntax** (*ASCII*)
*-UNION1* :: *pttrns => 'b set => 'b set* (‹(‹indent=3 notation=‹binder UN››UN -./ -)› [0, 10] 10)
*-UNION* :: *pttrn => 'a set => 'b set => 'b set* (‹(‹indent=3 notation=‹binder UN››UN -:-./ -)› [0, 0, 10] 10)

**syntax**
*-UNION1* :: *pttrns => 'b set => 'b set* (‹(‹indent=3 notation=‹binder ⋃››⋃ -./ -)› [0, 10] 10)
*-UNION* :: *pttrn => 'a set => 'b set => 'b set* (‹(‹indent=3 notation=‹binder ⋃››⋃ -∈-./ -)› [0, 0, 10] 10)

**syntax** (*latex* **output**)
*-UNION1* :: *pttrns => 'b set => 'b set* (‹(3⋃(‹unbreakable›$_{-}$)/ -)› [0, 10] 10)
*-UNION* :: *pttrn => 'a set => 'b set => 'b set* (‹(3⋃(‹unbreakable›$_{-\in-}$)/ -)› [0, 0, 10] 10)

**syntax-consts**
*-UNION1 -UNION* $\rightleftharpoons$ *Union*

**translations**
$\bigcup x\ y.\ f \rightleftharpoons \bigcup x.\ \bigcup y.\ f$

$\bigcup x.\ f \quad \rightleftharpoons \bigcup(CONST\ range\ (\lambda x.\ f))$
$\bigcup x{\in}A.\ f \quad \rightleftharpoons CONST\ Union\ ((\lambda x.\ f)\ `\ A)$

Note the difference between ordinary syntax of indexed unions and intersections (e.g. $\bigcup a_1{\in}A_1.\ B$) and their LaTeX rendition: $\bigcup_{a_1 \in A_1} B$.

**lemma** *disjoint-UN-iff*: $disjnt\ A\ (\bigcup i{\in}I.\ B\ i) \longleftrightarrow (\forall\, i{\in}I.\ disjnt\ A\ (B\ i))$
⟨*proof*⟩

**lemma** *UNION-eq*: $(\bigcup x{\in}A.\ B\ x) = \{y.\ \exists\, x{\in}A.\ y \in B\ x\}$
⟨*proof*⟩

**lemma** *bind-UNION* [*code*]: $Set.bind\ A\ f = \bigcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *member-bind* [*simp*]: $x \in Set.bind\ A\ f \longleftrightarrow x \in \bigcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *Union-SetCompr-eq*: $\bigcup\{f\ x|\ x.\ P\ x\} = \{a.\ \exists\, x.\ P\ x \wedge a \in f\ x\}$
⟨*proof*⟩

**lemma** *UN-iff* [*simp*]: $b \in (\bigcup x{\in}A.\ B\ x) \longleftrightarrow (\exists\, x{\in}A.\ b \in B\ x)$
⟨*proof*⟩

**lemma** *UN-I* [*intro*]: $a \in A \Longrightarrow b \in B\ a \Longrightarrow b \in (\bigcup x{\in}A.\ B\ x)$
— The order of the premises presupposes that $A$ is rigid; $b$ may be flexible.
⟨*proof*⟩

**lemma** *UN-E* [*elim!*]: $b \in (\bigcup x{\in}A.\ B\ x) \Longrightarrow (\bigwedge x.\ x{\in}A \Longrightarrow b \in B\ x \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *UN-upper*: $a \in A \Longrightarrow B\ a \subseteq (\bigcup x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *UN-least*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq C) \Longrightarrow (\bigcup x{\in}A.\ B\ x) \subseteq C$
⟨*proof*⟩

**lemma** *Collect-bex-eq*: $\{x.\ \exists\, y{\in}A.\ P\ x\ y\} = (\bigcup y{\in}A.\ \{x.\ P\ x\ y\})$
⟨*proof*⟩

**lemma** *UN-insert-distrib*: $u \in A \Longrightarrow (\bigcup x{\in}A.\ insert\ a\ (B\ x)) = insert\ a\ (\bigcup x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *UN-empty*: $(\bigcup x{\in}\{\}.\ B\ x) = \{\}$
⟨*proof*⟩

**lemma** *UN-empty2*: $(\bigcup x{\in}A.\ \{\}) = \{\}$
⟨*proof*⟩

**lemma** *UN-absorb*: $k \in I \Longrightarrow A\ k \cup (\bigcup i{\in}I.\ A\ i) = (\bigcup i{\in}I.\ A\ i)$
⟨*proof*⟩

**lemma** *UN-insert* [*simp*]: $(\bigcup x{\in}\mathit{insert}\ a\ A.\ B\ x) = B\ a \cup \bigcup(B\ `\ A)$
⟨*proof*⟩

**lemma** *UN-Un* [*simp*]: $(\bigcup i \in A \cup B.\ M\ i) = (\bigcup i{\in}A.\ M\ i) \cup (\bigcup i{\in}B.\ M\ i)$
⟨*proof*⟩

**lemma** *UN-UN-flatten*: $(\bigcup x \in (\bigcup y{\in}A.\ B\ y).\ C\ x) = (\bigcup y{\in}A.\ \bigcup x{\in}B\ y.\ C\ x)$
⟨*proof*⟩

**lemma** *UN-subset-iff*: $((\bigcup i{\in}I.\ A\ i) \subseteq B) = (\forall\, i{\in}I.\ A\ i \subseteq B)$
⟨*proof*⟩

**lemma** *UN-constant* [*simp*]: $(\bigcup y{\in}A.\ c) = (\mathit{if}\ A = \{\}\ \mathit{then}\ \{\}\ \mathit{else}\ c)$
⟨*proof*⟩

**lemma** *UNION-singleton-eq-range*: $(\bigcup x{\in}A.\ \{f\ x\}) = f\ `\ A$
⟨*proof*⟩

**lemma** *image-Union*: $f\ `\ \bigcup S = (\bigcup x{\in}S.\ f\ `\ x)$
⟨*proof*⟩

**lemma** *UNION-empty-conv*:
$\{\} = (\bigcup x{\in}A.\ B\ x) \longleftrightarrow (\forall\, x{\in}A.\ B\ x = \{\})$
$(\bigcup x{\in}A.\ B\ x) = \{\} \longleftrightarrow (\forall\, x{\in}A.\ B\ x = \{\})$
⟨*proof*⟩

**lemma** *Collect-ex-eq*: $\{x.\ \exists\, y.\ P\ x\ y\} = (\bigcup y.\ \{x.\ P\ x\ y\})$
⟨*proof*⟩

**lemma** *ball-UN*: $(\forall\, z \in \bigcup(B\ `\ A).\ P\ z) \longleftrightarrow (\forall\, x{\in}A.\ \forall\, z \in B\ x.\ P\ z)$
⟨*proof*⟩

**lemma** *bex-UN*: $(\exists\, z \in \bigcup(B\ `\ A).\ P\ z) \longleftrightarrow (\exists\, x{\in}A.\ \exists\, z{\in}B\ x.\ P\ z)$
⟨*proof*⟩

**lemma** *Un-eq-UN*: $A \cup B = (\bigcup b.\ \mathit{if}\ b\ \mathit{then}\ A\ \mathit{else}\ B)$
⟨*proof*⟩

**lemma** *UN-bool-eq*: $(\bigcup b.\ A\ b) = (A\ \mathit{True} \cup A\ \mathit{False})$
⟨*proof*⟩

**lemma** *UN-Pow-subset*: $(\bigcup x{\in}A.\ \mathit{Pow}\ (B\ x)) \subseteq \mathit{Pow}\ (\bigcup x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *UN-mono*:

$A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\,x \subseteq g\ x) \Longrightarrow$
$(\bigcup x{\in}A.\ f\,x) \subseteq (\bigcup x{\in}B.\ g\ x)$
⟨*proof*⟩

**lemma** *vimage-Union*: $f -` (\bigcup A) = (\bigcup X{\in}A.\ f -` X)$
⟨*proof*⟩

**lemma** *vimage-UN*: $f -` (\bigcup x{\in}A.\ B\ x) = (\bigcup x{\in}A.\ f -` B\ x)$
⟨*proof*⟩

**lemma** *vimage-eq-UN*: $f -` B = (\bigcup y{\in}B.\ f -` \{y\})$
— NOT suitable for rewriting
⟨*proof*⟩

**lemma** *image-UN*: $f\ `\ \bigcup (B\ `\ A) = (\bigcup x{\in}A.\ f\ `\ B\ x)$
⟨*proof*⟩

**lemma** *UN-singleton* [*simp*]: $(\bigcup x{\in}A.\ \{x\}) = A$
⟨*proof*⟩

**lemma** *inj-on-image*: *inj-on* $f\ (\bigcup A) \Longrightarrow$ *inj-on* $((`)\ f)\ A$
⟨*proof*⟩

### 11.6.5 Distributive laws

**lemma** *Int-Union*: $A \cap \bigcup B = (\bigcup C{\in}B.\ A \cap C)$
⟨*proof*⟩

**lemma** *Un-Inter*: $A \cup \bigcap B = (\bigcap C{\in}B.\ A \cup C)$
⟨*proof*⟩

**lemma** *Int-Union2*: $\bigcup B \cap A = (\bigcup C{\in}B.\ C \cap A)$
⟨*proof*⟩

**lemma** *INT-Int-distrib*: $(\bigcap i{\in}I.\ A\ i \cap B\ i) = (\bigcap i{\in}I.\ A\ i) \cap (\bigcap i{\in}I.\ B\ i)$
⟨*proof*⟩

**lemma** *UN-Un-distrib*: $(\bigcup i{\in}I.\ A\ i \cup B\ i) = (\bigcup i{\in}I.\ A\ i) \cup (\bigcup i{\in}I.\ B\ i)$
⟨*proof*⟩

**lemma** *Int-Inter-image*: $(\bigcap x{\in}C.\ A\ x \cap B\ x) = \bigcap (A\ `\ C) \cap \bigcap (B\ `\ C)$
⟨*proof*⟩

**lemma** *Int-Inter-eq*: $A \cap \bigcap \mathcal{B} = (\textit{if}\ \mathcal{B}{=}\{\}\ \textit{then}\ A\ \textit{else}\ (\bigcap B{\in}\mathcal{B}.\ A \cap B))$
$\bigcap \mathcal{B} \cap A = (\textit{if}\ \mathcal{B}{=}\{\}\ \textit{then}\ A\ \textit{else}\ (\bigcap B{\in}\mathcal{B}.\ B \cap A))$
⟨*proof*⟩

**lemma** *Un-Union-image*: $(\bigcup x{\in}C.\ A\ x \cup B\ x) = \bigcup (A\ `\ C) \cup \bigcup (B\ `\ C)$
— Devlin, Fundamentals of Contemporary Set Theory, page 12, exercise 5:

— Union of a family of unions
⟨*proof*⟩

**lemma** *Un-INT-distrib*: $B \cup (\bigcap i{\in}I.\ A\ i) = (\bigcap i{\in}I.\ B \cup A\ i)$
⟨*proof*⟩

**lemma** *Int-UN-distrib*: $B \cap (\bigcup i{\in}I.\ A\ i) = (\bigcup i{\in}I.\ B \cap A\ i)$
— Halmos, Naive Set Theory, page 35.
⟨*proof*⟩

**lemma** *Int-UN-distrib2*: $(\bigcup i{\in}I.\ A\ i) \cap (\bigcup j{\in}J.\ B\ j) = (\bigcup i{\in}I.\ \bigcup j{\in}J.\ A\ i \cap B\ j)$
⟨*proof*⟩

**lemma** *Un-INT-distrib2*: $(\bigcap i{\in}I.\ A\ i) \cup (\bigcap j{\in}J.\ B\ j) = (\bigcap i{\in}I.\ \bigcap j{\in}J.\ A\ i \cup B\ j)$
⟨*proof*⟩

**lemma** *Union-disjoint*: $(\bigcup C \cap A = \{\}) \longleftrightarrow (\forall B{\in}C.\ B \cap A = \{\})$
⟨*proof*⟩

**lemma** *SUP-UNION*: $(\bigsqcup x{\in}(\bigcup y{\in}A.\ g\ y).\ f\ x) = (\bigsqcup y{\in}A.\ \bigsqcup x{\in}g\ y.\ f\ x$ :: - :: *complete-lattice*)
⟨*proof*⟩

## 11.7 Injections and bijections

**lemma** *inj-on-Inter*: $S \neq \{\} \Longrightarrow (\bigwedge A.\ A \in S \Longrightarrow$ *inj-on f A*$) \Longrightarrow$ *inj-on f* $(\bigcap S)$
⟨*proof*⟩

**lemma** *inj-on-INTER*: $I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow$ *inj-on f* $(A\ i)) \Longrightarrow$ *inj-on f* $(\bigcap i \in I.\ A\ i)$
⟨*proof*⟩

**lemma** *inj-on-UNION-chain*:
**assumes** *chain*: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow A\ i \leq A\ j \vee A\ j \leq A\ i$
**and** *inj*: $\bigwedge i.\ i \in I \Longrightarrow$ *inj-on f* $(A\ i)$
**shows** *inj-on f* $(\bigcup i \in I.\ A\ i)$
⟨*proof*⟩

**lemma** *bij-betw-UNION-chain*:
**assumes** *chain*: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow A\ i \leq A\ j \vee A\ j \leq A\ i$
**and** *bij*: $\bigwedge i.\ i \in I \Longrightarrow$ *bij-betw f* $(A\ i)\ (A'\ i)$
**shows** *bij-betw f* $(\bigcup i \in I.\ A\ i)\ (\bigcup i \in I.\ A'\ i)$
⟨*proof*⟩

**lemma** *image-INT*: *inj-on f C* $\Longrightarrow \forall x{\in}A.\ B\ x \subseteq C \Longrightarrow j \in A \Longrightarrow f\ `\ (\bigcap (B\ `\ A)) = (\bigcap x{\in}A.\ f\ `\ B\ x)$

⟨*proof*⟩

**lemma** *bij-image-INT*: $bij\ f \Longrightarrow f\ `\ (\bigcap (B\ `\ A)) = (\bigcap x\in A.\ f\ `\ B\ x)$
⟨*proof*⟩

**lemma** *UNION-fun-upd*: $\bigcup (A(i := B)\ `\ J) = \bigcup (A\ `\ (J - \{i\})) \cup (if\ i \in J\ then\ B\ else\ \{\})$
⟨*proof*⟩

**lemma** *bij-betw-Pow*:
**assumes** *bij-betw f A B*
**shows** *bij-betw* (*image f*) (*Pow A*) (*Pow B*)
⟨*proof*⟩

### 11.7.1 Complement

**lemma** *Compl-INT* [*simp*]: $-\ (\bigcap x\in A.\ B\ x) = (\bigcup x\in A.\ -B\ x)$
⟨*proof*⟩

**lemma** *Compl-UN* [*simp*]: $-\ (\bigcup x\in A.\ B\ x) = (\bigcap x\in A.\ -B\ x)$
⟨*proof*⟩

### 11.7.2 Miniscoping and maxiscoping

Miniscoping: pushing in quantifiers and big Unions and Intersections.

**lemma** *UN-simps* [*simp*]:
$\bigwedge a\ B\ C.\ (\bigcup x\in C.\ insert\ a\ (B\ x)) = (if\ C=\{\}\ then\ \{\}\ else\ insert\ a\ (\bigcup x\in C.\ B\ x))$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A\ x \cup B) = ((if\ C=\{\}\ then\ \{\}\ else\ (\bigcup x\in C.\ A\ x) \cup B))$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A \cup B\ x) = ((if\ C=\{\}\ then\ \{\}\ else\ A \cup (\bigcup x\in C.\ B\ x)))$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A\ x \cap B) = ((\bigcup x\in C.\ A\ x) \cap B)$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A \cap B\ x) = (A \cap(\bigcup x\in C.\ B\ x))$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A\ x - B) = ((\bigcup x\in C.\ A\ x) - B)$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A - B\ x) = (A - (\bigcap x\in C.\ B\ x))$
$\bigwedge A\ B.\ (\bigcup x\in\bigcup A.\ B\ x) = (\bigcup y\in A.\ \bigcup x\in y.\ B\ x)$
$\bigwedge A\ B\ C.\ (\bigcup z\in(\bigcup (B\ `\ A)).\ C\ z) = (\bigcup x\in A.\ \bigcup z\in B\ x.\ C\ z)$
$\bigwedge A\ B\ f.\ (\bigcup x\in f`A.\ B\ x) = (\bigcup a\in A.\ B\ (f\ a))$
⟨*proof*⟩

**lemma** *INT-simps* [*simp*]:
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A\ x \cap B) = (if\ C=\{\}\ then\ UNIV\ else\ (\bigcap x\in C.\ A\ x) \cap B)$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A \cap B\ x) = (if\ C=\{\}\ then\ UNIV\ else\ A \cap(\bigcap x\in C.\ B\ x))$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A\ x - B) = (if\ C=\{\}\ then\ UNIV\ else\ (\bigcap x\in C.\ A\ x) - B)$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A - B\ x) = (if\ C=\{\}\ then\ UNIV\ else\ A - (\bigcup x\in C.\ B\ x))$
$\bigwedge a\ B\ C.\ (\bigcap x\in C.\ insert\ a\ (B\ x)) = insert\ a\ (\bigcap x\in C.\ B\ x)$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A\ x \cup B) = ((\bigcap x\in C.\ A\ x) \cup B)$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A \cup B\ x) = (A \cup (\bigcap x\in C.\ B\ x))$
$\bigwedge A\ B.\ (\bigcap x\in\bigcup A.\ B\ x) = (\bigcap y\in A.\ \bigcap x\in y.\ B\ x)$
$\bigwedge A\ B\ C.\ (\bigcap z\in(\bigcup (B\ `\ A)).\ C\ z) = (\bigcap x\in A.\ \bigcap z\in B\ x.\ C\ z)$

$\bigwedge A\ B\ f.\ (\bigcap x\in f`A.\ B\ x) = (\bigcap a\in A.\ B\ (f\ a))$
⟨*proof*⟩

**lemma** *UN-ball-bex-simps* [*simp*]:
$\bigwedge A\ P.\ (\forall x\in\bigcup A.\ P\ x) \longleftrightarrow (\forall y\in A.\ \forall x\in y.\ P\ x)$
$\bigwedge A\ B\ P.\ (\forall x\in(\bigcup (B\ `\ A)).\ P\ x) = (\forall a\in A.\ \forall x\in\ B\ a.\ P\ x)$
$\bigwedge A\ P.\ (\exists x\in\bigcup A.\ P\ x) \longleftrightarrow (\exists y\in A.\ \exists x\in y.\ P\ x)$
$\bigwedge A\ B\ P.\ (\exists x\in(\bigcup (B\ `\ A)).\ P\ x) \longleftrightarrow (\exists a\in A.\ \exists x\in B\ a.\ P\ x)$
⟨*proof*⟩

Maxiscoping: pulling out big Unions and Intersections.

**lemma** *UN-extend-simps*:
$\bigwedge a\ B\ C.\ insert\ a\ (\bigcup x\in C.\ B\ x) = (if\ C=\{\}\ then\ \{a\}\ else\ (\bigcup x\in C.\ insert\ a\ (B\ x)))$
$\bigwedge A\ B\ C.\ (\bigcup x\in C.\ A\ x) \cup B = (if\ C=\{\}\ then\ B\ else\ (\bigcup x\in C.\ A\ x \cup B))$
$\bigwedge A\ B\ C.\ A \cup (\bigcup x\in C.\ B\ x) = (if\ C=\{\}\ then\ A\ else\ (\bigcup x\in C.\ A \cup B\ x))$
$\bigwedge A\ B\ C.\ ((\bigcup x\in C.\ A\ x) \cap B) = (\bigcup x\in C.\ A\ x \cap B)$
$\bigwedge A\ B\ C.\ (A \cap (\bigcup x\in C.\ B\ x)) = (\bigcup x\in C.\ A \cap B\ x)$
$\bigwedge A\ B\ C.\ ((\bigcup x\in C.\ A\ x) - B) = (\bigcup x\in C.\ A\ x - B)$
$\bigwedge A\ B\ C.\ (A - (\bigcap x\in C.\ B\ x)) = (\bigcup x\in C.\ A - B\ x)$
$\bigwedge A\ B.\ (\bigcup y\in A.\ \bigcup x\in y.\ B\ x) = (\bigcup x\in\bigcup A.\ B\ x)$
$\bigwedge A\ B\ C.\ (\bigcup x\in A.\ \bigcup z\in B\ x.\ C\ z) = (\bigcup z\in(\bigcup (B\ `\ A)).\ C\ z)$
$\bigwedge A\ B\ f.\ (\bigcup a\in A.\ B\ (f\ a)) = (\bigcup x\in f`A.\ B\ x)$
⟨*proof*⟩

**lemma** *INT-extend-simps*:
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A\ x) \cap B = (if\ C=\{\}\ then\ B\ else\ (\bigcap x\in C.\ A\ x \cap B))$
$\bigwedge A\ B\ C.\ A \cap (\bigcap x\in C.\ B\ x) = (if\ C=\{\}\ then\ A\ else\ (\bigcap x\in C.\ A \cap B\ x))$
$\bigwedge A\ B\ C.\ (\bigcap x\in C.\ A\ x) - B = (if\ C=\{\}\ then\ UNIV - B\ else\ (\bigcap x\in C.\ A\ x - B))$
$\bigwedge A\ B\ C.\ A - (\bigcup x\in C.\ B\ x) = (if\ C=\{\}\ then\ A\ else\ (\bigcap x\in C.\ A - B\ x))$
$\bigwedge a\ B\ C.\ insert\ a\ (\bigcap x\in C.\ B\ x) = (\bigcap x\in C.\ insert\ a\ (B\ x))$
$\bigwedge A\ B\ C.\ ((\bigcap x\in C.\ A\ x) \cup B) = (\bigcap x\in C.\ A\ x \cup B)$
$\bigwedge A\ B\ C.\ A \cup (\bigcap x\in C.\ B\ x) = (\bigcap x\in C.\ A \cup B\ x)$
$\bigwedge A\ B.\ (\bigcap y\in A.\ \bigcap x\in y.\ B\ x) = (\bigcap x\in\bigcup A.\ B\ x)$
$\bigwedge A\ B\ C.\ (\bigcap x\in A.\ \bigcap z\in B\ x.\ C\ z) = (\bigcap z\in(\bigcup (B\ `\ A)).\ C\ z)$
$\bigwedge A\ B\ f.\ (\bigcap a\in A.\ B\ (f\ a)) = (\bigcap x\in f`A.\ B\ x)$
⟨*proof*⟩

Finally

**lemmas** *mem-simps* =
*insert-iff empty-iff Un-iff Int-iff Compl-iff Diff-iff*
*mem-Collect-eq UN-iff Union-iff INT-iff Inter-iff*
— Each of these has ALREADY been added [*simp*] above.

**end**

# 12 Wrapping Existing Freely Generated Type's Constructors

**theory** *Ctr-Sugar*
**imports** *HOL*
**keywords**
*print-case-translations* :: *diag* **and**
*free-constructors* :: *thy-goal*
**begin**

**consts**
$case\text{-}guard :: bool \Rightarrow {}'a \Rightarrow ({}'a \Rightarrow {}'b) \Rightarrow {}'b$
$case\text{-}nil :: {}'a \Rightarrow {}'b$
$case\text{-}cons :: ({}'a \Rightarrow {}'b) \Rightarrow ({}'a \Rightarrow {}'b) \Rightarrow {}'a \Rightarrow {}'b$
$case\text{-}elem :: {}'a \Rightarrow {}'b \Rightarrow {}'a \Rightarrow {}'b$
$case\text{-}abs :: ({}'c \Rightarrow {}'b) \Rightarrow {}'b$

**declare** [[*coercion-args case-guard* − + −]]
**declare** [[*coercion-args case-cons* − −]]
**declare** [[*coercion-args case-abs* −]]
**declare** [[*coercion-args case-elem* − +]]

⟨*ML*⟩

**lemma** *iffI-np*: $[\![x \Longrightarrow \neg\ y; \neg\ x \Longrightarrow y]\!] \Longrightarrow \neg\ x \longleftrightarrow y$
⟨*proof*⟩

**lemma** *iff-contradict*:
$\neg\ P \Longrightarrow P \longleftrightarrow Q \Longrightarrow Q \Longrightarrow R$
$\neg\ Q \Longrightarrow P \longleftrightarrow Q \Longrightarrow P \Longrightarrow R$
⟨*proof*⟩

⟨*ML*⟩

Coinduction method that avoids some boilerplate compared with coinduct.

⟨*ML*⟩

**end**

# 13 Knaster-Tarski Fixpoint Theorem and inductive definitions

**theory** *Inductive*
**imports** *Complete-Lattices Ctr-Sugar*
**keywords**
*inductive coinductive inductive-cases inductive-simps* :: *thy-defn* **and**
*monos* **and**
*print-inductives* :: *diag* **and**

*old-rep-datatype* :: *thy-goal* **and**
*primrec* :: *thy-defn*
**begin**

## 13.1 Least fixed points

**context** *complete-lattice*
**begin**

**definition** $lfp :: ('a \Rightarrow 'a) \Rightarrow 'a$
**where** $lfp\ f = Inf\ \{u.\ f\ u \leq u\}$

**lemma** *lfp-lowerbound*: $f\ A \leq A \Longrightarrow lfp\ f \leq A$
⟨*proof*⟩

**lemma** *lfp-greatest*: $(\bigwedge u.\ f\ u \leq u \Longrightarrow A \leq u) \Longrightarrow A \leq lfp\ f$
⟨*proof*⟩

**end**

**lemma** *lfp-fixpoint*:
**assumes** *mono f*
**shows** $f\ (lfp\ f) = lfp\ f$
⟨*proof*⟩

**lemma** *lfp-unfold*: $mono\ f \Longrightarrow lfp\ f = f\ (lfp\ f)$
⟨*proof*⟩

**lemma** *lfp-const*: $lfp\ (\lambda x.\ t) = t$
⟨*proof*⟩

**lemma** *lfp-eqI*: $mono\ F \Longrightarrow F\ x = x \Longrightarrow (\bigwedge z.\ F\ z = z \Longrightarrow x \leq z) \Longrightarrow lfp\ F = x$
⟨*proof*⟩

## 13.2 General induction rules for least fixed points

**lemma** *lfp-ordinal-induct* [*case-names mono step union*]:
**fixes** $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
**assumes** *mono*: *mono f*
**and** *P-f*: $\bigwedge S.\ P\ S \Longrightarrow S \leq lfp\ f \Longrightarrow P\ (f\ S)$
**and** *P-Union*: $\bigwedge M.\ \forall S{\in}M.\ P\ S \Longrightarrow P\ (Sup\ M)$
**shows** $P\ (lfp\ f)$
⟨*proof*⟩

**theorem** *lfp-induct*:
**assumes** *mono*: *mono f*
**and** *ind*: $f\ (inf\ (lfp\ f)\ P) \leq P$
**shows** $lfp\ f \leq P$
⟨*proof*⟩

**lemma** *lfp-induct-set*:
**assumes** *lfp*: $a \in lfp\ f$
**and** *mono*: *mono f*
**and** *hyp*: $\bigwedge x.\ x \in f\ (lfp\ f \cap \{x.\ P\ x\}) \Longrightarrow P\ x$
**shows** $P\ a$
⟨*proof*⟩

**lemma** *lfp-ordinal-induct-set*:
**assumes** *mono*: *mono f*
**and** *P-f*: $\bigwedge S.\ P\ S \Longrightarrow P\ (f\ S)$
**and** *P-Union*: $\bigwedge M.\ \forall S{\in}M.\ P\ S \Longrightarrow P\ (\bigcup M)$
**shows** $P\ (lfp\ f)$
⟨*proof*⟩

Definition forms of *lfp-unfold* and *lfp-induct*, to control unfolding.

**lemma** *def-lfp-unfold*: $h \equiv lfp\ f \Longrightarrow mono\ f \Longrightarrow h = f\ h$
⟨*proof*⟩

**lemma** *def-lfp-induct*: $A \equiv lfp\ f \Longrightarrow mono\ f \Longrightarrow f\ (inf\ A\ P) \le P \Longrightarrow A \le P$
⟨*proof*⟩

**lemma** *def-lfp-induct-set*:
$A \equiv lfp\ f \Longrightarrow mono\ f \Longrightarrow a \in A \Longrightarrow (\bigwedge x.\ x \in f\ (A \cap \{x.\ P\ x\}) \Longrightarrow P\ x) \Longrightarrow P\ a$
⟨*proof*⟩

Monotonicity of *lfp*!

**lemma** *lfp-mono*: $(\bigwedge Z.\ f\ Z \le g\ Z) \Longrightarrow lfp\ f \le lfp\ g$
⟨*proof*⟩

## 13.3 Greatest fixed points

**context** *complete-lattice*
**begin**

**definition** *gfp* :: $('a \Rightarrow 'a) \Rightarrow 'a$
**where** $gfp\ f = Sup\ \{u.\ u \le f\ u\}$

**lemma** *gfp-upperbound*: $X \le f\ X \Longrightarrow X \le gfp\ f$
⟨*proof*⟩

**lemma** *gfp-least*: $(\bigwedge u.\ u \le f\ u \Longrightarrow u \le X) \Longrightarrow gfp\ f \le X$
⟨*proof*⟩

**end**

**lemma** *lfp-le-gfp*: $mono\ f \Longrightarrow lfp\ f \le gfp\ f$
⟨*proof*⟩

**lemma** *gfp-fixpoint*:
**assumes** *mono f*
**shows** $f\ (gfp\ f) = gfp\ f$
⟨*proof*⟩

**lemma** *gfp-unfold*: $mono\ f \Longrightarrow gfp\ f = f\ (gfp\ f)$
⟨*proof*⟩

**lemma** *gfp-const*: $gfp\ (\lambda x.\ t) = t$
⟨*proof*⟩

**lemma** *gfp-eqI*: $mono\ F \Longrightarrow F\ x = x \Longrightarrow (\bigwedge z.\ F\ z = z \Longrightarrow z \leq x) \Longrightarrow gfp\ F = x$
⟨*proof*⟩

## 13.4 Coinduction rules for greatest fixed points

Weak version.

**lemma** *weak-coinduct*: $a \in X \Longrightarrow X \subseteq f\ X \Longrightarrow a \in gfp\ f$
⟨*proof*⟩

**lemma** *weak-coinduct-image*: $a \in X \Longrightarrow g`X \subseteq f\ (g`X) \Longrightarrow g\ a \in gfp\ f$
⟨*proof*⟩

**lemma** *coinduct-lemma*: $X \leq f\ (sup\ X\ (gfp\ f)) \Longrightarrow mono\ f \Longrightarrow sup\ X\ (gfp\ f) \leq f\ (sup\ X\ (gfp\ f))$
⟨*proof*⟩

Strong version, thanks to Coen and Frost.

**lemma** *coinduct-set*: $mono\ f \Longrightarrow a \in X \Longrightarrow X \subseteq f\ (X \cup gfp\ f) \Longrightarrow a \in gfp\ f$
⟨*proof*⟩

**lemma** *gfp-fun-UnI2*: $mono\ f \Longrightarrow a \in gfp\ f \Longrightarrow a \in f\ (X \cup gfp\ f)$
⟨*proof*⟩

**lemma** *gfp-ordinal-induct*[*case-names mono step union*]:
**fixes** $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
**assumes** *mono*: *mono f*
**and** *P-f*: $\bigwedge S.\ P\ S \Longrightarrow gfp\ f \leq S \Longrightarrow P\ (f\ S)$
**and** *P-Union*: $\bigwedge M.\ \forall S{\in}M.\ P\ S \Longrightarrow P\ (Inf\ M)$
**shows** $P\ (gfp\ f)$
⟨*proof*⟩

**lemma** *coinduct*:
**assumes** *mono*: *mono f*
**and** *ind*: $X \leq f\ (sup\ X\ (gfp\ f))$
**shows** $X \leq gfp\ f$
⟨*proof*⟩

## 13.5 Even Stronger Coinduction Rule, by Martin Coen

Weakens the condition $X \subseteq f\ X$ to one expressed using both *lfp* and *gfp*

**lemma** *coinduct3-mono-lemma*: $mono\ f \Longrightarrow mono\ (\lambda x.\ f\ x \cup X \cup B)$
⟨*proof*⟩

**lemma** *coinduct3-lemma*:
$X \subseteq f\ (lfp\ (\lambda x.\ f\ x \cup X \cup gfp\ f)) \Longrightarrow mono\ f \Longrightarrow$
$lfp\ (\lambda x.\ f\ x \cup X \cup gfp\ f) \subseteq f\ (lfp\ (\lambda x.\ f\ x \cup X \cup gfp\ f))$
⟨*proof*⟩

**lemma** *coinduct3*: $mono\ f \Longrightarrow a \in X \Longrightarrow X \subseteq f\ (lfp\ (\lambda x.\ f\ x \cup X \cup gfp\ f)) \Longrightarrow a \in gfp\ f$
⟨*proof*⟩

Definition forms of *gfp-unfold* and *coinduct*, to control unfolding.

**lemma** *def-gfp-unfold*: $A \equiv gfp\ f \Longrightarrow mono\ f \Longrightarrow A = f\ A$
⟨*proof*⟩

**lemma** *def-coinduct*: $A \equiv gfp\ f \Longrightarrow mono\ f \Longrightarrow X \leq f\ (sup\ X\ A) \Longrightarrow X \leq A$
⟨*proof*⟩

**lemma** *def-coinduct-set*: $A \equiv gfp\ f \Longrightarrow mono\ f \Longrightarrow a \in X \Longrightarrow X \subseteq f\ (X \cup A) \Longrightarrow a \in A$
⟨*proof*⟩

**lemma** *def-Collect-coinduct*:
$A \equiv gfp\ (\lambda w.\ Collect\ (P\ w)) \Longrightarrow mono\ (\lambda w.\ Collect\ (P\ w)) \Longrightarrow a \in X \Longrightarrow$
$(\bigwedge z.\ z \in X \Longrightarrow P\ (X \cup A)\ z) \Longrightarrow a \in A$
⟨*proof*⟩

**lemma** *def-coinduct3*: $A \equiv gfp\ f \Longrightarrow mono\ f \Longrightarrow a \in X \Longrightarrow X \subseteq f\ (lfp\ (\lambda x.\ f\ x \cup X \cup A)) \Longrightarrow a \in A$
⟨*proof*⟩

Monotonicity of *gfp*!

**lemma** *gfp-mono*: $(\bigwedge Z.\ f\ Z \leq g\ Z) \Longrightarrow gfp\ f \leq gfp\ g$
⟨*proof*⟩

## 13.6 Rules for fixed point calculus

**lemma** *lfp-rolling*:
**assumes** $mono\ g\ mono\ f$
**shows** $g\ (lfp\ (\lambda x.\ f\ (g\ x))) = lfp\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

**lemma** *lfp-lfp*:
**assumes** $f$: $\bigwedge x\ y\ w\ z.\ x \leq y \Longrightarrow w \leq z \Longrightarrow f\ x\ w \leq f\ y\ z$
**shows** $lfp\ (\lambda x.\ lfp\ (f\ x)) = lfp\ (\lambda x.\ f\ x\ x)$

⟨*proof*⟩

**lemma** *gfp-rolling*:
**assumes** *mono g mono f*
**shows** $g\ (gfp\ (\lambda x.\ f\ (g\ x))) = gfp\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

**lemma** *gfp-gfp*:
**assumes** $f$: $\bigwedge x\ y\ w\ z.\ x \le y \Longrightarrow w \le z \Longrightarrow f\ x\ w \le f\ y\ z$
**shows** $gfp\ (\lambda x.\ gfp\ (f\ x)) = gfp\ (\lambda x.\ f\ x\ x)$
⟨*proof*⟩

## 13.7 Inductive predicates and sets

Package setup.

**lemmas** *basic-monos* =
*subset-refl imp-refl disj-mono conj-mono ex-mono all-mono if-bool-eq-conj*
*Collect-mono in-mono vimage-mono*

**lemma** *le-rel-bool-arg-iff*: $X \le Y \longleftrightarrow X\ False \le Y\ False \wedge X\ True \le Y\ True$
⟨*proof*⟩

**lemma** *imp-conj-iff*: $((P \longrightarrow Q) \wedge P) = (P \wedge Q)$
⟨*proof*⟩

**lemma** *meta-fun-cong*: $P \equiv Q \Longrightarrow P\ a \equiv Q\ a$
⟨*proof*⟩

⟨*ML*⟩

**lemmas** [*mono*] =
*imp-refl disj-mono conj-mono ex-mono all-mono if-bool-eq-conj*
*imp-mono not-mono*
*Ball-def Bex-def*
*induct-rulify-fallback*

## 13.8 The Schroeder-Bernstein Theorem

See also:

- `$ISABELLE_HOME/src/HOL/ex/Set_Theory.thy`
- http://planetmath.org/proofofschroederbernsteintheoremusingtarskiknastertheorem
- Springer LNCS 828 (cover page)

**theorem** *Schroeder-Bernstein*:
**fixes** $f :: 'a \Rightarrow 'b$ **and** $g :: 'b \Rightarrow 'a$

**and** *A* :: *'a set* **and** *B* :: *'b set*
**assumes** *inj1*: *inj-on f A* **and** *sub1*: *f ' A* ⊆ *B*
**and** *inj2*: *inj-on g B* **and** *sub2*: *g ' B* ⊆ *A*
**shows** ∃ *h. bij-betw h A B*
⟨*proof*⟩

### 13.9 Inductive datatypes and primitive recursion

Package setup.

⟨*ML*⟩

Lambda-abstractions with pattern matching:

**syntax** (*ASCII*)
*-lam-pats-syntax* :: *cases-syn* ⇒ *'a* ⇒ *'b* (‹(‹*notation=abstraction*›%-)› *10*)
**syntax**
*-lam-pats-syntax* :: *cases-syn* ⇒ *'a* ⇒ *'b* (‹(‹*notation=abstraction*›λ-)› *10*)
⟨*ML*⟩

**end**

# 14 Cartesian products

**theory** *Product-Type*
**imports** *Typedef Inductive Fun*
**keywords** *inductive-set coinductive-set* :: *thy-defn*
**begin**

## 14.1 *bool* is a datatype

**free-constructors** (*discs-sels*) *case-bool* **for** *True* | *False*
⟨*proof*⟩

Avoid name clashes by prefixing the output of *old-rep-datatype* with *old*.

⟨*ML*⟩

**old-rep-datatype** *True False* ⟨*proof*⟩

⟨*ML*⟩

But erase the prefix for properties that are not generated by *free-constructors*.

⟨*ML*⟩

**lemmas** *induct* = *old.bool.induct*
**lemmas** *inducts* = *old.bool.inducts*
**lemmas** *rec* = *old.bool.rec*
**lemmas** *simps* = *bool.distinct bool.case bool.rec*

⟨*ML*⟩

**declare** *case-split* [*cases type*: *bool*]
— prefer plain propositional version

**lemma** [*code*]:
$HOL.equal\ False\ P \longleftrightarrow \neg\ P$
$HOL.equal\ True\ P \longleftrightarrow P$
$HOL.equal\ P\ False \longleftrightarrow \neg\ P$
$HOL.equal\ P\ True \longleftrightarrow P$
⟨*proof*⟩

**lemma** [*code nbe*]:
$HOL.equal\ P\ P \longleftrightarrow True$ **for** $P :: bool$
⟨*proof*⟩

**lemma** *If-case-cert*:
**assumes** $CASE \equiv (\lambda b.\ If\ b\ f\ g)$
**shows** $(CASE\ True \equiv f)\ \&\&\&\ (CASE\ False \equiv g)$
⟨*proof*⟩

⟨*ML*⟩

**code-printing**
**constant** $HOL.equal :: bool \Rightarrow bool \Rightarrow bool \rightharpoonup$ (*Haskell*) **infix** *4* ==
| **class-instance** $bool :: equal \rightharpoonup$ (*Haskell*) −

## 14.2 The *unit* type

**typedef** $unit = \{True\}$
⟨*proof*⟩

**definition** *Unity* :: *unit* (‹′(′)›)
**where** () = *Abs-unit True*

**lemma** *unit-eq* [*no-atp*]: *u* = ()
⟨*proof*⟩

Simplification procedure for *unit-eq*. Cannot use this rule directly — it loops!

⟨*ML*⟩

**free-constructors** *case-unit* **for** ()
⟨*proof*⟩

Avoid name clashes by prefixing the output of *old-rep-datatype* with *old*.

⟨*ML*⟩

**old-rep-datatype** () ⟨*proof*⟩

⟨*ML*⟩

But erase the prefix for properties that are not generated by *free-constructors*.

⟨*ML*⟩

**lemmas** *induct* = *old.unit.induct*
**lemmas** *inducts* = *old.unit.inducts*
**lemmas** *rec* = *old.unit.rec*
**lemmas** *simps* = *unit.case unit.rec*

⟨*ML*⟩

**lemma** *unit-all-eq1*: $(\bigwedge x::unit.\ PROP\ P\ x) \equiv PROP\ P\ ()$
⟨*proof*⟩

**lemma** *unit-all-eq2*: $(\bigwedge x::unit.\ PROP\ P) \equiv PROP\ P$
⟨*proof*⟩

This rewrite counters the effect of simproc *unit-eq* on $\lambda u::unit.\ f\ u$, replacing it by $f$ rather than by $\lambda u.\ f\ ()$.

**lemma** *unit-abs-eta-conv* [*simp*]: $(\lambda u::unit.\ f\ ()) = f$
⟨*proof*⟩

**lemma** *UNIV-unit*: $UNIV = \{()\}$
⟨*proof*⟩

**instantiation** *unit* :: *default*
**begin**

**definition** *default* = ()

**instance** ⟨*proof*⟩

**end**

**instantiation** *unit* :: {*complete-boolean-algebra*,*complete-linorder*,*wellorder*}
**begin**

**definition** *less-eq-unit* :: *unit* ⇒ *unit* ⇒ *bool*
**where** $(\text{-}::unit) \le \text{-} \longleftrightarrow True$

**lemma** *less-eq-unit* [*iff*]: $u \le v$ **for** *u v* :: *unit*
⟨*proof*⟩

**definition** *less-unit* :: *unit* ⇒ *unit* ⇒ *bool*
**where** $(\text{-}::unit) < \text{-} \longleftrightarrow False$

**lemma** *less-unit* [*iff*]: $\neg\ u < v$ **for** *u v* :: *unit*
⟨*proof*⟩

**definition** *bot-unit* :: *unit*
  **where** [*code-unfold*]: $\bot = ()$

**definition** *top-unit* :: *unit*
  **where** [*code-unfold*]: $\top = ()$

**definition** *inf-unit* :: *unit* $\Rightarrow$ *unit* $\Rightarrow$ *unit*
  **where** [*simp*]: $\text{-} \sqcap \text{-} = ()$

**definition** *sup-unit* :: *unit* $\Rightarrow$ *unit* $\Rightarrow$ *unit*
  **where** [*simp*]: $\text{-} \sqcup \text{-} = ()$

**definition** *Inf-unit* :: *unit set* $\Rightarrow$ *unit*
  **where** [*simp*]: $\bigsqcap \text{-} = ()$

**definition** *Sup-unit* :: *unit set* $\Rightarrow$ *unit*
  **where** [*simp*]: $\bigsqcup \text{-} = ()$

**definition** *uminus-unit* :: *unit* $\Rightarrow$ *unit*
  **where** [*simp*]: $- \text{-} = ()$

**declare** *less-eq-unit-def* [*abs-def*, *code-unfold*]
  *less-unit-def* [*abs-def*, *code-unfold*]
  *inf-unit-def* [*abs-def*, *code-unfold*]
  *sup-unit-def* [*abs-def*, *code-unfold*]
  *Inf-unit-def* [*abs-def*, *code-unfold*]
  *Sup-unit-def* [*abs-def*, *code-unfold*]
  *uminus-unit-def* [*abs-def*, *code-unfold*]

**instance**
  ⟨*proof*⟩

**end**

**lemma** [*code*]: *HOL.equal u v* $\longleftrightarrow$ *True* **for** *u v* :: *unit*
  ⟨*proof*⟩

**code-printing**
  **type-constructor** *unit* ⇀
    (*SML*) *unit*
    **and** (*OCaml*) *unit*
    **and** (*Haskell*) ()
    **and** (*Scala*) *Unit*
| **constant** *Unity* ⇀
    (*SML*) ()
    **and** (*OCaml*) ()
    **and** (*Haskell*) ()
    **and** (*Scala*) ()

| **class-instance** *unit* :: *equal* $\rightharpoonup$
  (*Haskell*) $-$
| **constant** *HOL.equal* :: *unit* $\Rightarrow$ *unit* $\Rightarrow$ *bool* $\rightharpoonup$
  (*Haskell*) **infix** *4* $==$

**code-reserved**
 (*SML*) *unit*
 **and** (*OCaml*) *unit*
 **and** (*Scala*) *Unit*

## 14.3 The product type

### 14.3.1 Type definition

**definition** *Pair-Rep* :: $'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
 **where** *Pair-Rep a b* $= (\lambda x\ y.\ x = a \wedge y = b)$

**definition** *prod* $= \{f.\ \exists\, a\ b.\ f = Pair\text{-}Rep\ (a{::}'a)\ (b{::}'b)\}$

**typedef** $('a,\ 'b)$ *prod* (‹(‹notation=‹infix ×››- ×/ -)› [*21*, *20*] *20*) = *prod* :: $('a \Rightarrow 'b \Rightarrow bool)$ *set*
 ⟨*proof*⟩

**type-notation** (*ASCII*)
 *prod* (**infixr** ‹∗› *20*)

**definition** *Pair* :: $'a \Rightarrow 'b \Rightarrow 'a \times 'b$
 **where** *Pair a b* = *Abs-prod* (*Pair-Rep a b*)

**lemma** *prod-cases*: $(\bigwedge a\ b.\ P\ (Pair\ a\ b)) \Longrightarrow P\ p$
 ⟨*proof*⟩

**free-constructors** *case-prod* **for** *Pair fst snd*
⟨*proof*⟩

Avoid name clashes by prefixing the output of *old-rep-datatype* with *old*.

⟨*ML*⟩

**old-rep-datatype** *Pair*
 ⟨*proof*⟩

⟨*ML*⟩

But erase the prefix for properties that are not generated by *free-constructors*.

⟨*ML*⟩

**declare** *old.prod.inject* [*iff del*]

**lemmas** *induct* = *old.prod.induct*

**lemmas** *inducts* = *old.prod.inducts*
**lemmas** *rec* = *old.prod.rec*
**lemmas** *simps* = *prod.inject prod.case prod.rec*

⟨*ML*⟩

**declare** *prod.case* [*nitpick-simp del*]
**declare** *old.prod.case-cong-weak* [*cong del*]
**declare** *prod.case-eq-if* [*mono*]
**declare** *prod.split* [*no-atp*]
**declare** *prod.split-asm* [*no-atp*]

*prod.split* could be declared as [*split*] done after the Splitter has been speeded up significantly; precompute the constants involved and don't do anything unless the current goal contains one of those constants.

### 14.3.2 Tuple syntax

Patterns – extends pre-defined type *pttrn* used in abstractions.

**nonterminal** *tuple-args* **and** *patterns*

**open-bundle** *tuple-syntax*
**begin**

**syntax**
*-tuple* :: *'a* ⇒ *tuple-args* ⇒ *'a* × *'b* (‹(‹indent=1 notation=‹mixfix tuple››'(-,/ -'))›)
*-tuple-arg* :: *'a* ⇒ *tuple-args* (‹-›)
*-tuple-args* :: *'a* ⇒ *tuple-args* ⇒ *tuple-args* (‹-,/ -›)
*-pattern* :: *pttrn* ⇒ *patterns* ⇒ *pttrn* (‹(‹open-block notation=‹pattern tuple››'(-,/ -'))›)
:: *pttrn* ⇒ *patterns* (‹-›)
*-patterns* :: *pttrn* ⇒ *patterns* ⇒ *patterns* (‹-,/ -›)
*-unit* :: *pttrn* (‹(‹open-block notation=‹pattern unit››'(')›)
**syntax-consts**
*-pattern -patterns* ⇌ *case-prod* **and**
*-unit* ⇌ *case-unit*
**translations**
$(x, y)$ ⇌ *CONST Pair x y*
*-pattern x y* ⇌ *CONST Pair x y*
*-patterns x y* ⇌ *CONST Pair x y*
*-tuple x* (*-tuple-args y z*) ⇌ *-tuple x* (*-tuple-arg* (*-tuple y z*))
λ(*x*, *y*, *zs*). *b* ⇌ *CONST case-prod* (λ*x* (*y*, *zs*). *b*)
λ(*x*, *y*). *b* ⇌ *CONST case-prod* (λ*x y*. *b*)
*-abs* (*CONST Pair x y*) *t* ⇀ λ(*x*, *y*). *t*
— This rule accommodates tuples in *case C* ... (*x*, *y*) ... ⇒ ...: The (*x*, *y*) is parsed as *Pair x y* because it is *logic*, not *pttrn*.

$\lambda()$. $b \rightleftharpoons$ *CONST case-unit b*
*-abs (CONST Unity) t* $\rightharpoonup$ $\lambda()$. *t*

**end**

print *case-prod f* as *case-prod f* and *case-prod f* as *case-prod f*

⟨*ML*⟩

Reconstruct pattern from (nested) *case-prod*s, avoiding eta-contraction of body; required for enclosing "let", if "let" does not avoid eta-contraction, which has been observed to occur.

⟨*ML*⟩

### 14.3.3 Code generator setup

**code-printing**
**type-constructor** *prod* $\rightharpoonup$
(*SML*) **infix** *2* ∗
**and** (*OCaml*) **infix** *2* ∗
**and** (*Haskell*) !((-),/ (-))
**and** (*Scala*) ((-),/ (-))
| **constant** *Pair* $\rightharpoonup$
(*SML*) !((-),/ (-))
**and** (*OCaml*) !((-),/ (-))
**and** (*Haskell*) !((-),/ (-))
**and** (*Scala*) !((-),/ (-))
| **class-instance** *prod* :: *equal* $\rightharpoonup$
(*Haskell*) −
| **constant** *HOL.equal* :: $'a \times 'b \Rightarrow 'a \times 'b \Rightarrow bool$ $\rightharpoonup$
(*Haskell*) **infix** *4* ==
| **constant** *fst* $\rightharpoonup$ (*Haskell*) *fst*
| **constant** *snd* $\rightharpoonup$ (*Haskell*) *snd*

### 14.3.4 Fundamental operations and properties

**lemma** *Pair-inject*: $(a, b) = (a', b') \Longrightarrow (a = a' \Longrightarrow b = b' \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *surj-pair* [*simp*]: $\exists\, x\ y.\ p = (x, y)$
⟨*proof*⟩

**lemma** *fst-eqD*: $fst\ (x, y) = a \Longrightarrow x = a$
⟨*proof*⟩

**lemma** *snd-eqD*: $snd\ (x, y) = a \Longrightarrow y = a$
⟨*proof*⟩

**lemma** *case-prod-unfold* [*nitpick-unfold*]: $case\text{-}prod = (\lambda c\ p.\ c\ (fst\ p)\ (snd\ p))$
⟨*proof*⟩

**lemma** *case-prod-conv* [*simp*, *code*]: $(case\ (a,\ b)\ of\ (c,\ d) \Rightarrow f\ c\ d) = f\ a\ b$
⟨*proof*⟩

**lemmas** *surjective-pairing* = *prod.collapse* [*symmetric*]

**lemma** *prod-eq-iff*: $s = t \longleftrightarrow fst\ s = fst\ t \wedge snd\ s = snd\ t$
⟨*proof*⟩

**lemma** *prod-eqI* [*intro?*]: $fst\ p = fst\ q \Longrightarrow snd\ p = snd\ q \Longrightarrow p = q$
⟨*proof*⟩

**lemma** *case-prodI*: $f\ a\ b \Longrightarrow case\ (a,\ b)\ of\ (c,\ d) \Rightarrow f\ c\ d$
⟨*proof*⟩

**lemma** *case-prodD*: $(case\ (a,\ b)\ of\ (c,\ d) \Rightarrow f\ c\ d) \Longrightarrow f\ a\ b$
⟨*proof*⟩

**lemma** *case-prod-Pair* [*simp*]: $case\text{-}prod\ Pair = id$
⟨*proof*⟩

**lemma** *case-prod-eta*: $(\lambda(x,\ y).\ f\ (x,\ y)) = f$
— Subsumes the old *split-Pair* when $f$ is the identity function.
⟨*proof*⟩

**lemma** *case-prod-comp*: $(case\ x\ of\ (a,\ b) \Rightarrow (f \circ g)\ a\ b) = f\ (g\ (fst\ x))\ (snd\ x)$
⟨*proof*⟩

**lemma** *The-case-prod*: $The\ (case\text{-}prod\ P) = (THE\ xy.\ P\ (fst\ xy)\ (snd\ xy))$
⟨*proof*⟩

**lemma** *cond-case-prod-eta*: $(\bigwedge x\ y.\ f\ x\ y = g\ (x,\ y)) \Longrightarrow (\lambda(x,\ y).\ f\ x\ y) = g$
⟨*proof*⟩

**lemma** *split-paired-all* [*no-atp*]: $(\bigwedge x.\ PROP\ P\ x) \equiv (\bigwedge a\ b.\ PROP\ P\ (a,\ b))$
⟨*proof*⟩

The rule *split-paired-all* does not work with the Simplifier because it also affects premises in congrence rules, where this can lead to premises of the form $\bigwedge a\ b.\ \ldots = ?P(a,\ b)$ which cannot be solved by reflexivity.

**lemmas** *split-tupled-all* = *split-paired-all unit-all-eq2*

⟨*ML*⟩

**lemma** *split-paired-All* [*simp*, *no-atp*]: $(\forall x.\ P\ x) \longleftrightarrow (\forall a\ b.\ P\ (a,\ b))$
— [*iff*] is not a good idea because it makes *blast* loop
⟨*proof*⟩

**lemma** *split-paired-Ex* [*simp*, *no-atp*]: $(\exists\, x.\ P\ x) \longleftrightarrow (\exists\, a\ b.\ P\ (a,\ b))$
⟨*proof*⟩

**lemma** *split-paired-The* [*no-atp*]: $(THE\ x.\ P\ x) = (THE\ (a,\ b).\ P\ (a,\ b))$
— Can't be added to simpset: loops!
⟨*proof*⟩

Simplification procedure for *cond-case-prod-eta*. Using *case-prod-eta* as a rewrite rule is not general enough, and using *cond-case-prod-eta* directly would render some existing proofs very inefficient; similarly for *prod.case-eq-if*.

⟨*ML*⟩

**lemma** *case-prod-beta′*: $(\lambda(x,y).\ f\ x\ y) = (\lambda x.\ f\ (fst\ x)\ (snd\ x))$
⟨*proof*⟩

*case-prod* used as a logical connective or set former.

These rules are for use with *blast*; could instead call *simp* using *prod.split* as rewrite.

**lemma** *case-prodI2*:
$\bigwedge p.\ (\bigwedge a\ b.\ p = (a,\ b) \Longrightarrow c\ a\ b) \Longrightarrow case\ p\ of\ (a,\ b) \Rightarrow c\ a\ b$
⟨*proof*⟩

**lemma** *case-prodI2′*:
$\bigwedge p.\ (\bigwedge a\ b.\ (a,\ b) = p \Longrightarrow c\ a\ b\ x) \Longrightarrow (case\ p\ of\ (a,\ b) \Rightarrow c\ a\ b)\ x$
⟨*proof*⟩

**lemma** *case-prodE* [*elim*!]:
$(case\ p\ of\ (a,\ b) \Rightarrow c\ a\ b) \Longrightarrow (\bigwedge x\ y.\ p = (x,\ y) \Longrightarrow c\ x\ y \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *case-prodE′* [*elim*!]:
$(case\ p\ of\ (a,\ b) \Rightarrow c\ a\ b)\ z \Longrightarrow (\bigwedge x\ y.\ p = (x,\ y) \Longrightarrow c\ x\ y\ z \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *case-prodE2*:
**assumes** *q*: $Q\ (case\ z\ of\ (a,\ b) \Rightarrow P\ a\ b)$
**and** *r*: $\bigwedge x\ y.\ z = (x,\ y) \Longrightarrow Q\ (P\ x\ y) \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

**lemma** *case-prodD′*: $(case\ (a,\ b)\ of\ (c,\ d) \Rightarrow R\ c\ d)\ c \Longrightarrow R\ a\ b\ c$
⟨*proof*⟩

**lemma** *mem-case-prodI*: $z \in c\ a\ b \Longrightarrow z \in (case\ (a,\ b)\ of\ (d,\ e) \Rightarrow c\ d\ e)$
⟨*proof*⟩

**lemma** *mem-case-prodI2* [*intro*!]:
$\bigwedge p.\ (\bigwedge a\ b.\ p = (a,\ b) \Longrightarrow z \in c\ a\ b) \Longrightarrow z \in (case\ p\ of\ (a,\ b) \Rightarrow c\ a\ b)$
⟨*proof*⟩

**declare** *mem-case-prodI* [*intro*!] — postponed to maintain traditional declaration order!
**declare** *case-prodI2'* [*intro*!] — postponed to maintain traditional declaration order!
**declare** *case-prodI2* [*intro*!] — postponed to maintain traditional declaration order!
**declare** *case-prodI* [*intro*!] — postponed to maintain traditional declaration order!

**lemma** *mem-case-prodE* [*elim*!]:
**assumes** $z \in case\text{-}prod\ c\ p$
**obtains** $x\ y$ **where** $p = (x,\ y)$ **and** $z \in c\ x\ y$
⟨*proof*⟩

⟨*ML*⟩

**lemma** *split-eta-SetCompr* [*simp*, *no-atp*]: $(\lambda u.\ \exists x\ y.\ u = (x,\ y) \wedge P\ (x,\ y)) = P$
⟨*proof*⟩

**lemma** *split-eta-SetCompr2* [*simp*, *no-atp*]: $(\lambda u.\ \exists x\ y.\ u = (x,\ y) \wedge P\ x\ y) = case\text{-}prod\ P$
⟨*proof*⟩

**lemma** *split-part* [*simp*]: $(\lambda(a,b).\ P \wedge Q\ a\ b) = (\lambda ab.\ P \wedge case\text{-}prod\ Q\ ab)$
— Allows simplifications of nested splits in case of independent predicates.
⟨*proof*⟩

**lemma** *split-comp-eq*:
**fixes** $f :: 'a \Rightarrow 'b \Rightarrow 'c$
**and** $g :: 'd \Rightarrow 'a$
**shows** $(\lambda u.\ f\ (g\ (fst\ u))\ (snd\ u)) = case\text{-}prod\ (\lambda x.\ f\ (g\ x))$
⟨*proof*⟩

**lemma** *pair-imageI* [*intro*]: $(a,\ b) \in A \Longrightarrow f\ a\ b \in (\lambda(a,\ b).\ f\ a\ b)$ ‘ $A$
⟨*proof*⟩

**lemma** *Collect-const-case-prod*[*simp*]: $\{(a,b).\ P\} = (if\ P\ then\ UNIV\ else\ \{\})$
⟨*proof*⟩

**lemma** *The-split-eq* [*simp*]: $(THE\ (x',y').\ x = x' \wedge y = y') = (x,\ y)$
⟨*proof*⟩

**lemma** *case-prod-beta*: $case\text{-}prod\ f\ p = f\ (fst\ p)\ (snd\ p)$
⟨*proof*⟩

**lemma** *prod-cases3* [*cases type*]:
**obtains** (*fields*) $a\ b\ c$ **where** $y = (a,\ b,\ c)$
⟨*proof*⟩

**lemma** *prod-induct3* [*case-names fields*, *induct type*]:
$(\bigwedge a\ b\ c.\ P\ (a,\ b,\ c)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *prod-cases4* [*cases type*]:
**obtains** (*fields*) $a\ b\ c\ d$ **where** $y = (a,\ b,\ c,\ d)$
⟨*proof*⟩

**lemma** *prod-induct4* [*case-names fields*, *induct type*]:
$(\bigwedge a\ b\ c\ d.\ P\ (a,\ b,\ c,\ d)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *prod-cases5* [*cases type*]:
**obtains** (*fields*) $a\ b\ c\ d\ e$ **where** $y = (a,\ b,\ c,\ d,\ e)$
⟨*proof*⟩

**lemma** *prod-induct5* [*case-names fields*, *induct type*]:
$(\bigwedge a\ b\ c\ d\ e.\ P\ (a,\ b,\ c,\ d,\ e)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *prod-cases6* [*cases type*]:
**obtains** (*fields*) $a\ b\ c\ d\ e\ f$ **where** $y = (a,\ b,\ c,\ d,\ e,\ f)$
⟨*proof*⟩

**lemma** *prod-induct6* [*case-names fields*, *induct type*]:
$(\bigwedge a\ b\ c\ d\ e\ f.\ P\ (a,\ b,\ c,\ d,\ e,\ f)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *prod-cases7* [*cases type*]:
**obtains** (*fields*) $a\ b\ c\ d\ e\ f\ g$ **where** $y = (a,\ b,\ c,\ d,\ e,\ f,\ g)$
⟨*proof*⟩

**lemma** *prod-induct7* [*case-names fields*, *induct type*]:
$(\bigwedge a\ b\ c\ d\ e\ f\ g.\ P\ (a,\ b,\ c,\ d,\ e,\ f,\ g)) \Longrightarrow P\ x$
⟨*proof*⟩

**definition** *internal-case-prod* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c$
**where** *internal-case-prod* $\equiv$ *case-prod*

**lemma** *internal-case-prod-conv*: *internal-case-prod* $c\ (a,\ b) = c\ a\ b$
⟨*proof*⟩

⟨*ML*⟩

**hide-const** *internal-case-prod*

### 14.3.5 Derived operations

**definition** *curry* :: $('a \times 'b \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c$
  **where** $curry = (\lambda c\ x\ y.\ c\ (x,\ y))$

**lemma** *curry-conv* [*simp*, *code*]: $curry\ f\ a\ b = f\ (a,\ b)$
  ⟨*proof*⟩

**lemma** *curryI* [*intro*!]: $f\ (a,\ b) \Longrightarrow curry\ f\ a\ b$
  ⟨*proof*⟩

**lemma** *curryD* [*dest*!]: $curry\ f\ a\ b \Longrightarrow f\ (a,\ b)$
  ⟨*proof*⟩

**lemma** *curryE*: $curry\ f\ a\ b \Longrightarrow (f\ (a,\ b) \Longrightarrow Q) \Longrightarrow Q$
  ⟨*proof*⟩

**lemma** *curry-case-prod* [*simp*]: $curry\ (case\text{-}prod\ f) = f$
  ⟨*proof*⟩

**lemma** *case-prod-curry* [*simp*]: $case\text{-}prod\ (curry\ f) = f$
  ⟨*proof*⟩

**lemma** *curry-K*: $curry\ (\lambda x.\ c) = (\lambda x\ y.\ c)$
  ⟨*proof*⟩

The composition-uncurry combinator.

**definition** *scomp* :: $('a \Rightarrow 'b \times 'c) \Rightarrow ('b \Rightarrow 'c \Rightarrow 'd) \Rightarrow 'a \Rightarrow 'd$ (**infixl** ‹∘→› *60*)
  **where** $f \circ\!\!\rightarrow g = (\lambda x.\ case\text{-}prod\ g\ (f\ x))$

**no-notation** *scomp* (**infixl** ‹∘→› *60*)

**bundle** *state-combinator-syntax*
**begin**
**notation** *fcomp* (**infixl** ‹∘>› *60*)
**notation** *scomp* (**infixl** ‹∘→› *60*)
**end**

**context**
  **includes** *state-combinator-syntax*
**begin**

**lemma** *scomp-unfold*: $(\circ\!\!\rightarrow) = (\lambda f\ g\ x.\ g\ (fst\ (f\ x))\ (snd\ (f\ x)))$
  ⟨*proof*⟩

**lemma** *scomp-apply* [*simp*]: $(f \circ\!\!\rightarrow g)\ x = \mathit{case\text{-}prod}\ g\ (f\ x)$
⟨*proof*⟩

**lemma** *Pair-scomp*: $\mathit{Pair}\ x \circ\!\!\rightarrow f = f\ x$
⟨*proof*⟩

**lemma** *scomp-Pair*: $x \circ\!\!\rightarrow \mathit{Pair} = x$
⟨*proof*⟩

**lemma** *scomp-scomp*: $(f \circ\!\!\rightarrow g) \circ\!\!\rightarrow h = f \circ\!\!\rightarrow (\lambda x.\ g\ x \circ\!\!\rightarrow h)$
⟨*proof*⟩

**lemma** *scomp-fcomp*: $(f \circ\!\!\rightarrow g) \circ\!> h = f \circ\!\!\rightarrow (\lambda x.\ g\ x \circ\!> h)$
⟨*proof*⟩

**lemma** *fcomp-scomp*: $(f \circ\!> g) \circ\!\!\rightarrow h = f \circ\!> (g \circ\!\!\rightarrow h)$
⟨*proof*⟩

**end**

**code-printing**
**constant** *scomp* ⇀ (*Eval*) **infixl** *3* #−>

*map-prod* — action of the product functor upon functions.

**definition** *map-prod* :: $('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow 'a \times 'b \Rightarrow 'c \times 'd$
**where** $\mathit{map\text{-}prod}\ f\ g = (\lambda(x,\ y).\ (f\ x,\ g\ y))$

**lemma** *map-prod-simp* [*simp*, *code*]: $\mathit{map\text{-}prod}\ f\ g\ (a,\ b) = (f\ a,\ g\ b)$
⟨*proof*⟩

**functor** *map-prod*: *map-prod*
⟨*proof*⟩

**lemma** *fst-map-prod* [*simp*]: $\mathit{fst}\ (\mathit{map\text{-}prod}\ f\ g\ x) = f\ (\mathit{fst}\ x)$
⟨*proof*⟩

**lemma** *snd-map-prod* [*simp*]: $\mathit{snd}\ (\mathit{map\text{-}prod}\ f\ g\ x) = g\ (\mathit{snd}\ x)$
⟨*proof*⟩

**lemma** *fst-comp-map-prod* [*simp*]: $\mathit{fst} \circ \mathit{map\text{-}prod}\ f\ g = f \circ \mathit{fst}$
⟨*proof*⟩

**lemma** *snd-comp-map-prod* [*simp*]: $\mathit{snd} \circ \mathit{map\text{-}prod}\ f\ g = g \circ \mathit{snd}$
⟨*proof*⟩

**lemma** *map-prod-compose*: $\mathit{map\text{-}prod}\ (f1 \circ f2)\ (g1 \circ g2) = (\mathit{map\text{-}prod}\ f1\ g1 \circ \mathit{map\text{-}prod}\ f2\ g2)$
⟨*proof*⟩

**lemma** *map-prod-ident* [*simp*]: *map-prod* $(\lambda x.\ x)$ $(\lambda y.\ y) = (\lambda z.\ z)$
⟨*proof*⟩

**lemma** *map-prod-imageI* [*intro*]: $(a,\ b) \in R \Longrightarrow (f\ a,\ g\ b) \in$ *map-prod* $f\ g$ ' $R$
⟨*proof*⟩

**lemma** *prod-fun-imageE* [*elim*!]:
**assumes** *major*: $c \in$ *map-prod* $f\ g$ ' $R$
**and** *cases*: $\bigwedge x\ y.\ c = (f\ x,\ g\ y) \Longrightarrow (x,\ y) \in R \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**definition** *apfst* :: $('a \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c \times 'b$
**where** *apfst* $f$ = *map-prod* $f$ *id*

**definition** *apsnd* :: $('b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'a \times 'c$
**where** *apsnd* $f$ = *map-prod* *id* $f$

**lemma** *apfst-conv* [*simp*, *code*]: *apfst* $f$ $(x,\ y) = (f\ x,\ y)$
⟨*proof*⟩

**lemma** *apsnd-conv* [*simp*, *code*]: *apsnd* $f$ $(x,\ y) = (x,\ f\ y)$
⟨*proof*⟩

**lemma** *fst-apfst* [*simp*]: *fst* (*apfst* $f\ x$) = $f$ (*fst* $x$)
⟨*proof*⟩

**lemma** *fst-comp-apfst* [*simp*]: *fst* $\circ$ *apfst* $f$ = $f \circ$ *fst*
⟨*proof*⟩

**lemma** *fst-apsnd* [*simp*]: *fst* (*apsnd* $f\ x$) = *fst* $x$
⟨*proof*⟩

**lemma** *fst-comp-apsnd* [*simp*]: *fst* $\circ$ *apsnd* $f$ = *fst*
⟨*proof*⟩

**lemma** *snd-apfst* [*simp*]: *snd* (*apfst* $f\ x$) = *snd* $x$
⟨*proof*⟩

**lemma** *snd-comp-apfst* [*simp*]: *snd* $\circ$ *apfst* $f$ = *snd*
⟨*proof*⟩

**lemma** *snd-apsnd* [*simp*]: *snd* (*apsnd* $f\ x$) = $f$ (*snd* $x$)
⟨*proof*⟩

**lemma** *snd-comp-apsnd* [*simp*]: *snd* $\circ$ *apsnd* $f$ = $f \circ$ *snd*
⟨*proof*⟩

**lemma** *apfst-compose*: $apfst\ f\ (apfst\ g\ x) = apfst\ (f \circ g)\ x$
⟨*proof*⟩

**lemma** *apsnd-compose*: $apsnd\ f\ (apsnd\ g\ x) = apsnd\ (f \circ g)\ x$
⟨*proof*⟩

**lemma** *apfst-apsnd* [*simp*]: $apfst\ f\ (apsnd\ g\ x) = (f\ (fst\ x),\ g\ (snd\ x))$
⟨*proof*⟩

**lemma** *apsnd-apfst* [*simp*]: $apsnd\ f\ (apfst\ g\ x) = (g\ (fst\ x),\ f\ (snd\ x))$
⟨*proof*⟩

**lemma** *apfst-id* [*simp*]: $apfst\ id = id$
⟨*proof*⟩

**lemma** *apsnd-id* [*simp*]: $apsnd\ id = id$
⟨*proof*⟩

**lemma** *apfst-eq-conv* [*simp*]: $apfst\ f\ x = apfst\ g\ x \longleftrightarrow f\ (fst\ x) = g\ (fst\ x)$
⟨*proof*⟩

**lemma** *apsnd-eq-conv* [*simp*]: $apsnd\ f\ x = apsnd\ g\ x \longleftrightarrow f\ (snd\ x) = g\ (snd\ x)$
⟨*proof*⟩

**lemma** *apsnd-apfst-commute*: $apsnd\ f\ (apfst\ g\ p) = apfst\ g\ (apsnd\ f\ p)$
⟨*proof*⟩

**context**
**begin**

⟨*ML*⟩

**definition** *swap* :: $'a \times 'b \Rightarrow 'b \times 'a$
**where** $swap\ p = (snd\ p,\ fst\ p)$

**end**

**lemma** *swap-simp* [*simp*]: $prod.swap\ (x,\ y) = (y,\ x)$
⟨*proof*⟩

**lemma** *swap-swap* [*simp*]: $prod.swap\ (prod.swap\ p) = p$
⟨*proof*⟩

**lemma** *swap-comp-swap* [*simp*]: $prod.swap \circ prod.swap = id$
⟨*proof*⟩

**lemma** *pair-in-swap-image* [*simp*]: $(y,\ x) \in prod.swap\ `\ A \longleftrightarrow (x,\ y) \in A$
⟨*proof*⟩

**lemma** *inj-swap* [*simp*]: *inj-on prod.swap A*
⟨*proof*⟩

**lemma** *swap-inj-on*: *inj-on* ($\lambda(i, j)$. $(j, i)$) *A*
⟨*proof*⟩

**lemma** *surj-swap* [*simp*]: *surj prod.swap*
⟨*proof*⟩

**lemma** *bij-swap* [*simp*]: *bij prod.swap*
⟨*proof*⟩

**lemma** *case-swap* [*simp*]: (*case prod.swap p of* $(y, x) \Rightarrow f\ x\ y$) = (*case p of* $(x, y) \Rightarrow f\ x\ y$)
⟨*proof*⟩

**lemma** *fst-swap* [*simp*]: *fst* (*prod.swap x*) = *snd x*
⟨*proof*⟩

**lemma** *snd-swap* [*simp*]: *snd* (*prod.swap x*) = *fst x*
⟨*proof*⟩

**lemma** *split-pairs*: $(A,B) = X \longleftrightarrow fst\ X = A \wedge snd\ X = B$
**and** *split-pairs2*: $X = (A,B) \longleftrightarrow fst\ X = A \wedge snd\ X = B$
⟨*proof*⟩

Disjoint union of a family of sets – Sigma.

**definition** *Sigma* :: $'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set$
**where** *Sigma A B* $\equiv \bigcup x{\in}A.\ \bigcup y{\in}B\ x.\ \{Pair\ x\ y\}$

**context**
**begin**
**qualified abbreviation** *Times* :: $'a\ set \Rightarrow 'b\ set \Rightarrow ('a \times 'b)\ set$ (**infixr** ‹×› *80*)
**where** $A \times B \equiv Sigma\ A\ (\lambda\text{-}.\ B)$
**end**

**bundle** *set-product-syntax*
**begin**
**notation** *Product-Type.Times* (**infixr** ‹×› *80*)
**end**

**syntax**
*-Sigma* :: $pttrn \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow ('a \times 'b)\ set$
(‹(‹*indent=3 notation=*‹*binder SIGMA*››*SIGMA* -:-./ -)› [*0*, *0*, *10*] *10*)
**syntax-consts**
*-Sigma* ⇌ *Sigma*
**translations**
*SIGMA x*:*A*. *B* ⇌ *CONST Sigma A* (λ*x*. *B*)

**lemma** *SigmaI* [*intro*!]: $a \in A \Longrightarrow b \in B\ a \Longrightarrow (a,\ b) \in Sigma\ A\ B$
⟨*proof*⟩

**lemma** *SigmaE* [*elim*!]: $c \in Sigma\ A\ B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in B\ x \Longrightarrow c = (x,\ y) \Longrightarrow P) \Longrightarrow P$
— The general elimination rule.
⟨*proof*⟩

Elimination of $(a,\ b) \in A \times B$ – introduces no eigenvariables.

**lemma** *SigmaD1*: $(a,\ b) \in Sigma\ A\ B \Longrightarrow a \in A$
⟨*proof*⟩

**lemma** *SigmaD2*: $(a,\ b) \in Sigma\ A\ B \Longrightarrow b \in B\ a$
⟨*proof*⟩

**lemma** *SigmaE2*: $(a,\ b) \in Sigma\ A\ B \Longrightarrow (a \in A \Longrightarrow b \in B\ a \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *Sigma-cong*: $A = B \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow C\ x = D\ x) \Longrightarrow (SIGMA\ x{:}A.\ C\ x) = (SIGMA\ x{:}B.\ D\ x)$
⟨*proof*⟩

**lemma** *Sigma-mono*: $A \subseteq C \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq D\ x) \Longrightarrow Sigma\ A\ B \subseteq Sigma\ C\ D$
⟨*proof*⟩

**lemma** *Sigma-empty1* [*simp*]: $Sigma\ \{\}\ B = \{\}$
⟨*proof*⟩

**lemma** *Sigma-empty2* [*simp*]: $A \times \{\} = \{\}$
⟨*proof*⟩

**lemma** *UNIV-Times-UNIV* [*simp*]: $UNIV \times UNIV = UNIV$
⟨*proof*⟩

**lemma** *Compl-Times-UNIV1* [*simp*]: $-\ (UNIV \times A) = UNIV \times (-A)$
⟨*proof*⟩

**lemma** *Compl-Times-UNIV2* [*simp*]: $-\ (A \times UNIV) = (-A) \times UNIV$
⟨*proof*⟩

**lemma** *mem-Sigma-iff* [*iff*]: $(a,\ b) \in Sigma\ A\ B \longleftrightarrow a \in A \wedge b \in B\ a$
⟨*proof*⟩

**lemma** *mem-Times-iff*: $x \in A \times B \longleftrightarrow fst\ x \in A \wedge snd\ x \in B$
⟨*proof*⟩

**lemma** *Sigma-empty-iff*: $(SIGMA\ i{:}I.\ X\ i) = \{\} \longleftrightarrow (\forall\, i{\in}I.\ X\ i = \{\})$
⟨*proof*⟩

**lemma** *Times-subset-cancel2*: $x \in C \Longrightarrow A \times C \subseteq B \times C \longleftrightarrow A \subseteq B$
⟨*proof*⟩

**lemma** *Times-eq-cancel2*: $x \in C \Longrightarrow A \times C = B \times C \longleftrightarrow A = B$
⟨*proof*⟩

**lemma** *Collect-case-prod-Sigma*: $\{(x, y).\ P\ x \wedge Q\ x\ y\} = (SIGMA\ x{:}Collect\ P.\ Collect\ (Q\ x))$
⟨*proof*⟩

**lemma** *Collect-case-prod* [*simp*]: $\{(a, b).\ P\ a \wedge Q\ b\} = Collect\ P \times Collect\ Q$
⟨*proof*⟩

**lemma** *Collect-case-prodD*: $x \in Collect\ (case\text{-}prod\ A) \Longrightarrow A\ (fst\ x)\ (snd\ x)$
⟨*proof*⟩

**lemma** *Collect-case-prod-mono*: $A \leq B \Longrightarrow Collect\ (case\text{-}prod\ A) \subseteq Collect\ (case\text{-}prod\ B)$
⟨*proof*⟩

**lemma** *Collect-split-mono-strong*:
$X = fst\ `\ A \Longrightarrow Y = snd\ `\ A \Longrightarrow \forall a{\in}X.\ \forall b \in Y.\ P\ a\ b \longrightarrow Q\ a\ b$
$\Longrightarrow A \subseteq Collect\ (case\text{-}prod\ P) \Longrightarrow A \subseteq Collect\ (case\text{-}prod\ Q)$
⟨*proof*⟩

**lemma** *UN-Times-distrib*: $(\bigcup(a, b){\in}A \times B.\ E\ a \times F\ b) = \bigcup(E\ `\ A) \times \bigcup(F\ `\ B)$
— Suggested by Pierre Chartier
⟨*proof*⟩

**lemma** *split-paired-Ball-Sigma* [*simp*, *no-atp*]: $(\forall z{\in}Sigma\ A\ B.\ P\ z) \longleftrightarrow (\forall x{\in}A.\ \forall y{\in}B\ x.\ P\ (x, y))$
⟨*proof*⟩

**lemma** *split-paired-Bex-Sigma* [*simp*, *no-atp*]: $(\exists z{\in}Sigma\ A\ B.\ P\ z) \longleftrightarrow (\exists x{\in}A.\ \exists y{\in}B\ x.\ P\ (x, y))$
⟨*proof*⟩

**lemma** *Sigma-Un-distrib1*: $Sigma\ (I \cup J)\ C = Sigma\ I\ C \cup Sigma\ J\ C$
⟨*proof*⟩

**lemma** *Sigma-Un-distrib2*: $(SIGMA\ i{:}I.\ A\ i \cup B\ i) = Sigma\ I\ A \cup Sigma\ I\ B$
⟨*proof*⟩

**lemma** *Sigma-Int-distrib1*: $Sigma\ (I \cap J)\ C = Sigma\ I\ C \cap Sigma\ J\ C$
⟨*proof*⟩

**lemma** *Sigma-Int-distrib2*: $(SIGMA\ i{:}I.\ A\ i \cap B\ i) = Sigma\ I\ A \cap Sigma\ I\ B$
⟨*proof*⟩

**lemma** *Sigma-Diff-distrib1*: $Sigma\ (I - J)\ C = Sigma\ I\ C - Sigma\ J\ C$
⟨*proof*⟩

**lemma** *Sigma-Diff-distrib2*: $(SIGMA\ i{:}I.\ A\ i - B\ i) = Sigma\ I\ A - Sigma\ I\ B$
⟨*proof*⟩

**lemma** *Sigma-Union*: $Sigma\ (\bigcup X)\ B = (\bigcup A{\in}X.\ Sigma\ A\ B)$
⟨*proof*⟩

**lemma** *Pair-vimage-Sigma*: $Pair\ x\ -`\ Sigma\ A\ f = (if\ x \in A\ then\ f\ x\ else\ \{\})$
⟨*proof*⟩

Non-dependent versions are needed to avoid the need for higher-order matching, especially when the rules are re-oriented.

**lemma** *Times-Un-distrib1*: $(A \cup B) \times C = A \times C \cup B \times C$
⟨*proof*⟩

**lemma** *Times-Int-distrib1*: $(A \cap B) \times C = A \times C \cap B \times C$
⟨*proof*⟩

**lemma** *Times-Diff-distrib1*: $(A - B) \times C = A \times C - B \times C$
⟨*proof*⟩

**lemma** *Times-empty* [*simp*]: $A \times B = \{\} \longleftrightarrow A = \{\} \vee B = \{\}$
⟨*proof*⟩

**lemma** *times-subset-iff*: $A \times C \subseteq B \times D \longleftrightarrow A{=}\{\} \vee C{=}\{\} \vee A \subseteq B \wedge C \subseteq D$
⟨*proof*⟩

**lemma** *times-eq-iff*: $A \times B = C \times D \longleftrightarrow A = C \wedge B = D \vee (A = \{\} \vee B = \{\}) \wedge (C = \{\} \vee D = \{\})$
⟨*proof*⟩

**lemma** *fst-image-times* [*simp*]: $fst\ `\ (A \times B) = (if\ B = \{\}\ then\ \{\}\ else\ A)$
⟨*proof*⟩

**lemma** *snd-image-times* [*simp*]: $snd\ `\ (A \times B) = (if\ A = \{\}\ then\ \{\}\ else\ B)$
⟨*proof*⟩

**lemma** *fst-image-Sigma*: $fst\ `\ (Sigma\ A\ B) = \{x \in A.\ B(x) \neq \{\}\}$
⟨*proof*⟩

**lemma** *snd-image-Sigma*: $snd\ `\ (Sigma\ A\ B) = (\bigcup\ x \in A.\ B\ x)$
⟨*proof*⟩

**lemma** *vimage-fst*: $fst\ -`\ A = A \times UNIV$
⟨*proof*⟩

**lemma** *vimage-snd*: $snd -` A = UNIV \times A$
⟨*proof*⟩

**lemma** *insert-Times-insert* [*simp*]:
$insert\ a\ A \times insert\ b\ B = insert\ (a,b)\ (A \times insert\ b\ B \cup \{a\} \times B)$
⟨*proof*⟩

**lemma** *sing-Times-sing*: $\{x\}\times\{y\} = \{(x,y)\}$
⟨*proof*⟩

**lemma** *vimage-Times*: $f -` (A \times B) = (fst \circ f) -` A \cap (snd \circ f) -` B$
⟨*proof*⟩

**lemma** *Times-Int-Times*: $A \times B \cap C \times D = (A \cap C) \times (B \cap D)$
⟨*proof*⟩

**lemma** *image-paired-Times*:
$(\lambda(x,y).\ (f\ x,\ g\ y))\ `\ (A \times B) = (f\ `\ A) \times (g\ `\ B)$
⟨*proof*⟩

**lemma** *Times-insert-right*: $A \times insert\ y\ B = (\lambda x.\ (x,\ y))\ `\ A \cup A \times B$
⟨*proof*⟩

**lemma** *Times-insert-left*: $insert\ x\ A \times B = (\lambda y.\ (x,\ y))\ `\ B \cup A \times B$
⟨*proof*⟩

**lemma** *product-swap*: $prod.swap\ `\ (A \times B) = B \times A$
⟨*proof*⟩

**lemma** *swap-product*: $(\lambda(i,\ j).\ (j,\ i))\ `\ (A \times B) = B \times A$
⟨*proof*⟩

**lemma** *image-split-eq-Sigma*: $(\lambda x.\ (f\ x,\ g\ x))\ `\ A = Sigma\ (f\ `\ A)\ (\lambda x.\ g\ `\ (f -` \{x\} \cap A))$
⟨*proof*⟩

**lemma** *subset-fst-snd*: $A \subseteq (fst\ `\ A \times snd\ `\ A)$
⟨*proof*⟩

**lemma** *inj-on-apfst* [*simp*]: $inj\text{-}on\ (apfst\ f)\ (A \times UNIV) \longleftrightarrow inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *inj-apfst* [*simp*]: $inj\ (apfst\ f) \longleftrightarrow inj\ f$
⟨*proof*⟩

**lemma** *inj-on-apsnd* [*simp*]: $inj\text{-}on\ (apsnd\ f)\ (UNIV \times A) \longleftrightarrow inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *inj-apsnd* [*simp*]: $inj\ (apsnd\ f) \longleftrightarrow inj\ f$

⟨*proof*⟩

**context**
**begin**

**qualified definition** *product* :: $'a$ *set* ⇒ $'b$ *set* ⇒ $('a \times 'b)$ *set*
**where** [*code-abbrev*]: *product* $A$ $B = A \times B$

**lemma** *member-product*: $x \in$ *Product-Type.product* $A$ $B \longleftrightarrow x \in A \times B$
⟨*proof*⟩

**end**

The following *map-prod* lemmas are due to Joachim Breitner:

**lemma** *map-prod-inj-on*:
**assumes** *inj-on* $f$ $A$
**and** *inj-on* $g$ $B$
**shows** *inj-on* (*map-prod* $f$ $g$) $(A \times B)$
⟨*proof*⟩

**lemma** *map-prod-surj*:
**fixes** $f$ :: $'a \Rightarrow 'b$
**and** $g$ :: $'c \Rightarrow 'd$
**assumes** *surj* $f$ **and** *surj* $g$
**shows** *surj* (*map-prod* $f$ $g$)
⟨*proof*⟩

**lemma** *map-prod-surj-on*:
**assumes** $f$ ‘ $A = A'$ **and** $g$ ‘ $B = B'$
**shows** *map-prod* $f$ $g$ ‘ $(A \times B) = A' \times B'$
⟨*proof*⟩

**lemma** *bij-betw-map-prod*:
**assumes** *bij-betw* $f$ $A$ $C$ *bij-betw* $g$ $B$ $D$
**shows** *bij-betw* (*map-prod* $f$ $g$) $(A \times B)$ $(C \times D)$
⟨*proof*⟩

## 14.4 Code generator setup for paired and tripled bounded set comprehension

**context**
**begin**

**qualified lemma** *paired-bounded-Collect-eq-filter* [*code-unfold*, *no-atp*]:
‹$\{(x, y).\ (x, y) \in A \wedge P\ x\ y\} =$ *Set.filter* $(\lambda(x, y).\ P\ x\ y)$ $A$›
⟨*proof*⟩ **lemma** *tripled-bounded-Collect-eq-filter* [*code-unfold*, *no-atp*]:
‹$\{(x, y, z).\ (x, y, z) \in A \wedge P\ x\ y\ z\} =$ *Set.filter* $(\lambda(x, y, z).\ P\ x\ y\ z)$ $A$›
⟨*proof*⟩

**end**

### 14.5 Simproc for rewriting a set comprehension into a point-free expression

⟨*ML*⟩

### 14.6 Lemmas about disjointness

**lemma** *disjnt-Times1-iff* [*simp*]: $disjnt\ (C \times A)\ (C \times B) \longleftrightarrow C = \{\} \vee disjnt\ A\ B$
⟨*proof*⟩

**lemma** *disjnt-Times2-iff* [*simp*]: $disjnt\ (A \times C)\ (B \times C) \longleftrightarrow C = \{\} \vee disjnt\ A\ B$
⟨*proof*⟩

**lemma** *disjnt-Sigma-iff*: $disjnt\ (Sigma\ A\ C)\ (Sigma\ B\ C) \longleftrightarrow (\forall i \in A \cap B.\ C\ i = \{\}) \vee disjnt\ A\ B$
⟨*proof*⟩

### 14.7 Inductively defined sets

⟨*ML*⟩

### 14.8 Legacy theorem bindings and duplicates

**lemmas** *fst-conv* = *prod.sel*(*1*)
**lemmas** *snd-conv* = *prod.sel*(*2*)
**lemmas** *split-def* = *case-prod-unfold*
**lemmas** *split-beta′* = *case-prod-beta′*
**lemmas** *split-beta* = *prod.case-eq-if*
**lemmas** *split-conv* = *case-prod-conv*
**lemmas** *split* = *case-prod-conv*

**hide-const** (**open**) *prod*

**end**

## 15 The Disjoint Sum of Two Types

**theory** *Sum-Type*
**imports** *Typedef Inductive Fun*
**begin**

### 15.1 Construction of the sum type and its basic abstract operations

**definition** *Inl-Rep* :: $'a \Rightarrow 'a \Rightarrow 'b \Rightarrow bool \Rightarrow bool$

**where** $Inl\text{-}Rep\ a\ x\ y\ p \longleftrightarrow x = a \wedge p$

**definition** $Inr\text{-}Rep :: 'b \Rightarrow 'a \Rightarrow 'b \Rightarrow bool \Rightarrow bool$
**where** $Inr\text{-}Rep\ b\ x\ y\ p \longleftrightarrow y = b \wedge \neg\ p$

**definition** $sum = \{f.\ (\exists\, a.\ f = Inl\text{-}Rep\ (a::'a)) \vee (\exists\, b.\ f = Inr\text{-}Rep\ (b::'b))\}$

**typedef** $('a,\ 'b)\ sum$ (**infixr** ‹+› *10*) $= sum :: ('a \Rightarrow 'b \Rightarrow bool \Rightarrow bool)\ set$
⟨*proof*⟩

**lemma** *Inl-RepI*: $Inl\text{-}Rep\ a \in sum$
⟨*proof*⟩

**lemma** *Inr-RepI*: $Inr\text{-}Rep\ b \in sum$
⟨*proof*⟩

**lemma** *inj-on-Abs-sum*: $A \subseteq sum \Longrightarrow inj\text{-}on\ Abs\text{-}sum\ A$
⟨*proof*⟩

**lemma** *Inl-Rep-inject*: $inj\text{-}on\ Inl\text{-}Rep\ A$
⟨*proof*⟩

**lemma** *Inr-Rep-inject*: $inj\text{-}on\ Inr\text{-}Rep\ A$
⟨*proof*⟩

**lemma** *Inl-Rep-not-Inr-Rep*: $Inl\text{-}Rep\ a \neq Inr\text{-}Rep\ b$
⟨*proof*⟩

**definition** $Inl :: 'a \Rightarrow 'a + 'b$
**where** $Inl = Abs\text{-}sum \circ Inl\text{-}Rep$

**definition** $Inr :: 'b \Rightarrow 'a + 'b$
**where** $Inr = Abs\text{-}sum \circ Inr\text{-}Rep$

**lemma** *inj-Inl* [*simp*]: $inj\text{-}on\ Inl\ A$
⟨*proof*⟩

**lemma** *Inl-inject*: $Inl\ x = Inl\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-Inr* [*simp*]: $inj\text{-}on\ Inr\ A$
⟨*proof*⟩

**lemma** *Inr-inject*: $Inr\ x = Inr\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *Inl-not-Inr*: $Inl\ a \neq Inr\ b$
⟨*proof*⟩

**lemma** *Inr-not-Inl*: $Inr\ b \neq Inl\ a$
⟨*proof*⟩

**lemma** *sumE*:
**assumes** $\bigwedge x::'a.\ s = Inl\ x \Longrightarrow P$
**and** $\bigwedge y::'b.\ s = Inr\ y \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**free-constructors** *case-sum* **for**
*isl*: *Inl projl*
| *Inr projr*
⟨*proof*⟩

Avoid name clashes by prefixing the output of *old-rep-datatype* with *old*.

⟨*ML*⟩

**old-rep-datatype** *Inl Inr*
⟨*proof*⟩

⟨*ML*⟩

But erase the prefix for properties that are not generated by *free-constructors*.

⟨*ML*⟩

**declare**
*old.sum.inject*[*iff del*]
*old.sum.distinct*(*1*)[*simp del*, *induct-simp del*]

**lemmas** *induct* = *old.sum.induct*
**lemmas** *inducts* = *old.sum.inducts*
**lemmas** *rec* = *old.sum.rec*
**lemmas** *simps* = *sum.inject sum.distinct sum.case sum.rec*

⟨*ML*⟩

**primrec** *map-sum* :: $('a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'd) \Rightarrow 'a + 'b \Rightarrow 'c + 'd$
**where**
$map\text{-}sum\ f1\ f2\ (Inl\ a) = Inl\ (f1\ a)$
| $map\text{-}sum\ f1\ f2\ (Inr\ a) = Inr\ (f2\ a)$

**functor** *map-sum*: *map-sum*
⟨*proof*⟩

**lemma** *split-sum-all*: $(\forall x.\ P\ x) \longleftrightarrow (\forall x.\ P\ (Inl\ x)) \wedge (\forall x.\ P\ (Inr\ x))$
⟨*proof*⟩

**lemma** *split-sum-ex*: $(\exists x.\ P\ x) \longleftrightarrow (\exists x.\ P\ (Inl\ x)) \vee (\exists x.\ P\ (Inr\ x))$
⟨*proof*⟩

## 15.2 Projections

**lemma** *case-sum-KK* [*simp*]: $\mathit{case\text{-}sum}\ (\lambda x.\ a)\ (\lambda x.\ a) = (\lambda x.\ a)$
⟨*proof*⟩

**lemma** *surjective-sum*: $\mathit{case\text{-}sum}\ (\lambda x{::}'a.\ f\ (\mathit{Inl}\ x))\ (\lambda y{::}'b.\ f\ (\mathit{Inr}\ y)) = f$
⟨*proof*⟩

**lemma** *case-sum-inject*:
**assumes** *a*: $\mathit{case\text{-}sum}\ f1\ f2 = \mathit{case\text{-}sum}\ g1\ g2$
**and** *r*: $f1 = g1 \Longrightarrow f2 = g2 \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**primrec** *Suml* :: $('a \Rightarrow 'c) \Rightarrow 'a + 'b \Rightarrow 'c$
**where** $\mathit{Suml}\ f\ (\mathit{Inl}\ x) = f\ x$

**primrec** *Sumr* :: $('b \Rightarrow 'c) \Rightarrow 'a + 'b \Rightarrow 'c$
**where** $\mathit{Sumr}\ f\ (\mathit{Inr}\ x) = f\ x$

**lemma** *Suml-inject*:
**assumes** $\mathit{Suml}\ f = \mathit{Suml}\ g$
**shows** $f = g$
⟨*proof*⟩

**lemma** *Sumr-inject*:
**assumes** $\mathit{Sumr}\ f = \mathit{Sumr}\ g$
**shows** $f = g$
⟨*proof*⟩

## 15.3 The Disjoint Sum of Sets

**definition** *Plus* :: $'a\ set \Rightarrow 'b\ set \Rightarrow ('a + 'b)\ set$ (**infixr** ‹<+>› *65*)
**where** $A <+> B = \mathit{Inl}\ `\ A \cup \mathit{Inr}\ `\ B$

**hide-const** (**open**) *Plus* — Valuable identifier

**lemma** *InlI* [*intro*!]: $a \in A \Longrightarrow \mathit{Inl}\ a \in A <+> B$
⟨*proof*⟩

**lemma** *InrI* [*intro*!]: $b \in B \Longrightarrow \mathit{Inr}\ b \in A <+> B$
⟨*proof*⟩

Exhaustion rule for sums, a degenerate form of induction

**lemma** *PlusE* [*elim*!]:
$u \in A <+> B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow u = \mathit{Inl}\ x \Longrightarrow P) \Longrightarrow (\bigwedge y.\ y \in B \Longrightarrow u = \mathit{Inr}\ y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *Plus-eq-empty-conv* [*simp*]: $A <+> B = \{\} \longleftrightarrow A = \{\} \wedge B = \{\}$

⟨*proof*⟩

**lemma** *UNIV-Plus-UNIV* [*simp*]: *UNIV* <+> *UNIV* = *UNIV*
⟨*proof*⟩

**lemma** *UNIV-sum*: *UNIV* = *Inl* ‘ *UNIV* ∪ *Inr* ‘ *UNIV*
⟨*proof*⟩

**hide-const** (**open**) *Suml Sumr sum*

**end**

# 16 Rings

**theory** *Rings*
**imports** *Groups Set Fun*
**begin**

## 16.1 Semirings and rings

**class** *semiring* = *ab-semigroup-add* + *semigroup-mult* +
**assumes** *distrib-right* [*algebra-simps*, *algebra-split-simps*]: $(a + b) * c = a * c + b * c$
**assumes** *distrib-left* [*algebra-simps*, *algebra-split-simps*]: $a * (b + c) = a * b + a * c$
**begin**

For the *combine-numerals* simproc

**lemma** *combine-common-factor*: $a * e + (b * e + c) = (a + b) * e + c$
⟨*proof*⟩

**end**

**class** *mult-zero* = *times* + *zero* +
**assumes** *mult-zero-left* [*simp*]: $0 * a = 0$
**assumes** *mult-zero-right* [*simp*]: $a * 0 = 0$
**begin**

**lemma** *mult-not-zero*: $a * b \neq 0 \Longrightarrow a \neq 0 \wedge b \neq 0$
⟨*proof*⟩

**end**

**class** *semiring-0* = *semiring* + *comm-monoid-add* + *mult-zero*

**class** *semiring-0-cancel* = *semiring* + *cancel-comm-monoid-add*
**begin**

**subclass** *semiring-0*

⟨*proof*⟩

**end**

**class** *comm-semiring* = *ab-semigroup-add* + *ab-semigroup-mult* +
  **assumes** *distrib*: $(a + b) * c = a * c + b * c$
**begin**

**subclass** *semiring*
⟨*proof*⟩

**end**

**class** *comm-semiring-0* = *comm-semiring* + *comm-monoid-add* + *mult-zero*
**begin**

**subclass** *semiring-0* ⟨*proof*⟩

**end**

**class** *comm-semiring-0-cancel* = *comm-semiring* + *cancel-comm-monoid-add*
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *comm-semiring-0* ⟨*proof*⟩

**end**

**class** *zero-neq-one* = *zero* + *one* +
  **assumes** *zero-neq-one* [*simp*]: $0 \neq 1$
**begin**

**lemma** *one-neq-zero* [*simp*]: $1 \neq 0$
  ⟨*proof*⟩

**definition** *of-bool* :: *bool* ⇒ *'a*
  **where** *of-bool p* = (*if p then 1 else 0*)

**lemma** *of-bool-eq* [*simp*, *code*]:
  *of-bool False* = *0*
  *of-bool True* = *1*
  ⟨*proof*⟩

**lemma** *of-bool-eq-iff*: *of-bool p* = *of-bool q* ⟷ *p* = *q*
  ⟨*proof*⟩

**lemma** *split-of-bool* [*split*]: *P* (*of-bool p*) ⟷ (*p* ⟶ *P 1*) ∧ (¬ *p* ⟶ *P 0*)
  ⟨*proof*⟩

**lemma** *split-of-bool-asm*: $P$ (*of-bool* $p$) $\longleftrightarrow \neg$ ($p \wedge \neg$ $P$ *1* $\vee \neg$ $p \wedge \neg$ $P$ *0*)
⟨*proof*⟩

**lemma** *of-bool-eq-0-iff* [*simp*]:
‹*of-bool* $P$ = *0* $\longleftrightarrow \neg$ $P$›
⟨*proof*⟩

**lemma** *of-bool-eq-1-iff* [*simp*]:
‹*of-bool* $P$ = *1* $\longleftrightarrow$ $P$›
⟨*proof*⟩

**end**

**class** *semiring-1* = *zero-neq-one* + *semiring-0* + *monoid-mult*
**begin**

**lemma** *of-bool-conj*:
*of-bool* ($P \wedge Q$) = *of-bool* $P$ $*$ *of-bool* $Q$
⟨*proof*⟩

**end**

**lemma** *lambda-zero*: ($\lambda h$::'$a$::*mult-zero*. *0*) = ($*$) *0*
⟨*proof*⟩

**lemma** *lambda-one*: ($\lambda x$::'$a$::*monoid-mult*. $x$) = ($*$) *1*
⟨*proof*⟩

## 16.2 Abstract divisibility

**class** *dvd* = *times*
**begin**

**definition** *dvd* :: '$a \Rightarrow$ '$a \Rightarrow$ *bool* (**infix** ‹*dvd*› *50*)
**where** $b$ *dvd* $a \longleftrightarrow (\exists\, k.\ a = b * k)$

**lemma** *dvdI* [*intro?*]: $a = b * k \Longrightarrow b$ *dvd* $a$
⟨*proof*⟩

**lemma** *dvdE* [*elim*]: $b$ *dvd* $a \Longrightarrow (\bigwedge k.\ a = b * k \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**end**

**context** *comm-monoid-mult*
**begin**

**subclass** *dvd* ⟨*proof*⟩

**lemma** *dvd-refl* [*simp*]: *a dvd a*
⟨*proof*⟩

**lemma** *dvd-trans* [*trans*]:
**assumes** *a dvd b* **and** *b dvd c*
**shows** *a dvd c*
⟨*proof*⟩

**lemma** *subset-divisors-dvd*: $\{c.\ c\ dvd\ a\} \subseteq \{c.\ c\ dvd\ b\} \longleftrightarrow a\ dvd\ b$
⟨*proof*⟩

**lemma** *strict-subset-divisors-dvd*: $\{c.\ c\ dvd\ a\} \subset \{c.\ c\ dvd\ b\} \longleftrightarrow a\ dvd\ b \wedge \neg\ b\ dvd\ a$
⟨*proof*⟩

**lemma** *one-dvd* [*simp*]: *1 dvd a*
⟨*proof*⟩

**lemma** *dvd-mult* [*simp*]: $a\ dvd\ (b * c)$ **if** *a dvd c*
⟨*proof*⟩

**lemma** *dvd-mult2* [*simp*]: $a\ dvd\ (b * c)$ **if** *a dvd b*
⟨*proof*⟩

**lemma** *dvd-triv-right* [*simp*]: $a\ dvd\ b * a$
⟨*proof*⟩

**lemma** *dvd-triv-left* [*simp*]: $a\ dvd\ a * b$
⟨*proof*⟩

**lemma** *mult-dvd-mono*:
**assumes** *a dvd b*
**and** *c dvd d*
**shows** $a * c\ dvd\ b * d$
⟨*proof*⟩

**lemma** *dvd-mult-left*: $a * b\ dvd\ c \Longrightarrow a\ dvd\ c$
⟨*proof*⟩

**lemma** *dvd-mult-right*: $a * b\ dvd\ c \Longrightarrow b\ dvd\ c$
⟨*proof*⟩

**end**

**class** *comm-semiring-1* = *zero-neq-one* + *comm-semiring-0* + *comm-monoid-mult*
**begin**

**subclass** *semiring-1* ⟨*proof*⟩

**lemma** *dvd-0-left-iff* [*simp*]: *0 dvd a* $\longleftrightarrow$ *a* = *0*
⟨*proof*⟩

**lemma** *dvd-0-right* [*iff*]: *a dvd 0*
⟨*proof*⟩

**lemma** *dvd-0-left*: *0 dvd a* $\Longrightarrow$ *a* = *0*
⟨*proof*⟩

**lemma** *dvd-add* [*simp*]:
**assumes** *a dvd b* **and** *a dvd c*
**shows** *a dvd* (*b* + *c*)
⟨*proof*⟩

**end**

**class** *semiring-1-cancel* = *semiring* + *cancel-comm-monoid-add*
+ *zero-neq-one* + *monoid-mult*
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *semiring-1* ⟨*proof*⟩

**end**

**class** *comm-semiring-1-cancel* =
*comm-semiring* + *cancel-comm-monoid-add* + *zero-neq-one* + *comm-monoid-mult*
+
**assumes** *right-diff-distrib'* [*algebra-simps*, *algebra-split-simps*]:
$a * (b - c) = a * b - a * c$
**begin**

**subclass** *semiring-1-cancel* ⟨*proof*⟩
**subclass** *comm-semiring-0-cancel* ⟨*proof*⟩
**subclass** *comm-semiring-1* ⟨*proof*⟩

**lemma** *left-diff-distrib'* [*algebra-simps*, *algebra-split-simps*]:
$(b - c) * a = b * a - c * a$
⟨*proof*⟩

**lemma** *dvd-add-times-triv-left-iff* [*simp*]: *a dvd c* ∗ *a* + *b* $\longleftrightarrow$ *a dvd b*
⟨*proof*⟩

**lemma** *dvd-add-times-triv-right-iff* [*simp*]: *a dvd b* + *c* ∗ *a* $\longleftrightarrow$ *a dvd b*
⟨*proof*⟩

**lemma** *dvd-add-triv-left-iff* [*simp*]: *a dvd a* + *b* $\longleftrightarrow$ *a dvd b*

⟨*proof*⟩

**lemma** *dvd-add-triv-right-iff* [*simp*]: $a$ *dvd* $b + a \longleftrightarrow a$ *dvd* $b$
⟨*proof*⟩

**lemma** *dvd-add-right-iff*:
**assumes** $a$ *dvd* $b$
**shows** $a$ *dvd* $b + c \longleftrightarrow a$ *dvd* $c$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *dvd-add-left-iff*: $a$ *dvd* $c \Longrightarrow a$ *dvd* $b + c \longleftrightarrow a$ *dvd* $b$
⟨*proof*⟩

**end**

**class** *ring* = *semiring* + *ab-group-add*
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

Distribution rules

**lemma** *minus-mult-left*: $- (a * b) = - a * b$
⟨*proof*⟩

**lemma** *minus-mult-right*: $- (a * b) = a * - b$
⟨*proof*⟩

Extract signs from products

**lemmas** *mult-minus-left* [*simp*] = *minus-mult-left* [*symmetric*]
**lemmas** *mult-minus-right* [*simp*] = *minus-mult-right* [*symmetric*]

**lemma** *minus-mult-minus* [*simp*]: $- a * - b = a * b$
⟨*proof*⟩

**lemma** *minus-mult-commute*: $- a * b = a * - b$
⟨*proof*⟩

**lemma** *right-diff-distrib* [*algebra-simps*, *algebra-split-simps*]:
$a * (b - c) = a * b - a * c$
⟨*proof*⟩

**lemma** *left-diff-distrib* [*algebra-simps*, *algebra-split-simps*]:
$(a - b) * c = a * c - b * c$
⟨*proof*⟩

**lemmas** *ring-distribs* = *distrib-left distrib-right left-diff-distrib right-diff-distrib*

**lemma** *eq-add-iff1*: $a * e + c = b * e + d \longleftrightarrow (a - b) * e + c = d$
⟨*proof*⟩

**lemma** *eq-add-iff2*: $a * e + c = b * e + d \longleftrightarrow c = (b - a) * e + d$
⟨*proof*⟩

**end**

**lemmas** *ring-distribs* = *distrib-left distrib-right left-diff-distrib right-diff-distrib*

**class** *comm-ring* = *comm-semiring* + *ab-group-add*
**begin**

**subclass** *ring* ⟨*proof*⟩
**subclass** *comm-semiring-0-cancel* ⟨*proof*⟩

**lemma** *square-diff-square-factored*: $x * x - y * y = (x + y) * (x - y)$
⟨*proof*⟩

**end**

**class** *ring-1* = *ring* + *zero-neq-one* + *monoid-mult*
**begin**

**subclass** *semiring-1-cancel* ⟨*proof*⟩

**lemma** *of-bool-not-iff*:
‹*of-bool* $(\neg P) = 1 - $ *of-bool* $P$›
⟨*proof*⟩

**lemma** *square-diff-one-factored*: $x * x - 1 = (x + 1) * (x - 1)$
⟨*proof*⟩

**end**

**class** *comm-ring-1* = *comm-ring* + *zero-neq-one* + *comm-monoid-mult*
**begin**

**subclass** *ring-1* ⟨*proof*⟩
**subclass** *comm-semiring-1-cancel*
⟨*proof*⟩

**lemma** *dvd-minus-iff* [*simp*]: $x$ *dvd* $- y \longleftrightarrow x$ *dvd* $y$
⟨*proof*⟩

**lemma** *minus-dvd-iff* [*simp*]: $- x$ *dvd* $y \longleftrightarrow x$ *dvd* $y$
⟨*proof*⟩

**lemma** *dvd-diff-right-iff*:
**assumes** $a$ *dvd* $b$
**shows** $a$ *dvd* $b - c \longleftrightarrow a$ *dvd* $c$ (**is** $?P \longleftrightarrow ?Q$)

⟨*proof*⟩

**lemma** *dvd-diff-left-iff*:
**shows** $a\ dvd\ c \Longrightarrow a\ dvd\ b - c \longleftrightarrow a\ dvd\ b$
⟨*proof*⟩

**lemma** *dvd-diff* [*simp*]: $x\ dvd\ y \Longrightarrow x\ dvd\ z \Longrightarrow x\ dvd\ (y - z)$
⟨*proof*⟩

**end**

## 16.3 Towards integral domains

**class** *semiring-no-zero-divisors* = *semiring-0* +
**assumes** *no-zero-divisors*: $a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow a * b \neq 0$
**begin**

**lemma** *divisors-zero*:
**assumes** $a * b = 0$
**shows** $a = 0 \lor b = 0$
⟨*proof*⟩

**lemma** *mult-eq-0-iff* [*simp*]: $a * b = 0 \longleftrightarrow a = 0 \lor b = 0$
⟨*proof*⟩

**end**

**class** *semiring-1-no-zero-divisors* = *semiring-1* + *semiring-no-zero-divisors*

**class** *semiring-no-zero-divisors-cancel* = *semiring-no-zero-divisors* +
**assumes** *mult-cancel-right* [*simp*]: $a * c = b * c \longleftrightarrow c = 0 \lor a = b$
**and** *mult-cancel-left* [*simp*]: $c * a = c * b \longleftrightarrow c = 0 \lor a = b$
**begin**

**lemma** *mult-left-cancel*: $c \neq 0 \Longrightarrow c * a = c * b \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *mult-right-cancel*: $c \neq 0 \Longrightarrow a * c = b * c \longleftrightarrow a = b$
⟨*proof*⟩

**end**

**class** *ring-no-zero-divisors* = *ring* + *semiring-no-zero-divisors*
**begin**

**subclass** *semiring-no-zero-divisors-cancel*
⟨*proof*⟩

**end**

**class** *ring-1-no-zero-divisors* = *ring-1* + *ring-no-zero-divisors*
**begin**

**subclass** *semiring-1-no-zero-divisors* ⟨*proof*⟩

**lemma** *square-eq-1-iff*: $x * x = 1 \longleftrightarrow x = 1 \lor x = - 1$
⟨*proof*⟩

**lemma** *mult-cancel-right1* [*simp*]: $c = b * c \longleftrightarrow c = 0 \lor b = 1$
⟨*proof*⟩

**lemma** *mult-cancel-right2* [*simp*]: $a * c = c \longleftrightarrow c = 0 \lor a = 1$
⟨*proof*⟩

**lemma** *mult-cancel-left1* [*simp*]: $c = c * b \longleftrightarrow c = 0 \lor b = 1$
⟨*proof*⟩

**lemma** *mult-cancel-left2* [*simp*]: $c * a = c \longleftrightarrow c = 0 \lor a = 1$
⟨*proof*⟩

**end**

**class** *semidom* = *comm-semiring-1-cancel* + *semiring-no-zero-divisors*
**begin**

**subclass** *semiring-1-no-zero-divisors* ⟨*proof*⟩

**end**

**class** *idom* = *comm-ring-1* + *semiring-no-zero-divisors*
**begin**

**subclass** *semidom* ⟨*proof*⟩

**subclass** *ring-1-no-zero-divisors* ⟨*proof*⟩

**lemma** *dvd-mult-cancel-right* [*simp*]:
$a * c \ dvd \ b * c \longleftrightarrow c = 0 \lor a \ dvd \ b$
⟨*proof*⟩

**lemma** *dvd-mult-cancel-left* [*simp*]:
$c * a \ dvd \ c * b \longleftrightarrow c = 0 \lor a \ dvd \ b$
⟨*proof*⟩

**lemma** *square-eq-iff*: $a * a = b * b \longleftrightarrow a = b \lor a = - b$
⟨*proof*⟩

**lemma** *inj-mult-left* [*simp*]: ‹$inj \ ((*) \ a) \longleftrightarrow a \neq 0$› (**is** ‹$?P \longleftrightarrow ?Q$›)

⟨proof⟩

**end**

**class** *idom-abs-sgn* = *idom* + *abs* + *sgn* +
**assumes** *sgn-mult-abs*: $sgn\ a * |a| = a$
**and** *sgn-sgn* [*simp*]: $sgn\ (sgn\ a) = sgn\ a$
**and** *abs-abs* [*simp*]: $||a|| = |a|$
**and** *abs-0* [*simp*]: $|0| = 0$
**and** *sgn-0* [*simp*]: $sgn\ 0 = 0$
**and** *sgn-1* [*simp*]: $sgn\ 1 = 1$
**and** *sgn-minus-1*: $sgn\ (-\ 1) = -\ 1$
**and** *sgn-mult*: $sgn\ (a * b) = sgn\ a * sgn\ b$
**begin**

**lemma** *sgn-eq-0-iff*:
$sgn\ a = 0 \longleftrightarrow a = 0$
⟨proof⟩

**lemma** *abs-eq-0-iff*:
$|a| = 0 \longleftrightarrow a = 0$
⟨proof⟩

**lemma** *abs-mult-sgn*:
$|a| * sgn\ a = a$
⟨proof⟩

**lemma** *abs-1* [*simp*]:
$|1| = 1$
⟨proof⟩

**lemma** *sgn-abs* [*simp*]:
$|sgn\ a| = of\text{-}bool\ (a \neq 0)$
⟨proof⟩

**lemma** *abs-sgn* [*simp*]:
$sgn\ |a| = of\text{-}bool\ (a \neq 0)$
⟨proof⟩

**lemma** *abs-mult*:
$|a * b| = |a| * |b|$
⟨proof⟩

**lemma** *sgn-minus* [*simp*]:
$sgn\ (-\ a) = -\ sgn\ a$
⟨proof⟩

**lemma** *abs-minus* [*simp*]:
$|-\ a| = |a|$

⟨*proof*⟩

**end**

## 16.4 (Partial) Division

**class** *divide* =
**fixes** *divide* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹*div*› *70*)

⟨*ML*⟩

**context** *semiring*
**begin**

**lemma** [*field-simps*, *field-split-simps*]:
**shows** *distrib-left-NO-MATCH*: *NO-MATCH* $(x\ div\ y)\ a \Longrightarrow a * (b + c) = a * b + a * c$
**and** *distrib-right-NO-MATCH*: *NO-MATCH* $(x\ div\ y)\ c \Longrightarrow (a + b) * c = a * c + b * c$
⟨*proof*⟩

**end**

**context** *ring*
**begin**

**lemma** [*field-simps*, *field-split-simps*]:
**shows** *left-diff-distrib-NO-MATCH*: *NO-MATCH* $(x\ div\ y)\ c \Longrightarrow (a - b) * c = a * c - b * c$
**and** *right-diff-distrib-NO-MATCH*: *NO-MATCH* $(x\ div\ y)\ a \Longrightarrow a * (b - c) = a * b - a * c$
⟨*proof*⟩

**end**

⟨*ML*⟩

**class** *divide-trivial* = *zero* + *one* + *divide* +
**assumes** *div-by-0* [*simp*]: ‹$a\ div\ 0 = 0$›
**and** *div-by-1* [*simp*]: ‹$a\ div\ 1 = a$›
**and** *div-0* [*simp*]: ‹$0\ div\ a = 0$›

Algebraic classes with division

**class** *semidom-divide* = *semidom* + *divide* +
**assumes** *nonzero-mult-div-cancel-right* [*simp*]: ‹$b \neq 0 \Longrightarrow (a * b)\ div\ b = a$›
**assumes** *semidom-div-by-0*: ‹$a\ div\ 0 = 0$›
**begin**

**lemma** *nonzero-mult-div-cancel-left* [*simp*]: ‹$a \neq 0 \Longrightarrow (a * b)\ div\ a = b$›

⟨*proof*⟩

**subclass** *divide-trivial*
⟨*proof*⟩

**subclass** *semiring-no-zero-divisors-cancel*
⟨*proof*⟩

**lemma** *div-self* [*simp*]: $a \neq 0 \Longrightarrow a\ div\ a = 1$
⟨*proof*⟩

**lemma** *dvd-div-eq-0-iff*:
**assumes** $b\ dvd\ a$
**shows** $a\ div\ b = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *dvd-div-eq-cancel*:
$a\ div\ c = b\ div\ c \Longrightarrow c\ dvd\ a \Longrightarrow c\ dvd\ b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *dvd-div-eq-iff*:
$c\ dvd\ a \Longrightarrow c\ dvd\ b \Longrightarrow a\ div\ c = b\ div\ c \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *inj-on-mult*:
*inj-on* $((*)\ a)\ A$ **if** $a \neq 0$
⟨*proof*⟩

**end**

**class** *idom-divide* = *idom* + *semidom-divide*
**begin**

**lemma** *dvd-neg-div*:
**assumes** $b\ dvd\ a$
**shows** $-\ a\ div\ b = -\ (a\ div\ b)$
⟨*proof*⟩

**lemma** *dvd-div-neg*:
**assumes** $b\ dvd\ a$
**shows** $a\ div\ -\ b = -\ (a\ div\ b)$
⟨*proof*⟩

**end**

**class** *algebraic-semidom* = *semidom-divide*
**begin**

Class *algebraic-semidom* enriches a integral domain by notions from algebra, like units in a ring. It is a separate class to avoid spoiling fields with notions which are degenerated there.

**lemma** *dvd-times-left-cancel-iff* [*simp*]:
**assumes** $a \neq 0$
**shows** $a * b$ *dvd* $a * c \longleftrightarrow b$ *dvd* $c$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *dvd-times-right-cancel-iff* [*simp*]:
**assumes** $a \neq 0$
**shows** $b * a$ *dvd* $c * a \longleftrightarrow b$ *dvd* $c$
⟨*proof*⟩

**lemma** *div-dvd-iff-mult*:
**assumes** $b \neq 0$ **and** $b$ *dvd* $a$
**shows** $a$ *div* $b$ *dvd* $c \longleftrightarrow a$ *dvd* $c * b$
⟨*proof*⟩

**lemma** *dvd-div-iff-mult*:
**assumes** $c \neq 0$ **and** $c$ *dvd* $b$
**shows** $a$ *dvd* $b$ *div* $c \longleftrightarrow a * c$ *dvd* $b$
⟨*proof*⟩

**lemma** *div-dvd-div* [*simp*]:
**assumes** $a$ *dvd* $b$ **and** $a$ *dvd* $c$
**shows** $b$ *div* $a$ *dvd* $c$ *div* $a \longleftrightarrow b$ *dvd* $c$
⟨*proof*⟩

**lemma** *div-add* [*simp*]:
**assumes** $c$ *dvd* $a$ **and** $c$ *dvd* $b$
**shows** $(a + b)$ *div* $c = a$ *div* $c + b$ *div* $c$
⟨*proof*⟩

**lemma** *div-mult-div-if-dvd*:
**assumes** $b$ *dvd* $a$ **and** $d$ *dvd* $c$
**shows** $(a$ *div* $b) * (c$ *div* $d) = (a * c)$ *div* $(b * d)$
⟨*proof*⟩

**lemma** *dvd-div-eq-mult*:
**assumes** $a \neq 0$ **and** $a$ *dvd* $b$
**shows** $b$ *div* $a = c \longleftrightarrow b = c * a$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *dvd-div-mult-self* [*simp*]: $a$ *dvd* $b \Longrightarrow b$ *div* $a * a = b$
⟨*proof*⟩

**lemma** *dvd-mult-div-cancel* [*simp*]: $a$ *dvd* $b \Longrightarrow a * (b$ *div* $a) = b$
⟨*proof*⟩

**lemma** *div-mult-swap*:
**assumes** $c$ *dvd* $b$
**shows** $a * (b \ div \ c) = (a * b) \ div \ c$
⟨*proof*⟩

**lemma** *dvd-div-mult*: $c$ *dvd* $b \Longrightarrow b \ div \ c * a = (b * a) \ div \ c$
⟨*proof*⟩

**lemma** *dvd-div-mult2-eq*:
**assumes** $b * c$ *dvd* $a$
**shows** $a \ div \ (b * c) = a \ div \ b \ div \ c$
⟨*proof*⟩

**lemma** *dvd-div-div-eq-mult*:
**assumes** $a \neq 0$ $c \neq 0$ **and** $a$ *dvd* $b$ $c$ *dvd* $d$
**shows** $b \ div \ a = d \ div \ c \longleftrightarrow b * c = a * d$
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

**lemma** *dvd-mult-imp-div*:
**assumes** $a * c$ *dvd* $b$
**shows** $a$ *dvd* $b \ div \ c$
⟨*proof*⟩

**lemma** *div-div-eq-right*:
**assumes** $c$ *dvd* $b$ $b$ *dvd* $a$
**shows** $a \ div \ (b \ div \ c) = a \ div \ b * c$
⟨*proof*⟩

**lemma** *div-div-div-same*:
**assumes** $d$ *dvd* $b$ $b$ *dvd* $a$
**shows** $(a \ div \ d) \ div \ (b \ div \ d) = a \ div \ b$
⟨*proof*⟩

Units: invertible elements in a ring

**abbreviation** *is-unit* :: $'a \Rightarrow bool$
**where** *is-unit* $a \equiv a$ *dvd* *1*

**lemma** *not-is-unit-0* [*simp*]: ¬ *is-unit 0*
⟨*proof*⟩

**lemma** *unit-imp-dvd* [*dest*]: *is-unit* $b \Longrightarrow b$ *dvd* $a$
⟨*proof*⟩

**lemma** *unit-dvdE*:
**assumes** *is-unit* $a$
**obtains** $c$ **where** $a \neq 0$ **and** $b = a * c$
⟨*proof*⟩

**lemma** *dvd-unit-imp-unit*: $a$ *dvd* $b \Longrightarrow$ *is-unit* $b \Longrightarrow$ *is-unit* $a$
⟨*proof*⟩

**lemma** *unit-div-1-unit* [*simp*, *intro*]:
**assumes** *is-unit* $a$
**shows** *is-unit* ($1$ *div* $a$)
⟨*proof*⟩

**lemma** *is-unitE* [*elim?*]:
**assumes** *is-unit* $a$
**obtains** $b$ **where** $a \neq 0$ **and** $b \neq 0$
**and** *is-unit* $b$ **and** $1$ *div* $a = b$ **and** $1$ *div* $b = a$
**and** $a * b = 1$ **and** $c$ *div* $a = c * b$
⟨*proof*⟩

**lemma** *unit-prod* [*intro*]: *is-unit* $a \Longrightarrow$ *is-unit* $b \Longrightarrow$ *is-unit* $(a * b)$
⟨*proof*⟩

**lemma** *is-unit-mult-iff*: *is-unit* $(a * b) \longleftrightarrow$ *is-unit* $a \wedge$ *is-unit* $b$
⟨*proof*⟩

**lemma** *unit-div* [*intro*]: *is-unit* $a \Longrightarrow$ *is-unit* $b \Longrightarrow$ *is-unit* ($a$ *div* $b$)
⟨*proof*⟩

**lemma** *mult-unit-dvd-iff*:
**assumes** *is-unit* $b$
**shows** $a * b$ *dvd* $c \longleftrightarrow a$ *dvd* $c$
⟨*proof*⟩

**lemma** *mult-unit-dvd-iff'*: *is-unit* $a \Longrightarrow (a * b)$ *dvd* $c \longleftrightarrow b$ *dvd* $c$
⟨*proof*⟩

**lemma** *dvd-mult-unit-iff*:
**assumes** *is-unit* $b$
**shows** $a$ *dvd* $c * b \longleftrightarrow a$ *dvd* $c$
⟨*proof*⟩

**lemma** *dvd-mult-unit-iff'*: *is-unit* $b \Longrightarrow a$ *dvd* $b * c \longleftrightarrow a$ *dvd* $c$
⟨*proof*⟩

**lemma** *div-unit-dvd-iff*: *is-unit* $b \Longrightarrow a$ *div* $b$ *dvd* $c \longleftrightarrow a$ *dvd* $c$
⟨*proof*⟩

**lemma** *dvd-div-unit-iff*: *is-unit* $b \Longrightarrow a$ *dvd* $c$ *div* $b \longleftrightarrow a$ *dvd* $c$
⟨*proof*⟩

**lemmas** *unit-dvd-iff* = *mult-unit-dvd-iff mult-unit-dvd-iff'*
*dvd-mult-unit-iff dvd-mult-unit-iff'*

*div-unit-dvd-iff dvd-div-unit-iff*

**lemma** *unit-mult-div-div* [*simp*]: *is-unit* $a \Longrightarrow b * (1\ div\ a) = b\ div\ a$
⟨*proof*⟩

**lemma** *unit-div-mult-self* [*simp*]: *is-unit* $a \Longrightarrow b\ div\ a * a = b$
⟨*proof*⟩

**lemma** *unit-div-1-div-1* [*simp*]: *is-unit* $a \Longrightarrow 1\ div\ (1\ div\ a) = a$
⟨*proof*⟩

**lemma** *unit-div-mult-swap*: *is-unit* $c \Longrightarrow a * (b\ div\ c) = (a * b)\ div\ c$
⟨*proof*⟩

**lemma** *unit-div-commute*: *is-unit* $b \Longrightarrow (a\ div\ b) * c = (a * c)\ div\ b$
⟨*proof*⟩

**lemma** *unit-eq-div1*: *is-unit* $b \Longrightarrow a\ div\ b = c \longleftrightarrow a = c * b$
⟨*proof*⟩

**lemma** *unit-eq-div2*: *is-unit* $b \Longrightarrow a = c\ div\ b \longleftrightarrow a * b = c$
⟨*proof*⟩

**lemma** *unit-mult-left-cancel*: *is-unit* $a \Longrightarrow a * b = a * c \longleftrightarrow b = c$
⟨*proof*⟩

**lemma** *unit-mult-right-cancel*: *is-unit* $a \Longrightarrow b * a = c * a \longleftrightarrow b = c$
⟨*proof*⟩

**lemma** *unit-div-cancel*:
**assumes** *is-unit* $a$
**shows** $b\ div\ a = c\ div\ a \longleftrightarrow b = c$
⟨*proof*⟩

**lemma** *is-unit-div-mult2-eq*:
**assumes** *is-unit* $b$ **and** *is-unit* $c$
**shows** $a\ div\ (b * c) = a\ div\ b\ div\ c$
⟨*proof*⟩

**lemma** *is-unit-div-mult-cancel-left*:
**assumes** $a \neq 0$ **and** *is-unit* $b$
**shows** $a\ div\ (a * b) = 1\ div\ b$
⟨*proof*⟩

**lemma** *is-unit-div-mult-cancel-right*:
**assumes** $a \neq 0$ **and** *is-unit* $b$
**shows** $a\ div\ (b * a) = 1\ div\ b$
⟨*proof*⟩

**lemma** *unit-div-eq-0-iff*:
**assumes** *is-unit b*
**shows** $a\ div\ b = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *div-mult-unit2*:
$is\text{-}unit\ c \Longrightarrow b\ dvd\ a \Longrightarrow a\ div\ (b * c) = a\ div\ b\ div\ c$
⟨*proof*⟩

Coprimality

**definition** $coprime :: 'a \Rightarrow 'a \Rightarrow bool$
**where** $coprime\ a\ b \longleftrightarrow (\forall c.\ c\ dvd\ a \longrightarrow c\ dvd\ b \longrightarrow is\text{-}unit\ c)$

**lemma** *coprimeI*:
**assumes** $\bigwedge c.\ c\ dvd\ a \Longrightarrow c\ dvd\ b \Longrightarrow is\text{-}unit\ c$
**shows** *coprime a b*
⟨*proof*⟩

**lemma** *not-coprimeI*:
**assumes** *c dvd a* **and** *c dvd b* **and** $\neg$ *is-unit c*
**shows** $\neg$ *coprime a b*
⟨*proof*⟩

**lemma** *coprime-common-divisor*:
*is-unit c* **if** *coprime a b* **and** *c dvd a* **and** *c dvd b*
⟨*proof*⟩

**lemma** *not-coprimeE*:
**assumes** $\neg$ *coprime a b*
**obtains** *c* **where** *c dvd a* **and** *c dvd b* **and** $\neg$ *is-unit c*
⟨*proof*⟩

**lemma** *coprime-imp-coprime*:
*coprime a b* **if** *coprime c d*
**and** $\bigwedge e.\ \neg\ is\text{-}unit\ e \Longrightarrow e\ dvd\ a \Longrightarrow e\ dvd\ b \Longrightarrow e\ dvd\ c$
**and** $\bigwedge e.\ \neg\ is\text{-}unit\ e \Longrightarrow e\ dvd\ a \Longrightarrow e\ dvd\ b \Longrightarrow e\ dvd\ d$
⟨*proof*⟩

**lemma** *coprime-divisors*:
*coprime a b* **if** *a dvd c b dvd d* **and** *coprime c d*
⟨*proof*⟩

**lemma** *coprime-self* [*simp*]:
$coprime\ a\ a \longleftrightarrow is\text{-}unit\ a$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *coprime-commute* [*ac-simps*]:
$coprime\ b\ a \longleftrightarrow coprime\ a\ b$
⟨*proof*⟩

**lemma** *is-unit-left-imp-coprime*:
*coprime a b* **if** *is-unit a*
⟨*proof*⟩

**lemma** *is-unit-right-imp-coprime*:
*coprime a b* **if** *is-unit b*
⟨*proof*⟩

**lemma** *coprime-1-left* [*simp*]:
*coprime 1 a*
⟨*proof*⟩

**lemma** *coprime-1-right* [*simp*]:
*coprime a 1*
⟨*proof*⟩

**lemma** *coprime-0-left-iff* [*simp*]:
*coprime 0 a* $\longleftrightarrow$ *is-unit a*
⟨*proof*⟩

**lemma** *coprime-0-right-iff* [*simp*]:
*coprime a 0* $\longleftrightarrow$ *is-unit a*
⟨*proof*⟩

**lemma** *coprime-mult-self-left-iff* [*simp*]:
*coprime* $(c * a)$ $(c * b)$ $\longleftrightarrow$ *is-unit c* $\wedge$ *coprime a b*
⟨*proof*⟩

**lemma** *coprime-mult-self-right-iff* [*simp*]:
*coprime* $(a * c)$ $(b * c)$ $\longleftrightarrow$ *is-unit c* $\wedge$ *coprime a b*
⟨*proof*⟩

**lemma** *coprime-absorb-left*:
**assumes** *x dvd y*
**shows** *coprime x y* $\longleftrightarrow$ *is-unit x*
⟨*proof*⟩

**lemma** *coprime-absorb-right*:
**assumes** *y dvd x*
**shows** *coprime x y* $\longleftrightarrow$ *is-unit y*
⟨*proof*⟩

**end**

**class** *unit-factor* =
**fixes** *unit-factor* :: $'a \Rightarrow 'a$

**class** *semidom-divide-unit-factor* = *semidom-divide* + *unit-factor* +

**assumes** *unit-factor-0* [*simp*]: *unit-factor 0 = 0*
**and** *is-unit-unit-factor*: *a dvd 1* $\Longrightarrow$ *unit-factor a = a*
**and** *unit-factor-is-unit*: *a* $\neq$ *0* $\Longrightarrow$ *unit-factor a dvd 1*
**and** *unit-factor-mult-unit-left*: *a dvd 1* $\Longrightarrow$ *unit-factor* (*a* $*$ *b*) = *a* $*$ *unit-factor b*
— This fine-grained hierarchy will later on allow lean normalization of polynomials
**begin**

**lemma** *unit-factor-mult-unit-right*: *a dvd 1* $\Longrightarrow$ *unit-factor* (*b* $*$ *a*) = *unit-factor b* $*$ *a*
⟨*proof*⟩

**lemmas** [*simp*] = *unit-factor-mult-unit-left unit-factor-mult-unit-right*

**end**

**class** *normalization-semidom* = *algebraic-semidom* + *semidom-divide-unit-factor* +
**fixes** *normalize* :: *'a* $\Rightarrow$ *'a*
**assumes** *unit-factor-mult-normalize* [*simp*]: *unit-factor a* $*$ *normalize a = a*
**and** *normalize-0* [*simp*]: *normalize 0 = 0*
**begin**

Class *normalization-semidom* cultivates the idea that each integral domain can be split into equivalence classes whose representants are associated, i.e. divide each other. *normalize* specifies a canonical representant for each equivalence class. The rationale behind this is that it is easier to reason about equality than equivalences, hence we prefer to think about equality of normalized values rather than associated elements.

**declare** *unit-factor-is-unit* [*iff*]

**lemma** *unit-factor-dvd* [*simp*]: *a* $\neq$ *0* $\Longrightarrow$ *unit-factor a dvd b*
⟨*proof*⟩

**lemma** *unit-factor-self* [*simp*]: *unit-factor a dvd a*
⟨*proof*⟩

**lemma** *normalize-mult-unit-factor* [*simp*]: *normalize a* $*$ *unit-factor a = a*
⟨*proof*⟩

**lemma** *normalize-eq-0-iff* [*simp*]: *normalize a = 0* $\longleftrightarrow$ *a = 0*
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *unit-factor-eq-0-iff* [*simp*]: *unit-factor a = 0* $\longleftrightarrow$ *a = 0*
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *div-unit-factor* [*simp*]: *a div unit-factor a = normalize a*

⟨proof⟩

**lemma** *normalize-div* [*simp*]: *normalize a div a* = *1 div unit-factor a*
⟨proof⟩

**lemma** *is-unit-normalize*:
**assumes** *is-unit a*
**shows** *normalize a* = *1*
⟨proof⟩

**lemma** *unit-factor-1* [*simp*]: *unit-factor 1* = *1*
⟨proof⟩

**lemma** *normalize-1* [*simp*]: *normalize 1* = *1*
⟨proof⟩

**lemma** *normalize-1-iff*: *normalize a* = *1* ⟷ *is-unit a*
(**is** *?lhs* ⟷ *?rhs*)
⟨proof⟩

**lemma** *div-normalize* [*simp*]: *a div normalize a* = *unit-factor a*
⟨proof⟩

**lemma** *mult-one-div-unit-factor* [*simp*]: *a* ∗ (*1 div unit-factor b*) = *a div unit-factor b*
⟨proof⟩

**lemma** *inv-unit-factor-eq-0-iff* [*simp*]:
*1 div unit-factor a* = *0* ⟷ *a* = *0*
(**is** *?lhs* ⟷ *?rhs*)
⟨proof⟩

**lemma** *unit-factor-idem* [*simp*]: *unit-factor* (*unit-factor a*) = *unit-factor a*
⟨proof⟩

**lemma** *normalize-unit-factor* [*simp*]: *a* ≠ *0* ⟹ *normalize* (*unit-factor a*) = *1*
⟨proof⟩

**lemma** *normalize-mult-unit-left* [*simp*]:
**assumes** *a dvd 1*
**shows** *normalize* (*a* ∗ *b*) = *normalize b*
⟨proof⟩

**lemma** *normalize-mult-unit-right* [*simp*]:
**assumes** *b dvd 1*
**shows** *normalize* (*a* ∗ *b*) = *normalize a*
⟨proof⟩

**lemma** *normalize-idem* [*simp*]: *normalize* (*normalize a*) = *normalize a*

⟨*proof*⟩

**lemma** *unit-factor-normalize* [*simp*]:
**assumes** $a \neq 0$
**shows** *unit-factor* (*normalize* *a*) = *1*
⟨*proof*⟩

**lemma** *normalize-dvd-iff* [*simp*]: *normalize a dvd b* ⟷ *a dvd b*
⟨*proof*⟩

**lemma** *dvd-normalize-iff* [*simp*]: *a dvd normalize b* ⟷ *a dvd b*
⟨*proof*⟩

**lemma** *normalize-idem-imp-unit-factor-eq*:
**assumes** *normalize a* = *a*
**shows** *unit-factor a* = *of-bool* ($a \neq 0$)
⟨*proof*⟩

**lemma** *normalize-idem-imp-is-unit-iff*:
**assumes** *normalize a* = *a*
**shows** *is-unit a* ⟷ *a* = *1*
⟨*proof*⟩

**lemma** *coprime-normalize-left-iff* [*simp*]:
*coprime* (*normalize a*) *b* ⟷ *coprime a b*
⟨*proof*⟩

**lemma** *coprime-normalize-right-iff* [*simp*]:
*coprime a* (*normalize b*) ⟷ *coprime a b*
⟨*proof*⟩

We avoid an explicit definition of associated elements but prefer explicit normalisation instead. In theory we could define an abbreviation like *associated a b* = (*normalize a* = *normalize b*) but this is counterproductive without suggestive infix syntax, which we do not want to sacrifice for this purpose here.

**lemma** *associatedI*:
**assumes** *a dvd b* **and** *b dvd a*
**shows** *normalize a* = *normalize b*
⟨*proof*⟩

**lemma** *associatedD1*: *normalize a* = *normalize b* ⟹ *a dvd b*
⟨*proof*⟩

**lemma** *associatedD2*: *normalize a* = *normalize b* ⟹ *b dvd a*
⟨*proof*⟩

**lemma** *associated-unit*: *normalize a* = *normalize b* ⟹ *is-unit a* ⟹ *is-unit b*
⟨*proof*⟩

**lemma** *associated-iff-dvd*: *normalize* $a$ = *normalize* $b \longleftrightarrow a$ *dvd* $b \wedge b$ *dvd* $a$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *associated-eqI*:
**assumes** $a$ *dvd* $b$ **and** $b$ *dvd* $a$
**assumes** *normalize* $a = a$ **and** *normalize* $b = b$
**shows** $a = b$
⟨*proof*⟩

**lemma** *normalize-unit-factor-eqI*:
**assumes** *normalize* $a$ = *normalize* $b$
**and** *unit-factor* $a$ = *unit-factor* $b$
**shows** $a = b$
⟨*proof*⟩

**lemma** *normalize-mult-normalize-left* [*simp*]: *normalize* (*normalize* $a * b$) = *normalize* ($a * b$)
⟨*proof*⟩

**lemma** *normalize-mult-normalize-right* [*simp*]: *normalize* ($a *$ *normalize* $b$) = *normalize* ($a * b$)
⟨*proof*⟩

**end**

**class** *normalization-semidom-multiplicative* = *normalization-semidom* +
**assumes** *unit-factor-mult*: *unit-factor* ($a * b$) = *unit-factor* $a *$ *unit-factor* $b$
**begin**

**lemma** *normalize-mult*: *normalize* ($a * b$) = *normalize* $a *$ *normalize* $b$
⟨*proof*⟩

**lemma** *dvd-unit-factor-div*:
**assumes** $b$ *dvd* $a$
**shows** *unit-factor* ($a$ *div* $b$) = *unit-factor* $a$ *div* *unit-factor* $b$
⟨*proof*⟩

**lemma** *dvd-normalize-div*:
**assumes** $b$ *dvd* $a$
**shows** *normalize* ($a$ *div* $b$) = *normalize* $a$ *div* *normalize* $b$
⟨*proof*⟩

**end**

Syntactic division remainder operator

**class** *modulo* = *dvd* + *divide* +

**fixes** *modulo* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹*mod*› *70*)

Arbitrary quotient and remainder partitions

**class** *semiring-modulo* = *comm-semiring-1-cancel* + *divide* + *modulo* +
**assumes** *div-mult-mod-eq*: ‹$a$ *div* $b * b + a$ *mod* $b = a$›
**begin**

**lemma** *mod-div-decomp*:
**fixes** $a$ $b$
**obtains** $q$ $r$ **where** $q = a$ *div* $b$ **and** $r = a$ *mod* $b$
**and** $a = q * b + r$
⟨*proof*⟩

**lemma** *mult-div-mod-eq*: $b * (a$ *div* $b) + a$ *mod* $b = a$
⟨*proof*⟩

**lemma** *mod-div-mult-eq*: $a$ *mod* $b + a$ *div* $b * b = a$
⟨*proof*⟩

**lemma** *mod-mult-div-eq*: $a$ *mod* $b + b * (a$ *div* $b) = a$
⟨*proof*⟩

**lemma** *minus-div-mult-eq-mod*: $a - a$ *div* $b * b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *minus-mult-div-eq-mod*: $a - b * (a$ *div* $b) = a$ *mod* $b$
⟨*proof*⟩

**lemma** *minus-mod-eq-div-mult*: $a - a$ *mod* $b = a$ *div* $b * b$
⟨*proof*⟩

**lemma** *minus-mod-eq-mult-div*: $a - a$ *mod* $b = b * (a$ *div* $b)$
⟨*proof*⟩

**lemma** *mod-0-imp-dvd* [*dest*!]:
$b$ *dvd* $a$ **if** $a$ *mod* $b = 0$
⟨*proof*⟩

**lemma** [*nitpick-unfold*]:
$a$ *mod* $b = a - a$ *div* $b * b$
⟨*proof*⟩

**end**

**class** *semiring-modulo-trivial* = *semiring-modulo* + *divide-trivial*
**begin**

**lemma** *mod-0* [*simp*]:
‹*0* *mod* $a$ = *0*›

⟨*proof*⟩

**lemma** *mod-by-0* [*simp*]:
‹$a$ *mod* $0 = a$›
⟨*proof*⟩

**lemma** *mod-by-1* [*simp*]:
‹$a$ *mod* $1 = 0$›
⟨*proof*⟩

**end**

## 16.5 Quotient and remainder in integral domains

**class** *semidom-modulo* = *algebraic-semidom* + *semiring-modulo*
**begin**

**subclass** *semiring-modulo-trivial* ⟨*proof*⟩

**lemma** *mod-self* [*simp*]:
$a$ *mod* $a = 0$
⟨*proof*⟩

**lemma** *dvd-imp-mod-0* [*simp*]:
$b$ *mod* $a = 0$ **if** $a$ *dvd* $b$
⟨*proof*⟩

**lemma** *mod-eq-0-iff-dvd*:
$a$ *mod* $b = 0 \longleftrightarrow b$ *dvd* $a$
⟨*proof*⟩

**lemma** *dvd-eq-mod-eq-0* [*nitpick-unfold*, *code*]:
$a$ *dvd* $b \longleftrightarrow b$ *mod* $a = 0$
⟨*proof*⟩

**lemma** *dvd-mod-iff*:
**assumes** $c$ *dvd* $b$
**shows** $c$ *dvd* $a$ *mod* $b \longleftrightarrow c$ *dvd* $a$
⟨*proof*⟩

**lemma** *dvd-mod-imp-dvd*:
**assumes** $c$ *dvd* $a$ *mod* $b$ **and** $c$ *dvd* $b$
**shows** $c$ *dvd* $a$
⟨*proof*⟩

**lemma** *dvd-minus-mod* [*simp*]:
$b$ *dvd* $a -$ $a$ *mod* $b$
⟨*proof*⟩

**lemma** *cancel-div-mod-rules*:
$((a\ div\ b) * b + a\ mod\ b) + c = a + c$
$(b * (a\ div\ b) + a\ mod\ b) + c = a + c$
⟨*proof*⟩

**end**

**class** *idom-modulo* = *idom* + *semidom-modulo*
**begin**

**subclass** *idom-divide* ⟨*proof*⟩

**lemma** *div-diff* [*simp*]:
$c\ dvd\ a \Longrightarrow c\ dvd\ b \Longrightarrow (a - b)\ div\ c = a\ div\ c - b\ div\ c$
⟨*proof*⟩

**end**

## 16.6 Interlude: basic tool support for algebraic and arithmetic calculations

**named-theorems** *arith arith facts* −− *only ground formulas*
⟨*ML*⟩

## 16.7 Ordered semirings and rings

The theory of partially ordered rings is taken from the books:

- *Lattice Theory* by Garret Birkhoff, American Mathematical Society, 1979
- *Partially Ordered Algebraic Systems*, Pergamon Press, 1963

Most of the used notions can also be looked up in

- http://www.mathworld.com by Eric Weisstein et. al.
- *Algebra I* by van der Waerden, Springer

**class** *ordered-semiring* = *semiring* + *ordered-comm-monoid-add* +
**assumes** *mult-left-mono*: $a \le b \Longrightarrow 0 \le c \Longrightarrow c * a \le c * b$
**assumes** *mult-right-mono*: $a \le b \Longrightarrow 0 \le c \Longrightarrow a * c \le b * c$
**begin**

**lemma** *mult-mono*: $a \le b \Longrightarrow c \le d \Longrightarrow 0 \le b \Longrightarrow 0 \le c \Longrightarrow a * c \le b * d$
⟨*proof*⟩

**lemma** *mult-mono'*: $a \le b \Longrightarrow c \le d \Longrightarrow 0 \le a \Longrightarrow 0 \le c \Longrightarrow a * c \le b * d$

⟨*proof*⟩

**end**

**lemma** *mono-mult*:
**fixes** $a$ :: $'a$::*ordered-semiring*
**shows** $a \geq 0 \Longrightarrow mono\ ((*)\ a)$
⟨*proof*⟩

**class** *ordered-semiring-0* = *semiring-0* + *ordered-semiring*
**begin**

**lemma** *mult-nonneg-nonneg* [*simp*]: $0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 \leq a * b$
⟨*proof*⟩

**lemma** *mult-nonneg-nonpos*: $0 \leq a \Longrightarrow b \leq 0 \Longrightarrow a * b \leq 0$
⟨*proof*⟩

**lemma** *mult-nonpos-nonneg*: $a \leq 0 \Longrightarrow 0 \leq b \Longrightarrow a * b \leq 0$
⟨*proof*⟩

Legacy – use *mult-nonpos-nonneg*.

**lemma** *mult-nonneg-nonpos2*: $0 \leq a \Longrightarrow b \leq 0 \Longrightarrow b * a \leq 0$
⟨*proof*⟩

**lemma** *split-mult-neg-le*: $(0 \leq a \land b \leq 0) \lor (a \leq 0 \land 0 \leq b) \Longrightarrow a * b \leq 0$
⟨*proof*⟩

**end**

**class** *zero-less-one* = *order* + *zero* + *one* +
**assumes** *zero-less-one* [*simp*]: $0 < 1$
**begin**

**subclass** *zero-neq-one*
⟨*proof*⟩

**lemma** *zero-le-one* [*simp*]:
‹$0 \leq 1$› ⟨*proof*⟩

**end**

**class** *ordered-semiring-1* = *ordered-semiring-0* + *semiring-1* + *zero-less-one*
**begin**

**lemma** *convex-bound-le*:
**assumes** $x \leq a$ **and** $y \leq a$ **and** $0 \leq u$ **and** $0 \leq v$ **and** $u + v = 1$
**shows** $u * x + v * y \leq a$
⟨*proof*⟩

**end**

**class** *ordered-cancel-semiring* = *ordered-semiring* + *cancel-comm-monoid-add*
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *ordered-semiring-0* ⟨*proof*⟩

**subclass** *ordered-cancel-ab-semigroup-add* ⟨*proof*⟩

**end**

**class** *linordered-semiring* = *ordered-semiring* + *linordered-cancel-ab-semigroup-add*
**begin**

**subclass** *ordered-cancel-semiring* ⟨*proof*⟩

**subclass** *ordered-cancel-comm-monoid-add* ⟨*proof*⟩

**subclass** *ordered-ab-semigroup-monoid-add-imp-le* ⟨*proof*⟩

**lemma** *mult-left-less-imp-less*: $c * a < c * b \Longrightarrow 0 \leq c \Longrightarrow a < b$
⟨*proof*⟩

**lemma** *mult-right-less-imp-less*: $a * c < b * c \Longrightarrow 0 \leq c \Longrightarrow a < b$
⟨*proof*⟩

**end**

**class** *ordered-semiring-strict* = *semiring* + *comm-monoid-add* + *ordered-cancel-ab-semigroup-add* +
**assumes** *mult-strict-left-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow c * a < c * b$
**assumes** *mult-strict-right-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow a * c < b * c$
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *ordered-semiring*
⟨*proof*⟩

**lemma** *mult-pos-pos*[*simp*]: $0 < a \Longrightarrow 0 < b \Longrightarrow 0 < a * b$
⟨*proof*⟩

**lemma** *mult-pos-neg*: $0 < a \Longrightarrow b < 0 \Longrightarrow a * b < 0$
⟨*proof*⟩

**lemma** *mult-neg-pos*: $a < 0 \Longrightarrow 0 < b \Longrightarrow a * b < 0$
⟨*proof*⟩

Strict monotonicity in both arguments

**lemma** *mult-strict-mono*:
**assumes** $a < b$ $c < d$ $0 < b$ $0 \leq c$
**shows** $a * c < b * d$
⟨*proof*⟩

This weaker variant has more natural premises

**lemma** *mult-strict-mono'*:
**assumes** $a < b$ **and** $c < d$ **and** $0 \leq a$ **and** $0 \leq c$
**shows** $a * c < b * d$
⟨*proof*⟩

**lemma** *mult-less-le-imp-less*:
**assumes** $a < b$ $c \leq d$ $0 \leq a$ $0 < c$
**shows** $a * c < b * d$
⟨*proof*⟩

**lemma** *mult-le-less-imp-less*:
**assumes** $a \leq b$ **and** $c < d$ **and** $0 < a$ **and** $0 \leq c$
**shows** $a * c < b * d$
⟨*proof*⟩

**end**

**class** *linordered-semiring-1* = *linordered-semiring* + *semiring-1* + *zero-less-one*
**begin**

**lemma** *convex-bound-le*:
**assumes** $x \leq a$ $y \leq a$ $0 \leq u$ $0 \leq v$ $u + v = 1$
**shows** $u * x + v * y \leq a$
⟨*proof*⟩

**end**

**subclass** (**in** *linordered-semiring-1*) *ordered-semiring-1* ⟨*proof*⟩

**class** *linordered-semiring-strict* = *semiring* + *comm-monoid-add* + *linordered-cancel-ab-semigroup-add* +
**assumes** *mult-strict-left-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow c * a < c * b$
**assumes** *mult-strict-right-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow a * c < b * c$
**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *linordered-semiring*
⟨*proof*⟩

**subclass** (**in** *linordered-semiring-strict*) *ordered-semiring-strict*
⟨*proof*⟩

**lemma** *mult-left-le-imp-le*: $c * a \leq c * b \Longrightarrow 0 < c \Longrightarrow a \leq b$
⟨*proof*⟩

**lemma** *mult-right-le-imp-le*: $a * c \leq b * c \Longrightarrow 0 < c \Longrightarrow a \leq b$
⟨*proof*⟩

**lemma** *zero-less-mult-pos*:
**assumes** $0 < a * b$ $0 < a$ **shows** $0 < b$
⟨*proof*⟩

**lemma** *zero-less-mult-pos2*:
**assumes** $0 < b * a$ $0 < a$ **shows** $0 < b$
⟨*proof*⟩

**end**

**class** *linordered-semiring-1-strict* = *linordered-semiring-strict* + *semiring-1* + *zero-less-one*
**begin**

**subclass** *linordered-semiring-1* ⟨*proof*⟩

**lemma** *convex-bound-lt*:
**assumes** $x < a$ $y < a$ $0 \leq u$ $0 \leq v$ $u + v = 1$
**shows** $u * x + v * y < a$
⟨*proof*⟩

**end**

**class** *ordered-semiring-1-strict* = *ordered-semiring-strict* + *semiring-1* + *zero-less-one*
— analogous to *linordered-semiring-1-strict* not requiring a total order
**begin**

**subclass** *ordered-semiring-1* ⟨*proof*⟩

**lemma** *convex-bound-lt*:
**assumes** $x < a$ **and** $y < a$ **and** $0 \leq u$ **and** $0 \leq v$ **and** $u + v = 1$
**shows** $u * x + v * y < a$
⟨*proof*⟩

**end**

**subclass** (**in** *linordered-semiring-1-strict*) *ordered-semiring-1-strict* ⟨*proof*⟩

**class** *ordered-comm-semiring* = *comm-semiring-0* + *ordered-ab-semigroup-add* +
**assumes** *comm-mult-left-mono*: $a \leq b \Longrightarrow 0 \leq c \Longrightarrow c * a \leq c * b$

**begin**

**subclass** *ordered-semiring*
⟨*proof*⟩

**end**

**class** *ordered-cancel-comm-semiring* = *ordered-comm-semiring* + *cancel-comm-monoid-add*
**begin**

**subclass** *comm-semiring-0-cancel* ⟨*proof*⟩
**subclass** *ordered-comm-semiring* ⟨*proof*⟩
**subclass** *ordered-cancel-semiring* ⟨*proof*⟩

**end**

**class** *linordered-comm-semiring-strict* = *comm-semiring-0* + *linordered-cancel-ab-semigroup-add* +
**assumes** *comm-mult-strict-left-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow c * a < c * b$
**begin**

**subclass** *linordered-semiring-strict*
⟨*proof*⟩

**subclass** *ordered-cancel-comm-semiring*
⟨*proof*⟩

**end**

**class** *ordered-comm-semiring-strict* = *comm-semiring-0* + *ordered-cancel-ab-semigroup-add* +
— analogous to *linordered-comm-semiring-strict* not requiring a total order
**assumes** *comm-mult-strict-left-mono*: $a < b \Longrightarrow 0 < c \Longrightarrow c * a < c * b$
**begin**

**subclass** *ordered-semiring-strict*
⟨*proof*⟩

**subclass** *ordered-cancel-comm-semiring*
⟨*proof*⟩

**end**

**subclass** (**in** *linordered-comm-semiring-strict*) *ordered-comm-semiring-strict*
⟨*proof*⟩

**class** *ordered-ring* = *ring* + *ordered-cancel-semiring*
**begin**

**subclass** *ordered-ab-group-add* ⟨*proof*⟩

**lemma** *less-add-iff1*: $a * e + c < b * e + d \longleftrightarrow (a - b) * e + c < d$
⟨*proof*⟩

**lemma** *less-add-iff2*: $a * e + c < b * e + d \longleftrightarrow c < (b - a) * e + d$
⟨*proof*⟩

**lemma** *le-add-iff1*: $a * e + c \leq b * e + d \longleftrightarrow (a - b) * e + c \leq d$
⟨*proof*⟩

**lemma** *le-add-iff2*: $a * e + c \leq b * e + d \longleftrightarrow c \leq (b - a) * e + d$
⟨*proof*⟩

**lemma** *mult-left-mono-neg*: $b \leq a \Longrightarrow c \leq 0 \Longrightarrow c * a \leq c * b$
⟨*proof*⟩

**lemma** *mult-right-mono-neg*: $b \leq a \Longrightarrow c \leq 0 \Longrightarrow a * c \leq b * c$
⟨*proof*⟩

**lemma** *mult-nonpos-nonpos*: $a \leq 0 \Longrightarrow b \leq 0 \Longrightarrow 0 \leq a * b$
⟨*proof*⟩

**lemma** *split-mult-pos-le*: $(0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0) \Longrightarrow 0 \leq a * b$
⟨*proof*⟩

**end**

**class** *abs-if* = *minus* + *uminus* + *ord* + *zero* + *abs* +
**assumes** *abs-if*: $|a| = (\mathit{if}\ a < 0\ \mathit{then}\ - a\ \mathit{else}\ a)$

**class** *linordered-ring* = *ring* + *linordered-semiring* + *linordered-ab-group-add* + *abs-if*
**begin**

**subclass** *ordered-ring* ⟨*proof*⟩

**subclass** *ordered-ab-group-add-abs*
⟨*proof*⟩

**lemma** *zero-le-square* [*simp*]: $0 \leq a * a$
⟨*proof*⟩

**lemma** *not-square-less-zero* [*simp*]: $\neg (a * a < 0)$
⟨*proof*⟩

**proposition** *abs-eq-iff*: $|x| = |y| \longleftrightarrow x = y \vee x = -y$
⟨*proof*⟩

**lemma** *abs-eq-iff'*:
$|a| = b \longleftrightarrow b \geq 0 \wedge (a = b \vee a = - b)$
⟨*proof*⟩

**lemma** *eq-abs-iff'*:
$a = |b| \longleftrightarrow a \geq 0 \wedge (b = a \vee b = - a)$
⟨*proof*⟩

**lemma** *sum-squares-ge-zero*: $0 \leq x * x + y * y$
⟨*proof*⟩

**lemma** *not-sum-squares-lt-zero*: $\neg\, x * x + y * y < 0$
⟨*proof*⟩

**end**

**class** *linordered-ring-strict* = *ring* + *linordered-semiring-strict*
+ *ordered-ab-group-add* + *abs-if*
**begin**

**subclass** *linordered-ring* ⟨*proof*⟩

**lemma** *mult-strict-left-mono-neg*: $b < a \Longrightarrow c < 0 \Longrightarrow c * a < c * b$
⟨*proof*⟩

**lemma** *mult-strict-right-mono-neg*: $b < a \Longrightarrow c < 0 \Longrightarrow a * c < b * c$
⟨*proof*⟩

**lemma** *mult-neg-neg*: $a < 0 \Longrightarrow b < 0 \Longrightarrow 0 < a * b$
⟨*proof*⟩

**subclass** *ring-no-zero-divisors*
⟨*proof*⟩

**lemma** *zero-less-mult-iff* [*algebra-split-simps*, *field-split-simps*]:
$0 < a * b \longleftrightarrow 0 < a \wedge 0 < b \vee a < 0 \wedge b < 0$
⟨*proof*⟩

**lemma** *zero-le-mult-iff* [*algebra-split-simps*, *field-split-simps*]:
$0 \leq a * b \longleftrightarrow 0 \leq a \wedge 0 \leq b \vee a \leq 0 \wedge b \leq 0$
⟨*proof*⟩

**lemma** *mult-less-0-iff* [*algebra-split-simps*, *field-split-simps*]:
$a * b < 0 \longleftrightarrow 0 < a \wedge b < 0 \vee a < 0 \wedge 0 < b$
⟨*proof*⟩

**lemma** *mult-le-0-iff* [*algebra-split-simps*, *field-split-simps*]:
$a * b \leq 0 \longleftrightarrow 0 \leq a \wedge b \leq 0 \vee a \leq 0 \wedge 0 \leq b$
⟨*proof*⟩

Cancellation laws for $c * a < c * b$ and $a * c < b * c$, also with the relations $\leq$ and equality.

These "disjunction" versions produce two cases when the comparison is an assumption, but effectively four when the comparison is a goal.

**lemma** *mult-less-cancel-right-disj*: $a * c < b * c \longleftrightarrow 0 < c \wedge a < b \vee c < 0 \wedge b < a$
$\langle proof \rangle$

**lemma** *mult-less-cancel-left-disj*: $c * a < c * b \longleftrightarrow 0 < c \wedge a < b \vee c < 0 \wedge b < a$
$\langle proof \rangle$

The "conjunction of implication" lemmas produce two cases when the comparison is a goal, but give four when the comparison is an assumption.

**lemma** *mult-less-cancel-right*: $a * c < b * c \longleftrightarrow (0 \leq c \longrightarrow a < b) \wedge (c \leq 0 \longrightarrow b < a)$
$\langle proof \rangle$

**lemma** *mult-less-cancel-left*: $c * a < c * b \longleftrightarrow (0 \leq c \longrightarrow a < b) \wedge (c \leq 0 \longrightarrow b < a)$
$\langle proof \rangle$

**lemma** *mult-le-cancel-right*: $a * c \leq b * c \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
$\langle proof \rangle$

**lemma** *mult-le-cancel-left*: $c * a \leq c * b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
$\langle proof \rangle$

**lemma** *mult-le-cancel-left-pos*: $0 < c \Longrightarrow c * a \leq c * b \longleftrightarrow a \leq b$
$\langle proof \rangle$

**lemma** *mult-le-cancel-left-neg*: $c < 0 \Longrightarrow c * a \leq c * b \longleftrightarrow b \leq a$
$\langle proof \rangle$

**lemma** *mult-less-cancel-left-pos*: $0 < c \Longrightarrow c * a < c * b \longleftrightarrow a < b$
$\langle proof \rangle$

**lemma** *mult-less-cancel-left-neg*: $c < 0 \Longrightarrow c * a < c * b \longleftrightarrow b < a$
$\langle proof \rangle$

**lemma** *mult-le-cancel-right-pos*: $0 < c \Longrightarrow a * c \leq b * c \longleftrightarrow a \leq b$
$\langle proof \rangle$

**lemma** *mult-le-cancel-right-neg*: $c < 0 \Longrightarrow a * c \leq b * c \longleftrightarrow b \leq a$
$\langle proof \rangle$

**lemma** *mult-less-cancel-right-pos*: $0 < c \Longrightarrow a * c < b * c \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *mult-less-cancel-right-neg*: $c < 0 \Longrightarrow a * c < b * c \longleftrightarrow b < a$
⟨*proof*⟩

**end**

**lemmas** *mult-sign-intros* =
*mult-nonneg-nonneg mult-nonneg-nonpos*
*mult-nonpos-nonneg mult-nonpos-nonpos*
*mult-pos-pos mult-pos-neg*
*mult-neg-pos mult-neg-neg*

**class** *ordered-comm-ring* = *comm-ring* + *ordered-comm-semiring*
**begin**

**subclass** *ordered-ring* ⟨*proof*⟩
**subclass** *ordered-cancel-comm-semiring* ⟨*proof*⟩

**end**

**class** *linordered-nonzero-semiring* = *ordered-comm-semiring* + *monoid-mult* + *linorder* + *zero-less-one* +
**assumes** *add-mono1*: $a < b \Longrightarrow a + 1 < b + 1$
**begin**

**subclass** *zero-neq-one* ⟨*proof*⟩

**subclass** *comm-semiring-1*
⟨*proof*⟩

**subclass** *ordered-semiring-1* ⟨*proof*⟩

**lemma** *not-one-le-zero* [*simp*]: $\neg\ 1 \leq 0$
⟨*proof*⟩

**lemma** *not-one-less-zero* [*simp*]: $\neg\ 1 < 0$
⟨*proof*⟩

**lemma** *of-bool-less-eq-iff* [*simp*]:
‹$\mathit{of\text{-}bool}\ P \leq \mathit{of\text{-}bool}\ Q \longleftrightarrow (P \longrightarrow Q)$›
⟨*proof*⟩

**lemma** *of-bool-less-iff* [*simp*]:
‹$\mathit{of\text{-}bool}\ P < \mathit{of\text{-}bool}\ Q \longleftrightarrow \neg\ P \wedge Q$›
⟨*proof*⟩

**lemma** *mult-left-le*: $c \leq 1 \Longrightarrow 0 \leq a \Longrightarrow a * c \leq a$

⟨*proof*⟩

**lemma** *mult-le-one*: $a \le 1 \Longrightarrow 0 \le b \Longrightarrow b \le 1 \Longrightarrow a * b \le 1$
⟨*proof*⟩

**lemma** *zero-less-two*: $0 < 1 + 1$
⟨*proof*⟩

**end**

**class** *linordered-semidom* = *semidom* + *linordered-comm-semiring-strict* + *zero-less-one* +
**assumes** *le-add-diff-inverse2* [*simp*]: $b \le a \Longrightarrow a - b + b = a$
**begin**

**subclass** *linordered-nonzero-semiring*
⟨*proof*⟩

**lemma** *zero-less-eq-of-bool* [*simp*]:
‹$0 \le$ *of-bool* $P$›
⟨*proof*⟩

**lemma** *zero-less-of-bool-iff* [*simp*]:
‹$0 <$ *of-bool* $P \longleftrightarrow P$›
⟨*proof*⟩

**lemma** *of-bool-less-eq-one* [*simp*]:
‹*of-bool* $P \le 1$›
⟨*proof*⟩

**lemma** *of-bool-less-one-iff* [*simp*]:
‹*of-bool* $P < 1 \longleftrightarrow \neg P$›
⟨*proof*⟩

**lemma** *of-bool-or-iff* [*simp*]:
‹*of-bool* $(P \vee Q) =$ *max* (*of-bool* $P$) (*of-bool* $Q$)›
⟨*proof*⟩

Addition is the inverse of subtraction.

**lemma** *le-add-diff-inverse* [*simp*]: $b \le a \Longrightarrow b + (a - b) = a$
⟨*proof*⟩

**lemma** *add-diff-inverse*: $\neg a < b \Longrightarrow b + (a - b) = a$
⟨*proof*⟩

**lemma** *add-le-imp-le-diff*:
**assumes** $i + k \le n$ **shows** $i \le n - k$
⟨*proof*⟩

**lemma** *add-le-add-imp-diff-le*:
**assumes** *1*: $i + k \leq n$
**and** *2*: $n \leq j + k$
**shows** $i + k \leq n \Longrightarrow n \leq j + k \Longrightarrow n - k \leq j$
⟨*proof*⟩

**lemma** *less-1-mult*: $1 < m \Longrightarrow 1 < n \Longrightarrow 1 < m * n$
⟨*proof*⟩

**lemma** *less-1-mult'*:
**shows** $1 < a \Longrightarrow 1 \leq b \Longrightarrow 1 < a * b$
⟨*proof*⟩

**end**

**class** *linordered-idom* = *comm-ring-1* + *linordered-comm-semiring-strict* +
*ordered-ab-group-add* + *abs-if* + *sgn* +
**assumes** *sgn-if*: *sgn x* = (*if x* = *0 then 0 else if 0* < *x then 1 else* − *1*)
**begin**

**subclass** *linordered-ring-strict* ⟨*proof*⟩

**subclass** *linordered-semiring-1-strict*
⟨*proof*⟩

**subclass** *ordered-comm-ring* ⟨*proof*⟩
**subclass** *idom* ⟨*proof*⟩

**subclass** *linordered-semidom*
⟨*proof*⟩

**subclass** *idom-abs-sgn*
⟨*proof*⟩

**lemma** *abs-bool-eq* [*simp*]:
‹|*of-bool P*| = *of-bool P*›
⟨*proof*⟩

**lemma** *linorder-neqE-linordered-idom*:
**assumes** $x \neq y$
**obtains** $x < y \mid y < x$
⟨*proof*⟩

These cancellation simp rules also produce two cases when the comparison is a goal.

**lemma** *mult-le-cancel-right1*: $c \leq b * c \longleftrightarrow (0 < c \longrightarrow 1 \leq b) \land (c < 0 \longrightarrow b \leq 1)$
⟨*proof*⟩

**lemma** *mult-le-cancel-right2*: $a * c \leq c \longleftrightarrow (0 < c \longrightarrow a \leq 1) \wedge (c < 0 \longrightarrow 1 \leq a)$
⟨*proof*⟩

**lemma** *mult-le-cancel-left1*: $c \leq c * b \longleftrightarrow (0 < c \longrightarrow 1 \leq b) \wedge (c < 0 \longrightarrow b \leq 1)$
⟨*proof*⟩

**lemma** *mult-le-cancel-left2*: $c * a \leq c \longleftrightarrow (0 < c \longrightarrow a \leq 1) \wedge (c < 0 \longrightarrow 1 \leq a)$
⟨*proof*⟩

**lemma** *mult-less-cancel-right1*: $c < b * c \longleftrightarrow (0 \leq c \longrightarrow 1 < b) \wedge (c \leq 0 \longrightarrow b < 1)$
⟨*proof*⟩

**lemma** *mult-less-cancel-right2*: $a * c < c \longleftrightarrow (0 \leq c \longrightarrow a < 1) \wedge (c \leq 0 \longrightarrow 1 < a)$
⟨*proof*⟩

**lemma** *mult-less-cancel-left1*: $c < c * b \longleftrightarrow (0 \leq c \longrightarrow 1 < b) \wedge (c \leq 0 \longrightarrow b < 1)$
⟨*proof*⟩

**lemma** *mult-less-cancel-left2*: $c * a < c \longleftrightarrow (0 \leq c \longrightarrow a < 1) \wedge (c \leq 0 \longrightarrow 1 < a)$
⟨*proof*⟩

**lemma** *sgn-0-0*: $sgn\ a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *sgn-1-pos*: $sgn\ a = 1 \longleftrightarrow a > 0$
⟨*proof*⟩

**lemma** *sgn-1-neg*: $sgn\ a = -\ 1 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *sgn-pos* [*simp*]: $0 < a \Longrightarrow sgn\ a = 1$
⟨*proof*⟩

**lemma** *sgn-neg* [*simp*]: $a < 0 \Longrightarrow sgn\ a = -\ 1$
⟨*proof*⟩

**lemma** *abs-sgn*: $|k| = k * sgn\ k$
⟨*proof*⟩

**lemma** *sgn-greater* [*simp*]: $0 < sgn\ a \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *sgn-less* [*simp*]: $sgn\ a < 0 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *abs-sgn-eq-1* [*simp*]:
$a \neq 0 \Longrightarrow |sgn\ a| = 1$
⟨*proof*⟩

**lemma** *abs-sgn-eq*: $|sgn\ a| = (if\ a = 0\ then\ 0\ else\ 1)$
⟨*proof*⟩

**lemma** *sgn-mult-self-eq* [*simp*]:
$sgn\ a * sgn\ a = of\text{-}bool\ (a \neq 0)$
⟨*proof*⟩

**lemma** *left-sgn-mult-self-eq* [*simp*]:
‹$sgn\ a * (sgn\ a * b) = of\text{-}bool\ (a \neq 0) * b$›
⟨*proof*⟩

**lemma** *abs-mult-self-eq* [*simp*]:
$|a| * |a| = a * a$
⟨*proof*⟩

**lemma** *same-sgn-sgn-add*:
$sgn\ (a + b) = sgn\ a$ **if** $sgn\ b = sgn\ a$
⟨*proof*⟩

**lemma** *same-sgn-abs-add*:
$|a + b| = |a| + |b|$ **if** $sgn\ b = sgn\ a$
⟨*proof*⟩

**lemma** *sgn-not-eq-imp*:
$sgn\ a = -\ sgn\ b$ **if** $sgn\ b \neq sgn\ a$ **and** $sgn\ a \neq 0$ **and** $sgn\ b \neq 0$
⟨*proof*⟩

**lemma** *abs-dvd-iff* [*simp*]: $|m|\ dvd\ k \longleftrightarrow m\ dvd\ k$
⟨*proof*⟩

**lemma** *dvd-abs-iff* [*simp*]: $m\ dvd\ |k| \longleftrightarrow m\ dvd\ k$
⟨*proof*⟩

**lemma** *dvd-if-abs-eq*: $|l| = |k| \Longrightarrow l\ dvd\ k$
⟨*proof*⟩

The following lemmas can be proven in more general structures, but are dangerous as simp rules in absence of $(-\ ?a = ?a) = (?a = 0)$, $(-\ ?a < ?a) = (0 < ?a)$, $(-\ ?a \leq ?a) = (0 \leq ?a)$.

**lemma** *equation-minus-iff-1* [*simp*, *no-atp*]: $1 = -\ a \longleftrightarrow a = -\ 1$
⟨*proof*⟩

**lemma** *minus-equation-iff-1* [*simp*, *no-atp*]: $- a = 1 \longleftrightarrow a = - 1$
⟨*proof*⟩

**lemma** *le-minus-iff-1* [*simp*, *no-atp*]: $1 \le - b \longleftrightarrow b \le - 1$
⟨*proof*⟩

**lemma** *minus-le-iff-1* [*simp*, *no-atp*]: $- a \le 1 \longleftrightarrow - 1 \le a$
⟨*proof*⟩

**lemma** *less-minus-iff-1* [*simp*, *no-atp*]: $1 < - b \longleftrightarrow b < - 1$
⟨*proof*⟩

**lemma** *minus-less-iff-1* [*simp*, *no-atp*]: $- a < 1 \longleftrightarrow - 1 < a$
⟨*proof*⟩

**lemma** *add-less-zeroD*:
**shows** $x+y < 0 \Longrightarrow x<0 \vee y<0$
⟨*proof*⟩

Is this really better than just rewriting with *abs-if*?

**lemma** *abs-split* [*no-atp*]: ‹$P\ |a| \longleftrightarrow (0 \le a \longrightarrow P\ a) \wedge (a < 0 \longrightarrow P\ (- a))$›
⟨*proof*⟩

**end**

**class** *discrete-linordered-semidom* = *linordered-semidom* +
**assumes** *less-iff-succ-less-eq*: ‹$a < b \longleftrightarrow a + 1 \le b$›
**begin**

**lemma** *less-eq-iff-succ-less*:
‹$a \le b \longleftrightarrow a < b + 1$›
⟨*proof*⟩

**end**

Reasoning about inequalities with division

**context** *linordered-semidom*
**begin**

**lemma** *less-add-one*: $a < a + 1$
⟨*proof*⟩

**end**

**context** *linordered-idom*
**begin**

**lemma** *mult-right-le-one-le*: $0 \le x \Longrightarrow 0 \le y \Longrightarrow y \le 1 \Longrightarrow x * y \le x$
⟨*proof*⟩

**lemma** *mult-left-le-one-le*: $0 \leq x \Longrightarrow 0 \leq y \Longrightarrow y \leq 1 \Longrightarrow y * x \leq x$
⟨*proof*⟩

**end**

Absolute Value

**context** *linordered-idom*
**begin**

**lemma** *mult-sgn-abs*: $sgn\ x * |x| = x$
⟨*proof*⟩

**lemma** *abs-one*: $|1| = 1$
⟨*proof*⟩

**end**

**class** *ordered-ring-abs* = *ordered-ring* + *ordered-ab-group-add-abs* +
**assumes** *abs-eq-mult*:
$(0 \leq a \vee a \leq 0) \wedge (0 \leq b \vee b \leq 0) \Longrightarrow |a * b| = |a| * |b|$

**context** *linordered-idom*
**begin**

**subclass** *ordered-ring-abs*
⟨*proof*⟩

**lemma** *abs-mult-self*: $|a| * |a| = a * a$
⟨*proof*⟩

**lemma** *abs-mult-less*:
**assumes** *ac*: $|a| < c$
**and** *bd*: $|b| < d$
**shows** $|a| * |b| < c * d$
⟨*proof*⟩

**lemma** *abs-less-iff*: $|a| < b \longleftrightarrow a < b \wedge - a < b$
⟨*proof*⟩

**lemma** *abs-mult-pos*: $0 \leq x \Longrightarrow |y| * x = |y * x|$
⟨*proof*⟩

**lemma** *abs-mult-pos'*: $0 \leq x \Longrightarrow x * |y| = |x * y|$
⟨*proof*⟩

**lemma** *abs-diff-less-iff*: $|x - a| < r \longleftrightarrow a - r < x \wedge x < a + r$
⟨*proof*⟩

**lemma** *abs-diff-le-iff*: $|x - a| \leq r \longleftrightarrow a - r \leq x \wedge x \leq a + r$
⟨*proof*⟩

**lemma** *abs-add-one-gt-zero*: $0 < 1 + |x|$
⟨*proof*⟩

**end**

### 16.8 Dioids

Dioids are the alternative extensions of semirings, a semiring can either be a ring or a dioid but never both.

**class** *dioid* = *semiring-1* + *canonically-ordered-monoid-add*
**begin**

**subclass** *ordered-semiring*
⟨*proof*⟩

**end**

**hide-fact** (**open**) *comm-mult-left-mono comm-mult-strict-left-mono distrib*

**code-identifier**
**code-module** *Rings* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

## 17 Natural numbers

**theory** *Nat*
**imports** *Inductive Typedef Fun Rings*
**begin**

### 17.1 Type *ind*

**typedecl** *ind*

**axiomatization** *Zero-Rep* :: *ind* **and** *Suc-Rep* :: *ind* ⇒ *ind*
— The axiom of infinity in 2 parts:
**where** *Suc-Rep-inject*: *Suc-Rep* $x$ = *Suc-Rep* $y \Longrightarrow x = y$
**and** *Suc-Rep-not-Zero-Rep*: *Suc-Rep* $x \neq$ *Zero-Rep*

### 17.2 Type nat

Type definition

**inductive** *Nat* :: *ind* ⇒ *bool*

**where**
*Zero-RepI*: *Nat Zero-Rep*
| *Suc-RepI*: *Nat* $i \Longrightarrow$ *Nat* (*Suc-Rep* $i$)

**typedef** *nat* = {*n*. *Nat n*}
**morphisms** *Rep-Nat Abs-Nat*
⟨*proof*⟩

**lemma** *Nat-Rep-Nat*: *Nat* (*Rep-Nat n*)
⟨*proof*⟩

**lemma** *Nat-Abs-Nat-inverse*: *Nat n* $\Longrightarrow$ *Rep-Nat* (*Abs-Nat n*) = *n*
⟨*proof*⟩

**lemma** *Nat-Abs-Nat-inject*: *Nat n* $\Longrightarrow$ *Nat m* $\Longrightarrow$ *Abs-Nat n* = *Abs-Nat m* $\longleftrightarrow$ *n* = *m*
⟨*proof*⟩

**instantiation** *nat* :: *zero*
**begin**

**definition** *Zero-nat-def*: *0* = *Abs-Nat Zero-Rep*

**instance** ⟨*proof*⟩

**end**

**definition** *Suc* :: *nat* $\Rightarrow$ *nat*
**where** *Suc n* = *Abs-Nat* (*Suc-Rep* (*Rep-Nat n*))

**lemma** *Suc-not-Zero*: *Suc m* $\neq$ *0*
⟨*proof*⟩

**lemma** *Zero-not-Suc*: *0* $\neq$ *Suc m*
⟨*proof*⟩

**lemma** *Suc-Rep-inject'*: *Suc-Rep x* = *Suc-Rep y* $\longleftrightarrow$ *x* = *y*
⟨*proof*⟩

**lemma** *nat-induct0*:
**assumes** *P 0* **and** $\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n)$
**shows** *P n*
⟨*proof*⟩

**free-constructors** *case-nat* **for** *0* :: *nat* | *Suc pred*
**where** *pred* (*0* :: *nat*) = (*0* :: *nat*)
⟨*proof*⟩
⟨*ML*⟩

**old-rep-datatype** *0* :: *nat Suc*
⟨*proof*⟩

⟨*ML*⟩

**declare** *old.nat.inject*[*iff del*]
**and** *old.nat.distinct*(*1*)[*simp del*, *induct-simp del*]

**lemmas** *induct* = *old.nat.induct*
**lemmas** *inducts* = *old.nat.inducts*
**lemmas** *rec* = *old.nat.rec*
**lemmas** *simps* = *nat.inject nat.distinct nat.case nat.rec*

⟨*ML*⟩

**abbreviation** *rec-nat* :: $'a \Rightarrow (nat \Rightarrow 'a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
**where** *rec-nat* ≡ *old.rec-nat*

**declare** *nat.sel*[*code del*]

**hide-const** (**open**) *Nat.pred* — hide everything related to the selector

**lemma** *nat-exhaust* [*case-names 0 Suc*, *cases type*: *nat*]:
$(y = 0 \Longrightarrow P) \Longrightarrow (\bigwedge nat.\ y = Suc\ nat \Longrightarrow P) \Longrightarrow P$
— for backward compatibility – names of variables differ
⟨*proof*⟩

**lemma** *nat-induct* [*case-names 0 Suc*, *induct type*: *nat*]:
**fixes** *n*
**assumes** *P 0* **and** $\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n)$
**shows** *P n*
— for backward compatibility – names of variables differ
⟨*proof*⟩

**hide-fact**
*nat-exhaust*
*nat-induct0*

⟨*ML*⟩

Injectiveness and distinctness lemmas

**lemma** *inj-Suc* [*simp*]:
*inj-on Suc N*
⟨*proof*⟩

**lemma** *bij-betw-Suc* [*simp*]:
$bij\text{-}betw\ Suc\ M\ N \longleftrightarrow Suc\ `\ M = N$
⟨*proof*⟩

**lemma** *Suc-neq-Zero*: $Suc\ m = 0 \Longrightarrow R$
⟨*proof*⟩

**lemma** *Zero-neq-Suc*: $0 = Suc\ m \Longrightarrow R$
⟨*proof*⟩

**lemma** *Suc-inject*: $Suc\ x = Suc\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *n-not-Suc-n*: $n \neq Suc\ n$
⟨*proof*⟩

**lemma** *Suc-n-not-n*: $Suc\ n \neq n$
⟨*proof*⟩

A special form of induction for reasoning about $m < n$ and $m - n$.

**lemma** *diff-induct*:
**assumes** $\bigwedge x.\ P\ x\ 0$
**and** $\bigwedge y.\ P\ 0\ (Suc\ y)$
**and** $\bigwedge x\ y.\ P\ x\ y \Longrightarrow P\ (Suc\ x)\ (Suc\ y)$
**shows** $P\ m\ n$
⟨*proof*⟩

## 17.3 Arithmetic operators

**instantiation** *nat* :: *comm-monoid-diff*
**begin**

**primrec** *plus-nat*
**where**
*add-0* [*code*]: $0 + n = (n::nat)$
| *add-Suc*: $Suc\ m + n = Suc\ (m + n)$

**lemma** *add-0-right* [*simp*]: $m + 0 = m$
**for** $m$ :: *nat*
⟨*proof*⟩

**lemma** *add-Suc-right* [*simp*]: $m + Suc\ n = Suc\ (m + n)$
⟨*proof*⟩

**lemma** *add-Suc-shift* [*code*]: $Suc\ m + n = m + Suc\ n$
⟨*proof*⟩

**primrec** *minus-nat*
**where**
*diff-0* [*code*]: $m - 0 = (m::nat)$
| *diff-Suc*: $m - Suc\ n = (case\ m - n\ of\ 0 \Rightarrow 0 \mid Suc\ k \Rightarrow k)$

**declare** *diff-Suc* [*simp del*]

**lemma** *diff-0-eq-0* [*simp*, *code*]: $0 - n = 0$
**for** $n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-Suc-Suc* [*simp*, *code*]: $Suc\ m - Suc\ n = m - n$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**hide-fact** (**open**) *add-0 add-0-right diff-0*

**instantiation** *nat* :: *comm-semiring-1-cancel*
**begin**

**definition** *One-nat-def* [*simp*]: $1 = Suc\ 0$

**primrec** *times-nat*
**where**
*mult-0*: $0 * n = (0::nat)$
| *mult-Suc*: $Suc\ m * n = n + (m * n)$

**lemma** *mult-0-right* [*simp*]: $m * 0 = 0$
**for** $m$ :: *nat*
⟨*proof*⟩

**lemma** *mult-Suc-right* [*simp*]: $m * Suc\ n = m + (m * n)$
⟨*proof*⟩

**lemma** *add-mult-distrib*: $(m + n) * k = (m * k) + (n * k)$
**for** $m\ n\ k$ :: *nat*
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

### 17.3.1 Addition

Reasoning about $m + 0 = 0$, etc.

**lemma** *add-is-0* [*iff*]: $m + n = 0 \longleftrightarrow m = 0 \wedge n = 0$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *add-is-1*: $m + n = Suc\ 0 \longleftrightarrow m = Suc\ 0 \wedge n = 0 \vee m = 0 \wedge n = Suc\ 0$

⟨*proof*⟩

**lemma** *one-is-add*: $Suc\ 0 = m + n \longleftrightarrow m = Suc\ 0 \land n = 0 \lor m = 0 \land n = Suc\ 0$
⟨*proof*⟩

**lemma** *add-eq-self-zero*: $m + n = m \Longrightarrow n = 0$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *plus-1-eq-Suc*:
$plus\ 1 = Suc$
⟨*proof*⟩

**lemma** *Suc-eq-plus1*: $Suc\ n = n + 1$
⟨*proof*⟩

**lemma** *Suc-eq-plus1-left*: $Suc\ n = 1 + n$
⟨*proof*⟩

### 17.3.2 Difference

**lemma** *Suc-diff-diff* [*simp*]: $(Suc\ m - n) - Suc\ k = m - n - k$
⟨*proof*⟩

**lemma** *diff-Suc-1*: $Suc\ n - 1 = n$
⟨*proof*⟩

**lemma** *diff-Suc-1'* [*simp*]: $Suc\ n - Suc\ 0 = n$
⟨*proof*⟩

### 17.3.3 Multiplication

**lemma** *mult-is-0* [*simp*]: $m * n = 0 \longleftrightarrow m = 0 \lor n = 0$ **for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-eq-1-iff* [*simp*]: $m * n = Suc\ 0 \longleftrightarrow m = Suc\ 0 \land n = Suc\ 0$
⟨*proof*⟩

**lemma** *one-eq-mult-iff* [*simp*]: $Suc\ 0 = m * n \longleftrightarrow m = Suc\ 0 \land n = Suc\ 0$
⟨*proof*⟩

**lemma** *nat-mult-eq-1-iff* [*simp*]: $m * n = 1 \longleftrightarrow m = 1 \land n = 1$
**and** *nat-1-eq-mult-iff* [*simp*]: $1 = m * n \longleftrightarrow m = 1 \land n = 1$ **for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-cancel1* [*simp*]: $k * m = k * n \longleftrightarrow m = n \lor k = 0$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-cancel2* [*simp*]: $m * k = n * k \longleftrightarrow m = n \lor k = 0$
**for** $k\ m\ n :: nat$
⟨*proof*⟩

**lemma** *Suc-mult-cancel1*: $Suc\ k * m = Suc\ k * n \longleftrightarrow m = n$
⟨*proof*⟩

## 17.4 Orders on *nat*

### 17.4.1 Operation definition

**instantiation** *nat* :: *linorder*
**begin**

**primrec** *less-eq-nat*
**where**
$(0::nat) \le n \longleftrightarrow True$
$|\ Suc\ m \le n \longleftrightarrow (case\ n\ of\ 0 \Rightarrow False\ |\ Suc\ n \Rightarrow m \le n)$

**declare** *less-eq-nat.simps* [*simp del*]

**lemma** *le0* [*iff*]: $0 \le n$ **for**
$n :: nat$
⟨*proof*⟩

**lemma** [*code*]: $0 \le n \longleftrightarrow True$
**for** $n :: nat$
⟨*proof*⟩

**definition** *less-nat*
**where** *less-eq-Suc-le*: $n < m \longleftrightarrow Suc\ n \le m$

**lemma** *Suc-le-mono* [*iff*]: $Suc\ n \le Suc\ m \longleftrightarrow n \le m$
⟨*proof*⟩

**lemma** *Suc-le-eq* [*code*]: $Suc\ m \le n \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** *le-0-eq* [*iff*]: $n \le 0 \longleftrightarrow n = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *not-less0* [*iff*]: $\neg\ n < 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *less-nat-zero-code* [*code*]: $n < 0 \longleftrightarrow False$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *Suc-less-eq* [*iff*]: $Suc\ m < Suc\ n \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** *less-Suc-eq-le* [*code*]: $m < Suc\ n \longleftrightarrow m \leq n$
⟨*proof*⟩

**lemma** *Suc-less-eq2*: $Suc\ n < m \longleftrightarrow (\exists\, m'.\ m = Suc\ m' \wedge n < m')$
⟨*proof*⟩

**lemma** *le-SucI*: $m \leq n \Longrightarrow m \leq Suc\ n$
⟨*proof*⟩

**lemma** *Suc-leD*: $Suc\ m \leq n \Longrightarrow m \leq n$
⟨*proof*⟩

**lemma** *less-SucI*: $m < n \Longrightarrow m < Suc\ n$
⟨*proof*⟩

**lemma** *Suc-lessD*: $Suc\ m < n \Longrightarrow m < n$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**instantiation** *nat* :: *order-bot*
**begin**

**definition** *bot-nat* :: *nat*
**where** *bot-nat* = *0*

**instance**
⟨*proof*⟩

**end**

**instance** *nat* :: *no-top*
⟨*proof*⟩

### 17.4.2 Introduction properties

**lemma** *lessI* [*iff*]: $n < Suc\ n$
⟨*proof*⟩

**lemma** *zero-less-Suc* [*iff*]: $0 < Suc\ n$
⟨*proof*⟩

### 17.4.3 Elimination properties

**lemma** *less-not-refl*: $\neg\ n < n$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *less-not-refl2*: $n < m \Longrightarrow m \neq n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *less-not-refl3*: $s < t \Longrightarrow s \neq t$
**for** $s\ t :: nat$
⟨*proof*⟩

**lemma** *less-irrefl-nat*: $n < n \Longrightarrow R$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *less-zeroE*: $n < 0 \Longrightarrow R$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *less-Suc-eq*: $m < Suc\ n \longleftrightarrow m < n \vee m = n$
⟨*proof*⟩

**lemma** *less-Suc0* [*iff*]: $(n < Suc\ 0) = (n = 0)$
⟨*proof*⟩

**lemma** *less-one* [*iff*]: $n < 1 \longleftrightarrow n = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *Suc-mono*: $m < n \Longrightarrow Suc\ m < Suc\ n$
⟨*proof*⟩

"Less than" is antisymmetric, sort of.

**lemma** *less-antisym*: $\neg\ n < m \Longrightarrow n < Suc\ m \Longrightarrow m = n$
⟨*proof*⟩

**lemma** *nat-neq-iff*: $m \neq n \longleftrightarrow m < n \vee n < m$
**for** $m\ n :: nat$
⟨*proof*⟩

### 17.4.4 Inductive (?) properties

**lemma** *Suc-lessI*: $m < n \Longrightarrow Suc\ m \neq n \Longrightarrow Suc\ m < n$
⟨*proof*⟩

**lemma** *lessE*:
**assumes** *major*: $i < k$

**and** *1*: $k = Suc\ i \Longrightarrow P$
**and** *2*: $\bigwedge j.\ i < j \Longrightarrow k = Suc\ j \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *less-SucE*:
**assumes** *major*: $m < Suc\ n$
**and** *less*: $m < n \Longrightarrow P$
**and** *eq*: $m = n \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *Suc-lessE*:
**assumes** *major*: $Suc\ i < k$
**and** *minor*: $\bigwedge j.\ i < j \Longrightarrow k = Suc\ j \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *Suc-less-SucD*: $Suc\ m < Suc\ n \Longrightarrow m < n$
⟨*proof*⟩

**lemma** *less-trans-Suc*:
**assumes** *le*: $i < j$
**shows** $j < k \Longrightarrow Suc\ i < k$
⟨*proof*⟩

Can be used with *less-Suc-eq* to get $n = m \lor n < m$.

**lemma** *not-less-eq*: $\neg\ m < n \longleftrightarrow n < Suc\ m$
⟨*proof*⟩

**lemma** *not-less-eq-eq*: $\neg\ m \le n \longleftrightarrow Suc\ n \le m$
⟨*proof*⟩

Properties of "less than or equal".

**lemma** *le-imp-less-Suc*: $m \le n \Longrightarrow m < Suc\ n$
⟨*proof*⟩

**lemma** *Suc-n-not-le-n*: $\neg\ Suc\ n \le n$
⟨*proof*⟩

**lemma** *le-Suc-eq*: $m \le Suc\ n \longleftrightarrow m \le n \lor m = Suc\ n$
⟨*proof*⟩

**lemma** *le-SucE*: $m \le Suc\ n \Longrightarrow (m \le n \Longrightarrow R) \Longrightarrow (m = Suc\ n \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *Suc-leI*: $m < n \Longrightarrow Suc\ m \le n$
⟨*proof*⟩

Stronger version of *Suc-leD*.

**lemma** *Suc-le-lessD*: $Suc\ m \leq n \Longrightarrow m < n$
⟨*proof*⟩

**lemma** *less-imp-le-nat*: $m < n \Longrightarrow m \leq n$ **for** $m\ n :: nat$
⟨*proof*⟩

For instance, $(Suc\ m < Suc\ n) = (Suc\ m \leq n) = (m < n)$

**lemmas** *le-simps* = *less-imp-le-nat less-Suc-eq-le Suc-le-eq*

Equivalence of $m \leq n$ and $m < n \vee m = n$

**lemma** *less-or-eq-imp-le*: $m < n \vee m = n \Longrightarrow m \leq n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *le-eq-less-or-eq*: $m \leq n \longleftrightarrow m < n \vee m = n$
**for** $m\ n :: nat$
⟨*proof*⟩

Useful with *blast*.

**lemma** *eq-imp-le*: $m = n \Longrightarrow m \leq n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *le-refl*: $n \leq n$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *le-trans*: $i \leq j \Longrightarrow j \leq k \Longrightarrow i \leq k$
**for** $i\ j\ k :: nat$
⟨*proof*⟩

**lemma** *le-antisym*: $m \leq n \Longrightarrow n \leq m \Longrightarrow m = n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *nat-less-le*: $m < n \longleftrightarrow m \leq n \wedge m \neq n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *le-neq-implies-less*: $m \leq n \Longrightarrow m \neq n \Longrightarrow m < n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *nat-le-linear*: $m \leq n \vee n \leq m$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemmas** *linorder-neqE-nat* = *linorder-neqE* [**where** $'a = nat$]

**lemma** *le-less-Suc-eq*: $m \le n \Longrightarrow n < Suc\ m \longleftrightarrow n = m$
⟨*proof*⟩

**lemma** *not-less-less-Suc-eq*: $\neg\ n < m \Longrightarrow n < Suc\ m \longleftrightarrow n = m$
⟨*proof*⟩

**lemmas** *not-less-simps* = *not-less-less-Suc-eq le-less-Suc-eq*

**lemma** *not0-implies-Suc*: $n \neq 0 \Longrightarrow \exists\, m.\ n = Suc\ m$
⟨*proof*⟩

**lemma** *gr0-implies-Suc*: $n > 0 \Longrightarrow \exists\, m.\ n = Suc\ m$
⟨*proof*⟩

**lemma** *gr-implies-not0*: $m < n \Longrightarrow n \neq 0$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *neq0-conv*[*iff*]: $n \neq 0 \longleftrightarrow 0 < n$
**for** $n$ :: *nat*
⟨*proof*⟩

This theorem is useful with *blast*

**lemma** *gr0I*: $(n = 0 \Longrightarrow False) \Longrightarrow 0 < n$
**for** $n$ :: *nat*
⟨*proof*⟩

**lemma** *gr0-conv-Suc*: $0 < n \longleftrightarrow (\exists\, m.\ n = Suc\ m)$
⟨*proof*⟩

**lemma** *not-gr0* [*iff*]: $\neg\ 0 < n \longleftrightarrow n = 0$
**for** $n$ :: *nat*
⟨*proof*⟩

**lemma** *Suc-le-D*: $Suc\ n \le m' \Longrightarrow \exists\, m.\ m' = Suc\ m$
⟨*proof*⟩

Useful in certain inductive arguments

**lemma** *less-Suc-eq-0-disj*: $m < Suc\ n \longleftrightarrow m = 0 \lor (\exists\, j.\ m = Suc\ j \land j < n)$
⟨*proof*⟩

**lemma** *All-less-Suc*: $(\forall\, i < Suc\ n.\ P\ i) = (P\ n \land (\forall\, i < n.\ P\ i))$
⟨*proof*⟩

**lemma** *All-less-Suc2*: $(\forall\, i < Suc\ n.\ P\ i) = (P\ 0 \land (\forall\, i < n.\ P(Suc\ i)))$
⟨*proof*⟩

**lemma** *Ex-less-Suc*: $(\exists\, i < Suc\ n.\ P\ i) = (P\ n \lor (\exists\, i < n.\ P\ i))$
⟨*proof*⟩

**lemma** *Ex-less-Suc2*: $(\exists\, i < Suc\ n.\ P\ i) = (P\ 0 \lor (\exists\, i < n.\ P(Suc\ i)))$
⟨*proof*⟩

*mono* (non-strict) doesn't imply increasing, as the function could be constant

**lemma** *strict-mono-imp-increasing*:
**fixes** $n::nat$
**assumes** *strict-mono* $f$ **shows** $f\ n \geq n$
⟨*proof*⟩

### 17.4.5 Monotonicity of Addition

**lemma** *Suc-pred* [*simp*]: $n > 0 \Longrightarrow Suc\ (n - Suc\ 0) = n$
⟨*proof*⟩

**lemma** *Suc-diff-1* [*simp*]: $0 < n \Longrightarrow Suc\ (n - 1) = n$
⟨*proof*⟩

**lemma** *nat-add-left-cancel-le* [*simp*]: $k + m \leq k + n \longleftrightarrow m \leq n$
**for** $k\ m\ n :: nat$
⟨*proof*⟩

**lemma** *nat-add-left-cancel-less* [*simp*]: $k + m < k + n \longleftrightarrow m < n$
**for** $k\ m\ n :: nat$
⟨*proof*⟩

**lemma** *add-gr-0* [*iff*]: $m + n > 0 \longleftrightarrow m > 0 \lor n > 0$
**for** $m\ n :: nat$
⟨*proof*⟩

strict, in 1st argument

**lemma** *add-less-mono1*: $i < j \Longrightarrow i + k < j + k$
**for** $i\ j\ k :: nat$
⟨*proof*⟩

strict, in both arguments

**lemma** *add-less-mono*:
**fixes** $i\ j\ k\ l :: nat$
**assumes** $i < j\ k < l$ **shows** $i + k < j + l$
⟨*proof*⟩

**lemma** *less-imp-Suc-add*: $m < n \Longrightarrow \exists\, k.\ n = Suc\ (m + k)$
⟨*proof*⟩

**lemma** *le-Suc-ex*: $k \leq l \Longrightarrow (\exists\, n.\ l = k + n)$
**for** $k\ l :: nat$
⟨*proof*⟩

**lemma** *less-natE*:

**assumes** ‹$m < n$›
**obtains** $q$ **where** ‹$n = Suc\ (m + q)$›
⟨*proof*⟩

strict, in 1st argument; proof is by induction on $k > 0$

**lemma** *mult-less-mono2*:
**fixes** $i\ j :: nat$
**assumes** $i < j$ **and** $0 < k$
**shows** $k * i < k * j$
⟨*proof*⟩

Addition is the inverse of subtraction: if $n \leq m$ then $n + (m - n) = m$.

**lemma** *add-diff-inverse-nat*: $\neg\ m < n \Longrightarrow n + (m - n) = m$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *nat-le-iff-add*: $m \leq n \longleftrightarrow (\exists\, k.\ n = m + k)$
**for** $m\ n :: nat$
⟨*proof*⟩

The naturals form an ordered *semidom* and a *dioid*.

**instance** *nat* :: *discrete-linordered-semidom*
⟨*proof*⟩

**instance** *nat* :: *dioid*
⟨*proof*⟩

**declare** *le0*[*simp del*] — This is now $0 \leq ?x$
**declare** *le-0-eq*[*simp del*] — This is now $(?n \leq 0) = (?n = 0)$
**declare** *not-less0*[*simp del*] — This is now $\neg\ ?n < 0$
**declare** *not-gr0*[*simp del*] — This is now $(\neg\ 0 < ?n) = (?n = 0)$

**instance** *nat* :: *ordered-cancel-comm-monoid-add* ⟨*proof*⟩
**instance** *nat* :: *ordered-cancel-comm-monoid-diff* ⟨*proof*⟩

### 17.4.6 *min* and *max*

**global-interpretation** *bot-nat-0*: *ordering-top* ‹($\geq$)› ‹($>$)› ‹$0$::*nat*›
⟨*proof*⟩

**global-interpretation** *max-nat*: *semilattice-neutr-order max* ‹$0$::*nat*› ‹($\geq$)› ‹($>$)›
⟨*proof*⟩

**lemma** *mono-Suc*: *mono Suc*
⟨*proof*⟩

**lemma** *min-0L* [*simp*]: $min\ 0\ n = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *min-0R* [*simp*]: $min\ n\ 0 = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *min-Suc-Suc* [*simp*]: $min\ (Suc\ m)\ (Suc\ n) = Suc\ (min\ m\ n)$
⟨*proof*⟩

**lemma** *min-Suc1*: $min\ (Suc\ n)\ m = (case\ m\ of\ 0 \Rightarrow 0 \mid Suc\ m' \Rightarrow Suc(min\ n\ m'))$
⟨*proof*⟩

**lemma** *min-Suc2*: $min\ m\ (Suc\ n) = (case\ m\ of\ 0 \Rightarrow 0 \mid Suc\ m' \Rightarrow Suc(min\ m'\ n))$
⟨*proof*⟩

**lemma** *max-0L* [*simp*]: $max\ 0\ n = n$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *max-0R* [*simp*]: $max\ n\ 0 = n$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *max-Suc-Suc* [*simp*]: $max\ (Suc\ m)\ (Suc\ n) = Suc\ (max\ m\ n)$
⟨*proof*⟩

**lemma** *max-Suc1*: $max\ (Suc\ n)\ m = (case\ m\ of\ 0 \Rightarrow Suc\ n \mid Suc\ m' \Rightarrow Suc\ (max\ n\ m'))$
⟨*proof*⟩

**lemma** *max-Suc2*: $max\ m\ (Suc\ n) = (case\ m\ of\ 0 \Rightarrow Suc\ n \mid Suc\ m' \Rightarrow Suc\ (max\ m'\ n))$
⟨*proof*⟩

**lemma** *nat-mult-min-left*: $min\ m\ n * q = min\ (m * q)\ (n * q)$
**for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *nat-mult-min-right*: $m * min\ n\ q = min\ (m * n)\ (m * q)$
**for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *nat-add-max-left*: $max\ m\ n + q = max\ (m + q)\ (n + q)$
**for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *nat-add-max-right*: $m + max\ n\ q = max\ (m + n)\ (m + q)$
**for** $m\ n\ q :: nat$

⟨*proof*⟩

**lemma** *nat-mult-max-left*: $max\ m\ n * q = max\ (m * q)\ (n * q)$
**for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *nat-mult-max-right*: $m * max\ n\ q = max\ (m * n)\ (m * q)$
**for** $m\ n\ q :: nat$
⟨*proof*⟩

### 17.4.7 Additional theorems about $(\leq)$

Complete induction, aka course-of-values induction

**instance** $nat :: wellorder$
⟨*proof*⟩

**lemma** *Least-eq-0*[*simp*]: $P\ 0 \Longrightarrow Least\ P = 0$
**for** $P :: nat \Rightarrow bool$
⟨*proof*⟩

**lemma** *Least-Suc*:
**assumes** $P\ n\ \neg\ P\ 0$
**shows** $(LEAST\ n.\ P\ n) = Suc\ (LEAST\ m.\ P\ (Suc\ m))$
⟨*proof*⟩

**lemma** *Least-Suc2*: $P\ n \Longrightarrow Q\ m \Longrightarrow \neg\ P\ 0 \Longrightarrow \forall k.\ P\ (Suc\ k) = Q\ k \Longrightarrow Least\ P = Suc\ (Least\ Q)$
⟨*proof*⟩

**lemma** *ex-least-nat-le*:
**fixes** $P :: nat \Rightarrow bool$
**assumes** $P\ n$
**shows** $\exists k{\leq}n.\ (\forall i{<}k.\ \neg\ P\ i) \wedge P\ k$
⟨*proof*⟩

**lemma** *ex-least-nat-less*:
**fixes** $P :: nat \Rightarrow bool$
**assumes** $P\ n\ \neg\ P\ 0$
**shows** $\exists k{<}n.\ (\forall i{\leq}k.\ \neg\ P\ i) \wedge P\ (Suc\ k)$
⟨*proof*⟩

**lemma** *nat-less-induct*:
**fixes** $P :: nat \Rightarrow bool$
**assumes** $\bigwedge n.\ \forall m.\ m < n \longrightarrow P\ m \Longrightarrow P\ n$
**shows** $P\ n$
⟨*proof*⟩

**lemma** *measure-induct-rule* [*case-names less*]:
**fixes** $f :: 'a \Rightarrow 'b::wellorder$
**assumes** *step*: $\bigwedge x.\ (\bigwedge y.\ f\ y < f\ x \Longrightarrow P\ y) \Longrightarrow P\ x$
**shows** $P\ a$
⟨*proof*⟩

old style induction rules:

**lemma** *measure-induct*:
**fixes** $f :: 'a \Rightarrow 'b::wellorder$
**shows** $(\bigwedge x.\ \forall y.\ f\ y < f\ x \longrightarrow P\ y \Longrightarrow P\ x) \Longrightarrow P\ a$
⟨*proof*⟩

**lemma** *full-nat-induct*:
**assumes** *step*: $\bigwedge n.\ (\forall m.\ Suc\ m \le n \longrightarrow P\ m) \Longrightarrow P\ n$
**shows** $P\ n$
⟨*proof*⟩

An induction rule for establishing binary relations

**lemma** *less-Suc-induct* [*consumes 1*]:
**assumes** *less*: $i < j$
**and** *step*: $\bigwedge i.\ P\ i\ (Suc\ i)$
**and** *trans*: $\bigwedge i\ j\ k.\ i < j \Longrightarrow j < k \Longrightarrow P\ i\ j \Longrightarrow P\ j\ k \Longrightarrow P\ i\ k$
**shows** $P\ i\ j$
⟨*proof*⟩

The method of infinite descent, frequently used in number theory. Provided by Roelof Oosterhuis. $P\ n$ is true for all natural numbers if

- case "0": given $n = 0$ prove $P\ n$
- case "smaller": given $n > 0$ and $\neg\ P\ n$ prove there exists a smaller natural number $m$ such that $\neg\ P\ m$.

**lemma** *infinite-descent*: $(\bigwedge n.\ \neg\ P\ n \Longrightarrow \exists\, m<n.\ \neg\ P\ m) \Longrightarrow P\ n$ **for** $P :: nat \Rightarrow bool$
— compact version without explicit base case
⟨*proof*⟩

**lemma** *infinite-descent0* [*case-names 0 smaller*]:
**fixes** $P :: nat \Rightarrow bool$
**assumes** $P\ 0$
**and** $\bigwedge n.\ n > 0 \Longrightarrow \neg\ P\ n \Longrightarrow \exists\, m.\ m < n \wedge \neg\ P\ m$
**shows** $P\ n$
⟨*proof*⟩

Infinite descent using a mapping to *nat*: $P\ x$ is true for all $x \in D$ if there exists a $V \in D \Rightarrow nat$ and

- case "0": given $V\ x = 0$ prove $P\ x$

- "smaller": given $V\ x > 0$ and $\neg\ P\ x$ prove there exists a $y \in D$ such that $V\ y < V\ x$ and $\neg\ P\ y$.

**corollary** *infinite-descent0-measure* [*case-names 0 smaller*]:
**fixes** $V :: 'a \Rightarrow nat$
**assumes** *1*: $\bigwedge x.\ V\ x = 0 \Longrightarrow P\ x$
**and** *2*: $\bigwedge x.\ V\ x > 0 \Longrightarrow \neg\ P\ x \Longrightarrow \exists\, y.\ V\ y < V\ x \wedge \neg\ P\ y$
**shows** $P\ x$
⟨*proof*⟩

Again, without explicit base case:

**lemma** *infinite-descent-measure*:
**fixes** $V :: 'a \Rightarrow nat$
**assumes** $\bigwedge x.\ \neg\ P\ x \Longrightarrow \exists\, y.\ V\ y < V\ x \wedge \neg\ P\ y$
**shows** $P\ x$
⟨*proof*⟩

A (clumsy) way of lifting $<$ monotonicity to $\leq$ monotonicity

**lemma** *less-mono-imp-le-mono*:
**fixes** $f :: nat \Rightarrow nat$
**and** $i\ j :: nat$
**assumes** $\bigwedge i\ j{::}nat.\ i < j \Longrightarrow f\ i < f\ j$
**and** $i \leq j$
**shows** $f\ i \leq f\ j$
⟨*proof*⟩

non-strict, in 1st argument

**lemma** *add-le-mono1*: $i \leq j \Longrightarrow i + k \leq j + k$
**for** $i\ j\ k :: nat$
⟨*proof*⟩

non-strict, in both arguments

**lemma** *add-le-mono*: $i \leq j \Longrightarrow k \leq l \Longrightarrow i + k \leq j + l$
**for** $i\ j\ k\ l :: nat$
⟨*proof*⟩

**lemma** *le-add2*: $n \leq m + n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *le-add1*: $n \leq n + m$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *less-add-Suc1*: $i < Suc\ (i + m)$
⟨*proof*⟩

**lemma** *less-add-Suc2*: $i < Suc\ (m + i)$

⟨*proof*⟩

**lemma** *less-iff-Suc-add*: $m < n \longleftrightarrow (\exists k.\ n = Suc\ (m + k))$
⟨*proof*⟩

**lemma** *trans-le-add1*: $i \leq j \Longrightarrow i \leq j + m$
**for** $i\ j\ m$ :: *nat*
⟨*proof*⟩

**lemma** *trans-le-add2*: $i \leq j \Longrightarrow i \leq m + j$
**for** $i\ j\ m$ :: *nat*
⟨*proof*⟩

**lemma** *trans-less-add1*: $i < j \Longrightarrow i < j + m$
**for** $i\ j\ m$ :: *nat*
⟨*proof*⟩

**lemma** *trans-less-add2*: $i < j \Longrightarrow i < m + j$
**for** $i\ j\ m$ :: *nat*
⟨*proof*⟩

**lemma** *add-lessD1*: $i + j < k \Longrightarrow i < k$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *not-add-less1* [*iff*]: $\neg\ i + j < i$
**for** $i\ j$ :: *nat*
⟨*proof*⟩

**lemma** *not-add-less2* [*iff*]: $\neg\ j + i < i$
**for** $i\ j$ :: *nat*
⟨*proof*⟩

**lemma** *add-leD1*: $m + k \leq n \Longrightarrow m \leq n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *add-leD2*: $m + k \leq n \Longrightarrow k \leq n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *add-leE*: $m + k \leq n \Longrightarrow (m \leq n \Longrightarrow k \leq n \Longrightarrow R) \Longrightarrow R$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

needs $\bigwedge k$ for *ac-simps* to work

**lemma** *less-add-eq-less*: $\bigwedge k.\ k < l \Longrightarrow m + l = k + n \Longrightarrow m < n$
**for** $l\ m\ n$ :: *nat*
⟨*proof*⟩

### 17.4.8 More results about difference

**lemma** *Suc-diff-le*: $n \leq m \Longrightarrow Suc\ m - n = Suc\ (m - n)$
⟨*proof*⟩

**lemma** *diff-less-Suc*: $m - n < Suc\ m$
⟨*proof*⟩

**lemma** *diff-le-self* [*simp*]: $m - n \leq m$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *less-imp-diff-less*: $j < k \Longrightarrow j - n < k$
**for** $j\ k\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-Suc-less* [*simp*]: $0 < n \Longrightarrow n - Suc\ i < n$
⟨*proof*⟩

**lemma** *diff-add-assoc*: $k \leq j \Longrightarrow (i + j) - k = i + (j - k)$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *add-diff-assoc* [*simp*]: $k \leq j \Longrightarrow i + (j - k) = i + j - k$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *diff-add-assoc2*: $k \leq j \Longrightarrow (j + i) - k = (j - k) + i$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *add-diff-assoc2* [*simp*]: $k \leq j \Longrightarrow j - k + i = j + i - k$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *le-imp-diff-is-add*: $i \leq j \Longrightarrow (j - i = k) = (j = k + i)$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *diff-is-0-eq* [*simp*]: $m - n = 0 \longleftrightarrow m \leq n$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-is-0-eq'* [*simp*]: $m \leq n \Longrightarrow m - n = 0$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *zero-less-diff* [*simp*]: $0 < n - m \longleftrightarrow m < n$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *less-imp-add-positive*:
**assumes** $i < j$
**shows** $\exists k::nat.\ 0 < k \wedge i + k = j$
$\langle proof \rangle$

a nice rewrite for bounded subtraction

**lemma** *nat-minus-add-max*: $n - m + m = max\ n\ m$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *nat-diff-split*: $P\ (a - b) \longleftrightarrow (a < b \longrightarrow P\ 0) \wedge (\forall d.\ a = b + d \longrightarrow P\ d)$
**for** $a\ b :: nat$
— elimination of $-$ on *nat*
$\langle proof \rangle$

**lemma** *nat-diff-split-asm*: $P\ (a - b) \longleftrightarrow \neg\ (a < b \wedge \neg\ P\ 0 \vee (\exists d.\ a = b + d \wedge \neg\ P\ d))$
**for** $a\ b :: nat$
— elimination of $-$ on *nat* in assumptions
$\langle proof \rangle$

**lemmas** *nat-diff-splits* = *nat-diff-split nat-diff-split-asm*

**lemma** *Suc-pred'*: $0 < n \Longrightarrow n = Suc(n - 1)$
$\langle proof \rangle$

**lemma** *add-eq-if*: $m + n = (if\ m = 0\ then\ n\ else\ Suc\ ((m - 1) + n))$
$\langle proof \rangle$

**lemma** *mult-eq-if*: $m * n = (if\ m = 0\ then\ 0\ else\ n + ((m - 1) * n))$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *Suc-diff-eq-diff-pred*: $0 < n \Longrightarrow Suc\ m - n = m - (n - 1)$
$\langle proof \rangle$

**lemma** *diff-Suc-eq-diff-pred*: $m - Suc\ n = (m - 1) - n$
$\langle proof \rangle$

**lemma** *Let-Suc* [*simp*]: $Let\ (Suc\ n)\ f \equiv f\ (Suc\ n)$
$\langle proof \rangle$

### 17.4.9 Monotonicity of multiplication

**lemma** *mult-le-mono1*: $i \leq j \Longrightarrow i * k \leq j * k$
**for** $i\ j\ k :: nat$
$\langle proof \rangle$

**lemma** *mult-le-mono2*: $i \leq j \Longrightarrow k * i \leq k * j$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

$\leq$ monotonicity, BOTH arguments

**lemma** *mult-le-mono*: $i \leq j \Longrightarrow k \leq l \Longrightarrow i * k \leq j * l$
**for** $i\ j\ k\ l$ :: *nat*
⟨*proof*⟩

**lemma** *mult-less-mono1*: $i < j \Longrightarrow 0 < k \Longrightarrow i * k < j * k$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

Differs from the standard *zero-less-mult-iff* in that there are no negative numbers.

**lemma** *nat-0-less-mult-iff* [*simp*]: $0 < m * n \longleftrightarrow 0 < m \land 0 < n$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *one-le-mult-iff* [*simp*]: $Suc\ 0 \leq m * n \longleftrightarrow Suc\ 0 \leq m \land Suc\ 0 \leq n$
⟨*proof*⟩

**lemma** *mult-less-cancel2* [*simp*]: $m * k < n * k \longleftrightarrow 0 < k \land m < n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-less-cancel1* [*simp*]: $k * m < k * n \longleftrightarrow 0 < k \land m < n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-le-cancel1* [*simp*]: $k * m \leq k * n \longleftrightarrow (0 < k \longrightarrow m \leq n)$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *mult-le-cancel2* [*simp*]: $m * k \leq n * k \longleftrightarrow (0 < k \longrightarrow m \leq n)$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *Suc-mult-less-cancel1*: $Suc\ k * m < Suc\ k * n \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** *Suc-mult-le-cancel1*: $Suc\ k * m \leq Suc\ k * n \longleftrightarrow m \leq n$
⟨*proof*⟩

**lemma** *le-square*: $m \leq m * m$
**for** $m$ :: *nat*
⟨*proof*⟩

**lemma** *le-cube*: $m \leq m * (m * m)$

**for** *m* :: *nat*
⟨*proof*⟩

Lemma for *gcd*

**lemma** *mult-eq-self-implies-10*:
**fixes** *m n* :: *nat*
**assumes** $m = m * n$ **shows** $n = 1 \vee m = 0$
⟨*proof*⟩

**lemma** *mono-times-nat*:
**fixes** *n* :: *nat*
**assumes** $n > 0$
**shows** *mono* (*times n*)
⟨*proof*⟩

The lattice order on *nat*.

**instantiation** *nat* :: *distrib-lattice*
**begin**

**definition** (*inf* :: *nat* ⇒ *nat* ⇒ *nat*) = *min*

**definition** (*sup* :: *nat* ⇒ *nat* ⇒ *nat*) = *max*

**instance**
⟨*proof*⟩

**end**

## 17.5 Natural operation of natural numbers on functions

We use the same logical constant for the power operations on functions and relations, in order to share the same syntax.

**consts** *compow* :: *nat* ⇒ *'a* ⇒ *'a*

**abbreviation** *compower* :: *'a* ⇒ *nat* ⇒ *'a* (**infixr** ‹^^› *80*)
**where** *f* ^^ *n* ≡ *compow n f*

**notation** (*latex* **output**)
*compower* (‹(-⁻)› [*1000*] *1000*)

$f$ ^^ $n = f \circ \ldots \circ f$, the $n$-fold composition of $f$

**overloading**
*funpow* ≡ *compow* :: *nat* ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*)
**begin**

**primrec** *funpow* :: *nat* ⇒ (*'a* ⇒ *'a*) ⇒ *'a* ⇒ *'a*
**where**
*funpow 0 f* = *id*

| *funpow* (*Suc n*) $f = f \circ$ *funpow n f*

**end**

**lemma** *funpow-0* [*simp*]: $(f$ ^^ $0)\ x = x$
⟨*proof*⟩

**lemma** *funpow-Suc-right*: $f$ ^^ *Suc n* $= f$ ^^ $n \circ f$
⟨*proof*⟩

**lemmas** *funpow-simps-right* = *funpow.simps*(*1*) *funpow-Suc-right*

For code generation.

**context**
**begin**

**qualified definition** *funpow* :: *nat* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ '*a* $\Rightarrow$ '*a*
**where** *funpow-code-def* [*code-abbrev*]: *funpow* = *compow*

**lemma** [*code*]:
*funpow 0 f* = *id*
*funpow* (*Suc n*) $f = f \circ$ *funpow n f*
⟨*proof*⟩

**end**

**lemma** *funpow-add*: $f$ ^^ $(m + n) = f$ ^^ $m \circ f$ ^^ $n$
⟨*proof*⟩

**lemma** *funpow-mult*: $(f$ ^^ $m)$ ^^ $n = f$ ^^ $(m * n)$
**for** *f* :: '*a* $\Rightarrow$ '*a*
⟨*proof*⟩

**lemma** *funpow-swap1*: $f\ ((f$ ^^ $n)\ x) = (f$ ^^ $n)\ (f\ x)$
⟨*proof*⟩

**lemma** *comp-funpow*: *comp f* ^^ $n$ = *comp* $(f$ ^^ $n)$
**for** *f* :: '*a* $\Rightarrow$ '*a*
⟨*proof*⟩

**lemma** *Suc-funpow*[*simp*]: *Suc* ^^ $n = ((+)\ n)$
⟨*proof*⟩

**lemma** *id-funpow*[*simp*]: *id* ^^ $n$ = *id*
⟨*proof*⟩

**lemma** *funpow-mono*: *mono f* $\Longrightarrow A \le B \Longrightarrow (f$ ^^ $n)\ A \le (f$ ^^ $n)\ B$
**for** *f* :: '*a* $\Rightarrow$ ('*a*::*order*)
⟨*proof*⟩

**lemma** *funpow-mono2*:
**assumes** *mono f*
**and** $i \leq j$
**and** $x \leq y$
**and** $x \leq f\ x$
**shows** $(f \hat{\ } \hat{\ } i)\ x \leq (f \hat{\ } \hat{\ } j)\ y$
⟨*proof*⟩

**lemma** *inj-fn*[*simp*]:
**fixes** $f::'a \Rightarrow 'a$
**assumes** *inj f*
**shows** *inj* $(f \hat{\ } \hat{\ } n)$
⟨*proof*⟩

**lemma** *surj-fn*[*simp*]:
**fixes** $f::'a \Rightarrow 'a$
**assumes** *surj f*
**shows** *surj* $(f \hat{\ } \hat{\ } n)$
⟨*proof*⟩

**lemma** *bij-fn*[*simp*]:
**fixes** $f::'a \Rightarrow 'a$
**assumes** *bij f*
**shows** *bij* $(f \hat{\ } \hat{\ } n)$
⟨*proof*⟩

**lemma** *bij-betw-funpow*:
**assumes** *bij-betw f S S* **shows** *bij-betw* $(f \hat{\ } \hat{\ } n)\ S\ S$
⟨*proof*⟩

## 17.6 Kleene iteration

**lemma** *Kleene-iter-lpfp*:
**fixes** $f :: 'a::order\text{-}bot \Rightarrow 'a$
**assumes** *mono f*
**and** $f\ p \leq p$
**shows** $(f \hat{\ } \hat{\ } k)\ bot \leq p$
⟨*proof*⟩

**lemma** *lfp-Kleene-iter*:
**assumes** *mono f*
**and** $(f \hat{\ } \hat{\ } Suc\ k)\ bot = (f \hat{\ } \hat{\ } k)\ bot$
**shows** $lfp\ f = (f \hat{\ } \hat{\ } k)\ bot$
⟨*proof*⟩

**lemma** *mono-pow*: $mono\ f \Longrightarrow mono\ (f \hat{\ } \hat{\ } n)$
**for** $f :: 'a \Rightarrow 'a::order$
⟨*proof*⟩

**lemma** *lfp-funpow*:
  **assumes** *f*: *mono f*
  **shows** *lfp* $(f$ ^^ *Suc n*$)$ = *lfp f*
⟨*proof*⟩

**lemma** *gfp-funpow*:
  **assumes** *f*: *mono f*
  **shows** *gfp* $(f$ ^^ *Suc n*$)$ = *gfp f*
⟨*proof*⟩

**lemma** *Kleene-iter-gpfp*:
  **fixes** *f* :: *'a::order-top* ⇒ *'a*
  **assumes** *mono f*
    **and** $p \leq f\ p$
  **shows** $p \leq (f$ ^^ $k)$ *top*
⟨*proof*⟩

**lemma** *gfp-Kleene-iter*:
  **assumes** *mono f*
    **and** $(f$ ^^ *Suc k*$)$ *top* = $(f$ ^^ $k)$ *top*
  **shows** *gfp f* = $(f$ ^^ $k)$ *top*
    (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

## 17.7 Embedding of the naturals into any *semiring-1*: *of-nat*

**context** *semiring-1*
**begin**

**definition** *of-nat* :: *nat* ⇒ *'a*
  **where** *of-nat n* = (*plus 1* ^^ *n*) *0*

**lemma** *of-nat-simps* [*simp*]:
  **shows** *of-nat-0*: *of-nat 0* = *0*
    **and** *of-nat-Suc*: *of-nat* (*Suc m*) = *1* + *of-nat m*
  ⟨*proof*⟩

**lemma** *of-nat-1* [*simp*]: *of-nat 1* = *1*
  ⟨*proof*⟩

**lemma** *of-nat-add* [*simp*]: *of-nat* $(m + n)$ = *of-nat m* + *of-nat n*
  ⟨*proof*⟩

**lemma** *of-nat-mult* [*simp*]: *of-nat* $(m * n)$ = *of-nat m* ∗ *of-nat n*
  ⟨*proof*⟩

**lemma** *mult-of-nat-commute*: *of-nat x* ∗ *y* = *y* ∗ *of-nat x*
  ⟨*proof*⟩

**primrec** *of-nat-aux* :: $('a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a$
**where**
*of-nat-aux inc 0 i = i*
| *of-nat-aux inc (Suc n) i = of-nat-aux inc n (inc i)* — tail recursive

**lemma** *of-nat-code*: *of-nat n = of-nat-aux* $(\lambda i.\ i + 1)$ *n 0*
⟨*proof*⟩

**lemma** *of-nat-of-bool* [*simp*]:
*of-nat (of-bool P) = of-bool P*
⟨*proof*⟩

**end**

**declare** *of-nat-code* [*code*]

**context** *semiring-1-cancel*
**begin**

**lemma** *of-nat-diff* [*simp*]:
‹*of-nat (m − n) = of-nat m − of-nat n*› **if** ‹$n \leq m$›
⟨*proof*⟩

**lemma** *of-nat-diff-if*: ‹*of-nat (m − n) = (if* $n \leq m$ *then of-nat m − of-nat n else 0)*›
⟨*proof*⟩

**end**

Class for unital semirings with characteristic zero. Includes non-ordered rings like the complex numbers.

**class** *semiring-char-0 = semiring-1 +*
**assumes** *inj-of-nat*: *inj of-nat*
**begin**

**lemma** *of-nat-eq-iff* [*simp*]: *of-nat m = of-nat n* $\longleftrightarrow$ *m = n*
⟨*proof*⟩

Special cases where either operand is zero

**lemma** *of-nat-0-eq-iff* [*simp*]: *0 = of-nat n* $\longleftrightarrow$ *0 = n*
⟨*proof*⟩

**lemma** *of-nat-eq-0-iff* [*simp*]: *of-nat m = 0* $\longleftrightarrow$ *m = 0*
⟨*proof*⟩

**lemma** *of-nat-1-eq-iff* [*simp*]: *1 = of-nat n* $\longleftrightarrow$ *n=1*
⟨*proof*⟩

**lemma** *of-nat-eq-1-iff* [*simp*]: *of-nat* $n = 1 \longleftrightarrow n=1$
⟨*proof*⟩

**lemma** *of-nat-neq-0* [*simp*]: *of-nat* (*Suc* $n$) $\neq 0$
⟨*proof*⟩

**lemma** *of-nat-0-neq* [*simp*]: $0 \neq$ *of-nat* (*Suc* $n$)
⟨*proof*⟩

**end**

**class** *ring-char-0* = *ring-1* + *semiring-char-0*

**lemma** (**in** *ordered-semiring-1*) *of-nat-0-le-iff* [*simp*]: $0 \leq$ *of-nat* $n$
⟨*proof*⟩

**context** *linordered-nonzero-semiring*
**begin**

**lemma** *of-nat-less-0-iff* [*simp*]: $\neg$ *of-nat* $m < 0$
⟨*proof*⟩

**lemma** *of-nat-mono*[*simp*]: $i \leq j \Longrightarrow$ *of-nat* $i \leq$ *of-nat* $j$
⟨*proof*⟩

**lemma** *of-nat-less-iff* [*simp*]: *of-nat* $m <$ *of-nat* $n \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** *of-nat-le-iff* [*simp*]: *of-nat* $m \leq$ *of-nat* $n \longleftrightarrow m \leq n$
⟨*proof*⟩

**lemma** *less-imp-of-nat-less*: $m < n \Longrightarrow$ *of-nat* $m <$ *of-nat* $n$
⟨*proof*⟩

**lemma** *of-nat-less-imp-less*: *of-nat* $m <$ *of-nat* $n \Longrightarrow m < n$
⟨*proof*⟩

Every *linordered-nonzero-semiring* has characteristic zero.

**subclass** *semiring-char-0*
⟨*proof*⟩

Special cases where either operand is zero

**lemma** *of-nat-le-0-iff* [*simp*]: *of-nat* $m \leq 0 \longleftrightarrow m = 0$
⟨*proof*⟩

**lemma** *of-nat-0-less-iff* [*simp*]: $0 <$ *of-nat* $n \longleftrightarrow 0 < n$
⟨*proof*⟩

**end**

**context** *linordered-nonzero-semiring*
**begin**

**lemma** *of-nat-max*: *of-nat* (*max x y*) = *max* (*of-nat x*) (*of-nat y*)
⟨*proof*⟩

**lemma** *of-nat-min*: *of-nat* (*min x y*) = *min* (*of-nat x*) (*of-nat y*)
⟨*proof*⟩

**end**

**context** *linordered-semidom*
**begin**

**subclass** *linordered-nonzero-semiring* ⟨*proof*⟩

**subclass** *semiring-char-0* ⟨*proof*⟩

**end**

**context** *linordered-idom*
**begin**

**lemma** *abs-of-nat* [*simp*]:
|*of-nat n*| = *of-nat n*
⟨*proof*⟩

**lemma** *sgn-of-nat* [*simp*]:
*sgn* (*of-nat n*) = *of-bool* (*n* > *0*)
⟨*proof*⟩

**end**

**lemma** *of-nat-id* [*simp*]: *of-nat n* = *n*
⟨*proof*⟩

**lemma** *of-nat-eq-id* [*simp*]: *of-nat* = *id*
⟨*proof*⟩

## 17.8 The set of natural numbers

**context** *semiring-1*
**begin**

**definition** *Nats* :: *'a set* (‹ℕ›)
**where** ℕ = *range of-nat*

**lemma** *of-nat-in-Nats* [*simp*]: *of-nat n* ∈ ℕ

⟨*proof*⟩

**lemma** *Nats-0* [*simp*]: $0 \in \mathbb{N}$
⟨*proof*⟩

**lemma** *Nats-1* [*simp*]: $1 \in \mathbb{N}$
⟨*proof*⟩

**lemma** *Nats-add* [*simp*]: $a \in \mathbb{N} \Longrightarrow b \in \mathbb{N} \Longrightarrow a + b \in \mathbb{N}$
⟨*proof*⟩

**lemma** *Nats-mult* [*simp*]: $a \in \mathbb{N} \Longrightarrow b \in \mathbb{N} \Longrightarrow a * b \in \mathbb{N}$
⟨*proof*⟩

**lemma** *Nats-cases* [*cases set*: *Nats*]:
**assumes** $x \in \mathbb{N}$
**obtains** (*of-nat*) $n$ **where** $x = \mathit{of\text{-}nat}\ n$
⟨*proof*⟩

**lemma** *Nats-induct* [*case-names of-nat*, *induct set*: *Nats*]: $x \in \mathbb{N} \Longrightarrow (\bigwedge n.\ P\ (\mathit{of\text{-}nat}\ n)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *Nats-nonempty* [*simp*]: $\mathbb{N} \neq \{\}$
⟨*proof*⟩

**end**

**lemma** *Nats-diff* [*simp*]:
**fixes** $a$:: *'a::linordered-idom*
**assumes** $a \in \mathbb{N}$ $b \in \mathbb{N}$ $b \leq a$ **shows** $a - b \in \mathbb{N}$
⟨*proof*⟩

## 17.9 Further arithmetic facts concerning the natural numbers

**lemma** *subst-equals*:
**assumes** $t = s$ **and** $u = t$
**shows** $u = s$
⟨*proof*⟩

**locale** *nat-arith*
**begin**

**lemma** *add1*: $(A::'a::\mathit{comm\text{-}monoid\text{-}add}) \equiv k + a \Longrightarrow A + b \equiv k + (a + b)$
⟨*proof*⟩

**lemma** *add2*: $(B::'a::\mathit{comm\text{-}monoid\text{-}add}) \equiv k + b \Longrightarrow a + B \equiv k + (a + b)$
⟨*proof*⟩

**lemma** *suc1*: $A == k + a \Longrightarrow Suc\ A \equiv k + Suc\ a$
⟨*proof*⟩

**lemma** *rule0*: $(a::'a::comm\text{-}monoid\text{-}add) \equiv a + 0$
⟨*proof*⟩

**end**

⟨*ML*⟩

**context** *preorder*
**begin**

**lemma** *lift-Suc-mono-le*:
**assumes** *mono*: $\bigwedge n.\ f\ n \leq f\ (Suc\ n)$
**and** $n \leq n'$
**shows** $f\ n \leq f\ n'$
⟨*proof*⟩

**lemma** *lift-Suc-antimono-le*:
**assumes** *mono*: $\bigwedge n.\ f\ n \geq f\ (Suc\ n)$
**and** $n \leq n'$
**shows** $f\ n \geq f\ n'$
⟨*proof*⟩

**lemma** *lift-Suc-mono-less*:
**assumes** *mono*: $\bigwedge n.\ f\ n < f\ (Suc\ n)$
**and** $n < n'$
**shows** $f\ n < f\ n'$
⟨*proof*⟩

**lemma** *lift-Suc-mono-less-iff*: $(\bigwedge n.\ f\ n < f\ (Suc\ n)) \Longrightarrow f\ n < f\ m \longleftrightarrow n < m$
⟨*proof*⟩

**end**

**lemma** *mono-iff-le-Suc*: $mono\ f \longleftrightarrow (\forall n.\ f\ n \leq f\ (Suc\ n))$
⟨*proof*⟩

**lemma** *antimono-iff-le-Suc*: $antimono\ f \longleftrightarrow (\forall n.\ f\ (Suc\ n) \leq f\ n)$
⟨*proof*⟩

**lemma** *strict-mono-Suc-iff*: $strict\text{-}mono\ f \longleftrightarrow (\forall n.\ f\ n < f\ (Suc\ n))$
⟨*proof*⟩

**lemma** *strict-mono-add*: $strict\text{-}mono\ (\lambda n::'a::linordered\text{-}semidom.\ n + k)$
⟨*proof*⟩

**lemma** *mono-nat-linear-lb*:
  **fixes** $f :: nat \Rightarrow nat$
  **assumes** $\bigwedge m\ n.\ m < n \Longrightarrow f\ m < f\ n$
  **shows** $f\ m + k \leq f\ (m + k)$
$\langle proof \rangle$

**lemma** *bex-const1-if-mono-below-diag*: **fixes** $f :: nat \Rightarrow nat$ **assumes** *mono f*
**shows** $f\ n < n \Longrightarrow \exists\, i{<}n.\ f(Suc\ i) = f\ i$
$\langle proof \rangle$

**lemma** *bex-const1-if-mono-below-diag-Suc*:
**fixes** $f :: nat \Rightarrow nat$ **assumes** *mono f* $f(Suc\ m) \leq m$
**shows** $\exists\, i{\leq}m.\ f\ (Suc\ i) = f\ i$
$\langle proof \rangle$

Subtraction laws, mostly by Clemens Ballarin

**lemma** *diff-less-mono*:
  **fixes** $a\ b\ c :: nat$
  **assumes** $a < b$ **and** $c \leq a$
  **shows** $a - c < b - c$
$\langle proof \rangle$

**lemma** *less-diff-conv*: $i < j - k \longleftrightarrow i + k < j$
  **for** $i\ j\ k :: nat$
  $\langle proof \rangle$

**lemma** *less-diff-conv2*: $k \leq j \Longrightarrow j - k < i \longleftrightarrow j < i + k$
  **for** $j\ k\ i :: nat$
  $\langle proof \rangle$

**lemma** *le-diff-conv*: $j - k \leq i \longleftrightarrow j \leq i + k$
  **for** $j\ k\ i :: nat$
  $\langle proof \rangle$

**lemma** *diff-diff-cancel* [*simp*]: $i \leq n \Longrightarrow n - (n - i) = i$
  **for** $i\ n :: nat$
  $\langle proof \rangle$

**lemma** *diff-less* [*simp*]: $0 < n \Longrightarrow 0 < m \Longrightarrow m - n < m$
  **for** $i\ n :: nat$
  $\langle proof \rangle$

Simplification of relational expressions involving subtraction

**lemma** *diff-diff-eq*: $k \leq m \Longrightarrow k \leq n \Longrightarrow m - k - (n - k) = m - n$
  **for** $m\ n\ k :: nat$
  $\langle proof \rangle$

**hide-fact** (**open**) *diff-diff-eq*

**lemma** *eq-diff-iff*: $k \leq m \Longrightarrow k \leq n \Longrightarrow m - k = n - k \longleftrightarrow m = n$
**for** $m\ n\ k :: nat$
⟨*proof*⟩

**lemma** *less-diff-iff*: $k \leq m \Longrightarrow k \leq n \Longrightarrow m - k < n - k \longleftrightarrow m < n$
**for** $m\ n\ k :: nat$
⟨*proof*⟩

**lemma** *le-diff-iff*: $k \leq m \Longrightarrow k \leq n \Longrightarrow m - k \leq n - k \longleftrightarrow m \leq n$
**for** $m\ n\ k :: nat$
⟨*proof*⟩

**lemma** *le-diff-iff'*: $a \leq c \Longrightarrow b \leq c \Longrightarrow c - a \leq c - b \longleftrightarrow b \leq a$
**for** $a\ b\ c :: nat$
⟨*proof*⟩

(Anti)Monotonicity of subtraction – by Stephan Merz

**lemma** *diff-le-mono*: $m \leq n \Longrightarrow m - l \leq n - l$
**for** $m\ n\ l :: nat$
⟨*proof*⟩

**lemma** *diff-le-mono2*: $m \leq n \Longrightarrow l - n \leq l - m$
**for** $m\ n\ l :: nat$
⟨*proof*⟩

**lemma** *diff-less-mono2*: $m < n \Longrightarrow m < l \Longrightarrow l - n < l - m$
**for** $m\ n\ l :: nat$
⟨*proof*⟩

**lemma** *diffs0-imp-equal*: $m - n = 0 \Longrightarrow n - m = 0 \Longrightarrow m = n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *min-diff*: $min\ (m - i)\ (n - i) = min\ m\ n - i$
**for** $m\ n\ i :: nat$
⟨*proof*⟩

**lemma** *inj-on-diff-nat*:
**fixes** $k :: nat$
**assumes** $\bigwedge n.\ n \in N \Longrightarrow k \leq n$
**shows** $inj\text{-}on\ (\lambda n.\ n - k)\ N$
⟨*proof*⟩

Rewriting to pull differences out

**lemma** *diff-diff-right* [*simp*]: $k \leq j \Longrightarrow i - (j - k) = i + k - j$
**for** $i\ j\ k :: nat$
⟨*proof*⟩

**lemma** *diff-Suc-diff-eq1* [*simp*]:

**assumes** $k \leq j$
**shows** $i - Suc\ (j - k) = i + k - Suc\ j$
⟨*proof*⟩

**lemma** *diff-Suc-diff-eq2* [*simp*]:
**assumes** $k \leq j$
**shows** $Suc\ (j - k) - i = Suc\ j - (k + i)$
⟨*proof*⟩

**lemma** *Suc-diff-Suc*:
**assumes** $n < m$
**shows** $Suc\ (m - Suc\ n) = m - n$
⟨*proof*⟩

**lemma** *one-less-mult*: $Suc\ 0 < n \Longrightarrow Suc\ 0 < m \Longrightarrow Suc\ 0 < m * n$
⟨*proof*⟩

**lemma** *n-less-m-mult-n*: $0 < n \Longrightarrow Suc\ 0 < m \Longrightarrow n < m * n$
⟨*proof*⟩

**lemma** *n-less-n-mult-m*: $0 < n \Longrightarrow Suc\ 0 < m \Longrightarrow n < n * m$
⟨*proof*⟩

Induction starting beyond zero

**lemma** *nat-induct-at-least* [*consumes 1*, *case-names base Suc*]:
$P\ n$ **if** $n \geq m\ P\ m\ \bigwedge n.\ n \geq m \Longrightarrow P\ n \Longrightarrow P\ (Suc\ n)$
⟨*proof*⟩

**lemma** *nat-induct-non-zero* [*consumes 1*, *case-names 1 Suc*]:
$P\ n$ **if** $n > 0\ P\ 1\ \bigwedge n.\ n > 0 \Longrightarrow P\ n \Longrightarrow P\ (Suc\ n)$
⟨*proof*⟩

Specialized induction principles that work "backwards":

**lemma** *inc-induct* [*consumes 1*, *case-names base step*]:
**assumes** *less*: $i \leq j$
**and** *base*: $P\ j$
**and** *step*: $\bigwedge n.\ i \leq n \Longrightarrow n < j \Longrightarrow P\ (Suc\ n) \Longrightarrow P\ n$
**shows** $P\ i$
⟨*proof*⟩

**lemma** *strict-inc-induct* [*consumes 1*, *case-names base step*]:
**assumes** *less*: $i < j$
**and** *base*: $\bigwedge i.\ j = Suc\ i \Longrightarrow P\ i$
**and** *step*: $\bigwedge i.\ Suc\ i < j \Longrightarrow P\ (Suc\ i) \Longrightarrow P\ i$
**shows** $P\ i$
⟨*proof*⟩

**lemma** *zero-induct-lemma*: $P\ k \Longrightarrow (\bigwedge n.\ P\ (Suc\ n) \Longrightarrow P\ n) \Longrightarrow P\ (k - i)$
⟨*proof*⟩

**lemma** *zero-induct*: $P\ k \Longrightarrow (\bigwedge n.\ P\ (Suc\ n) \Longrightarrow P\ n) \Longrightarrow P\ 0$
⟨*proof*⟩

Further induction rule similar to $[\![?i \le ?j;\ ?P\ ?j;\ \bigwedge n.\ [\![?i \le n;\ n < ?j;\ ?P\ (Suc\ n)]\!] \Longrightarrow ?P\ n]\!] \Longrightarrow ?P\ ?i$.

**lemma** *dec-induct* [*consumes 1*, *case-names base step*]:
$i \le j \Longrightarrow P\ i \Longrightarrow (\bigwedge n.\ i \le n \Longrightarrow n < j \Longrightarrow P\ n \Longrightarrow P\ (Suc\ n)) \Longrightarrow P\ j$
⟨*proof*⟩

**lemma** *transitive-stepwise-le*:
**assumes** $m \le n\ \bigwedge x.\ R\ x\ x\ \bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$ **and** $\bigwedge n.\ R\ n\ (Suc\ n)$
**shows** $R\ m\ n$
⟨*proof*⟩

### 17.9.1 Greatest operator

**lemma** *ex-has-greatest-nat*:
$P\ (k{::}nat) \Longrightarrow \forall y.\ P\ y \longrightarrow y \le b \Longrightarrow \exists x.\ P\ x \wedge (\forall y.\ P\ y \longrightarrow y \le x)$
⟨*proof*⟩

**lemma**
**fixes** $k{::}nat$
**assumes** $P\ k$ **and** *minor*: $\bigwedge y.\ P\ y \Longrightarrow y \le b$
**shows** *GreatestI-nat*: $P\ (Greatest\ P)$
**and** *Greatest-le-nat*: $k \le Greatest\ P$
⟨*proof*⟩

**lemma** *GreatestI-ex-nat*:
$[\![\ \exists k{::}nat.\ P\ k;\ \ \bigwedge y.\ P\ y \Longrightarrow y \le b\ ]\!] \Longrightarrow P\ (Greatest\ P)$
⟨*proof*⟩

## 17.10 Monotonicity of *funpow*

**lemma** *funpow-increasing*: $m \le n \Longrightarrow mono\ f \Longrightarrow (f\ \hat{}\hat{}\ n)\ \top \le (f\ \hat{}\hat{}\ m)\ \top$
**for** $f :: 'a{::}order\text{-}top \Rightarrow 'a$
⟨*proof*⟩

**lemma** *funpow-decreasing*: $m \le n \Longrightarrow mono\ f \Longrightarrow (f\ \hat{}\hat{}\ m)\ \bot \le (f\ \hat{}\hat{}\ n)\ \bot$
**for** $f :: 'a{::}order\text{-}bot \Rightarrow 'a$
⟨*proof*⟩

**lemma** *mono-funpow*: $mono\ Q \Longrightarrow mono\ (\lambda i.\ (Q\ \hat{}\hat{}\ i)\ \bot)$
**for** $Q :: 'a{::}order\text{-}bot \Rightarrow 'a$
⟨*proof*⟩

**lemma** *antimono-funpow*: $mono\ Q \Longrightarrow antimono\ (\lambda i.\ (Q\ \hat{}\hat{}\ i)\ \top)$
**for** $Q :: 'a{::}order\text{-}top \Rightarrow 'a$
⟨*proof*⟩

## 17.11 Kleene's fixed point theorem for continuous functions

Kleene's fixed point theorem shows that the *lfp* of a omega-continuous function can be obtained as the supremum of an omega chain. It only requires an omega-complete partial order. We prove it here for complete lattices because the latter structures are not defined in Main but the theorem is also useful for complete lattices.

**definition** *omega-chain* :: $(nat \Rightarrow ('a{::}complete\text{-}lattice)) \Rightarrow bool$ **where**
$omega\text{-}chain\ C = (\forall\, i.\ C\ i \le C(Suc\ i))$

**definition** *omega-cont* :: $(('a{::}complete\text{-}lattice) \Rightarrow ('b{::}complete\text{-}lattice)) \Rightarrow bool$ **where**
$omega\text{-}cont\ f = (\forall\, C.\ omega\text{-}chain\ C \longrightarrow f(SUP\ n.\ C\ n) = (SUP\ n.\ f(C\ n)))$

**lemma** *omega-chain-mono*: $omega\text{-}chain\ C \Longrightarrow i \le j \Longrightarrow C\ i \le C\ j$
⟨*proof*⟩

**lemma** *mono-if-omega-cont*: **fixes** $f :: ('a{::}complete\text{-}lattice) \Rightarrow ('b{::}complete\text{-}lattice)$
**assumes** *omega-cont f* **shows** *mono f*
⟨*proof*⟩

**lemma** *omega-chain-iterates*: **fixes** $f :: ('a{::}complete\text{-}lattice) \Rightarrow 'a$
**assumes** *mono f* **shows** $omega\text{-}chain(\lambda n.\ (f^{\frown\frown}n)\ bot)$
⟨*proof*⟩

**theorem** *Kleene-lfp*:
**assumes** *omega-cont f* **shows** $lfp\ f = (SUP\ n.\ (f^{\frown\frown}n)\ bot)$ (**is** $- = ?U$)
⟨*proof*⟩

## 17.12 The divides relation on *nat*

**lemma** *dvd-1-left* [*iff*]: *Suc 0 dvd k*
⟨*proof*⟩

**lemma** *dvd-1-iff-1* [*simp*]: $m\ dvd\ Suc\ 0 \longleftrightarrow m = Suc\ 0$
⟨*proof*⟩

**lemma** *nat-dvd-1-iff-1* [*simp*]: $m\ dvd\ 1 \longleftrightarrow m = 1$
**for** $m :: nat$
⟨*proof*⟩

**lemma** *dvd-antisym*: $m\ dvd\ n \Longrightarrow n\ dvd\ m \Longrightarrow m = n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *dvd-diff-nat* [*simp*]: $k\ dvd\ m \Longrightarrow k\ dvd\ n \Longrightarrow k\ dvd\ (m - n)$
**for** $k\ m\ n :: nat$
⟨*proof*⟩

**lemma** *dvd-diffD*:
**fixes** $k\ m\ n :: nat$
**assumes** $k\ dvd\ m - n\ k\ dvd\ n\ n \leq m$
**shows** $k\ dvd\ m$
$\langle proof \rangle$

**lemma** *dvd-diffD1*: $k\ dvd\ m - n \Longrightarrow k\ dvd\ m \Longrightarrow n \leq m \Longrightarrow k\ dvd\ n$
**for** $k\ m\ n :: nat$
$\langle proof \rangle$

**lemma** *dvd-mult-cancel*:
**fixes** $m\ n\ k :: nat$
**assumes** $k * m\ dvd\ k * n$ **and** $0 < k$
**shows** $m\ dvd\ n$
$\langle proof \rangle$

**lemma** *dvd-mult-cancel1*:
**fixes** $m\ n :: nat$
**assumes** $0 < m$
**shows** $m * n\ dvd\ m \longleftrightarrow n = 1$
$\langle proof \rangle$

**lemma** *dvd-mult-cancel2*: $0 < m \Longrightarrow n * m\ dvd\ m \longleftrightarrow n = 1$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *dvd-imp-le*: $k\ dvd\ n \Longrightarrow 0 < n \Longrightarrow k \leq n$
**for** $k\ n :: nat$
$\langle proof \rangle$

**lemma** *nat-dvd-not-less*: $0 < m \Longrightarrow m < n \Longrightarrow \neg\ n\ dvd\ m$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *less-eq-dvd-minus*:
**fixes** $m\ n :: nat$
**assumes** $m \leq n$
**shows** $m\ dvd\ n \longleftrightarrow m\ dvd\ n - m$
$\langle proof \rangle$

**lemma** *dvd-minus-self*: $m\ dvd\ n - m \longleftrightarrow n < m \vee m\ dvd\ n$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *dvd-minus-add*:
**fixes** $m\ n\ q\ r :: nat$
**assumes** $q \leq n\ q \leq r * m$
**shows** $m\ dvd\ n - q \longleftrightarrow m\ dvd\ n + (r * m - q)$
$\langle proof \rangle$

## 17.13 Aliasses

**lemma** *nat-mult-1*: $1 * n = n$
**for** $n$ :: *nat*
⟨*proof*⟩

**lemma** *nat-mult-1-right*: $n * 1 = n$
**for** $n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-mult-distrib*: $(m - n) * k = (m * k) - (n * k)$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-mult-distrib2*: $k * (m - n) = (k * m) - (k * n)$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *le-diff-conv2*: $k \leq j \Longrightarrow (i \leq j - k) = (i + k \leq j)$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *diff-self-eq-0* [*simp*]: $m - m = 0$
**for** $m$ :: *nat*
⟨*proof*⟩

**lemma** *diff-diff-left* [*simp*]: $i - j - k = i - (j + k)$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *diff-commute*: $i - j - k = i - k - j$
**for** $i\ j\ k$ :: *nat*
⟨*proof*⟩

**lemma** *diff-add-inverse*: $(n + m) - n = m$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-add-inverse2*: $(m + n) - n = m$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-cancel*: $(k + m) - (k + n) = m - n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-cancel2*: $(m + k) - (n + k) = m - n$
**for** $k\ m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *diff-add-0*: $n - (n + m) = 0$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *add-mult-distrib2*: $k * (m + n) = (k * m) + (k * n)$
**for** $k\ m\ n :: nat$
⟨*proof*⟩

**lemmas** *nat-distrib* =
*add-mult-distrib distrib-left diff-mult-distrib diff-mult-distrib2*

### 17.14 Size of a datatype value

**class** *size* =
**fixes** *size* :: $'a \Rightarrow nat$ — see further theory *Wellfounded*

**instantiation** *nat* :: *size*
**begin**

**definition** *size-nat* **where** [*simp*, *code*]: *size* (*n*::*nat*) = *n*

**instance** ⟨*proof*⟩

**end**

**lemmas** *size-nat* = *size-nat-def*

**lemma** *size-neq-size-imp-neq*: *size* $x \neq$ *size* $y \Longrightarrow x \neq y$
⟨*proof*⟩

### 17.15 Code module namespace

**code-identifier**
**code-module** *Nat* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**hide-const** (**open**) *of-nat-aux*

**end**

# 18 Fields

**theory** *Fields*
**imports** *Nat*
**begin**

### 18.1 Division rings

A division ring is like a field, but without the commutativity requirement.

**class** *inverse* = *divide* +
  **fixes** *inverse* :: $'a \Rightarrow 'a$
**begin**

**abbreviation** *inverse-divide* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** ‹'/› *70*)
**where**
  *inverse-divide* ≡ *divide*

**end**

Setup for linear arithmetic prover

⟨*ML*⟩

**lemmas** [*linarith-split*] = *nat-diff-split split-min split-max abs-split*

Lemmas *divide-simps* move division to the outside and eliminates them on (in)equalities.

**named-theorems** *divide-simps rewrite rules to eliminate divisions*

**class** *division-ring* = *ring-1* + *inverse* +
  **assumes** *left-inverse* [*simp*]: $a \neq 0 \Longrightarrow inverse\ a * a = 1$
  **assumes** *right-inverse* [*simp*]: $a \neq 0 \Longrightarrow a * inverse\ a = 1$
  **assumes** *divide-inverse*: $a\ /\ b = a * inverse\ b$
  **assumes** *inverse-zero* [*simp*]: $inverse\ 0 = 0$
**begin**

**subclass** *ring-1-no-zero-divisors*
⟨*proof*⟩

**lemma** *nonzero-imp-inverse-nonzero*:
  $a \neq 0 \Longrightarrow inverse\ a \neq 0$
⟨*proof*⟩

**lemma** *inverse-zero-imp-zero*:
  **assumes** $inverse\ a = 0$ **shows** $a = 0$
⟨*proof*⟩

**lemma** *inverse-unique*:
  **assumes** *ab*: $a * b = 1$
  **shows** $inverse\ a = b$
⟨*proof*⟩

**lemma** *nonzero-inverse-minus-eq*:
  $a \neq 0 \Longrightarrow inverse\ (-\ a) = -\ inverse\ a$
⟨*proof*⟩

**lemma** *nonzero-inverse-inverse-eq*:
  $a \neq 0 \Longrightarrow inverse\ (inverse\ a) = a$
⟨*proof*⟩

**lemma** *nonzero-inverse-eq-imp-eq*:
**assumes** $inverse\ a = inverse\ b$ **and** $a \neq 0$ **and** $b \neq 0$
**shows** $a = b$
⟨*proof*⟩

**lemma** *inverse-1* [*simp*]: $inverse\ 1 = 1$
⟨*proof*⟩

**subclass** *divide-trivial*
⟨*proof*⟩

**lemma** *nonzero-inverse-mult-distrib*:
**assumes** $a \neq 0$ **and** $b \neq 0$
**shows** $inverse\ (a * b) = inverse\ b * inverse\ a$
⟨*proof*⟩

**lemma** *division-ring-inverse-add*:
$a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow inverse\ a + inverse\ b = inverse\ a * (a + b) * inverse\ b$
⟨*proof*⟩

**lemma** *division-ring-inverse-diff*:
$a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow inverse\ a - inverse\ b = inverse\ a * (b - a) * inverse\ b$
⟨*proof*⟩

**lemma** *right-inverse-eq*: $b \neq 0 \Longrightarrow a\ /\ b = 1 \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *nonzero-inverse-eq-divide*: $a \neq 0 \Longrightarrow inverse\ a = 1\ /\ a$
⟨*proof*⟩

**lemma** *divide-self* [*simp*]: $a \neq 0 \Longrightarrow a\ /\ a = 1$
⟨*proof*⟩

**lemma** *inverse-eq-divide* [*field-simps*, *field-split-simps*, *divide-simps*]: $inverse\ a = 1\ /\ a$
⟨*proof*⟩

**lemma** *add-divide-distrib*: $(a+b)\ /\ c = a/c + b/c$
⟨*proof*⟩

**lemma** *times-divide-eq-right* [*simp*]: $a * (b\ /\ c) = (a * b)\ /\ c$
⟨*proof*⟩

**lemma** *minus-divide-left*: $-\ (a\ /\ b) = (-a)\ /\ b$
⟨*proof*⟩

**lemma** *nonzero-minus-divide-right*: $b \neq 0 \Longrightarrow -\ (a\ /\ b) = a\ /\ (-\ b)$
⟨*proof*⟩

**lemma** *nonzero-minus-divide-divide*: $b \neq 0 \Longrightarrow (-a) \ / \ (-b) = a \ / \ b$
⟨*proof*⟩

**lemma** *divide-minus-left* [*simp*]: $(-a) \ / \ b = - \ (a \ / \ b)$
⟨*proof*⟩

**lemma** *diff-divide-distrib*: $(a - b) \ / \ c = a \ / \ c - b \ / \ c$
⟨*proof*⟩

**lemma** *nonzero-eq-divide-eq* [*field-simps*]: $c \neq 0 \Longrightarrow a = b \ / \ c \longleftrightarrow a * c = b$
⟨*proof*⟩

**lemma** *nonzero-divide-eq-eq* [*field-simps*]: $c \neq 0 \Longrightarrow b \ / \ c = a \longleftrightarrow b = a * c$
⟨*proof*⟩

**lemma** *nonzero-neg-divide-eq-eq* [*field-simps*]: $b \neq 0 \Longrightarrow - \ (a \ / \ b) = c \longleftrightarrow - \ a = c * b$
⟨*proof*⟩

**lemma** *nonzero-neg-divide-eq-eq2* [*field-simps*]: $b \neq 0 \Longrightarrow c = - \ (a \ / \ b) \longleftrightarrow c * b = - \ a$
⟨*proof*⟩

**lemma** *divide-eq-imp*: $c \neq 0 \Longrightarrow b = a * c \Longrightarrow b \ / \ c = a$
⟨*proof*⟩

**lemma** *eq-divide-imp*: $c \neq 0 \Longrightarrow a * c = b \Longrightarrow a = b \ / \ c$
⟨*proof*⟩

**lemma** *add-divide-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow x + y \ / \ z = (x * z + y) \ / \ z$
⟨*proof*⟩

**lemma** *divide-add-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow x \ / \ z + y = (x + y * z) \ / \ z$
⟨*proof*⟩

**lemma** *diff-divide-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow x - y \ / \ z = (x * z - y) \ / \ z$
⟨*proof*⟩

**lemma** *minus-divide-add-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow - \ (x \ / \ z) + y = (- \ x + y * z) \ / \ z$
⟨*proof*⟩

**lemma** *divide-diff-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow x \ / \ z - y = (x - y * z) \ / \ z$
⟨*proof*⟩

**lemma** *minus-divide-diff-eq-iff* [*field-simps*]:
$z \neq 0 \Longrightarrow -(x / z) - y = (-x - y * z) / z$
⟨*proof*⟩

**lemma** *division-ring-divide-zero*:
$a / 0 = 0$
⟨*proof*⟩

**lemma** *divide-self-if* [*simp*]:
$a / a = (\mathit{if}\ a = 0\ \mathit{then}\ 0\ \mathit{else}\ 1)$
⟨*proof*⟩

**lemma** *inverse-nonzero-iff-nonzero* [*simp*]:
$\mathit{inverse}\ a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *inverse-minus-eq* [*simp*]:
$\mathit{inverse}\ (-\ a) = -\ \mathit{inverse}\ a$
⟨*proof*⟩

**lemma** *inverse-inverse-eq* [*simp*]:
$\mathit{inverse}\ (\mathit{inverse}\ a) = a$
⟨*proof*⟩

**lemma** *inverse-eq-imp-eq*:
$\mathit{inverse}\ a = \mathit{inverse}\ b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *inverse-eq-iff-eq* [*simp*]:
$\mathit{inverse}\ a = \mathit{inverse}\ b \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *mult-commute-imp-mult-inverse-commute*:
**assumes** $y * x = x * y$
**shows** $\mathit{inverse}\ y * x = x * \mathit{inverse}\ y$
⟨*proof*⟩

**lemmas** *mult-inverse-of-nat-commute* =
*mult-commute-imp-mult-inverse-commute*[*OF mult-of-nat-commute*]

**lemma** *divide-divide-eq-left′*:
$(a / b) / c = a / (c * b)$
⟨*proof*⟩

**lemma** *add-divide-eq-if-simps* [*field-split-simps*, *divide-simps*]:
$a + b / z = (\mathit{if}\ z = 0\ \mathit{then}\ a\ \mathit{else}\ (a * z + b) / z)$
$a / z + b = (\mathit{if}\ z = 0\ \mathit{then}\ b\ \mathit{else}\ (a + b * z) / z)$
$-(a / z) + b = (\mathit{if}\ z = 0\ \mathit{then}\ b\ \mathit{else}\ (-a + b * z) / z)$

$a - b / z = (\textit{if } z = 0 \textit{ then } a \textit{ else } (a * z - b) / z)$
$a / z - b = (\textit{if } z = 0 \textit{ then } -b \textit{ else } (a - b * z) / z)$
$- (a / z) - b = (\textit{if } z = 0 \textit{ then } -b \textit{ else } (- a - b * z) / z)$
⟨*proof*⟩

**lemma** [*field-split-simps*, *divide-simps*]:
**shows** *divide-eq-eq*: $b / c = a \longleftrightarrow (\textit{if } c \neq 0 \textit{ then } b = a * c \textit{ else } a = 0)$
**and** *eq-divide-eq*: $a = b / c \longleftrightarrow (\textit{if } c \neq 0 \textit{ then } a * c = b \textit{ else } a = 0)$
**and** *minus-divide-eq-eq*: $- (b / c) = a \longleftrightarrow (\textit{if } c \neq 0 \textit{ then } - b = a * c \textit{ else } a = 0)$
**and** *eq-minus-divide-eq*: $a = - (b / c) \longleftrightarrow (\textit{if } c \neq 0 \textit{ then } a * c = - b \textit{ else } a = 0)$
⟨*proof*⟩

**end**

## 18.2 Fields

**class** *field* = *comm-ring-1* + *inverse* +
**assumes** *field-inverse*: $a \neq 0 \Longrightarrow \textit{inverse } a * a = 1$
**assumes** *field-divide-inverse*: $a / b = a * \textit{inverse } b$
**assumes** *field-inverse-zero*: $\textit{inverse } 0 = 0$
**begin**

**subclass** *division-ring*
⟨*proof*⟩

**subclass** *idom-divide*
⟨*proof*⟩

There is no slick version using division by zero.

**lemma** *inverse-add*:
$a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow \textit{inverse } a + \textit{inverse } b = (a + b) * \textit{inverse } a * \textit{inverse } b$
⟨*proof*⟩

**lemma** *nonzero-mult-divide-mult-cancel-left* [*simp*]:
**assumes** [*simp*]: $c \neq 0$
**shows** $(c * a) / (c * b) = a / b$
⟨*proof*⟩

**lemma** *nonzero-mult-divide-mult-cancel-right* [*simp*]:
$c \neq 0 \Longrightarrow (a * c) / (b * c) = a / b$
⟨*proof*⟩

**lemma** *times-divide-eq-left* [*simp*]: $(b / c) * a = (b * a) / c$
⟨*proof*⟩

**lemma** *divide-inverse-commute*: $a / b = \textit{inverse } b * a$
⟨*proof*⟩

**lemma** *add-frac-eq*:
**assumes** $y \neq 0$ **and** $z \neq 0$
**shows** $x / y + w / z = (x * z + w * y) / (y * z)$
⟨*proof*⟩

Special Cancellation Simprules for Division

**lemma** *nonzero-divide-mult-cancel-right* [*simp*]:
$b \neq 0 \Longrightarrow b / (a * b) = 1 / a$
⟨*proof*⟩

**lemma** *nonzero-divide-mult-cancel-left* [*simp*]:
$a \neq 0 \Longrightarrow a / (a * b) = 1 / b$
⟨*proof*⟩

**lemma** *nonzero-mult-divide-mult-cancel-left2* [*simp*]:
$c \neq 0 \Longrightarrow (c * a) / (b * c) = a / b$
⟨*proof*⟩

**lemma** *nonzero-mult-divide-mult-cancel-right2* [*simp*]:
$c \neq 0 \Longrightarrow (a * c) / (c * b) = a / b$
⟨*proof*⟩

**lemma** *diff-frac-eq*:
$y \neq 0 \Longrightarrow z \neq 0 \Longrightarrow x / y - w / z = (x * z - w * y) / (y * z)$
⟨*proof*⟩

**lemma** *frac-eq-eq*:
$y \neq 0 \Longrightarrow z \neq 0 \Longrightarrow (x / y = w / z) = (x * z = w * y)$
⟨*proof*⟩

**lemma** *divide-minus1* [*simp*]: $x / - 1 = - x$
⟨*proof*⟩

This version builds in division by zero while also re-orienting the right-hand side.

**lemma** *inverse-mult-distrib* [*simp*]:
*inverse* $(a * b) =$ *inverse* $a *$ *inverse* $b$
⟨*proof*⟩

**lemma** *inverse-divide* [*simp*]:
*inverse* $(a / b) = b / a$
⟨*proof*⟩

Calculations with fractions

There is a whole bunch of simp-rules just for class *field* but none for class *field* and *nonzero-divides* because the latter are covered by a simproc.

**lemmas** *mult-divide-mult-cancel-left* = *nonzero-mult-divide-mult-cancel-left*

**lemmas** *mult-divide-mult-cancel-right* = *nonzero-mult-divide-mult-cancel-right*

**lemma** *divide-divide-eq-right* [*simp*]:
$a\ /\ (b\ /\ c) = (a * c)\ /\ b$
⟨*proof*⟩

**lemma** *divide-divide-eq-left* [*simp*]:
$(a\ /\ b)\ /\ c = a\ /\ (b * c)$
⟨*proof*⟩

**lemma** *divide-divide-times-eq*:
$(x\ /\ y)\ /\ (z\ /\ w) = (x * w)\ /\ (y * z)$
⟨*proof*⟩

Special Cancellation Simprules for Division

**lemma** *mult-divide-mult-cancel-left-if* [*simp*]:
**shows** $(c * a)\ /\ (c * b) = (\mathit{if}\ c = 0\ \mathit{then}\ 0\ \mathit{else}\ a\ /\ b)$
⟨*proof*⟩

Division and Unary Minus

**lemma** *minus-divide-right*:
$-\ (a\ /\ b) = a\ /\ -\ b$
⟨*proof*⟩

**lemma** *divide-minus-right* [*simp*]:
$a\ /\ -\ b = -\ (a\ /\ b)$
⟨*proof*⟩

**lemma** *minus-divide-divide*:
$(-\ a)\ /\ (-\ b) = a\ /\ b$
⟨*proof*⟩

**lemma** *inverse-eq-1-iff* [*simp*]:
$\mathit{inverse}\ x = 1 \longleftrightarrow x = 1$
⟨*proof*⟩

**lemma** *divide-eq-0-iff* [*simp*]:
$a\ /\ b = 0 \longleftrightarrow a = 0 \lor b = 0$
⟨*proof*⟩

**lemma** *divide-cancel-right* [*simp*]:
$a\ /\ c = b\ /\ c \longleftrightarrow c = 0 \lor a = b$
⟨*proof*⟩

**lemma** *divide-cancel-left* [*simp*]:
$c\ /\ a = c\ /\ b \longleftrightarrow c = 0 \lor a = b$
⟨*proof*⟩

**lemma** *divide-eq-1-iff* [*simp*]:
$a\ /\ b = 1 \longleftrightarrow b \neq 0 \wedge a = b$
⟨*proof*⟩

**lemma** *one-eq-divide-iff* [*simp*]:
$1 = a\ /\ b \longleftrightarrow b \neq 0 \wedge a = b$
⟨*proof*⟩

**lemma** *divide-eq-minus-1-iff*:
$(a\ /\ b = -\ 1) \longleftrightarrow b \neq 0 \wedge a = -\ b$
⟨*proof*⟩

**lemma** *times-divide-times-eq*:
$(x\ /\ y) * (z\ /\ w) = (x * z)\ /\ (y * w)$
⟨*proof*⟩

**lemma** *add-frac-num*:
$y \neq 0 \Longrightarrow x\ /\ y + z = (x + z * y)\ /\ y$
⟨*proof*⟩

**lemma** *add-num-frac*:
$y \neq 0 \Longrightarrow z + x\ /\ y = (x + z * y)\ /\ y$
⟨*proof*⟩

**lemma** *dvd-field-iff*:
$a\ dvd\ b \longleftrightarrow (a = 0 \longrightarrow b = 0)$
⟨*proof*⟩

**lemma** *inj-divide-right* [*simp*]:
$inj\ (\lambda b.\ b\ /\ a) \longleftrightarrow a \neq 0$
⟨*proof*⟩

**end**

**class** *field-char-0* = *field* + *ring-char-0*

## 18.3 Ordered fields

**class** *field-abs-sgn* = *field* + *idom-abs-sgn*
**begin**

**lemma** *sgn-inverse* [*simp*]:
$sgn\ (inverse\ a) = inverse\ (sgn\ a)$
⟨*proof*⟩

**lemma** *abs-inverse* [*simp*]:
$|inverse\ a| = inverse\ |a|$
⟨*proof*⟩

**lemma** *sgn-divide* [*simp*]:
  $sgn\ (a\ /\ b) = sgn\ a\ /\ sgn\ b$
  ⟨*proof*⟩

**lemma** *abs-divide* [*simp*]:
  $|a\ /\ b| = |a|\ /\ |b|$
  ⟨*proof*⟩

**end**

**class** *linordered-field* = *field* + *linordered-idom*
**begin**

**lemma** *positive-imp-inverse-positive*:
  **assumes** *a-gt-0*: $0 < a$
  **shows** $0 < inverse\ a$
⟨*proof*⟩

**lemma** *negative-imp-inverse-negative*:
  $a < 0 \Longrightarrow inverse\ a < 0$
  ⟨*proof*⟩

**lemma** *inverse-le-imp-le*:
  **assumes** *invle*: $inverse\ a \leq inverse\ b$ **and** *apos*: $0 < a$
  **shows** $b \leq a$
⟨*proof*⟩

**lemma** *inverse-positive-imp-positive*:
  **assumes** *inv-gt-0*: $0 < inverse\ a$ **and** *nz*: $a \neq 0$
  **shows** $0 < a$
⟨*proof*⟩

**lemma** *inverse-negative-imp-negative*:
  **assumes** *inv-less-0*: $inverse\ a < 0$ **and** *nz*: $a \neq 0$
  **shows** $a < 0$
⟨*proof*⟩

**lemma** *linordered-field-no-lb*:
  $\forall x.\ \exists y.\ y < x$
⟨*proof*⟩

**lemma** *linordered-field-no-ub*:
  $\forall\ x.\ \exists y.\ y > x$
⟨*proof*⟩

**lemma** *less-imp-inverse-less*:
  **assumes** *less*: $a < b$ **and** *apos*: $0 < a$
  **shows** $inverse\ b < inverse\ a$
⟨*proof*⟩

**lemma** *inverse-less-imp-less*:
**assumes** $inverse\ a < inverse\ b$ $0 < a$
**shows** $b < a$
⟨*proof*⟩

Both premises are essential. Consider -1 and 1.

**lemma** *inverse-less-iff-less* [*simp*]:
$0 < a \Longrightarrow 0 < b \Longrightarrow inverse\ a < inverse\ b \longleftrightarrow b < a$
⟨*proof*⟩

**lemma** *le-imp-inverse-le*:
$a \le b \Longrightarrow 0 < a \Longrightarrow inverse\ b \le inverse\ a$
⟨*proof*⟩

**lemma** *inverse-le-iff-le* [*simp*]:
$0 < a \Longrightarrow 0 < b \Longrightarrow inverse\ a \le inverse\ b \longleftrightarrow b \le a$
⟨*proof*⟩

These results refer to both operands being negative. The opposite-sign case is trivial, since inverse preserves signs.

**lemma** *inverse-le-imp-le-neg*:
**assumes** $inverse\ a \le inverse\ b$ $b < 0$
**shows** $b \le a$
⟨*proof*⟩

**lemma** *less-imp-inverse-less-neg*:
**assumes** $a < b$ $b < 0$
**shows** $inverse\ b < inverse\ a$
⟨*proof*⟩

**lemma** *inverse-less-imp-less-neg*:
**assumes** $inverse\ a < inverse\ b$ $b < 0$
**shows** $b < a$
⟨*proof*⟩

**lemma** *inverse-less-iff-less-neg* [*simp*]:
$a < 0 \Longrightarrow b < 0 \Longrightarrow inverse\ a < inverse\ b \longleftrightarrow b < a$
⟨*proof*⟩

**lemma** *le-imp-inverse-le-neg*:
$a \le b \Longrightarrow b < 0 \Longrightarrow inverse\ b \le inverse\ a$
⟨*proof*⟩

**lemma** *inverse-le-iff-le-neg* [*simp*]:
$a < 0 \Longrightarrow b < 0 \Longrightarrow inverse\ a \le inverse\ b \longleftrightarrow b \le a$
⟨*proof*⟩

**lemma** *one-less-inverse*:

$0 < a \Longrightarrow a < 1 \Longrightarrow 1 < inverse\ a$
⟨*proof*⟩

**lemma** *one-le-inverse*:
$0 < a \Longrightarrow a \leq 1 \Longrightarrow 1 \leq inverse\ a$
⟨*proof*⟩

**lemma** *pos-le-divide-eq* [*field-simps*]:
**assumes** $0 < c$
**shows** $a \leq b\ /\ c \longleftrightarrow a * c \leq b$
⟨*proof*⟩

**lemma** *pos-less-divide-eq* [*field-simps*]:
**assumes** $0 < c$
**shows** $a < b\ /\ c \longleftrightarrow a * c < b$
⟨*proof*⟩

**lemma** *neg-less-divide-eq* [*field-simps*]:
**assumes** $c < 0$
**shows** $a < b\ /\ c \longleftrightarrow b < a * c$
⟨*proof*⟩

**lemma** *neg-le-divide-eq* [*field-simps*]:
**assumes** $c < 0$
**shows** $a \leq b\ /\ c \longleftrightarrow b \leq a * c$
⟨*proof*⟩

**lemma** *pos-divide-le-eq* [*field-simps*]:
**assumes** $0 < c$
**shows** $b\ /\ c \leq a \longleftrightarrow b \leq a * c$
⟨*proof*⟩

**lemma** *pos-divide-less-eq* [*field-simps*]:
**assumes** $0 < c$
**shows** $b\ /\ c < a \longleftrightarrow b < a * c$
⟨*proof*⟩

**lemma** *neg-divide-le-eq* [*field-simps*]:
**assumes** $c < 0$
**shows** $b\ /\ c \leq a \longleftrightarrow a * c \leq b$
⟨*proof*⟩

**lemma** *neg-divide-less-eq* [*field-simps*]:
**assumes** $c < 0$
**shows** $b\ /\ c < a \longleftrightarrow a * c < b$
⟨*proof*⟩

The following *field-simps* rules are necessary, as minus is always moved atop of division but we want to get rid of division.

**lemma** *pos-le-minus-divide-eq* [*field-simps*]: $0 < c \Longrightarrow a \leq - (b / c) \longleftrightarrow a * c \leq - b$
⟨*proof*⟩

**lemma** *neg-le-minus-divide-eq* [*field-simps*]: $c < 0 \Longrightarrow a \leq - (b / c) \longleftrightarrow - b \leq a * c$
⟨*proof*⟩

**lemma** *pos-less-minus-divide-eq* [*field-simps*]: $0 < c \Longrightarrow a < - (b / c) \longleftrightarrow a * c < - b$
⟨*proof*⟩

**lemma** *neg-less-minus-divide-eq* [*field-simps*]: $c < 0 \Longrightarrow a < - (b / c) \longleftrightarrow - b < a * c$
⟨*proof*⟩

**lemma** *pos-minus-divide-less-eq* [*field-simps*]: $0 < c \Longrightarrow - (b / c) < a \longleftrightarrow - b < a * c$
⟨*proof*⟩

**lemma** *neg-minus-divide-less-eq* [*field-simps*]: $c < 0 \Longrightarrow - (b / c) < a \longleftrightarrow a * c < - b$
⟨*proof*⟩

**lemma** *pos-minus-divide-le-eq* [*field-simps*]: $0 < c \Longrightarrow - (b / c) \leq a \longleftrightarrow - b \leq a * c$
⟨*proof*⟩

**lemma** *neg-minus-divide-le-eq* [*field-simps*]: $c < 0 \Longrightarrow - (b / c) \leq a \longleftrightarrow a * c \leq - b$
⟨*proof*⟩

**lemma** *frac-less-eq*:
$y \neq 0 \Longrightarrow z \neq 0 \Longrightarrow x / y < w / z \longleftrightarrow (x * z - w * y) / (y * z) < 0$
⟨*proof*⟩

**lemma** *frac-le-eq*:
$y \neq 0 \Longrightarrow z \neq 0 \Longrightarrow x / y \leq w / z \longleftrightarrow (x * z - w * y) / (y * z) \leq 0$
⟨*proof*⟩

**lemma** *divide-pos-pos*[*simp*]:
$0 < x \Longrightarrow 0 < y \Longrightarrow 0 < x / y$
⟨*proof*⟩

**lemma** *divide-nonneg-pos*:
$0 \leq x \Longrightarrow 0 < y \Longrightarrow 0 \leq x / y$
⟨*proof*⟩

**lemma** *divide-neg-pos*:

$x < 0 \Longrightarrow 0 < y \Longrightarrow x \ / \ y < 0$
⟨proof⟩

**lemma** *divide-nonpos-pos*:
$x \leq 0 \Longrightarrow 0 < y \Longrightarrow x \ / \ y \leq 0$
⟨proof⟩

**lemma** *divide-pos-neg*:
$0 < x \Longrightarrow y < 0 \Longrightarrow x \ / \ y < 0$
⟨proof⟩

**lemma** *divide-nonneg-neg*:
$0 \leq x \Longrightarrow y < 0 \Longrightarrow x \ / \ y \leq 0$
⟨proof⟩

**lemma** *divide-neg-neg*:
$x < 0 \Longrightarrow y < 0 \Longrightarrow 0 < x \ / \ y$
⟨proof⟩

**lemma** *divide-nonpos-neg*:
$x \leq 0 \Longrightarrow y < 0 \Longrightarrow 0 \leq x \ / \ y$
⟨proof⟩

**lemma** *divide-strict-right-mono*:
$[\![a < b;\ 0 < c]\!] \Longrightarrow a \ / \ c < b \ / \ c$
⟨proof⟩

**lemma** *divide-strict-right-mono-neg*:
**assumes** $b < a$ $c < 0$ **shows** $a \ / \ c < b \ / \ c$
⟨proof⟩

The last premise ensures that $a$ and $b$ have the same sign

**lemma** *divide-strict-left-mono*:
$[\![b < a;\ 0 < c;\ 0 < a*b]\!] \Longrightarrow c \ / \ a < c \ / \ b$
⟨proof⟩

**lemma** *divide-left-mono*:
$[\![b \leq a;\ 0 \leq c;\ 0 < a*b]\!] \Longrightarrow c \ / \ a \leq c \ / \ b$
⟨proof⟩

**lemma** *divide-strict-left-mono-neg*:
$[\![a < b;\ c < 0;\ 0 < a*b]\!] \Longrightarrow c \ / \ a < c \ / \ b$
⟨proof⟩

**lemma** *mult-imp-div-pos-le*: $0 < y \Longrightarrow x \leq z * y \Longrightarrow x \ / \ y \leq z$
⟨proof⟩

**lemma** *mult-imp-le-div-pos*: $0 < y \Longrightarrow z * y \leq x \Longrightarrow z \leq x \ / \ y$

⟨*proof*⟩

**lemma** *mult-imp-div-pos-less*: $0 < y \Longrightarrow x < z * y \Longrightarrow x / y < z$
⟨*proof*⟩

**lemma** *mult-imp-less-div-pos*: $0 < y \Longrightarrow z * y < x \Longrightarrow z < x / y$
⟨*proof*⟩

**lemma** *frac-le*:
**assumes** $0 \leq y$ $x \leq y$ $0 < w$ $w \leq z$
**shows** $x / z \leq y / w$
⟨*proof*⟩

**lemma** *frac-less*:
**assumes** $0 \leq x$ $x < y$ $0 < w$ $w \leq z$
**shows** $x / z < y / w$
⟨*proof*⟩

**lemma** *frac-less2*:
**assumes** $0 < x$ $x \leq y$ $0 < w$ $w < z$
**shows** $x / z < y / w$
⟨*proof*⟩

As above, with a better name

**lemma** *divide-mono*:
$[\![b \leq a;\ c \leq d;\ 0 < b;\ 0 \leq c]\!] \Longrightarrow c / a \leq d / b$
⟨*proof*⟩

**lemma** *less-half-sum*: $a < b \Longrightarrow a < (a+b) / (1+1)$
⟨*proof*⟩

**lemma** *gt-half-sum*: $a < b \Longrightarrow (a+b)/(1+1) < b$
⟨*proof*⟩

**subclass** *unbounded-dense-linorder*
⟨*proof*⟩

**subclass** *field-abs-sgn* ⟨*proof*⟩

**lemma** *inverse-sgn* [*simp*]:
*inverse* (*sgn a*) = *sgn a*
⟨*proof*⟩

**lemma** *divide-sgn* [*simp*]:
$a$ / *sgn b* = $a$ * *sgn b*
⟨*proof*⟩

**lemma** *nonzero-abs-inverse*:
$a \neq 0 \Longrightarrow |inverse\ a| = inverse\ |a|$

⟨*proof*⟩

**lemma** *nonzero-abs-divide*:
$b \neq 0 \Longrightarrow |a\ /\ b| = |a|\ /\ |b|$
⟨*proof*⟩

**lemma** *field-le-epsilon*:
**assumes** *e*: $\bigwedge e.\ 0 < e \Longrightarrow x \leq y + e$
**shows** $x \leq y$
⟨*proof*⟩

**lemma** *inverse-positive-iff-positive* [*simp*]: $(0 < \mathit{inverse}\ a) = (0 < a)$
⟨*proof*⟩

**lemma** *inverse-negative-iff-negative* [*simp*]: $(\mathit{inverse}\ a < 0) = (a < 0)$
⟨*proof*⟩

**lemma** *inverse-nonnegative-iff-nonnegative* [*simp*]: $0 \leq \mathit{inverse}\ a \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *inverse-nonpositive-iff-nonpositive* [*simp*]: $\mathit{inverse}\ a \leq 0 \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *one-less-inverse-iff*: $1 < \mathit{inverse}\ x \longleftrightarrow 0 < x \wedge x < 1$
⟨*proof*⟩

**lemma** *one-le-inverse-iff*: $1 \leq \mathit{inverse}\ x \longleftrightarrow 0 < x \wedge x \leq 1$
⟨*proof*⟩

**lemma** *inverse-less-1-iff*: $\mathit{inverse}\ x < 1 \longleftrightarrow x \leq 0 \vee 1 < x$
⟨*proof*⟩

**lemma** *inverse-le-1-iff*: $\mathit{inverse}\ x \leq 1 \longleftrightarrow x \leq 0 \vee 1 \leq x$
⟨*proof*⟩

**lemma** [*field-split-simps*, *divide-simps*]:
**shows** *le-divide-eq*: $a \leq b\ /\ c \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ a * c \leq b\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ b \leq a * c\ \mathit{else}\ a \leq 0)$
**and** *divide-le-eq*: $b\ /\ c \leq a \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ b \leq a * c\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ a * c \leq b\ \mathit{else}\ 0 \leq a)$
**and** *less-divide-eq*: $a < b\ /\ c \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ a * c < b\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ b < a * c\ \mathit{else}\ a < 0)$
**and** *divide-less-eq*: $b\ /\ c < a \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ b < a * c\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ a * c < b\ \mathit{else}\ 0 < a)$
**and** *le-minus-divide-eq*: $a \leq -\ (b\ /\ c) \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ a * c \leq -\ b\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ -\ b \leq a * c\ \mathit{else}\ a \leq 0)$
**and** *minus-divide-le-eq*: $-\ (b\ /\ c) \leq a \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ -\ b \leq a * c\ \mathit{else}\ \mathit{if}\ c < 0\ \mathit{then}\ a * c \leq -\ b\ \mathit{else}\ 0 \leq a)$
**and** *less-minus-divide-eq*: $a < -\ (b\ /\ c) \longleftrightarrow (\mathit{if}\ 0 < c\ \mathit{then}\ a * c < -\ b\ \mathit{else}$

*if* $c < 0$ *then* $- b < a * c$ *else* $a < 0$)
**and** *minus-divide-less-eq*: $- (b / c) < a \longleftrightarrow$ (*if* $0 < c$ *then* $- b < a * c$ *else* *if* $c < 0$ *then* $a * c < - b$ *else* $0 < a$)
⟨*proof*⟩

Division and Signs

**lemma**
**shows** *zero-less-divide-iff*: $0 < a / b \longleftrightarrow 0 < a \wedge 0 < b \vee a < 0 \wedge b < 0$
**and** *divide-less-0-iff*: $a / b < 0 \longleftrightarrow 0 < a \wedge b < 0 \vee a < 0 \wedge 0 < b$
**and** *zero-le-divide-iff*: $0 \leq a / b \longleftrightarrow 0 \leq a \wedge 0 \leq b \vee a \leq 0 \wedge b \leq 0$
**and** *divide-le-0-iff*: $a / b \leq 0 \longleftrightarrow 0 \leq a \wedge b \leq 0 \vee a \leq 0 \wedge 0 \leq b$
⟨*proof*⟩

Division and the Number One

Simplify expressions equated with 1

**lemma** *zero-eq-1-divide-iff* [*simp*]: $0 = 1 / a \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *one-divide-eq-0-iff* [*simp*]: $1 / a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

Simplify expressions such as $0 < 1/x$ to $0 < x$

**lemma** *zero-le-divide-1-iff* [*simp*]:
$0 \leq 1 / a \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *zero-less-divide-1-iff* [*simp*]:
$0 < 1 / a \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *divide-le-0-1-iff* [*simp*]:
$1 / a \leq 0 \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *divide-less-0-1-iff* [*simp*]:
$1 / a < 0 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *divide-right-mono*:
$[\![a \leq b; 0 \leq c]\!] \Longrightarrow a/c \leq b/c$
⟨*proof*⟩

**lemma** *divide-right-mono-neg*: $a \leq b \Longrightarrow c \leq 0 \Longrightarrow b / c \leq a / c$
⟨*proof*⟩

**lemma** *divide-left-mono-neg*: $a \leq b \Longrightarrow c \leq 0 \Longrightarrow 0 < a * b \Longrightarrow c / a \leq c / b$
⟨*proof*⟩

**lemma** *inverse-le-iff*: $inverse\ a \leq inverse\ b \longleftrightarrow (0 < a * b \longrightarrow b \leq a) \wedge (a * b \leq 0 \longrightarrow a \leq b)$
⟨*proof*⟩

**lemma** *inverse-less-iff*: $inverse\ a < inverse\ b \longleftrightarrow (0 < a * b \longrightarrow b < a) \wedge (a * b \leq 0 \longrightarrow a < b)$
⟨*proof*⟩

**lemma** *divide-le-cancel*: $a\ /\ c \leq b\ /\ c \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
⟨*proof*⟩

**lemma** *divide-less-cancel*: $a\ /\ c < b\ /\ c \longleftrightarrow (0 < c \longrightarrow a < b) \wedge (c < 0 \longrightarrow b < a) \wedge c \neq 0$
⟨*proof*⟩

Simplify quotients that are compared with the value 1.

**lemma** *le-divide-eq-1*:
$(1 \leq b\ /\ a) = ((0 < a \wedge a \leq b) \vee (a < 0 \wedge b \leq a))$
⟨*proof*⟩

**lemma** *divide-le-eq-1*:
$(b\ /\ a \leq 1) = ((0 < a \wedge b \leq a) \vee (a < 0 \wedge a \leq b) \vee a=0)$
⟨*proof*⟩

**lemma** *less-divide-eq-1*:
$(1 < b\ /\ a) = ((0 < a \wedge a < b) \vee (a < 0 \wedge b < a))$
⟨*proof*⟩

**lemma** *divide-less-eq-1*:
$(b\ /\ a < 1) = ((0 < a \wedge b < a) \vee (a < 0 \wedge a < b) \vee a=0)$
⟨*proof*⟩

**lemma** *divide-nonneg-nonneg* [*simp*]:
$0 \leq x \Longrightarrow 0 \leq y \Longrightarrow 0 \leq x\ /\ y$
⟨*proof*⟩

**lemma** *divide-nonpos-nonpos*:
$x \leq 0 \Longrightarrow y \leq 0 \Longrightarrow 0 \leq x\ /\ y$
⟨*proof*⟩

**lemma** *divide-nonneg-nonpos*:
$0 \leq x \Longrightarrow y \leq 0 \Longrightarrow x\ /\ y \leq 0$
⟨*proof*⟩

**lemma** *divide-nonpos-nonneg*:
$x \leq 0 \Longrightarrow 0 \leq y \Longrightarrow x\ /\ y \leq 0$
⟨*proof*⟩

Conditional Simplification Rules: No Case Splits

**lemma** *le-divide-eq-1-pos* [*simp*]:
$0 < a \Longrightarrow (1 \le b/a) = (a \le b)$
⟨*proof*⟩

**lemma** *le-divide-eq-1-neg* [*simp*]:
$a < 0 \Longrightarrow (1 \le b/a) = (b \le a)$
⟨*proof*⟩

**lemma** *divide-le-eq-1-pos* [*simp*]:
$0 < a \Longrightarrow (b/a \le 1) = (b \le a)$
⟨*proof*⟩

**lemma** *divide-le-eq-1-neg* [*simp*]:
$a < 0 \Longrightarrow (b/a \le 1) = (a \le b)$
⟨*proof*⟩

**lemma** *less-divide-eq-1-pos* [*simp*]:
$0 < a \Longrightarrow (1 < b/a) = (a < b)$
⟨*proof*⟩

**lemma** *less-divide-eq-1-neg* [*simp*]:
$a < 0 \Longrightarrow (1 < b/a) = (b < a)$
⟨*proof*⟩

**lemma** *divide-less-eq-1-pos* [*simp*]:
$0 < a \Longrightarrow (b/a < 1) = (b < a)$
⟨*proof*⟩

**lemma** *divide-less-eq-1-neg* [*simp*]:
$a < 0 \Longrightarrow b/a < 1 \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *eq-divide-eq-1* [*simp*]:
$(1 = b/a) = ((a \neq 0 \wedge a = b))$
⟨*proof*⟩

**lemma** *divide-eq-eq-1* [*simp*]:
$(b/a = 1) = ((a \neq 0 \wedge a = b))$
⟨*proof*⟩

**lemma** *abs-div-pos*: $0 < y \Longrightarrow |x| \mathbin{/} y = |x \mathbin{/} y|$
⟨*proof*⟩

**lemma** *zero-le-divide-abs-iff* [*simp*]: $(0 \le a \mathbin{/} |b|) = (0 \le a \vee b = 0)$
⟨*proof*⟩

**lemma** *divide-le-0-abs-iff* [*simp*]: $(a \mathbin{/} |b| \le 0) = (a \le 0 \vee b = 0)$
⟨*proof*⟩

**lemma** *field-le-mult-one-interval*:
  **assumes** $*$: $\bigwedge z$. $\llbracket\ 0 < z\ ;\ z < 1\ \rrbracket \Longrightarrow z * x \leq y$
  **shows** $x \leq y$
$\langle proof \rangle$

For creating values between $u$ and $v$.

**lemma** *scaling-mono*:
  **assumes** $u \leq v\ 0 \leq r\ r \leq s$
  **shows** $u + r * (v - u) / s \leq v$
$\langle proof \rangle$

**end**

Min/max Simplification Rules

**lemma** *min-mult-distrib-left*:
  **fixes** $x::'a::linordered\text{-}idom$
  **shows** $p * min\ x\ y = (if\ 0 \leq p\ then\ min\ (p{*}x)\ (p{*}y)\ else\ max\ (p{*}x)\ (p{*}y))$
$\langle proof \rangle$

**lemma** *min-mult-distrib-right*:
  **fixes** $x::'a::linordered\text{-}idom$
  **shows** $min\ x\ y * p = (if\ 0 \leq p\ then\ min\ (x{*}p)\ (y{*}p)\ else\ max\ (x{*}p)\ (y{*}p))$
$\langle proof \rangle$

**lemma** *min-divide-distrib-right*:
  **fixes** $x::'a::linordered\text{-}field$
  **shows** $min\ x\ y\ /\ p = (if\ 0 \leq p\ then\ min\ (x/p)\ (y/p)\ else\ max\ (x/p)\ (y/p))$
$\langle proof \rangle$

**lemma** *max-mult-distrib-left*:
  **fixes** $x::'a::linordered\text{-}idom$
  **shows** $p * max\ x\ y = (if\ 0 \leq p\ then\ max\ (p{*}x)\ (p{*}y)\ else\ min\ (p{*}x)\ (p{*}y))$
$\langle proof \rangle$

**lemma** *max-mult-distrib-right*:
  **fixes** $x::'a::linordered\text{-}idom$
  **shows** $max\ x\ y * p = (if\ 0 \leq p\ then\ max\ (x{*}p)\ (y{*}p)\ else\ min\ (x{*}p)\ (y{*}p))$
$\langle proof \rangle$

**lemma** *max-divide-distrib-right*:
  **fixes** $x::'a::linordered\text{-}field$
  **shows** $max\ x\ y\ /\ p = (if\ 0 \leq p\ then\ max\ (x/p)\ (y/p)\ else\ min\ (x/p)\ (y/p))$
$\langle proof \rangle$

**hide-fact** (**open**) *field-inverse field-divide-inverse field-inverse-zero*

**code-identifier**
  **code-module** *Fields* $\rightharpoonup$ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 19 Relations – as sets of pairs, and binary predicates

**theory** *Relation*
**imports** *Product-Type Sum-Type Fields*
**begin**

A preliminary: classical rules for reasoning on predicates

**declare** *predicate1I* [*Pure.intro*!, *intro*!]
**declare** *predicate1D* [*Pure.dest*, *dest*]
**declare** *predicate2I* [*Pure.intro*!, *intro*!]
**declare** *predicate2D* [*Pure.dest*, *dest*]
**declare** *bot1E* [*elim*!]
**declare** *bot2E* [*elim*!]
**declare** *top1I* [*intro*!]
**declare** *top2I* [*intro*!]
**declare** *inf1I* [*intro*!]
**declare** *inf2I* [*intro*!]
**declare** *inf1E* [*elim*!]
**declare** *inf2E* [*elim*!]
**declare** *sup1I1* [*intro?*]
**declare** *sup2I1* [*intro?*]
**declare** *sup1I2* [*intro?*]
**declare** *sup2I2* [*intro?*]
**declare** *sup1E* [*elim*!]
**declare** *sup2E* [*elim*!]
**declare** *sup1CI* [*intro*!]
**declare** *sup2CI* [*intro*!]
**declare** *Inf1-I* [*intro*!]
**declare** *INF1-I* [*intro*!]
**declare** *Inf2-I* [*intro*!]
**declare** *INF2-I* [*intro*!]
**declare** *Inf1-D* [*elim*]
**declare** *INF1-D* [*elim*]
**declare** *Inf2-D* [*elim*]
**declare** *INF2-D* [*elim*]
**declare** *Inf1-E* [*elim*]
**declare** *INF1-E* [*elim*]
**declare** *Inf2-E* [*elim*]
**declare** *INF2-E* [*elim*]
**declare** *Sup1-I* [*intro*]
**declare** *SUP1-I* [*intro*]
**declare** *Sup2-I* [*intro*]
**declare** *SUP2-I* [*intro*]
**declare** *Sup1-E* [*elim*!]
**declare** *SUP1-E* [*elim*!]

**declare** *Sup2-E* [*elim*!]
**declare** *SUP2-E* [*elim*!]

## 19.1 Fundamental

### 19.1.1 Relations as sets of pairs

**type-synonym** $'a\ rel = ('a \times 'a)\ set$

**lemma** *subrelI*: $(\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow (x,\ y) \in s) \Longrightarrow r \subseteq s$
— Version of *subsetI* for binary relations
⟨*proof*⟩

**lemma** *lfp-induct2*:
$(a,\ b) \in lfp\ f \Longrightarrow mono\ f \Longrightarrow$
$(\bigwedge a\ b.\ (a,\ b) \in f\ (lfp\ f \cap \{(x,\ y).\ P\ x\ y\}) \Longrightarrow P\ a\ b) \Longrightarrow P\ a\ b$
— Version of *lfp-induct* for binary relations
⟨*proof*⟩

### 19.1.2 Conversions between set and predicate relations

**lemma** *pred-equals-eq* [*pred-set-conv*]: $(\lambda x.\ x \in R) = (\lambda x.\ x \in S) \longleftrightarrow R = S$
⟨*proof*⟩

**lemma** *pred-equals-eq2* [*pred-set-conv*]: $(\lambda x\ y.\ (x,\ y) \in R) = (\lambda x\ y.\ (x,\ y) \in S) \longleftrightarrow R = S$
⟨*proof*⟩

**lemma** *pred-subset-eq* [*pred-set-conv*]: $(\lambda x.\ x \in R) \le (\lambda x.\ x \in S) \longleftrightarrow R \subseteq S$
⟨*proof*⟩

**lemma** *pred-subset-eq2* [*pred-set-conv*]: $(\lambda x\ y.\ (x,\ y) \in R) \le (\lambda x\ y.\ (x,\ y) \in S) \longleftrightarrow R \subseteq S$
⟨*proof*⟩

**lemma** *bot-empty-eq* [*pred-set-conv*]: $\bot = (\lambda x.\ x \in \{\})$
⟨*proof*⟩

**lemma** *bot-empty-eq2* [*pred-set-conv*]: $\bot = (\lambda x\ y.\ (x,\ y) \in \{\})$
⟨*proof*⟩

**lemma** *top-empty-eq*: $\top = (\lambda x.\ x \in UNIV)$
⟨*proof*⟩

**lemma** *top-empty-eq2*: $\top = (\lambda x\ y.\ (x,\ y) \in UNIV)$
⟨*proof*⟩

**lemma** *inf-Int-eq* [*pred-set-conv*]: $(\lambda x.\ x \in R) \sqcap (\lambda x.\ x \in S) = (\lambda x.\ x \in R \cap S)$
⟨*proof*⟩

**lemma** *inf-Int-eq2* [*pred-set-conv*]: $(\lambda x\ y.\ (x, y) \in R) \sqcap (\lambda x\ y.\ (x, y) \in S) = (\lambda x\ y.\ (x, y) \in R \cap S)$
⟨*proof*⟩

**lemma** *sup-Un-eq* [*pred-set-conv*]: $(\lambda x.\ x \in R) \sqcup (\lambda x.\ x \in S) = (\lambda x.\ x \in R \cup S)$
⟨*proof*⟩

**lemma** *sup-Un-eq2* [*pred-set-conv*]: $(\lambda x\ y.\ (x, y) \in R) \sqcup (\lambda x\ y.\ (x, y) \in S) = (\lambda x\ y.\ (x, y) \in R \cup S)$
⟨*proof*⟩

**lemma** *INF-INT-eq* [*pred-set-conv*]: $(\bigsqcap i{\in}S.\ (\lambda x.\ x \in r\ i)) = (\lambda x.\ x \in (\bigcap i{\in}S.\ r\ i))$
⟨*proof*⟩

**lemma** *INF-INT-eq2* [*pred-set-conv*]: $(\bigsqcap i{\in}S.\ (\lambda x\ y.\ (x, y) \in r\ i)) = (\lambda x\ y.\ (x, y) \in (\bigcap i{\in}S.\ r\ i))$
⟨*proof*⟩

**lemma** *SUP-UN-eq* [*pred-set-conv*]: $(\bigsqcup i{\in}S.\ (\lambda x.\ x \in r\ i)) = (\lambda x.\ x \in (\bigcup i{\in}S.\ r\ i))$
⟨*proof*⟩

**lemma** *SUP-UN-eq2* [*pred-set-conv*]: $(\bigsqcup i{\in}S.\ (\lambda x\ y.\ (x, y) \in r\ i)) = (\lambda x\ y.\ (x, y) \in (\bigcup i{\in}S.\ r\ i))$
⟨*proof*⟩

**lemma** *Inf-INT-eq* [*pred-set-conv*]: $\bigsqcap S = (\lambda x.\ x \in (\bigcap (Collect\ `\ S)))$
⟨*proof*⟩

**lemma** *INF-Int-eq* [*pred-set-conv*]: $(\bigsqcap i{\in}S.\ (\lambda x.\ x \in i)) = (\lambda x.\ x \in \bigcap S)$
⟨*proof*⟩

**lemma** *Inf-INT-eq2* [*pred-set-conv*]: $\bigsqcap S = (\lambda x\ y.\ (x, y) \in (\bigcap (Collect\ `\ case\text{-}prod\ `\ S)))$
⟨*proof*⟩

**lemma** *INF-Int-eq2* [*pred-set-conv*]: $(\bigsqcap i{\in}S.\ (\lambda x\ y.\ (x, y) \in i)) = (\lambda x\ y.\ (x, y) \in \bigcap S)$
⟨*proof*⟩

**lemma** *Sup-SUP-eq* [*pred-set-conv*]: $\bigsqcup S = (\lambda x.\ x \in \bigcup (Collect\ `\ S))$
⟨*proof*⟩

**lemma** *SUP-Sup-eq* [*pred-set-conv*]: $(\bigsqcup i{\in}S.\ (\lambda x.\ x \in i)) = (\lambda x.\ x \in \bigcup S)$
⟨*proof*⟩

**lemma** *Sup-SUP-eq2* [*pred-set-conv*]: $\bigsqcup S = (\lambda x\ y.\ (x, y) \in (\bigcup (Collect\ `\ case\text{-}prod\ `\ S)))$

⟨*proof*⟩

**lemma** *SUP-Sup-eq2* [*pred-set-conv*]: $(\bigsqcup i \in S.\ (\lambda x\ y.\ (x,\ y) \in i)) = (\lambda x\ y.\ (x,\ y) \in \bigcup S)$
⟨*proof*⟩

## 19.2 Properties of relations

### 19.2.1 Reflexivity

**definition** *refl-on* :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow bool$
**where** *refl-on* $A\ r \longleftrightarrow (\forall x \in A.\ (x,\ x) \in r)$

**abbreviation** *refl* :: $'a\ rel \Rightarrow bool$ — reflexivity over a type
**where** *refl* $\equiv$ *refl-on UNIV*

**definition** *reflp-on* :: $'a\ set \Rightarrow ('a \Rightarrow\ 'a \Rightarrow bool) \Rightarrow bool$
**where** *reflp-on* $A\ R \longleftrightarrow (\forall x \in A.\ R\ x\ x)$

**abbreviation** *reflp* :: $('a \Rightarrow\ 'a \Rightarrow bool) \Rightarrow bool$
**where** *reflp* $\equiv$ *reflp-on UNIV*

**lemma** *reflp-def*[*no-atp*]: *reflp* $R \longleftrightarrow (\forall x.\ R\ x\ x)$
⟨*proof*⟩

*reflp-def* is for backward compatibility.

**lemma** *reflp-on-refl-on-eq*[*pred-set-conv*]: *reflp-on* $A\ (\lambda a\ b.\ (a,\ b) \in r) \longleftrightarrow$ *refl-on* $A\ r$
⟨*proof*⟩

**lemmas** *reflp-refl-eq* = *reflp-on-refl-on-eq*[*of UNIV*]

**lemma** *refl-onI* [*intro?*]: $(\bigwedge x.\ x \in A \Longrightarrow (x,\ x) \in r) \Longrightarrow$ *refl-on* $A\ r$
⟨*proof*⟩

**lemma** *reflI*: $(\bigwedge x.\ (x,\ x) \in r) \Longrightarrow$ *refl* $r$
⟨*proof*⟩

**lemma** *reflp-onI*:
$(\bigwedge x.\ x \in A \Longrightarrow R\ x\ x) \Longrightarrow$ *reflp-on* $A\ R$
⟨*proof*⟩

**lemma** *reflpI*[*intro?*]: $(\bigwedge x.\ R\ x\ x) \Longrightarrow$ *reflp* $R$
⟨*proof*⟩

**lemma** *refl-onD*: *refl-on* $A\ r \Longrightarrow a \in A \Longrightarrow (a,\ a) \in r$
⟨*proof*⟩

**lemma** *reflD*: *refl* $r \Longrightarrow (a,\ a) \in r$
⟨*proof*⟩

**lemma** *reflp-onD*:
*reflp-on* $A$ $R \Longrightarrow x \in A \Longrightarrow R\ x\ x$
⟨*proof*⟩

**lemma** *reflpD*[*dest?*]: *reflp* $R \Longrightarrow R\ x\ x$
⟨*proof*⟩

**lemma** *reflpE*:
**assumes** *reflp* $r$
**obtains** $r\ x\ x$
⟨*proof*⟩

**lemma** *refl-on-top*[*simp*]: *refl-on* $A$ $\top$
⟨*proof*⟩

**lemma** *reflp-on-top*[*simp*]: *reflp-on* $A$ $\top$
⟨*proof*⟩

**lemma** *reflp-on-mono-strong*:
*reflp-on* $B$ $R \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow$ *reflp-on* $A$ $Q$
⟨*proof*⟩

**lemma** *reflp-on-mono*[*mono*]: $A \subseteq B \Longrightarrow R \leq Q \Longrightarrow$ *reflp-on* $B$ $R \leq$ *reflp-on* $A$ $Q$
⟨*proof*⟩

**lemma** *reflp-on-subset*: *reflp-on* $B$ $R \Longrightarrow A \subseteq B \Longrightarrow$ *reflp-on* $A$ $R$
⟨*proof*⟩

**lemma** *reflp-on-image*: *reflp-on* $(f\ `\ A)$ $R \longleftrightarrow$ *reflp-on* $A$ $(\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *refl-on-Int*: *refl-on* $A$ $r \Longrightarrow$ *refl-on* $B$ $s \Longrightarrow$ *refl-on* $(A \cap B)$ $(r \cap s)$
⟨*proof*⟩

**lemma** *reflp-on-inf*: *reflp-on* $A$ $R \Longrightarrow$ *reflp-on* $B$ $S \Longrightarrow$ *reflp-on* $(A \cap B)$ $(R \sqcap S)$
⟨*proof*⟩

**lemma** *reflp-inf*: *reflp* $r \Longrightarrow$ *reflp* $s \Longrightarrow$ *reflp* $(r \sqcap s)$
⟨*proof*⟩

**lemma** *refl-on-Un*: *refl-on* $A$ $r \Longrightarrow$ *refl-on* $B$ $s \Longrightarrow$ *refl-on* $(A \cup B)$ $(r \cup s)$
⟨*proof*⟩

**lemma** *reflp-on-sup*: *reflp-on* $A$ $R \Longrightarrow$ *reflp-on* $B$ $S \Longrightarrow$ *reflp-on* $(A \cup B)$ $(R \sqcup S)$
⟨*proof*⟩

**lemma** *reflp-sup*: $reflp\ r \Longrightarrow reflp\ s \Longrightarrow reflp\ (r \sqcup s)$
⟨*proof*⟩

**lemma** *refl-on-INTER*: $\forall x{\in}S.\ refl\text{-}on\ (A\ x)\ (r\ x) \Longrightarrow refl\text{-}on\ (\bigcap(A\ `\ S))\ (\bigcap(r\ `\ S))$
⟨*proof*⟩

**lemma** *reflp-on-Inf*: $\forall x{\in}S.\ reflp\text{-}on\ (A\ x)\ (R\ x) \Longrightarrow reflp\text{-}on\ (\bigcap(A\ `\ S))\ (\prod(R\ `\ S))$
⟨*proof*⟩

**lemma** *refl-on-UNION*: $\forall x{\in}S.\ refl\text{-}on\ (A\ x)\ (r\ x) \Longrightarrow refl\text{-}on\ (\bigcup(A\ `\ S))\ (\bigcup(r\ `\ S))$
⟨*proof*⟩

**lemma** *reflp-on-Sup*: $\forall x{\in}S.\ reflp\text{-}on\ (A\ x)\ (R\ x) \Longrightarrow reflp\text{-}on\ (\bigcup(A\ `\ S))\ (\bigsqcup(R\ `\ S))$
⟨*proof*⟩

**lemma** *refl-on-empty* [*simp*]: *refl-on* {} *r*
⟨*proof*⟩

**lemma** *reflp-on-empty* [*simp*]: *reflp-on* {} *R*
⟨*proof*⟩

**lemma** *refl-on-singleton* [*simp*]: *refl-on* {*x*} {(*x*, *x*)}
⟨*proof*⟩

**lemma** *reflp-on-equality* [*simp*]: *reflp-on A* (=)
⟨*proof*⟩

**lemma** (**in** *preorder*) *reflp-on-le*[*simp*]: *reflp-on A* (≤)
⟨*proof*⟩

**lemma** (**in** *preorder*) *reflp-on-ge*[*simp*]: *reflp-on A* (≥)
⟨*proof*⟩

### 19.2.2 Irreflexivity

**definition** *irrefl-on* :: *'a set* ⇒ *'a rel* ⇒ *bool* **where**
$irrefl\text{-}on\ A\ r \longleftrightarrow (\forall a \in A.\ (a, a) \notin r)$

**abbreviation** *irrefl* :: *'a rel* ⇒ *bool* **where**
*irrefl* ≡ *irrefl-on UNIV*

**definition** *irreflp-on* :: *'a set* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* **where**
$irreflp\text{-}on\ A\ R \longleftrightarrow (\forall a \in A.\ \neg\ R\ a\ a)$

**abbreviation** *irreflp* :: (*'a* ⇒ *'a* ⇒ *bool*) ⇒ *bool* **where**

*irreflp* ≡ *irreflp-on UNIV*

**lemma** *irrefl-def*[*no-atp*]: *irrefl r* $\longleftrightarrow$ ($\forall$ *a*. (*a*, *a*) $\notin$ *r*)
⟨*proof*⟩

**lemma** *irreflp-def*[*no-atp*]: *irreflp R* $\longleftrightarrow$ ($\forall$ *a*. $\neg$ *R a a*)
⟨*proof*⟩

*irrefl-def* and *irreflp-def* are for backward compatibility.

**lemma** *irreflp-on-irrefl-on-eq* [*pred-set-conv*]: *irreflp-on A* ($\lambda$*a b*. (*a*, *b*) $\in$ *r*) $\longleftrightarrow$ *irrefl-on A r*
⟨*proof*⟩

**lemmas** *irreflp-irrefl-eq* = *irreflp-on-irrefl-on-eq*[*of UNIV*]

**lemma** *irrefl-onI*: ($\bigwedge$*a*. *a* $\in$ *A* $\Longrightarrow$ (*a*, *a*) $\notin$ *r*) $\Longrightarrow$ *irrefl-on A r*
⟨*proof*⟩

**lemma** *irreflI*[*intro?*]: ($\bigwedge$*a*. (*a*, *a*) $\notin$ *r*) $\Longrightarrow$ *irrefl r*
⟨*proof*⟩

**lemma** *irreflp-onI*: ($\bigwedge$*a*. *a* $\in$ *A* $\Longrightarrow$ $\neg$ *R a a*) $\Longrightarrow$ *irreflp-on A R*
⟨*proof*⟩

**lemma** *irreflpI*[*intro?*]: ($\bigwedge$*a*. $\neg$ *R a a*) $\Longrightarrow$ *irreflp R*
⟨*proof*⟩

**lemma** *irrefl-onD*: *irrefl-on A r* $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ (*a*, *a*) $\notin$ *r*
⟨*proof*⟩

**lemma** *irreflD*: *irrefl r* $\Longrightarrow$ (*x*, *x*) $\notin$ *r*
⟨*proof*⟩

**lemma** *irreflp-onD*: *irreflp-on A R* $\Longrightarrow$ *a* $\in$ *A* $\Longrightarrow$ $\neg$ *R a a*
⟨*proof*⟩

**lemma** *irreflpD*: *irreflp R* $\Longrightarrow$ $\neg$ *R x x*
⟨*proof*⟩

**lemma** *irrefl-on-bot*[*simp*]: *irrefl-on A* $\bot$
⟨*proof*⟩

**lemma** *irreflp-on-bot*[*simp*]: *irreflp-on A* $\bot$
⟨*proof*⟩

**lemma** *irrefl-on-distinct* [*code*]: *irrefl-on A r* $\longleftrightarrow$ ($\forall$ (*a*, *b*) $\in$ *r*. *a* $\in$ *A* $\longrightarrow$ *b* $\in$ *A* $\longrightarrow$ *a* $\neq$ *b*)
⟨*proof*⟩

**lemmas** *irrefl-distinct* = *irrefl-on-distinct* — For backward compatibility

**lemma** *irreflp-on-mono-strong*:
$irreflp\text{-}on\ B\ Q \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow irreflp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *irreflp-on-mono*[*mono*]: $A \subseteq B \Longrightarrow R \leq Q \Longrightarrow irreflp\text{-}on\ B\ Q \leq irreflp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *irrefl-on-subset*: $irrefl\text{-}on\ B\ r \Longrightarrow A \subseteq B \Longrightarrow irrefl\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *irreflp-on-subset*: $irreflp\text{-}on\ B\ R \Longrightarrow A \subseteq B \Longrightarrow irreflp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *irreflp-on-image*: $irreflp\text{-}on\ (f\ `\ A)\ R \longleftrightarrow irreflp\text{-}on\ A\ (\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** (**in** *preorder*) *irreflp-on-less*[*simp*]: $irreflp\text{-}on\ A\ (<)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *irreflp-on-greater*[*simp*]: $irreflp\text{-}on\ A\ (>)$
⟨*proof*⟩

### 19.2.3 Asymmetry

**definition** *asym-on* :: $'a\ set \Rightarrow 'a\ rel \Rightarrow bool$ **where**
$asym\text{-}on\ A\ r \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ (x,\ y) \in r \longrightarrow (y,\ x) \notin r)$

**abbreviation** *asym* :: $'a\ rel \Rightarrow bool$ **where**
$asym \equiv asym\text{-}on\ UNIV$

**definition** *asymp-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$asymp\text{-}on\ A\ R \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ R\ x\ y \longrightarrow \neg\ R\ y\ x)$

**abbreviation** *asymp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$asymp \equiv asymp\text{-}on\ UNIV$

**lemma** *asymp-on-asym-on-eq*[*pred-set-conv*]: $asymp\text{-}on\ A\ (\lambda x\ y.\ (x,\ y) \in r) \longleftrightarrow asym\text{-}on\ A\ r$
⟨*proof*⟩

**lemmas** *asymp-asym-eq* = *asymp-on-asym-on-eq*[*of UNIV*] — For backward compatibility

**lemma** *asym-onI*[*intro*]:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \notin r) \Longrightarrow asym\text{-}on\ A\ r$

⟨*proof*⟩

**lemma** *asymI*[*intro*]: $(\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow (y,\ x) \notin r) \Longrightarrow asym\ r$
⟨*proof*⟩

**lemma** *asymp-onI*[*intro*]:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow \neg\ R\ y\ x) \Longrightarrow asymp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *asympI*[*intro*]: $(\bigwedge x\ y.\ R\ x\ y \Longrightarrow \neg\ R\ y\ x) \Longrightarrow asymp\ R$
⟨*proof*⟩

**lemma** *asym-onD*: $asym\text{-}on\ A\ r \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \notin r$
⟨*proof*⟩

**lemma** *asymD*: $asym\ r \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \notin r$
⟨*proof*⟩

**lemma** *asymp-onD*: $asymp\text{-}on\ A\ R \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow \neg\ R\ y\ x$
⟨*proof*⟩

**lemma** *asympD*: $asymp\ R \Longrightarrow R\ x\ y \Longrightarrow \neg\ R\ y\ x$
⟨*proof*⟩

**lemma** *asym-on-bot*[*simp*]: $asym\text{-}on\ A\ \bot$
⟨*proof*⟩

**lemma** *asymp-on-bot*[*simp*]: $asymp\text{-}on\ A\ \bot$
⟨*proof*⟩

**lemma** *asym-iff*: $asym\ r \longleftrightarrow (\forall\ x\ y.\ (x,y) \in r \longrightarrow (y,x) \notin r)$
⟨*proof*⟩

**lemma** *asymp-on-mono-strong*:
$asymp\text{-}on\ B\ Q \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow asymp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *asymp-on-mono*[*mono*]: $A \subseteq B \Longrightarrow R \leq Q \Longrightarrow asymp\text{-}on\ B\ Q \leq asymp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *asym-on-subset*: $asym\text{-}on\ B\ r \Longrightarrow A \subseteq B \Longrightarrow asym\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *asymp-on-subset*: $asymp\text{-}on\ B\ R \Longrightarrow A \subseteq B \Longrightarrow asymp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *asymp-on-image*: $asymp\text{-}on\ (f\ `\ A)\ R \longleftrightarrow asymp\text{-}on\ A\ (\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *irrefl-on-if-asym-on*[*simp*]: $asym\text{-}on\ A\ r \Longrightarrow irrefl\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *irreflp-on-if-asymp-on*[*simp*]: $asymp\text{-}on\ A\ r \Longrightarrow irreflp\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** (**in** *preorder*) *asymp-on-less*[*simp*]: $asymp\text{-}on\ A\ (<)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *asymp-on-greater*[*simp*]: $asymp\text{-}on\ A\ (>)$
⟨*proof*⟩

### 19.2.4 Symmetry

**definition** *sym-on* :: $'a\ set \Rightarrow 'a\ rel \Rightarrow bool$ **where**
$sym\text{-}on\ A\ r \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ (x, y) \in r \longrightarrow (y, x) \in r)$

**abbreviation** *sym* :: $'a\ rel \Rightarrow bool$ **where**
$sym \equiv sym\text{-}on\ UNIV$

**definition** *symp-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$symp\text{-}on\ A\ R \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ R\ x\ y \longrightarrow R\ y\ x)$

**abbreviation** *symp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$symp \equiv symp\text{-}on\ UNIV$

**lemma** *sym-def*[*no-atp*]: $sym\ r \longleftrightarrow (\forall x\ y.\ (x, y) \in r \longrightarrow (y, x) \in r)$
⟨*proof*⟩

**lemma** *symp-def*[*no-atp*]: $symp\ R \longleftrightarrow (\forall x\ y.\ R\ x\ y \longrightarrow R\ y\ x)$
⟨*proof*⟩

*sym-def* and *symp-def* are for backward compatibility.

**lemma** *symp-on-sym-on-eq*[*pred-set-conv*]: $symp\text{-}on\ A\ (\lambda x\ y.\ (x, y) \in r) \longleftrightarrow sym\text{-}on\ A\ r$
⟨*proof*⟩

**lemmas** *symp-sym-eq* = *symp-on-sym-on-eq*[*of UNIV*] — For backward compatibility

**lemma** *sym-onI*: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x, y) \in r \Longrightarrow (y, x) \in r) \Longrightarrow sym\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *symI* [*intro?*]: $(\bigwedge x\ y.\ (x, y) \in r \Longrightarrow (y, x) \in r) \Longrightarrow sym\ r$

⟨proof⟩

**lemma** *symp-onI*: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x) \Longrightarrow$ *symp-on* $A$ $R$
⟨proof⟩

**lemma** *sympI* [*intro?*]: $(\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x) \Longrightarrow$ *symp* $R$
⟨proof⟩

**lemma** *symE*:
**assumes** *sym* $r$ **and** $(b,\ a) \in r$
**obtains** $(a,\ b) \in r$
⟨proof⟩

**lemma** *sympE*:
**assumes** *symp* $r$ **and** $r\ b\ a$
**obtains** $r\ a\ b$
⟨proof⟩

**lemma** *sym-onD*: *sym-on* $A\ r \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \in r$
⟨proof⟩

**lemma** *symD* [*dest?*]: *sym* $r \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \in r$
⟨proof⟩

**lemma** *symp-onD*: *symp-on* $A\ R \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$
⟨proof⟩

**lemma** *sympD* [*dest?*]: *symp* $R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$
⟨proof⟩

**lemma** *sym-on-bot*[*simp*]: *sym-on* $A\ \bot$
⟨proof⟩

**lemma** *symp-on-bot*[*simp*]: *symp-on* $A\ \bot$
⟨proof⟩

**lemma** *sym-on-top*[*simp*]: *sym-on* $A\ \top$
⟨proof⟩

**lemma** *symp-on-top*[*simp*]: *symp-on* $A\ \top$
⟨proof⟩

**lemma** *sym-on-subset*: *sym-on* $B\ r \Longrightarrow A \subseteq B \Longrightarrow$ *sym-on* $A\ r$
⟨proof⟩

**lemma** *symp-on-subset*: *symp-on* $B\ R \Longrightarrow A \subseteq B \Longrightarrow$ *symp-on* $A\ R$
⟨proof⟩

**lemma** *symp-on-image*: $symp\text{-}on\ (f\ `\ A)\ R \longleftrightarrow symp\text{-}on\ A\ (\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *symp-on-equality*[*simp*]: $symp\text{-}on\ A\ (=)$
⟨*proof*⟩

**lemma** *sym-Int*: $sym\ r \Longrightarrow sym\ s \Longrightarrow sym\ (r \cap s)$
⟨*proof*⟩

**lemma** *symp-inf*: $symp\ r \Longrightarrow symp\ s \Longrightarrow symp\ (r \sqcap s)$
⟨*proof*⟩

**lemma** *sym-Un*: $sym\ r \Longrightarrow sym\ s \Longrightarrow sym\ (r \cup s)$
⟨*proof*⟩

**lemma** *symp-sup*: $symp\ r \Longrightarrow symp\ s \Longrightarrow symp\ (r \sqcup s)$
⟨*proof*⟩

**lemma** *sym-INTER*: $\forall\, x{\in}S.\ sym\ (r\ x) \Longrightarrow sym\ (\bigcap (r\ `\ S))$
⟨*proof*⟩

**lemma** *symp-INF*: $\forall\, x{\in}S.\ symp\ (r\ x) \Longrightarrow symp\ (\bigsqcap (r\ `\ S))$
⟨*proof*⟩

**lemma** *sym-UNION*: $\forall\, x{\in}S.\ sym\ (r\ x) \Longrightarrow sym\ (\bigcup (r\ `\ S))$
⟨*proof*⟩

**lemma** *symp-SUP*: $\forall\, x{\in}S.\ symp\ (r\ x) \Longrightarrow symp\ (\bigsqcup (r\ `\ S))$
⟨*proof*⟩

### 19.2.5 Antisymmetry

**definition** *antisym-on* :: $'a\ set \Rightarrow\ 'a\ rel \Rightarrow bool$ **where**
$antisym\text{-}on\ A\ r \longleftrightarrow (\forall\, x \in A.\ \forall\, y \in A.\ (x,\ y) \in r \longrightarrow (y,\ x) \in r \longrightarrow x = y)$

**abbreviation** *antisym* :: $'a\ rel \Rightarrow bool$ **where**
$antisym \equiv antisym\text{-}on\ UNIV$

**definition** *antisymp-on* :: $'a\ set \Rightarrow ('a \Rightarrow\ 'a \Rightarrow bool) \Rightarrow bool$ **where**
$antisymp\text{-}on\ A\ R \longleftrightarrow (\forall\, x \in A.\ \forall\, y \in A.\ R\ x\ y \longrightarrow R\ y\ x \longrightarrow x = y)$

**abbreviation** *antisymp* :: $('a \Rightarrow\ 'a \Rightarrow bool) \Rightarrow bool$ **where**
$antisymp \equiv antisymp\text{-}on\ UNIV$

**lemma** *antisym-def*[*no-atp*]: $antisym\ r \longleftrightarrow (\forall\, x\ y.\ (x,\ y) \in r \longrightarrow (y,\ x) \in r \longrightarrow x = y)$
⟨*proof*⟩

**lemma** *antisymp-def*[*no-atp*]: $antisymp\ R \longleftrightarrow (\forall\, x\ y.\ R\ x\ y \longrightarrow R\ y\ x \longrightarrow x =$

$y)$
⟨proof⟩

*antisym-def* and *antisymp-def* are for backward compatibility.

**lemma** *antisymp-on-antisym-on-eq*[*pred-set-conv*]:
$antisymp\text{-}on\ A\ (\lambda x\ y.\ (x,\ y) \in r) \longleftrightarrow antisym\text{-}on\ A\ r$
⟨proof⟩

**lemmas** *antisymp-antisym-eq* = *antisymp-on-antisym-on-eq*[*of UNIV*] — For backward compatibility

**lemma** *antisym-onI*:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \in r \Longrightarrow x = y) \Longrightarrow antisym\text{-}on\ A\ r$
⟨proof⟩

**lemma** *antisymI* [*intro?*]:
$(\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow (y,\ x) \in r \Longrightarrow x = y) \Longrightarrow antisym\ r$
⟨proof⟩

**lemma** *antisymp-onI*:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y) \Longrightarrow antisymp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *antisympI* [*intro?*]:
$(\bigwedge x\ y.\ R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y) \Longrightarrow antisymp\ R$
⟨proof⟩

**lemma** *antisym-onD*:
$antisym\text{-}on\ A\ r \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \in r \Longrightarrow x = y$
⟨proof⟩

**lemma** *antisymD* [*dest?*]:
$antisym\ r \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ x) \in r \Longrightarrow x = y$
⟨proof⟩

**lemma** *antisymp-onD*:
$antisymp\text{-}on\ A\ R \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
⟨proof⟩

**lemma** *antisympD* [*dest?*]:
$antisymp\ R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x \Longrightarrow x = y$
⟨proof⟩

**lemma** *antisym-on-bot*[*simp*]: $antisym\text{-}on\ A\ \bot$
⟨proof⟩

**lemma** *antisymp-on-bot*[*simp*]: $antisymp\text{-}on\ A\ \bot$
⟨proof⟩

**lemma** *antisymp-on-mono-stronger*:
**fixes**
$A$ :: $'a$ *set* **and** $R$ :: $'a \Rightarrow 'a \Rightarrow$ *bool* **and**
$B$ :: $'b$ *set* **and** $Q$ :: $'b \Rightarrow 'b \Rightarrow$ *bool* **and**
$f$ :: $'a \Rightarrow 'b$
**assumes** *antisymp-on* $B$ $Q$ **and** $f$ ‘ $A \subseteq B$ **and**
*Q-imp-R*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ (f\ x)\ (f\ y)$ **and**
*inj-f*: *inj-on* $f$ $A$
**shows** *antisymp-on* $A$ $R$
⟨*proof*⟩

**lemma** *antisymp-on-mono-strong*:
*antisymp-on* $B$ $Q \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow$ *antisymp-on* $A$ $R$
⟨*proof*⟩

**lemma** *antisymp-on-mono*[*mono*]: $A \subseteq B \Longrightarrow R \leq Q \Longrightarrow$ *antisymp-on* $B$ $Q \leq$ *antisymp-on* $A$ $R$
⟨*proof*⟩

**lemma** *antisym-on-subset*: *antisym-on* $B$ $r \Longrightarrow A \subseteq B \Longrightarrow$ *antisym-on* $A$ $r$
⟨*proof*⟩

**lemma** *antisymp-on-subset*: *antisymp-on* $B$ $R \Longrightarrow A \subseteq B \Longrightarrow$ *antisymp-on* $A$ $R$
⟨*proof*⟩

**lemma** *antisymp-on-image*:
**assumes** *inj-on* $f$ $A$
**shows** *antisymp-on* ($f$ ‘ $A$) $R \longleftrightarrow$ *antisymp-on* $A$ $(\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *antisym-subset*:
$r \subseteq s \Longrightarrow$ *antisym* $s \Longrightarrow$ *antisym* $r$
⟨*proof*⟩

**lemma** *antisymp-less-eq*:
$r \leq s \Longrightarrow$ *antisymp* $s \Longrightarrow$ *antisymp* $r$
⟨*proof*⟩

**lemma** *antisymp-on-equality*[*simp*]: *antisymp-on* $A$ $(=)$
⟨*proof*⟩

**lemma** *antisym-singleton* [*simp*]:
*antisym* $\{x\}$
⟨*proof*⟩

**lemma** *antisym-on-if-asym-on*: *asym-on* $A$ $r \Longrightarrow$ *antisym-on* $A$ $r$
⟨*proof*⟩

**lemma** *antisymp-on-if-asymp-on*: $asymp\text{-}on\ A\ R \Longrightarrow antisymp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** (**in** *preorder*) *antisymp-on-less*[*simp*]: $antisymp\text{-}on\ A\ (<)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *antisymp-on-greater*[*simp*]: $antisymp\text{-}on\ A\ (>)$
⟨*proof*⟩

**lemma** (**in** *order*) *antisymp-on-le*[*simp*]: $antisymp\text{-}on\ A\ (\leq)$
⟨*proof*⟩

**lemma** (**in** *order*) *antisymp-on-ge*[*simp*]: $antisymp\text{-}on\ A\ (\geq)$
⟨*proof*⟩

### 19.2.6 Transitivity

**definition** *trans-on* :: $'a\ set \Rightarrow 'a\ rel \Rightarrow bool$ **where**
$trans\text{-}on\ A\ r \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ (x,\ y) \in r \longrightarrow (y,\ z) \in r \longrightarrow (x,\ z) \in r)$

**abbreviation** *trans* :: $'a\ rel \Rightarrow bool$ **where**
$trans \equiv trans\text{-}on\ UNIV$

**definition** *transp-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$transp\text{-}on\ A\ R \longleftrightarrow (\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ R\ x\ y \longrightarrow R\ y\ z \longrightarrow R\ x\ z)$

**abbreviation** *transp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$transp \equiv transp\text{-}on\ UNIV$

**lemma** *trans-def*[*no-atp*]: $trans\ r \longleftrightarrow (\forall x\ y\ z.\ (x,\ y) \in r \longrightarrow (y,\ z) \in r \longrightarrow (x,\ z) \in r)$
⟨*proof*⟩

**lemma** *transp-def*: $transp\ R \longleftrightarrow (\forall x\ y\ z.\ R\ x\ y \longrightarrow R\ y\ z \longrightarrow R\ x\ z)$
⟨*proof*⟩

*trans-def* and *transp-def* are for backward compatibility.

**lemma** *transp-on-trans-on-eq*[*pred-set-conv*]: $transp\text{-}on\ A\ (\lambda x\ y.\ (x,\ y) \in r) \longleftrightarrow trans\text{-}on\ A\ r$
⟨*proof*⟩

**lemmas** *transp-trans-eq* = *transp-on-trans-on-eq*[*of UNIV*] — For backward compatibility

**lemma** *trans-onI*:
$(\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ z) \in r \Longrightarrow (x,\ z) \in r) \Longrightarrow$

*trans-on A r*
⟨*proof*⟩

**lemma** *transI* [*intro?*]: $(\bigwedge x\ y\ z.\ (x,\ y) \in r \Longrightarrow (y,\ z) \in r \Longrightarrow (x,\ z) \in r) \Longrightarrow$ *trans r*
⟨*proof*⟩

**lemma** *transp-onI*:
$(\bigwedge x\ y\ z.\ x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z) \Longrightarrow$ *transp-on A R*
⟨*proof*⟩

**lemma** *transpI* [*intro?*]: $(\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z) \Longrightarrow$ *transp R*
⟨*proof*⟩

**lemma** *transE*:
**assumes** *trans r* **and** $(x,\ y) \in r$ **and** $(y,\ z) \in r$
**obtains** $(x,\ z) \in r$
⟨*proof*⟩

**lemma** *transpE*:
**assumes** *transp r* **and** *r x y* **and** *r y z*
**obtains** *r x z*
⟨*proof*⟩

**lemma** *trans-onD*:
*trans-on A r* $\Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ z) \in r \Longrightarrow (x,\ z) \in r$
⟨*proof*⟩

**lemma** *transD*[*dest?*]: *trans r* $\Longrightarrow (x,\ y) \in r \Longrightarrow (y,\ z) \in r \Longrightarrow (x,\ z) \in r$
⟨*proof*⟩

**lemma** *transp-onD*: *transp-on A R* $\Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow z \in A \Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

**lemma** *transpD*[*dest?*]: *transp R* $\Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

**lemma** *trans-on-subset*: *trans-on B r* $\Longrightarrow A \subseteq B \Longrightarrow$ *trans-on A r*
⟨*proof*⟩

**lemma** *transp-on-subset*: *transp-on B R* $\Longrightarrow A \subseteq B \Longrightarrow$ *transp-on A R*
⟨*proof*⟩

**lemma** *transp-on-image*: *transp-on* $(f$ ‘ $A)\ R \longleftrightarrow$ *transp-on A* $(\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *trans-Int*: $trans\ r \Longrightarrow trans\ s \Longrightarrow trans\ (r \cap s)$
⟨*proof*⟩

**lemma** *transp-inf*: $transp\ r \Longrightarrow transp\ s \Longrightarrow transp\ (r \sqcap s)$
⟨*proof*⟩

**lemma** *trans-INTER*: $\forall\, x{\in}S.\ trans\ (r\ x) \Longrightarrow trans\ (\bigcap (r\ `\ S))$
⟨*proof*⟩

**lemma** *transp-INF*: $\forall\, x{\in}S.\ transp\ (r\ x) \Longrightarrow transp\ (\bigsqcap (r\ `\ S))$
⟨*proof*⟩

**lemma** *trans-on-join* [*code*]:
$trans\text{-}on\ A\ r \longleftrightarrow (\forall\, (x,\ y1) \in r.\ x \in A \longrightarrow y1 \in A \longrightarrow$
$(\forall\, (y2,\ z) \in r.\ y1 = y2 \longrightarrow z \in A \longrightarrow (x,\ z) \in r))$
⟨*proof*⟩

**lemma** *trans-join*: $trans\ r \longleftrightarrow (\forall\, (x,\ y1) \in r.\ \forall\, (y2,\ z) \in r.\ y1 = y2 \longrightarrow (x,\ z) \in r)$
⟨*proof*⟩

**lemma** *transp-trans*: $transp\ r \longleftrightarrow trans\ \{(x,\ y).\ r\ x\ y\}$
⟨*proof*⟩

**lemma** *transp-on-equality*[*simp*]: $transp\text{-}on\ A\ (=)$
⟨*proof*⟩

**lemma** *trans-on-bot*[*simp*]: $trans\text{-}on\ A\ \bot$
⟨*proof*⟩

**lemma** *transp-on-bot*[*simp*]: $transp\text{-}on\ A\ \bot$
⟨*proof*⟩

**lemma** *trans-on-top*[*simp*]: $trans\text{-}on\ A\ \top$
⟨*proof*⟩

**lemma** *transp-on-top*[*simp*]: $transp\text{-}on\ A\ \top$
⟨*proof*⟩

**lemma** *transp-empty* [*simp*]: $transp\ (\lambda x\ y.\ False)$
⟨*proof*⟩

**lemma** *trans-singleton* [*simp*]: $trans\ \{(a,\ a)\}$
⟨*proof*⟩

**lemma** *transp-singleton* [*simp*]: $transp\ (\lambda x\ y.\ x = a \land y = a)$
⟨*proof*⟩

**lemma** *asym-on-iff-irrefl-on-if-trans-on*: *trans-on A r* $\Longrightarrow$ *asym-on A r* $\longleftrightarrow$ *irrefl-on A r*
⟨*proof*⟩

**lemma** *asymp-on-iff-irreflp-on-if-transp-on*: *transp-on A R* $\Longrightarrow$ *asymp-on A R* $\longleftrightarrow$ *irreflp-on A R*
⟨*proof*⟩

**lemma** (**in** *preorder*) *transp-on-le*[*simp*]: *transp-on A* $(\leq)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *transp-on-less*[*simp*]: *transp-on A* $(<)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *transp-on-ge*[*simp*]: *transp-on A* $(\geq)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *transp-on-greater*[*simp*]: *transp-on A* $(>)$
⟨*proof*⟩

### 19.2.7 Totality

**definition** *total-on* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *bool* **where**
*total-on A r* $\longleftrightarrow$ $(\forall x{\in}A.\ \forall y{\in}A.\ x \neq y \longrightarrow (x, y) \in r \vee (y, x) \in r)$

**abbreviation** *total* :: *'a rel* $\Rightarrow$ *bool* **where**
*total* $\equiv$ *total-on UNIV*

**definition** *totalp-on* :: *'a set* $\Rightarrow$ (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
*totalp-on A R* $\longleftrightarrow$ $(\forall x \in A.\ \forall y \in A.\ x \neq y \longrightarrow R\ x\ y \vee R\ y\ x)$

**abbreviation** *totalp* :: (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
*totalp* $\equiv$ *totalp-on UNIV*

**lemma** *totalp-on-total-on-eq*[*pred-set-conv*]: *totalp-on A* $(\lambda x\ y.\ (x, y) \in r)$ $\longleftrightarrow$ *total-on A r*
⟨*proof*⟩

**lemma** *total-onI* [*intro?*]:
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \Longrightarrow (x, y) \in r \vee (y, x) \in r) \Longrightarrow$ *total-on A r*
⟨*proof*⟩

**lemma** *totalI*: $(\bigwedge x\ y.\ x \neq y \Longrightarrow (x, y) \in r \vee (y, x) \in r) \Longrightarrow$ *total r*
⟨*proof*⟩

**lemma** *totalp-onI*: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \Longrightarrow R\ x\ y \vee R\ y\ x) \Longrightarrow$ *totalp-on A R*
⟨*proof*⟩

**lemma** *totalpI*: $(\bigwedge x\ y.\ x \neq y \Longrightarrow R\ x\ y \vee R\ y\ x) \Longrightarrow totalp\ R$
⟨*proof*⟩

**lemma** *totalp-onD*:
$totalp\text{-}on\ A\ R \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \Longrightarrow R\ x\ y \vee R\ y\ x$
⟨*proof*⟩

**lemma** *totalpD*: $totalp\ R \Longrightarrow x \neq y \Longrightarrow R\ x\ y \vee R\ y\ x$
⟨*proof*⟩

**lemma** *total-on-top*[*simp*]: $total\text{-}on\ A\ \top$
⟨*proof*⟩

**lemma** *totalp-on-top*[*simp*]: $totalp\text{-}on\ A\ \top$
⟨*proof*⟩

**lemma** *totalp-on-mono-stronger*:
**fixes**
$A :: 'a\ set$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and**
$B :: 'b\ set$ **and** $Q :: 'b \Rightarrow 'b \Rightarrow bool$ **and**
$f :: 'a \Rightarrow 'b$
**assumes** $totalp\text{-}on\ B\ Q$ **and** $f\ `\ A \subseteq B$ **and**
*Q-imp-R*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ (f\ x)\ (f\ y) \Longrightarrow R\ x\ y$ **and**
*inj-f*: $inj\text{-}on\ f\ A$
**shows** $totalp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *totalp-on-mono-stronger-alt*:
**fixes**
$A :: 'a\ set$ **and** $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and**
$B :: 'b\ set$ **and** $Q :: 'b \Rightarrow 'b \Rightarrow bool$ **and**
$f :: 'b \Rightarrow 'a$
**assumes** $totalp\text{-}on\ B\ Q$ **and** $A \subseteq f\ `\ B$ **and**
*Q-imp-R*: $\bigwedge x\ y.\ x \in B \Longrightarrow y \in B \Longrightarrow Q\ x\ y \Longrightarrow R\ (f\ x)\ (f\ y)$
**shows** $totalp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *totalp-on-mono-strong*:
$totalp\text{-}on\ B\ Q \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ x\ y \Longrightarrow R\ x\ y) \Longrightarrow totalp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *totalp-on-mono*[*mono*]: $A \subseteq B \Longrightarrow Q \leq R \Longrightarrow totalp\text{-}on\ B\ Q \leq totalp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *total-on-subset*: $total\text{-}on\ B\ r \Longrightarrow A \subseteq B \Longrightarrow total\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *totalp-on-subset*: $totalp\text{-}on\ B\ R \Longrightarrow A \subseteq B \Longrightarrow totalp\text{-}on\ A\ R$
⟨*proof*⟩

**lemma** *totalp-on-image*:
**assumes** *inj-on f A*
**shows** $totalp\text{-}on\ (f\ `\ A)\ R \longleftrightarrow totalp\text{-}on\ A\ (\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
⟨*proof*⟩

**lemma** *total-on-empty* [*simp*]: $total\text{-}on\ \{\}\ r$
⟨*proof*⟩

**lemma** *totalp-on-empty* [*simp*]: $totalp\text{-}on\ \{\}\ R$
⟨*proof*⟩

**lemma** *total-on-singleton* [*simp*]: $total\text{-}on\ \{x\}\ r$
⟨*proof*⟩

**lemma** *totalp-on-singleton* [*simp*]: $totalp\text{-}on\ \{x\}\ R$
⟨*proof*⟩

**lemma** (**in** *linorder*) *totalp-on-less*[*simp*]: $totalp\text{-}on\ A\ (<)$
⟨*proof*⟩

**lemma** (**in** *linorder*) *totalp-on-greater*[*simp*]: $totalp\text{-}on\ A\ (>)$
⟨*proof*⟩

**lemma** (**in** *linorder*) *totalp-on-le*[*simp*]: $totalp\text{-}on\ A\ (\leq)$
⟨*proof*⟩

**lemma** (**in** *linorder*) *totalp-on-ge*[*simp*]: $totalp\text{-}on\ A\ (\geq)$
⟨*proof*⟩

### 19.2.8 Left uniqueness

**definition** *left-unique* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$ **where**
$left\text{-}unique\ R \longleftrightarrow (\forall\, x\ y\ z.\ R\ x\ z \longrightarrow R\ y\ z \longrightarrow x = y)$

**lemma** *left-uniqueI*: $(\bigwedge x\ y\ z.\ A\ x\ z \Longrightarrow A\ y\ z \Longrightarrow x = y) \Longrightarrow left\text{-}unique\ A$
⟨*proof*⟩

**lemma** *left-uniqueD*: $left\text{-}unique\ A \Longrightarrow A\ x\ z \Longrightarrow A\ y\ z \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *left-unique-iff-Uniq*: $left\text{-}unique\ r \longleftrightarrow (\forall\, y.\ \exists_{\leq 1} x.\ r\ x\ y)$
⟨*proof*⟩

**lemma** *left-unique-bot*[*simp*]: $left\text{-}unique\ \bot$
⟨*proof*⟩

**lemma** *left-unique-mono-strong*:
$left\text{-}unique\ Q \Longrightarrow (\bigwedge x\ y.\ R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow left\text{-}unique\ R$
⟨*proof*⟩

**lemma** *left-unique-mono*[*mono*]: $R \le Q \Longrightarrow left\text{-}unique\ Q \le left\text{-}unique\ R$
⟨*proof*⟩

### 19.2.9 Right uniqueness

**definition** *single-valued* :: $('a \times 'b)\ set \Rightarrow bool$
**where** $single\text{-}valued\ r \longleftrightarrow (\forall\, x\ y.\ (x,\ y) \in r \longrightarrow (\forall\, z.\ (x,\ z) \in r \longrightarrow y = z))$

**definition** *right-unique* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$ **where**
$right\text{-}unique\ R \longleftrightarrow (\forall\, x\ y\ z.\ R\ x\ y \longrightarrow R\ x\ z \longrightarrow y = z)$

**lemma** *right-unique-single-valued-eq* [*pred-set-conv*]:
$right\text{-}unique\ (\lambda x\ y.\ (x,\ y) \in r) \longleftrightarrow single\text{-}valued\ r$
⟨*proof*⟩

**lemma** *right-unique-iff-Uniq*:
$right\text{-}unique\ r \longleftrightarrow (\forall\, x.\ \exists_{\le 1} y.\ r\ x\ y)$
⟨*proof*⟩

**lemma** *single-valuedI*:
$(\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow (\bigwedge z.\ (x,\ z) \in r \Longrightarrow y = z)) \Longrightarrow single\text{-}valued\ r$
⟨*proof*⟩

**lemma** *right-uniqueI*: $(\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ x\ z \Longrightarrow y = z) \Longrightarrow right\text{-}unique\ R$
⟨*proof*⟩

**lemma** *single-valuedD*:
$single\text{-}valued\ r \Longrightarrow (x,\ y) \in r \Longrightarrow (x,\ z) \in r \Longrightarrow y = z$
⟨*proof*⟩

**lemma** *right-uniqueD*: $right\text{-}unique\ R \Longrightarrow R\ x\ y \Longrightarrow R\ x\ z \Longrightarrow y = z$
⟨*proof*⟩

**lemma** *single-valued-empty* [*simp*]:
$single\text{-}valued\ \{\}$
⟨*proof*⟩

**lemma** *right-unique-bot*[*simp*]: $right\text{-}unique\ \bot$
⟨*proof*⟩

**lemma** *right-unique-mono-strong*:
$right\text{-}unique\ Q \Longrightarrow (\bigwedge x\ y.\ R\ x\ y \Longrightarrow Q\ x\ y) \Longrightarrow right\text{-}unique\ R$
⟨*proof*⟩

**lemma** *right-unique-mono*[*mono*]: $R \le Q \Longrightarrow right\text{-}unique\ Q \le right\text{-}unique\ R$

⟨*proof*⟩

**lemma** *single-valued-subset*:
$r \subseteq s \Longrightarrow$ *single-valued* $s \Longrightarrow$ *single-valued* $r$
⟨*proof*⟩

**lemma** *right-unique-less-eq*: $r \leq s \Longrightarrow$ *right-unique* $s \Longrightarrow$ *right-unique* $r$
⟨*proof*⟩

## 19.3 Relation operations

### 19.3.1 The identity relation

**definition** *Id* :: *'a rel*
**where** $Id = \{p.\ \exists x.\ p = (x, x)\}$

**lemma** *IdI* [*intro*]: $(a, a) \in Id$
⟨*proof*⟩

**lemma** *IdE* [*elim*!]: $p \in Id \Longrightarrow (\bigwedge x.\ p = (x, x) \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *pair-in-Id-conv* [*iff*]: $(a, b) \in Id \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *refl-Id*: *refl Id*
⟨*proof*⟩

**lemma** *antisym-Id*: *antisym Id*
— A strange result, since *Id* is also symmetric.
⟨*proof*⟩

**lemma** *sym-Id*: *sym Id*
⟨*proof*⟩

**lemma** *trans-Id*: *trans Id*
⟨*proof*⟩

**lemma** *single-valued-Id* [*simp*]: *single-valued Id*
⟨*proof*⟩

**lemma** *irrefl-diff-Id* [*simp*]: *irrefl* $(r - Id)$
⟨*proof*⟩

**lemma** *trans-on-diff-Id*: *trans-on* $A\ r \Longrightarrow$ *antisym-on* $A\ r \Longrightarrow$ *trans-on* $A\ (r - Id)$
⟨*proof*⟩

**lemma** *trans-diff-Id*[*no-atp*]: *trans* $r \Longrightarrow$ *antisym* $r \Longrightarrow$ *trans* $(r - Id)$
⟨*proof*⟩

**lemma** *total-on-diff-Id* [*simp*]: $total\text{-}on\ A\ (r - Id) = total\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *Id-fstsnd-eq*: $Id = \{x.\ fst\ x = snd\ x\}$
⟨*proof*⟩

### 19.3.2 Diagonal: identity over a set

**definition** *Id-on* :: $'a\ set \Rightarrow 'a\ rel$
**where** $Id\text{-}on\ A = (\bigcup x{\in}A.\ \{(x,\ x)\})$

**lemma** *Id-on-empty* [*simp*]: $Id\text{-}on\ \{\} = \{\}$
⟨*proof*⟩

**lemma** *Id-on-eqI*: $a = b \Longrightarrow a \in A \Longrightarrow (a,\ b) \in Id\text{-}on\ A$
⟨*proof*⟩

**lemma** *Id-onI* [*intro*!]: $a \in A \Longrightarrow (a,\ a) \in Id\text{-}on\ A$
⟨*proof*⟩

**lemma** *Id-onE* [*elim*!]: $c \in Id\text{-}on\ A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow c = (x,\ x) \Longrightarrow P) \Longrightarrow P$
— The general elimination rule.
⟨*proof*⟩

**lemma** *Id-on-iff*: $(x,\ y) \in Id\text{-}on\ A \longleftrightarrow x = y \wedge x \in A$
⟨*proof*⟩

**lemma** *Id-on-def'* [*nitpick-unfold*]: $Id\text{-}on\ \{x.\ A\ x\} = Collect\ (\lambda(x,\ y).\ x = y \wedge A\ x)$
⟨*proof*⟩

**lemma** *Id-on-subset-Times*: $Id\text{-}on\ A \subseteq A \times A$
⟨*proof*⟩

**lemma** *refl-on-Id-on*: $refl\text{-}on\ A\ (Id\text{-}on\ A)$
⟨*proof*⟩

**lemma** *antisym-Id-on* [*simp*]: $antisym\ (Id\text{-}on\ A)$
⟨*proof*⟩

**lemma** *sym-Id-on* [*simp*]: $sym\ (Id\text{-}on\ A)$
⟨*proof*⟩

**lemma** *trans-Id-on* [*simp*]: $trans\ (Id\text{-}on\ A)$
⟨*proof*⟩

**lemma** *single-valued-Id-on* [*simp*]: $single\text{-}valued\ (Id\text{-}on\ A)$
⟨*proof*⟩

### 19.3.3 Composition

**inductive-set** *relcomp* :: $('a \times 'b)$ *set* $\Rightarrow$ $('b \times 'c)$ *set* $\Rightarrow$ $('a \times 'c)$ *set*
**for** $r$ :: $('a \times 'b)$ *set* **and** $s$ :: $('b \times 'c)$ *set*
**where** *relcompI* [*intro*]: $(a, b) \in r \Longrightarrow (b, c) \in s \Longrightarrow (a, c) \in$ *relcomp r s*

**open-bundle** *relcomp-syntax*
**begin**
**notation** *relcomp* (**infixr** ‹*O*› *75*) **and** *relcompp* (**infixr** ‹*OO*› *75*)
**end**

**lemmas** *relcomppI* = *relcompp.intros*

For historic reasons, the elimination rules are not wholly corresponding. Feel free to consolidate this.

**inductive-cases** *relcompEpair*: $(a, c) \in r\ O\ s$
**inductive-cases** *relcomppE* [*elim*!]: $(r\ OO\ s)\ a\ c$

**lemma** *relcompE* [*elim*!]: $xz \in r\ O\ s \Longrightarrow$
$(\bigwedge x\ y\ z.\ xz = (x, z) \Longrightarrow (x, y) \in r \Longrightarrow (y, z) \in s \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *R-O-Id* [*simp*]: $R\ O\ Id = R$
⟨*proof*⟩

**lemma** *Id-O-R* [*simp*]: $Id\ O\ R = R$
⟨*proof*⟩

**lemma** *relcomp-empty1* [*simp*]: $\{\}\ O\ R = \{\}$
⟨*proof*⟩

**lemma** *relcompp-bot1* [*simp*]: $\bot\ OO\ R = \bot$
⟨*proof*⟩

**lemma** *relcomp-empty2* [*simp*]: $R\ O\ \{\} = \{\}$
⟨*proof*⟩

**lemma** *relcompp-bot2* [*simp*]: $R\ OO\ \bot = \bot$
⟨*proof*⟩

**lemma** *O-assoc*: $(R\ O\ S)\ O\ T = R\ O\ (S\ O\ T)$
⟨*proof*⟩

**lemma** *relcompp-assoc*: $(r\ OO\ s)\ OO\ t = r\ OO\ (s\ OO\ t)$
⟨*proof*⟩

**lemma** *trans-O-subset*: *trans* $r \Longrightarrow r\ O\ r \subseteq r$
⟨*proof*⟩

**lemma** *transp-relcompp-less-eq*: *transp* $r \Longrightarrow r\ OO\ r \le r$

⟨*proof*⟩

**lemma** *relcomp-mono*: $r' \subseteq r \Longrightarrow s' \subseteq s \Longrightarrow r'\ O\ s' \subseteq r\ O\ s$
⟨*proof*⟩

**lemma** *relcompp-mono*: $r' \leq r \Longrightarrow s' \leq s \Longrightarrow r'\ OO\ s' \leq r\ OO\ s$
⟨*proof*⟩

**lemma** *relcomp-subset-Sigma*: $r \subseteq A \times B \Longrightarrow s \subseteq B \times C \Longrightarrow r\ O\ s \subseteq A \times C$
⟨*proof*⟩

**lemma** *relcomp-distrib* [*simp*]: $R\ O\ (S \cup T) = (R\ O\ S) \cup (R\ O\ T)$
⟨*proof*⟩

**lemma** *relcompp-distrib* [*simp*]: $R\ OO\ (S \sqcup T) = R\ OO\ S \sqcup R\ OO\ T$
⟨*proof*⟩

**lemma** *relcomp-distrib2* [*simp*]: $(S \cup T)\ O\ R = (S\ O\ R) \cup (T\ O\ R)$
⟨*proof*⟩

**lemma** *relcompp-distrib2* [*simp*]: $(S \sqcup T)\ OO\ R = S\ OO\ R \sqcup T\ OO\ R$
⟨*proof*⟩

**lemma** *relcomp-UNION-distrib*: $s\ O\ \bigcup(r\ `\ I) = (\bigcup i{\in}I.\ s\ O\ r\ i)$
⟨*proof*⟩

**lemma** *relcompp-SUP-distrib*: $s\ OO\ \bigsqcup(r\ `\ I) = (\bigsqcup i{\in}I.\ s\ OO\ r\ i)$
⟨*proof*⟩

**lemma** *relcomp-UNION-distrib2*: $\bigcup(r\ `\ I)\ O\ s = (\bigcup i{\in}I.\ r\ i\ O\ s)$
⟨*proof*⟩

**lemma** *relcompp-SUP-distrib2*: $\bigsqcup(r\ `\ I)\ OO\ s = (\bigsqcup i{\in}I.\ r\ i\ OO\ s)$
⟨*proof*⟩

**lemma** *single-valued-relcomp*: *single-valued* $r \Longrightarrow$ *single-valued* $s \Longrightarrow$ *single-valued* $(r\ O\ s)$
⟨*proof*⟩

**lemma** *relcomp-unfold*: $r\ O\ s = \{(x, z).\ \exists\, y.\ (x, y) \in r \wedge (y, z) \in s\}$
⟨*proof*⟩

**lemma** *relcompp-apply*: $(R\ OO\ S)\ a\ c \longleftrightarrow (\exists\, b.\ R\ a\ b \wedge S\ b\ c)$
⟨*proof*⟩

**lemma** *eq-OO*: $(=)\ OO\ R = R$
⟨*proof*⟩

**lemma** *OO-eq*: $R\ OO\ (=) = R$

⟨*proof*⟩

### 19.3.4 Converse

**inductive-set** *converse* :: $('a \times 'b)$ *set* $\Rightarrow$ $('b \times 'a)$ *set*
**for** $r$ :: $('a \times 'b)$ *set*
**where** $(a, b) \in r \Longrightarrow (b, a) \in$ *converse* $r$

**open-bundle** *converse-syntax*
**begin**
**notation**
*converse* (‹(‹notation=‹postfix −1››-$^{-1}$)› [1000] 999) **and**
*conversep* (‹(‹notation=‹postfix −1−1››-$^{-1-1}$)› [1000] 1000)
**notation** (*ASCII*)
*converse* (‹(‹notation=‹postfix −1››-^−1)› [1000] 999) **and**
*conversep* (‹(‹notation=‹postfix −1−1››-^−−1)› [1000] 1000)
**end**

**lemma** *converseI* [*sym*]: $(a, b) \in r \Longrightarrow (b, a) \in r^{-1}$
⟨*proof*⟩

**lemma** *conversepI* : $r\ a\ b \Longrightarrow r^{-1-1}\ b\ a$
⟨*proof*⟩

**lemma** *converseD* [*sym*]: $(a, b) \in r^{-1} \Longrightarrow (b, a) \in r$
⟨*proof*⟩

**lemma** *conversepD* : $r^{-1-1}\ b\ a \Longrightarrow r\ a\ b$
⟨*proof*⟩

**lemma** *converseE* [*elim*!]: $yx \in r^{-1} \Longrightarrow (\bigwedge x\ y.\ yx = (y, x) \Longrightarrow (x, y) \in r \Longrightarrow P) \Longrightarrow P$
— More general than *converseD*, as it "splits" the member of the relation.
⟨*proof*⟩

**lemmas** *conversepE* [*elim*!] = *conversep.cases*

**lemma** *converse-iff* [*iff*]: $(a, b) \in r^{-1} \longleftrightarrow (b, a) \in r$
⟨*proof*⟩

**lemma** *conversep-iff* [*iff*]: $r^{-1-1}\ a\ b = r\ b\ a$
⟨*proof*⟩

**lemma** *converse-converse* [*simp*]: $(r^{-1})^{-1} = r$
⟨*proof*⟩

**lemma** *conversep-conversep* [*simp*]: $(r^{-1-1})^{-1-1} = r$
⟨*proof*⟩

**lemma** *converse-empty*[*simp*]: $\{\}^{-1} = \{\}$
⟨*proof*⟩

**lemma** *converse-UNIV*[*simp*]: $UNIV^{-1} = UNIV$
⟨*proof*⟩

**lemma** *converse-relcomp*: $(r\ O\ s)^{-1} = s^{-1}\ O\ r^{-1}$
⟨*proof*⟩

**lemma** *converse-relcompp*: $(r\ OO\ s)^{-1-1} = s^{-1-1}\ OO\ r^{-1-1}$
⟨*proof*⟩

**lemma** *converse-Int*: $(r \cap s)^{-1} = r^{-1} \cap s^{-1}$
⟨*proof*⟩

**lemma** *converse-meet*: $(r \sqcap s)^{-1-1} = r^{-1-1} \sqcap s^{-1-1}$
⟨*proof*⟩

**lemma** *converse-Un*: $(r \cup s)^{-1} = r^{-1} \cup s^{-1}$
⟨*proof*⟩

**lemma** *converse-join*: $(r \sqcup s)^{-1-1} = r^{-1-1} \sqcup s^{-1-1}$
⟨*proof*⟩

**lemma** *converse-INTER*: $(\bigcap (r\ `\ S))^{-1} = (\bigcap x{\in}S.\ (r\ x)^{-1})$
⟨*proof*⟩

**lemma** *converse-UNION*: $(\bigcup (r\ `\ S))^{-1} = (\bigcup x{\in}S.\ (r\ x)^{-1})$
⟨*proof*⟩

**lemma** *converse-mono*[*simp*]: $r^{-1} \subseteq s\ ^{-1} \longleftrightarrow r \subseteq s$
⟨*proof*⟩

**lemma** *conversep-mono*[*simp*]: $r^{-1-1} \leq s\ ^{-1-1} \longleftrightarrow r \leq s$
⟨*proof*⟩

**lemma** *converse-inject*[*simp*]: $r^{-1} = s\ ^{-1} \longleftrightarrow r = s$
⟨*proof*⟩

**lemma** *conversep-inject*[*simp*]: $r^{-1-1} = s\ ^{-1-1} \longleftrightarrow r = s$
⟨*proof*⟩

**lemma** *converse-subset-swap*: $r \subseteq s\ ^{-1} \longleftrightarrow r\ ^{-1} \subseteq s$
⟨*proof*⟩

**lemma** *conversep-le-swap*: $r \leq s\ ^{-1-1} \longleftrightarrow r\ ^{-1-1} \leq s$
⟨*proof*⟩

**lemma** *converse-Id* [*simp*]: $Id^{-1} = Id$

⟨proof⟩

**lemma** *converse-Id-on* [*simp*]: $(Id\text{-}on\ A)^{-1} = Id\text{-}on\ A$
⟨proof⟩

**lemma** *refl-on-converse* [*simp*]: $refl\text{-}on\ A\ (r^{-1}) = refl\text{-}on\ A\ r$
⟨proof⟩

**lemma** *reflp-on-conversp* [*simp*]: $reflp\text{-}on\ A\ R^{-1-1} \longleftrightarrow reflp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *irrefl-on-converse* [*simp*]: $irrefl\text{-}on\ A\ (r^{-1}) = irrefl\text{-}on\ A\ r$
⟨proof⟩

**lemma** *irreflp-on-converse* [*simp*]: $irreflp\text{-}on\ A\ (r^{-1-1}) = irreflp\text{-}on\ A\ r$
⟨proof⟩

**lemma** *sym-on-converse* [*simp*]: $sym\text{-}on\ A\ (r^{-1}) = sym\text{-}on\ A\ r$
⟨proof⟩

**lemma** *symp-on-conversep* [*simp*]: $symp\text{-}on\ A\ R^{-1-1} = symp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *asym-on-converse* [*simp*]: $asym\text{-}on\ A\ (r^{-1}) = asym\text{-}on\ A\ r$
⟨proof⟩

**lemma** *asymp-on-conversep* [*simp*]: $asymp\text{-}on\ A\ R^{-1-1} = asymp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *antisym-on-converse* [*simp*]: $antisym\text{-}on\ A\ (r^{-1}) = antisym\text{-}on\ A\ r$
⟨proof⟩

**lemma** *antisymp-on-conversep* [*simp*]: $antisymp\text{-}on\ A\ R^{-1-1} = antisymp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *trans-on-converse* [*simp*]: $trans\text{-}on\ A\ (r^{-1}) = trans\text{-}on\ A\ r$
⟨proof⟩

**lemma** *transp-on-conversep* [*simp*]: $transp\text{-}on\ A\ R^{-1-1} = transp\text{-}on\ A\ R$
⟨proof⟩

**lemma** *sym-conv-converse-eq*: $sym\ r \longleftrightarrow r^{-1} = r$
⟨proof⟩

**lemma** *sym-Un-converse*: $sym\ (r \cup r^{-1})$
⟨proof⟩

**lemma** *sym-Int-converse*: $sym\ (r \cap r^{-1})$
⟨proof⟩

**lemma** *total-on-converse* [*simp*]: *total-on* $A$ $(r^{-1})$ = *total-on* $A$ $r$
⟨*proof*⟩

**lemma** *totalp-on-converse* [*simp*]: *totalp-on* $A$ $R^{-1-1}$ = *totalp-on* $A$ $R$
⟨*proof*⟩

**lemma** *left-unique-conversep*[*simp*]: *left-unique* $A^{-1-1}$ ⟷ *right-unique* $A$
⟨*proof*⟩

**lemma** *right-unique-conversep*[*simp*]: *right-unique* $A^{-1-1}$ ⟷ *left-unique* $A$
⟨*proof*⟩

**lemma** *conversep-noteq* [*simp*]: $(\neq)^{-1-1} = (\neq)$
⟨*proof*⟩

**lemma** *conversep-eq* [*simp*]: $(=)^{-1-1} = (=)$
⟨*proof*⟩

**lemma** *converse-unfold* [*code*]: $r^{-1} = \{(y, x).\ (x, y) \in r\}$
⟨*proof*⟩

### 19.3.5 Domain, range and field

**inductive-set** *Domain* :: $('a \times 'b)$ *set* ⇒ *'a set* **for** $r$ :: $('a \times 'b)$ *set*
**where** *DomainI* [*intro*]: $(a, b) \in r \Longrightarrow a \in$ *Domain* $r$

**lemmas** *DomainPI* = *Domainp.DomainI*

**inductive-cases** *DomainE* [*elim*!]: $a \in$ *Domain* $r$
**inductive-cases** *DomainpE* [*elim*!]: *Domainp* $r$ $a$

**inductive-set** *Range* :: $('a \times 'b)$ *set* ⇒ *'b set* **for** $r$ :: $('a \times 'b)$ *set*
**where** *RangeI* [*intro*]: $(a, b) \in r \Longrightarrow b \in$ *Range* $r$

**lemmas** *RangePI* = *Rangep.RangeI*

**inductive-cases** *RangeE* [*elim*!]: $b \in$ *Range* $r$
**inductive-cases** *RangepE* [*elim*!]: *Rangep* $r$ $b$

**definition** *Field* :: *'a rel* ⇒ *'a set*
**where** *Field* $r$ = *Domain* $r$ ∪ *Range* $r$

**lemma** *Field-iff*: $x \in$ *Field* $r$ ⟷ $(\exists y.\ (x,y) \in r \lor (y,x) \in r)$
⟨*proof*⟩

**lemma** *FieldI1*: $(i, j) \in R \Longrightarrow i \in$ *Field* $R$
⟨*proof*⟩

**lemma** *FieldI2*: $(i, j) \in R \Longrightarrow j \in Field\ R$
$\langle proof \rangle$

**lemma** *Domain-fst* [*code*]: $Domain\ r = fst\ `\ r$
$\langle proof \rangle$

**lemma** *Range-snd* [*code*]: $Range\ r = snd\ `\ r$
$\langle proof \rangle$

**lemma** *fst-eq-Domain*: $fst\ `\ R = Domain\ R$
$\langle proof \rangle$

**lemma** *snd-eq-Range*: $snd\ `\ R = Range\ R$
$\langle proof \rangle$

**lemma** *range-fst* [*simp*]: $range\ fst = UNIV$
$\langle proof \rangle$

**lemma** *range-snd* [*simp*]: $range\ snd = UNIV$
$\langle proof \rangle$

**lemma** *Domain-empty* [*simp*]: $Domain\ \{\} = \{\}$
$\langle proof \rangle$

**lemma** *Range-empty* [*simp*]: $Range\ \{\} = \{\}$
$\langle proof \rangle$

**lemma** *Field-empty* [*simp*]: $Field\ \{\} = \{\}$
$\langle proof \rangle$

**lemma** *Domain-empty-iff*: $Domain\ r = \{\} \longleftrightarrow r = \{\}$
$\langle proof \rangle$

**lemma** *Range-empty-iff*: $Range\ r = \{\} \longleftrightarrow r = \{\}$
$\langle proof \rangle$

**lemma** *Domain-insert* [*simp*]: $Domain\ (insert\ (a, b)\ r) = insert\ a\ (Domain\ r)$
$\langle proof \rangle$

**lemma** *Range-insert* [*simp*]: $Range\ (insert\ (a, b)\ r) = insert\ b\ (Range\ r)$
$\langle proof \rangle$

**lemma** *Field-insert* [*simp*]: $Field\ (insert\ (a, b)\ r) = \{a, b\} \cup Field\ r$
$\langle proof \rangle$

**lemma** *Domain-iff*: $a \in Domain\ r \longleftrightarrow (\exists\, y.\ (a, y) \in r)$
$\langle proof \rangle$

**lemma** *Range-iff*: $a \in Range\ r \longleftrightarrow (\exists\, y.\ (y, a) \in r)$

⟨*proof*⟩

**lemma** *Domain-Id* [*simp*]: *Domain Id* = *UNIV*
⟨*proof*⟩

**lemma** *Range-Id* [*simp*]: *Range Id* = *UNIV*
⟨*proof*⟩

**lemma** *Domain-Id-on* [*simp*]: *Domain* (*Id-on A*) = *A*
⟨*proof*⟩

**lemma** *Range-Id-on* [*simp*]: *Range* (*Id-on A*) = *A*
⟨*proof*⟩

**lemma** *Domain-Un-eq*: *Domain* ($A \cup B$) = *Domain A* ∪ *Domain B*
⟨*proof*⟩

**lemma** *Range-Un-eq*: *Range* ($A \cup B$) = *Range A* ∪ *Range B*
⟨*proof*⟩

**lemma** *Field-Un* [*simp*]: *Field* ($r \cup s$) = *Field r* ∪ *Field s*
⟨*proof*⟩

**lemma** *Domain-Int-subset*: *Domain* ($A \cap B$) ⊆ *Domain A* ∩ *Domain B*
⟨*proof*⟩

**lemma** *Range-Int-subset*: *Range* ($A \cap B$) ⊆ *Range A* ∩ *Range B*
⟨*proof*⟩

**lemma** *Domain-Diff-subset*: *Domain A* − *Domain B* ⊆ *Domain* ($A - B$)
⟨*proof*⟩

**lemma** *Range-Diff-subset*: *Range A* − *Range B* ⊆ *Range* ($A - B$)
⟨*proof*⟩

**lemma** *Domain-Union*: *Domain* ($\bigcup S$) = ($\bigcup A \in S$. *Domain A*)
⟨*proof*⟩

**lemma** *Range-Union*: *Range* ($\bigcup S$) = ($\bigcup A \in S$. *Range A*)
⟨*proof*⟩

**lemma** *Field-Union* [*simp*]: *Field* ($\bigcup R$) = $\bigcup$(*Field* ‘ *R*)
⟨*proof*⟩

**lemma** *Domain-converse* [*simp*]: *Domain* ($r^{-1}$) = *Range r*
⟨*proof*⟩

**lemma** *Range-converse* [*simp*]: *Range* ($r^{-1}$) = *Domain r*
⟨*proof*⟩

**lemma** *Field-converse* [*simp*]: *Field* $(r^{-1})$ = *Field r*
⟨*proof*⟩

**lemma** *Domain-Collect-case-prod* [*simp*]: *Domain* $\{(x, y).\ P\ x\ y\} = \{x.\ \exists\, y.\ P\ x\ y\}$
⟨*proof*⟩

**lemma** *Range-Collect-case-prod* [*simp*]: *Range* $\{(x, y).\ P\ x\ y\} = \{y.\ \exists\, x.\ P\ x\ y\}$
⟨*proof*⟩

**lemma** *Domain-mono*: $r \subseteq s \Longrightarrow$ *Domain r* $\subseteq$ *Domain s*
⟨*proof*⟩

**lemma** *Range-mono*: $r \subseteq s \Longrightarrow$ *Range r* $\subseteq$ *Range s*
⟨*proof*⟩

**lemma** *mono-Field*: $r \subseteq s \Longrightarrow$ *Field r* $\subseteq$ *Field s*
⟨*proof*⟩

**lemma** *Domain-unfold*: *Domain r* $= \{x.\ \exists\, y.\ (x, y) \in r\}$
⟨*proof*⟩

**lemma** *Field-square* [*simp*]: *Field* $(x \times x) = x$
⟨*proof*⟩

### 19.3.6 Image of a set under a relation

**definition** *Image* :: $('a \times 'b)$ *set* $\Rightarrow$ *'a set* $\Rightarrow$ *'b set* (**infixr** ‹‘‘› *90*)
**where** $r\ ``\ s = \{y.\ \exists\, x{\in}s.\ (x, y) \in r\}$

**lemma** *Image-iff*: $b \in r``A \longleftrightarrow (\exists\, x{\in}A.\ (x, b) \in r)$
⟨*proof*⟩

**lemma** *Image-singleton*: $r``\{a\} = \{b.\ (a, b) \in r\}$
⟨*proof*⟩

**lemma** *Image-singleton-iff* [*iff*]: $b \in r``\{a\} \longleftrightarrow (a, b) \in r$
⟨*proof*⟩

**lemma** *ImageI* [*intro*]: $(a, b) \in r \Longrightarrow a \in A \Longrightarrow b \in r``A$
⟨*proof*⟩

**lemma** *ImageE* [*elim!*]: $b \in r\ ``\ A \Longrightarrow (\bigwedge x.\ (x, b) \in r \Longrightarrow x \in A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *rev-ImageI*: $a \in A \Longrightarrow (a, b) \in r \Longrightarrow b \in r\ ``\ A$
— This version's more effective when we already have the required $a$
⟨*proof*⟩

**lemma** *Image-empty1* [*simp*]: $\{\}$ `` $X = \{\}$
⟨*proof*⟩

**lemma** *Image-empty2* [*simp*]: $R$``$\{\} = \{\}$
⟨*proof*⟩

**lemma** *Image-Id* [*simp*]: *Id* `` $A = A$
⟨*proof*⟩

**lemma** *Image-Id-on* [*simp*]: *Id-on* $A$ `` $B = A \cap B$
⟨*proof*⟩

**lemma** *Image-Int-subset*: $R$ `` $(A \cap B) \subseteq R$ `` $A \cap R$ `` $B$
⟨*proof*⟩

**lemma** *Image-Int-eq*: *single-valued* (*converse* $R$) $\Longrightarrow R$ `` $(A \cap B) = R$ `` $A \cap R$ `` $B$
⟨*proof*⟩

**lemma** *Image-Un*: $R$ `` $(A \cup B) = R$ `` $A \cup R$ `` $B$
⟨*proof*⟩

**lemma** *Un-Image*: $(R \cup S)$ `` $A = R$ `` $A \cup S$ `` $A$
⟨*proof*⟩

**lemma** *Image-subset*: $r \subseteq A \times B \Longrightarrow r$``$C \subseteq B$
⟨*proof*⟩

**lemma** *Image-eq-UN*: $r$``$B = (\bigcup y \in B.\ r$``$\{y\})$
— NOT suitable for rewriting
⟨*proof*⟩

**lemma** *Image-mono*: $r' \subseteq r \Longrightarrow A' \subseteq A \Longrightarrow (r'$ `` $A') \subseteq (r$ `` $A)$
⟨*proof*⟩

**lemma** *Image-UN*: $r$ `` $(\bigcup (B$ ` $A)) = (\bigcup x \in A.\ r$ `` $(B\ x))$
⟨*proof*⟩

**lemma** *UN-Image*: $(\bigcup i \in I.\ X\ i)$ `` $S = (\bigcup i \in I.\ X\ i$ `` $S)$
⟨*proof*⟩

**lemma** *Image-INT-subset*: $(r$ `` $(\bigcap (B$ ` $A))) \subseteq (\bigcap x \in A.\ r$ `` $(B\ x))$
⟨*proof*⟩

Converse inclusion requires some assumptions

**lemma** *Image-INT-eq*:
**assumes** *single-valued* $(r^{-1})$
**and** $A \neq \{\}$

**shows** $r$ `` $(\bigcap(B$ ` $A)) = (\bigcap x{\in}A.\ r$ `` $B\ x)$
⟨proof⟩

**lemma** *Image-subset-eq*: $r$``$A \subseteq B \longleftrightarrow A \subseteq -\ ((r^{-1})$ `` $(-\ B))$
⟨proof⟩

**lemma** *Image-Collect-case-prod* [*simp*]: $\{(x,\ y).\ P\ x\ y\}$ `` $A = \{y.\ \exists\, x{\in}A.\ P\ x\ y\}$
⟨proof⟩

**lemma** *Sigma-Image*: $(SIGMA\ x{:}A.\ B\ x)$ `` $X = (\bigcup x{\in}X \cap A.\ B\ x)$
⟨proof⟩

**lemma** *relcomp-Image*: $(X\ O\ Y)$ `` $Z = Y$ `` $(X$ `` $Z)$
⟨proof⟩

### 19.3.7 Inverse image

**definition** *inv-image* :: $'b\ rel \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a\ rel$
**where** *inv-image* $r\ f = \{(x,\ y).\ (f\ x,\ f\ y) \in r\}$

**definition** *inv-imagep* :: $('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
**where** *inv-imagep* $r\ f = (\lambda x\ y.\ r\ (f\ x)\ (f\ y))$

**lemma** [*pred-set-conv*]: *inv-imagep* $(\lambda x\ y.\ (x,\ y) \in r)\ f = (\lambda x\ y.\ (x,\ y) \in$ *inv-image* $r\ f)$
⟨proof⟩

**lemma** *sym-inv-image*: $sym\ r \Longrightarrow sym\ (inv\text{-}image\ r\ f)$
⟨proof⟩

**lemma** *trans-inv-image*: $trans\ r \Longrightarrow trans\ (inv\text{-}image\ r\ f)$
⟨proof⟩

**lemma** *total-inv-image*: $[\![inj\ f;\ total\ r]\!] \Longrightarrow total\ (inv\text{-}image\ r\ f)$
⟨proof⟩

**lemma** *asym-inv-image*: $asym\ R \Longrightarrow asym\ (inv\text{-}image\ R\ f)$
⟨proof⟩

**lemma** *in-inv-image*[*simp*]: $(x,\ y) \in inv\text{-}image\ r\ f \longleftrightarrow (f\ x,\ f\ y) \in r$
⟨proof⟩

**lemma** *converse-inv-image*[*simp*]: $(inv\text{-}image\ R\ f)^{-1} = inv\text{-}image\ (R^{-1})\ f$
⟨proof⟩

**lemma** *in-inv-imagep* [*simp*]: $inv\text{-}imagep\ r\ f\ x\ y = r\ (f\ x)\ (f\ y)$
⟨proof⟩

### 19.3.8 Powerset

**definition** *Powp* :: $('a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
**where** $Powp\ A = (\lambda B.\ \forall\, x \in B.\ A\ x)$

**lemma** *Powp-Pow-eq* [*pred-set-conv*]: $Powp\ (\lambda x.\ x \in A) = (\lambda x.\ x \in Pow\ A)$
⟨*proof*⟩

**lemmas** *Powp-mono* [*mono*] = *Pow-mono* [*to-pred*]

**end**

# 20 Finite sets

**theory** *Finite-Set*
**imports** *Product-Type Sum-Type Fields Relation*
**begin**

## 20.1 Predicate for finite sets

**context notes** [[*inductive-internals*]]
**begin**

**inductive** *finite* :: $'a\ set \Rightarrow bool$
**where**
*emptyI* [*simp*, *intro*!]: *finite* {}
| *insertI* [*simp*, *intro*!]: $finite\ A \Longrightarrow finite\ (insert\ a\ A)$

**end**

⟨*ML*⟩

**declare** [[*simproc del*: *finite-Collect*]]

**lemma** *finite-induct* [*case-names empty insert*, *induct set*: *finite*]:
— Discharging $x \notin F$ entails extra work.
**assumes** *finite F*
**assumes** *P* {}
**and** *insert*: $\bigwedge x\ F.\ finite\ F \Longrightarrow x \notin F \Longrightarrow P\ F \Longrightarrow P\ (insert\ x\ F)$
**shows** *P F*
⟨*proof*⟩

**lemma** *infinite-finite-induct* [*case-names infinite empty insert*]:
**assumes** *infinite*: $\bigwedge A.\ \neg\ finite\ A \Longrightarrow P\ A$
**and** *empty*: *P* {}
**and** *insert*: $\bigwedge x\ F.\ finite\ F \Longrightarrow x \notin F \Longrightarrow P\ F \Longrightarrow P\ (insert\ x\ F)$
**shows** *P A*
⟨*proof*⟩

### 20.1.1 Choice principles

**lemma** *ex-new-if-finite*: — does not depend on def of finite at all
**assumes** $\neg$ *finite* (*UNIV* :: *'a set*) **and** *finite* $A$
**shows** $\exists\, a{::}'a.\ a \notin A$
⟨*proof*⟩

A finite choice principle. Does not need the SOME choice operator.

**lemma** *finite-set-choice*: *finite* $A \Longrightarrow \forall\, x{\in}A.\ \exists\, y.\ P\ x\ y \Longrightarrow \exists\, f.\ \forall\, x{\in}A.\ P\ x\ (f\ x)$
⟨*proof*⟩

### 20.1.2 Finite sets are the images of initial segments of natural numbers

**lemma** *finite-imp-nat-seg-image-inj-on*:
**assumes** *finite* $A$
**shows** $\exists\, (n{::}nat)\ f.\ A = f\ \text{`}\ \{i.\ i < n\} \wedge$ *inj-on* $f\ \{i.\ i < n\}$
⟨*proof*⟩

**lemma** *nat-seg-image-imp-finite*: $A = f\ \text{`}\ \{i{::}nat.\ i < n\} \Longrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *finite-conv-nat-seg-image*: *finite* $A \longleftrightarrow (\exists\, n\ f.\ A = f\ \text{`}\ \{i{::}nat.\ i < n\})$
⟨*proof*⟩

**lemma** *finite-imp-inj-to-nat-seg*:
**assumes** *finite* $A$
**shows** $\exists f\ n.\ f\ \text{`}\ A = \{i{::}nat.\ i < n\} \wedge$ *inj-on* $f\ A$
⟨*proof*⟩

**lemma** *finite-Collect-less-nat* [*iff*]: *finite* $\{n{::}nat.\ n < k\}$
⟨*proof*⟩

**lemma** *finite-Collect-le-nat* [*iff*]: *finite* $\{n{::}nat.\ n \le k\}$
⟨*proof*⟩

## 20.2 Finiteness and common set operations

**lemma** *rev-finite-subset*: *finite* $B \Longrightarrow A \subseteq B \Longrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *finite-subset*: $A \subseteq B \Longrightarrow$ *finite* $B \Longrightarrow$ *finite* $A$
⟨*proof*⟩

⟨*ML*⟩

**declare** [[*simproc del*: *finite*]]

**lemma** *finite-UnI*:

**assumes** *finite* $F$ **and** *finite* $G$
**shows** *finite* $(F \cup G)$
⟨*proof*⟩

**lemma** *finite-Un* [*iff*]: *finite* $(F \cup G) \longleftrightarrow$ *finite* $F \wedge$ *finite* $G$
⟨*proof*⟩

**lemma** *finite-insert* [*simp*]: *finite* (*insert* $a$ $A$) $\longleftrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *finite-Int* [*simp*, *intro*]: *finite* $F \vee$ *finite* $G \Longrightarrow$ *finite* $(F \cap G)$
⟨*proof*⟩

**lemma** *finite-Collect-conjI* [*simp*, *intro*]:
*finite* $\{x.\ P\ x\} \vee$ *finite* $\{x.\ Q\ x\} \Longrightarrow$ *finite* $\{x.\ P\ x \wedge Q\ x\}$
⟨*proof*⟩

**lemma** *finite-Collect-disjI* [*simp*]:
*finite* $\{x.\ P\ x \vee Q\ x\} \longleftrightarrow$ *finite* $\{x.\ P\ x\} \wedge$ *finite* $\{x.\ Q\ x\}$
⟨*proof*⟩

**lemma** *finite-Diff* [*simp*, *intro*]: *finite* $A \Longrightarrow$ *finite* $(A - B)$
⟨*proof*⟩

**lemma** *finite-Diff2* [*simp*]:
**assumes** *finite* $B$
**shows** *finite* $(A - B) \longleftrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *finite-Diff-insert* [*iff*]: *finite* ($A$ − *insert* $a$ $B$) $\longleftrightarrow$ *finite* $(A - B)$
⟨*proof*⟩

**lemma** *finite-compl* [*simp*]:
*finite* $(A :: 'a\ set) \Longrightarrow$ *finite* $(-\ A) \longleftrightarrow$ *finite* $(UNIV :: 'a\ set)$
⟨*proof*⟩

**lemma** *finite-Collect-not* [*simp*]:
*finite* $\{x :: 'a.\ P\ x\} \Longrightarrow$ *finite* $\{x.\ \neg\ P\ x\} \longleftrightarrow$ *finite* $(UNIV :: 'a\ set)$
⟨*proof*⟩

**lemma** *finite-Union* [*simp*, *intro*]:
*finite* $A \Longrightarrow (\bigwedge M.\ M \in A \Longrightarrow$ *finite* $M) \Longrightarrow$ *finite* $(\bigcup A)$
⟨*proof*⟩

**lemma** *finite-UN-I* [*intro*]:
*finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow$ *finite* $(B\ a)) \Longrightarrow$ *finite* $(\bigcup a{\in}A.\ B\ a)$
⟨*proof*⟩

**lemma** *finite-UN* [*simp*]: *finite* $A \Longrightarrow$ *finite* $(\bigcup (B\ `\ A)) \longleftrightarrow (\forall\, x{\in}A.$ *finite* $(B\ x))$

⟨*proof*⟩

**lemma** *finite-Inter* [*intro*]: $\exists A{\in}M.\ \mathit{finite}\ A \Longrightarrow \mathit{finite}\ (\bigcap M)$
⟨*proof*⟩

**lemma** *finite-INT* [*intro*]: $\exists x{\in}I.\ \mathit{finite}\ (A\ x) \Longrightarrow \mathit{finite}\ (\bigcap x{\in}I.\ A\ x)$
⟨*proof*⟩

**lemma** *finite-imageI* [*simp*, *intro*]: $\mathit{finite}\ F \Longrightarrow \mathit{finite}\ (h\ \text{`}\ F)$
⟨*proof*⟩

**lemma** *finite-image-set* [*simp*]: $\mathit{finite}\ \{x.\ P\ x\} \Longrightarrow \mathit{finite}\ \{f\ x\ |x.\ P\ x\}$
⟨*proof*⟩

**lemma** *finite-image-set2*:
$\mathit{finite}\ \{x.\ P\ x\} \Longrightarrow \mathit{finite}\ \{y.\ Q\ y\} \Longrightarrow \mathit{finite}\ \{f\ x\ y\ |x\ y.\ P\ x \wedge Q\ y\}$
⟨*proof*⟩

**lemma** *finite-imageD*:
**assumes** $\mathit{finite}\ (f\ \text{`}\ A)$ **and** $\mathit{inj\text{-}on}\ f\ A$
**shows** $\mathit{finite}\ A$
⟨*proof*⟩

**lemma** *finite-image-iff*: $\mathit{inj\text{-}on}\ f\ A \Longrightarrow \mathit{finite}\ (f\ \text{`}\ A) \longleftrightarrow \mathit{finite}\ A$
⟨*proof*⟩

**lemma** *finite-surj*: $\mathit{finite}\ A \Longrightarrow B \subseteq f\ \text{`}\ A \Longrightarrow \mathit{finite}\ B$
⟨*proof*⟩

**lemma** *finite-range-imageI*: $\mathit{finite}\ (\mathit{range}\ g) \Longrightarrow \mathit{finite}\ (\mathit{range}\ (\lambda x.\ f\ (g\ x)))$
⟨*proof*⟩

**lemma** *finite-subset-image*:
**assumes** $\mathit{finite}\ B$
**shows** $B \subseteq f\ \text{`}\ A \Longrightarrow \exists C{\subseteq}A.\ \mathit{finite}\ C \wedge B = f\ \text{`}\ C$
⟨*proof*⟩

**lemma** *all-subset-image*: $(\forall B.\ B \subseteq f\ \text{`}\ A \longrightarrow P\ B) \longleftrightarrow (\forall B.\ B \subseteq A \longrightarrow P(f\ \text{`}\ B))$
⟨*proof*⟩

**lemma** *all-finite-subset-image*:
$(\forall B.\ \mathit{finite}\ B \wedge B \subseteq f\ \text{`}\ A \longrightarrow P\ B) \longleftrightarrow (\forall B.\ \mathit{finite}\ B \wedge B \subseteq A \longrightarrow P\ (f\ \text{`}\ B))$
⟨*proof*⟩

**lemma** *ex-finite-subset-image*:
$(\exists B.\ \mathit{finite}\ B \wedge B \subseteq f\ \text{`}\ A \wedge P\ B) \longleftrightarrow (\exists B.\ \mathit{finite}\ B \wedge B \subseteq A \wedge P\ (f\ \text{`}\ B))$
⟨*proof*⟩

**lemma** *finite-vimage-IntI*: $finite\ F \Longrightarrow inj\text{-}on\ h\ A \Longrightarrow finite\ (h\ -`\ F \cap A)$
⟨*proof*⟩

**lemma** *finite-finite-vimage-IntI*:
**assumes** $finite\ F$
**and** $\bigwedge y.\ y \in F \Longrightarrow finite\ ((h\ -`\ \{y\}) \cap A)$
**shows** $finite\ (h\ -`\ F \cap A)$
⟨*proof*⟩

**lemma** *finite-vimageI*: $finite\ F \Longrightarrow inj\ h \Longrightarrow finite\ (h\ -`\ F)$
⟨*proof*⟩

**lemma** *finite-vimageD'*: $finite\ (f\ -`\ A) \Longrightarrow A \subseteq range\ f \Longrightarrow finite\ A$
⟨*proof*⟩

**lemma** *finite-vimageD*: $finite\ (h\ -`\ F) \Longrightarrow surj\ h \Longrightarrow finite\ F$
⟨*proof*⟩

**lemma** *finite-vimage-iff*: $bij\ h \Longrightarrow finite\ (h\ -`\ F) \longleftrightarrow finite\ F$
⟨*proof*⟩

**lemma** *finite-inverse-image-gen*:
**assumes** $finite\ A\ \ inj\text{-}on\ f\ D$
**shows** $finite\ \{j{\in}D.\ f\ j \in A\}$
⟨*proof*⟩

**lemma** *finite-inverse-image*:
**assumes** $finite\ A\ \ inj\ f$
**shows** $finite\ \{j.\ f\ j \in A\}$
⟨*proof*⟩

**lemma** *finite-Collect-bex* [*simp*]:
**assumes** $finite\ A$
**shows** $finite\ \{x.\ \exists\, y{\in}A.\ Q\ x\ y\} \longleftrightarrow (\forall\, y{\in}A.\ finite\ \{x.\ Q\ x\ y\})$
⟨*proof*⟩

**lemma** *finite-Collect-bounded-ex* [*simp*]:
**assumes** $finite\ \{y.\ P\ y\}$
**shows** $finite\ \{x.\ \exists\, y.\ P\ y \wedge Q\ x\ y\} \longleftrightarrow (\forall\, y.\ P\ y \longrightarrow finite\ \{x.\ Q\ x\ y\})$
⟨*proof*⟩

**lemma** *finite-Plus*: $finite\ A \Longrightarrow finite\ B \Longrightarrow finite\ (A <+> B)$
⟨*proof*⟩

**lemma** *finite-PlusD*:
**fixes** $A :: 'a\ set$ **and** $B :: 'b\ set$
**assumes** *fin*: $finite\ (A <+> B)$
**shows** $finite\ A\ \ finite\ B$
⟨*proof*⟩

**lemma** *finite-Plus-iff* [*simp*]: *finite* $(A <+> B) \longleftrightarrow$ *finite* $A \wedge$ *finite* $B$
⟨*proof*⟩

**lemma** *finite-Plus-UNIV-iff* [*simp*]:
*finite* $(UNIV :: ('a + 'b)\ set) \longleftrightarrow$ *finite* $(UNIV :: 'a\ set) \wedge$ *finite* $(UNIV :: 'b\ set)$
⟨*proof*⟩

**lemma** *finite-SigmaI* [*simp*, *intro*]:
*finite* $A \Longrightarrow (\bigwedge a.\ a{\in}A \Longrightarrow$ *finite* $(B\ a)) \Longrightarrow$ *finite* $(SIGMA\ a{:}A.\ B\ a)$
⟨*proof*⟩

**lemma** *finite-SigmaI2*:
**assumes** *finite* $\{x{\in}A.\ B\ x \neq \{\}\}$
**and** $\bigwedge a.\ a \in A \Longrightarrow$ *finite* $(B\ a)$
**shows** *finite* $(Sigma\ A\ B)$
⟨*proof*⟩

**lemma** *finite-cartesian-product*: *finite* $A \Longrightarrow$ *finite* $B \Longrightarrow$ *finite* $(A \times B)$
⟨*proof*⟩

**lemma** *finite-Prod-UNIV*:
*finite* $(UNIV :: 'a\ set) \Longrightarrow$ *finite* $(UNIV :: 'b\ set) \Longrightarrow$ *finite* $(UNIV :: ('a \times 'b)\ set)$
⟨*proof*⟩

**lemma** *finite-cartesian-productD1*:
**assumes** *finite* $(A \times B)$ **and** $B \neq \{\}$
**shows** *finite* $A$
⟨*proof*⟩

**lemma** *finite-cartesian-productD2*:
**assumes** *finite* $(A \times B)$ **and** $A \neq \{\}$
**shows** *finite* $B$
⟨*proof*⟩

**lemma** *finite-cartesian-product-iff*:
*finite* $(A \times B) \longleftrightarrow (A = \{\} \vee B = \{\} \vee ($*finite* $A \wedge$ *finite* $B))$
⟨*proof*⟩

**lemma** *finite-prod*:
*finite* $(UNIV :: ('a \times 'b)\ set) \longleftrightarrow$ *finite* $(UNIV :: 'a\ set) \wedge$ *finite* $(UNIV :: 'b\ set)$
⟨*proof*⟩

**lemma** *finite-Pow-iff* [*iff*]: *finite* $(Pow\ A) \longleftrightarrow$ *finite* $A$
⟨*proof*⟩

**corollary** *finite-Collect-subsets* [*simp*, *intro*]: *finite* $A \Longrightarrow$ *finite* $\{B.\ B \subseteq A\}$
⟨*proof*⟩

**lemma** *finite-set*: *finite* (*UNIV* :: *'a set set*) $\longleftrightarrow$ *finite* (*UNIV* :: *'a set*)
⟨*proof*⟩

**lemma** *finite-UnionD*: *finite* $(\bigcup A) \Longrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *finite-bind*:
**assumes** *finite* $S$
**assumes** $\forall x \in S.$ *finite* $(f\ x)$
**shows** *finite* (*Set.bind* $S\ f$)
⟨*proof*⟩

**lemma** *finite-filter* [*simp*]: *finite* $S \Longrightarrow$ *finite* (*Set.filter* $P\ S$)
⟨*proof*⟩

**lemma** *finite-set-of-finite-funs*:
**assumes** *finite* $A$ *finite* $B$
**shows** *finite* $\{f.\ \forall x.\ (x \in A \longrightarrow f\ x \in B) \land (x \notin A \longrightarrow f\ x = d)\}$ (**is** *finite ?S*)
⟨*proof*⟩

**lemma** *not-finite-existsD*:
**assumes** $\neg$ *finite* $\{a.\ P\ a\}$
**shows** $\exists\, a.\ P\ a$
⟨*proof*⟩

**lemma** *finite-converse* [*iff*]: *finite* $(r^{-1}) \longleftrightarrow$ *finite* $r$
⟨*proof*⟩

**lemma** *finite-Domain*: *finite* $r \Longrightarrow$ *finite* (*Domain* $r$)
⟨*proof*⟩

**lemma** *finite-Range*: *finite* $r \Longrightarrow$ *finite* (*Range* $r$)
⟨*proof*⟩

**lemma** *finite-Field*: *finite* $r \Longrightarrow$ *finite* (*Field* $r$)
⟨*proof*⟩

**lemma** *finite-Image*[*simp*]: *finite* $R \Longrightarrow$ *finite* $(R$ `` $A)$
⟨*proof*⟩

## 20.3 Further induction rules on finite sets

**lemma** *finite-ne-induct* [*case-names singleton insert*, *consumes 2*]:
**assumes** *finite* $F$ **and** $F \neq \{\}$
**assumes** $\bigwedge x.\ P\ \{x\}$
**and** $\bigwedge x\ F.$ *finite* $F \Longrightarrow F \neq \{\} \Longrightarrow x \notin F \Longrightarrow P\ F \Longrightarrow P$ (*insert* $x\ F$)

**shows** $P\ F$
$\langle proof \rangle$

**lemma** *finite-subset-induct* [*consumes 2*, *case-names empty insert*]:
**assumes** *finite* $F$ **and** $F \subseteq A$
**and** *empty*: $P\ \{\}$
**and** *insert*: $\bigwedge a\ F.\ finite\ F \Longrightarrow a \in A \Longrightarrow a \notin F \Longrightarrow P\ F \Longrightarrow P\ (insert\ a\ F)$
**shows** $P\ F$
$\langle proof \rangle$

**lemma** *finite-empty-induct*:
**assumes** *finite* $A$
**and** $P\ A$
**and** *remove*: $\bigwedge a\ A.\ finite\ A \Longrightarrow a \in A \Longrightarrow P\ A \Longrightarrow P\ (A - \{a\})$
**shows** $P\ \{\}$
$\langle proof \rangle$

**lemma** *finite-update-induct* [*consumes 1*, *case-names const update*]:
**assumes** *finite*: *finite* $\{a.\ f\ a \neq c\}$
**and** *const*: $P\ (\lambda a.\ c)$
**and** *update*: $\bigwedge a\ b\ f.\ finite\ \{a.\ f\ a \neq c\} \Longrightarrow f\ a = c \Longrightarrow b \neq c \Longrightarrow P\ f \Longrightarrow P$
$(f(a := b))$
**shows** $P\ f$
$\langle proof \rangle$

**lemma** *finite-subset-induct'* [*consumes 2*, *case-names empty insert*]:
**assumes** *finite* $F$ **and** $F \subseteq A$
**and** *empty*: $P\ \{\}$
**and** *insert*: $\bigwedge a\ F.\ [\![finite\ F;\ a \in A;\ F \subseteq A;\ a \notin F;\ P\ F\ ]\!] \Longrightarrow P\ (insert\ a\ F)$
**shows** $P\ F$
$\langle proof \rangle$

## 20.4 Class *finite*

**class** *finite* =
**assumes** *finite-UNIV*: *finite* $(UNIV :: 'a\ set)$
**begin**

**lemma** *finite* [*simp*]: *finite* $(A :: 'a\ set)$
$\langle proof \rangle$

**lemma** *finite-code* [*code*]: *finite* $(A :: 'a\ set) \longleftrightarrow True$
$\langle proof \rangle$

**end**

**instance** *prod* :: (*finite*, *finite*) *finite*
$\langle proof \rangle$

**lemma** *inj-graph*: $inj\ (\lambda f.\ \{(x,\ y).\ y = f\ x\})$
⟨*proof*⟩

**instance** *fun* :: (*finite*, *finite*) *finite*
⟨*proof*⟩

**instance** *bool* :: *finite*
⟨*proof*⟩

**instance** *set* :: (*finite*) *finite*
⟨*proof*⟩

**instance** *unit* :: *finite*
⟨*proof*⟩

**instance** *sum* :: (*finite*, *finite*) *finite*
⟨*proof*⟩

## 20.5 A basic fold functional for finite sets

The intended behaviour is $fold\ f\ z\ \{x_1,\ \ldots,\ x_n\} = f\ x_1\ (\ldots\ (f\ x_n\ z)\ldots)$ if $f$ is "left-commutative". The commutativity requirement is relativised to the carrier set $S$:

**locale** *comp-fun-commute-on* =
**fixes** $S$ :: $'a\ set$
**fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'b$
**assumes** *comp-fun-commute-on*: $x \in S \Longrightarrow y \in S \Longrightarrow f\ y \circ f\ x = f\ x \circ f\ y$
**begin**

**lemma** *fun-left-comm*: $x \in S \Longrightarrow y \in S \Longrightarrow f\ y\ (f\ x\ z) = f\ x\ (f\ y\ z)$
⟨*proof*⟩

**lemma** *commute-left-comp*: $x \in S \Longrightarrow y \in S \Longrightarrow f\ y \circ (f\ x \circ g) = f\ x \circ (f\ y \circ g)$
⟨*proof*⟩

**end**

**inductive** *fold-graph* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow bool$
**for** $f$ :: $'a \Rightarrow 'b \Rightarrow 'b$ **and** $z$ :: $'b$
**where**
*emptyI* [*intro*]: *fold-graph* $f\ z\ \{\}\ z$
| *insertI* [*intro*]: $x \notin A \Longrightarrow$ *fold-graph* $f\ z\ A\ y \Longrightarrow$ *fold-graph* $f\ z\ (insert\ x\ A)\ (f\ x\ y)$

**inductive-cases** *empty-fold-graphE* [*elim*!]: *fold-graph* $f\ z\ \{\}\ x$

**lemma** *fold-graph-closed-lemma*:
*fold-graph* $f\ z\ A\ x \land x \in B$
**if** *fold-graph* $g\ z\ A\ x$

$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow f\ a\ b = g\ a\ b$
$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow g\ a\ b \in B$
$z \in B$
⟨*proof*⟩

**lemma** *fold-graph-closed-eq*:
*fold-graph f z A = fold-graph g z A*
**if** $\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow f\ a\ b = g\ a\ b$
$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow g\ a\ b \in B$
$z \in B$
⟨*proof*⟩

**definition** *fold* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set \Rightarrow 'b$
**where** *fold f z A* = (*if finite A then* (*THE y. fold-graph f z A y*) *else z*)

**lemma** *fold-closed-eq*: *fold f z A = fold g z A*
**if** $\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow f\ a\ b = g\ a\ b$
$\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow g\ a\ b \in B$
$z \in B$
⟨*proof*⟩

A tempting alternative for the definition is *if finite A then THE y. fold-graph f z A y else e*. It allows the removal of finiteness assumptions from the theorems *fold-comm*, *fold-reindex* and *fold-distrib*. The proofs become ugly. It is not worth the effort. (???)

**lemma** *finite-imp-fold-graph*: *finite A* $\Longrightarrow \exists x.$ *fold-graph f z A x*
⟨*proof*⟩

### 20.5.1 From *fold-graph* to *fold*

**context** *comp-fun-commute-on*
**begin**

**lemma** *fold-graph-finite*:
**assumes** *fold-graph f z A y*
**shows** *finite A*
⟨*proof*⟩

**lemma** *fold-graph-insertE-aux*:
**assumes** $A \subseteq S$
**assumes** *fold-graph f z A y* $a \in A$
**shows** $\exists y'.\ y = f\ a\ y' \wedge$ *fold-graph f z* $(A - \{a\})\ y'$
⟨*proof*⟩

**lemma** *fold-graph-insertE*:
**assumes** *insert x A* $\subseteq S$
**assumes** *fold-graph f z* (*insert x A*) *v* **and** $x \notin A$
**obtains** *y* **where** $v = f\ x\ y$ **and** *fold-graph f z A y*
⟨*proof*⟩

**lemma** *fold-graph-determ*:
  **assumes** $A \subseteq S$
  **assumes** *fold-graph f z A x fold-graph f z A y*
  **shows** $y = x$
  ⟨*proof*⟩

**lemma** *fold-equality*: $A \subseteq S \Longrightarrow$ *fold-graph f z A y* $\Longrightarrow$ *fold f z A* $= y$
  ⟨*proof*⟩

**lemma** *fold-graph-fold*:
  **assumes** $A \subseteq S$
  **assumes** *finite A*
  **shows** *fold-graph f z A* (*fold f z A*)
⟨*proof*⟩

The base case for *fold*:

**lemma** (**in** $-$) *fold-infinite* [*simp*]: $\neg$ *finite A* $\Longrightarrow$ *fold f z A* $= z$
  ⟨*proof*⟩

**lemma** (**in** $-$) *fold-empty* [*simp*]: *fold f z* $\{\} = z$
  ⟨*proof*⟩

The various recursion equations for *fold*:

**lemma** *fold-insert* [*simp*]:
  **assumes** *insert x A* $\subseteq S$
  **assumes** *finite A* **and** $x \notin A$
  **shows** *fold f z* (*insert x A*) $=$ *f x* (*fold f z A*)
⟨*proof*⟩

**declare** (**in** $-$) *empty-fold-graphE* [*rule del*] *fold-graph.intros* [*rule del*]
  — No more proofs involve these.

**lemma** *fold-fun-left-comm*:
  **assumes** *insert x A* $\subseteq S$ *finite A*
  **shows** *f x* (*fold f z A*) $=$ *fold f* (*f x z*) *A*
  ⟨*proof*⟩

**lemma** *fold-insert2*:
  *insert x A* $\subseteq S \Longrightarrow$ *finite A* $\Longrightarrow x \notin A \Longrightarrow$ *fold f z* (*insert x A*) $=$ *fold f* (*f x z*) *A*
  ⟨*proof*⟩

**lemma** *fold-rec*:
  **assumes** $A \subseteq S$
  **assumes** *finite A* **and** $x \in A$
  **shows** *fold f z A* $=$ *f x* (*fold f z* ($A - \{x\}$))
⟨*proof*⟩

**lemma** *fold-insert-remove*:
**assumes** *insert* $x$ $A \subseteq S$
**assumes** *finite* $A$
**shows** *fold* $f$ $z$ (*insert* $x$ $A$) $= f\ x$ (*fold* $f$ $z$ $(A - \{x\})$)
⟨*proof*⟩

**lemma** *fold-set-union-disj*:
**assumes** $A \subseteq S$ $B \subseteq S$
**assumes** *finite* $A$ *finite* $B$ $A \cap B = \{\}$
**shows** *Finite-Set.fold* $f$ $z$ $(A \cup B) =$ *Finite-Set.fold* $f$ (*Finite-Set.fold* $f$ $z$ $A$) $B$
⟨*proof*⟩

**end**

Other properties of *fold*:

**lemma** *finite-set-fold-single* [*simp*]: *Finite-Set.fold* $f$ $z$ $\{x\} = f\ x\ z$
⟨*proof*⟩

**lemma** *fold-graph-image*:
**assumes** *inj-on* $g$ $A$
**shows** *fold-graph* $f$ $z$ $(g$ ‘ $A) =$ *fold-graph* $(f \circ g)$ $z$ $A$
⟨*proof*⟩

**lemma** *fold-image*:
**assumes** *inj-on* $g$ $A$
**shows** *fold* $f$ $z$ $(g$ ‘ $A) =$ *fold* $(f \circ g)$ $z$ $A$
⟨*proof*⟩

**lemma** *fold-cong*:
**assumes** *comp-fun-commute-on* $S$ $f$ *comp-fun-commute-on* $S$ $g$
**and** $A \subseteq S$ *finite* $A$
**and** *cong*: $\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x$
**and** $s = t$ **and** $A = B$
**shows** *fold* $f$ $s$ $A =$ *fold* $g$ $t$ $B$
⟨*proof*⟩

A simplified version for idempotent functions:

**locale** *comp-fun-idem-on* = *comp-fun-commute-on* +
**assumes** *comp-fun-idem-on*: $x \in S \Longrightarrow f\ x \circ f\ x = f\ x$
**begin**

**lemma** *fun-left-idem*: $x \in S \Longrightarrow f\ x\ (f\ x\ z) = f\ x\ z$
⟨*proof*⟩

**lemma** *fold-insert-idem*:
**assumes** *insert* $x$ $A \subseteq S$
**assumes** *fin*: *finite* $A$
**shows** *fold* $f$ $z$ (*insert* $x$ $A$) $= f\ x$ (*fold* $f$ $z$ $A$)

⟨*proof*⟩

**declare** *fold-insert* [*simp del*] *fold-insert-idem* [*simp*]

**lemma** *fold-insert-idem2*: $insert\ x\ A \subseteq S \Longrightarrow finite\ A \Longrightarrow fold\ f\ z\ (insert\ x\ A) = fold\ f\ (f\ x\ z)\ A$
⟨*proof*⟩

**end**

### 20.5.2 Liftings to *comp-fun-commute-on* etc.

**lemma** (**in** *comp-fun-commute-on*) *comp-comp-fun-commute-on*:
$range\ g \subseteq S \Longrightarrow comp\text{-}fun\text{-}commute\text{-}on\ R\ (f \circ g)$
⟨*proof*⟩

**lemma** (**in** *comp-fun-idem-on*) *comp-comp-fun-idem-on*:
**assumes** $range\ g \subseteq S$
**shows** $comp\text{-}fun\text{-}idem\text{-}on\ R\ (f \circ g)$
⟨*proof*⟩

**lemma** (**in** *comp-fun-commute-on*) *comp-fun-commute-on-funpow*:
$comp\text{-}fun\text{-}commute\text{-}on\ S\ (\lambda x.\ f\ x\ \widehat{}\widehat{}\ g\ x)$
⟨*proof*⟩

### 20.5.3 *UNIV* as carrier set

**locale** *comp-fun-commute* =
**fixes** $f :: 'a \Rightarrow 'b \Rightarrow 'b$
**assumes** *comp-fun-commute*: $f\ y \circ f\ x = f\ x \circ f\ y$
**begin**

**lemma** (**in** −) *comp-fun-commute-def'*: $comp\text{-}fun\text{-}commute\ f = comp\text{-}fun\text{-}commute\text{-}on\ UNIV\ f$
⟨*proof*⟩

We abuse the *rewrites* functionality of locales to remove trivial assumptions that result from instantiating the carrier set to *UNIV*.

**sublocale** *comp-fun-commute-on UNIV f*
**rewrites** $\bigwedge X.\ (X \subseteq UNIV) \equiv True$
**and** $\bigwedge x.\ x \in UNIV \equiv True$
**and** $\bigwedge P.\ (True \Longrightarrow P) \equiv Trueprop\ P$
**and** $\bigwedge P\ Q.\ (True \Longrightarrow PROP\ P \Longrightarrow PROP\ Q) \equiv (PROP\ P \Longrightarrow True \Longrightarrow PROP\ Q)$
⟨*proof*⟩

**end**

**lemma** (**in** *comp-fun-commute*) *comp-comp-fun-commute*: $comp\text{-}fun\text{-}commute\ (f\ o\ g)$

⟨*proof*⟩

**lemma** (**in** *comp-fun-commute*) *comp-fun-commute-funpow*: *comp-fun-commute* ($\lambda x.$ $f\ x\ \widehat{\ }\widehat{\ }\ g\ x$)
⟨*proof*⟩

**locale** *comp-fun-idem* = *comp-fun-commute* +
**assumes** *comp-fun-idem*: $f\ x\ o\ f\ x = f\ x$
**begin**

**lemma** (**in** −) *comp-fun-idem-def'*: *comp-fun-idem* $f$ = *comp-fun-idem-on UNIV* $f$
⟨*proof*⟩

Again, we abuse the *rewrites* functionality of locales to remove trivial assumptions that result from instantiating the carrier set to *UNIV*.

**sublocale** *comp-fun-idem-on UNIV f*
**rewrites** $\bigwedge X.\ (X \subseteq UNIV) \equiv True$
**and** $\bigwedge x.\ x \in UNIV \equiv True$
**and** $\bigwedge P.\ (True \Longrightarrow P) \equiv Trueprop\ P$
**and** $\bigwedge P\ Q.\ (True \Longrightarrow PROP\ P \Longrightarrow PROP\ Q) \equiv (PROP\ P \Longrightarrow True \Longrightarrow PROP\ Q)$
⟨*proof*⟩

**end**

**lemma** (**in** *comp-fun-idem*) *comp-comp-fun-idem*: *comp-fun-idem* ($f\ o\ g$)
⟨*proof*⟩

### 20.5.4 Expressing set operations via *fold*

**lemma** *comp-fun-commute-const*: *comp-fun-commute* ($\lambda$-. $f$)
⟨*proof*⟩

**lemma** *comp-fun-idem-insert*: *comp-fun-idem insert*
⟨*proof*⟩

**lemma** *comp-fun-idem-remove*: *comp-fun-idem Set.remove*
⟨*proof*⟩

**lemma** (**in** *semilattice-inf*) *comp-fun-idem-inf*: *comp-fun-idem inf*
⟨*proof*⟩

**lemma** (**in** *semilattice-sup*) *comp-fun-idem-sup*: *comp-fun-idem sup*
⟨*proof*⟩

**lemma** *union-fold-insert*:
**assumes** *finite A*
**shows** $A \cup B = fold\ insert\ B\ A$

⟨*proof*⟩

**lemma** *minus-fold-remove*:
**assumes** *finite A*
**shows** $B - A = fold\ Set.remove\ B\ A$
⟨*proof*⟩

**lemma** *comp-fun-commute-filter-fold*:
$comp\text{-}fun\text{-}commute\ (\lambda x\ A'.\ if\ P\ x\ then\ Set.insert\ x\ A'\ else\ A')$
⟨*proof*⟩

**lemma** *Set-filter-fold*:
**assumes** *finite A*
**shows** $Set.filter\ P\ A = fold\ (\lambda x\ A'.\ if\ P\ x\ then\ Set.insert\ x\ A'\ else\ A')\ \{\}\ A$
⟨*proof*⟩

**lemma** *inter-Set-filter*:
**assumes** *finite B*
**shows** $A \cap B = Set.filter\ (\lambda x.\ x \in A)\ B$
⟨*proof*⟩

**lemma** *image-fold-insert*:
**assumes** *finite A*
**shows** $image\ f\ A = fold\ (\lambda k\ A.\ Set.insert\ (f\ k)\ A)\ \{\}\ A$
⟨*proof*⟩

**lemma** *Ball-fold*:
**assumes** *finite A*
**shows** $Ball\ A\ P = fold\ (\lambda k\ s.\ s \wedge P\ k)\ True\ A$
⟨*proof*⟩

**lemma** *Bex-fold*:
**assumes** *finite A*
**shows** $Bex\ A\ P = fold\ (\lambda k\ s.\ s \vee P\ k)\ False\ A$
⟨*proof*⟩

**lemma** *comp-fun-commute-Pow-fold*: $comp\text{-}fun\text{-}commute\ (\lambda x\ A.\ A \cup Set.insert\ x$
$`\ A)$
⟨*proof*⟩

**lemma** *Pow-fold*:
**assumes** *finite A*
**shows** $Pow\ A = fold\ (\lambda x\ A.\ A \cup Set.insert\ x\ `\ A)\ \{\{\}\}\ A$
⟨*proof*⟩

**lemma** *fold-union-pair*:
**assumes** *finite B*
**shows** $(\bigcup y \in B.\ \{(x,\ y)\}) \cup A = fold\ (\lambda y.\ Set.insert\ (x,\ y))\ A\ B$
⟨*proof*⟩

**lemma** *comp-fun-commute-product-fold*:
*finite* $B \Longrightarrow$ *comp-fun-commute* $(\lambda x\ z.\ fold\ (\lambda y.\ Set.insert\ (x,\ y))\ z\ B)$
⟨*proof*⟩

**lemma** *product-fold*:
**assumes** *finite A finite B*
**shows** $A \times B = fold\ (\lambda x\ z.\ fold\ (\lambda y.\ Set.insert\ (x,\ y))\ z\ B)\ \{\}\ A$
⟨*proof*⟩

**context** *complete-lattice*
**begin**

**lemma** *inf-Inf-fold-inf*:
**assumes** *finite A*
**shows** *inf* (*Inf A*) $B =$ *fold inf B A*
⟨*proof*⟩

**lemma** *sup-Sup-fold-sup*:
**assumes** *finite A*
**shows** *sup* (*Sup A*) $B =$ *fold sup B A*
⟨*proof*⟩

**lemma** *Inf-fold-inf*: *finite A* $\Longrightarrow$ *Inf A* $=$ *fold inf top A*
⟨*proof*⟩

**lemma** *Sup-fold-sup*: *finite A* $\Longrightarrow$ *Sup A* $=$ *fold sup bot A*
⟨*proof*⟩

**lemma** *inf-INF-fold-inf*:
**assumes** *finite A*
**shows** *inf B* $(\bigsqcap(f\ `\ A)) = fold\ (inf \circ f)\ B\ A$ (**is** *?inf* $=$ *?fold*)
⟨*proof*⟩

**lemma** *sup-SUP-fold-sup*:
**assumes** *finite A*
**shows** *sup B* $(\bigsqcup(f\ `\ A)) = fold\ (sup \circ f)\ B\ A$ (**is** *?sup* $=$ *?fold*)
⟨*proof*⟩

**lemma** *INF-fold-inf*: *finite A* $\Longrightarrow \bigsqcap(f\ `\ A) = fold\ (inf \circ f)\ top\ A$
⟨*proof*⟩

**lemma** *SUP-fold-sup*: *finite A* $\Longrightarrow \bigsqcup(f\ `\ A) = fold\ (sup \circ f)\ bot\ A$
⟨*proof*⟩

**lemma** *finite-Inf-in*:
**assumes** *finite A* $A \neq \{\}$ **and** *inf*: $\bigwedge x\ y.\ [\![x \in A;\ y \in A]\!] \Longrightarrow inf\ x\ y \in A$
**shows** *Inf A* $\in A$
⟨*proof*⟩

**lemma** *finite-Sup-in*:
**assumes** $finite\ A\ A \neq \{\}$ **and** $sup$: $\bigwedge x\ y.\ [\![x \in A;\ y \in A]\!] \Longrightarrow sup\ x\ y \in A$
**shows** $Sup\ A \in A$
⟨*proof*⟩

**end**

### 20.5.5 Expressing relation operations via *fold*

**lemma** *Id-on-fold*:
**assumes** *finite A*
**shows** $Id\text{-}on\ A = Finite\text{-}Set.fold\ (\lambda x.\ Set.insert\ (Pair\ x\ x))\ \{\}\ A$
⟨*proof*⟩

**lemma** *comp-fun-commute-Image-fold*:
$comp\text{-}fun\text{-}commute\ (\lambda(x,y)\ A.\ if\ x \in S\ then\ Set.insert\ y\ A\ else\ A)$
⟨*proof*⟩

**lemma** *Image-fold*:
**assumes** *finite R*
**shows** $R\ ``\ S = Finite\text{-}Set.fold\ (\lambda(x,y)\ A.\ if\ x \in S\ then\ Set.insert\ y\ A\ else\ A)$ $\{\}\ R$
⟨*proof*⟩

**lemma** *insert-relcomp-union-fold*:
**assumes** *finite S*
**shows** $\{x\}\ O\ S \cup X = Finite\text{-}Set.fold\ (\lambda(w,z)\ A'.\ if\ snd\ x = w\ then\ Set.insert$ $(fst\ x,z)\ A'\ else\ A')\ X\ S$
⟨*proof*⟩

**lemma** *insert-relcomp-fold*:
**assumes** *finite S*
**shows** $Set.insert\ x\ R\ O\ S =$
$Finite\text{-}Set.fold\ (\lambda(w,z)\ A'.\ if\ snd\ x = w\ then\ Set.insert\ (fst\ x,z)\ A'\ else\ A')\ (R$ $O\ S)\ S$
⟨*proof*⟩

**lemma** *comp-fun-commute-relcomp-fold*:
**assumes** *finite S*
**shows** $comp\text{-}fun\text{-}commute\ (\lambda(x,y)\ A.$
$Finite\text{-}Set.fold\ (\lambda(w,z)\ A'.\ if\ y = w\ then\ Set.insert\ (x,z)\ A'\ else\ A')\ A\ S)$
⟨*proof*⟩

**lemma** *relcomp-fold*:
**assumes** *finite R finite S*
**shows** $R\ O\ S = Finite\text{-}Set.fold$
$(\lambda(x,y)\ A.\ Finite\text{-}Set.fold\ (\lambda(w,z)\ A'.\ if\ y = w\ then\ Set.insert\ (x,z)\ A'\ else\ A')$ $A\ S)\ \{\}\ R$

$\langle proof \rangle$

## 20.6 Locales as mini-packages for fold operations

### 20.6.1 The natural case

**locale** *folding-on* =
  **fixes** $S$ :: $'a\ set$
  **fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'b$ **and** $z$ :: $'b$
  **assumes** *comp-fun-commute-on*: $x \in S \Longrightarrow y \in S \Longrightarrow f\ y\ o\ f\ x = f\ x\ o\ f\ y$
**begin**

**interpretation** *fold?*: *comp-fun-commute-on S f*
  $\langle proof \rangle$

**definition** $F$ :: $'a\ set \Rightarrow 'b$
  **where** *eq-fold*: $F\ A = Finite\text{-}Set.fold\ f\ z\ A$

**lemma** *empty* [*simp*]: $F\ \{\} = z$
  $\langle proof \rangle$

**lemma** *infinite* [*simp*]: $\neg\ finite\ A \Longrightarrow F\ A = z$
  $\langle proof \rangle$

**lemma** *insert* [*simp*]:
  **assumes** $insert\ x\ A \subseteq S$ **and** $finite\ A$ **and** $x \notin A$
  **shows** $F\ (insert\ x\ A) = f\ x\ (F\ A)$
$\langle proof \rangle$

**lemma** *remove*:
  **assumes** $A \subseteq S$ **and** $finite\ A$ **and** $x \in A$
  **shows** $F\ A = f\ x\ (F\ (A - \{x\}))$
$\langle proof \rangle$

**lemma** *insert-remove*:
  **assumes** $insert\ x\ A \subseteq S$ **and** $finite\ A$
  **shows** $F\ (insert\ x\ A) = f\ x\ (F\ (A - \{x\}))$
  $\langle proof \rangle$

**end**

### 20.6.2 With idempotency

**locale** *folding-idem-on* = *folding-on* +
  **assumes** *comp-fun-idem-on*: $x \in S \Longrightarrow y \in S \Longrightarrow f\ x \circ f\ x = f\ x$
**begin**

**declare** *insert* [*simp del*]

**interpretation** *fold?*: *comp-fun-idem-on S f*

⟨*proof*⟩

**lemma** *insert-idem* [*simp*]:
**assumes** *insert x A* $\subseteq$ *S* **and** *finite A*
**shows** *F* (*insert x A*) = *f x* (*F A*)
⟨*proof*⟩

**end**

### 20.6.3 *UNIV* as the carrier set

**locale** *folding* =
**fixes** *f* :: $'a \Rightarrow 'b \Rightarrow 'b$ **and** *z* :: $'b$
**assumes** *comp-fun-commute*: $f\ y \circ f\ x = f\ x \circ f\ y$
**begin**

**lemma** (**in** −) *folding-def'*: *folding f* = *folding-on UNIV f*
⟨*proof*⟩

Again, we abuse the *rewrites* functionality of locales to remove trivial assumptions that result from instantiating the carrier set to *UNIV*.

**sublocale** *folding-on UNIV f*
**rewrites** $\bigwedge X.\ (X \subseteq \mathit{UNIV}) \equiv \mathit{True}$
**and** $\bigwedge x.\ x \in \mathit{UNIV} \equiv \mathit{True}$
**and** $\bigwedge P.\ (\mathit{True} \Longrightarrow P) \equiv \mathit{Trueprop}\ P$
**and** $\bigwedge P\ Q.\ (\mathit{True} \Longrightarrow \mathit{PROP}\ P \Longrightarrow \mathit{PROP}\ Q) \equiv (\mathit{PROP}\ P \Longrightarrow \mathit{True} \Longrightarrow \mathit{PROP}\ Q)$
⟨*proof*⟩

**end**

**locale** *folding-idem* = *folding* +
**assumes** *comp-fun-idem*: $f\ x \circ f\ x = f\ x$
**begin**

**lemma** (**in** −) *folding-idem-def'*: *folding-idem f* = *folding-idem-on UNIV f*
⟨*proof*⟩

Again, we abuse the *rewrites* functionality of locales to remove trivial assumptions that result from instantiating the carrier set to *UNIV*.

**sublocale** *folding-idem-on UNIV f*
**rewrites** $\bigwedge X.\ (X \subseteq \mathit{UNIV}) \equiv \mathit{True}$
**and** $\bigwedge x.\ x \in \mathit{UNIV} \equiv \mathit{True}$
**and** $\bigwedge P.\ (\mathit{True} \Longrightarrow P) \equiv \mathit{Trueprop}\ P$
**and** $\bigwedge P\ Q.\ (\mathit{True} \Longrightarrow \mathit{PROP}\ P \Longrightarrow \mathit{PROP}\ Q) \equiv (\mathit{PROP}\ P \Longrightarrow \mathit{True} \Longrightarrow \mathit{PROP}\ Q)$
⟨*proof*⟩

**end**

## 20.7 Finite cardinality

The traditional definition *card* $A \equiv LEAST\ n.\ \exists f.\ A = \{f\ i\ |i.\ i < n\}$ is ugly to work with. But now that we have *fold* things are easy:

**global-interpretation** *card*: *folding* $\lambda$-. *Suc 0*
**defines** *card* = *folding-on.F* ($\lambda$-. *Suc*) *0*
⟨*proof*⟩

**lemma** *card-insert-disjoint*: *finite* $A \Longrightarrow x \notin A \Longrightarrow$ *card* (*insert* $x$ $A$) = *Suc* (*card* $A$)
⟨*proof*⟩

**lemma** *card-insert-if*: *finite* $A \Longrightarrow$ *card* (*insert* $x$ $A$) = (*if* $x \in A$ *then card* $A$ *else Suc* (*card* $A$))
⟨*proof*⟩

**lemma** *card-ge-0-finite*: *card* $A > 0 \Longrightarrow$ *finite* $A$
⟨*proof*⟩

**lemma** *card-0-eq* [*simp*]: *finite* $A \Longrightarrow$ *card* $A = 0 \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *finite-UNIV-card-ge-0*: *finite* (*UNIV* :: *'a set*) $\Longrightarrow$ *card* (*UNIV* :: *'a set*) $> 0$
⟨*proof*⟩

**lemma** *card-eq-0-iff*: *card* $A = 0 \longleftrightarrow A = \{\} \vee \neg$ *finite* $A$
⟨*proof*⟩

**lemma** *card-range-greater-zero*: *finite* (*range* $f$) $\Longrightarrow$ *card* (*range* $f$) $> 0$
⟨*proof*⟩

**lemma** *card-gt-0-iff*: $0 <$ *card* $A \longleftrightarrow A \neq \{\} \wedge$ *finite* $A$
⟨*proof*⟩

**lemma** *card-Suc-Diff1*:
**assumes** *finite* $A$ $x \in A$ **shows** *Suc* (*card* $(A - \{x\})$) = *card* $A$
⟨*proof*⟩

**lemma** *card-insert-le-m1*:
**assumes** $n > 0$ *card* $y \leq n - 1$ **shows** *card* (*insert* $x$ $y$) $\leq n$
⟨*proof*⟩

**lemma** *card-Diff-singleton*:
**assumes** $x \in A$ **shows** *card* $(A - \{x\})$ = *card* $A - 1$
⟨*proof*⟩

**lemma** *card-Diff-singleton-if*:
*card* $(A - \{x\})$ = (*if* $x \in A$ *then card* $A - 1$ *else card* $A$)

⟨*proof*⟩

**lemma** *card-Diff-insert*[*simp*]:
**assumes** $a \in A$ **and** $a \notin B$
**shows** $card\ (A - insert\ a\ B) = card\ (A - B) - 1$
⟨*proof*⟩

**lemma** *card-insert-le*: $card\ A \leq card\ (insert\ x\ A)$
⟨*proof*⟩

**lemma** *card-Collect-less-nat*[*simp*]: $card\ \{i::nat.\ i < n\} = n$
⟨*proof*⟩

**lemma** *card-Collect-le-nat*[*simp*]: $card\ \{i::nat.\ i \leq n\} = Suc\ n$
⟨*proof*⟩

**lemma** *card-mono*:
**assumes** *finite B* **and** $A \subseteq B$
**shows** $card\ A \leq card\ B$
⟨*proof*⟩

**lemma** *card-seteq*:
**assumes** *finite B* **and** $A$: $A \subseteq B$ $card\ B \leq card\ A$
**shows** $A = B$
⟨*proof*⟩

**lemma** *psubset-card-mono*: $finite\ B \Longrightarrow A < B \Longrightarrow card\ A < card\ B$
⟨*proof*⟩

**lemma** *card-Un-Int*:
**assumes** *finite A finite B*
**shows** $card\ A + card\ B = card\ (A \cup B) + card\ (A \cap B)$
⟨*proof*⟩

**lemma** *card-Un-disjoint*: $finite\ A \Longrightarrow finite\ B \Longrightarrow A \cap B = \{\} \Longrightarrow card\ (A \cup B) = card\ A + card\ B$
⟨*proof*⟩

**lemma** *card-Un-disjnt*: $[\![finite\ A;\ finite\ B;\ disjnt\ A\ B]\!] \Longrightarrow card\ (A \cup B) = card\ A + card\ B$
⟨*proof*⟩

**lemma** *card-Un-le*: $card\ (A \cup B) \leq card\ A + card\ B$
⟨*proof*⟩

**lemma** *card-Diff-subset*:
**assumes** *finite B*
**and** $B \subseteq A$
**shows** $card\ (A - B) = card\ A - card\ B$

⟨*proof*⟩

**lemma** *card-Diff-subset-Int*:
**assumes** *finite* $(A \cap B)$
**shows** *card* $(A - B) =$ *card* $A -$ *card* $(A \cap B)$
⟨*proof*⟩

**lemma** *card-Int-Diff*:
**assumes** *finite* $A$
**shows** *card* $A =$ *card* $(A \cap B) +$ *card* $(A - B)$
⟨*proof*⟩

**lemma** *diff-card-le-card-Diff*:
**assumes** *finite* $B$
**shows** *card* $A -$ *card* $B \leq$ *card* $(A - B)$
⟨*proof*⟩

**lemma** *card-le-sym-Diff*:
**assumes** *finite* $A$ *finite* $B$ *card* $A \leq$ *card* $B$
**shows** *card*$(A - B) \leq$ *card*$(B - A)$
⟨*proof*⟩

**lemma** *card-less-sym-Diff*:
**assumes** *finite* $A$ *finite* $B$ *card* $A <$ *card* $B$
**shows** *card*$(A - B) <$ *card*$(B - A)$
⟨*proof*⟩

**lemma** *card-Diff1-less-iff*: *card* $(A - \{x\}) <$ *card* $A \longleftrightarrow$ *finite* $A \wedge x \in A$
⟨*proof*⟩

**lemma** *card-Diff1-less*: *finite* $A \Longrightarrow x \in A \Longrightarrow$ *card* $(A - \{x\}) <$ *card* $A$
⟨*proof*⟩

**lemma** *card-Diff2-less*:
**assumes** *finite* $A$ $x \in A$ $y \in A$ **shows** *card* $(A - \{x\} - \{y\}) <$ *card* $A$
⟨*proof*⟩

**lemma** *card-Diff1-le*: *card* $(A - \{x\}) \leq$ *card* $A$
⟨*proof*⟩

**lemma** *card-psubset*: *finite* $B \Longrightarrow A \subseteq B \Longrightarrow$ *card* $A <$ *card* $B \Longrightarrow A < B$
⟨*proof*⟩

**lemma** *card-le-inj*:
**assumes** *fA*: *finite* $A$
**and** *fB*: *finite* $B$
**and** *c*: *card* $A \leq$ *card* $B$
**shows** $\exists f.\ f$ ‘ $A \subseteq B \wedge$ *inj-on* $f$ $A$
⟨*proof*⟩

**lemma** *card-subset-eq*:
**assumes** *fB*: *finite B*
**and** *AB*: $A \subseteq B$
**and** *c*: *card* $A$ = *card* $B$
**shows** $A = B$
⟨*proof*⟩

**lemma** *insert-partition*:
$x \notin F \Longrightarrow \forall c1 \in insert\ x\ F.\ \forall c2 \in insert\ x\ F.\ c1 \neq c2 \longrightarrow c1 \cap c2 = \{\} \Longrightarrow x \cap \bigcup F = \{\}$
⟨*proof*⟩

**lemma** *finite-psubset-induct* [*consumes 1*, *case-names psubset*]:
**assumes** *finite*: *finite A*
**and** *major*: $\bigwedge A.\ finite\ A \Longrightarrow (\bigwedge B.\ B \subset A \Longrightarrow P\ B) \Longrightarrow P\ A$
**shows** $P\ A$
⟨*proof*⟩

**lemma** *finite-induct-select* [*consumes 1*, *case-names empty select*]:
**assumes** *finite S*
**and** $P\ \{\}$
**and** *select*: $\bigwedge T.\ T \subset S \Longrightarrow P\ T \Longrightarrow \exists s \in S - T.\ P\ (insert\ s\ T)$
**shows** $P\ S$
⟨*proof*⟩

**lemma** *remove-induct* [*case-names empty infinite remove*]:
**assumes** *empty*: $P\ (\{\} :: 'a\ set)$
**and** *infinite*: $\neg\ finite\ B \Longrightarrow P\ B$
**and** *remove*: $\bigwedge A.\ finite\ A \Longrightarrow A \neq \{\} \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow P\ (A - \{x\})) \Longrightarrow P\ A$
**shows** $P\ B$
⟨*proof*⟩

**lemma** *finite-remove-induct* [*consumes 1*, *case-names empty remove*]:
**fixes** $P :: 'a\ set \Rightarrow bool$
**assumes** *finite B*
**and** $P\ \{\}$
**and** $\bigwedge A.\ finite\ A \Longrightarrow A \neq \{\} \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow P\ (A - \{x\})) \Longrightarrow P\ A$
**defines** $B' \equiv B$
**shows** $P\ B'$
⟨*proof*⟩

Main cardinality theorem.

**lemma** *card-partition* [*rule-format*]:
$finite\ C \Longrightarrow finite\ (\bigcup C) \Longrightarrow (\forall c \in C.\ card\ c = k) \Longrightarrow$
$(\forall c1 \in C.\ \forall c2 \in C.\ c1 \neq c2 \longrightarrow c1 \cap c2 = \{\}) \Longrightarrow$
$k * card\ C = card\ (\bigcup C)$

⟨proof⟩

**lemma** *card-eq-UNIV-imp-eq-UNIV*:
**assumes** *fin*: $finite\ (UNIV :: 'a\ set)$
**and** *card*: $card\ A = card\ (UNIV :: 'a\ set)$
**shows** $A = (UNIV :: 'a\ set)$
⟨proof⟩

The form of a finite set of given cardinality

**lemma** *card-eq-SucD*:
**assumes** $card\ A = Suc\ k$
**shows** $\exists\, b\ B.\ A = insert\ b\ B \wedge b \notin B \wedge card\ B = k \wedge (k = 0 \longrightarrow B = \{\})$
⟨proof⟩

**lemma** *card-Suc-eq*:
$card\ A = Suc\ k \longleftrightarrow$
$(\exists\, b\ B.\ A = insert\ b\ B \wedge b \notin B \wedge card\ B = k \wedge (k = 0 \longrightarrow B = \{\}))$
⟨proof⟩

**lemma** *card-Suc-eq-finite*:
$card\ A = Suc\ k \longleftrightarrow (\exists\, b\ B.\ A = insert\ b\ B \wedge b \notin B \wedge card\ B = k \wedge finite\ B)$
⟨proof⟩

**lemma** *card-1-singletonE*:
**assumes** $card\ A = 1$
**obtains** $x$ **where** $A = \{x\}$
⟨proof⟩

**lemma** *is-singleton-iff-card-eq-Suc-0* [*code*]:
‹$is\text{-}singleton\ A \longleftrightarrow card\ A = Suc\ 0$›
⟨proof⟩

**lemma** *is-singleton-altdef*:
‹$is\text{-}singleton\ A \longleftrightarrow card\ A = 1$›
⟨proof⟩

**lemma** *card-eq-Suc-0-iff-is-singleton*:
‹$card\ A = Suc\ 0 \longleftrightarrow is\text{-}singleton\ A$›
⟨proof⟩

**lemma** *card-1-singleton-iff*:
‹$card\ A = Suc\ 0 \longleftrightarrow (\exists\, x.\ A = \{x\})$›
⟨proof⟩

**lemma** *card-le-Suc0-iff-eq*:
**assumes** $finite\ A$
**shows** $card\ A \leq Suc\ 0 \longleftrightarrow (\forall\, a1 \in A.\ \forall\, a2 \in A.\ a1 = a2)$ (**is** $?C = ?A$)
⟨proof⟩

**lemma** *card-le-Suc-iff*:
$Suc\ n \leq card\ A = (\exists\, a\ B.\ A = insert\ a\ B \wedge a \notin B \wedge n \leq card\ B \wedge finite\ B)$
⟨*proof*⟩

**lemma** *finite-fun-UNIVD2*:
**assumes** *fin*: $finite\ (UNIV :: ('a \Rightarrow 'b)\ set)$
**shows** $finite\ (UNIV :: 'b\ set)$
⟨*proof*⟩

**lemma** *card-UNIV-unit* [*simp*]: $card\ (UNIV :: unit\ set) = 1$
⟨*proof*⟩

**lemma** *infinite-arbitrarily-large*:
**assumes** $\neg\ finite\ A$
**shows** $\exists\, B.\ finite\ B \wedge card\ B = n \wedge B \subseteq A$
⟨*proof*⟩

**corollary** *finite-arbitrarily-large-disj*:
$[\![\ \neg\ finite(UNIV::'a\ set);\ finite\ (A::'a\ set)\ ]\!] \Longrightarrow \exists\, B.\ finite\ B \wedge card\ B = n \wedge A \cap B = \{\}$
⟨*proof*⟩

Sometimes, to prove that a set is finite, it is convenient to work with finite subsets and to show that their cardinalities are uniformly bounded. This possibility is formalized in the next criterion.

**lemma** *finite-if-finite-subsets-card-bdd*:
**assumes** $\bigwedge G.\ G \subseteq F \Longrightarrow finite\ G \Longrightarrow card\ G \leq C$
**shows** $finite\ F \wedge card\ F \leq C$
⟨*proof*⟩

**lemma** *obtain-subset-with-card-n*:
**assumes** $n \leq card\ S$
**obtains** $T$ **where** $T \subseteq S$ $card\ T = n$ $finite\ T$
⟨*proof*⟩

**lemma** *exists-subset-between*:
**assumes**
$card\ A \leq n$
$n \leq card\ C$
$A \subseteq C$
$finite\ C$
**shows** $\exists\, B.\ A \subseteq B \wedge B \subseteq C \wedge card\ B = n$
⟨*proof*⟩

### 20.7.1 Cardinality of image

**lemma** *card-image-le*: $finite\ A \Longrightarrow card\ (f\ `\ A) \leq card\ A$
⟨*proof*⟩

**lemma** *card-image*: $inj\text{-}on\ f\ A \Longrightarrow card\ (f\ `\ A) = card\ A$
⟨*proof*⟩

**lemma** *bij-betw-same-card*: $bij\text{-}betw\ f\ A\ B \Longrightarrow card\ A = card\ B$
⟨*proof*⟩

**lemma** *endo-inj-surj*: $finite\ A \Longrightarrow f\ `\ A \subseteq A \Longrightarrow inj\text{-}on\ f\ A \Longrightarrow f\ `\ A = A$
⟨*proof*⟩

**lemma** *eq-card-imp-inj-on*:
**assumes** $finite\ A\ card(f\ `\ A) = card\ A$
**shows** $inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *inj-on-iff-eq-card*: $finite\ A \Longrightarrow inj\text{-}on\ f\ A \longleftrightarrow card\ (f\ `\ A) = card\ A$
⟨*proof*⟩

**lemma** *card-inj-on-le*:
**assumes** $inj\text{-}on\ f\ A\ f\ `\ A \subseteq B\ finite\ B$
**shows** $card\ A \leq card\ B$
⟨*proof*⟩

**lemma** *inj-on-iff-card-le*:
$[\![\ finite\ A;\ finite\ B\ ]\!] \Longrightarrow (\exists f.\ inj\text{-}on\ f\ A \wedge f\ `\ A \leq B) = (card\ A \leq card\ B)$
⟨*proof*⟩

**lemma** *surj-card-le*: $finite\ A \Longrightarrow B \subseteq f\ `\ A \Longrightarrow card\ B \leq card\ A$
⟨*proof*⟩

**lemma** *card-bij-eq*:
$inj\text{-}on\ f\ A \Longrightarrow f\ `\ A \subseteq B \Longrightarrow inj\text{-}on\ g\ B \Longrightarrow g\ `\ B \subseteq A \Longrightarrow finite\ A \Longrightarrow finite\ B$
$\Longrightarrow card\ A = card\ B$
⟨*proof*⟩

**lemma** *bij-betw-finite*: $bij\text{-}betw\ f\ A\ B \Longrightarrow finite\ A \longleftrightarrow finite\ B$
⟨*proof*⟩

**lemma** *inj-on-finite*: $inj\text{-}on\ f\ A \Longrightarrow f\ `\ A \leq B \Longrightarrow finite\ B \Longrightarrow finite\ A$
⟨*proof*⟩

**lemma** *card-vimage-inj-on-le*:
**assumes** $inj\text{-}on\ f\ D\ finite\ A$
**shows** $card\ (f-`A \cap D) \leq card\ A$
⟨*proof*⟩

**lemma** *card-vimage-inj*: $inj\ f \Longrightarrow A \subseteq range\ f \Longrightarrow card\ (f\ -`\ A) = card\ A$
⟨*proof*⟩

**lemma** *card-inverse*[*simp*]: $card\ (R^{-1}) = card\ R$

⟨*proof*⟩

### 20.7.2 Pigeonhole Principles

**lemma** *pigeonhole*: $card\ A > card\ (f\ `\ A) \Longrightarrow \neg\ inj\text{-}on\ f\ A$
⟨*proof*⟩

**lemma** *pigeonhole-infinite*:
**assumes** $\neg\ finite\ A$ **and** $finite\ (f`A)$
**shows** $\exists\, a0{\in}A.\ \neg\ finite\ \{a{\in}A.\ f\ a = f\ a0\}$
⟨*proof*⟩

**lemma** *pigeonhole-infinite-rel*:
**assumes** $\neg\ finite\ A$
**and** $finite\ B$
**and** $\forall\, a{\in}A.\ \exists\, b{\in}B.\ R\ a\ b$
**shows** $\exists\, b{\in}B.\ \neg\ finite\ \{a{:}A.\ R\ a\ b\}$
⟨*proof*⟩

### 20.7.3 Cardinality of sums

**lemma** *card-Plus*:
**assumes** $finite\ A\ finite\ B$
**shows** $card\ (A <+> B) = card\ A + card\ B$
⟨*proof*⟩

**lemma** *card-Plus-conv-if*:
$card\ (A <+> B) = (if\ finite\ A \wedge finite\ B\ then\ card\ A + card\ B\ else\ 0)$
⟨*proof*⟩

Relates to equivalence classes. Based on a theorem of F. Kammüller.

**lemma** *dvd-partition*:
**assumes** $f$: $finite\ (\bigcup C)$
**and** $\forall\, c{\in}C.\ k\ dvd\ card\ c\ \forall\, c1{\in}C.\ \forall\, c2{\in}C.\ c1 \neq c2 \longrightarrow c1 \cap c2 = \{\}$
**shows** $k\ dvd\ card\ (\bigcup C)$
⟨*proof*⟩

## 20.8 Minimal and maximal elements of finite sets

**context begin**

**qualified lemma**
**assumes** $finite\ A$ **and** $asymp\text{-}on\ A\ R$ **and** $transp\text{-}on\ A\ R$ **and** $\exists\, x \in A.\ P\ x$
**shows**
*bex-min-element-with-property*: $\exists\, x \in A.\ P\ x \wedge (\forall\, y \in A.\ R\ y\ x \longrightarrow \neg\ P\ y)$ **and**
*bex-max-element-with-property*: $\exists\, x \in A.\ P\ x \wedge (\forall\, y \in A.\ R\ x\ y \longrightarrow \neg\ P\ y)$
⟨*proof*⟩ **lemma**
**assumes** $finite\ A$ **and** $asymp\text{-}on\ A\ R$ **and** $transp\text{-}on\ A\ R$ **and** $A \neq \{\}$
**shows**
*bex-min-element*: $\exists\, m \in A.\ \forall\, x \in A.\ x \neq m \longrightarrow \neg\ R\ x\ m$ **and**

*bex-max-element*: $\exists\, m \in A.\ \forall\, x \in A.\ x \neq m \longrightarrow \neg\ R\ m\ x$
⟨*proof*⟩

**end**

The following alternative form might sometimes be easier to work with.

**lemma** *is-min-element-in-set-iff*:
*asymp-on* $A\ R \Longrightarrow (\forall\, y \in A.\ y \neq x \longrightarrow \neg\ R\ y\ x) \longleftrightarrow (\forall\, y.\ R\ y\ x \longrightarrow y \notin A)$
⟨*proof*⟩

**lemma** *is-max-element-in-set-iff*:
*asymp-on* $A\ R \Longrightarrow (\forall\, y \in A.\ y \neq x \longrightarrow \neg\ R\ x\ y) \longleftrightarrow (\forall\, y.\ R\ x\ y \longrightarrow y \notin A)$
⟨*proof*⟩

**context begin**

**qualified lemma**
**assumes** *finite* $A$ **and** $A \neq \{\}$ **and** *transp-on* $A\ R$ **and** *totalp-on* $A\ R$
**shows**
*bex-least-element*: $\exists\, l \in A.\ \forall\, x \in A.\ x \neq l \longrightarrow R\ l\ x$ **and**
*bex-greatest-element*: $\exists\, g \in A.\ \forall\, x \in A.\ x \neq g \longrightarrow R\ x\ g$
⟨*proof*⟩

**end**

### 20.8.1 Finite orders

**context** *order*
**begin**

**lemma** *finite-has-maximal*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $\exists\ m \in A.\ \forall\ b \in A.\ m \leq b \longrightarrow m = b$
⟨*proof*⟩

**lemma** *finite-has-maximal2*:
$[\![$ *finite* $A;\ a \in A\ ]\!] \Longrightarrow \exists\ m \in A.\ a \leq m \wedge (\forall\ b \in A.\ m \leq b \longrightarrow m = b)$
⟨*proof*⟩

**lemma** *finite-has-minimal*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $\exists\ m \in A.\ \forall\ b \in A.\ b \leq m \longrightarrow m = b$
⟨*proof*⟩

**lemma** *finite-has-minimal2*:
$[\![$ *finite* $A;\ a \in A\ ]\!] \Longrightarrow \exists\ m \in A.\ m \leq a \wedge (\forall\ b \in A.\ b \leq m \longrightarrow m = b)$
⟨*proof*⟩

**end**

### 20.8.2 Relating injectivity and surjectivity

**lemma** *finite-surj-inj*:
  **assumes** *finite A* $A \subseteq f$ ‘ $A$
  **shows** *inj-on f A*
⟨*proof*⟩

**lemma** *finite-UNIV-surj-inj*: *finite*(*UNIV*:: *'a set*) $\Longrightarrow$ *surj f* $\Longrightarrow$ *inj f*
  **for** $f$ :: $'a \Rightarrow 'a$
  ⟨*proof*⟩

**lemma** *finite-UNIV-inj-surj*: *finite*(*UNIV*:: *'a set*) $\Longrightarrow$ *inj f* $\Longrightarrow$ *surj f*
  **for** $f$ :: $'a \Rightarrow 'a$
  ⟨*proof*⟩

**lemma** *surjective-iff-injective-gen*:
  **assumes** *fS*: *finite S*
    **and** *fT*: *finite T*
    **and** *c*: *card S* = *card T*
    **and** *ST*: $f$ ‘ $S \subseteq T$
  **shows** $(\forall\, y \in T.\ \exists\, x \in S.\ f\ x = y) \longleftrightarrow$ *inj-on f S*
  (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**hide-const** (**open**) *Finite-Set.fold*

## 20.9 Infinite Sets

Some elementary facts about infinite sets, mostly by Stephan Merz. Beware! Because "infinite" merely abbreviates a negation, these lemmas may not work well with *blast*.

**abbreviation** *infinite* :: *'a set* $\Rightarrow$ *bool*
  **where** *infinite S* $\equiv \neg$ *finite S*

Infinite sets are non-empty, and if we remove some elements from an infinite set, the result is still infinite.

**lemma** *infinite-UNIV-nat* [*iff*]: *infinite* (*UNIV* :: *nat set*)
⟨*proof*⟩

**lemma** *infinite-UNIV-char-0*: *infinite* (*UNIV* :: *'a*::*semiring-char-0 set*)
⟨*proof*⟩

**lemma** *infinite-imp-nonempty*: *infinite S* $\Longrightarrow S \neq \{\}$
  ⟨*proof*⟩

**lemma** *infinite-remove*: *infinite S* $\Longrightarrow$ *infinite* $(S - \{a\})$
  ⟨*proof*⟩

**lemma** *Diff-infinite-finite*:

**assumes** *finite* $T$ *infinite* $S$
**shows** *infinite* $(S - T)$
⟨*proof*⟩

**lemma** *Un-infinite*: *infinite* $S \Longrightarrow$ *infinite* $(S \cup T)$
⟨*proof*⟩

**lemma** *infinite-Un*: *infinite* $(S \cup T) \longleftrightarrow$ *infinite* $S \vee$ *infinite* $T$
⟨*proof*⟩

**lemma** *infinite-super*:
**assumes** $S \subseteq T$
**and** *infinite* $S$
**shows** *infinite* $T$
⟨*proof*⟩

**proposition** *infinite-coinduct* [*consumes 1*, *case-names infinite*]:
**assumes** $X\ A$
**and** *step*: $\bigwedge A.\ X\ A \Longrightarrow \exists x{\in}A.\ X\ (A - \{x\}) \vee$ *infinite* $(A - \{x\})$
**shows** *infinite* $A$
⟨*proof*⟩

For any function with infinite domain and finite range there is some element that is the image of infinitely many domain elements. In particular, any infinite sequence of elements from a finite set contains some element that occurs infinitely often.

**lemma** *inf-img-fin-dom'*:
**assumes** *img*: *finite* $(f\ `\ A)$
**and** *dom*: *infinite* $A$
**shows** $\exists\, y \in f\ `\ A.$ *infinite* $(f\ -`\ \{y\} \cap A)$
⟨*proof*⟩

**lemma** *inf-img-fin-domE'*:
**assumes** *finite* $(f\ `\ A)$ **and** *infinite* $A$
**obtains** $y$ **where** $y \in f`A$ **and** *infinite* $(f\ -`\ \{y\} \cap A)$
⟨*proof*⟩

**lemma** *inf-img-fin-dom*:
**assumes** *img*: *finite* $(f`A)$ **and** *dom*: *infinite* $A$
**shows** $\exists\, y \in f`A.$ *infinite* $(f\ -`\ \{y\})$
⟨*proof*⟩

**lemma** *inf-img-fin-domE*:
**assumes** *finite* $(f`A)$ **and** *infinite* $A$
**obtains** $y$ **where** $y \in f`A$ **and** *infinite* $(f\ -`\ \{y\})$
⟨*proof*⟩

**proposition** *finite-image-absD*: *finite* $(abs\ `\ S) \Longrightarrow$ *finite* $S$
**for** $S$ :: *'a::linordered-ring set*

⟨*proof*⟩

### 20.10 The finite powerset operator

**definition** *Fpow* :: $'a\ set \Rightarrow\ 'a\ set\ set$
**where** $Fpow\ A \equiv \{X.\ X \subseteq A \wedge finite\ X\}$

**lemma** *Fpow-mono*: $A \subseteq B \Longrightarrow Fpow\ A \subseteq Fpow\ B$
⟨*proof*⟩

**lemma** *empty-in-Fpow*: $\{\} \in Fpow\ A$
⟨*proof*⟩

**lemma** *Fpow-not-empty*: $Fpow\ A \neq \{\}$
⟨*proof*⟩

**lemma** *Fpow-subset-Pow*: $Fpow\ A \subseteq Pow\ A$
⟨*proof*⟩

**lemma** *Fpow-Pow-finite*: $Fpow\ A = Pow\ A\ Int\ \{A.\ finite\ A\}$
⟨*proof*⟩

**lemma** *inj-on-image-Fpow*:
  **assumes** *inj-on f A*
  **shows** *inj-on* (*image f*) (*Fpow A*)
  ⟨*proof*⟩

**lemma** *image-Fpow-mono*:
  **assumes** $f$ ‘ $A \subseteq B$
  **shows** (*image f*) ‘ (*Fpow A*) $\subseteq$ *Fpow B*
  ⟨*proof*⟩

**end**

## 21 Reflexive and Transitive closure of a relation

**theory** *Transitive-Closure*
  **imports** *Finite-Set*
  **abbrevs** $\widehat{}{*} = {}^{*}\ {}^{**}$
    **and** $\widehat{}{+} = {}^{+}\ {}^{++}$
    **and** $\widehat{}{=} = {}^{=}\ {}^{==}$
**begin**

⟨*ML*⟩

*rtrancl* is reflexive/transitive closure, *trancl* is transitive closure, *reflcl* is reflexive closure.

These postfix operators have *maximum priority*, forcing their operands to be atomic.

**context notes** [[*inductive-internals*]]
**begin**

**inductive-set** *rtrancl* :: ($'a \times 'a$) *set* $\Rightarrow$ ($'a \times 'a$) *set* (‹(‹notation=‹postfix ∗››-$^*$)› [*1000*] *999*)
**for** *r* :: ($'a \times 'a$) *set*
**where**
*rtrancl-refl* [*intro*!, *Pure.intro*!, *simp*]: $(a, a) \in r^*$
| *rtrancl-into-rtrancl* [*Pure.intro*]: $(a, b) \in r^* \Longrightarrow (b, c) \in r \Longrightarrow (a, c) \in r^*$

**inductive-set** *trancl* :: ($'a \times 'a$) *set* $\Rightarrow$ ($'a \times 'a$) *set* (‹(‹notation=‹postfix +››-$^+$)› [*1000*] *999*)
**for** *r* :: ($'a \times 'a$) *set*
**where**
*r-into-trancl* [*intro*, *Pure.intro*]: $(a, b) \in r \Longrightarrow (a, b) \in r^+$
| *trancl-into-trancl* [*Pure.intro*]: $(a, b) \in r^+ \Longrightarrow (b, c) \in r \Longrightarrow (a, c) \in r^+$

**notation**
*rtranclp* (‹(‹notation=‹postfix ∗∗››-$^{**}$)› [*1000*] *1000*) **and**
*tranclp* (‹(‹notation=‹postfix ++››-$^{++}$)› [*1000*] *1000*)

**declare**
*rtrancl-def* [*nitpick-unfold del*]
*rtranclp-def* [*nitpick-unfold del*]
*trancl-def* [*nitpick-unfold del*]
*tranclp-def* [*nitpick-unfold del*]

**end**

**lemma** *trancl-incr*: $r \subseteq r^+$
⟨*proof*⟩

**abbreviation** *reflcl* :: ($'a \times 'a$) *set* $\Rightarrow$ ($'a \times 'a$) *set* (‹(‹notation=‹postfix =››-$^=$)› [*1000*] *999*)
**where** $r^= \equiv r \cup Id$

**abbreviation** *reflclp* :: ($'a \Rightarrow 'a \Rightarrow bool$) $\Rightarrow$ $'a \Rightarrow 'a \Rightarrow bool$ (‹(‹notation=‹postfix ==››-$^{==}$)› [*1000*] *1000*)
**where** $r^{==} \equiv sup\ r\ (=)$

**notation** (*ASCII*)
*rtrancl* (‹(‹notation=‹postfix ∗››-^∗)› [*1000*] *999*) **and**
*trancl* (‹(‹notation=‹postfix +››-^+)› [*1000*] *999*) **and**
*reflcl* (‹(‹notation=‹postfix =››-^=)› [*1000*] *999*) **and**
*rtranclp* (‹(‹notation=‹postfix ∗∗››-^∗∗)› [*1000*] *1000*) **and**
*tranclp* (‹(‹notation=‹postfix ++››-^++)› [*1000*] *1000*) **and**
*reflclp* (‹(‹notation=‹postfix ==››-^==)› [*1000*] *1000*)

**bundle** *rtrancl-syntax*

**begin**
**notation**
*rtrancl* (‹(‹*notation*=‹*postfix* ∗››-$^*$)› [*1000*] *999*) **and**
*rtranclp* (‹(‹*notation*=‹*postfix* ∗∗››-$^{**}$)› [*1000*] *1000*)
**notation** (*ASCII*)
*rtrancl* (‹(‹*notation*=‹*postfix* ∗››-^∗)› [*1000*] *999*) **and**
*rtranclp* (‹(‹*notation*=‹*postfix* ∗∗››-^∗∗)› [*1000*] *1000*)
**end**

**bundle** *trancl-syntax*
**begin**
**notation**
*trancl* (‹(‹*notation*=‹*postfix* +››-$^+$)› [*1000*] *999*) **and**
*tranclp* (‹(‹*notation*=‹*postfix* ++››-$^{++}$)› [*1000*] *1000*)
**notation** (*ASCII*)
*trancl* (‹(‹*notation*=‹*postfix* +››-^+)› [*1000*] *999*) **and**
*tranclp* (‹(‹*notation*=‹*postfix* ++››-^++)› [*1000*] *1000*)
**end**

**bundle** *reflcl-syntax*
**begin**
**notation**
*reflcl* (‹(‹*notation*=‹*postfix* =››-$^=$)› [*1000*] *999*) **and**
*reflclp* (‹(‹*notation*=‹*postfix* ==››-$^{==}$)› [*1000*] *1000*)
**notation** (*ASCII*)
*reflcl* (‹(‹*notation*=‹*postfix* =››-^=)› [*1000*] *999*) **and**
*reflclp* (‹(‹*notation*=‹*postfix* ==››-^==)› [*1000*] *1000*)
**end**

## 21.1 Reflexive closure

**lemma** *reflcl-set-eq* [*pred-set-conv*]: $(sup\ (\lambda x\ y.\ (x, y) \in r)\ (=)) = (\lambda x\ y.\ (x, y) \in r \cup Id)$
⟨*proof*⟩

**lemma** *refl-reflcl*[*simp*]: *refl* $(r^=)$
⟨*proof*⟩

**lemma** *reflp-on-reflclp*[*simp*]: *reflp-on* $A\ R^{==}$
⟨*proof*⟩

**lemma** *antisym-on-reflcl*[*simp*]: *antisym-on* $A\ (r^=) \longleftrightarrow$ *antisym-on* $A\ r$
⟨*proof*⟩

**lemma** *antisymp-on-reflclp*[*simp*]: *antisymp-on* $A\ R^{==} \longleftrightarrow$ *antisymp-on* $A\ R$
⟨*proof*⟩

**lemma** *trans-on-reflcl*[*simp*]: *trans-on* $A\ r \Longrightarrow$ *trans-on* $A\ (r^=)$
⟨*proof*⟩

**lemma** *transp-on-reflclp*[*simp*]: $transp\text{-}on\ A\ R \Longrightarrow transp\text{-}on\ A\ R^{==}$
⟨*proof*⟩

**lemma** *antisymp-on-reflclp-if-asymp-on*:
**assumes** $asymp\text{-}on\ A\ R$
**shows** $antisymp\text{-}on\ A\ R^{==}$
⟨*proof*⟩

**lemma** *antisym-on-reflcl-if-asym-on*: $asym\text{-}on\ A\ R \Longrightarrow antisym\text{-}on\ A\ (R^{=})$
⟨*proof*⟩

**lemma** *reflclp-idemp* [*simp*]: $(P^{==})^{==} = P^{==}$
⟨*proof*⟩

**lemma** *reflclp-ident-if-reflp*[*simp*]: $reflp\ R \Longrightarrow R^{==} = R$
⟨*proof*⟩

The following are special cases of *reflclp-ident-if-reflp*, but they appear duplicated in multiple, independent theories, which causes name clashes.

**lemma** (**in** *preorder*) *reflclp-less-eq*[*simp*]: $(\leq)^{==} = (\leq)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *reflclp-greater-eq*[*simp*]: $(\geq)^{==} = (\geq)$
⟨*proof*⟩

**lemma** *order-reflclp-if-transp-and-asymp*:
**assumes** $transp\ R$ **and** $asymp\ R$
**shows** $class.order\ R^{==}\ R$
⟨*proof*⟩

## 21.2 Reflexive-transitive closure

**lemma** *r-into-rtrancl* [*intro*]: $\bigwedge p.\ p \in r \Longrightarrow p \in r^*$
— *rtrancl* of $r$ contains $r$
⟨*proof*⟩

**lemma** *r-into-rtranclp* [*intro*]: $r\ x\ y \Longrightarrow r^{**}\ x\ y$
— *rtrancl* of $r$ contains $r$
⟨*proof*⟩

**lemma** *rtranclp-mono*: $r \leq s \Longrightarrow r^{**} \leq s^{**}$
— monotonicity of *rtrancl*
⟨*proof*⟩

**lemma** *mono-rtranclp*[*mono*]: $(\bigwedge a\ b.\ x\ a\ b \longrightarrow y\ a\ b) \Longrightarrow x^{**}\ a\ b \longrightarrow y^{**}\ a\ b$
⟨*proof*⟩

**lemmas** $rtrancl\text{-}mono = rtranclp\text{-}mono$ [*to-set*]

**theorem** *rtranclp-induct* [*consumes 1*, *case-names base step*, *induct set*: *rtranclp*]:
  **assumes** *a*: $r^{**}\ a\ b$
    **and** *cases*: $P\ a\ \bigwedge y\ z.\ r^{**}\ a\ y \Longrightarrow r\ y\ z \Longrightarrow P\ y \Longrightarrow P\ z$
  **shows** $P\ b$
  ⟨*proof*⟩

**lemmas** *rtrancl-induct* [*induct set*: *rtrancl*] = *rtranclp-induct* [*to-set*]

**lemmas** *rtranclp-induct2* =
  *rtranclp-induct*[*of* - (*ax*,*ay*) (*bx*,*by*), *split-rule*, *consumes 1*, *case-names refl step*]

**lemmas** *rtrancl-induct2* =
  *rtrancl-induct*[*of* (*ax*,*ay*) (*bx*,*by*), *split-format* (*complete*), *consumes 1*, *case-names refl step*]

**lemma** *refl-rtrancl*: *refl* $(r^*)$
  ⟨*proof*⟩

Transitivity of transitive closure.

**lemma** *trans-rtrancl*: *trans* $(r^*)$
⟨*proof*⟩

**lemmas** *rtrancl-trans* = *trans-rtrancl* [*THEN transD*]

**lemma** *rtranclp-trans*:
  **assumes** $r^{**}\ x\ y$
    **and** $r^{**}\ y\ z$
  **shows** $r^{**}\ x\ z$
  ⟨*proof*⟩

**lemma** *rtranclE* [*cases set*: *rtrancl*]:
  **fixes** $a\ b :: {'a}$
  **assumes** *major*: $(a,\ b) \in r^*$
  **obtains**
    (*base*) $a = b$
  | (*step*) $y$ **where** $(a,\ y) \in r^*$ **and** $(y,\ b) \in r$
  — elimination of *rtrancl* – by induction on a special formula
⟨*proof*⟩

**lemma** *rtrancl-Int-subset*: $Id \subseteq s \Longrightarrow (r^* \cap s)\ O\ r \subseteq s \Longrightarrow r^* \subseteq s$
  ⟨*proof*⟩

**lemma** *converse-rtranclp-into-rtranclp*: $r\ a\ b \Longrightarrow r^{**}\ b\ c \Longrightarrow r^{**}\ a\ c$
  ⟨*proof*⟩

**lemmas** *converse-rtrancl-into-rtrancl* = *converse-rtranclp-into-rtranclp* [*to-set*]

More $r^*$ equations and inclusions.

**lemma** *rtranclp-idemp* [*simp*]: $(r^{**})^{**} = r^{**}$
⟨*proof*⟩

**lemmas** *rtrancl-idemp* [*simp*] = *rtranclp-idemp* [*to-set*]

**lemma** *rtrancl-idemp-self-comp* [*simp*]: $R^* \ O \ R^* = R^*$
⟨*proof*⟩

**lemma** *rtrancl-subset-rtrancl*: $r \subseteq s^* \Longrightarrow r^* \subseteq s^*$
⟨*proof*⟩

**lemma** *rtranclp-subset*: $R \leq S \Longrightarrow S \leq R^{**} \Longrightarrow S^{**} = R^{**}$
⟨*proof*⟩

**lemmas** *rtrancl-subset* = *rtranclp-subset* [*to-set*]

**lemma** *rtranclp-sup-rtranclp*: $(sup\ (R^{**})\ (S^{**}))^{**} = (sup\ R\ S)^{**}$
⟨*proof*⟩

**lemmas** *rtrancl-Un-rtrancl* = *rtranclp-sup-rtranclp* [*to-set*]

**lemma** *rtranclp-reflclp* [*simp*]: $(R^{==})^{**} = R^{**}$
⟨*proof*⟩

**lemmas** *rtrancl-reflcl* [*simp*] = *rtranclp-reflclp* [*to-set*]

**lemma** *rtrancl-r-diff-Id*: $(r - Id)^* = r^*$
⟨*proof*⟩

**lemma** *rtranclp-r-diff-Id*: $(inf\ r\ (\neq))^{**} = r^{**}$
⟨*proof*⟩

**theorem** *rtranclp-converseD*:
**assumes** $(r^{-1-1})^{**}\ x\ y$
**shows** $r^{**}\ y\ x$
⟨*proof*⟩

**lemmas** *rtrancl-converseD* = *rtranclp-converseD* [*to-set*]

**theorem** *rtranclp-converseI*:
**assumes** $r^{**}\ y\ x$
**shows** $(r^{-1-1})^{**}\ x\ y$
⟨*proof*⟩

**lemmas** *rtrancl-converseI* = *rtranclp-converseI* [*to-set*]

**lemma** *rtrancl-converse*: $(r^{-1})^* = (r^*)^{-1}$
⟨*proof*⟩

**lemma** *sym-rtrancl*: $sym\ r \Longrightarrow sym\ (r^*)$
⟨*proof*⟩

**theorem** *converse-rtranclp-induct* [*consumes 1*, *case-names base step*]:
**assumes** *major*: $r^{**}\ a\ b$
**and** *cases*: $P\ b\ \bigwedge y\ z.\ r\ y\ z \Longrightarrow r^{**}\ z\ b \Longrightarrow P\ z \Longrightarrow P\ y$
**shows** $P\ a$
⟨*proof*⟩

**lemmas** *converse-rtrancl-induct* = *converse-rtranclp-induct* [*to-set*]

**lemmas** *converse-rtranclp-induct2* =
*converse-rtranclp-induct* [*of* - (*ax*, *ay*) (*bx*, *by*), *split-rule*, *consumes 1*, *case-names refl step*]

**lemmas** *converse-rtrancl-induct2* =
*converse-rtrancl-induct* [*of* (*ax*, *ay*) (*bx*, *by*), *split-format* (*complete*),
*consumes 1*, *case-names refl step*]

**lemma** *converse-rtranclpE* [*consumes 1*, *case-names base step*]:
**assumes** *major*: $r^{**}\ x\ z$
**and** *cases*: $x = z \Longrightarrow P\ \bigwedge y.\ r\ x\ y \Longrightarrow r^{**}\ y\ z \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemmas** *converse-rtranclE* = *converse-rtranclpE* [*to-set*]

**lemmas** *converse-rtranclpE2* = *converse-rtranclpE* [*of* - (*xa*,*xb*) (*za*,*zb*), *split-rule*]

**lemmas** *converse-rtranclE2* = *converse-rtranclE* [*of* (*xa*,*xb*) (*za*,*zb*), *split-rule*]

**lemma** *r-comp-rtrancl-eq*: $r\ O\ r^* = r^*\ O\ r$
⟨*proof*⟩

**lemma** *rtrancl-unfold*: $r^* = Id \cup r^*\ O\ r$
⟨*proof*⟩

**lemma** *rtrancl-Un-separatorE*:
$(a, b) \in (P \cup Q)^* \Longrightarrow \forall x\ y.\ (a, x) \in P^* \longrightarrow (x, y) \in Q \longrightarrow x = y \Longrightarrow (a, b) \in P^*$
⟨*proof*⟩

**lemma** *rtrancl-Un-separator-converseE*:
$(a, b) \in (P \cup Q)^* \Longrightarrow \forall x\ y.\ (x, b) \in P^* \longrightarrow (y, x) \in Q \longrightarrow y = x \Longrightarrow (a, b) \in P^*$
⟨*proof*⟩

**lemma** *Image-closed-trancl*:
**assumes** $r\ ``\ X \subseteq X$

**shows** $r^* `` X = X$
⟨*proof*⟩

**lemma** *rtranclp-ident-if-reflp-and-transp*:
**assumes** *reflp* $R$ **and** *transp* $R$
**shows** $R^{**} = R$
⟨*proof*⟩

The following are special cases of *rtranclp-ident-if-reflp-and-transp*, but they appear duplicated in multiple, independent theories, which causes name clashes.

**lemma** (**in** *preorder*) *rtranclp-less-eq*[*simp*]: $(\leq)^{**} = (\leq)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *rtranclp-greater-eq*[*simp*]: $(\geq)^{**} = (\geq)$
⟨*proof*⟩

## 21.3 Transitive closure

**lemma** *totalp-on-tranclp*: *totalp-on* $A$ $R \Longrightarrow$ *totalp-on* $A$ (*tranclp* $R$)
⟨*proof*⟩

**lemma** *total-on-trancl*: *total-on* $A$ $r \Longrightarrow$ *total-on* $A$ (*trancl* $r$)
⟨*proof*⟩

**lemma** *trancl-mono*:
**assumes** $p \in r^+$ $r \subseteq s$
**shows** $p \in s^+$
⟨*proof*⟩

**lemma** *trancl-mono-subset*: $A \subseteq B \Longrightarrow A\hat{}+ \subseteq B\hat{}+$
⟨*proof*⟩

**lemma** *r-into-trancl'*: $\bigwedge p.\ p \in r \Longrightarrow p \in r^+$
⟨*proof*⟩

Conversions between *trancl* and *rtrancl*.

**lemma** *tranclp-into-rtranclp*: $r^{++}\ a\ b \Longrightarrow r^{**}\ a\ b$
⟨*proof*⟩

**lemmas** *trancl-into-rtrancl* = *tranclp-into-rtranclp* [*to-set*]

**lemma** *rtranclp-into-tranclp1*:
**assumes** $r^{**}\ a\ b$
**shows** $r\ b\ c \Longrightarrow r^{++}\ a\ c$
⟨*proof*⟩

**lemmas** *rtrancl-into-trancl1* = *rtranclp-into-tranclp1* [*to-set*]

**lemma** *rtranclp-into-tranclp2*:
**assumes** $r\ a\ b$ $r^{**}\ b\ c$ **shows** $r^{++}\ a\ c$
— intro rule from $r$ and *rtrancl*
⟨*proof*⟩

**lemmas** *rtrancl-into-trancl2* = *rtranclp-into-tranclp2* [*to-set*]

Nice induction rule for *trancl*

**lemma** *tranclp-induct* [*consumes 1*, *case-names base step*, *induct pred*: *tranclp*]:
**assumes** *a*: $r^{++}\ a\ b$
**and** *cases*: $\bigwedge y.\ r\ a\ y \Longrightarrow P\ y$ $\bigwedge y\ z.\ r^{++}\ a\ y \Longrightarrow r\ y\ z \Longrightarrow P\ y \Longrightarrow P\ z$
**shows** $P\ b$
⟨*proof*⟩

**lemmas** *trancl-induct* [*induct set*: *trancl*] = *tranclp-induct* [*to-set*]

**lemmas** *tranclp-induct2* =
*tranclp-induct* [*of - (ax, ay) (bx, by)*, *split-rule*, *consumes 1*, *case-names base step*]

**lemmas** *trancl-induct2* =
*trancl-induct* [*of (ax, ay) (bx, by)*, *split-format (complete)*,
*consumes 1*, *case-names base step*]

**lemma** *tranclp-trans-induct*:
**assumes** *major*: $r^{++}\ x\ y$
**and** *cases*: $\bigwedge x\ y.\ r\ x\ y \Longrightarrow P\ x\ y$ $\bigwedge x\ y\ z.\ r^{++}\ x\ y \Longrightarrow P\ x\ y \Longrightarrow r^{++}\ y\ z \Longrightarrow P\ y\ z \Longrightarrow P\ x\ z$
**shows** $P\ x\ y$
— Another induction rule for trancl, incorporating transitivity
⟨*proof*⟩

**lemmas** *trancl-trans-induct* = *tranclp-trans-induct* [*to-set*]

**lemma** *tranclE* [*cases set*: *trancl*]:
**assumes** $(a, b) \in r^+$
**obtains**
(*base*) $(a, b) \in r$
| (*step*) $c$ **where** $(a, c) \in r^+$ **and** $(c, b) \in r$
⟨*proof*⟩

**lemma** *trancl-Int-subset*: $r \subseteq s \Longrightarrow (r^+ \cap s)\ O\ r \subseteq s \Longrightarrow r^+ \subseteq s$
⟨*proof*⟩

**lemma** *trancl-unfold*: $r^+ = r \cup r^+\ O\ r$
⟨*proof*⟩

Transitivity of $r^+$

**lemma** *trans-trancl* [*simp*]: *trans* $(r^+)$
⟨*proof*⟩

**lemmas** *trancl-trans* = *trans-trancl* [*THEN transD*]

**lemma** *tranclp-trans*:
**assumes** $r^{++}\ x\ y$
**and** $r^{++}\ y\ z$
**shows** $r^{++}\ x\ z$
⟨*proof*⟩

**lemma** *trancl-id* [*simp*]: *trans* $r \Longrightarrow r^+ = r$
⟨*proof*⟩

**lemma** *rtranclp-tranclp-tranclp*:
**assumes** $r^{**}\ x\ y$
**shows** $\bigwedge z.\ r^{++}\ y\ z \Longrightarrow r^{++}\ x\ z$
⟨*proof*⟩

**lemmas** *rtrancl-trancl-trancl* = *rtranclp-tranclp-tranclp* [*to-set*]

**lemma** *tranclp-into-tranclp2*: $r\ a\ b \Longrightarrow r^{++}\ b\ c \Longrightarrow r^{++}\ a\ c$
⟨*proof*⟩

**lemmas** *trancl-into-trancl2* = *tranclp-into-tranclp2* [*to-set*]

**lemma** *trancl-trancl-Un*: $(A\hat{\ }+ \cup B)\hat{\ }+ = (A \cup B)\hat{\ }+$
⟨*proof*⟩

**lemma** *trancl-absorb-subset-trancl*: $B \subseteq A\hat{\ }+ \Longrightarrow (A \cup B)\hat{\ }+ = A\hat{\ }+$
⟨*proof*⟩

**lemma** *tranclp-converseI*:
**assumes** $(r^{++})^{-1-1}\ x\ y$ **shows** $(r^{-1-1})^{++}\ x\ y$
⟨*proof*⟩

**lemmas** *trancl-converseI* = *tranclp-converseI* [*to-set*]

**lemma** *tranclp-converseD*:
**assumes** $(r^{-1-1})^{++}\ x\ y$ **shows** $(r^{++})^{-1-1}\ x\ y$
⟨*proof*⟩

**lemmas** *trancl-converseD* = *tranclp-converseD* [*to-set*]

**lemma** *tranclp-converse*: $(r^{-1-1})^{++} = (r^{++})^{-1-1}$
⟨*proof*⟩

**lemmas** *trancl-converse* = *tranclp-converse* [*to-set*]

**lemma** *sym-trancl*: $sym\ r \Longrightarrow sym\ (r^+)$
⟨*proof*⟩

**lemma** *converse-tranclp-induct* [*consumes 1*, *case-names base step*]:
**assumes** *major*: $r^{++}\ a\ b$
**and** *cases*: $\bigwedge y.\ r\ y\ b \Longrightarrow P\ y\ \bigwedge y\ z.\ r\ y\ z \Longrightarrow r^{++}\ z\ b \Longrightarrow P\ z \Longrightarrow P\ y$
**shows** $P\ a$
⟨*proof*⟩

**lemmas** *converse-trancl-induct* = *converse-tranclp-induct* [*to-set*]

**lemma** *tranclpD*: $R^{++}\ x\ y \Longrightarrow \exists z.\ R\ x\ z \wedge R^{**}\ z\ y$
⟨*proof*⟩

**lemmas** *tranclD* = *tranclpD* [*to-set*]

**lemma** *converse-tranclpE*:
**assumes** *major*: *tranclp* $r\ x\ z$
**and** *base*: $r\ x\ z \Longrightarrow P$
**and** *step*: $\bigwedge y.\ r\ x\ y \Longrightarrow tranclp\ r\ y\ z \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemmas** *converse-tranclE* = *converse-tranclpE* [*to-set*]

**lemma** *tranclD2*: $(x,\ y) \in R^+ \Longrightarrow \exists z.\ (x,\ z) \in R^* \wedge (z,\ y) \in R$
⟨*proof*⟩

**lemma** *irrefl-tranclI*: $r^{-1} \cap r^* = \{\} \Longrightarrow (x,\ x) \notin r^+$
⟨*proof*⟩

**lemma** *irrefl-trancl-rD*: $\forall x.\ (x,\ x) \notin r^+ \Longrightarrow (x,\ y) \in r \Longrightarrow x \neq y$
⟨*proof*⟩

**lemma** *trancl-subset-Sigma-aux*: $(a,\ b) \in r^* \Longrightarrow r \subseteq A \times A \Longrightarrow a = b \vee a \in A$
⟨*proof*⟩

**lemma** *trancl-subset-Sigma*:
**assumes** $r \subseteq A \times A$ **shows** $r^+ \subseteq A \times A$
⟨*proof*⟩

**lemma** *reflclp-tranclp* [*simp*]: $(r^{++})^{==} = r^{**}$
⟨*proof*⟩

**lemmas** *reflcl-trancl* [*simp*] = *reflclp-tranclp* [*to-set*]

**lemma** *trancl-reflcl* [*simp*]: $(r^=)^+ = r^*$
⟨*proof*⟩

**lemma** *rtrancl-trancl-reflcl* [*code*]: $r^* = (r^+)^=$
⟨*proof*⟩

**lemma** *trancl-empty* [*simp*]: $\{\}^+ = \{\}$
⟨*proof*⟩

**lemma** *rtrancl-empty* [*simp*]: $\{\}^* = Id$
⟨*proof*⟩

**lemma** *rtrancl--Id*[*simp*]: $Id^* = Id$
⟨*proof*⟩

**lemma** *rtranclpD*: $R^{**}\ a\ b \Longrightarrow a = b \vee a \neq b \wedge R^{++}\ a\ b$
⟨*proof*⟩

**lemmas** *rtranclD* = *rtranclpD* [*to-set*]

**lemma** *rtrancl-eq-or-trancl*: $(x,y) \in R^* \longleftrightarrow x = y \vee x \neq y \wedge (x,\ y) \in R^+$
⟨*proof*⟩

**lemma** *trancl-unfold-right*: $r^+ = r^*\ O\ r$
⟨*proof*⟩

**lemma** *trancl-unfold-left*: $r^+ = r\ O\ r^*$
⟨*proof*⟩

**lemma** *tranclp-unfold-left*: $r\hat{}{+}{+} = r\ OO\ r\hat{}{*}{*}$
⟨*proof*⟩

**lemma** *trancl-insert*: $(insert\ (y,\ x)\ r)^+ = r^+ \cup \{(a,\ b).\ (a,\ y) \in r^* \wedge (x,\ b) \in r^*\}$
— primitive recursion for *trancl* over finite relations
⟨*proof*⟩

**lemma** *trancl-insert2*:
$(insert\ (a,\ b)\ r)^+ = r^+ \cup \{(x,\ y).\ ((x,\ a) \in r^+ \vee x = a) \wedge ((b,\ y) \in r^+ \vee y = b)\}$
⟨*proof*⟩

**lemma** *rtrancl-insert*: $(insert\ (a,b)\ r)^* = r^* \cup \{(x,\ y).\ (x,\ a) \in r^* \wedge (b,\ y) \in r^*\}$
⟨*proof*⟩

Simplifying nested closures

**lemma** *rtrancl-trancl-absorb*[*simp*]: $(R^*)^+ = R^*$
⟨*proof*⟩

**lemma** *trancl-rtrancl-absorb*[*simp*]: $(R^+)^* = R^*$
⟨*proof*⟩

**lemma** *rtrancl-reflcl-absorb*[*simp*]: $(R^*)^= = R^*$

⟨*proof*⟩

*Domain* and *Range*

**lemma** *Domain-rtrancl* [*simp*]: $Domain\ (R^*) = UNIV$
⟨*proof*⟩

**lemma** *Range-rtrancl* [*simp*]: $Range\ (R^*) = UNIV$
⟨*proof*⟩

**lemma** *rtrancl-Un-subset*: $(R^* \cup S^*) \subseteq (R \cup S)^*$
⟨*proof*⟩

**lemma** *in-rtrancl-UnI*: $x \in R^* \vee x \in S^* \Longrightarrow x \in (R \cup S)^*$
⟨*proof*⟩

**lemma** *trancl-domain* [*simp*]: $Domain\ (r^+) = Domain\ r$
⟨*proof*⟩

**lemma** *trancl-range* [*simp*]: $Range\ (r^+) = Range\ r$
⟨*proof*⟩

**lemma** *Not-Domain-rtrancl*:
**assumes** $x \notin Domain\ R$ **shows** $(x,\ y) \in R^* \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *trancl-subset-Field2*: $r^+ \subseteq Field\ r \times Field\ r$
⟨*proof*⟩

**lemma** *finite-trancl*[*simp*]: $finite\ (r^+) = finite\ r$
⟨*proof*⟩

**lemma** *finite-rtrancl-Image*[*simp*]: **assumes** *finite R finite A* **shows** $finite\ (R^*\ ``\ A)$
⟨*proof*⟩

More about converse *rtrancl* and *trancl*, should be merged with main body.

**lemma** *single-valued-confluent*:
**assumes** *single-valued r* **and** *xy*: $(x,\ y) \in r^*$ **and** *xz*: $(x,\ z) \in r^*$
**shows** $(y,\ z) \in r^* \vee (z,\ y) \in r^*$
⟨*proof*⟩

**lemma** *r-r-into-trancl*: $(a,\ b) \in R \Longrightarrow (b,\ c) \in R \Longrightarrow (a,\ c) \in R^+$
⟨*proof*⟩

**lemma** *trancl-into-trancl*: $(a,\ b) \in r^+ \Longrightarrow (b,\ c) \in r \Longrightarrow (a,\ c) \in r^+$
⟨*proof*⟩

**lemma** *tranclp-rtranclp-tranclp*:
**assumes** $r^{++}\ a\ b\ r^{**}\ b\ c$ **shows** $r^{++}\ a\ c$

⟨*proof*⟩

**lemma** *rtranclp-conversep*: $r^{-1-1**} = r^{**-1-1}$
⟨*proof*⟩

**lemmas** *symp-rtranclp* = *sym-rtrancl*[*to-pred*]

**lemmas** *symp-conv-conversep-eq* = *sym-conv-converse-eq*[*to-pred*]

**lemmas** *rtranclp-tranclp-absorb* [*simp*] = *rtrancl-trancl-absorb*[*to-pred*]
**lemmas** *tranclp-rtranclp-absorb* [*simp*] = *trancl-rtrancl-absorb*[*to-pred*]
**lemmas** *rtranclp-reflclp-absorb* [*simp*] = *rtrancl-reflcl-absorb*[*to-pred*]

**lemmas** *trancl-rtrancl-trancl* = *tranclp-rtranclp-tranclp* [*to-set*]

**lemmas** *transitive-closure-trans* [*trans*] =
*r-r-into-trancl trancl-trans rtrancl-trans*
*trancl.trancl-into-trancl trancl-into-trancl2*
*rtrancl.rtrancl-into-rtrancl converse-rtrancl-into-rtrancl*
*rtrancl-trancl-trancl trancl-rtrancl-trancl*

**lemmas** *transitive-closurep-trans′* [*trans*] =
*tranclp-trans rtranclp-trans*
*tranclp.trancl-into-trancl tranclp-into-tranclp2*
*rtranclp.rtrancl-into-rtrancl converse-rtranclp-into-rtranclp*
*rtranclp-tranclp-tranclp tranclp-rtranclp-tranclp*

**declare** *trancl-into-rtrancl* [*elim*]

**lemma** *tranclp-ident-if-transp*:
**assumes** *transp R*
**shows** $R^{++} = R$
⟨*proof*⟩

The following are special cases of *tranclp-ident-if-transp*, but they appear duplicated in multiple, independent theories, which causes name clashes.

**lemma** (**in** *preorder*) *tranclp-less*[*simp*]: $(<)^{++} = (<)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *tranclp-less-eq*[*simp*]: $(\leq)^{++} = (\leq)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *tranclp-greater*[*simp*]: $(>)^{++} = (>)$
⟨*proof*⟩

**lemma** (**in** *preorder*) *tranclp-greater-eq*[*simp*]: $(\geq)^{++} = (\geq)$
⟨*proof*⟩

## 21.4 Symmetric closure

**definition** *symclp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
**where** $symclp\ r\ x\ y \longleftrightarrow r\ x\ y \lor r\ y\ x$

**lemma** *symclpI* [*simp*, *intro?*]:
  **shows** *symclpI1*: $r\ x\ y \Longrightarrow symclp\ r\ x\ y$
    **and** *symclpI2*: $r\ y\ x \Longrightarrow symclp\ r\ x\ y$
  ⟨*proof*⟩

**lemma** *symclpE* [*consumes 1*, *cases pred*]:
  **assumes** $symclp\ r\ x\ y$
  **obtains** (*base*) $r\ x\ y$ | (*sym*) $r\ y\ x$
  ⟨*proof*⟩

**lemma** *symclp-pointfree*: $symclp\ r = sup\ r\ r^{-1-1}$
  ⟨*proof*⟩

**lemma** *symclp-greater*: $r \leq symclp\ r$
  ⟨*proof*⟩

**lemma** *symclp-conversep* [*simp*]: $symclp\ r^{-1-1} = symclp\ r$
  ⟨*proof*⟩

**lemma** *symp-on-symclp* [*simp*]: $symp\text{-}on\ A\ (symclp\ R)$
  ⟨*proof*⟩

**lemma** *symp-symclp-eq*: $symp\ r \Longrightarrow symclp\ r = r$
  ⟨*proof*⟩

**lemma** *symp-rtranclp-symclp* [*simp*]: $symp\ (symclp\ r)^{**}$
  ⟨*proof*⟩

**lemma** *rtranclp-symclp-sym* [*sym*]: $(symclp\ r)^{**}\ x\ y \Longrightarrow (symclp\ r)^{**}\ y\ x$
  ⟨*proof*⟩

**lemma** *symclp-idem* [*simp*]: $symclp\ (symclp\ r) = symclp\ r$
  ⟨*proof*⟩

**lemma** *reflp-on-rtranclp* [*simp*]: $reflp\text{-}on\ A\ R^{**}$
  ⟨*proof*⟩

## 21.5 The power operation on relations

$R\ \widehat{}\widehat{}\ n = R\ O\ \ldots\ O\ R$, the n-fold composition of $R$

**overloading**
  $relpow \equiv compow :: nat \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
  $relpowp \equiv compow :: nat \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool)$
**begin**

**primrec** *relpow* :: $nat \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
  **where**
    $relpow\ 0\ R = Id$
  | $relpow\ (Suc\ n)\ R = (R\ \widehat{}\widehat{}\ n)\ O\ R$

**primrec** *relpowp* :: $nat \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool)$
  **where**
    $relpowp\ 0\ R = HOL.eq$
  | $relpowp\ (Suc\ n)\ R = (R\ \widehat{}\widehat{}\ n)\ OO\ R$

**end**

**lemmas** *relpowp-Suc-right* = *relpowp.simps(2)*

**lemma** *relpowp-relpow-eq* [*pred-set-conv*]:
  $(\lambda x\ y.\ (x, y) \in R)\ \widehat{}\widehat{}\ n = (\lambda x\ y.\ (x, y) \in R\ \widehat{}\widehat{}\ n)$ **for** $R :: 'a\ rel$
  ⟨*proof*⟩

For code generation:

**definition** *relpow* :: $nat \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
  **where** *relpow-code-def* [*code-abbrev*]: *relpow* = *compow*

**definition** *relpowp* :: $nat \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool)$
  **where** *relpowp-code-def* [*code-abbrev*]: *relpowp* = *compow*

**lemma** [*code*]:
  $relpow\ 0\ R = Id$
  $relpow\ (Suc\ n)\ R = relpow\ n\ R\ O\ R$
  ⟨*proof*⟩

**lemma** [*code*]:
  $relpowp\ 0\ R = HOL.eq$
  $relpowp\ (Suc\ n)\ R = relpowp\ n\ R\ OO\ R$
  ⟨*proof*⟩

**hide-const** (**open**) *relpow*
**hide-const** (**open**) *relpowp*

**lemma** *relpow-1* [*simp*]: $R\ \widehat{}\widehat{}\ 1 = R$
  **for** $R :: ('a \times 'a)\ set$
  ⟨*proof*⟩

**lemma** *relpowp-1* [*simp*]: $P\ \widehat{}\widehat{}\ 1 = P$
  **for** $P :: 'a \Rightarrow 'a \Rightarrow bool$
  ⟨*proof*⟩

**lemma** *relpowp-Suc-0* [*simp*]: $P\ \widehat{}\widehat{}\ (Suc\ 0) = P$
  **for** $P :: 'a \Rightarrow 'a \Rightarrow bool$

⟨*proof*⟩

**lemma** *relpow-0-I*: $(x, x) \in R$ ^^ $0$
⟨*proof*⟩

**lemma** *relpowp-0-I*: $(P$ ^^ $0)\ x\ x$
⟨*proof*⟩

**lemma** *relpow-Suc-I*: $(x, y) \in R$ ^^ $n \Longrightarrow (y, z) \in R \Longrightarrow (x, z) \in R$ ^^ $Suc\ n$
⟨*proof*⟩

**lemma** *relpowp-Suc-I*[*trans*]: $(P$ ^^ $n)\ x\ y \Longrightarrow P\ y\ z \Longrightarrow (P$ ^^ $Suc\ n)\ x\ z$
⟨*proof*⟩

**lemma** *relpow-Suc-I2*: $(x, y) \in R \Longrightarrow (y, z) \in R$ ^^ $n \Longrightarrow (x, z) \in R$ ^^ $Suc\ n$
⟨*proof*⟩

**lemma** *relpowp-Suc-I2*[*trans*]: $P\ x\ y \Longrightarrow (P$ ^^ $n)\ y\ z \Longrightarrow (P$ ^^ $Suc\ n)\ x\ z$
⟨*proof*⟩

**lemma** *relpow-0-E*: $(x, y) \in R$ ^^ $0 \Longrightarrow (x = y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *relpowp-0-E*: $(P$ ^^ $0)\ x\ y \Longrightarrow (x = y \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *relpow-Suc-E*: $(x, z) \in R$ ^^ $Suc\ n \Longrightarrow (\bigwedge y.\ (x, y) \in R$ ^^ $n \Longrightarrow (y, z) \in R \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *relpowp-Suc-E*: $(P$ ^^ $Suc\ n)\ x\ z \Longrightarrow (\bigwedge y.\ (P$ ^^ $n)\ x\ y \Longrightarrow P\ y\ z \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *relpow-E*:
$(x, z) \in R$ ^^ $n \Longrightarrow$
$(n = 0 \Longrightarrow x = z \Longrightarrow P) \Longrightarrow$
$(\bigwedge y\ m.\ n = Suc\ m \Longrightarrow (x, y) \in R$ ^^ $m \Longrightarrow (y, z) \in R \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *relpowp-E*:
$(P$ ^^ $n)\ x\ z \Longrightarrow$
$(n = 0 \Longrightarrow x = z \Longrightarrow Q) \Longrightarrow$
$(\bigwedge y\ m.\ n = Suc\ m \Longrightarrow (P$ ^^ $m)\ x\ y \Longrightarrow P\ y\ z \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *relpow-Suc-D2*: $(x, z) \in R$ ^^ $Suc\ n \Longrightarrow (\exists\, y.\ (x, y) \in R \wedge (y, z) \in R$ ^^ $n)$
⟨*proof*⟩

**lemma** *relpowp-Suc-D2*: $(P \widehat{\ } \widehat{\ } Suc\ n)\ x\ z \Longrightarrow \exists\, y.\ P\ x\ y \wedge (P \widehat{\ } \widehat{\ }\ n)\ y\ z$
⟨*proof*⟩

**lemma** *relpow-Suc-E2*: $(x, z) \in R \widehat{\ } \widehat{\ } Suc\ n \Longrightarrow (\bigwedge y.\ (x, y) \in R \Longrightarrow (y, z) \in R \widehat{\ } \widehat{\ }\ n \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *relpowp-Suc-E2*: $(P \widehat{\ } \widehat{\ } Suc\ n)\ x\ z \Longrightarrow (\bigwedge y.\ P\ x\ y \Longrightarrow (P \widehat{\ } \widehat{\ }\ n)\ y\ z \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *relpow-Suc-D2'*: $\forall\, x\ y\ z.\ (x, y) \in R \widehat{\ } \widehat{\ }\ n \wedge (y, z) \in R \longrightarrow (\exists\, w.\ (x, w) \in R \wedge (w, z) \in R \widehat{\ } \widehat{\ }\ n)$
⟨*proof*⟩

**lemma** *relpowp-Suc-D2'*: $\forall\, x\ y\ z.\ (P \widehat{\ } \widehat{\ }\ n)\ x\ y \wedge P\ y\ z \longrightarrow (\exists\, w.\ P\ x\ w \wedge (P \widehat{\ } \widehat{\ }\ n)\ w\ z)$
⟨*proof*⟩

**lemma** *relpow-E2*:
**assumes** $(x, z) \in R \widehat{\ } \widehat{\ }\ n\ n = 0 \Longrightarrow x = z \Longrightarrow P$
$\bigwedge y\ m.\ n = Suc\ m \Longrightarrow (x, y) \in R \Longrightarrow (y, z) \in R \widehat{\ } \widehat{\ }\ m \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *relpowp-E2*:
$(P \widehat{\ } \widehat{\ }\ n)\ x\ z \Longrightarrow$
$(n = 0 \Longrightarrow x = z \Longrightarrow Q) \Longrightarrow$
$(\bigwedge y\ m.\ n = Suc\ m \Longrightarrow P\ x\ y \Longrightarrow (P \widehat{\ } \widehat{\ }\ m)\ y\ z \Longrightarrow Q) \Longrightarrow Q$
⟨*proof*⟩

**lemma** *relpowp-trans*[*trans*]: $(R \widehat{\ } \widehat{\ }\ i)\ x\ y \Longrightarrow (R \widehat{\ } \widehat{\ }\ j)\ y\ z \Longrightarrow (R \widehat{\ } \widehat{\ }\ (i + j))\ x\ z$
⟨*proof*⟩

**lemma** *relpowp-mono*:
**fixes** $x\ y :: 'a$
**shows** $(\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ x\ y) \Longrightarrow (R \widehat{\ } \widehat{\ }\ n)\ x\ y \Longrightarrow (S \widehat{\ } \widehat{\ }\ n)\ x\ y$
⟨*proof*⟩

**lemma** *relpow-trans*[*trans*]: $(x, y) \in R \widehat{\ } \widehat{\ }\ i \Longrightarrow (y, z) \in R \widehat{\ } \widehat{\ }\ j \Longrightarrow (x, z) \in R \widehat{\ } \widehat{\ }\ (i + j)$
⟨*proof*⟩

**lemma** *relpowp-left-unique*:
**fixes** $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $n :: nat$ **and** $x\ y\ z :: 'a$
**assumes** *lunique*: $\bigwedge x\ y\ z.\ R\ x\ z \Longrightarrow R\ y\ z \Longrightarrow x = y$
**shows** $(R \widehat{\ } \widehat{\ }\ n)\ x\ z \Longrightarrow (R \widehat{\ } \widehat{\ }\ n)\ y\ z \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *relpow-left-unique*:
**fixes** $R :: ('a \times 'a)\ set$ **and** $n :: nat$ **and** $x\ y\ z :: 'a$
**shows** $(\bigwedge x\ y\ z.\ (x,\ z) \in R \Longrightarrow (y,\ z) \in R \Longrightarrow x = y) \Longrightarrow$
$(x,\ z) \in R\ \widehat{}\widehat{}\ n \Longrightarrow (y,\ z) \in R\ \widehat{}\widehat{}\ n \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *relpowp-right-unique*:
**fixes** $R :: 'a \Rightarrow 'a \Rightarrow bool$ **and** $n :: nat$ **and** $x\ y\ z :: 'a$
**assumes** *runique*: $\bigwedge x\ y\ z.\ R\ x\ y \Longrightarrow R\ x\ z \Longrightarrow y = z$
**shows** $(R\ \widehat{}\widehat{}\ n)\ x\ y \Longrightarrow (R\ \widehat{}\widehat{}\ n)\ x\ z \Longrightarrow y = z$
⟨*proof*⟩

**lemma** *relpow-right-unique*:
**fixes** $R :: ('a \times 'a)\ set$ **and** $n :: nat$ **and** $x\ y\ z :: 'a$
**shows** $(\bigwedge x\ y\ z.\ (x,\ y) \in R \Longrightarrow (x,\ z) \in R \Longrightarrow y = z) \Longrightarrow$
$(x,\ y) \in (R\ \widehat{}\widehat{}\ n) \Longrightarrow (x,\ z) \in (R\ \widehat{}\widehat{}\ n) \Longrightarrow y = z$
⟨*proof*⟩

**lemma** *relpow-add*: $R\ \widehat{}\widehat{}\ (m + n) = R\widehat{}\widehat{}m\ O\ R\widehat{}\widehat{}n$
⟨*proof*⟩

**lemma** *relpowp-add*: $P\ \widehat{}\widehat{}\ (m + n) = P\ \widehat{}\widehat{}\ m\ OO\ P\ \widehat{}\widehat{}\ n$
⟨*proof*⟩

**lemma** *relpow-commute*: $R\ O\ R\ \widehat{}\widehat{}\ n = R\ \widehat{}\widehat{}\ n\ O\ R$
⟨*proof*⟩

**lemma** *relpowp-commute*: $P\ OO\ P\ \widehat{}\widehat{}\ n = P\ \widehat{}\widehat{}\ n\ OO\ P$
⟨*proof*⟩

**lemma** *relpowp-Suc-left*: $R\ \widehat{}\widehat{}\ Suc\ n = R\ OO\ (R\ \widehat{}\widehat{}\ n)$
⟨*proof*⟩

**lemma** *relpow-empty*: $0 < n \Longrightarrow (\{\} :: ('a \times 'a)\ set)\ \widehat{}\widehat{}\ n = \{\}$
⟨*proof*⟩

**lemma** *relpowp-bot*: $0 < n \Longrightarrow (\bot :: 'a \Rightarrow 'a \Rightarrow bool)\ \widehat{}\widehat{}\ n = \bot$
⟨*proof*⟩

**lemma** *rtrancl-imp-UN-relpow*:
**assumes** $p \in R^*$
**shows** $p \in (\bigcup n.\ R\ \widehat{}\widehat{}\ n)$
⟨*proof*⟩

**lemma** *rtranclp-imp-Sup-relpowp*:
**assumes** $(P^{**})\ x\ y$
**shows** $(\bigsqcup n.\ P\ \widehat{}\widehat{}\ n)\ x\ y$
⟨*proof*⟩

**lemma** *relpow-imp-rtrancl*:
  **assumes** $p \in R \frown\frown n$
  **shows** $p \in R^*$
⟨*proof*⟩

**lemma** *relpowp-imp-rtranclp*: $(P \frown\frown n)\ x\ y \Longrightarrow (P^{**})\ x\ y$
  ⟨*proof*⟩

**lemma** *rtrancl-is-UN-relpow*: $R^* = (\bigcup n.\ R \frown\frown n)$
  ⟨*proof*⟩

**lemma** *rtranclp-is-Sup-relpowp*: $P^{**} = (\bigsqcup n.\ P \frown\frown n)$
  ⟨*proof*⟩

**lemma** *rtrancl-power*: $p \in R^* \longleftrightarrow (\exists n.\ p \in R \frown\frown n)$
  ⟨*proof*⟩

**lemma** *rtranclp-power*: $(P^{**})\ x\ y \longleftrightarrow (\exists n.\ (P \frown\frown n)\ x\ y)$
  ⟨*proof*⟩

**lemma** *trancl-power*: $p \in R^+ \longleftrightarrow (\exists n > 0.\ p \in R \frown\frown n)$
⟨*proof*⟩

**lemma** *tranclp-power*: $(P^{++})\ x\ y \longleftrightarrow (\exists n > 0.\ (P \frown\frown n)\ x\ y)$
  ⟨*proof*⟩

**lemma** *rtrancl-imp-relpow*: $p \in R^* \Longrightarrow \exists n.\ p \in R \frown\frown n$
  ⟨*proof*⟩

**lemma** *rtranclp-imp-relpowp*: $(P^{**})\ x\ y \Longrightarrow \exists n.\ (P \frown\frown n)\ x\ y$
  ⟨*proof*⟩

By Sternagel/Thiemann:

**lemma** *relpow-fun-conv*: $(a,\ b) \in R \frown\frown n \longleftrightarrow (\exists f.\ f\ 0 = a \wedge f\ n = b \wedge (\forall i<n.\ (f\ i,\ f\ (Suc\ i)) \in R))$
⟨*proof*⟩

**lemma** *relpowp-fun-conv*: $(P \frown\frown n)\ x\ y \longleftrightarrow (\exists f.\ f\ 0 = x \wedge f\ n = y \wedge (\forall i<n.\ P\ (f\ i)\ (f\ (Suc\ i))))$
  ⟨*proof*⟩

**lemma** *relpow-finite-bounded1*:
  **fixes** $R :: ('a \times 'a)\ set$
  **assumes** *finite* $R$ **and** $k > 0$
  **shows** $R\frown\frown k \subseteq (\bigcup n\in\{n.\ 0 < n \wedge n \leq card\ R\}.\ R\frown\frown n)$
    (**is** - $\subseteq$ *?r*)
⟨*proof*⟩

**lemma** *relpow-finite-bounded*:
**fixes** $R :: ('a \times 'a)\ set$
**assumes** *finite R*
**shows** $R\widehat{\ }\widehat{\ }k \subseteq (\bigcup n\in\{n.\ n \leq card\ R\}.\ R\widehat{\ }\widehat{\ }n)$
⟨*proof*⟩

**lemma** *rtrancl-finite-eq-relpow*: $finite\ R \Longrightarrow R^* = (\bigcup n\in\{n.\ n \leq card\ R\}.\ R\widehat{\ }\widehat{\ }n)$
⟨*proof*⟩

**lemma** *trancl-finite-eq-relpow*:
**assumes** *finite R* **shows** $R^+ = (\bigcup n\in\{n.\ 0 < n \wedge n \leq card\ R\}.\ R\widehat{\ }\widehat{\ }n)$
⟨*proof*⟩

**lemma** *finite-relcomp*[*simp*,*intro*]:
**assumes** *finite R* **and** *finite S*
**shows** *finite* $(R\ O\ S)$
⟨*proof*⟩

**lemma** *finite-relpow* [*simp*, *intro*]:
**fixes** $R :: ('a \times 'a)\ set$
**assumes** *finite R*
**shows** $n > 0 \Longrightarrow finite\ (R\widehat{\ }\widehat{\ }n)$
⟨*proof*⟩

**lemma** *single-valued-relpow*:
**fixes** $R :: ('a \times 'a)\ set$
**shows** $single\text{-}valued\ R \Longrightarrow single\text{-}valued\ (R\ \widehat{\ }\widehat{\ }\ n)$
⟨*proof*⟩

## 21.6 Bounded transitive closure

**definition** $ntrancl :: nat \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
**where** $ntrancl\ n\ R = (\bigcup i\in\{i.\ 0 < i \wedge i \leq Suc\ n\}.\ R\ \widehat{\ }\widehat{\ }\ i)$

**lemma** *ntrancl-Zero* [*simp*, *code*]: $ntrancl\ 0\ R = R$
⟨*proof*⟩

**lemma** *ntrancl-Suc* [*simp*]: $ntrancl\ (Suc\ n)\ R = ntrancl\ n\ R\ O\ (Id \cup R)$
⟨*proof*⟩

**lemma** [*code*]: $ntrancl\ (Suc\ n)\ r = (let\ r' = ntrancl\ n\ r\ in\ r' \cup r'\ O\ r)$
⟨*proof*⟩

**lemma** *finite-trancl-ntranl*: $finite\ R \Longrightarrow trancl\ R = ntrancl\ (card\ R - 1)\ R$
⟨*proof*⟩

## 21.7 Acyclic relations

**definition** $acyclic :: ('a \times 'a)\ set \Rightarrow bool$
**where** $acyclic\ r \longleftrightarrow (\forall x.\ (x,x) \notin r^+)$

**abbreviation** *acyclicP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
  **where** $acyclicP\ r \equiv acyclic\ \{(x,\ y).\ r\ x\ y\}$

**lemma** *acyclic-irrefl* [*code*]: $acyclic\ r \longleftrightarrow irrefl\ (r^+)$
  ⟨*proof*⟩

**lemma** *acyclicI*: $\forall x.\ (x,\ x) \notin r^+ \Longrightarrow acyclic\ r$
  ⟨*proof*⟩

**lemma** (**in** *preorder*) *acyclicI-order*:
  **assumes** $*$: $\bigwedge a\ b.\ (a,\ b) \in r \Longrightarrow f\ b < f\ a$
  **shows** *acyclic r*
⟨*proof*⟩

**lemma** *acyclic-insert* [*iff*]: $acyclic\ (insert\ (y,\ x)\ r) \longleftrightarrow acyclic\ r \wedge (x,\ y) \notin r^*$
  ⟨*proof*⟩

**lemma** *acyclic-converse* [*iff*]: $acyclic\ (r^{-1}) \longleftrightarrow acyclic\ r$
  ⟨*proof*⟩

**lemmas** *acyclicP-converse* [*iff*] = *acyclic-converse* [*to-pred*]

**lemma** *acyclic-impl-antisym-rtrancl*: $acyclic\ r \Longrightarrow antisym\ (r^*)$
  ⟨*proof*⟩

**lemma** *acyclic-subset*: $acyclic\ s \Longrightarrow r \subseteq s \Longrightarrow acyclic\ r$
  ⟨*proof*⟩

## 21.8 Setup of transitivity reasoner

⟨*ML*⟩

**lemma** *transp-rtranclp* [*simp*]: $transp\ R^{**}$
  ⟨*proof*⟩

Optional methods.

⟨*ML*⟩

**end**

# 22 Well-founded Recursion

**theory** *Wellfounded*
  **imports** *Transitive-Closure*
**begin**

## 22.1 Basic Definitions

**definition** *wf-on* :: $'a\ set \Rightarrow 'a\ rel \Rightarrow bool$ **where**
$wf\text{-}on\ A\ r \longleftrightarrow (\forall P.\ (\forall x \in A.\ (\forall y \in A.\ (y, x) \in r \longrightarrow P\ y) \longrightarrow P\ x) \longrightarrow (\forall x \in A.\ P\ x))$

**abbreviation** *wf* :: $('a \times 'a)\ set \Rightarrow bool$ **where**
$wf \equiv wf\text{-}on\ UNIV$

**definition** *wfp-on* :: $'a\ set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$wfp\text{-}on\ A\ R \longleftrightarrow (\forall P.\ (\forall x \in A.\ (\forall y \in A.\ R\ y\ x \longrightarrow P\ y) \longrightarrow P\ x) \longrightarrow (\forall x \in A.\ P\ x))$

**abbreviation** *wfP* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
$wfP \equiv wfp\text{-}on\ UNIV$

**alias** $wfp = wfP$

We keep old name *wfp* for backward compatibility, but offer new name *wfp* to be consistent with similar predicates, e.g., *asymp*, *transp*, *totalp*.

## 22.2 Equivalence of Definitions

**lemma** *wfp-on-wf-on-eq*[*pred-set-conv*]: $wfp\text{-}on\ A\ (\lambda x\ y.\ (x, y) \in r) \longleftrightarrow wf\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *wf-def*: $wf\ r \longleftrightarrow (\forall P.\ (\forall x.\ (\forall y.\ (y, x) \in r \longrightarrow P\ y) \longrightarrow P\ x) \longrightarrow (\forall x.\ P\ x))$
⟨*proof*⟩

**lemma** *wfp-def*: $wfp\ r \longleftrightarrow wf\ \{(x, y).\ r\ x\ y\}$
⟨*proof*⟩

**lemma** *wfp-wf-eq*: $wfp\ (\lambda x\ y.\ (x, y) \in r) = wf\ r$
⟨*proof*⟩

## 22.3 Induction Principles

**lemma** *wf-on-induct*[*consumes 1*, *case-names in-set less*, *induct set*: *wf-on*]:
**assumes** $wf\text{-}on\ A\ r$ **and** $x \in A$ **and** $\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow (y, x) \in r \Longrightarrow P\ y) \Longrightarrow P\ x$
**shows** $P\ x$
⟨*proof*⟩

**lemma** *wfp-on-induct*[*consumes 1*, *case-names in-set less*, *induct pred*: *wfp-on*]:
**assumes** $wfp\text{-}on\ A\ r$ **and** $x \in A$ **and** $\bigwedge x.\ x \in A \Longrightarrow (\bigwedge y.\ y \in A \Longrightarrow r\ y\ x \Longrightarrow P\ y) \Longrightarrow P\ x$
**shows** $P\ x$
⟨*proof*⟩

**lemma** *wf-induct*:
**assumes** $wf\ r$
**and** $\bigwedge x.\ \forall y.\ (y,\ x) \in r \longrightarrow P\ y \Longrightarrow P\ x$
**shows** $P\ a$
⟨*proof*⟩

**lemmas** *wfp-induct* = *wf-induct* [*to-pred*]

**lemmas** *wf-induct-rule* = *wf-induct* [*rule-format*, *consumes 1*, *case-names less*, *induct set*: *wf*]

**lemmas** *wfp-induct-rule* = *wf-induct-rule* [*to-pred*, *induct set*: *wfp*]

**lemma** *wf-on-iff-wf*: $wf\text{-}on\ A\ r \longleftrightarrow wf\ \{(x,\ y) \in r.\ x \in A \wedge y \in A\}$
⟨*proof*⟩

## 22.4 Introduction Rules

**lemma** *wfUNIVI*: $(\bigwedge P\ x.\ (\forall x.\ (\forall y.\ (y,\ x) \in r \longrightarrow P\ y) \longrightarrow P\ x) \Longrightarrow P\ x) \Longrightarrow wf\ r$
⟨*proof*⟩

**lemmas** *wfpUNIVI* = *wfUNIVI* [*to-pred*]

Restriction to domain $A$ and range $B$. If $r$ is well-founded over their intersection, then *wf r*.

**lemma** *wfI*:
**assumes** $r \subseteq A \times B$
**and** $\bigwedge x\ P.\ [\![\forall x.\ (\forall y.\ (y,\ x) \in r \longrightarrow P\ y) \longrightarrow P\ x;\ \ x \in A;\ x \in B]\!] \Longrightarrow P\ x$
**shows** $wf\ r$
⟨*proof*⟩

## 22.5 Ordering Properties

**lemma** *wf-not-sym*: $wf\ r \Longrightarrow (a,\ x) \in r \Longrightarrow (x,\ a) \notin r$
⟨*proof*⟩

**lemma** *wf-asym*:
**assumes** $wf\ r\ (a,\ x) \in r$
**obtains** $(x,\ a) \notin r$
⟨*proof*⟩

**lemma** *wf-imp-asym*: $wf\ r \Longrightarrow asym\ r$
⟨*proof*⟩

**lemma** *wfp-imp-asymp*: $wfp\ r \Longrightarrow asymp\ r$
⟨*proof*⟩

**lemma** *wf-not-refl* [*simp*]: *wf r* $\Longrightarrow$ $(a, a) \notin r$
⟨*proof*⟩

**lemma** *wf-irrefl*:
**assumes** *wf r*
**obtains** $(a, a) \notin r$
⟨*proof*⟩

**lemma** *wf-imp-irrefl*:
**assumes** *wf r* **shows** *irrefl r*
⟨*proof*⟩

**lemma** *wfp-imp-irreflp*: *wfp r* $\Longrightarrow$ *irreflp r*
⟨*proof*⟩

**lemma** *wf-wellorderI*:
**assumes** *wf*: *wf* $\{(x::'a::ord, y).\ x < y\}$
**and** *lin*: *OFCLASS*('*a*::*ord*, *linorder-class*)
**shows** *OFCLASS*('*a*::*ord*, *wellorder-class*)
⟨*proof*⟩

**lemma** (**in** *wellorder*) *wf*: *wf* $\{(x, y).\ x < y\}$
⟨*proof*⟩

**lemma** (**in** *wellorder*) *wfp-on-less*[*simp*]: *wfp-on A* $(<)$
⟨*proof*⟩

## 22.6 Basic Results

Point-free characterization of well-foundedness

**lemma** *wf-onE-pf*:
**assumes** *wf*: *wf-on A r* **and** $B \subseteq A$ **and** $B \subseteq r$ ‘‘ $B$
**shows** $B = \{\}$
⟨*proof*⟩

**lemma** *wfE-pf*: *wf R* $\Longrightarrow A \subseteq R$ ‘‘ $A \Longrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *wf-onI-pf*:
**assumes** $\bigwedge B.\ B \subseteq A \Longrightarrow B \subseteq R$ ‘‘ $B \Longrightarrow B = \{\}$
**shows** *wf-on A R*
⟨*proof*⟩

**lemma** *wfI-pf*: $(\bigwedge A.\ A \subseteq R$ ‘‘ $A \Longrightarrow A = \{\}) \Longrightarrow$ *wf R*
⟨*proof*⟩

### 22.6.1 Minimal-element characterization of well-foundedness

**lemma** *wf-on-iff-ex-minimal*: $wf\text{-}on\ A\ R \longleftrightarrow (\forall B \subseteq A.\ B \neq \{\} \longrightarrow (\exists z \in B.\ \forall y.\ (y, z) \in R \longrightarrow y \notin B))$
⟨*proof*⟩

**lemma** *wf-iff-ex-minimal*: $wf\ R \longleftrightarrow (\forall B.\ B \neq \{\} \longrightarrow (\exists z \in B.\ \forall y.\ (y, z) \in R \longrightarrow y \notin B))$
⟨*proof*⟩

**lemma** *wfp-on-iff-ex-minimal*: $wfp\text{-}on\ A\ R \longleftrightarrow (\forall B \subseteq A.\ B \neq \{\} \longrightarrow (\exists z \in B.\ \forall y.\ R\ y\ z \longrightarrow y \notin B))$
⟨*proof*⟩

**lemma** *wfp-iff-ex-minimal*: $wfp\ R \longleftrightarrow (\forall B.\ B \neq \{\} \longrightarrow (\exists z \in B.\ \forall y.\ R\ y\ z \longrightarrow y \notin B))$
⟨*proof*⟩

**lemma** *wfE-min*:
**assumes** *wf*: $wf\ R$ **and** *Q*: $x \in Q$
**obtains** $z$ **where** $z \in Q\ \bigwedge y.\ (y, z) \in R \Longrightarrow y \notin Q$
⟨*proof*⟩

**lemma** *wfE-min'*:
$wf\ R \Longrightarrow Q \neq \{\} \Longrightarrow (\bigwedge z.\ z \in Q \Longrightarrow (\bigwedge y.\ (y, z) \in R \Longrightarrow y \notin Q) \Longrightarrow thesis) \Longrightarrow thesis$
⟨*proof*⟩

**lemma** *wfI-min*:
**assumes** *a*: $\bigwedge x\ Q.\ x \in Q \Longrightarrow \exists z{\in}Q.\ \forall y.\ (y, z) \in R \longrightarrow y \notin Q$
**shows** $wf\ R$
⟨*proof*⟩

**lemma** *wf-eq-minimal*: $wf\ r \longleftrightarrow (\forall Q\ x.\ x \in Q \longrightarrow (\exists z{\in}Q.\ \forall y.\ (y, z) \in r \longrightarrow y \notin Q))$
⟨*proof*⟩

**lemmas** *wfp-eq-minimal* = *wf-eq-minimal* [*to-pred*]

### 22.6.2 Finite characterization of well-foundedness

**lemma** *strict-partial-order-wfp-on-finite-set*:
**assumes** $transp\text{-}on\ \mathcal{X}\ R$ **and** $asymp\text{-}on\ \mathcal{X}\ R$ **and** $finite\ \mathcal{X}$
**shows** $wfp\text{-}on\ \mathcal{X}\ R$
⟨*proof*⟩

### 22.6.3 Antimonotonicity

**lemma** *wfp-on-mono-stronger*:
**fixes**

$A :: 'a\ set$ **and** $B :: 'b\ set$ **and**
$f :: 'a \Rightarrow 'b$ **and**
$R :: 'b \Rightarrow 'b \Rightarrow bool$ **and** $Q :: 'a \Rightarrow 'a \Rightarrow bool$
**assumes**
*wf*: *wfp-on* $B\ R$ **and**
*sub*: $f$ ‘ $A \subseteq B$ **and**
*mono*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ x\ y \Longrightarrow R\ (f\ x)\ (f\ y)$
**shows** *wfp-on* $A\ Q$
⟨*proof*⟩

**lemma** *wf-on-mono-stronger*:
**assumes**
*wf-on* $B\ r$ **and**
$f$ ‘ $A \subseteq B$ **and**
$(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in q \Longrightarrow (f\ x,\ f\ y) \in r)$
**shows** *wf-on* $A\ q$
⟨*proof*⟩

**lemma** *wf-on-mono-strong*:
**assumes** *wf-on* $B\ r$ **and** $A \subseteq B$ **and** $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in q \Longrightarrow (x,\ y) \in r)$
**shows** *wf-on* $A\ q$
⟨*proof*⟩

**lemma** *wfp-on-mono-strong*:
*wfp-on* $B\ R \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow Q\ x\ y \Longrightarrow R\ x\ y) \Longrightarrow$ *wfp-on* $A\ Q$
⟨*proof*⟩

**lemma** *wf-on-mono*: $A \subseteq B \Longrightarrow q \subseteq r \Longrightarrow$ *wf-on* $B\ r \leq$ *wf-on* $A\ q$
⟨*proof*⟩

**lemma** *wfp-on-mono*: $A \subseteq B \Longrightarrow Q \leq R \Longrightarrow$ *wfp-on* $B\ R \leq$ *wfp-on* $A\ Q$
⟨*proof*⟩

**lemma** *wf-on-subset*: *wf-on* $B\ r \Longrightarrow A \subseteq B \Longrightarrow$ *wf-on* $A\ r$
⟨*proof*⟩

**lemma** *wfp-on-subset*: *wfp-on* $B\ R \Longrightarrow A \subseteq B \Longrightarrow$ *wfp-on* $A\ R$
⟨*proof*⟩

### 22.6.4 Equivalence between *wfp-on* and *wfp*

**lemma** *wfp-on-iff-wfp*: *wfp-on* $A\ R \longleftrightarrow$ *wfp* $(\lambda x\ y.\ R\ x\ y \wedge\ x \in A \wedge y \in A)$
(**is** *?LHS* $\longleftrightarrow$ *?RHS*)
⟨*proof*⟩

### 22.6.5 Well-foundedness of transitive closure

**lemma** *bex-rtrancl-min-element-if-wf-on*:

**assumes** *wf*: *wf-on* $A$ $r$ **and** *x-in*: $x \in A$
**shows** $\exists\, y \in A.\ (y,\ x) \in r^{*} \wedge \neg(\exists\, z \in A.\ (z,\ y) \in r)$
⟨*proof*⟩

**lemma** *bex-rtransclp-min-element-if-wfp-on*: *wfp-on* $A$ $R \Longrightarrow x \in A \Longrightarrow \exists\, y{\in}A.$ $R^{**}$ $y$ $x \wedge \neg$ $(\exists\, z{\in}A.\ R\ z\ y)$
⟨*proof*⟩

**lemma** *ex-terminating-rtranclp-strong*:
**assumes** *wf*: *wfp-on* $\{x'.\ R^{**}\ x\ x'\}$ $R^{-1-1}$
**shows** $\exists\, y.\ R^{**}\ x\ y \wedge (\nexists\, z.\ R\ y\ z)$
⟨*proof*⟩

**lemma** *ex-terminating-rtranclp*:
**assumes** *wf*: *wfp* $R^{-1-1}$
**shows** $\exists\, y.\ R^{**}\ x\ y \wedge (\nexists\, z.\ R\ y\ z)$
⟨*proof*⟩

**lemma** *wf-trancl*:
**assumes** *wf* $r$
**shows** *wf* $(r^{+})$
⟨*proof*⟩

**lemmas** *wfp-tranclp* = *wf-trancl* [*to-pred*]

**lemma** *wf-converse-trancl*: *wf* $(r^{-1}) \Longrightarrow$ *wf* $((r^{+})^{-1})$
⟨*proof*⟩

Well-foundedness of subsets

**lemma** *wf-subset*: *wf* $r \Longrightarrow p \subseteq r \Longrightarrow$ *wf* $p$
⟨*proof*⟩

**lemmas** *wfp-subset* = *wf-subset* [*to-pred*]

Well-foundedness of the empty relation

**lemma** *wf-on-bot*[*iff*]: *wf-on* $A$ $\bot$
⟨*proof*⟩

**lemma** *wfp-on-bot*[*iff*]: *wfp-on* $A$ $\bot$
⟨*proof*⟩

**lemma** *wfp-empty* [*iff*]: *wfp* $(\lambda x\ y.$ *False*$)$
⟨*proof*⟩

**lemma** *wf-Int1*: *wf* $r \Longrightarrow$ *wf* $(r \cap r')$
⟨*proof*⟩

**lemma** *wf-Int2*: *wf* $r \Longrightarrow$ *wf* $(r' \cap r)$
⟨*proof*⟩

Exponentiation.

**lemma** *wf-exp*:
  **assumes** $wf\ (R\ \widehat{\ }\widehat{\ }\ n)$
  **shows** $wf\ R$
$\langle proof \rangle$

Well-foundedness of *insert*.

**lemma** *wf-insert* [*iff*]: $wf\ (insert\ (y,x)\ r) \longleftrightarrow wf\ r \wedge (x,y) \notin r^*$ (**is** $?lhs = ?rhs$)
$\langle proof \rangle$

### 22.6.6 Well-foundedness of image

**lemma** *wf-map-prod-image-Dom-Ran*:
  **fixes** $r{::}\ ('a \times 'a)\ set$
    **and** $f{::}\ 'a \Rightarrow 'b$
  **assumes** *wf-r*: $wf\ r$
    **and** *inj*: $\bigwedge a\ a'.\ a \in Domain\ r \Longrightarrow a' \in Range\ r \Longrightarrow f\ a = f\ a' \Longrightarrow a = a'$
  **shows** $wf\ (map\text{-}prod\ f\ f\ `\ r)$
$\langle proof \rangle$

**lemma** *wf-map-prod-image*: $wf\ r \Longrightarrow inj\ f \Longrightarrow wf\ (map\text{-}prod\ f\ f\ `\ r)$
$\langle proof \rangle$

**lemma** *wfp-on-image*: $wfp\text{-}on\ (f\ `\ A)\ R \longleftrightarrow wfp\text{-}on\ A\ (\lambda a\ b.\ R\ (f\ a)\ (f\ b))$
$\langle proof \rangle$

## 22.7 Well-Foundedness Results for Unions

**lemma** *wf-union-compatible*:
  **assumes** $wf\ R\ wf\ S$
  **assumes** $R\ O\ S \subseteq R$
  **shows** $wf\ (R \cup S)$
$\langle proof \rangle$

Well-foundedness of indexed union with disjoint domains and ranges.

**lemma** *wf-UN*:
  **assumes** *r*: $\bigwedge i.\ i \in I \Longrightarrow wf\ (r\ i)$
    **and** *disj*: $\bigwedge i\ j.\ [\![i \in I;\ j \in I;\ r\ i \neq r\ j]\!] \Longrightarrow Domain\ (r\ i) \cap Range\ (r\ j) = \{\}$
  **shows** $wf\ (\bigcup i{\in}I.\ r\ i)$
  $\langle proof \rangle$

**lemma** *wfp-SUP*:
  $\forall i.\ wfp\ (r\ i) \Longrightarrow \forall i\ j.\ r\ i \neq r\ j \longrightarrow inf\ (Domainp\ (r\ i))\ (Rangep\ (r\ j)) = bot$
$\Longrightarrow$
    $wfp\ (\bigsqcup (range\ r))$
  $\langle proof \rangle$

**lemma** *wf-Union*:
  **assumes** $\forall r{\in}R.\ wf\ r$

**and** $\forall r \in R.\ \forall s \in R.\ r \neq s \longrightarrow \textit{Domain } r \cap \textit{Range } s = \{\}$
**shows** $\textit{wf } (\bigcup R)$
$\langle \textit{proof} \rangle$

Intuition: We find an $R \cup S$-min element of a nonempty subset $A$ by case distinction.

1. There is a step $a\ {-R\rightarrow}\ b$ with $a,\ b \in A$. Pick an $R$-min element $z$ of the (nonempty) set $\{a \in A \mid \exists\, b \in A.\ a\ {-R\rightarrow}\ b\}$. By definition, there is $z' \in A$ s.t. $z\ {-R\rightarrow}\ z'$. Because $z$ is $R$-min in the subset, $z'$ must be $R$-min in $A$. Because $z'$ has an $R$-predecessor, it cannot have an $S$-successor and is thus $S$-min in $A$ as well.

2. There is no such step. Pick an $S$-min element of $A$. In this case it must be an $R$-min element of $A$ as well.

**lemma** *wf-Un*: $\textit{wf } r \Longrightarrow \textit{wf } s \Longrightarrow \textit{Domain } r \cap \textit{Range } s = \{\} \Longrightarrow \textit{wf } (r \cup s)$
$\langle \textit{proof} \rangle$

**lemma** *wf-union-merge*: $\textit{wf } (R \cup S) = \textit{wf } (R\ O\ R \cup S\ O\ R \cup S)$
(**is** $\textit{wf } ?A = \textit{wf } ?B$)
$\langle \textit{proof} \rangle$

**lemma** *wf-comp-self*: $\textit{wf } R \longleftrightarrow \textit{wf } (R\ O\ R)$ — special case
$\langle \textit{proof} \rangle$

## 22.8 Well-Foundedness of Composition

Bachmair and Dershowitz 1986, Lemma 2. [Provided by Tjark Weber]

**lemma** *qc-wf-relto-iff*:
**assumes** $R\ O\ S \subseteq (R \cup S)^*\ O\ R$ — R quasi-commutes over S
**shows** $\textit{wf } (S^*\ O\ R\ O\ S^*) \longleftrightarrow \textit{wf } R$
(**is** $\textit{wf } ?S \longleftrightarrow$ -)
$\langle \textit{proof} \rangle$

**corollary** *wf-relcomp-compatible*:
**assumes** $\textit{wf } R$ **and** $R\ O\ S \subseteq S\ O\ R$
**shows** $\textit{wf } (S\ O\ R)$
$\langle \textit{proof} \rangle$

## 22.9 Acyclic relations

**lemma** *wf-acyclic*: $\textit{wf } r \Longrightarrow \textit{acyclic } r$
$\langle \textit{proof} \rangle$

**lemmas** $\textit{wfp-acyclicP} = \textit{wf-acyclic}\ [\textit{to-pred}]$

### 22.9.1 Wellfoundedness of finite acyclic relations

**lemma** *finite-acyclic-wf*:
**assumes** *finite r acyclic r* **shows** *wf r*
⟨*proof*⟩

**lemma** *finite-acyclic-wf-converse*: *finite r* $\Longrightarrow$ *acyclic r* $\Longrightarrow$ *wf* $(r^{-1})$
⟨*proof*⟩

Observe that the converse of an irreflexive, transitive, and finite relation is again well-founded. Thus, we may employ it for well-founded induction.

**lemma** *wf-converse*:
**assumes** *irrefl r* **and** *trans r* **and** *finite r*
**shows** *wf* $(r^{-1})$
⟨*proof*⟩

**lemma** *wf-iff-acyclic-if-finite*: *finite r* $\Longrightarrow$ *wf r* = *acyclic r*
⟨*proof*⟩

## 22.10 *nat* is well-founded

**lemma** *less-nat-rel*: $(<) = (\lambda m\ n.\ n = Suc\ m)^{++}$
⟨*proof*⟩

**definition** *pred-nat* :: $(nat \times nat)$ *set*
**where** *pred-nat* = $\{(m, n).\ n = Suc\ m\}$

**definition** *less-than* :: $(nat \times nat)$ *set*
**where** *less-than* = *pred-nat*$^+$

**lemma** *less-eq*: $(m, n) \in$ *pred-nat*$^+ \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** *pred-nat-trancl-eq-le*: $(m, n) \in$ *pred-nat*$^* \longleftrightarrow m \le n$
⟨*proof*⟩

**lemma** *wf-pred-nat*: *wf pred-nat*
⟨*proof*⟩

**lemma** *wf-less-than* [*iff*]: *wf less-than*
⟨*proof*⟩

**lemma** *trans-less-than* [*iff*]: *trans less-than*
⟨*proof*⟩

**lemma** *less-than-iff* [*iff*]: $((x,y) \in$ *less-than*$) = (x<y)$
⟨*proof*⟩

**lemma** *irrefl-less-than*: *irrefl less-than*
⟨*proof*⟩

**lemma** *asym-less-than*: *asym less-than*
⟨*proof*⟩

**lemma** *total-less-than*: *total less-than* **and** *total-on-less-than* [*simp*]: *total-on A less-than*
⟨*proof*⟩

**lemma** *wf-less*: *wf* $\{(x, y::nat).\ x < y\}$
⟨*proof*⟩

## 22.11 Accessible Part

Inductive definition of the accessible part *acc r* of a relation; see also [6].

**inductive-set** *acc* :: $('a \times 'a)$ *set* $\Rightarrow$ $'a$ *set* **for** *r* :: $('a \times 'a)$ *set*
**where** *accI*: $(\bigwedge y.\ (y, x) \in r \Longrightarrow y \in acc\ r) \Longrightarrow x \in acc\ r$

**abbreviation** *termip* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$
**where** *termip* $r \equiv accp\ (r^{-1-1})$

**abbreviation** *termi* :: $('a \times 'a)$ *set* $\Rightarrow$ $'a$ *set*
**where** *termi* $r \equiv acc\ (r^{-1})$

**lemmas** *accpI* = *accp.accI*

**lemma** *accp-eq-acc* [*code*]: $accp\ r = (\lambda x.\ x \in Wellfounded.acc\ \{(x, y).\ r\ x\ y\})$
⟨*proof*⟩

Induction rules

**theorem** *accp-induct*:
**assumes** *major*: *accp r a*
**assumes** *hyp*: $\bigwedge x.\ accp\ r\ x \Longrightarrow \forall y.\ r\ y\ x \longrightarrow P\ y \Longrightarrow P\ x$
**shows** *P a*
⟨*proof*⟩

**lemmas** *accp-induct-rule* = *accp-induct* [*rule-format*, *induct set*: *accp*]

**theorem** *accp-downward*: $accp\ r\ b \Longrightarrow r\ a\ b \Longrightarrow accp\ r\ a$
⟨*proof*⟩

**lemma** *not-accp-down*:
**assumes** *na*: $\neg$ *accp R x*
**obtains** *z* **where** *R z x* **and** $\neg$ *accp R z*
⟨*proof*⟩

**lemma** *accp-downwards-aux*: $r^{**}\ b\ a \Longrightarrow accp\ r\ a \longrightarrow accp\ r\ b$
⟨*proof*⟩

**theorem** *accp-downwards*: $accp\ r\ a \Longrightarrow r^{**}\ b\ a \Longrightarrow accp\ r\ b$

⟨*proof*⟩

**theorem** *accp-wfpI*: $\forall\, x.\ accp\ r\ x \Longrightarrow wfp\ r$
⟨*proof*⟩

**theorem** *accp-wfpD*: $wfp\ r \Longrightarrow accp\ r\ x$
⟨*proof*⟩

**theorem** *wfp-iff-accp*: $wfp\ r = (\forall\, x.\ accp\ r\ x)$
⟨*proof*⟩

Smaller relations have bigger accessible parts:

**lemma** *accp-subset*:
**assumes** $R1 \le R2$
**shows** $accp\ R2 \le accp\ R1$
⟨*proof*⟩

This is a generalized induction theorem that works on subsets of the accessible part.

**lemma** *accp-subset-induct*:
**assumes** *subset*: $D \le accp\ R$
**and** *dcl*: $\bigwedge x\ z.\ D\ x \Longrightarrow R\ z\ x \Longrightarrow D\ z$
**and** $D\ x$
**and** *istep*: $\bigwedge x.\ D\ x \Longrightarrow (\bigwedge z.\ R\ z\ x \Longrightarrow P\ z) \Longrightarrow P\ x$
**shows** $P\ x$
⟨*proof*⟩

Set versions of the above theorems

**lemmas** *acc-induct* = *accp-induct* [*to-set*]
**lemmas** *acc-induct-rule* = *acc-induct* [*rule-format*, *induct set*: *acc*]
**lemmas** *acc-downward* = *accp-downward* [*to-set*]
**lemmas** *not-acc-down* = *not-accp-down* [*to-set*]
**lemmas** *acc-downwards-aux* = *accp-downwards-aux* [*to-set*]
**lemmas** *acc-downwards* = *accp-downwards* [*to-set*]
**lemmas** *acc-wfI* = *accp-wfpI* [*to-set*]
**lemmas** *acc-wfD* = *accp-wfpD* [*to-set*]
**lemmas** *wf-iff-acc* = *wfp-iff-accp* [*to-set*]
**lemmas** *acc-subset* = *accp-subset* [*to-set*]
**lemmas** *acc-subset-induct* = *accp-subset-induct* [*to-set*]

## 22.12 Tools for building wellfounded relations

Inverse Image

**lemma** *wf-inv-image* [*simp*,*intro*!]:
**fixes** $f :: 'a \Rightarrow 'b$
**assumes** *wf r*
**shows** *wf* (*inv-image r f*)
⟨*proof*⟩

**lemma** *wfp-on-inv-imagep*:
  **assumes** *wf*: *wfp-on* $(f \text{ ` } A)$ $R$
  **shows** *wfp-on* $A$ (*inv-imagep* $R$ $f$)
  ⟨*proof*⟩

### 22.12.1 Conversion to a known well-founded relation

**lemma** *wfp-on-if-convertible-to-wfp-on*:
  **assumes**
    *wf*: *wfp-on* $(f \text{ ` } A)$ $Q$ **and**
    *convertible*: $(\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow R\ x\ y \Longrightarrow Q\ (f\ x)\ (f\ y))$
  **shows** *wfp-on* $A$ $R$
  ⟨*proof*⟩

**lemma** *wf-on-if-convertible-to-wf-on*: *wf-on* $(f \text{ ` } A)$ $Q \Longrightarrow (\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow (x,\ y) \in R \Longrightarrow (f\ x,\ f\ y) \in Q) \Longrightarrow$ *wf-on* $A$ $R$
  ⟨*proof*⟩

**lemma** *wf-if-convertible-to-wf*:
  **fixes** $r$ :: *'a rel* **and** $s$ :: *'b rel* **and** $f$ :: $'a \Rightarrow 'b$
  **assumes** *wf* $s$ **and** *convertible*: $\bigwedge x\ y.\ (x,\ y) \in r \Longrightarrow (f\ x,\ f\ y) \in s$
  **shows** *wf* $r$
⟨*proof*⟩

**lemma** *wfp-if-convertible-to-wfp*: *wfp* $S \Longrightarrow (\bigwedge x\ y.\ R\ x\ y \Longrightarrow S\ (f\ x)\ (f\ y)) \Longrightarrow$ *wfp* $R$
  ⟨*proof*⟩

Converting to *nat* is a very common special case that might be found more easily by Sledgehammer.

**lemma** *wfp-if-convertible-to-nat*:
  **fixes** $f$ :: - $\Rightarrow$ *nat*
  **shows** $(\bigwedge x\ y.\ R\ x\ y \Longrightarrow f\ x < f\ y) \Longrightarrow$ *wfp* $R$
  ⟨*proof*⟩

### 22.12.2 Measure functions into *nat*

**definition** *measure* :: $('a \Rightarrow nat) \Rightarrow ('a \times 'a)$ *set*
  **where** *measure* = *inv-image less-than*

**lemma** *in-measure*[*simp*, *code-unfold*]: $(x,\ y) \in$ *measure* $f \longleftrightarrow f\ x < f\ y$
  ⟨*proof*⟩

**lemma** *wf-measure* [*iff*]: *wf* (*measure* $f$)
  ⟨*proof*⟩

**lemma** *wf-if-measure*: $(\bigwedge x.\ P\ x \Longrightarrow f(g\ x) < f\ x) \Longrightarrow$ *wf* $\{(y,x).\ P\ x \wedge y = g\ x\}$
  **for** $f$ :: $'a \Rightarrow nat$
  ⟨*proof*⟩

### 22.12.3 Lexicographic combinations

**definition** *lex-prod* :: $('a \times 'a)$ *set* $\Rightarrow$ $('b \times 'b)$ *set* $\Rightarrow$ $(('a \times 'b) \times ('a \times 'b))$ *set*
(**infixr** ‹<∗lex∗>› *80*)
**where** $ra <\!*lex*\!> rb = \{((a, b), (a', b')).\ (a, a') \in ra \lor a = a' \land (b, b') \in rb\}$

**lemma** *in-lex-prod*[*simp*]: $((a, b), (a', b')) \in r <\!*lex*\!> s \longleftrightarrow (a, a') \in r \lor a = a' \land (b, b') \in s$
⟨*proof*⟩

**lemma** *wf-on-lex-prod*[*intro*]:
**assumes** *wfA*: *wf-on* $A\ r_A$ **and** *wfB*: *wf-on* $B\ r_B$
**shows** *wf-on* $(A \times B)\ (r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *wf-lex-prod* [*intro*!]:
**assumes** *wf ra wf rb*
**shows** *wf* $(ra <\!*lex*\!> rb)$
⟨*proof*⟩

**lemma** *refl-lex-prod*[*simp*]: *refl* $r_B \Longrightarrow$ *refl* $(r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *irrefl-on-lex-prod*[*simp*]:
*irrefl-on* $A\ r_A \Longrightarrow$ *irrefl-on* $B\ r_B \Longrightarrow$ *irrefl-on* $(A \times B)\ (r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *irrefl-lex-prod*[*simp*]: *irrefl* $r_A \Longrightarrow$ *irrefl* $r_B \Longrightarrow$ *irrefl* $(r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *sym-on-lex-prod*[*simp*]:
*sym-on* $A\ r_A \Longrightarrow$ *sym-on* $B\ r_B \Longrightarrow$ *sym-on* $(A \times B)\ (r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *sym-lex-prod*[*simp*]:
*sym* $r_A \Longrightarrow$ *sym* $r_B \Longrightarrow$ *sym* $(r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *asym-on-lex-prod*[*simp*]:
*asym-on* $A\ r_A \Longrightarrow$ *asym-on* $B\ r_B \Longrightarrow$ *asym-on* $(A \times B)\ (r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *asym-lex-prod*[*simp*]:
*asym* $r_A \Longrightarrow$ *asym* $r_B \Longrightarrow$ *asym* $(r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *trans-on-lex-prod*[*simp*]:
**assumes** *trans-on* $A\ r_A$ **and** *trans-on* $B\ r_B$
**shows** *trans-on* $(A \times B)\ (r_A <\!*lex*\!> r_B)$
⟨*proof*⟩

**lemma** *trans-lex-prod* [*simp*,*intro*!]: *trans* $r_A \Longrightarrow$ *trans* $r_B \Longrightarrow$ *trans* ($r_A$ <∗*lex*∗> $r_B$)
⟨*proof*⟩

**lemma** *total-on-lex-prod*[*simp*]:
*total-on* $A$ $r_A \Longrightarrow$ *total-on* $B$ $r_B \Longrightarrow$ *total-on* ($A \times B$) ($r_A$ <∗*lex*∗> $r_B$)
⟨*proof*⟩

**lemma** *total-lex-prod*[*simp*]: *total* $r_A \Longrightarrow$ *total* $r_B \Longrightarrow$ *total* ($r_A$ <∗*lex*∗> $r_B$)
⟨*proof*⟩

lexicographic combinations with measure functions

**definition** *mlex-prod* :: ($'a \Rightarrow$ *nat*) $\Rightarrow$ ($'a \times 'a$) *set* $\Rightarrow$ ($'a \times 'a$) *set* (**infixr** ‹<∗*mlex*∗>› *80*)
**where** *f* <∗*mlex*∗> *R* = *inv-image* (*less-than* <∗*lex*∗> *R*) ($\lambda x.$ (*f x*, *x*))

**lemma**
*wf-mlex*: *wf R* $\Longrightarrow$ *wf* (*f* <∗*mlex*∗> *R*) **and**
*mlex-less*: $f\,x < f\,y \Longrightarrow (x, y) \in f$ <∗*mlex*∗> *R* **and**
*mlex-leq*: $f\,x \leq f\,y \Longrightarrow (x, y) \in R \Longrightarrow (x, y) \in f$ <∗*mlex*∗> *R* **and**
*mlex-iff*: $(x, y) \in f$ <∗*mlex*∗> $R \longleftrightarrow f\,x < f\,y \lor f\,x = f\,y \land (x, y) \in R$
⟨*proof*⟩

Proper subset relation on finite sets.

**definition** *finite-psubset* :: ($'a$ *set* $\times$ $'a$ *set*) *set*
**where** *finite-psubset* = $\{(A, B).\ A \subset B \land$ *finite* $B\}$

**lemma** *wf-finite-psubset*[*simp*]: *wf finite-psubset*
⟨*proof*⟩

**lemma** *trans-finite-psubset*: *trans finite-psubset*
⟨*proof*⟩

**lemma** *in-finite-psubset*[*simp*]: $(A, B) \in$ *finite-psubset* $\longleftrightarrow A \subset B \land$ *finite* $B$
⟨*proof*⟩

max- and min-extension of order to finite sets

**inductive-set** *max-ext* :: ($'a \times 'a$) *set* $\Rightarrow$ ($'a$ *set* $\times$ $'a$ *set*) *set*
**for** *R* :: ($'a \times 'a$) *set*
**where** *max-extI*[*intro*]:
*finite* $X \Longrightarrow$ *finite* $Y \Longrightarrow Y \neq \{\} \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow \exists y \in Y.\ (x, y) \in R) \Longrightarrow (X, Y) \in$ *max-ext R*

**lemma** *max-ext-wf*:
**assumes** *wf*: *wf r*
**shows** *wf* (*max-ext r*)
⟨*proof*⟩

**lemma** *max-ext-additive*: $(A, B) \in max\text{-}ext\ R \Longrightarrow (C, D) \in max\text{-}ext\ R \Longrightarrow (A \cup C, B \cup D) \in max\text{-}ext\ R$
⟨*proof*⟩

**definition** *min-ext* :: $('a \times 'a)\ set \Rightarrow ('a\ set \times 'a\ set)\ set$
**where** $min\text{-}ext\ r = \{(X, Y) \mid X\ Y.\ X \neq \{\} \wedge (\forall y \in Y.\ (\exists x \in X.\ (x, y) \in r))\}$

**lemma** *min-ext-wf*:
**assumes** *wf r*
**shows** *wf* (*min-ext r*)
⟨*proof*⟩

### 22.12.4 Bounded increase must terminate

**lemma** *wf-bounded-measure*:
**fixes** $ub :: 'a \Rightarrow nat$
**and** $f :: 'a \Rightarrow nat$
**assumes** $\bigwedge a\ b.\ (b, a) \in r \Longrightarrow ub\ b \leq ub\ a \wedge ub\ a \geq f\ b \wedge f\ b > f\ a$
**shows** *wf r*
⟨*proof*⟩

**lemma** *wf-bounded-set*:
**fixes** $ub :: 'a \Rightarrow 'b\ set$
**and** $f :: 'a \Rightarrow 'b\ set$
**assumes** $\bigwedge a\ b.\ (b,a) \in r \Longrightarrow finite\ (ub\ a) \wedge ub\ b \subseteq ub\ a \wedge ub\ a \supseteq f\ b \wedge f\ b \supset f\ a$
**shows** *wf r*
⟨*proof*⟩

**lemma** *finite-subset-wf*:
**assumes** *finite A*
**shows** $wf\ \{(X, Y).\ X \subset Y \wedge Y \subseteq A\}$
⟨*proof*⟩

**hide-const** (**open**) *acc accp*

## 22.13 Code Generation Setup

Code equations with *wf* or *wfp* on the left-hand side are not supported by the code generation module because of the *UNIV* hidden behind the abbreviations. To sidestep this problem, we provide the following wrapper definitions and use *code-abbrev* to register the definitions with the pre- and post-processors of the code generator.

**definition** *wf-code* :: $('a \times 'a)\ set \Rightarrow bool$ **where**
[*code-abbrev*]: $wf\text{-}code\ r \longleftrightarrow wf\ r$

**definition** *wfp-code* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$ **where**
[*code-abbrev*]: $wfp\text{-}code\ R \longleftrightarrow wfp\ R$

**end**

# 23 Well-Founded Recursion Combinator

**theory** *Wfrec*
**imports** *Wellfounded*
**begin**

**inductive** $wfrec\text{-}rel$ :: $('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
**for** $R\ F$
**where** $wfrecI$: $(\bigwedge z.\ (z,\ x) \in R \Longrightarrow wfrec\text{-}rel\ R\ F\ z\ (g\ z)) \Longrightarrow wfrec\text{-}rel\ R\ F\ x\ (F\ g\ x)$

**definition** $cut$ :: $('a \Rightarrow 'b) \Rightarrow ('a \times 'a)\ set \Rightarrow 'a \Rightarrow 'a \Rightarrow 'b$
**where** $cut\ f\ R\ x = (\lambda y.\ if\ (y,\ x) \in R\ then\ f\ y\ else\ undefined)$

**definition** $adm\text{-}wf$ :: $('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)) \Rightarrow bool$
**where** $adm\text{-}wf\ R\ F \longleftrightarrow (\forall f\ g\ x.\ (\forall z.\ (z,\ x) \in R \longrightarrow f\ z = g\ z) \longrightarrow F\ f\ x = F\ g\ x)$

**definition** $wfrec$ :: $('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b)) \Rightarrow ('a \Rightarrow 'b)$
**where** $wfrec\ R\ F = (\lambda x.\ THE\ y.\ wfrec\text{-}rel\ R\ (\lambda f\ x.\ F\ (cut\ f\ R\ x)\ x)\ x\ y)$

**lemma** $cuts\text{-}eq$: $(cut\ f\ R\ x = cut\ g\ R\ x) \longleftrightarrow (\forall y.\ (y,\ x) \in R \longrightarrow f\ y = g\ y)$
$\langle proof \rangle$

**lemma** $cut\text{-}apply$: $(x,\ a) \in R \Longrightarrow cut\ f\ R\ a\ x = f\ x$
$\langle proof \rangle$

Inductive characterization of *wfrec* combinator; for details see: John Harrison, "Inductive definitions: automation and application".

**lemma** $theI\text{-}unique$: $\exists !x.\ P\ x \Longrightarrow P\ x \longleftrightarrow x = The\ P$
$\langle proof \rangle$

**lemma** $wfrec\text{-}unique$:
**assumes** $adm\text{-}wf\ R\ F\ wf\ R$
**shows** $\exists !y.\ wfrec\text{-}rel\ R\ F\ x\ y$
$\langle proof \rangle$

**lemma** $adm\text{-}lemma$: $adm\text{-}wf\ R\ (\lambda f\ x.\ F\ (cut\ f\ R\ x)\ x)$
$\langle proof \rangle$

**lemma** $wfrec$: $wf\ R \Longrightarrow wfrec\ R\ F\ a = F\ (cut\ (wfrec\ R\ F)\ R\ a)\ a$
$\langle proof \rangle$

This form avoids giant explosions in proofs. NOTE USE OF $\equiv$.

**lemma** $def\text{-}wfrec$: $f \equiv wfrec\ R\ F \Longrightarrow wf\ R \Longrightarrow f\ a = F\ (cut\ f\ R\ a)\ a$
$\langle proof \rangle$

### 23.0.1 Well-founded recursion via genuine fixpoints

**lemma** *wfrec-fixpoint*:
 **assumes** *wf*: *wf R*
  **and** *adm*: *adm-wf R F*
 **shows** $wfrec\ R\ F = F\ (wfrec\ R\ F)$
⟨*proof*⟩

**lemma** *wfrec-def-adm*: $f \equiv wfrec\ R\ F \Longrightarrow wf\ R \Longrightarrow adm\text{-}wf\ R\ F \Longrightarrow f = F\ f$
 ⟨*proof*⟩

## 23.1 Wellfoundedness of *same-fst*

**definition** *same-fst* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow ('b \times 'b)\ set) \Rightarrow (('a \times 'b) \times ('a \times 'b))\ set$
 **where** $same\text{-}fst\ P\ R = \{((x', y'), (x, y)) \,.\, x' = x \wedge P\ x \wedge (y',y) \in R\ x\}$
 — For *wfrec* declarations where the first n parameters stay unchanged in the recursive call.

**lemma** *same-fstI* [*intro*!]: $P\ x \Longrightarrow (y', y) \in R\ x \Longrightarrow ((x, y'), (x, y)) \in same\text{-}fst\ P\ R$
 ⟨*proof*⟩

**lemma** *wf-same-fst*:
 **assumes** $\bigwedge x.\ P\ x \Longrightarrow wf\ (R\ x)$
 **shows** $wf\ (same\text{-}fst\ P\ R)$
⟨*proof*⟩

**end**

# 24 Orders as Relations

**theory** *Order-Relation*
**imports** *Wfrec*
**begin**

## 24.1 Orders on a set

**definition** $preorder\text{-}on\ A\ r \equiv r \subseteq A \times A \wedge refl\text{-}on\ A\ r \wedge trans\ r$

**definition** $partial\text{-}order\text{-}on\ A\ r \equiv preorder\text{-}on\ A\ r \wedge antisym\ r$

**definition** $linear\text{-}order\text{-}on\ A\ r \equiv partial\text{-}order\text{-}on\ A\ r \wedge total\text{-}on\ A\ r$

**definition** $strict\text{-}linear\text{-}order\text{-}on\ A\ r \equiv trans\ r \wedge irrefl\ r \wedge total\text{-}on\ A\ r$

**definition** $well\text{-}order\text{-}on\ A\ r \equiv linear\text{-}order\text{-}on\ A\ r \wedge wf(r - Id)$

**lemmas** *order-on-defs* =
 *preorder-on-def partial-order-on-def linear-order-on-def*

*strict-linear-order-on-def well-order-on-def*

**lemma** *partial-order-onD*:
**assumes** *partial-order-on A r* **shows** *refl-on A r* **and** *trans r* **and** *antisym r* **and** $r \subseteq A \times A$
⟨*proof*⟩

**lemma** *preorder-on-empty*[*simp*]: *preorder-on* {} {}
⟨*proof*⟩

**lemma** *partial-order-on-empty*[*simp*]: *partial-order-on* {} {}
⟨*proof*⟩

**lemma** *lnear-order-on-empty*[*simp*]: *linear-order-on* {} {}
⟨*proof*⟩

**lemma** *well-order-on-empty*[*simp*]: *well-order-on* {} {}
⟨*proof*⟩

**lemma** *preorder-on-converse*[*simp*]: *preorder-on A* $(r^{-1})$ = *preorder-on A r*
⟨*proof*⟩

**lemma** *partial-order-on-converse*[*simp*]: *partial-order-on A* $(r^{-1})$ = *partial-order-on A r*
⟨*proof*⟩

**lemma** *linear-order-on-converse*[*simp*]: *linear-order-on A* $(r^{-1})$ = *linear-order-on A r*
⟨*proof*⟩

**lemma** *partial-order-on-acyclic*:
*partial-order-on A r* $\Longrightarrow$ *acyclic* (*r* − *Id*)
⟨*proof*⟩

**lemma** *partial-order-on-well-order-on*:
*finite r* $\Longrightarrow$ *partial-order-on A r* $\Longrightarrow$ *wf* (*r* − *Id*)
⟨*proof*⟩

**lemma** *strict-linear-order-on-diff-Id*: *linear-order-on A r* $\Longrightarrow$ *strict-linear-order-on A* (*r* − *Id*)
⟨*proof*⟩

**lemma** *linear-order-on-singleton* [*simp*]: *linear-order-on* {*x*} {(*x*, *x*)}
⟨*proof*⟩

**lemma** *linear-order-on-acyclic*:
**assumes** *linear-order-on A r*

**shows** *acyclic* $(r - Id)$
⟨*proof*⟩

**lemma** *linear-order-on-well-order-on*:
**assumes** *finite r*
**shows** *linear-order-on A r* $\longleftrightarrow$ *well-order-on A r*
⟨*proof*⟩

## 24.2 Orders on the field

**abbreviation** *Refl r* $\equiv$ *refl-on* (*Field r*) *r*

**abbreviation** *Preorder r* $\equiv$ *preorder-on* (*Field r*) *r*

**abbreviation** *Partial-order r* $\equiv$ *partial-order-on* (*Field r*) *r*

**abbreviation** *Total r* $\equiv$ *total-on* (*Field r*) *r*

**abbreviation** *Linear-order r* $\equiv$ *linear-order-on* (*Field r*) *r*

**abbreviation** *Well-order r* $\equiv$ *well-order-on* (*Field r*) *r*

**lemma** *subset-Image-Image-iff*:
*Preorder r* $\Longrightarrow A \subseteq$ *Field r* $\Longrightarrow B \subseteq$ *Field r* $\Longrightarrow$
$r$ `` $A \subseteq r$ `` $B \longleftrightarrow (\forall a \in A. \exists b \in B. (b, a) \in r)$
⟨*proof*⟩

**lemma** *subset-Image1-Image1-iff*:
*Preorder r* $\Longrightarrow a \in$ *Field r* $\Longrightarrow b \in$ *Field r* $\Longrightarrow r$ `` $\{a\} \subseteq r$ `` $\{b\} \longleftrightarrow (b, a) \in r$
⟨*proof*⟩

**lemma** *Refl-antisym-eq-Image1-Image1-iff*:
**assumes** *Refl r*
**and** *as*: *antisym r*
**and** *abf*: $a \in$ *Field r* $b \in$ *Field r*
**shows** $r$ `` $\{a\} = r$ `` $\{b\} \longleftrightarrow a = b$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *Partial-order-eq-Image1-Image1-iff*:
*Partial-order r* $\Longrightarrow a \in$ *Field r* $\Longrightarrow b \in$ *Field r* $\Longrightarrow r$ `` $\{a\} = r$ `` $\{b\} \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *Total-Id-Field*:
**assumes** *Total r*
**and** *not-Id*: $\neg\ r \subseteq Id$

**shows** *Field* $r$ = *Field* $(r - Id)$
⟨*proof*⟩

## 24.3 Relations given by a predicate and the field

**definition** *relation-of* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow ('a \times 'a)\ set$
**where** *relation-of* $P\ A \equiv \{\ (a,\ b) \in A \times A.\ P\ a\ b\ \}$

**lemma** *refl-relation-ofD*: *refl* (*relation-of* $R\ S$) $\Longrightarrow$ *reflp-on* $S\ R$
⟨*proof*⟩

**lemma** *irrefl-relation-ofD*: *irrefl* (*relation-of* $R\ S$) $\Longrightarrow$ *irreflp-on* $S\ R$
⟨*proof*⟩

**lemma** *sym-relation-of*[*simp*]: *sym* (*relation-of* $R\ S$) $\longleftrightarrow$ *symp-on* $S\ R$
⟨*proof*⟩

**lemma** *asym-relation-of*[*simp*]: *asym* (*relation-of* $R\ S$) $\longleftrightarrow$ *asymp-on* $S\ R$
⟨*proof*⟩

**lemma** *antisym-relation-of*[*simp*]: *antisym* (*relation-of* $R\ S$) $\longleftrightarrow$ *antisymp-on* $S$ $R$
⟨*proof*⟩

**lemma** *trans-relation-of*[*simp*]: *trans* (*relation-of* $R\ S$) $\longleftrightarrow$ *transp-on* $S\ R$
⟨*proof*⟩

**lemma** *total-relation-ofD*: *total* (*relation-of* $R\ S$) $\Longrightarrow$ *totalp-on* $S\ R$
⟨*proof*⟩

**lemma** *Field-relation-of*:
**assumes** *relation-of* $P\ A \subseteq A \times A$ **and** *refl-on* $A$ (*relation-of* $P\ A$)
**shows** *Field* (*relation-of* $P\ A$) = $A$
⟨*proof*⟩

**lemma** *partial-order-on-relation-ofI*:
**assumes** *refl*: $\bigwedge a.\ a \in A \Longrightarrow P\ a\ a$
**and** *trans*: $\bigwedge a\ b\ c.\ [\![\ a \in A;\ b \in A;\ c \in A\ ]\!] \Longrightarrow P\ a\ b \Longrightarrow P\ b\ c \Longrightarrow P\ a\ c$
**and** *antisym*: $\bigwedge a\ b.\ [\![\ a \in A;\ b \in A\ ]\!] \Longrightarrow P\ a\ b \Longrightarrow P\ b\ a \Longrightarrow a = b$
**shows** *partial-order-on* $A$ (*relation-of* $P\ A$)
⟨*proof*⟩

**lemma** *Partial-order-relation-ofI*:
**assumes** *partial-order-on* $A$ (*relation-of* $P\ A$)
**shows** *Partial-order* (*relation-of* $P\ A$)
⟨*proof*⟩

## 24.4 Orders on a type

**abbreviation** *strict-linear-order* $\equiv$ *strict-linear-order-on UNIV*

**abbreviation** *linear-order* $\equiv$ *linear-order-on UNIV*

**abbreviation** *well-order* $\equiv$ *well-order-on UNIV*

## 24.5 Order-like relations

In this subsection, we develop basic concepts and results pertaining to order-like relations, i.e., to reflexive and/or transitive and/or symmetric and/or total relations. We also further define upper and lower bounds operators.

### 24.5.1 Auxiliaries

**corollary** *well-order-on-domain*: *well-order-on* $A\ r \Longrightarrow (a,\ b) \in r \Longrightarrow a \in A \wedge b \in A$
$\langle proof \rangle$

**lemma** *well-order-on-Field*: *well-order-on* $A\ r \Longrightarrow A = \mathit{Field}\ r$
$\langle proof \rangle$

**lemma** *well-order-on-Well-order*: *well-order-on* $A\ r \Longrightarrow A = \mathit{Field}\ r \wedge$ *Well-order* $r$
$\langle proof \rangle$

**lemma** *Total-subset-Id*:
**assumes** *Total* $r$
**and** $r \subseteq \mathit{Id}$
**shows** $r = \{\} \vee (\exists\, a.\ r = \{(a,\ a)\})$
$\langle proof \rangle$

**lemma** *Linear-order-in-diff-Id*:
**assumes** *Linear-order* $r$
**and** $a \in \mathit{Field}\ r$
**and** $b \in \mathit{Field}\ r$
**shows** $(a,\ b) \in r \longleftrightarrow (b,\ a) \notin r - \mathit{Id}$
$\langle proof \rangle$

### 24.5.2 The upper and lower bounds operators

Here we define upper ("above") and lower ("below") bounds operators. We think of $r$ as a *non-strict* relation. The suffix $S$ at the names of some operators indicates that the bounds are strict – e.g., *underS* $a$ is the set of all strict lower bounds of $a$ (w.r.t. $r$). Capitalization of the first letter in the name reminds that the operator acts on sets, rather than on individual elements.

**definition** *under* :: $'a\ rel \Rightarrow\ 'a \Rightarrow\ 'a\ set$
**where** *under* $r\ a \equiv \{b.\ (b,\ a) \in r\}$

**definition** $underS :: 'a\ rel \Rightarrow 'a \Rightarrow 'a\ set$
**where** $underS\ r\ a \equiv \{b.\ b \neq a \wedge (b,\ a) \in r\}$

**definition** $Under :: 'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ set$
**where** $Under\ r\ A \equiv \{b \in Field\ r.\ \forall\, a \in A.\ (b,\ a) \in r\}$

**definition** $UnderS :: 'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ set$
**where** $UnderS\ r\ A \equiv \{b \in Field\ r.\ \forall\, a \in A.\ b \neq a \wedge (b,\ a) \in r\}$

**definition** $above :: 'a\ rel \Rightarrow 'a \Rightarrow 'a\ set$
**where** $above\ r\ a \equiv \{b.\ (a,\ b) \in r\}$

**definition** $aboveS :: 'a\ rel \Rightarrow 'a \Rightarrow 'a\ set$
**where** $aboveS\ r\ a \equiv \{b.\ b \neq a \wedge (a,\ b) \in r\}$

**definition** $Above :: 'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ set$
**where** $Above\ r\ A \equiv \{b \in Field\ r.\ \forall\, a \in A.\ (a,\ b) \in r\}$

**definition** $AboveS :: 'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ set$
**where** $AboveS\ r\ A \equiv \{b \in Field\ r.\ \forall\, a \in A.\ b \neq a \wedge (a,\ b) \in r\}$

**definition** $ofilter :: 'a\ rel \Rightarrow 'a\ set \Rightarrow bool$
**where** $ofilter\ r\ A \equiv A \subseteq Field\ r \wedge (\forall\, a \in A.\ under\ r\ a \subseteq A)$

Note: In the definitions of $Above[S]$ and $Under[S]$, we bounded comprehension by $Field\ r$ in order to properly cover the case of $A$ being empty.

**lemma** *underS-subset-under*: $underS\ r\ a \subseteq under\ r\ a$
$\langle proof \rangle$

**lemma** *underS-notIn*: $a \notin underS\ r\ a$
$\langle proof \rangle$

**lemma** *Refl-under-in*: $Refl\ r \Longrightarrow a \in Field\ r \Longrightarrow a \in under\ r\ a$
$\langle proof \rangle$

**lemma** *AboveS-disjoint*: $A \cap (AboveS\ r\ A) = \{\}$
$\langle proof \rangle$

**lemma** *in-AboveS-underS*: $a \in Field\ r \Longrightarrow a \in AboveS\ r\ (underS\ r\ a)$
$\langle proof \rangle$

**lemma** *Refl-under-underS*: $Refl\ r \Longrightarrow a \in Field\ r \Longrightarrow under\ r\ a = underS\ r\ a \cup \{a\}$
$\langle proof \rangle$

**lemma** *underS-empty*: $a \notin Field\ r \Longrightarrow underS\ r\ a = \{\}$
$\langle proof \rangle$

**lemma** *under-Field*: *under r a* $\subseteq$ *Field r*
⟨*proof*⟩

**lemma** *underS-Field*: *underS r a* $\subseteq$ *Field r*
⟨*proof*⟩

**lemma** *underS-Field2*: $a \in$ *Field r* $\Longrightarrow$ *underS r a* $\subset$ *Field r*
⟨*proof*⟩

**lemma** *underS-Field3*: *Field r* $\neq \{\}$ $\Longrightarrow$ *underS r a* $\subset$ *Field r*
⟨*proof*⟩

**lemma** *AboveS-Field*: *AboveS r A* $\subseteq$ *Field r*
⟨*proof*⟩

**lemma** *under-incr*:
**assumes** *trans r*
**and** $(a, b) \in r$
**shows** *under r a* $\subseteq$ *under r b*
⟨*proof*⟩

**lemma** *underS-incr*:
**assumes** *trans r*
**and** *antisym r*
**and** *ab*: $(a, b) \in r$
**shows** *underS r a* $\subseteq$ *underS r b*
⟨*proof*⟩

**lemma** *underS-incl-iff*:
**assumes** *LO*: *Linear-order r*
**and** *INa*: $a \in$ *Field r*
**and** *INb*: $b \in$ *Field r*
**shows** *underS r a* $\subseteq$ *underS r b* $\longleftrightarrow (a, b) \in r$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *finite-Partial-order-induct*[*consumes 3*, *case-names step*]:
**assumes** *Partial-order r*
**and** $x \in$ *Field r*
**and** *finite r*
**and** *step*: $\bigwedge x.\ x \in$ *Field r* $\Longrightarrow (\bigwedge y.\ y \in$ *aboveS r x* $\Longrightarrow P\ y) \Longrightarrow P\ x$
**shows** $P\ x$
⟨*proof*⟩

**lemma** *finite-Linear-order-induct*[*consumes 3*, *case-names step*]:
**assumes** *Linear-order r*
**and** $x \in$ *Field r*
**and** *finite r*
**and** *step*: $\bigwedge x.\ x \in$ *Field r* $\Longrightarrow (\bigwedge y.\ y \in$ *aboveS r x* $\Longrightarrow P\ y) \Longrightarrow P\ x$

**shows** $P\ x$
⟨*proof*⟩

## 24.6 Variations on Well-Founded Relations

This subsection contains some variations of the results from *HOL.Wellfounded*:

- means for slightly more direct definitions by well-founded recursion;
- variations of well-founded induction;
- means for proving a linear order to be a well-order.

### 24.6.1 Characterizations of well-foundedness

A transitive relation is well-founded iff it is "locally" well-founded, i.e., iff its restriction to the lower bounds of of any element is well-founded.

**lemma** *trans-wf-iff*:
**assumes** $trans\ r$
**shows** $wf\ r \longleftrightarrow (\forall\, a.\ wf\ (r \cap (r^{-1}\,``\{a\} \times r^{-1}\,``\{a\})))$
⟨*proof*⟩

A transitive relation is well-founded if all initial segments are finite.

**corollary** *wf-finite-segments*:
**assumes** $irrefl\ r$ **and** $trans\ r$ **and** $\bigwedge x.\ finite\ \{y.\ (y,\ x) \in r\}$
**shows** $wf\ r$
⟨*proof*⟩

The next lemma is a variation of *wf-eq-minimal* from Wellfounded, allowing one to assume the set included in the field.

**lemma** *wf-eq-minimal2*: $wf\ r \longleftrightarrow (\forall\, A.\ A \subseteq Field\ r \wedge A \neq \{\} \longrightarrow (\exists\, a \in A.\ \forall\, a' \in A.\ (a',\ a) \notin r))$
⟨*proof*⟩

### 24.6.2 Characterizations of well-foundedness

The next lemma and its corollary enable one to prove that a linear order is a well-order in a way which is more standard than via well-foundedness of the strict version of the relation.

**lemma** *Linear-order-wf-diff-Id*:
**assumes** $Linear\text{-}order\ r$
**shows** $wf\ (r - Id) \longleftrightarrow (\forall\, A \subseteq Field\ r.\ A \neq \{\} \longrightarrow (\exists\, a \in A.\ \forall\, a' \in A.\ (a,\ a') \in r))$
⟨*proof*⟩

**corollary** *Linear-order-Well-order-iff*:
*Linear-order* $r \Longrightarrow$
*Well-order* $r \longleftrightarrow (\forall\, A \subseteq \mathit{Field}\ r.\ A \neq \{\} \longrightarrow (\exists\, a \in A.\ \forall\, a' \in A.\ (a,\ a') \in r))$
⟨*proof*⟩

**end**

# 25 Hilbert's Epsilon-Operator and the Axiom of Choice

**theory** *Hilbert-Choice*
**imports** *Wellfounded*
**keywords** *specification* :: *thy-goal-defn*
**begin**

## 25.1 Hilbert's epsilon

**axiomatization** *Eps* :: $('a \Rightarrow \mathit{bool}) \Rightarrow 'a$
**where** *someI*: $P\ x \Longrightarrow P\ (\mathit{Eps}\ P)$

**syntax** (*epsilon*)
*-Eps* :: *pttrn* ⇒ *bool* ⇒ *'a* (‹(‹*indent=3 notation=*‹*binder* ϵ››ϵ -./ -)› [*0*, *10*] *10*)
**syntax** (*input*)
*-Eps* :: *pttrn* ⇒ *bool* ⇒ *'a* (‹(‹*indent=3 notation=*‹*binder* @››@ -./ -)› [*0*, *10*] *10*)
**syntax**
*-Eps* :: *pttrn* ⇒ *bool* ⇒ *'a* (‹(‹*indent=3 notation=*‹*binder SOME*››*SOME* -./ -)› [*0*, *10*] *10*)

**syntax-consts** *-Eps* ⇌ *Eps*

**translations**
$\mathit{SOME}\ x.\ P \rightleftharpoons \mathit{CONST}\ \mathit{Eps}\ (\lambda x.\ P)$

⟨*ML*⟩

**definition** *inv-into* :: $'a\ \mathit{set} \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a)$ **where**
$\mathit{inv\text{-}into}\ A\ f = (\lambda x.\ \mathit{SOME}\ y.\ y \in A \wedge f\ y = x)$

**lemma** *inv-into-def2*: $\mathit{inv\text{-}into}\ A\ f\ x = (\mathit{SOME}\ y.\ y \in A \wedge f\ y = x)$
⟨*proof*⟩

**abbreviation** *inv* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a)$ **where**
$\mathit{inv} \equiv \mathit{inv\text{-}into}\ \mathit{UNIV}$

## 25.2 Hilbert's Epsilon-operator

**lemma** *Eps-cong*:
**assumes** $\bigwedge x.\ P\ x = Q\ x$

**shows** $Eps\ P = Eps\ Q$
$\langle proof \rangle$

Easier to use than *someI* if the witness comes from an existential formula.

**lemma** *someI-ex* [*elim?*]: $\exists x.\ P\ x \Longrightarrow P\ (SOME\ x.\ P\ x)$
$\langle proof \rangle$

**lemma** *some-eq-imp*:
**assumes** $Eps\ P = a\ P\ b$ **shows** $P\ a$
$\langle proof \rangle$

Easier to use than *someI* because the conclusion has only one occurrence of $P$.

**lemma** *someI2*: $P\ a \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (SOME\ x.\ P\ x)$
$\langle proof \rangle$

Easier to use than *someI2* if the witness comes from an existential formula.

**lemma** *someI2-ex*: $\exists a.\ P\ a \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (SOME\ x.\ P\ x)$
$\langle proof \rangle$

**lemma** *someI2-bex*: $\exists a{\in}A.\ P\ a \Longrightarrow (\bigwedge x.\ x \in A \land P\ x \Longrightarrow Q\ x) \Longrightarrow Q\ (SOME\ x.\ x \in A \land P\ x)$
$\langle proof \rangle$

**lemma** *some-equality* [*intro*]: $P\ a \Longrightarrow (\bigwedge x.\ P\ x \Longrightarrow x = a) \Longrightarrow (SOME\ x.\ P\ x) = a$
$\langle proof \rangle$

**lemma** *some1-equality*: $\exists !x.\ P\ x \Longrightarrow P\ a \Longrightarrow (SOME\ x.\ P\ x) = a$
$\langle proof \rangle$

**lemma** *some-eq-ex*: $P\ (SOME\ x.\ P\ x) \longleftrightarrow (\exists x.\ P\ x)$
$\langle proof \rangle$

**lemma** *some-in-eq*: $(SOME\ x.\ x \in A) \in A \longleftrightarrow A \neq \{\}$
$\langle proof \rangle$

**lemma** *some-eq-trivial* [*simp*]: $(SOME\ y.\ y = x) = x$
$\langle proof \rangle$

**lemma** *some-sym-eq-trivial* [*simp*]: $(SOME\ y.\ x = y) = x$
$\langle proof \rangle$

## 25.3 Axiom of Choice, Proved Using the Description Operator

**lemma** *choice*: $\forall x.\ \exists y.\ Q\ x\ y \Longrightarrow \exists f.\ \forall x.\ Q\ x\ (f\ x)$
$\langle proof \rangle$

**lemma** *bchoice*: $\forall x \in S.\ \exists y.\ Q\ x\ y \Longrightarrow \exists f.\ \forall x \in S.\ Q\ x\ (f\ x)$
⟨*proof*⟩

**lemma** *choice-iff*: $(\forall x.\ \exists y.\ Q\ x\ y) \longleftrightarrow (\exists f.\ \forall x.\ Q\ x\ (f\ x))$
⟨*proof*⟩

**lemma** *choice-iff'*: $(\forall x.\ P\ x \longrightarrow (\exists y.\ Q\ x\ y)) \longleftrightarrow (\exists f.\ \forall x.\ P\ x \longrightarrow Q\ x\ (f\ x))$
⟨*proof*⟩

**lemma** *bchoice-iff*: $(\forall x \in S.\ \exists y.\ Q\ x\ y) \longleftrightarrow (\exists f.\ \forall x \in S.\ Q\ x\ (f\ x))$
⟨*proof*⟩

**lemma** *bchoice-iff'*: $(\forall x \in S.\ P\ x \longrightarrow (\exists y.\ Q\ x\ y)) \longleftrightarrow (\exists f.\ \forall x \in S.\ P\ x \longrightarrow Q\ x\ (f\ x))$
⟨*proof*⟩

**lemma** *dependent-nat-choice*:
**assumes** *1*: $\exists x.\ P\ 0\ x$
**and** *2*: $\bigwedge x\ n.\ P\ n\ x \Longrightarrow \exists y.\ P\ (Suc\ n)\ y \wedge Q\ n\ x\ y$
**shows** $\exists f.\ \forall n.\ P\ n\ (f\ n) \wedge Q\ n\ (f\ n)\ (f\ (Suc\ n))$
⟨*proof*⟩

**lemma** *finite-subset-Union*:
**assumes** *finite* $A\ A \subseteq \bigcup \mathcal{B}$
**obtains** $\mathcal{F}$ **where** *finite* $\mathcal{F}\ \mathcal{F} \subseteq \mathcal{B}\ A \subseteq \bigcup \mathcal{F}$
⟨*proof*⟩

## 25.4 Getting an element of a nonempty set

**definition** *some-elem* :: $'a\ set \Rightarrow 'a$
**where** *some-elem* $A = (SOME\ x.\ x \in A)$

**lemma** *some-elem-eq* [*simp*]: *some-elem* $\{x\} = x$
⟨*proof*⟩

**lemma** *some-elem-nonempty*: $A \neq \{\} \Longrightarrow$ *some-elem* $A \in A$
⟨*proof*⟩

**lemma** *is-singleton-some-elem*: *is-singleton* $A \longleftrightarrow A = \{$*some-elem* $A\}$
⟨*proof*⟩

**lemma** *some-elem-image-unique*:
**assumes** $A \neq \{\}$
**and** $*$: $\bigwedge y.\ y \in A \Longrightarrow f\ y = a$
**shows** *some-elem* $(f\ `\ A) = a$
⟨*proof*⟩

## 25.5 Function Inverse

**lemma** *inv-def*: *inv* $f = (\lambda y.\ SOME\ x.\ f\ x = y)$

⟨proof⟩

**lemma** *inv-into-into*: $x \in f \text{ ` } A \Longrightarrow \textit{inv-into } A\ f\ x \in A$
⟨proof⟩

**lemma** *inv-identity* [*simp*]: $\textit{inv}\ (\lambda a.\ a) = (\lambda a.\ a)$
⟨proof⟩

**lemma** *inv-id* [*simp*]: $\textit{inv id} = \textit{id}$
⟨proof⟩

**lemma** *inv-into-f-f* [*simp*]: $\textit{inj-on } f\ A \Longrightarrow x \in A \Longrightarrow \textit{inv-into } A\ f\ (f\ x) = x$
⟨proof⟩

**lemma** *inv-f-f*: $\textit{inj } f \Longrightarrow \textit{inv } f\ (f\ x) = x$
⟨proof⟩

**lemma** *f-inv-into-f*: $y \in f\text{`}A \Longrightarrow f\ (\textit{inv-into } A\ f\ y) = y$
⟨proof⟩

**lemma** *inv-into-f-eq*: $\textit{inj-on } f\ A \Longrightarrow x \in A \Longrightarrow f\ x = y \Longrightarrow \textit{inv-into } A\ f\ y = x$
⟨proof⟩

**lemma** *inv-f-eq*: $\textit{inj } f \Longrightarrow f\ x = y \Longrightarrow \textit{inv } f\ y = x$
⟨proof⟩

**lemma** *inj-imp-inv-eq*: $\textit{inj } f \Longrightarrow \forall x.\ f\ (g\ x) = x \Longrightarrow \textit{inv } f = g$
⟨proof⟩

But is it useful?

**lemma** *inj-transfer*:
**assumes** *inj*: *inj f*
**and** *minor*: $\bigwedge y.\ y \in \textit{range } f \Longrightarrow P\ (\textit{inv } f\ y)$
**shows** $P\ x$
⟨proof⟩

**lemma** *inj-iff*: $\textit{inj } f \longleftrightarrow \textit{inv } f \circ f = \textit{id}$
⟨proof⟩

**lemma** *inv-o-cancel*[*simp*]: $\textit{inj } f \Longrightarrow \textit{inv } f \circ f = \textit{id}$
⟨proof⟩

**lemma** *o-inv-o-cancel*[*simp*]: $\textit{inj } f \Longrightarrow g \circ \textit{inv } f \circ f = g$
⟨proof⟩

**lemma** *inv-into-image-cancel*[*simp*]: $\textit{inj-on } f\ A \Longrightarrow S \subseteq A \Longrightarrow \textit{inv-into } A\ f \text{ ` } f \text{ ` } S = S$
⟨proof⟩

**lemma** *inj-imp-surj-inv*: $inj\ f \Longrightarrow surj\ (inv\ f)$
⟨*proof*⟩

**lemma** *surj-f-inv-f*: $surj\ f \Longrightarrow f\ (inv\ f\ y) = y$
⟨*proof*⟩

**lemma** *bij-inv-eq-iff*: $bij\ p \Longrightarrow x = inv\ p\ y \longleftrightarrow p\ x = y$
⟨*proof*⟩

**lemma** *inv-into-injective*:
**assumes** *eq*: $inv\text{-}into\ A\ f\ x = inv\text{-}into\ A\ f\ y$
**and** *x*: $x \in f`A$
**and** *y*: $y \in f`A$
**shows** $x = y$
⟨*proof*⟩

**lemma** *inj-on-inv-into*: $B \subseteq f`A \Longrightarrow inj\text{-}on\ (inv\text{-}into\ A\ f)\ B$
⟨*proof*⟩

**lemma** *inj-imp-bij-betw-inv*: $inj\ f \Longrightarrow bij\text{-}betw\ (inv\ f)\ (f\ `\ M)\ M$
⟨*proof*⟩

**lemma** *bij-betw-inv-into*: $bij\text{-}betw\ f\ A\ B \Longrightarrow bij\text{-}betw\ (inv\text{-}into\ A\ f)\ B\ A$
⟨*proof*⟩

**lemma** *surj-imp-inj-inv*: $surj\ f \Longrightarrow inj\ (inv\ f)$
⟨*proof*⟩

**lemma** *surj-iff*: $surj\ f \longleftrightarrow f \circ inv\ f = id$
⟨*proof*⟩

**lemma** *surj-iff-all*: $surj\ f \longleftrightarrow (\forall\, x.\ f\ (inv\ f\ x) = x)$
⟨*proof*⟩

**lemma** *surj-imp-inv-eq*:
**assumes** *surj f* **and** *gf*: $\bigwedge x.\ g\ (f\ x) = x$
**shows** $inv\ f = g$
⟨*proof*⟩

**lemma** *bij-imp-bij-inv*: $bij\ f \Longrightarrow bij\ (inv\ f)$
⟨*proof*⟩

**lemma** *inv-equality*: $(\bigwedge x.\ g\ (f\ x) = x) \Longrightarrow (\bigwedge y.\ f\ (g\ y) = y) \Longrightarrow inv\ f = g$
⟨*proof*⟩

**lemma** *inv-inv-eq*: $bij\ f \Longrightarrow inv\ (inv\ f) = f$
⟨*proof*⟩

$bij\ (inv\ f)$ implies little about *f*. Consider $f :: bool \Rightarrow bool$ such that *f True*

$= f\ False = True$. Then it ia consistent with axiom *someI* that *inv f* could be any function at all, including the identity function. If $inv\ f = id$ then *inv f* is a bijection, but *inj f*, *surj f* and $inv\ (inv\ f) = f$ all fail.

**lemma** *inv-into-comp*:
  $inj\text{-}on\ f\ (g\ `\ A) \Longrightarrow inj\text{-}on\ g\ A \Longrightarrow x \in f\ `\ g\ `\ A \Longrightarrow$
    $inv\text{-}into\ A\ (f \circ g)\ x = (inv\text{-}into\ A\ g \circ inv\text{-}into\ (g\ `\ A)\ f)\ x$
  ⟨*proof*⟩

**lemma** *o-inv-distrib*: $bij\ f \Longrightarrow bij\ g \Longrightarrow inv\ (f \circ g) = inv\ g \circ inv\ f$
  ⟨*proof*⟩

**lemma** *image-f-inv-f*: $surj\ f \Longrightarrow f\ `\ (inv\ f\ `\ A) = A$
  ⟨*proof*⟩

**lemma** *image-inv-f-f*: $inj\ f \Longrightarrow inv\ f\ `\ (f\ `\ A) = A$
  ⟨*proof*⟩

**lemma** *bij-image-Collect-eq*:
  **assumes** *bij f*
  **shows** $f\ `\ Collect\ P = \{y.\ P\ (inv\ f\ y)\}$
⟨*proof*⟩

**lemma** *bij-vimage-eq-inv-image*:
  **assumes** *bij f*
  **shows** $f\ -`\ A = inv\ f\ `\ A$
⟨*proof*⟩

**lemma** *inv-fn-o-fn-is-id*:
  **fixes** $f{::}'a \Rightarrow 'a$
  **assumes** *bij f*
  **shows** $((inv\ f)^{\frown\frown}n)\ o\ (f^{\frown\frown}n) = (\lambda x.\ x)$
⟨*proof*⟩

**lemma** *fn-o-inv-fn-is-id*:
  **fixes** $f{::}'a \Rightarrow 'a$
  **assumes** *bij f*
  **shows** $(f^{\frown\frown}n)\ o\ ((inv\ f)^{\frown\frown}n) = (\lambda x.\ x)$
⟨*proof*⟩

**lemma** *inv-fn*:
  **fixes** $f{::}'a \Rightarrow 'a$
  **assumes** *bij f*
  **shows** $inv\ (f^{\frown\frown}n) = ((inv\ f)^{\frown\frown}n)$
⟨*proof*⟩

**lemma** *funpow-inj-finite*:
  **assumes** ‹*inj p*› ‹$finite\ \{y.\ \exists\, n.\ y = (p\ ^{\frown\frown}\ n)\ x\}$›
  **obtains** *n* **where** ‹$n > 0$› ‹$(p\ ^{\frown\frown}\ n)\ x = x$›
⟨*proof*⟩

**lemma** *mono-inv*:
  **fixes** $f{::}'a{::}linorder \Rightarrow 'b{::}linorder$
  **assumes** *mono f bij f*
  **shows** *mono* (*inv f*)
⟨*proof*⟩

**lemma** *strict-mono-inv-on-range*:
  **fixes** $f :: 'a{::}linorder \Rightarrow 'b{::}order$
  **assumes** *strict-mono f*
  **shows** *strict-mono-on* (*range f*) (*inv f*)
⟨*proof*⟩

**lemma** *mono-bij-Inf*:
  **fixes** $f :: 'a{::}complete\text{-}linorder \Rightarrow 'b{::}complete\text{-}linorder$
  **assumes** *mono f bij f*
  **shows** $f\ (Inf\ A) = Inf\ (f`A)$
⟨*proof*⟩

**lemma** *finite-fun-UNIVD1*:
  **assumes** *fin*: *finite* $(UNIV :: ('a \Rightarrow 'b)\ set)$
    **and** *card*: *card* $(UNIV :: 'b\ set) \neq Suc\ 0$
  **shows** *finite* $(UNIV :: 'a\ set)$
⟨*proof*⟩

Every infinite set contains a countable subset. More precisely we show that a set $S$ is infinite if and only if there exists an injective function from the naturals into $S$.

The "only if" direction is harder because it requires the construction of a sequence of pairwise different elements of an infinite set $S$. The idea is to construct a sequence of non-empty and infinite subsets of $S$ obtained by successively removing elements of $S$.

**lemma** *infinite-countable-subset*:
  **assumes** *inf*: $\neg$ *finite S*
  **shows** $\exists f{::}nat \Rightarrow 'a.\ inj\ f \wedge range\ f \subseteq S$
  — Courtesy of Stephan Merz
⟨*proof*⟩

**lemma** *infinite-iff-countable-subset*: $\neg$ *finite S* $\longleftrightarrow (\exists f{::}nat \Rightarrow 'a.\ inj\ f \wedge range\ f \subseteq S)$
  — Courtesy of Stephan Merz
  ⟨*proof*⟩

**lemma** *image-inv-into-cancel*:
  **assumes** *surj*: $f`A = A'$
    **and** *sub*: $B' \subseteq A'$
  **shows** $f\ `((inv\text{-}into\ A\ f)`B') = B'$

⟨*proof*⟩

**lemma** *inv-into-inv-into-eq*:
**assumes** *bij-betw f A A′*
**and** *a*: $a \in A$
**shows** *inv-into A′* (*inv-into A f*) $a = f\ a$
⟨*proof*⟩

**lemma** *inj-on-iff-surj*:
**assumes** $A \neq \{\}$
**shows** $(\exists f.\ \mathit{inj\text{-}on}\ f\ A \land f\ `\ A \subseteq A') \longleftrightarrow (\exists g.\ g\ `\ A' = A)$
⟨*proof*⟩

**lemma** *Ex-inj-on-UNION-Sigma*:
$\exists f.\ (\mathit{inj\text{-}on}\ f\ (\bigcup i \in I.\ A\ i) \land f\ `\ (\bigcup i \in I.\ A\ i) \subseteq (\mathit{SIGMA}\ i : I.\ A\ i))$
⟨*proof*⟩

**lemma** *inv-unique-comp*:
**assumes** *fg*: $f \circ g = id$
**and** *gf*: $g \circ f = id$
**shows** $\mathit{inv}\ f = g$
⟨*proof*⟩

**lemma** *subset-image-inj*:
$S \subseteq f\ `\ T \longleftrightarrow (\exists U.\ U \subseteq T \land \mathit{inj\text{-}on}\ f\ U \land S = f\ `\ U)$
⟨*proof*⟩

## 25.6 Other Consequences of Hilbert's Epsilon

Hilbert's Epsilon and the *split* Operator

Looping simprule!

**lemma** *split-paired-Eps*: $(\mathit{SOME}\ x.\ P\ x) = (\mathit{SOME}\ (a, b).\ P\ (a, b))$
⟨*proof*⟩

**lemma** *Eps-case-prod*: $\mathit{Eps}\ (\mathit{case\text{-}prod}\ P) = (\mathit{SOME}\ xy.\ P\ (\mathit{fst}\ xy)\ (\mathit{snd}\ xy))$
⟨*proof*⟩

**lemma** *Eps-case-prod-eq* [*simp*]: $(\mathit{SOME}\ (x', y').\ x = x' \land y = y') = (x, y)$
⟨*proof*⟩

A relation is wellfounded iff it has no infinite descending chain.

**lemma** *wf-iff-no-infinite-down-chain*: $\mathit{wf}\ r \longleftrightarrow (\nexists f.\ \forall i.\ (f\ (\mathit{Suc}\ i), f\ i) \in r)$
(**is** - $\longleftrightarrow \neg$ *?ex*)
⟨*proof*⟩

**lemma** *wf-no-infinite-down-chainE*:
**assumes** *wf r*
**obtains** *k* **where** $(f\ (\mathit{Suc}\ k), f\ k) \notin r$

⟨*proof*⟩

A dynamically-scoped fact for TFL

**lemma** *tfl-some*: $\forall P\ x.\ P\ x \longrightarrow P\ (Eps\ P)$
⟨*proof*⟩

## 25.7 An aside: bounded accessible part

Finite monotone eventually stable sequences

**lemma** *finite-mono-remains-stable-implies-strict-prefix*:
**fixes** $f :: nat \Rightarrow 'a::order$
**assumes** $S$: $finite\ (range\ f)\ mono\ f$
**and** $eq$: $\forall n.\ f\ n = f\ (Suc\ n) \longrightarrow f\ (Suc\ n) = f\ (Suc\ (Suc\ n))$
**shows** $\exists N.\ (\forall n \le N.\ \forall m \le N.\ m < n \longrightarrow f\ m < f\ n) \wedge (\forall n \ge N.\ f\ N = f\ n)$
⟨*proof*⟩

**lemma** *finite-mono-strict-prefix-implies-finite-fixpoint*:
**fixes** $f :: nat \Rightarrow 'a\ set$
**assumes** $S$: $\bigwedge i.\ f\ i \subseteq S\ finite\ S$
**and** $ex$: $\exists N.\ (\forall n \le N.\ \forall m \le N.\ m < n \longrightarrow f\ m \subset f\ n) \wedge (\forall n \ge N.\ f\ N = f\ n)$
**shows** $f\ (card\ S) = (\bigcup n.\ f\ n)$
⟨*proof*⟩

## 25.8 More on injections, bijections, and inverses

**locale** *bijection* =
**fixes** $f :: 'a \Rightarrow 'a$
**assumes** *bij*: *bij f*
**begin**

**lemma** *bij-inv*: *bij* (*inv f*)
⟨*proof*⟩

**lemma** *surj* [*simp*]: *surj f*
⟨*proof*⟩

**lemma** *inj*: *inj f*
⟨*proof*⟩

**lemma** *surj-inv* [*simp*]: *surj* (*inv f*)
⟨*proof*⟩

**lemma** *inj-inv*: *inj* (*inv f*)
⟨*proof*⟩

**lemma** *eqI*: $f\ a = f\ b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *eq-iff* [*simp*]: $f\ a = f\ b \longleftrightarrow a = b$

⟨*proof*⟩

**lemma** *eq-invI*: $inv\ f\ a = inv\ f\ b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *eq-inv-iff* [*simp*]: $inv\ f\ a = inv\ f\ b \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *inv-left* [*simp*]: $inv\ f\ (f\ a) = a$
⟨*proof*⟩

**lemma** *inv-comp-left* [*simp*]: $inv\ f \circ f = id$
⟨*proof*⟩

**lemma** *inv-right* [*simp*]: $f\ (inv\ f\ a) = a$
⟨*proof*⟩

**lemma** *inv-comp-right* [*simp*]: $f \circ inv\ f = id$
⟨*proof*⟩

**lemma** *inv-left-eq-iff* [*simp*]: $inv\ f\ a = b \longleftrightarrow f\ b = a$
⟨*proof*⟩

**lemma** *inv-right-eq-iff* [*simp*]: $b = inv\ f\ a \longleftrightarrow f\ b = a$
⟨*proof*⟩

**end**

**lemma** *infinite-imp-bij-betw*:
**assumes** *infinite*: $\neg$ *finite* $A$
**shows** $\exists\, h.\ bij\text{-}betw\ h\ A\ (A - \{a\})$
⟨*proof*⟩

**lemma** *infinite-imp-bij-betw2*:
**assumes** $\neg$ *finite* $A$
**shows** $\exists\, h.\ bij\text{-}betw\ h\ A\ (A \cup \{a\})$
⟨*proof*⟩

**lemma** *bij-betw-inv-into-left*: $bij\text{-}betw\ f\ A\ A' \Longrightarrow a \in A \Longrightarrow inv\text{-}into\ A\ f\ (f\ a) = a$
⟨*proof*⟩

**lemma** *bij-betw-inv-into-right*: $bij\text{-}betw\ f\ A\ A' \Longrightarrow a' \in A' \Longrightarrow f\ (inv\text{-}into\ A\ f\ a')$
$= a'$
⟨*proof*⟩

**lemma** *bij-betw-inv-into-subset*:
$bij\text{-}betw\ f\ A\ A' \Longrightarrow B \subseteq A \Longrightarrow f\ `\ B = B' \Longrightarrow bij\text{-}betw\ (inv\text{-}into\ A\ f)\ B'\ B$
⟨*proof*⟩

## 25.9 Specification package – Hilbertized version

**lemma** *exE-some*: $Ex\ P \Longrightarrow c \equiv Eps\ P \Longrightarrow P\ c$
⟨*proof*⟩

⟨*ML*⟩

## 25.10 Complete Distributive Lattices – Properties depending on Hilbert Choice

**context** *complete-distrib-lattice*
**begin**

**lemma** *Sup-Inf*: $\bigsqcup\ (Inf\ `\ A) = \bigsqcap\ (Sup\ `\ \{f\ `\ A\ |f.\ \forall B{\in}A.\ f\ B \in B\})$
⟨*proof*⟩

**lemma** *dual-complete-distrib-lattice*:
*class.complete-distrib-lattice Sup Inf sup* $(\geq)$ $(>)$ *inf* $\top$ $\bot$
⟨*proof*⟩

**lemma** *sup-Inf*: $a \sqcup \bigsqcap B = \bigsqcap((\sqcup)\ a\ `\ B)$
⟨*proof*⟩

**lemma** *inf-Sup*: $a \sqcap \bigsqcup B = \bigsqcup((\sqcap)\ a\ `\ B)$
⟨*proof*⟩

**lemma** *INF-SUP*: $(\bigsqcap y.\ \bigsqcup x.\ P\ x\ y) = (\bigsqcup f.\ \bigsqcap x.\ P\ (f\ x)\ x)$
⟨*proof*⟩

**lemma** *INF-SUP-set*: $(\bigsqcap B{\in}A.\ \bigsqcup(g\ `\ B)) = (\bigsqcup B{\in}\{f\ `\ A\ |f.\ \forall C{\in}A.\ f\ C \in C\}.\ \bigsqcap(g\ `\ B))$
(**is** - = $(\bigsqcup B{\in}?F.$ -))
⟨*proof*⟩

**lemma** *SUP-INF*: $(\bigsqcup y.\ \bigsqcap x.\ P\ x\ y) = (\bigsqcap x.\ \bigsqcup y.\ P\ (x\ y)\ y)$
⟨*proof*⟩

**lemma** *SUP-INF-set*: $(\bigsqcup x{\in}A.\ \bigsqcap\ (g\ `\ x)) = (\bigsqcap x{\in}\{f\ `\ A\ |f.\ \forall Y{\in}A.\ f\ Y \in Y\}.\ \bigsqcup\ (g\ `\ x))$
⟨*proof*⟩

**end**

**context** *complete-distrib-lattice*
**begin**

**lemma** *sup-INF*: $a \sqcup (\bigsqcap b{\in}B.\ f\ b) = (\bigsqcap b{\in}B.\ a \sqcup f\ b)$
⟨*proof*⟩

**lemma** *inf-SUP*: $a \sqcap (\bigsqcup b{\in}B.\ f\ b) = (\bigsqcup b{\in}B.\ a \sqcap f\ b)$
⟨*proof*⟩

**lemma** *Inf-sup*: $\bigsqcap B \sqcup a = (\bigsqcap b{\in}B.\ b \sqcup a)$
⟨*proof*⟩

**lemma** *Sup-inf*: $\bigsqcup B \sqcap a = (\bigsqcup b{\in}B.\ b \sqcap a)$
⟨*proof*⟩

**lemma** *INF-sup*: $(\bigsqcap b{\in}B.\ f\ b) \sqcup a = (\bigsqcap b{\in}B.\ f\ b \sqcup a)$
⟨*proof*⟩

**lemma** *SUP-inf*: $(\bigsqcup b{\in}B.\ f\ b) \sqcap a = (\bigsqcup b{\in}B.\ f\ b \sqcap a)$
⟨*proof*⟩

**lemma** *Inf-sup-eq-top-iff*: $(\bigsqcap B \sqcup a = \top) \longleftrightarrow (\forall b{\in}B.\ b \sqcup a = \top)$
⟨*proof*⟩

**lemma** *Sup-inf-eq-bot-iff*: $(\bigsqcup B \sqcap a = \bot) \longleftrightarrow (\forall b{\in}B.\ b \sqcap a = \bot)$
⟨*proof*⟩

**lemma** *INF-sup-distrib2*: $(\bigsqcap a{\in}A.\ f\ a) \sqcup (\bigsqcap b{\in}B.\ g\ b) = (\bigsqcap a{\in}A.\ \bigsqcap b{\in}B.\ f\ a \sqcup g\ b)$
⟨*proof*⟩

**lemma** *SUP-inf-distrib2*: $(\bigsqcup a{\in}A.\ f\ a) \sqcap (\bigsqcup b{\in}B.\ g\ b) = (\bigsqcup a{\in}A.\ \bigsqcup b{\in}B.\ f\ a \sqcap g\ b)$
⟨*proof*⟩

**end**

**instantiation** *set* :: (*type*) *complete-distrib-lattice*
**begin**
**instance** ⟨*proof*⟩
**end**

**instance** *set* :: (*type*) *complete-boolean-algebra* ⟨*proof*⟩

**instantiation** *fun* :: (*type*, *complete-distrib-lattice*) *complete-distrib-lattice*
**begin**
**instance** ⟨*proof*⟩
**end**

**instance** *fun* :: (*type*, *complete-boolean-algebra*) *complete-boolean-algebra* ⟨*proof*⟩

**context** *complete-linorder*
**begin**

**subclass** *complete-distrib-lattice*

⟨*proof*⟩
**end**

**end**

# 26 Zorn's Lemma and the Well-ordering Theorem

**theory** *Zorn*
  **imports** *Order-Relation Hilbert-Choice*
**begin**

## 26.1 Zorn's Lemma for the Subset Relation

### 26.1.1 Results that do not require an order

Let $P$ be a binary predicate on the set $A$.

**locale** *pred-on* =
  **fixes** $A$ :: $'a$ *set*
    **and** $P$ :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** ‹⊏› *50*)
**begin**

**abbreviation** *Peq* :: $'a \Rightarrow 'a \Rightarrow bool$ (**infix** ‹⊑› *50*)
  **where** $x \sqsubseteq y \equiv P^{==}\ x\ y$

A chain is a totally ordered subset of $A$.

**definition** *chain* :: $'a\ set \Rightarrow bool$
  **where** $chain\ C \longleftrightarrow C \subseteq A \wedge (\forall x{\in}C.\ \forall y{\in}C.\ x \sqsubseteq y \vee y \sqsubseteq x)$

We call a chain that is a proper superset of some set $X$, but not necessarily a chain itself, a superchain of $X$.

**abbreviation** *superchain* :: $'a\ set \Rightarrow 'a\ set \Rightarrow bool$ (**infix** ‹<c› *50*)
  **where** $X <c\ C \equiv chain\ C \wedge X \subset C$

A maximal chain is a chain that does not have a superchain.

**definition** *maxchain* :: $'a\ set \Rightarrow bool$
  **where** $maxchain\ C \longleftrightarrow chain\ C \wedge (\nexists S.\ C <c\ S)$

We define the successor of a set to be an arbitrary superchain, if such exists, or the set itself, otherwise.

**definition** *suc* :: $'a\ set \Rightarrow 'a\ set$
  **where** $suc\ C = (if\ \neg\ chain\ C \vee maxchain\ C\ then\ C\ else\ (SOME\ D.\ C <c\ D))$

**lemma** *chainI* [*Pure.intro?*]: $C \subseteq A \Longrightarrow (\bigwedge x\ y.\ x \in C \Longrightarrow y \in C \Longrightarrow x \sqsubseteq y \vee y \sqsubseteq x) \Longrightarrow chain\ C$
  ⟨*proof*⟩

**lemma** *chain-total*: $chain\ C \Longrightarrow x \in C \Longrightarrow y \in C \Longrightarrow x \sqsubseteq y \vee y \sqsubseteq x$

⟨*proof*⟩

**lemma** *not-chain-suc* [*simp*]: $\neg$ *chain* $X \Longrightarrow$ *suc* $X = X$
⟨*proof*⟩

**lemma** *maxchain-suc* [*simp*]: *maxchain* $X \Longrightarrow$ *suc* $X = X$
⟨*proof*⟩

**lemma** *suc-subset*: $X \subseteq$ *suc* $X$
⟨*proof*⟩

**lemma** *chain-empty* [*simp*]: *chain* $\{\}$
⟨*proof*⟩

**lemma** *not-maxchain-Some*: *chain* $C \Longrightarrow \neg$ *maxchain* $C \Longrightarrow C <c$ (*SOME* $D$. $C <c$ $D$)
⟨*proof*⟩

**lemma** *suc-not-equals*: *chain* $C \Longrightarrow \neg$ *maxchain* $C \Longrightarrow$ *suc* $C \neq C$
⟨*proof*⟩

**lemma** *subset-suc*:
**assumes** $X \subseteq Y$
**shows** $X \subseteq$ *suc* $Y$
⟨*proof*⟩

We build a set $\mathcal{C}$ that is closed under applications of *suc* and contains the union of all its subsets.

**inductive-set** *suc-Union-closed* (‹$\mathcal{C}$›)
**where**
*suc*: $X \in \mathcal{C} \Longrightarrow$ *suc* $X \in \mathcal{C}$
| *Union* [*unfolded Pow-iff*]: $X \in$ *Pow* $\mathcal{C} \Longrightarrow \bigcup X \in \mathcal{C}$

Since the empty set as well as the set itself is a subset of every set, $\mathcal{C}$ contains at least $\{\} \in \mathcal{C}$ and $\bigcup\ \mathcal{C} \in \mathcal{C}$.

**lemma** *suc-Union-closed-empty*: $\{\} \in \mathcal{C}$
**and** *suc-Union-closed-Union*: $\bigcup\mathcal{C} \in \mathcal{C}$
⟨*proof*⟩

Thus closure under *suc* will hit a maximal chain eventually, as is shown below.

**lemma** *suc-Union-closed-induct* [*consumes 1*, *case-names suc Union*, *induct pred*: *suc-Union-closed*]:
**assumes** $X \in \mathcal{C}$
**and** $\bigwedge X.\ X \in \mathcal{C} \Longrightarrow Q\ X \Longrightarrow Q$ (*suc* $X$)
**and** $\bigwedge X.\ X \subseteq \mathcal{C} \Longrightarrow \forall x{\in}X.\ Q\ x \Longrightarrow Q\ (\bigcup X)$
**shows** $Q\ X$
⟨*proof*⟩

**lemma** *suc-Union-closed-cases* [*consumes 1*, *case-names suc Union*, *cases pred*: *suc-Union-closed*]:
  **assumes** $X \in \mathcal{C}$
    **and** $\bigwedge Y.\ X = suc\ Y \Longrightarrow Y \in \mathcal{C} \Longrightarrow Q$
    **and** $\bigwedge Y.\ X = \bigcup Y \Longrightarrow Y \subseteq \mathcal{C} \Longrightarrow Q$
  **shows** $Q$
  ⟨*proof*⟩

On chains, *suc* yields a chain.

**lemma** *chain-suc*:
  **assumes** *chain X*
  **shows** *chain* (*suc X*)
  ⟨*proof*⟩

**lemma** *chain-sucD*:
  **assumes** *chain X*
  **shows** $suc\ X \subseteq A \wedge chain\ (suc\ X)$
⟨*proof*⟩

**lemma** *suc-Union-closed-total'*:
  **assumes** $X \in \mathcal{C}$ **and** $Y \in \mathcal{C}$
    **and** $*$: $\bigwedge Z.\ Z \in \mathcal{C} \Longrightarrow Z \subseteq Y \Longrightarrow Z = Y \vee suc\ Z \subseteq Y$
  **shows** $X \subseteq Y \vee suc\ Y \subseteq X$
  ⟨*proof*⟩

**lemma** *suc-Union-closed-subsetD*:
  **assumes** $Y \subseteq X$ **and** $X \in \mathcal{C}$ **and** $Y \in \mathcal{C}$
  **shows** $X = Y \vee suc\ Y \subseteq X$
  ⟨*proof*⟩

The elements of $\mathcal{C}$ are totally ordered by the subset relation.

**lemma** *suc-Union-closed-total*:
  **assumes** $X \in \mathcal{C}$ **and** $Y \in \mathcal{C}$
  **shows** $X \subseteq Y \vee Y \subseteq X$
⟨*proof*⟩

Once we hit a fixed point w.r.t. *suc*, all other elements of $\mathcal{C}$ are subsets of this fixed point.

**lemma** *suc-Union-closed-suc*:
  **assumes** $X \in \mathcal{C}$ **and** $Y \in \mathcal{C}$ **and** $suc\ Y = Y$
  **shows** $X \subseteq Y$
  ⟨*proof*⟩

**lemma** *eq-suc-Union*:
  **assumes** $X \in \mathcal{C}$
  **shows** $suc\ X = X \longleftrightarrow X = \bigcup \mathcal{C}$
    (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *suc-in-carrier*:
 **assumes** $X \subseteq A$
 **shows** *suc* $X \subseteq A$
 ⟨*proof*⟩

**lemma** *suc-Union-closed-in-carrier*:
 **assumes** $X \in \mathcal{C}$
 **shows** $X \subseteq A$
 ⟨*proof*⟩

All elements of $\mathcal{C}$ are chains.

**lemma** *suc-Union-closed-chain*:
 **assumes** $X \in \mathcal{C}$
 **shows** *chain* $X$
 ⟨*proof*⟩

### 26.1.2 Hausdorff's Maximum Principle

There exists a maximal totally ordered subset of $A$. (Note that we do not require $A$ to be partially ordered.)

**theorem** *Hausdorff*: $\exists\, C.$ *maxchain* $C$
⟨*proof*⟩

Make notation $\mathcal{C}$ available again.

**no-notation** *suc-Union-closed* (‹$\mathcal{C}$›)

**lemma** *chain-extend*: *chain* $C \Longrightarrow z \in A \Longrightarrow \forall\, x{\in}C.\ x \sqsubseteq z \Longrightarrow$ *chain* $(\{z\} \cup C)$
 ⟨*proof*⟩

**lemma** *maxchain-imp-chain*: *maxchain* $C \Longrightarrow$ *chain* $C$
 ⟨*proof*⟩

**end**

Hide constant *pred-on.suc-Union-closed*, which was just needed for the proof of Hausforff's maximum principle.

**hide-const** *pred-on.suc-Union-closed*

**lemma** *chain-mono*:
 **assumes** $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow P\ x\ y \Longrightarrow Q\ x\ y$
  **and** *pred-on.chain* $A\ P\ C$
 **shows** *pred-on.chain* $A\ Q\ C$
 ⟨*proof*⟩

### 26.1.3 Results for the proper subset relation

**interpretation** *subset*: *pred-on* $A$ $(\subset)$ **for** $A$ ⟨*proof*⟩

**lemma** *subset-maxchain-max*:
  **assumes** *subset.maxchain* $A$ $C$
    **and** $X \in A$
    **and** $\bigcup C \subseteq X$
  **shows** $\bigcup C = X$
$\langle proof \rangle$

**lemma** *subset-chain-def*: $\bigwedge\mathcal{A}$. *subset.chain* $\mathcal{A}$ $\mathcal{C}$ = $(\mathcal{C} \subseteq \mathcal{A} \wedge (\forall X{\in}\mathcal{C}.\ \forall Y{\in}\mathcal{C}.\ X \subseteq Y \vee Y \subseteq X))$
  $\langle proof \rangle$

**lemma** *subset-chain-insert*:
  *subset.chain* $\mathcal{A}$ (*insert* $B$ $\mathcal{B}$) $\longleftrightarrow$ $B \in \mathcal{A} \wedge (\forall X{\in}\mathcal{B}.\ X \subseteq B \vee B \subseteq X) \wedge$ *subset.chain* $\mathcal{A}$ $\mathcal{B}$
  $\langle proof \rangle$

### 26.1.4 Zorn's lemma

If every chain has an upper bound, then there is a maximal set.

**theorem** *subset-Zorn*:
  **assumes** $\bigwedge C$. *subset.chain* $A$ $C \Longrightarrow \exists U{\in}A.\ \forall X{\in}C.\ X \subseteq U$
  **shows** $\exists M{\in}A.\ \forall X{\in}A.\ M \subseteq X \longrightarrow X = M$
$\langle proof \rangle$

Alternative version of Zorn's lemma for the subset relation.

**lemma** *subset-Zorn'*:
  **assumes** $\bigwedge C$. *subset.chain* $A$ $C \Longrightarrow \bigcup C \in A$
  **shows** $\exists M{\in}A.\ \forall X{\in}A.\ M \subseteq X \longrightarrow X = M$
$\langle proof \rangle$

## 26.2 Zorn's Lemma for Partial Orders

Relate old to new definitions.

**definition** *chain-subset* :: $'a$ *set set* $\Rightarrow$ *bool* (‹$chain_\subseteq$›)
  **where** $chain_\subseteq$ $C \longleftrightarrow (\forall A{\in}C.\ \forall B{\in}C.\ A \subseteq B \vee B \subseteq A)$

**definition** *chains* :: $'a$ *set set* $\Rightarrow$ $'a$ *set set set*
  **where** *chains* $A = \{C.\ C \subseteq A \wedge chain_\subseteq\ C\}$

**definition** *Chains* :: $('a \times 'a)$ *set* $\Rightarrow$ $'a$ *set set*
  **where** *Chains* $r = \{C.\ \forall a{\in}C.\ \forall b{\in}C.\ (a, b) \in r \vee (b, a) \in r\}$

**lemma** *chains-extend*: $c \in$ *chains* $S \Longrightarrow z \in S \Longrightarrow \forall x \in c.\ x \subseteq z \Longrightarrow \{z\} \cup c \in$ *chains* $S$
  **for** $z$ :: $'a$ *set*
  $\langle proof \rangle$

**lemma** *mono-Chains*: $r \subseteq s \Longrightarrow Chains\ r \subseteq Chains\ s$
⟨*proof*⟩

**lemma** *chain-subset-alt-def*: $chain_{\subseteq}\ C = subset.chain\ UNIV\ C$
⟨*proof*⟩

**lemma** *chains-alt-def*: $chains\ A = \{C.\ subset.chain\ A\ C\}$
⟨*proof*⟩

**lemma** *Chains-subset*: $Chains\ r \subseteq \{C.\ pred\text{-}on.chain\ UNIV\ (\lambda x\ y.\ (x,\ y) \in r)\ C\}$
⟨*proof*⟩

**lemma** *Chains-subset′*:
**assumes** *refl r*
**shows** $\{C.\ pred\text{-}on.chain\ UNIV\ (\lambda x\ y.\ (x,\ y) \in r)\ C\} \subseteq Chains\ r$
⟨*proof*⟩

**lemma** *Chains-alt-def*:
**assumes** *refl r*
**shows** $Chains\ r = \{C.\ pred\text{-}on.chain\ UNIV\ (\lambda x\ y.\ (x,\ y) \in r)\ C\}$
⟨*proof*⟩

**lemma** *Chains-relation-of*:
**assumes** $C \in Chains\ (relation\text{-}of\ P\ A)$ **shows** $C \subseteq A$
⟨*proof*⟩

**lemma** *pairwise-chain-Union*:
**assumes** *P*: $\bigwedge S.\ S \in \mathcal{C} \Longrightarrow pairwise\ R\ S$ **and** $chain_{\subseteq}\ \mathcal{C}$
**shows** $pairwise\ R\ (\bigcup \mathcal{C})$
⟨*proof*⟩

**lemma** *Zorn-Lemma*: $\forall\ C\in chains\ A.\ \bigcup C \in A \Longrightarrow \exists\ M\in A.\ \forall\ X\in A.\ M \subseteq X \longrightarrow X = M$
⟨*proof*⟩

**lemma** *Zorn-Lemma2*: $\forall\ C\in chains\ A.\ \exists\ U\in A.\ \forall\ X\in C.\ X \subseteq U \Longrightarrow \exists\ M\in A.\ \forall\ X\in A.\ M \subseteq X \longrightarrow X = M$
⟨*proof*⟩

## 26.3 Other variants of Zorn's Lemma

**lemma** *chainsD*: $c \in chains\ S \Longrightarrow x \in c \Longrightarrow y \in c \Longrightarrow x \subseteq y \vee y \subseteq x$
⟨*proof*⟩

**lemma** *chainsD2*: $c \in chains\ S \Longrightarrow c \subseteq S$
⟨*proof*⟩

**lemma** *Zorns-po-lemma*:
**assumes** *po*: *Partial-order r*

**and** *u*: $\bigwedge C.\ C \in \mathit{Chains}\ r \Longrightarrow \exists\, u{\in}\mathit{Field}\ r.\ \forall\, a{\in}C.\ (a,\ u) \in r$
**shows** $\exists\, m{\in}\mathit{Field}\ r.\ \forall\, a{\in}\mathit{Field}\ r.\ (m,\ a) \in r \longrightarrow a = m$
⟨*proof*⟩

**lemma** *predicate-Zorn*:
**assumes** *po*: *partial-order-on A* (*relation-of P A*)
**and** *ch*: $\bigwedge C.\ C \in \mathit{Chains}\ (\mathit{relation\text{-}of}\ P\ A) \Longrightarrow \exists\, u \in A.\ \forall\, a \in C.\ P\ a\ u$
**shows** $\exists\, m \in A.\ \forall\, a \in A.\ P\ m\ a \longrightarrow a = m$
⟨*proof*⟩

**lemma** *Union-in-chain*: $[\![\mathit{finite}\ \mathcal{B};\ \mathcal{B} \neq \{\};\ \mathit{subset.chain}\ \mathcal{A}\ \mathcal{B}]\!] \Longrightarrow \bigcup\mathcal{B} \in \mathcal{B}$
⟨*proof*⟩

**lemma** *Inter-in-chain*: $[\![\mathit{finite}\ \mathcal{B};\ \mathcal{B} \neq \{\};\ \mathit{subset.chain}\ \mathcal{A}\ \mathcal{B}]\!] \Longrightarrow \bigcap\mathcal{B} \in \mathcal{B}$
⟨*proof*⟩

**lemma** *finite-subset-Union-chain*:
**assumes** $\mathit{finite}\ A\ \ A \subseteq \bigcup\mathcal{B}\ \ \mathcal{B} \neq \{\}$ **and** *sub*: $\mathit{subset.chain}\ \mathcal{A}\ \mathcal{B}$
**obtains** $B$ **where** $B \in \mathcal{B}\ \ A \subseteq B$
⟨*proof*⟩

**lemma** *subset-Zorn-nonempty*:
**assumes** $\mathcal{A} \neq \{\}$ **and** *ch*: $\bigwedge\mathcal{C}.\ [\![\mathcal{C}{\neq}\{\};\ \mathit{subset.chain}\ \mathcal{A}\ \mathcal{C}]\!] \Longrightarrow \bigcup\mathcal{C} \in \mathcal{A}$
**shows** $\exists\, M{\in}\mathcal{A}.\ \forall\, X{\in}\mathcal{A}.\ M \subseteq X \longrightarrow X = M$
⟨*proof*⟩

## 26.4 The Well Ordering Theorem

**definition** *init-seg-of* :: $(('a \times 'a)\ \mathit{set} \times ('a \times 'a)\ \mathit{set})\ \mathit{set}$
**where** $\mathit{init\text{-}seg\text{-}of} = \{(r,\ s).\ r \subseteq s \land (\forall\, a\ b\ c.\ (a,\ b) \in s \land (b,\ c) \in r \longrightarrow (a,\ b) \in r)\}$

**abbreviation** *initial-segment-of-syntax* :: $('a \times 'a)\ \mathit{set} \Rightarrow ('a \times 'a)\ \mathit{set} \Rightarrow \mathit{bool}$
(**infix** ‹*initial'-segment'-of*› *55*)
**where** $r\ \mathit{initial\text{-}segment\text{-}of}\ s \equiv (r,\ s) \in \mathit{init\text{-}seg\text{-}of}$

**lemma** *refl-on-init-seg-of* [*simp*]: *r initial-segment-of r*
⟨*proof*⟩

**lemma** *trans-init-seg-of*:
$r\ \mathit{initial\text{-}segment\text{-}of}\ s \Longrightarrow s\ \mathit{initial\text{-}segment\text{-}of}\ t \Longrightarrow r\ \mathit{initial\text{-}segment\text{-}of}\ t$
⟨*proof*⟩

**lemma** *antisym-init-seg-of*: $r\ \mathit{initial\text{-}segment\text{-}of}\ s \Longrightarrow s\ \mathit{initial\text{-}segment\text{-}of}\ r \Longrightarrow r = s$
⟨*proof*⟩

**lemma** *Chains-init-seg-of-Union*: $R \in \mathit{Chains}\ \mathit{init\text{-}seg\text{-}of} \Longrightarrow r{\in}R \Longrightarrow r\ \mathit{initial\text{-}segment\text{-}of}\ \bigcup R$

⟨*proof*⟩

**lemma** *chain-subset-trans-Union*:
**assumes** $chain_\subseteq\ R\ \forall r \in R.\ trans\ r$
**shows** $trans\ (\bigcup R)$
⟨*proof*⟩

**lemma** *chain-subset-antisym-Union*:
**assumes** $chain_\subseteq\ R\ \forall r \in R.\ antisym\ r$
**shows** $antisym\ (\bigcup R)$
⟨*proof*⟩

**lemma** *chain-subset-Total-Union*:
**assumes** $chain_\subseteq\ R$ **and** $\forall r \in R.\ Total\ r$
**shows** $Total\ (\bigcup R)$
⟨*proof*⟩

**lemma** *wf-Union-wf-init-segs*:
**assumes** $R \in Chains\ init\text{-}seg\text{-}of$
**and** $\forall r \in R.\ wf\ r$
**shows** $wf\ (\bigcup R)$
⟨*proof*⟩

**lemma** *initial-segment-of-Diff*: $p\ initial\text{-}segment\text{-}of\ q \Longrightarrow p - s\ initial\text{-}segment\text{-}of\ q - s$
⟨*proof*⟩

**lemma** *Chains-inits-DiffI*: $R \in Chains\ init\text{-}seg\text{-}of \Longrightarrow \{r - s\ |r.\ r \in R\} \in Chains\ init\text{-}seg\text{-}of$
⟨*proof*⟩

**theorem** *well-ordering*: $\exists r::'a\ rel.\ Well\text{-}order\ r \wedge Field\ r = UNIV$
⟨*proof*⟩

**corollary** *well-order-on*: $\exists r::'a\ rel.\ well\text{-}order\text{-}on\ A\ r$
⟨*proof*⟩

**lemma** *dependent-wf-choice*:
**fixes** $P :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
**assumes** $wf\ R$
**and** *adm*: $\bigwedge f\ g\ x\ r.\ (\bigwedge z.\ (z, x) \in R \Longrightarrow f\ z = g\ z) \Longrightarrow P\ f\ x\ r = P\ g\ x\ r$
**and** *P*: $\bigwedge x\ f.\ (\bigwedge y.\ (y, x) \in R \Longrightarrow P\ f\ y\ (f\ y)) \Longrightarrow \exists r.\ P\ f\ x\ r$
**shows** $\exists f.\ \forall x.\ P\ f\ x\ (f\ x)$
⟨*proof*⟩

**lemma** (**in** *wellorder*) *dependent-wellorder-choice*:
**assumes** $\bigwedge r\ f\ g\ x.\ (\bigwedge y.\ y < x \Longrightarrow f\ y = g\ y) \Longrightarrow P\ f\ x\ r = P\ g\ x\ r$
**and** *P*: $\bigwedge x\ f.\ (\bigwedge y.\ y < x \Longrightarrow P\ f\ y\ (f\ y)) \Longrightarrow \exists r.\ P\ f\ x\ r$
**shows** $\exists f.\ \forall x.\ P\ f\ x\ (f\ x)$

⟨*proof*⟩

**end**

# 27 Well-Order Relations as Needed by Bounded Natural Functors

**theory** *BNF-Wellorder-Relation*
**imports** *Order-Relation*
**begin**

In this section, we develop basic concepts and results pertaining to well-order relations. Note that we consider well-order relations as *non-strict relations*, i.e., as containing the diagonals of their fields.

**locale** *wo-rel* =
**fixes** *r* :: *'a rel*
**assumes** *WELL*: *Well-order r*
**begin**

The following context encompasses all this section. In other words, for the whole section, we consider a fixed well-order relation *r*.

**abbreviation** *under* **where** *under* ≡ *Order-Relation.under r*
**abbreviation** *underS* **where** *underS* ≡ *Order-Relation.underS r*
**abbreviation** *Under* **where** *Under* ≡ *Order-Relation.Under r*
**abbreviation** *UnderS* **where** *UnderS* ≡ *Order-Relation.UnderS r*
**abbreviation** *above* **where** *above* ≡ *Order-Relation.above r*
**abbreviation** *aboveS* **where** *aboveS* ≡ *Order-Relation.aboveS r*
**abbreviation** *Above* **where** *Above* ≡ *Order-Relation.Above r*
**abbreviation** *AboveS* **where** *AboveS* ≡ *Order-Relation.AboveS r*
**abbreviation** *ofilter* **where** *ofilter* ≡ *Order-Relation.ofilter r*
**lemmas** *ofilter-def* = *Order-Relation.ofilter-def*[*of r*]

## 27.1 Auxiliaries

**lemma** *REFL*: *Refl r*
⟨*proof*⟩

**lemma** *TRANS*: *trans r*
⟨*proof*⟩

**lemma** *ANTISYM*: *antisym r*
⟨*proof*⟩

**lemma** *TOTAL*: *Total r*
⟨*proof*⟩

**lemma** *TOTALS*: $\forall a \in Field\ r.\ \forall b \in Field\ r.\ (a,b) \in r \lor (b,a) \in r$
⟨*proof*⟩

**lemma** *LIN*: *Linear-order r*
⟨*proof*⟩

**lemma** *WF*: *wf* $(r - Id)$
⟨*proof*⟩

**lemma** *cases-Total*:
$\bigwedge$ *phi a b*. $[\![\{a,b\} <= Field\ r;\ ((a,b) \in r \Longrightarrow phi\ a\ b);\ ((b,a) \in r \Longrightarrow phi\ a\ b)]\!]$
$\Longrightarrow phi\ a\ b$
⟨*proof*⟩

**lemma** *cases-Total3*:
$\bigwedge$ *phi a b*. $[\![\{a,b\} \leq Field\ r;\ ((a,b) \in r - Id \vee (b,a) \in r - Id \Longrightarrow phi\ a\ b);$
$(a = b \Longrightarrow phi\ a\ b)]\!] \Longrightarrow phi\ a\ b$
⟨*proof*⟩

## 27.2 Well-founded induction and recursion adapted to non-strict well-order relations

Here we provide induction and recursion principles specific to *non-strict* well-order relations. Although minor variations of those for well-founded relations, they will be useful for doing away with the tediousness of having to take out the diagonal each time in order to switch to a well-founded relation.

**lemma** *well-order-induct*:
**assumes** *IND*: $\bigwedge x.\ \forall y.\ y \neq x \wedge (y,\ x) \in r \longrightarrow P\ y \Longrightarrow P\ x$
**shows** $P\ a$
⟨*proof*⟩

**definition**
*worec* :: $(('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
**where**
*worec* $F \equiv wfrec\ (r - Id)\ F$

**definition**
*adm-wo* :: $(('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow bool$
**where**
*adm-wo* $H \equiv \forall f\ g\ x.\ (\forall y \in underS\ x.\ f\ y = g\ y) \longrightarrow H\ f\ x = H\ g\ x$

**lemma** *worec-fixpoint*:
**assumes** *ADM*: *adm-wo H*
**shows** *worec* $H = H$ (*worec* $H$)
⟨*proof*⟩

## 27.3 The notions of maximum, minimum, supremum, successor and order filter

We define the successor *of a set*, and not of an element (the latter is of course a particular case). Also, we define the maximum *of two elements*, *max2*, and the minimum *of a set*, *minim* – we chose these variants since we consider them the most useful for well-orders. The minimum is defined in terms of the auxiliary relational operator *isMinim*. Then, supremum and successor are defined in terms of minimum as expected. The minimum is only meaningful for non-empty sets, and the successor is only meaningful for sets for which strict upper bounds exist. Order filters for well-orders are also known as "initial segments".

**definition** *max2* :: $'a \Rightarrow 'a \Rightarrow 'a$
  **where** *max2 a b* $\equiv$ *if* $(a,b) \in r$ *then b else a*

**definition** *isMinim* :: $'a$ *set* $\Rightarrow 'a \Rightarrow$ *bool*
  **where** *isMinim A b* $\equiv b \in A \wedge (\forall a \in A.\ (b,a) \in r)$

**definition** *minim* :: $'a$ *set* $\Rightarrow 'a$
  **where** *minim A* $\equiv$ *THE b. isMinim A b*

**definition** *supr* :: $'a$ *set* $\Rightarrow 'a$
  **where** *supr A* $\equiv$ *minim* (*Above A*)

**definition** *suc* :: $'a$ *set* $\Rightarrow 'a$
  **where** *suc A* $\equiv$ *minim* (*AboveS A*)

### 27.3.1 Properties of max2

**lemma** *max2-greater-among*:
  **assumes** $a \in$ *Field r* **and** $b \in$ *Field r*
  **shows** $(a,$ *max2 a b*$) \in r \wedge (b,$ *max2 a b*$) \in r \wedge$ *max2 a b* $\in \{a,b\}$
⟨*proof*⟩

**lemma** *max2-greater*:
  **assumes** $a \in$ *Field r* **and** $b \in$ *Field r*
  **shows** $(a,$ *max2 a b*$) \in r \wedge (b,$ *max2 a b*$) \in r$
  ⟨*proof*⟩

**lemma** *max2-among*:
  **assumes** $a \in$ *Field r* **and** $b \in$ *Field r*
  **shows** *max2 a b* $\in \{a, b\}$
  ⟨*proof*⟩

**lemma** *max2-equals1*:
  **assumes** $a \in$ *Field r* **and** $b \in$ *Field r*
  **shows** (*max2 a b* $= a) = ((b,a) \in r)$
  ⟨*proof*⟩

**lemma** *max2-equals2*:
**assumes** $a \in$ *Field* $r$ **and** $b \in$ *Field* $r$
**shows** $($*max2* $a$ $b = b) = ((a,b) \in r)$
⟨*proof*⟩

**lemma** *in-notinI*:
**assumes** $(j,i) \notin r \lor j = i$ **and** $i \in$ *Field* $r$ **and** $j \in$ *Field* $r$
**shows** $(i,j) \in r$ ⟨*proof*⟩

### 27.3.2 Existence and uniqueness for isMinim and well-definedness of minim

**lemma** *isMinim-unique*:
**assumes** *isMinim* $B$ $a$ *isMinim* $B$ $a'$
**shows** $a = a'$
⟨*proof*⟩

**lemma** *Well-order-isMinim-exists*:
**assumes** *SUB*: $B \leq$ *Field* $r$ **and** *NE*: $B \neq \{\}$
**shows** $\exists\, b.$ *isMinim* $B$ $b$
⟨*proof*⟩

**lemma** *minim-isMinim*:
**assumes** *SUB*: $B \leq$ *Field* $r$ **and** *NE*: $B \neq \{\}$
**shows** *isMinim* $B$ $($*minim* $B)$
⟨*proof*⟩

### 27.3.3 Properties of minim

**lemma** *minim-in*:
**assumes** $B \leq$ *Field* $r$ **and** $B \neq \{\}$
**shows** *minim* $B \in B$
⟨*proof*⟩

**lemma** *minim-inField*:
**assumes** $B \leq$ *Field* $r$ **and** $B \neq \{\}$
**shows** *minim* $B \in$ *Field* $r$
⟨*proof*⟩

**lemma** *minim-least*:
**assumes** *SUB*: $B \leq$ *Field* $r$ **and** *IN*: $b \in B$
**shows** $($*minim* $B,\ b) \in r$
⟨*proof*⟩

**lemma** *equals-minim*:
**assumes** *SUB*: $B \leq$ *Field* $r$ **and** *IN*: $a \in B$ **and**
*LEAST*: $\bigwedge b.\ b \in B \Longrightarrow (a,b) \in r$
**shows** $a =$ *minim* $B$
⟨*proof*⟩

### 27.3.4 Properties of successor

**lemma** *suc-AboveS*:
**assumes** *SUB*: $B \leq$ *Field r* **and** *ABOVES*: *AboveS B* $\neq \{\}$
**shows** *suc B* $\in$ *AboveS B*
⟨*proof*⟩

**lemma** *suc-greater*:
**assumes** *SUB*: $B \leq$ *Field r* **and** *ABOVES*: *AboveS B* $\neq \{\}$ **and** *IN*: $b \in B$
**shows** *suc B* $\neq b \wedge (b,$*suc B*$) \in r$
⟨*proof*⟩

**lemma** *suc-least-AboveS*:
**assumes** *ABOVES*: $a \in$ *AboveS B*
**shows** (*suc B*,$a$) $\in r$
⟨*proof*⟩

**lemma** *suc-inField*:
**assumes** $B \leq$ *Field r* **and** *AboveS B* $\neq \{\}$
**shows** *suc B* $\in$ *Field r*
⟨*proof*⟩

**lemma** *equals-suc-AboveS*:
**assumes** $B \leq$ *Field r* **and** $a \in$ *AboveS B* **and** $\bigwedge a'.\ a' \in$ *AboveS B* $\Longrightarrow (a,a')$ $\in r$
**shows** $a =$ *suc B*
⟨*proof*⟩

**lemma** *suc-underS*:
**assumes** *IN*: $a \in$ *Field r*
**shows** $a =$ *suc* (*underS a*)
⟨*proof*⟩

### 27.3.5 Properties of order filters

**lemma** *under-ofilter*: *ofilter* (*under a*)
⟨*proof*⟩

**lemma** *underS-ofilter*: *ofilter* (*underS a*)
⟨*proof*⟩

**lemma** *Field-ofilter*:
*ofilter* (*Field r*)
⟨*proof*⟩

**lemma** *ofilter-underS-Field*:
*ofilter A* = (($\exists\, a \in$ *Field r*. $A =$ *underS a*) $\vee$ ($A =$ *Field r*))
⟨*proof*⟩

**lemma** *ofilter-UNION*:

$(\bigwedge i.\ i \in I \Longrightarrow \textit{ofilter}(A\ i)) \Longrightarrow \textit{ofilter}\ (\bigcup i \in I.\ A\ i)$
⟨*proof*⟩

**lemma** *ofilter-under-UNION*:
**assumes** *ofilter* $A$
**shows** $A = (\bigcup a \in A.\ \textit{under}\ a)$
⟨*proof*⟩

### 27.3.6 Other properties

**lemma** *ofilter-linord*:
**assumes** *OF1*: *ofilter* $A$ **and** *OF2*: *ofilter* $B$
**shows** $A \leq B \vee B \leq A$
⟨*proof*⟩

**lemma** *ofilter-AboveS-Field*:
**assumes** *ofilter* $A$
**shows** $A \cup (\textit{AboveS}\ A) = \textit{Field}\ r$
⟨*proof*⟩

**lemma** *suc-ofilter-in*:
**assumes** *OF*: *ofilter* $A$ **and** *ABOVE-NE*: $\textit{AboveS}\ A \neq \{\}$ **and**
*REL*: $(b,\textit{suc}\ A) \in r$ **and** *DIFF*: $b \neq \textit{suc}\ A$
**shows** $b \in A$
⟨*proof*⟩

**end**

**end**

# 28 Well-Order Embeddings as Needed by Bounded Natural Functors

**theory** *BNF-Wellorder-Embedding*
**imports** *Hilbert-Choice BNF-Wellorder-Relation*
**begin**

In this section, we introduce well-order *embeddings* and *isomorphisms* and prove their basic properties. The notion of embedding is considered from the point of view of the theory of ordinals, and therefore requires the source to be injected as an *initial segment* (i.e., *order filter*) of the target. A main result of this section is the existence of embeddings (in one direction or another) between any two well-orders, having as a consequence the fact that, given any two sets on any two types, one is smaller than (i.e., can be injected into) the other.

## 28.1 Auxiliaries

**lemma** *UNION-inj-on-ofilter*:
  **assumes** *WELL*: *Well-order r* **and**
    *OF*: $\bigwedge i.\ i \in I \Longrightarrow$ *wo-rel.ofilter r* $(A\ i)$ **and**
    *INJ*: $\bigwedge i.\ i \in I \Longrightarrow$ *inj-on f* $(A\ i)$
  **shows** *inj-on f* $(\bigcup i \in I.\ A\ i)$
⟨*proof*⟩

**lemma** *under-underS-bij-betw*:
  **assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
    *IN*: $a \in$ *Field r* **and** *IN'*: $f\ a \in$ *Field r'* **and**
    *BIJ*: *bij-betw f* (*underS r a*) (*underS r'* $(f\ a)$)
  **shows** *bij-betw f* (*under r a*) (*under r'* $(f\ a)$)
⟨*proof*⟩

## 28.2 (Well-order) embeddings, strict embeddings, isomorphisms and order-compatible functions

Standardly, a function is an embedding of a well-order in another if it injectively and order-compatibly maps the former into an order filter of the latter. Here we opt for a more succinct definition (operator *embed*), asking that, for any element in the source, the function should be a bijection between the set of strict lower bounds of that element and the set of strict lower bounds of its image. (Later we prove equivalence with the standard definition – lemma *embed-iff-compat-inj-on-ofilter*.) A *strict embedding* (operator *embedS*) is a non-bijective embedding and an isomorphism (operator *iso*) is a bijective embedding.

**definition** *embed* :: $'a$ *rel* $\Rightarrow$ $'a'$ *rel* $\Rightarrow$ $('a \Rightarrow 'a') \Rightarrow$ *bool*
  **where**
    *embed r r' f* $\equiv \forall a \in$ *Field r*. *bij-betw f* (*under r a*) (*under r'* $(f\ a)$)

**lemmas** *embed-defs* = *embed-def embed-def*[*abs-def*]

Strict embeddings:

**definition** *embedS* :: $'a$ *rel* $\Rightarrow$ $'a'$ *rel* $\Rightarrow$ $('a \Rightarrow 'a') \Rightarrow$ *bool*
  **where**
    *embedS r r' f* $\equiv$ *embed r r' f* $\wedge \neg$ *bij-betw f* (*Field r*) (*Field r'*)

**lemmas** *embedS-defs* = *embedS-def embedS-def*[*abs-def*]

**definition** *iso* :: $'a$ *rel* $\Rightarrow$ $'a'$ *rel* $\Rightarrow$ $('a \Rightarrow 'a') \Rightarrow$ *bool*
  **where**
    *iso r r' f* $\equiv$ *embed r r' f* $\wedge$ *bij-betw f* (*Field r*) (*Field r'*)

**lemmas** *iso-defs* = *iso-def iso-def*[*abs-def*]

**definition** *compat* :: $'a$ *rel* $\Rightarrow$ $'a'$ *rel* $\Rightarrow$ $('a \Rightarrow 'a') \Rightarrow$ *bool*

**where**
*compat* $r\ r'\ f \equiv \forall\ a\ b.\ (a,b) \in r \longrightarrow (f\ a,\ f\ b) \in r'$

**lemma** *compat-wf*:
**assumes** *CMP*: *compat* $r\ r'\ f$ **and** *WF*: *wf* $r'$
**shows** *wf* $r$
⟨*proof*⟩

**lemma** *id-embed*: *embed* $r\ r$ *id*
⟨*proof*⟩

**lemma** *id-iso*: *iso* $r\ r$ *id*
⟨*proof*⟩

**lemma** *embed-compat*:
**assumes** *EMB*: *embed* $r\ r'\ f$
**shows** *compat* $r\ r'\ f$
⟨*proof*⟩

**lemma** *embed-in-Field*:
**assumes** *EMB*: *embed* $r\ r'\ f$ **and** *IN*: $a \in$ *Field* $r$
**shows** $f\ a \in$ *Field* $r'$
⟨*proof*⟩

**lemma** *comp-embed*:
**assumes** *EMB*: *embed* $r\ r'\ f$ **and** *EMB'*: *embed* $r'\ r''\ f'$
**shows** *embed* $r\ r''\ (f' \circ f)$
⟨*proof*⟩

**lemma** *comp-iso*:
**assumes** *EMB*: *iso* $r\ r'\ f$ **and** *EMB'*: *iso* $r'\ r''\ f'$
**shows** *iso* $r\ r''\ (f' \circ f)$
⟨*proof*⟩

That *embedS* is also preserved by function composition shall be proved only later.

**lemma** *embed-Field*: *embed* $r\ r'\ f \Longrightarrow f`($*Field* $r) \leq$ *Field* $r'$
⟨*proof*⟩

**lemma** *embed-preserves-ofilter*:
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$ **and**
*EMB*: *embed* $r\ r'\ f$ **and** *OF*: *wo-rel.ofilter* $r\ A$
**shows** *wo-rel.ofilter* $r'\ (f`A)$
⟨*proof*⟩

**lemma** *embed-Field-ofilter*:
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$ **and**
*EMB*: *embed* $r\ r'\ f$
**shows** *wo-rel.ofilter* $r'\ (f`($*Field* $r))$

⟨*proof*⟩

**lemma** *embed-inj-on*:
**assumes** *WELL*: *Well-order r* **and** *EMB*: *embed r r′ f*
**shows** *inj-on f (Field r)*
⟨*proof*⟩

**lemma** *embed-underS*:
**assumes** *WELL*: *Well-order r* **and**
*EMB*: *embed r r′ f* **and** *IN*: $a \in$ *Field r*
**shows** *bij-betw f (underS r a) (underS r′ (f a))*
⟨*proof*⟩

**lemma** *embed-iff-compat-inj-on-ofilter*:
**assumes** *WELL*: *Well-order r* **and** *WELL′*: *Well-order r′*
**shows** *embed r r′ f* = (*compat r r′ f* $\wedge$ *inj-on f (Field r)* $\wedge$ *wo-rel.ofilter r′ (f‘(Field r))*)
⟨*proof*⟩

**lemma** *inv-into-ofilter-embed*:
**assumes** *WELL*: *Well-order r* **and** *OF*: *wo-rel.ofilter r A* **and**
*BIJ*: $\forall$ *b* $\in$ *A. bij-betw f (under r b) (under r′ (f b))* **and**
*IMAGE*: *f ‘ A = Field r′*
**shows** *embed r′ r (inv-into A f)*
⟨*proof*⟩

**lemma** *inv-into-underS-embed*:
**assumes** *WELL*: *Well-order r* **and**
*BIJ*: $\forall$ *b* $\in$ *underS r a. bij-betw f (under r b) (under r′ (f b))* **and**
*IN*: $a \in$ *Field r* **and**
*IMAGE*: *f ‘ (underS r a) = Field r′*
**shows** *embed r′ r (inv-into (underS r a) f)*
⟨*proof*⟩

**lemma** *inv-into-Field-embed*:
**assumes** *WELL*: *Well-order r* **and** *EMB*: *embed r r′ f* **and**
*IMAGE*: *Field r′* $\leq$ *f ‘ (Field r)*
**shows** *embed r′ r (inv-into (Field r) f)*
⟨*proof*⟩

**lemma** *inv-into-Field-embed-bij-betw*:
**assumes** *EMB*: *embed r r′ f* **and** *BIJ*: *bij-betw f (Field r) (Field r′)*
**shows** *embed r′ r (inv-into (Field r) f)*
⟨*proof*⟩

## 28.3 Given any two well-orders, one can be embedded in the other

Here is an overview of the proof of of this fact, stated in theorem *wellorders-totally-ordered*:

Fix the well-orders $r$::$'a$ *rel* and $r'$::$'a'$ *rel.* Attempt to define an embedding $f$::$'a \Rightarrow 'a'$ from $r$ to $r'$ in the natural way by well-order recursion ("hoping" that *Field* $r$ turns out to be smaller than *Field* $r'$), but also record, at the recursive step, in a function $g$::$'a \Rightarrow$ *bool*, the extra information of whether *Field* $r'$ gets exhausted or not.

If *Field* $r'$ does not get exhausted, then *Field* $r$ is indeed smaller and $f$ is the desired embedding from $r$ to $r'$ (lemma *wellorders-totally-ordered-aux*).

Otherwise, it means that *Field* $r'$ is the smaller one, and the inverse of (the "good" segment of) $f$ is the desired embedding from $r'$ to $r$ (lemma *wellorders-totally-ordered-aux2*).

**lemma** *wellorders-totally-ordered-aux*:
**fixes** $r$ ::$'a$ *rel* **and** $r'$::$'a'$ *rel* **and**
$f$ :: $'a \Rightarrow 'a'$ **and** $a$::$'a$
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$ **and** *IN*: $a \in$ *Field* $r$ **and**
*IH*: $\forall\, b \in$ *underS* $r$ $a$. *bij-betw* $f$ (*under* $r$ $b$) (*under* $r'$ ($f$ $b$)) **and**
*NOT*: $f$ ‘ (*underS* $r$ $a$) $\neq$ *Field* $r'$ **and** *SUC*: $f$ $a$ = *wo-rel.suc* $r'$ ($f$‘(*underS* $r$ $a$))
**shows** *bij-betw* $f$ (*under* $r$ $a$) (*under* $r'$ ($f$ $a$))
⟨*proof*⟩

**lemma** *wellorders-totally-ordered-aux2*:
**fixes** $r$ ::$'a$ *rel* **and** $r'$::$'a'$ *rel* **and**
$f$ :: $'a \Rightarrow 'a'$ **and** $g$ :: $'a \Rightarrow$ *bool* **and** $a$::$'a$
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$ **and**
*MAIN1*:
$\bigwedge a$. (*False* $\notin$ $g$‘(*underS* $r$ $a$) $\wedge$ $f$‘(*underS* $r$ $a$) $\neq$ *Field* $r'$
$\longrightarrow$ $f$ $a$ = *wo-rel.suc* $r'$ ($f$‘(*underS* $r$ $a$)) $\wedge$ $g$ $a$ = *True*)
$\wedge$
($\neg$(*False* $\notin$ ($g$‘(*underS* $r$ $a$)) $\wedge$ $f$‘(*underS* $r$ $a$) $\neq$ *Field* $r'$)
$\longrightarrow$ $g$ $a$ = *False*) **and**
*MAIN2*: $\bigwedge a$. $a \in$ *Field* $r$ $\wedge$ *False* $\notin$ $g$‘(*under* $r$ $a$) $\longrightarrow$
*bij-betw* $f$ (*under* $r$ $a$) (*under* $r'$ ($f$ $a$)) **and**
*Case*: $a \in$ *Field* $r$ $\wedge$ *False* $\in$ $g$‘(*under* $r$ $a$)
**shows** $\exists f'$. *embed* $r'$ $r$ $f'$
⟨*proof*⟩

**theorem** *wellorders-totally-ordered*:
**fixes** $r$ ::$'a$ *rel* **and** $r'$::$'a'$ *rel*
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$
**shows** ($\exists f$. *embed* $r$ $r'$ $f$) $\vee$ ($\exists f'$. *embed* $r'$ $r$ $f'$)
⟨*proof*⟩

## 28.4 Uniqueness of embeddings

Here we show a fact complementary to the one from the previous subsection – namely, that between any two well-orders there is *at most* one embedding, and is the one definable by the expected well-order recursive equation. As a consequence, any two embeddings of opposite directions are mutually inverse.

**lemma** *embed-determined*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r r' f* **and** *IN*: $a \in$ *Field r*
**shows** $f\ a = $ *wo-rel.suc* $r'$ ($f$'(*underS r a*))
⟨*proof*⟩

**lemma** *embed-unique*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMBf*: *embed r r' f* **and** *EMBg*: *embed r r' g*
**shows** $a \in$ *Field r* $\longrightarrow f\ a = g\ a$
⟨*proof*⟩

**lemma** *embed-bothWays-inverse*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r r' f* **and** *EMB'*: *embed r' r f'*
**shows** ($\forall a \in$ *Field r*. $f'(f\ a) = a$) $\wedge$ ($\forall a' \in$ *Field r'*. $f(f'\ a') = a'$)
⟨*proof*⟩

**lemma** *embed-bothWays-bij-betw*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r r' f* **and** *EMB'*: *embed r' r g*
**shows** *bij-betw f* (*Field r*) (*Field r'*)
⟨*proof*⟩

**lemma** *embed-bothWays-iso*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r r' f* **and** *EMB'*: *embed r' r g*
**shows** *iso r r' f*
⟨*proof*⟩

## 28.5 More properties of embeddings, strict embeddings and isomorphisms

**lemma** *embed-bothWays-Field-bij-betw*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r r' f* **and** *EMB'*: *embed r' r f'*
**shows** *bij-betw f* (*Field r*) (*Field r'*)
⟨*proof*⟩

**lemma** *embedS-comp-embed*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**and** *EMB*: *embedS r r' f* **and** *EMB'*: *embed r' r'' f'*
**shows** *embedS r r''* ($f' \circ f$)
⟨*proof*⟩

**lemma** *embed-comp-embedS*:

**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$
**and** *EMB*: *embed* $r$ $r'$ $f$ **and** *EMB'*: *embedS* $r'$ $r''$ $f'$
**shows** *embedS* $r$ $r''$ $(f' \circ f)$
⟨*proof*⟩

**lemma** *embed-comp-iso*:
**assumes** *EMB*: *embed* $r$ $r'$ $f$ **and** *EMB'*: *iso* $r'$ $r''$ $f'$
**shows** *embed* $r$ $r''$ $(f' \circ f)$ ⟨*proof*⟩

**lemma** *iso-comp-embed*:
**assumes** *EMB*: *iso* $r$ $r'$ $f$ **and** *EMB'*: *embed* $r'$ $r''$ $f'$
**shows** *embed* $r$ $r''$ $(f' \circ f)$
⟨*proof*⟩

**lemma** *embedS-comp-iso*:
**assumes** *EMB*: *embedS* $r$ $r'$ $f$ **and** *EMB'*: *iso* $r'$ $r''$ $f'$
**shows** *embedS* $r$ $r''$ $(f' \circ f)$
⟨*proof*⟩

**lemma** *iso-comp-embedS*:
**assumes** *WELL*: *Well-order* $r$ **and** *WELL'*: *Well-order* $r'$
**and** *EMB*: *iso* $r$ $r'$ $f$ **and** *EMB'*: *embedS* $r'$ $r''$ $f'$
**shows** *embedS* $r$ $r''$ $(f' \circ f)$
⟨*proof*⟩

**lemma** *embedS-Field*:
**assumes** *WELL*: *Well-order* $r$ **and** *EMB*: *embedS* $r$ $r'$ $f$
**shows** $f$ ‘ (*Field* $r$) $<$ *Field* $r'$
⟨*proof*⟩

**lemma** *embedS-iff*:
**assumes** *WELL*: *Well-order* $r$ **and** *ISO*: *embed* $r$ $r'$ $f$
**shows** *embedS* $r$ $r'$ $f$ = ($f$ ‘ (*Field* $r$) $<$ *Field* $r'$)
⟨*proof*⟩

**lemma** *iso-Field*: *iso* $r$ $r'$ $f \Longrightarrow f$ ‘ (*Field* $r$) = *Field* $r'$
⟨*proof*⟩

**lemma** *iso-iff*:
**assumes** *Well-order* $r$
**shows** *iso* $r$ $r'$ $f$ = (*embed* $r$ $r'$ $f$ $\wedge$ $f$ ‘ (*Field* $r$) = *Field* $r'$)
⟨*proof*⟩

**lemma** *iso-iff2*: *iso* $r$ $r'$ $f \longleftrightarrow$
*bij-betw* $f$ (*Field* $r$) (*Field* $r'$) $\wedge$
($\forall a \in$ *Field* $r$. $\forall b \in$ *Field* $r$. $(a, b) \in r \longleftrightarrow (f\ a, f\ b) \in r'$)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *iso-iff3*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** *iso r r' f* = (*bij-betw f* (*Field r*) (*Field r'*) ∧ *compat r r' f*)
⟨*proof*⟩

**lemma** *iso-imp-inj-on*:
**assumes** *iso r r' f* **shows** *inj-on f* (*Field r*)
⟨*proof*⟩

**lemma** *iso-backward-Field*:
**assumes** $x \in$ *Field r' iso r r' f*
**shows** *inv-into* (*Field r*) *f x* ∈ *Field r*
⟨*proof*⟩

**lemma** *iso-backward*:
**assumes** $(x,y) \in r'$ **and** *iso*: *iso r r' f*
**shows** (*inv-into* (*Field r*) *f x*, *inv-into* (*Field r*) *f y*) ∈ *r*
⟨*proof*⟩

**lemma** *iso-forward*:
**assumes** $(x,y) \in r$ *iso r r' f* **shows** (*f x*,*f y*) ∈ *r'*
⟨*proof*⟩

**lemma** *iso-trans*:
**assumes** *trans r* **and** *iso*: *iso r r' f* **shows** *trans r'*
⟨*proof*⟩

**lemma** *iso-Total*:
**assumes** *Total r* **and** *iso*: *iso r r' f* **shows** *Total r'*
⟨*proof*⟩

**lemma** *iso-wf*:
**assumes** *wf r* **and** *iso*: *iso r r' f* **shows** *wf r'*
⟨*proof*⟩

**end**

# 29 Constructions on Wellorders as Needed by Bounded Natural Functors

**theory** *BNF-Wellorder-Constructions*
**imports** *BNF-Wellorder-Embedding*
**begin**

In this section, we study basic constructions on well-orders, such as restriction to a set/order filter, copy via direct images, ordinal-like sum of disjoint well-orders, and bounded square. We also define between well-orders the relations *ordLeq*, of being embedded (abbreviated $\leq o$), *ordLess*, of being strictly embedded (abbreviated $<o$), and *ordIso*, of being isomorphic (abbreviated $=o$). We study the connections between these relations, order filters, and the aforementioned constructions. A main result of this section is that $<o$ is well-founded.

## 29.1 Restriction to a set

**abbreviation** $Restr :: 'a\ rel \Rightarrow 'a\ set \Rightarrow 'a\ rel$
**where** $Restr\ r\ A \equiv r\ Int\ (A \times A)$

**lemma** *Restr-subset*:
$A \leq B \Longrightarrow Restr\ (Restr\ r\ B)\ A = Restr\ r\ A$
⟨*proof*⟩

**lemma** *Restr-Field*: $Restr\ r\ (Field\ r) = r$
⟨*proof*⟩

**lemma** *Refl-Restr*: $Refl\ r \Longrightarrow Refl(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *linear-order-on-Restr*:
$linear\text{-}order\text{-}on\ A\ r \Longrightarrow linear\text{-}order\text{-}on\ (A \cap above\ r\ x)\ (Restr\ r\ (above\ r\ x))$
⟨*proof*⟩

**lemma** *antisym-Restr*:
$antisym\ r \Longrightarrow antisym(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *Total-Restr*:
$Total\ r \Longrightarrow Total(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *total-on-imp-Total-Restr*: $total\text{-}on\ A\ r \Longrightarrow Total\ (Restr\ r\ A)$
⟨*proof*⟩

**lemma** *trans-Restr*:
$trans\ r \Longrightarrow trans(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *Preorder-Restr*:
**assumes** $Preorder\ r$
**shows** $Preorder(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *Partial-order-Restr*:
$Partial\text{-}order\ r \Longrightarrow Partial\text{-}order(Restr\ r\ A)$
⟨*proof*⟩

**lemma** *Linear-order-Restr*:

*Linear-order r* $\Longrightarrow$ *Linear-order*(*Restr r A*)
⟨*proof*⟩

**lemma** *Well-order-Restr*:
**assumes** *Well-order r*
**shows** *Well-order*(*Restr r A*)
⟨*proof*⟩

**lemma** *Field-Restr-subset*: *Field*(*Restr r A*) $\leq$ *A*
⟨*proof*⟩

**lemma** *Refl-Field-Restr*:
*Refl r* $\Longrightarrow$ *Field*(*Restr r A*) = (*Field r*) *Int A*
⟨*proof*⟩

**lemma** *Refl-Field-Restr2*:
⟦*Refl r*; *A* $\leq$ *Field r*⟧ $\Longrightarrow$ *Field*(*Restr r A*) = *A*
⟨*proof*⟩

**lemma** *well-order-on-Restr*:
**assumes** *WELL*: *Well-order r* **and** *SUB*: *A* $\leq$ *Field r*
**shows** *well-order-on A* (*Restr r A*)
⟨*proof*⟩

## 29.2 Order filters versus restrictions and embeddings

**lemma** *Field-Restr-ofilter*:
⟦*Well-order r*; *wo-rel.ofilter r A*⟧ $\Longrightarrow$ *Field*(*Restr r A*) = *A*
⟨*proof*⟩

**lemma** *ofilter-Restr-under*:
**assumes** *WELL*: *Well-order r* **and** *OF*: *wo-rel.ofilter r A* **and** *IN*: *a* $\in$ *A*
**shows** *under* (*Restr r A*) *a* = *under r a*
⟨*proof*⟩

**lemma** *ofilter-embed*:
**assumes** *Well-order r*
**shows** *wo-rel.ofilter r A* = (*A* $\leq$ *Field r* $\wedge$ *embed* (*Restr r A*) *r id*)
⟨*proof*⟩

**lemma** *ofilter-Restr-Int*:
**assumes** *WELL*: *Well-order r* **and** *OFA*: *wo-rel.ofilter r A*
**shows** *wo-rel.ofilter* (*Restr r B*) (*A Int B*)
⟨*proof*⟩

**lemma** *ofilter-Restr-subset*:
**assumes** *WELL*: *Well-order r* **and** *OFA*: *wo-rel.ofilter r A* **and** *SUB*: *A* $\leq$ *B*
**shows** *wo-rel.ofilter* (*Restr r B*) *A*
⟨*proof*⟩

**lemma** *ofilter-subset-embed*:
**assumes** *WELL*: *Well-order r* **and**
*OFA*: *wo-rel.ofilter r A* **and** *OFB*: *wo-rel.ofilter r B*
**shows** $(A \leq B) = (embed\ (Restr\ r\ A)\ (Restr\ r\ B)\ id)$
⟨*proof*⟩

**lemma** *ofilter-subset-embedS-iso*:
**assumes** *WELL*: *Well-order r* **and**
*OFA*: *wo-rel.ofilter r A* **and** *OFB*: *wo-rel.ofilter r B*
**shows** $((A < B) = (embedS\ (Restr\ r\ A)\ (Restr\ r\ B)\ id)) \wedge$
$((A = B) = (iso\ (Restr\ r\ A)\ (Restr\ r\ B)\ id))$
⟨*proof*⟩

**lemma** *ofilter-subset-embedS*:
**assumes** *WELL*: *Well-order r* **and**
*OFA*: *wo-rel.ofilter r A* **and** *OFB*: *wo-rel.ofilter r B*
**shows** $(A < B) = embedS\ (Restr\ r\ A)\ (Restr\ r\ B)\ id$
⟨*proof*⟩

**lemma** *embed-implies-iso-Restr*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*EMB*: *embed r' r f*
**shows** $iso\ r'\ (Restr\ r\ (f\ `\ (Field\ r')))\ f$
⟨*proof*⟩

## 29.3 The strict inclusion on proper ofilters is well-founded

**definition** *ofilterIncl* :: *'a rel ⇒ 'a set rel*
**where**
$ofilterIncl\ r \equiv \{(A,B).\ wo\text{-}rel.ofilter\ r\ A \wedge A \neq Field\ r \wedge$
$wo\text{-}rel.ofilter\ r\ B \wedge B \neq Field\ r \wedge A < B\}$

**lemma** *wf-ofilterIncl*:
**assumes** *WELL*: *Well-order r*
**shows** *wf(ofilterIncl r)*
⟨*proof*⟩

## 29.4 Ordering the well-orders by existence of embeddings

We define three relations between well-orders:

- *ordLeq*, of being embedded (abbreviated $\leq o$);
- *ordLess*, of being strictly embedded (abbreviated $<o$);
- *ordIso*, of being isomorphic (abbreviated $=o$).

The prefix "ord" and the index "o" in these names stand for "ordinal-like". These relations shall be proved to be inter-connected in a similar fashion as the trio $\leq$, $<$, $=$ associated to a total order on a set.

**definition** *ordLeq* :: $('a\ rel * 'a'\ rel)\ set$
**where**
*ordLeq* $= \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ embed\ r\ r'\ f)\}$

**abbreviation** *ordLeq2* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** ‹<=o› *50*)
**where** $r <=o\ r' \equiv (r,r') \in ordLeq$

**abbreviation** *ordLeq3* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** ‹≤o› *50*)
**where** $r \leq o\ r' \equiv r <=o\ r'$

**definition** *ordLess* :: $('a\ rel * 'a'\ rel)\ set$
**where**
*ordLess* $= \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ embedS\ r\ r'\ f)\}$

**abbreviation** *ordLess2* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** ‹<o› *50*)
**where** $r <o\ r' \equiv (r,r') \in ordLess$

**definition** *ordIso* :: $('a\ rel * 'a'\ rel)\ set$
**where**
*ordIso* $= \{(r,r').\ Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge (\exists f.\ iso\ r\ r'\ f)\}$

**abbreviation** *ordIso2* :: $'a\ rel \Rightarrow 'a'\ rel \Rightarrow bool$ (**infix** ‹=o› *50*)
**where** $r =o\ r' \equiv (r,r') \in ordIso$

**lemmas** *ordRels-def* = *ordLeq-def ordLess-def ordIso-def*

**lemma** *ordLeq-Well-order-simp*:
**assumes** $r \leq o\ r'$
**shows** $Well\text{-}order\ r \wedge Well\text{-}order\ r'$
⟨*proof*⟩

Notice that the relations $\leq o$, $<o$, $=o$ connect well-orders on potentially *distinct* types. However, some of the lemmas below, including the next one, restrict implicitly the type of these relations to $(('a\ rel) * ('a\ rel))\ set$ , i.e., to $'a\ rel\ rel$.

**lemma** *ordLeq-reflexive*:
$Well\text{-}order\ r \Longrightarrow r \leq o\ r$
⟨*proof*⟩

**lemma** *ordLeq-transitive*[*trans*]:
**assumes** $r \leq o\ r'$ **and** $r' \leq o\ r''$
**shows** $r \leq o\ r''$
⟨*proof*⟩

**lemma** *ordLeq-total*:
$[\![Well\text{-}order\ r;\ Well\text{-}order\ r']\!] \Longrightarrow r \leq o\ r' \vee r' \leq o\ r$
⟨*proof*⟩

**lemma** *ordIso-reflexive*:
  *Well-order* $r \Longrightarrow r =o\ r$
  ⟨*proof*⟩

**lemma** *ordIso-transitive*[*trans*]:
  **assumes** $*$: $r =o\ r'$ **and** $**$: $r' =o\ r''$
  **shows** $r =o\ r''$
  ⟨*proof*⟩

**lemma** *ordIso-symmetric*:
  **assumes** $*$: $r =o\ r'$
  **shows** $r' =o\ r$
⟨*proof*⟩

**lemma** *ordLeq-ordLess-trans*[*trans*]:
  **assumes** $r \leq o\ r'$ **and** $r' <o\ r''$
  **shows** $r <o\ r''$
⟨*proof*⟩

**lemma** *ordLess-ordLeq-trans*[*trans*]:
  **assumes** $r <o\ r'$ **and** $r' \leq o\ r''$
  **shows** $r <o\ r''$
  ⟨*proof*⟩

**lemma** *ordLeq-ordIso-trans*[*trans*]:
  **assumes** $r \leq o\ r'$ **and** $r' =o\ r''$
  **shows** $r \leq o\ r''$
  ⟨*proof*⟩

**lemma** *ordIso-ordLeq-trans*[*trans*]:
  **assumes** $r =o\ r'$ **and** $r' \leq o\ r''$
  **shows** $r \leq o\ r''$
  ⟨*proof*⟩

**lemma** *ordLess-ordIso-trans*[*trans*]:
  **assumes** $r <o\ r'$ **and** $r' =o\ r''$
  **shows** $r <o\ r''$
  ⟨*proof*⟩

**lemma** *ordIso-ordLess-trans*[*trans*]:
  **assumes** $r =o\ r'$ **and** $r' <o\ r''$
  **shows** $r <o\ r''$
  ⟨*proof*⟩

**lemma** *ordLess-not-embed*:
  **assumes** $r <o\ r'$
  **shows** $\neg(\exists f'.\ embed\ r'\ r\ f')$
⟨*proof*⟩

**lemma** *ordLess-Field*:
**assumes** *OL*: $r1 <o\ r2$ **and** *EMB*: *embed r1 r2 f*
**shows** $\neg\ (f`(Field\ r1) = Field\ r2)$
⟨*proof*⟩

**lemma** *ordLess-iff*:
$r <o\ r' = (Well\text{-}order\ r \wedge Well\text{-}order\ r' \wedge \neg(\exists f'.\ embed\ r'\ r\ f'))$
⟨*proof*⟩

**lemma** *ordLess-irreflexive*: $\neg\ r <o\ r$
⟨*proof*⟩

**lemma** *ordLeq-iff-ordLess-or-ordIso*:
$r \leq o\ r' = (r <o\ r' \vee r =o\ r')$
⟨*proof*⟩

**lemma** *ordIso-iff-ordLeq*:
$(r =o\ r') = (r \leq o\ r' \wedge r' \leq o\ r)$
⟨*proof*⟩

**lemma** *not-ordLess-ordLeq*:
$r <o\ r' \Longrightarrow \neg\ r' \leq o\ r$
⟨*proof*⟩

**lemma** *not-ordLeq-ordLess*:
$r \leq o\ r' \Longrightarrow \neg\ r' <o\ r$
⟨*proof*⟩

**lemma** *ordLess-or-ordLeq*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** $r <o\ r' \vee r' \leq o\ r$
⟨*proof*⟩

**lemma** *not-ordLess-ordIso*:
$r <o\ r' \Longrightarrow \neg\ r =o\ r'$
⟨*proof*⟩

**lemma** *not-ordLeq-iff-ordLess*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** $(\neg\ r' \leq o\ r) = (r <o\ r')$
⟨*proof*⟩

**lemma** *not-ordLess-iff-ordLeq*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** $(\neg\ r' <o\ r) = (r \leq o\ r')$
⟨*proof*⟩

**lemma** *ordLess-transitive*[*trans*]:
$[\![r <o\ r';\ r' <o\ r'']\!] \Longrightarrow r <o\ r''$

⟨*proof*⟩

**corollary** *ordLess-trans*: *trans ordLess*
⟨*proof*⟩

**lemmas** *ordIso-equivalence* = *ordIso-transitive ordIso-reflexive ordIso-symmetric*

**lemma** *ordIso-imp-ordLeq*:
$r =o\ r' \Longrightarrow r \leq o\ r'$
⟨*proof*⟩

**lemma** *ordLess-imp-ordLeq*:
$r <o\ r' \Longrightarrow r \leq o\ r'$
⟨*proof*⟩

**lemma** *ofilter-subset-ordLeq*:
**assumes** *WELL*: *Well-order r* **and**
*OFA*: *wo-rel.ofilter r A* **and** *OFB*: *wo-rel.ofilter r B*
**shows** $(A \leq B) = (Restr\ r\ A \leq o\ Restr\ r\ B)$
⟨*proof*⟩

**lemma** *ofilter-subset-ordLess*:
**assumes** *WELL*: *Well-order r* **and**
*OFA*: *wo-rel.ofilter r A* **and** *OFB*: *wo-rel.ofilter r B*
**shows** $(A < B) = (Restr\ r\ A <o\ Restr\ r\ B)$
⟨*proof*⟩

**lemma** *ofilter-ordLess*:
$[\![Well\text{-}order\ r;\ wo\text{-}rel.ofilter\ r\ A]\!] \Longrightarrow (A < Field\ r) = (Restr\ r\ A <o\ r)$
⟨*proof*⟩

**corollary** *underS-Restr-ordLess*:
**assumes** *Well-order r* **and** $Field\ r \neq \{\}$
**shows** $Restr\ r\ (underS\ r\ a) <o\ r$
⟨*proof*⟩

**lemma** *embed-ordLess-ofilterIncl*:
**assumes**
*OL12*: $r1 <o\ r2$ **and** *OL23*: $r2 <o\ r3$ **and**
*EMB13*: *embed r1 r3 f13* **and** *EMB23*: *embed r2 r3 f23*
**shows** $(f13`(Field\ r1),\ f23`(Field\ r2)) \in (ofilterIncl\ r3)$
⟨*proof*⟩

**lemma** *ordLess-iff-ordIso-Restr*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** $(r' <o\ r) = (\exists\, a \in Field\ r.\ r' =o\ Restr\ r\ (underS\ r\ a))$
⟨*proof*⟩

**lemma** *internalize-ordLess*:

$(r' <o\ r) = (\exists p.\ Field\ p < Field\ r \wedge r' =o\ p \wedge p <o\ r)$
⟨*proof*⟩

**lemma** *internalize-ordLeq*:
$(r' \leq o\ r) = (\exists p.\ Field\ p \leq Field\ r \wedge r' =o\ p \wedge p \leq o\ r)$
⟨*proof*⟩

**lemma** *ordLeq-iff-ordLess-Restr*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'*
**shows** $(r \leq o\ r') = (\forall a \in Field\ r.\ Restr\ r\ (underS\ r\ a) <o\ r')$
⟨*proof*⟩

**lemma** *finite-ordLess-infinite*:
**assumes** *WELL*: *Well-order r* **and** *WELL'*: *Well-order r'* **and**
*FIN*: *finite*(*Field r*) **and** *INF*: ¬*finite*(*Field r'*)
**shows** $r <o\ r'$
⟨*proof*⟩

**lemma** *finite-well-order-on-ordIso*:
**assumes** *FIN*: *finite A* **and**
*WELL*: *well-order-on A r* **and** *WELL'*: *well-order-on A r'*
**shows** $r =o\ r'$
⟨*proof*⟩

## 29.5 $<o$ is well-founded

Of course, it only makes sense to state that the $<o$ is well-founded on the restricted type *'a rel rel*. We prove this by first showing that, for any set of well-orders all embedded in a fixed well-order, the function mapping each well-order in the set to an order filter of the fixed well-order is compatible w.r.t. to $<o$ versus *strict inclusion*; and we already know that strict inclusion of order filters is well-founded.

**definition** *ord-to-filter* :: *'a rel* ⇒ *'a rel* ⇒ *'a set*
**where** *ord-to-filter r0 r* ≡ (*SOME f. embed r r0 f*) ' (*Field r*)

**lemma** *ord-to-filter-compat*:
$compat\ (ordLess\ Int\ (ordLess^{-1}``\{r0\} \times ordLess^{-1}``\{r0\}))$
(*ofilterIncl r0*)
(*ord-to-filter r0*)
⟨*proof*⟩

**theorem** *wf-ordLess*: *wf ordLess*
⟨*proof*⟩

**corollary** *exists-minim-Well-order*:
**assumes** *NE*: $R \neq \{\}$ **and** *WELL*: $\forall r \in R.$ *Well-order r*
**shows** $\exists r \in R.\ \forall r' \in R.\ r \leq o\ r'$
⟨*proof*⟩

## 29.6 Copy via direct images

The direct image operator is the dual of the inverse image operator *inv-image* from *Relation.thy*. It is useful for transporting a well-order between different types.

**definition** *dir-image* :: $'a\ rel \Rightarrow ('a \Rightarrow 'a') \Rightarrow 'a'\ rel$
**where**
$dir\text{-}image\ r\ f = \{(f\ a,\ f\ b)|\ a\ b.\ (a,b) \in r\}$

**lemma** *dir-image-Field*:
$Field(dir\text{-}image\ r\ f) = f\ `\ (Field\ r)$
⟨*proof*⟩

**lemma** *dir-image-minus-Id*:
$inj\text{-}on\ f\ (Field\ r) \Longrightarrow (dir\text{-}image\ r\ f) - Id = dir\text{-}image\ (r - Id)\ f$
⟨*proof*⟩

**lemma** *dir-image-subset*:
**assumes** $r \subseteq A \times B$
**shows** $dir\text{-}image\ r\ f \subseteq f\ `\ A \times f\ `\ B$
⟨*proof*⟩

**lemma** *Refl-dir-image*:
**assumes** *Refl r*
**shows** $Refl(dir\text{-}image\ r\ f)$
⟨*proof*⟩

**lemma** *trans-dir-image*:
**assumes** *TRANS*: *trans r* **and** *INJ*: $inj\text{-}on\ f\ (Field\ r)$
**shows** $trans(dir\text{-}image\ r\ f)$
⟨*proof*⟩

**lemma** *Preorder-dir-image*:
**assumes** *Preorder r* **and** *inj*: $inj\text{-}on\ f\ (Field\ r)$
**shows** $Preorder\ (dir\text{-}image\ r\ f)$
⟨*proof*⟩

**lemma** *antisym-dir-image*:
**assumes** *AN*: *antisym r* **and** *INJ*: $inj\text{-}on\ f\ (Field\ r)$
**shows** $antisym(dir\text{-}image\ r\ f)$
⟨*proof*⟩

**lemma** *Partial-order-dir-image*:
$[\![Partial\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow Partial\text{-}order\ (dir\text{-}image\ r\ f)$
⟨*proof*⟩

**lemma** *Total-dir-image*:
**assumes** *TOT*: *Total r* **and** *INJ*: $inj\text{-}on\ f\ (Field\ r)$
**shows** $Total(dir\text{-}image\ r\ f)$

$\langle proof \rangle$

**lemma** *Linear-order-dir-image*:
$[\![Linear\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow Linear\text{-}order\ (dir\text{-}image\ r\ f)$
$\langle proof \rangle$

**lemma** *wf-dir-image*:
**assumes** *WF*: *wf r* **and** *INJ*: *inj-on f* (*Field r*)
**shows** *wf*(*dir-image r f*)
$\langle proof \rangle$

**lemma** *Well-order-dir-image*:
$[\![Well\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow Well\text{-}order\ (dir\text{-}image\ r\ f)$
$\langle proof \rangle$

**lemma** *dir-image-bij-betw*:
$[\![inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow bij\text{-}betw\ f\ (Field\ r)\ (Field\ (dir\text{-}image\ r\ f))$
$\langle proof \rangle$

**lemma** *dir-image-compat*:
*compat r* (*dir-image r f*) *f*
$\langle proof \rangle$

**lemma** *dir-image-iso*:
$[\![Well\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow iso\ r\ (dir\text{-}image\ r\ f)\ f$
$\langle proof \rangle$

**lemma** *dir-image-ordIso*:
$[\![Well\text{-}order\ r;\ inj\text{-}on\ f\ (Field\ r)]\!] \Longrightarrow r =o\ dir\text{-}image\ r\ f$
$\langle proof \rangle$

**lemma** *Well-order-iso-copy*:
**assumes** *WELL*: *well-order-on A r* **and** *BIJ*: *bij-betw f A A'*
**shows** $\exists\, r'.\ well\text{-}order\text{-}on\ A'\ r' \wedge r =o\ r'$
$\langle proof \rangle$

## 29.7 Bounded square

This construction essentially defines, for an order relation $r$, a lexicographic order *bsqr r* on (*Field r*) $\times$ (*Field r*), applying the following criteria (in this order):

- compare the maximums;
- compare the first components;
- compare the second components.

The only application of this construction that we are aware of is at proving that the square of an infinite set has the same cardinal as that set. The essential property required there (and which is ensured by this construction) is that any proper order filter of the product order is included in a rectangle, i.e., in a product of proper filters on the original relation (assumed to be a well-order).

**definition** *bsqr* :: $'a\ rel => ('a * 'a)rel$
  **where**
    $bsqr\ r = \{((a1,a2),(b1,b2)).$
        $\{a1,a2,b1,b2\} \leq Field\ r\ \wedge$
        $(a1 = b1 \wedge a2 = b2 \vee$
         $(wo\text{-}rel.max2\ r\ a1\ a2,\ wo\text{-}rel.max2\ r\ b1\ b2) \in r - Id \vee$
         $wo\text{-}rel.max2\ r\ a1\ a2 = wo\text{-}rel.max2\ r\ b1\ b2 \wedge (a1,b1) \in r - Id \vee$
         $wo\text{-}rel.max2\ r\ a1\ a2 = wo\text{-}rel.max2\ r\ b1\ b2 \wedge a1 = b1\ \wedge (a2,b2) \in r$
$- Id$
        $)\}$

**lemma** *Field-bsqr*:
  $Field\ (bsqr\ r) = Field\ r \times Field\ r$
⟨*proof*⟩

**lemma** *bsqr-subset*:
  **assumes** $r \subseteq Field\ r \times Field\ r$
  **shows** $bsqr\ r \subseteq Field\ (bsqr\ r) \times Field\ (bsqr\ r)$
  ⟨*proof*⟩

**lemma** *bsqr-Refl*: $Refl(bsqr\ r)$
  ⟨*proof*⟩

**lemma** *bsqr-Trans*:
  **assumes** *Well-order r*
  **shows** *trans* $(bsqr\ r)$
  ⟨*proof*⟩

**lemma** *bsqr-antisym*:
  **assumes** *Well-order r*
  **shows** *antisym* $(bsqr\ r)$
⟨*proof*⟩

**lemma** *bsqr-Total*:
  **assumes** *Well-order r*
  **shows** $Total(bsqr\ r)$
⟨*proof*⟩

**lemma** *bsqr-Linear-order*:
  **assumes** *Well-order r*
  **shows** $Linear\text{-}order(bsqr\ r)$
⟨*proof*⟩

**lemma** *bsqr-Well-order*:
  **assumes** *Well-order r*

**shows** *Well-order*(*bsqr r*)
⟨*proof*⟩

**lemma** *bsqr-max2*:
**assumes** *WELL*: *Well-order r* **and** *LEQ*: $((a1,a2),(b1,b2)) \in$ *bsqr r*
**shows** (*wo-rel.max2 r a1 a2*, *wo-rel.max2 r b1 b2*) $\in r$
⟨*proof*⟩

**lemma** *bsqr-ofilter*:
**assumes** *WELL*: *Well-order r* **and**
*OF*: *wo-rel.ofilter* (*bsqr r*) *D* **and** *SUB*: $D <$ *Field r* $\times$ *Field r* **and**
*NE*: $\neg$ ($\exists$ *a*. *Field r* = *under r a*)
**shows** $\exists$ *A*. *wo-rel.ofilter r A* $\wedge$ $A <$ *Field r* $\wedge$ $D \leq A \times A$
⟨*proof*⟩

**definition** *Func* **where**
*Func A B* = {*f* . ($\forall$ $a \in A$. $f\ a \in B$) $\wedge$ ($\forall$ *a*. $a \notin A \longrightarrow f\ a =$ *undefined*)}

**lemma** *Func-empty*:
*Func* {} *B* = {$\lambda x$. *undefined*}
⟨*proof*⟩

**lemma** *Func-elim*:
**assumes** $g \in$ *Func A B* **and** $a \in A$
**shows** $\exists$ *b*. $b \in B \wedge g\ a = b$
⟨*proof*⟩

**definition** *curr* **where**
*curr A f* $\equiv$ $\lambda$ *a*. *if* $a \in A$ *then* $\lambda b$. *f* (*a*,*b*) *else undefined*

**lemma** *curr-in*:
**assumes** *f*: $f \in$ *Func* ($A \times B$) *C*
**shows** *curr A f* $\in$ *Func A* (*Func B C*)
⟨*proof*⟩

**lemma** *curr-inj*:
**assumes** $f1 \in$ *Func* ($A \times B$) *C* **and** $f2 \in$ *Func* ($A \times B$) *C*
**shows** *curr A f1* = *curr A f2* $\longleftrightarrow$ $f1 = f2$
⟨*proof*⟩

**lemma** *curr-surj*:
**assumes** $g \in$ *Func A* (*Func B C*)
**shows** $\exists$ $f \in$ *Func* ($A \times B$) *C*. *curr A f* = *g*
⟨*proof*⟩

**lemma** *bij-betw-curr*:
*bij-betw* (*curr A*) (*Func* ($A \times B$) *C*) (*Func A* (*Func B C*))
⟨*proof*⟩

**definition** *Func-map* **where**
*Func-map B2 f1 f2 g b2* $\equiv$ *if b2* $\in$ *B2 then f1* (*g* (*f2 b2*)) *else undefined*

**lemma** *Func-map*:
**assumes** *g*: *g* $\in$ *Func A2 A1* **and** *f1*: *f1* ' *A1* $\subseteq$ *B1* **and** *f2*: *f2* ' *B2* $\subseteq$ *A2*
**shows** *Func-map B2 f1 f2 g* $\in$ *Func B2 B1*
⟨*proof*⟩

**lemma** *Func-non-emp*:
**assumes** *B* $\neq$ {}
**shows** *Func A B* $\neq$ {}
⟨*proof*⟩

**lemma** *Func-is-emp*:
*Func A B* = {} $\longleftrightarrow$ *A* $\neq$ {} $\wedge$ *B* = {} (**is** *?L* $\longleftrightarrow$ *?R*)
⟨*proof*⟩

**lemma** *Func-map-surj*:
**assumes** *B1*: *f1* ' *A1* = *B1* **and** *A2*: *inj-on f2 B2 f2* ' *B2* $\subseteq$ *A2*
**and** *B2A2*: *B2* = {} $\Longrightarrow$ *A2* = {}
**shows** *Func B2 B1* = *Func-map B2 f1 f2* ' *Func A2 A1*
⟨*proof*⟩

**end**

# 30 Cardinal-Order Relations as Needed by Bounded Natural Functors

**theory** *BNF-Cardinal-Order-Relation*
**imports** *Zorn BNF-Wellorder-Constructions*
**begin**

In this section, we define cardinal-order relations to be minim well-orders on their field. Then we define the cardinal of a set to be *some* cardinal-order relation on that set, which will be unique up to order isomorphism. Then we study the connection between cardinals and:

- standard set-theoretic constructions: products, sums, unions, lists, powersets, set-of finite sets operator;
- finiteness and infiniteness (in particular, with the numeric cardinal operator for finite sets, *card*, from the theory *Finite-Sets.thy*).

On the way, we define the canonical $\omega$ cardinal and finite cardinals. We also define (again, up to order isomorphism) the successor of a cardinal, and show that any cardinal admits a successor.

Main results of this section are the existence of cardinal relations and the facts that, in the presence of infiniteness, most of the standard set-theoretic constructions (except for the powerset) *do not increase cardinality*. In particular, e.g., the set of words/lists over any infinite set has the same cardinality (hence, is in bijection) with that set.

## 30.1 Cardinal orders

A cardinal order in our setting shall be a well-order *minim* w.r.t. the order-embedding relation, $\leq o$ (which is the same as being *minimal* w.r.t. the strict order-embedding relation, $<o$), among all the well-orders on its field.

**definition** *card-order-on* :: *'a set* ⇒ *'a rel* ⇒ *bool*
  **where**
  $\textit{card-order-on } A\ r \equiv \textit{well-order-on } A\ r \wedge (\forall r'.\ \textit{well-order-on } A\ r' \longrightarrow r \leq o\ r')$

**abbreviation** $\textit{Card-order } r \equiv \textit{card-order-on } (\textit{Field } r)\ r$
**abbreviation** $\textit{card-order } r \equiv \textit{card-order-on } \textit{UNIV } r$

**lemma** *card-order-on-well-order-on*:
  **assumes** *card-order-on A r*
  **shows** *well-order-on A r*
  ⟨*proof*⟩

**lemma** *card-order-on-Card-order*:
  $\textit{card-order-on } A\ r \Longrightarrow A = \textit{Field } r \wedge \textit{Card-order } r$
  ⟨*proof*⟩

The existence of a cardinal relation on any given set (which will mean that any set has a cardinal) follows from two facts:

- Zermelo's theorem (proved in *Zorn.thy* as theorem *well-order-on*), which states that on any given set there exists a well-order;
- The well-founded-ness of $<o$, ensuring that then there exists a minimal such well-order, i.e., a cardinal order.

**theorem** *card-order-on*: $\exists r.\ \textit{card-order-on } A\ r$
⟨*proof*⟩

**lemma** *card-order-on-ordIso*:
  **assumes** *CO*: *card-order-on A r* **and** *CO'*: *card-order-on A r'*
  **shows** $r =o\ r'$
  ⟨*proof*⟩

**lemma** *Card-order-ordIso*:
  **assumes** *CO*: *Card-order r* **and** *ISO*: $r' =o\ r$
  **shows** *Card-order r'*
  ⟨*proof*⟩

**lemma** *Card-order-ordIso2*:

**assumes** *CO*: *Card-order* $r$ **and** *ISO*: $r =o\ r'$
**shows** *Card-order* $r'$
⟨*proof*⟩

## 30.2 Cardinal of a set

We define the cardinal of set to be *some* cardinal order on that set. We shall prove that this notion is unique up to order isomorphism, meaning that order isomorphism shall be the true identity of cardinals.

**definition** *card-of* :: *'a set* ⇒ *'a rel* (‹(‹*open-block notation*=‹*mixfix card-of*››|-|)›)
**where** *card-of* $A$ = (*SOME* $r$. *card-order-on* $A$ $r$)

**lemma** *card-of-card-order-on*: *card-order-on* $A$ $|A|$
⟨*proof*⟩

**lemma** *card-of-well-order-on*: *well-order-on* $A$ $|A|$
⟨*proof*⟩

**lemma** *Field-card-of*: *Field* $|A| = A$
⟨*proof*⟩

**lemma** *card-of-Card-order*: *Card-order* $|A|$
⟨*proof*⟩

**corollary** *ordIso-card-of-imp-Card-order*:
$r =o\ |A| \Longrightarrow$ *Card-order* $r$
⟨*proof*⟩

**lemma** *card-of-Well-order*: *Well-order* $|A|$
⟨*proof*⟩

**lemma** *card-of-refl*: $|A| =o\ |A|$
⟨*proof*⟩

**lemma** *card-of-least*: *well-order-on* $A$ $r \Longrightarrow |A| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-ordIso*:
$(\exists f.$ *bij-betw* $f\ A\ B) = (\ |A| =o\ |B|\ )$
⟨*proof*⟩

**lemma** *card-of-ordLeq*:
$(\exists f.$ *inj-on* $f\ A \wedge f$ ‘ $A \leq B) = (\ |A| \leq o\ |B|\ )$
⟨*proof*⟩

**lemma** *card-of-ordLeq2*:
$A \neq \{\} \Longrightarrow (\exists g.\ g$ ‘ $B = A) = (\ |A| \leq o\ |B|\ )$
⟨*proof*⟩

**lemma** *card-of-ordLess*:
$(\neg(\exists f.\ \textit{inj-on}\ f\ A \wedge f\ `\ A \leq B)) = (\ |B| <o\ |A|\ )$
⟨*proof*⟩

**lemma** *card-of-ordLess2*:
$B \neq \{\} \Longrightarrow (\neg(\exists f.\ f\ `\ A = B)) = (\ |A| <o\ |B|\ )$
⟨*proof*⟩

**lemma** *card-of-ordIsoI*:
**assumes** *bij-betw f A B*
**shows** $|A| =o\ |B|$
⟨*proof*⟩

**lemma** *card-of-ordLeqI*:
**assumes** *inj-on f A* **and** $\bigwedge a.\ a \in A \Longrightarrow f\ a \in B$
**shows** $|A| \leq o\ |B|$
⟨*proof*⟩

**lemma** *card-of-unique*:
$\textit{card-order-on}\ A\ r \Longrightarrow r =o\ |A|$
⟨*proof*⟩

**lemma** *card-of-mono1*:
$A \leq B \Longrightarrow |A| \leq o\ |B|$
⟨*proof*⟩

**lemma** *card-of-mono2*:
**assumes** $r \leq o\ r'$
**shows** $|\textit{Field}\ r| \leq o\ |\textit{Field}\ r'|$
⟨*proof*⟩

**lemma** *card-of-cong*: $r =o\ r' \Longrightarrow |\textit{Field}\ r| =o\ |\textit{Field}\ r'|$
⟨*proof*⟩

**lemma** *card-of-Field-ordIso*:
**assumes** *Card-order r*
**shows** $|\textit{Field}\ r| =o\ r$
⟨*proof*⟩

**lemma** *Card-order-iff-ordIso-card-of*:
$\textit{Card-order}\ r = (r =o\ |\textit{Field}\ r|\ )$
⟨*proof*⟩

**lemma** *Card-order-iff-ordLeq-card-of*:
$\textit{Card-order}\ r = (r \leq o\ |\textit{Field}\ r|\ )$
⟨*proof*⟩

**lemma** *Card-order-iff-Restr-underS*:
**assumes** *Well-order r*

**shows** $Card\text{-}order\ r = (\forall a \in Field\ r.\ Restr\ r\ (underS\ r\ a) <o\ |Field\ r|\ )$
⟨*proof*⟩

**lemma** *card-of-underS*:
**assumes** $r$: $Card\text{-}order\ r$ **and** $a$: $a \in Field\ r$
**shows** $|underS\ r\ a| <o\ r$
⟨*proof*⟩

**lemma** *ordLess-Field*:
**assumes** $r <o\ r'$
**shows** $|Field\ r| <o\ r'$
⟨*proof*⟩

**lemma** *internalize-card-of-ordLeq*:
$(\ |A| \leq o\ r) = (\exists B \leq Field\ r.\ |A| =o\ |B| \wedge |B| \leq o\ r)$
⟨*proof*⟩

**lemma** *internalize-card-of-ordLeq2*:
$(\ |A| \leq o\ |C|\ ) = (\exists B \leq C.\ |A| =o\ |B| \wedge |B| \leq o\ |C|\ )$
⟨*proof*⟩

## 30.3 Cardinals versus set operations on arbitrary sets

Here we embark in a long journey of simple results showing that the standard set-theoretic operations are well-behaved w.r.t. the notion of cardinal – essentially, this means that they preserve the "cardinal identity" $=o$ and are monotonic w.r.t. $\leq o$.

**lemma** *card-of-empty*: $|\{\}| \leq o\ |A|$
⟨*proof*⟩

**lemma** *card-of-empty1*:
**assumes** $Well\text{-}order\ r \vee Card\text{-}order\ r$
**shows** $|\{\}| \leq o\ r$
⟨*proof*⟩

**corollary** *Card-order-empty*:
$Card\text{-}order\ r \Longrightarrow |\{\}| \leq o\ r$ ⟨*proof*⟩

**lemma** *card-of-empty2*:
**assumes** $|A| =o\ |\{\}|$
**shows** $A = \{\}$
⟨*proof*⟩

**lemma** *card-of-empty3*:
**assumes** $|A| \leq o\ |\{\}|$
**shows** $A = \{\}$
⟨*proof*⟩

**lemma** *card-of-empty-ordIso*:
$|\{\}::'a\ set| =o\ |\{\}::'b\ set|$
⟨*proof*⟩

**lemma** *card-of-image*:
$|f\ `\ A| \leq o\ |A|$
⟨*proof*⟩

**lemma** *surj-imp-ordLeq*:
**assumes** $B \subseteq f\ `\ A$
**shows** $|B| \leq o\ |A|$
⟨*proof*⟩

**lemma** *card-of-singl-ordLeq*:
**assumes** $A \neq \{\}$
**shows** $|\{b\}| \leq o\ |A|$
⟨*proof*⟩

**corollary** *Card-order-singl-ordLeq*:
$[\![Card\text{-}order\ r;\ Field\ r \neq \{\}]\!] \Longrightarrow |\{b\}| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Pow*: $|A| <o\ |Pow\ A|$
⟨*proof*⟩

**corollary** *Card-order-Pow*:
$Card\text{-}order\ r \Longrightarrow r <o\ |Pow(Field\ r)|$
⟨*proof*⟩

**lemma** *card-of-Plus1*: $|A| \leq o\ |A <+> B|$ **and** *card-of-Plus2*: $|B| \leq o\ |A <+> B|$
⟨*proof*⟩

**corollary** *Card-order-Plus1*:
$Card\text{-}order\ r \Longrightarrow r \leq o\ |(Field\ r) <+> B|$
⟨*proof*⟩

**corollary** *Card-order-Plus2*:
$Card\text{-}order\ r \Longrightarrow r \leq o\ |A <+> (Field\ r)|$
⟨*proof*⟩

**lemma** *card-of-Plus-empty1*: $|A| =o\ |A <+> \{\}|$
⟨*proof*⟩

**lemma** *card-of-Plus-empty2*: $|A| =o\ |\{\} <+> A|$
⟨*proof*⟩

**lemma** *card-of-Plus-commute*: $|A <+> B| =o\ |B <+> A|$
⟨*proof*⟩

**lemma** *card-of-Plus-assoc*:
**fixes** $A :: 'a\ set$ **and** $B :: 'b\ set$ **and** $C :: 'c\ set$
**shows** $|(A <+> B) <+> C| =o |A <+> B <+> C|$
⟨*proof*⟩

**lemma** *card-of-Plus-mono1*:
**assumes** $|A| \leq o\ |B|$
**shows** $|A <+> C| \leq o\ |B <+> C|$
⟨*proof*⟩

**corollary** *ordLeq-Plus-mono1*:
**assumes** $r \leq o\ r'$
**shows** $|(Field\ r) <+> C| \leq o\ |(Field\ r') <+> C|$
⟨*proof*⟩

**lemma** *card-of-Plus-mono2*:
**assumes** $|A| \leq o\ |B|$
**shows** $|C <+> A| \leq o\ |C <+> B|$
⟨*proof*⟩

**corollary** *ordLeq-Plus-mono2*:
**assumes** $r \leq o\ r'$
**shows** $|A <+> (Field\ r)| \leq o\ |A <+> (Field\ r')|$
⟨*proof*⟩

**lemma** *card-of-Plus-mono*:
**assumes** $|A| \leq o\ |B|$ **and** $|C| \leq o\ |D|$
**shows** $|A <+> C| \leq o\ |B <+> D|$
⟨*proof*⟩

**corollary** *ordLeq-Plus-mono*:
**assumes** $r \leq o\ r'$ **and** $p \leq o\ p'$
**shows** $|(Field\ r) <+> (Field\ p)| \leq o\ |(Field\ r') <+> (Field\ p')|$
⟨*proof*⟩

**lemma** *card-of-Plus-cong1*:
**assumes** $|A| =o |B|$
**shows** $|A <+> C| =o |B <+> C|$
⟨*proof*⟩

**corollary** *ordIso-Plus-cong1*:
**assumes** $r =o r'$
**shows** $|(Field\ r) <+> C| =o |(Field\ r') <+> C|$
⟨*proof*⟩

**lemma** *card-of-Plus-cong2*:
**assumes** $|A| =o |B|$
**shows** $|C <+> A| =o |C <+> B|$
⟨*proof*⟩

**corollary** *ordIso-Plus-cong2*:
**assumes** $r =o\ r'$
**shows** $|A <+> (Field\ r)| =o\ |A <+> (Field\ r')|$
⟨*proof*⟩

**lemma** *card-of-Plus-cong*:
**assumes** $|A| =o\ |B|$ **and** $|C| =o\ |D|$
**shows** $|A <+> C| =o\ |B <+> D|$
⟨*proof*⟩

**corollary** *ordIso-Plus-cong*:
**assumes** $r =o\ r'$ **and** $p =o\ p'$
**shows** $|(Field\ r) <+> (Field\ p)| =o\ |(Field\ r') <+> (Field\ p')|$
⟨*proof*⟩

**lemma** *card-of-Un-Plus-ordLeq*:
$|A \cup B| \leq o\ |A <+> B|$
⟨*proof*⟩

**lemma** *card-of-Times1*:
**assumes** $A \neq \{\}$
**shows** $|B| \leq o\ |B \times A|$
⟨*proof*⟩

**lemma** *card-of-Times-commute*: $|A \times B| =o\ |B \times A|$
⟨*proof*⟩

**lemma** *card-of-Times2*:
**assumes** $A \neq \{\}$ **shows** $|B| \leq o\ |A \times B|$
⟨*proof*⟩

**corollary** *Card-order-Times1*:
$[\![Card\text{-}order\ r;\ B \neq \{\}]\!] \Longrightarrow r \leq o\ |(Field\ r) \times B|$
⟨*proof*⟩

**corollary** *Card-order-Times2*:
$[\![Card\text{-}order\ r;\ A \neq \{\}]\!] \Longrightarrow r \leq o\ |A \times (Field\ r)|$
⟨*proof*⟩

**lemma** *card-of-Times3*: $|A| \leq o\ |A \times A|$
⟨*proof*⟩

**lemma** *card-of-Plus-Times-bool*: $|A <+> A| =o\ |A \times (UNIV::bool\ set)|$
⟨*proof*⟩

**lemma** *card-of-Times-mono1*:
**assumes** $|A| \leq o\ |B|$
**shows** $|A \times C| \leq o\ |B \times C|$

⟨*proof*⟩

**corollary** *ordLeq-Times-mono1*:
  **assumes** $r \le o\ r'$
  **shows** $|(Field\ r) \times C| \le o\ |(Field\ r') \times C|$
  ⟨*proof*⟩

**lemma** *card-of-Times-mono2*:
  **assumes** $|A| \le o\ |B|$
  **shows** $|C \times A| \le o\ |C \times B|$
  ⟨*proof*⟩

**corollary** *ordLeq-Times-mono2*:
  **assumes** $r \le o\ r'$
  **shows** $|A \times (Field\ r)| \le o\ |A \times (Field\ r')|$
  ⟨*proof*⟩

**lemma** *card-of-Sigma-mono1*:
  **assumes** $\forall\, i \in I.\ |A\ i| \le o\ |B\ i|$
  **shows** $|SIGMA\ i : I.\ A\ i| \le o\ |SIGMA\ i : I.\ B\ i|$
⟨*proof*⟩

**lemma** *card-of-UNION-Sigma*:
  $|\bigcup i \in I.\ A\ i| \le o\ |SIGMA\ i : I.\ A\ i|$
  ⟨*proof*⟩

**lemma** *card-of-bool*:
  **assumes** $a1 \neq a2$
  **shows** $|UNIV::bool\ set| =o\ |\{a1,a2\}|$
⟨*proof*⟩

**lemma** *card-of-Plus-Times-aux*:
  **assumes** $A2$: $a1 \neq a2 \wedge \{a1,a2\} \le A$ **and**
    $LEQ$: $|A| \le o\ |B|$
  **shows** $|A <+> B| \le o\ |A \times B|$
⟨*proof*⟩

**lemma** *card-of-Plus-Times*:
  **assumes** $A2$: $a1 \neq a2 \wedge \{a1,a2\} \le A$ **and** $B2$: $b1 \neq b2 \wedge \{b1,b2\} \le B$
  **shows** $|A <+> B| \le o\ |A \times B|$
⟨*proof*⟩

**lemma** *card-of-Times-Plus-distrib*:
  $|A \times (B <+> C)| =o\ |A \times B <+> A \times C|$ (**is** $|?RHS| =o\ |?LHS|$)
⟨*proof*⟩

**lemma** *card-of-ordLeq-finite*:
  **assumes** $|A| \le o\ |B|$ **and** *finite B*
  **shows** *finite A*

⟨*proof*⟩

**lemma** *card-of-ordLeq-infinite*:
**assumes** $|A| \leq o\ |B|$ **and** $\neg$ *finite A*
**shows** $\neg$ *finite B*
⟨*proof*⟩

**lemma** *card-of-ordIso-finite*:
**assumes** $|A| =o\ |B|$
**shows** *finite A* = *finite B*
⟨*proof*⟩

**lemma** *card-of-ordIso-finite-Field*:
**assumes** *Card-order r* **and** $r =o\ |A|$
**shows** *finite*(*Field r*) = *finite A*
⟨*proof*⟩

## 30.4 Cardinals versus set operations involving infinite sets

Here we show that, for infinite sets, most set-theoretic constructions do not increase the cardinality. The cornerstone for this is theorem *Card-order-Times-same-infinite*, which states that self-product does not increase cardinality – the proof of this fact adapts a standard set-theoretic argument, as presented, e.g., in the proof of theorem 1.5.11 at page 47 in [4]. Then everything else follows fairly easily.

**lemma** *infinite-iff-card-of-nat*:
$\neg$ *finite A* $\longleftrightarrow$ ( $|UNIV{::}nat\ set| \leq o\ |A|$ )
⟨*proof*⟩

The next two results correspond to the ZF fact that all infinite cardinals are limit ordinals:

**lemma** *Card-order-infinite-not-under*:
**assumes** *CARD*: *Card-order r* **and** *INF*: $\neg$*finite* (*Field r*)
**shows** $\neg$ ($\exists\, a.$ *Field r* = *under r a*)
⟨*proof*⟩

**lemma** *infinite-Card-order-limit*:
**assumes** *r*: *Card-order r* **and** $\neg$*finite* (*Field r*)
**and** *a*: $a \in$ *Field r*
**shows** $\exists\, b \in$ *Field r*. $a \neq b \wedge (a,b) \in r$
⟨*proof*⟩

**theorem** *Card-order-Times-same-infinite*:
**assumes** *CO*: *Card-order r* **and** *INF*: $\neg$*finite*(*Field r*)
**shows** $|Field\ r \times Field\ r| \leq o\ r$
⟨*proof*⟩

**corollary** *card-of-Times-same-infinite*:

**assumes** $\neg$*finite* $A$
**shows** $|A \times A| =o |A|$
⟨*proof*⟩

**lemma** *card-of-Times-infinite*:
**assumes** *INF*: $\neg$*finite* $A$ **and** *NE*: $B \neq \{\}$ **and** *LEQ*: $|B| \leq o |A|$
**shows** $|A \times B| =o |A| \wedge |B \times A| =o |A|$
⟨*proof*⟩

**corollary** *Card-order-Times-infinite*:
**assumes** *INF*: $\neg$*finite*(*Field* $r$) **and** *CARD*: *Card-order* $r$ **and**
*NE*: *Field* $p \neq \{\}$ **and** *LEQ*: $p \leq o\ r$
**shows** $|\ (Field\ r) \times (Field\ p)\ | =o\ r \wedge |\ (Field\ p) \times (Field\ r)\ | =o\ r$
⟨*proof*⟩

**lemma** *card-of-Sigma-ordLeq-infinite*:
**assumes** *INF*: $\neg$*finite* $B$ **and**
*LEQ-I*: $|I| \leq o |B|$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o |B|$
**shows** $|SIGMA\ i : I.\ A\ i| \leq o |B|$
⟨*proof*⟩

**lemma** *card-of-Sigma-ordLeq-infinite-Field*:
**assumes** *INF*: $\neg$*finite* (*Field* $r$) **and** $r$: *Card-order* $r$ **and**
*LEQ-I*: $|I| \leq o\ r$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o\ r$
**shows** $|SIGMA\ i : I.\ A\ i| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Times-ordLeq-infinite-Field*:
$[\![\neg finite\ (Field\ r);\ |A| \leq o\ r;\ |B| \leq o\ r;\ Card\text{-}order\ r]\!] \Longrightarrow |A \times B| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Times-infinite-simps*:
$[\![\neg finite\ A;\ B \neq \{\};\ |B| \leq o |A|]\!] \Longrightarrow |A \times B| =o |A|$
$[\![\neg finite\ A;\ B \neq \{\};\ |B| \leq o |A|]\!] \Longrightarrow |A| =o |A \times B|$
$[\![\neg finite\ A;\ B \neq \{\};\ |B| \leq o |A|]\!] \Longrightarrow |B \times A| =o |A|$
$[\![\neg finite\ A;\ B \neq \{\};\ |B| \leq o |A|]\!] \Longrightarrow |A| =o |B \times A|$
⟨*proof*⟩

**lemma** *card-of-UNION-ordLeq-infinite*:
**assumes** *INF*: $\neg$*finite* $B$ **and** *LEQ-I*: $|I| \leq o |B|$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o |B|$
**shows** $|\bigcup i \in I.\ A\ i| \leq o |B|$
⟨*proof*⟩

**corollary** *card-of-UNION-ordLeq-infinite-Field*:
**assumes** *INF*: $\neg$*finite* (*Field* $r$) **and** $r$: *Card-order* $r$ **and**
*LEQ-I*: $|I| \leq o\ r$ **and** *LEQ*: $\forall i \in I.\ |A\ i| \leq o\ r$
**shows** $|\bigcup i \in I.\ A\ i| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Plus-infinite1*:
**assumes** *INF*: $\neg$*finite* $A$ **and** *LEQ*: $|B| \leq o\ |A|$
**shows** $|A <+> B| =o\ |A|$
⟨*proof*⟩

**lemma** *card-of-Plus-infinite2*:
**assumes** *INF*: $\neg$*finite* $A$ **and** *LEQ*: $|B| \leq o\ |A|$
**shows** $|B <+> A| =o\ |A|$
⟨*proof*⟩

**lemma** *card-of-Plus-infinite*:
**assumes** *INF*: $\neg$*finite* $A$ **and** *LEQ*: $|B| \leq o\ |A|$
**shows** $|A <+> B| =o\ |A| \wedge |B <+> A| =o\ |A|$
⟨*proof*⟩

**corollary** *Card-order-Plus-infinite*:
**assumes** *INF*: $\neg$*finite*(*Field* $r$) **and** *CARD*: *Card-order* $r$ **and**
*LEQ*: $p \leq o\ r$
**shows** $|$ (*Field* $r$) $<+>$ (*Field* $p$) $| =o\ r \wedge |$ (*Field* $p$) $<+>$ (*Field* $r$) $| =o\ r$
⟨*proof*⟩

## 30.5 The cardinal $\omega$ and the finite cardinals

The cardinal $\omega$, of natural numbers, shall be the standard non-strict order relation on *nat*, that we abbreviate by *natLeq*. The finite cardinals shall be the restrictions of these relations to the numbers smaller than fixed numbers $n$, that we abbreviate by *natLeq-on* $n$.

**definition** (*natLeq*::(*nat* $*$ *nat*) *set*) $\equiv \{(x,y).\ x \leq y\}$
**definition** (*natLess*::(*nat* $*$ *nat*) *set*) $\equiv \{(x,y).\ x < y\}$

**abbreviation** *natLeq-on* :: *nat* $\Rightarrow$ (*nat* $*$ *nat*) *set*
**where** *natLeq-on* $n \equiv \{(x,y).\ x < n \wedge y < n \wedge x \leq y\}$

**lemma** *infinite-cartesian-product*:
**assumes** $\neg$*finite* $A$ $\neg$*finite* $B$
**shows** $\neg$*finite* $(A \times B)$
⟨*proof*⟩

### 30.5.1 First as well-orders

**lemma** *Field-natLeq*: *Field natLeq* = (*UNIV*::*nat set*)
⟨*proof*⟩

**lemma** *natLeq-Refl*: *Refl natLeq*
⟨*proof*⟩

**lemma** *natLeq-trans*: *trans natLeq*
⟨*proof*⟩

**lemma** *natLeq-Preorder*: *Preorder natLeq*
⟨*proof*⟩

**lemma** *natLeq-antisym*: *antisym natLeq*
⟨*proof*⟩

**lemma** *natLeq-Partial-order*: *Partial-order natLeq*
⟨*proof*⟩

**lemma** *natLeq-Total*: *Total natLeq*
⟨*proof*⟩

**lemma** *natLeq-Linear-order*: *Linear-order natLeq*
⟨*proof*⟩

**lemma** *natLeq-natLess-Id*: $natLess = natLeq - Id$
⟨*proof*⟩

**lemma** *natLeq-Well-order*: *Well-order natLeq*
⟨*proof*⟩

**lemma** *Field-natLeq-on*: $Field\ (natLeq\text{-}on\ n) = \{x.\ x < n\}$
⟨*proof*⟩

**lemma** *natLeq-underS-less*: $underS\ natLeq\ n = \{x.\ x < n\}$
⟨*proof*⟩

**lemma** *Restr-natLeq*: $Restr\ natLeq\ \{x.\ x < n\} = natLeq\text{-}on\ n$
⟨*proof*⟩

**lemma** *Restr-natLeq2*:
$Restr\ natLeq\ (underS\ natLeq\ n) = natLeq\text{-}on\ n$
⟨*proof*⟩

**lemma** *natLeq-on-Well-order*: $Well\text{-}order(natLeq\text{-}on\ n)$
⟨*proof*⟩

**corollary** *natLeq-on-well-order-on*: $well\text{-}order\text{-}on\ \{x.\ x < n\}\ (natLeq\text{-}on\ n)$
⟨*proof*⟩

**lemma** *natLeq-on-wo-rel*: $wo\text{-}rel(natLeq\text{-}on\ n)$
⟨*proof*⟩

### 30.5.2 Then as cardinals

**lemma** *natLeq-Card-order*: *Card-order natLeq*
⟨*proof*⟩

**corollary** *card-of-Field-natLeq*:

$|Field\ natLeq| =o\ natLeq$
$\langle proof \rangle$

**corollary** *card-of-nat*:
$|UNIV::nat\ set| =o\ natLeq$
$\langle proof \rangle$

**corollary** *infinite-iff-natLeq-ordLeq*:
$\neg finite\ A = (\ natLeq \leq o\ |A|\ )$
$\langle proof \rangle$

**corollary** *finite-iff-ordLess-natLeq*:
$finite\ A = (\ |A| <o\ natLeq)$
$\langle proof \rangle$

## 30.6 The successor of a cardinal

First we define $isCardSuc\ r\ r'$, the notion of $r'$ being a successor cardinal of $r$. Although the definition does not require $r$ to be a cardinal, only this case will be meaningful.

**definition** $isCardSuc :: 'a\ rel \Rightarrow 'a\ set\ rel \Rightarrow bool$
**where**
$isCardSuc\ r\ r' \equiv$
$Card\text{-}order\ r' \wedge r <o\ r' \wedge$
$(\forall (r''::'a\ set\ rel).\ Card\text{-}order\ r'' \wedge r <o\ r'' \longrightarrow r' \leq o\ r'')$

Now we introduce the cardinal-successor operator *cardSuc*, by picking *some* cardinal-order relation fulfilling *isCardSuc*. Again, the picked item shall be proved unique up to order-isomorphism.

**definition** $cardSuc :: 'a\ rel \Rightarrow 'a\ set\ rel$
**where** $cardSuc\ r \equiv SOME\ r'.\ isCardSuc\ r\ r'$

**lemma** *exists-minim-Card-order*:
$[\![R \neq \{\}; \forall r \in R.\ Card\text{-}order\ r]\!] \Longrightarrow \exists r \in R.\ \forall r' \in R.\ r \leq o\ r'$
$\langle proof \rangle$

**lemma** *exists-isCardSuc*:
**assumes** $Card\text{-}order\ r$
**shows** $\exists r'.\ isCardSuc\ r\ r'$
$\langle proof \rangle$

**lemma** *cardSuc-isCardSuc*:
**assumes** $Card\text{-}order\ r$
**shows** $isCardSuc\ r\ (cardSuc\ r)$
$\langle proof \rangle$

**lemma** *cardSuc-Card-order*:
$Card\text{-}order\ r \Longrightarrow Card\text{-}order(cardSuc\ r)$

⟨*proof*⟩

**lemma** *cardSuc-greater*:
$Card\text{-}order\ r \Longrightarrow r <o\ cardSuc\ r$
⟨*proof*⟩

**lemma** *cardSuc-ordLeq*:
$Card\text{-}order\ r \Longrightarrow r \leq o\ cardSuc\ r$
⟨*proof*⟩

The minimality property of *cardSuc* originally present in its definition is local to the type $'a\ set\ rel$, i.e., that of *cardSuc r*:

**lemma** *cardSuc-least-aux*:
$[\![Card\text{-}order\ (r::'a\ rel);\ Card\text{-}order\ (r'::'a\ set\ rel);\ r <o\ r']\!] \Longrightarrow cardSuc\ r \leq o\ r'$
⟨*proof*⟩

But from this we can infer general minimality:

**lemma** *cardSuc-least*:
**assumes** *CARD*: $Card\text{-}order\ r$ **and** $CARD'$: $Card\text{-}order\ r'$ **and** *LESS*: $r <o\ r'$
**shows** $cardSuc\ r \leq o\ r'$
⟨*proof*⟩

**lemma** *cardSuc-ordLess-ordLeq*:
**assumes** *CARD*: $Card\text{-}order\ r$ **and** $CARD'$: $Card\text{-}order\ r'$
**shows** $(r <o\ r') = (cardSuc\ r \leq o\ r')$
⟨*proof*⟩

**lemma** *cardSuc-ordLeq-ordLess*:
**assumes** *CARD*: $Card\text{-}order\ r$ **and** $CARD'$: $Card\text{-}order\ r'$
**shows** $(r' <o\ cardSuc\ r) = (r' \leq o\ r)$
⟨*proof*⟩

**lemma** *cardSuc-mono-ordLeq*:
**assumes** *CARD*: $Card\text{-}order\ r$ **and** $CARD'$: $Card\text{-}order\ r'$
**shows** $(cardSuc\ r \leq o\ cardSuc\ r') = (r \leq o\ r')$
⟨*proof*⟩

**lemma** *cardSuc-invar-ordIso*:
**assumes** *CARD*: $Card\text{-}order\ r$ **and** $CARD'$: $Card\text{-}order\ r'$
**shows** $(cardSuc\ r =o\ cardSuc\ r') = (r =o\ r')$
⟨*proof*⟩

**lemma** *card-of-cardSuc-finite*:
$finite(Field(cardSuc\ |A|\ )) = finite\ A$
⟨*proof*⟩

**lemma** *cardSuc-finite*:
**assumes** $Card\text{-}order\ r$
**shows** $finite\ (Field\ (cardSuc\ r)) = finite\ (Field\ r)$

⟨*proof*⟩

**lemma** *Field-cardSuc-not-empty*:
**assumes** *Card-order r*
**shows** *Field* (*cardSuc r*) $\neq$ {}
⟨*proof*⟩

**typedef** *'a suc* = *Field* (*cardSuc* |*UNIV* :: *'a set*| )
⟨*proof*⟩

**definition** *card-suc* :: *'a rel* $\Rightarrow$ *'a suc rel* **where**
*card-suc* $\equiv$ $\lambda$-. *map-prod Abs-suc Abs-suc* ' *cardSuc* |*UNIV* :: *'a set*|

**lemma** *Field-card-suc*: *Field* (*card-suc r*) = *UNIV*
⟨*proof*⟩

**lemma** *card-suc-alt*: *card-suc r* = *dir-image* (*cardSuc* |*UNIV* :: *'a set*| ) *Abs-suc*
⟨*proof*⟩

**lemma** *cardSuc-Well-order*: *Card-order r* $\Longrightarrow$ *Well-order*(*cardSuc r*)
⟨*proof*⟩

**lemma** *cardSuc-ordIso-card-suc*:
**assumes** *card-order r*
**shows** *cardSuc r =o card-suc* (*r* :: *'a rel*)
⟨*proof*⟩

**lemma** *Card-order-card-suc*: *card-order r* $\Longrightarrow$ *Card-order* (*card-suc r*)
⟨*proof*⟩

**lemma** *card-order-card-suc*: *card-order r* $\Longrightarrow$ *card-order* (*card-suc r*)
⟨*proof*⟩

**lemma** *card-suc-greater*: *card-order r* $\Longrightarrow$ *r <o card-suc r*
⟨*proof*⟩

**lemma** *card-of-Plus-ordLess-infinite*:
**assumes** *INF*: $\neg$*finite C* **and** *LESS1*: |*A*| *<o* |*C*| **and** *LESS2*: |*B*| *<o* |*C*|
**shows** |*A* <+> *B*| *<o* |*C*|
⟨*proof*⟩

**lemma** *card-of-Plus-ordLess-infinite-Field*:
**assumes** *INF*: $\neg$*finite* (*Field r*) **and** *r*: *Card-order r* **and**
*LESS1*: |*A*| *<o r* **and** *LESS2*: |*B*| *<o r*
**shows** |*A* <+> *B*| *<o r*
⟨*proof*⟩

**lemma** *card-of-Plus-ordLeq-infinite-Field*:
**assumes** *r*: $\neg$*finite* (*Field r*) **and** *A*: |*A*| $\leq$*o r* **and** *B*: |*B*| $\leq$*o r*

**and** *c*: *Card-order r*
**shows** $|A <+> B| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Un-ordLeq-infinite-Field*:
**assumes** *C*: $\neg$*finite* (*Field r*) **and** *A*: $|A| \leq o\ r$ **and** *B*: $|B| \leq o\ r$
**and** *Card-order r*
**shows** $|A\ Un\ B| \leq o\ r$
⟨*proof*⟩

**lemma** *card-of-Un-ordLess-infinite*:
**assumes** *INF*: $\neg$*finite C* **and**
*LESS1*: $|A| <o\ |C|$ **and** *LESS2*: $|B| <o\ |C|$
**shows** $|A \cup B| <o\ |C|$
⟨*proof*⟩

**lemma** *card-of-Un-ordLess-infinite-Field*:
**assumes** *INF*: $\neg$*finite* (*Field r*) **and** *r*: *Card-order r* **and**
*LESS1*: $|A| <o\ r$ **and** *LESS2*: $|B| <o\ r$
**shows** $|A\ Un\ B| <o\ r$
⟨*proof*⟩

## 30.7 Regular cardinals

**definition** *cofinal* **where**
*cofinal A r* $\equiv \forall a \in$ *Field r*. $\exists b \in A.\ a \neq b \wedge (a,b) \in r$

**definition** *regularCard* **where**
*regularCard r* $\equiv \forall K.\ K \leq$ *Field r* $\wedge$ *cofinal K r* $\longrightarrow |K| =o\ r$

**definition** *relChain* **where**
*relChain r As* $\equiv \forall i\ j.\ (i,j) \in r \longrightarrow As\ i \leq As\ j$

**lemma** *regularCard-UNION*:
**assumes** *r*: *Card-order r* *regularCard r*
**and** *As*: *relChain r As*
**and** *Bsub*: $B \leq (\bigcup i \in$ *Field r*. *As i*)
**and** *cardB*: $|B| <o\ r$
**shows** $\exists i \in$ *Field r*. $B \leq As\ i$
⟨*proof*⟩

**lemma** *infinite-cardSuc-regularCard*:
**assumes** *r-inf*: $\neg$*finite* (*Field r*) **and** *r-card*: *Card-order r*
**shows** *regularCard* (*cardSuc r*)
⟨*proof*⟩

**lemma** *cardSuc-UNION*:
**assumes** *r*: *Card-order r* **and** $\neg$*finite* (*Field r*)
**and** *As*: *relChain* (*cardSuc r*) *As*

**and** *Bsub*: $B \leq (\bigcup\ i \in \mathit{Field}\ (\mathit{cardSuc}\ r).\ \mathit{As}\ i)$
**and** *cardB*: $|B| \leq o\ r$
**shows** $\exists\, i \in \mathit{Field}\ (\mathit{cardSuc}\ r).\ B \leq \mathit{As}\ i$
⟨*proof*⟩

## 30.8 Others

**lemma** *card-of-Func-Times*:
$|\mathit{Func}\ (A \times B)\ C| =o\ |\mathit{Func}\ A\ (\mathit{Func}\ B\ C)|$
⟨*proof*⟩

**lemma** *card-of-Pow-Func*:
$|\mathit{Pow}\ A| =o\ |\mathit{Func}\ A\ (\mathit{UNIV}::\mathit{bool}\ \mathit{set})|$
⟨*proof*⟩

**lemma** *card-of-Func-UNIV*:
$|\mathit{Func}\ (\mathit{UNIV}::'a\ \mathit{set})\ (B::'b\ \mathit{set})| =o\ |\{f::'a \Rightarrow 'b.\ \mathit{range}\ f \subseteq B\}|$
⟨*proof*⟩

**lemma** *Func-Times-Range*:
$|\mathit{Func}\ A\ (B \times C)| =o\ |\mathit{Func}\ A\ B \times \mathit{Func}\ A\ C|$ (**is** $|?LHS| =o\ |?RHS|$)
⟨*proof*⟩

## 30.9 Regular vs. stable cardinals

**definition** *stable* :: $'a\ \mathit{rel} \Rightarrow \mathit{bool}$
**where**
$\mathit{stable}\ r \equiv \forall\,(A::'a\ \mathit{set})\ (F :: 'a \Rightarrow 'a\ \mathit{set}).$
$|A| <o\ r \wedge (\forall\, a \in A.\ |F\ a| <o\ r)$
$\longrightarrow |\mathit{SIGMA}\ a : A.\ F\ a| <o\ r$

**lemma** *regularCard-stable*:
**assumes** *cr*: *Card-order r* **and** *ir*: $\neg$*finite* (*Field r*) **and** *reg*: *regularCard r*
**shows** *stable r*
⟨*proof*⟩

**lemma** *stable-regularCard*:
**assumes** *cr*: *Card-order r* **and** *ir*: $\neg$*finite* (*Field r*) **and** *st*: *stable r*
**shows** *regularCard r*
⟨*proof*⟩

**lemma** *internalize-card-of-ordLess*:
$(\ |A| <o\ r) = (\exists\, B < \mathit{Field}\ r.\ |A| =o\ |B| \wedge |B| <o\ r)$
⟨*proof*⟩

**lemma** *card-of-Sigma-cong1*:
**assumes** $\forall\, i \in I.\ |A\ i| =o\ |B\ i|$
**shows** $|\mathit{SIGMA}\ i : I.\ A\ i| =o\ |\mathit{SIGMA}\ i : I.\ B\ i|$
⟨*proof*⟩

**lemma** *card-of-Sigma-cong2*:
**assumes** *bij-betw f* $(I::'i\ set)$ $(J::'j\ set)$
**shows** $|SIGMA\ i : I.\ (A::'j \Rightarrow 'a\ set)\ (f\ i)| =o\ |SIGMA\ j : J.\ A\ j|$
⟨*proof*⟩

**lemma** *card-of-Sigma-cong*:
**assumes** *BIJ*: *bij-betw f I J* **and** *ISO*: $\forall j \in J.\ |A\ j| =o\ |B\ j|$
**shows** $|SIGMA\ i : I.\ A\ (f\ i)| =o\ |SIGMA\ j : J.\ B\ j|$
⟨*proof*⟩

**lemma** *stable-elim*:
**assumes** *ST*: *stable r* **and** *A-LESS*: $|A| <o\ r$ **and**
*F-LESS*: $\bigwedge a.\ a \in A \Longrightarrow |F\ a| <o\ r$
**shows** $|SIGMA\ a : A.\ F\ a| <o\ r$
⟨*proof*⟩

**lemma** *stable-natLeq*: *stable natLeq*
⟨*proof*⟩

**corollary** *regularCard-natLeq*: *regularCard natLeq*
⟨*proof*⟩

**lemma** *stable-ordIso1*:
**assumes** *ST*: *stable r* **and** *ISO*: $r' =o\ r$
**shows** *stable* $r'$
⟨*proof*⟩

**lemma** *stable-UNION*:
**assumes** *stable r* **and** $|A| <o\ r$ **and** $\bigwedge a.\ a \in A \Longrightarrow |F\ a| <o\ r$
**shows** $|\bigcup a \in A.\ F\ a| <o\ r$
⟨*proof*⟩

**corollary** *card-of-UNION-ordLess-infinite*:
**assumes** *stable* $|B|$ **and** $|I| <o\ |B|$ **and** $\forall i \in I.\ |A\ i| <o\ |B|$
**shows** $|\bigcup i \in I.\ A\ i| <o\ |B|$
⟨*proof*⟩

**corollary** *card-of-UNION-ordLess-infinite-Field*:
**assumes** *ST*: *stable r* **and** *r*: *Card-order r* **and**
*LEQ-I*: $|I| <o\ r$ **and** *LEQ*: $\forall i \in I.\ |A\ i| <o\ r$
**shows** $|\bigcup i \in I.\ A\ i| <o\ r$
⟨*proof*⟩

**end**

# 31 Cardinal Arithmetic as Needed by Bounded Natural Functors

**theory** *BNF-Cardinal-Arithmetic*
**imports** *BNF-Cardinal-Order-Relation*
**begin**

**lemma** *dir-image*: $[\![\bigwedge x\ y.\ (f\ x = f\ y) = (x = y);\ Card\text{-}order\ r]\!] \Longrightarrow r =o\ dir\text{-}image\ r\ f$
$\langle proof \rangle$

**lemma** *card-order-dir-image*:
**assumes** *bij*: *bij f* **and** *co*: *card-order r*
**shows** *card-order* (*dir-image r f*)
$\langle proof \rangle$

**lemma** *ordIso-refl*: $Card\text{-}order\ r \Longrightarrow r =o\ r$
$\langle proof \rangle$

**lemma** *ordLeq-refl*: $Card\text{-}order\ r \Longrightarrow r \le o\ r$
$\langle proof \rangle$

**lemma** *card-of-ordIso-subst*: $A = B \Longrightarrow |A| =o\ |B|$
$\langle proof \rangle$

**lemma** *Field-card-order*: $card\text{-}order\ r \Longrightarrow Field\ r = UNIV$
$\langle proof \rangle$

## 31.1 Zero

**definition** *czero* **where**
$czero = card\text{-}of\ \{\}$

**lemma** *czero-ordIso*: $czero =o\ czero$
$\langle proof \rangle$

**lemma** *card-of-ordIso-czero-iff-empty*:
$|A| =o\ (czero :: 'b\ rel) \longleftrightarrow A = (\{\} :: 'a\ set)$
$\langle proof \rangle$

**abbreviation** *Cnotzero* **where**
$Cnotzero\ (r :: 'a\ rel) \equiv \neg(r =o\ (czero :: 'a\ rel)) \wedge Card\text{-}order\ r$

**lemma** *Cnotzero-imp-not-empty*: $Cnotzero\ r \Longrightarrow Field\ r \neq \{\}$
$\langle proof \rangle$

**lemma** *czeroI*:

$[\![$*Card-order r*; *Field r* = {}$]\!] \Longrightarrow$ *r* =*o czero*
⟨*proof*⟩

**lemma** *czeroE*:
*r* =*o czero* $\Longrightarrow$ *Field r* = {}
⟨*proof*⟩

**lemma** *Cnotzero-mono*:
$[\![$*Cnotzero r*; *Card-order q*; *r* ≤*o q*$]\!] \Longrightarrow$ *Cnotzero q*
⟨*proof*⟩

## 31.2 (In)finite cardinals

**definition** *cinfinite* **where**
*cinfinite r* ≡ (¬ *finite* (*Field r*))

**abbreviation** *Cinfinite* **where**
*Cinfinite r* ≡ *cinfinite r* ∧ *Card-order r*

**definition** *cfinite* **where**
*cfinite r* = *finite* (*Field r*)

**abbreviation** *Cfinite* **where**
*Cfinite r* ≡ *cfinite r* ∧ *Card-order r*

**lemma** *Cfinite-ordLess-Cinfinite*: $[\![$*Cfinite r*; *Cinfinite s*$]\!] \Longrightarrow$ *r* <*o s*
⟨*proof*⟩

**lemmas** *natLeq-card-order* = *natLeq-Card-order*[*unfolded Field-natLeq*]

**lemma** *natLeq-cinfinite*: *cinfinite natLeq*
⟨*proof*⟩

**lemma** *natLeq-Cinfinite*: *Cinfinite natLeq*
⟨*proof*⟩

**lemma** *natLeq-ordLeq-cinfinite*:
**assumes** *inf*: *Cinfinite r*
**shows** *natLeq* ≤*o r*
⟨*proof*⟩

**lemma** *cinfinite-not-czero*: *cinfinite r* $\Longrightarrow$ ¬ (*r* =*o* (*czero* :: *'a rel*))
⟨*proof*⟩

**lemma** *Cinfinite-Cnotzero*: *Cinfinite r* $\Longrightarrow$ *Cnotzero r*
⟨*proof*⟩

**lemma** *Cinfinite-cong*: $[\![$*r1* =*o r2*; *Cinfinite r1*$]\!] \Longrightarrow$ *Cinfinite r2*
⟨*proof*⟩

**lemma** *cinfinite-mono*: $[\![r1 \leq o\ r2;\ cinfinite\ r1]\!] \Longrightarrow cinfinite\ r2$
⟨*proof*⟩

**lemma** *regularCard-ordIso*:
**assumes** $k =o\ k'$ **and** *Cinfinite k* **and** *regularCard k*
**shows** *regularCard k'*
⟨*proof*⟩

**corollary** *card-of-UNION-ordLess-infinite-Field-regularCard*:
**assumes** *regularCard r* **and** *Cinfinite r* **and** $|I| <o\ r$ **and** $\forall i \in I.\ |A\ i| <o\ r$
**shows** $|\bigcup i \in I.\ A\ i| <o\ r$
⟨*proof*⟩

## 31.3 Binary sum

**definition** *csum* (**infixr** ‹+c› *65*)
**where** $r1 +c\ r2 \equiv |Field\ r1 <+> Field\ r2|$

**lemma** *Field-csum*: $Field\ (r +c\ s) = Inl\ `\ Field\ r \cup Inr\ `\ Field\ s$
⟨*proof*⟩

**lemma** *Card-order-csum*: $Card\text{-}order\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *csum-Cnotzero1*: $Cnotzero\ r1 \Longrightarrow Cnotzero\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *card-order-csum*:
**assumes** *card-order r1 card-order r2*
**shows** $card\text{-}order\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *cinfinite-csum*:
$cinfinite\ r1 \vee cinfinite\ r2 \Longrightarrow cinfinite\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *Cinfinite-csum*:
$Cinfinite\ r1 \vee Cinfinite\ r2 \Longrightarrow Cinfinite\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *Cinfinite-csum1*: $Cinfinite\ r1 \Longrightarrow Cinfinite\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *Cinfinite-csum-weak*:
$[\![Cinfinite\ r1;\ Cinfinite\ r2]\!] \Longrightarrow Cinfinite\ (r1 +c\ r2)$
⟨*proof*⟩

**lemma** *csum-cong*: $[\![p1 =o\ r1;\ p2 =o\ r2]\!] \Longrightarrow p1 +c\ p2 =o\ r1 +c\ r2$

⟨proof⟩

**lemma** *csum-cong1*: $p1 =o\ r1 \Longrightarrow p1 +c\ q =o\ r1 +c\ q$
⟨proof⟩

**lemma** *csum-cong2*: $p2 =o\ r2 \Longrightarrow q +c\ p2 =o\ q +c\ r2$
⟨proof⟩

**lemma** *csum-mono*: $[\![p1 \le o\ r1;\ p2 \le o\ r2]\!] \Longrightarrow p1 +c\ p2 \le o\ r1 +c\ r2$
⟨proof⟩

**lemma** *csum-mono1*: $p1 \le o\ r1 \Longrightarrow p1 +c\ q \le o\ r1 +c\ q$
⟨proof⟩

**lemma** *csum-mono2*: $p2 \le o\ r2 \Longrightarrow q +c\ p2 \le o\ q +c\ r2$
⟨proof⟩

**lemma** *ordLeq-csum1*: *Card-order* $p1 \Longrightarrow p1 \le o\ p1 +c\ p2$
⟨proof⟩

**lemma** *ordLeq-csum2*: *Card-order* $p2 \Longrightarrow p2 \le o\ p1 +c\ p2$
⟨proof⟩

**lemma** *csum-com*: $p1 +c\ p2 =o\ p2 +c\ p1$
⟨proof⟩

**lemma** *csum-assoc*: $(p1 +c\ p2) +c\ p3 =o\ p1 +c\ p2 +c\ p3$
⟨proof⟩

**lemma** *Cfinite-csum*: $[\![$*Cfinite* $r$; *Cfinite* $s]\!] \Longrightarrow$ *Cfinite* $(r +c\ s)$
⟨proof⟩

**lemma** *csum-csum*: $(r1 +c\ r2) +c\ (r3 +c\ r4) =o\ (r1 +c\ r3) +c\ (r2 +c\ r4)$
⟨proof⟩

**lemma** *Plus-csum*: $|A <+> B| =o\ |A| +c\ |B|$
⟨proof⟩

**lemma** *Un-csum*: $|A \cup B| \le o\ |A| +c\ |B|$
⟨proof⟩

## 31.4 One

**definition** *cone* **where**
*cone* = *card-of* $\{()\}$

**lemma** *Card-order-cone*: *Card-order cone*
⟨proof⟩

**lemma** *Cfinite-cone*: *Cfinite cone*
⟨*proof*⟩

**lemma** *cone-not-czero*: $\neg$ (*cone =o czero*)
⟨*proof*⟩

**lemma** *cone-ordLeq-Cnotzero*: *Cnotzero r* $\Longrightarrow$ *cone* $\leq$*o r*
⟨*proof*⟩

## 31.5 Two

**definition** *ctwo* **where**
*ctwo* = |*UNIV* :: *bool set*|

**lemma** *Card-order-ctwo*: *Card-order ctwo*
⟨*proof*⟩

**lemma** *ctwo-not-czero*: $\neg$ (*ctwo =o czero*)
⟨*proof*⟩

**lemma** *ctwo-Cnotzero*: *Cnotzero ctwo*
⟨*proof*⟩

## 31.6 Family sum

**definition** *Csum* **where**
*Csum r rs* ≡ |*SIGMA i* : *Field r*. *Field* (*rs i*)|

**syntax** *-Csum* ::
*pttrn* => ('*a* * '*a*) *set* => '*b* * '*b set* => (('*a* * '*b*) * ('*a* * '*b*)) *set*
(‹(‹indent=3 notation=‹binder CSUM››CSUM -:-. -)› [0, 51, 10] 10)

**syntax-consts**
*-Csum* == *Csum*

**translations**
*CSUM i*:*r*. *rs* == *CONST Csum r* (%*i*. *rs*)

**lemma** *SIGMA-CSUM*: |*SIGMA i* : *I*. *As i*| = (*CSUM i* : |*I*|. |*As i*| )
⟨*proof*⟩

## 31.7 Product

**definition** *cprod* (**infixr** ‹*c› *80*) **where**
*r1* *c *r2* = |*Field r1* × *Field r2*|

**lemma** *card-order-cprod*:
**assumes** *card-order r1 card-order r2*
**shows** *card-order* (*r1* *c *r2*)

⟨*proof*⟩

**lemma** *Card-order-cprod*: $Card\text{-}order\ (r1\ {*}c\ r2)$
⟨*proof*⟩

**lemma** *cprod-mono1*: $p1 \le o\ r1 \Longrightarrow p1\ {*}c\ q \le o\ r1\ {*}c\ q$
⟨*proof*⟩

**lemma** *cprod-mono2*: $p2 \le o\ r2 \Longrightarrow q\ {*}c\ p2 \le o\ q\ {*}c\ r2$
⟨*proof*⟩

**lemma** *cprod-mono*: $[\![p1 \le o\ r1;\ p2 \le o\ r2]\!] \Longrightarrow p1\ {*}c\ p2 \le o\ r1\ {*}c\ r2$
⟨*proof*⟩

**lemma** *ordLeq-cprod2*: $[\![Cnotzero\ p1;\ Card\text{-}order\ p2]\!] \Longrightarrow p2 \le o\ p1\ {*}c\ p2$
⟨*proof*⟩

**lemma** *cinfinite-cprod*: $[\![cinfinite\ r1;\ cinfinite\ r2]\!] \Longrightarrow cinfinite\ (r1\ {*}c\ r2)$
⟨*proof*⟩

**lemma** *cinfinite-cprod2*: $[\![Cnotzero\ r1;\ Cinfinite\ r2]\!] \Longrightarrow cinfinite\ (r1\ {*}c\ r2)$
⟨*proof*⟩

**lemma** *Cinfinite-cprod2*: $[\![Cnotzero\ r1;\ Cinfinite\ r2]\!] \Longrightarrow Cinfinite\ (r1\ {*}c\ r2)$
⟨*proof*⟩

**lemma** *cprod-cong*: $[\![p1 =o\ r1;\ p2 =o\ r2]\!] \Longrightarrow p1\ {*}c\ p2 =o\ r1\ {*}c\ r2$
⟨*proof*⟩

**lemma** *cprod-cong1*: $[\![p1 =o\ r1]\!] \Longrightarrow p1\ {*}c\ p2 =o\ r1\ {*}c\ p2$
⟨*proof*⟩

**lemma** *cprod-cong2*: $p2 =o\ r2 \Longrightarrow q\ {*}c\ p2 =o\ q\ {*}c\ r2$
⟨*proof*⟩

**lemma** *cprod-com*: $p1\ {*}c\ p2 =o\ p2\ {*}c\ p1$
⟨*proof*⟩

**lemma** *card-of-Csum-Times*:
$\forall i \in I.\ |A\ i| \le o\ |B| \Longrightarrow (CSUM\ i : |I|.\ |A\ i|\ ) \le o\ |I|\ {*}c\ |B|$
⟨*proof*⟩

**lemma** *card-of-Csum-Times′*:
**assumes** $Card\text{-}order\ r\ \forall i \in I.\ |A\ i| \le o\ r$
**shows** $(CSUM\ i : |I|.\ |A\ i|\ ) \le o\ |I|\ {*}c\ r$
⟨*proof*⟩

**lemma** *cprod-csum-distrib1*: $r1\ {*}c\ r2\ {+}c\ r1\ {*}c\ r3 =o\ r1\ {*}c\ (r2\ {+}c\ r3)$
⟨*proof*⟩

**lemma** *csum-absorb2'*: $[\![Card\text{-}order\ r2;\ r1 \leq o\ r2;\ cinfinite\ r1 \lor cinfinite\ r2]\!] \Longrightarrow r1 +c\ r2 =o\ r2$
⟨*proof*⟩

**lemma** *csum-absorb1'*:
**assumes** *card*: *Card-order r2*
**and** *r12*: $r1 \leq o\ r2$ **and** *cr12*: $cinfinite\ r1 \lor cinfinite\ r2$
**shows** $r2 +c\ r1 =o\ r2$
⟨*proof*⟩

**lemma** *csum-absorb1*: $[\![Cinfinite\ r2;\ r1 \leq o\ r2]\!] \Longrightarrow r2 +c\ r1 =o\ r2$
⟨*proof*⟩

**lemma** *csum-absorb2*: $[\![Cinfinite\ r2\ ;\ r1 \leq o\ r2]\!] \Longrightarrow r1 +c\ r2 =o\ r2$
⟨*proof*⟩

**lemma** *regularCard-csum*:
**assumes** *Cinfinite r Cinfinite s regularCard r regularCard s*
**shows** *regularCard* $(r +c\ s)$
⟨*proof*⟩

**lemma** *csum-mono-strict*:
**assumes** *Card-order*: *Card-order r Card-order q*
**and** *Cinfinite*: *Cinfinite* $r'$ *Cinfinite* $q'$
**and** *less*: $r <o\ r'\ q <o\ q'$
**shows** $r +c\ q <o\ r' +c\ q'$
⟨*proof*⟩

## 31.8 Exponentiation

**definition** *cexp* (**infixr** ‹^c› *90*) **where**
$r1\ \hat{}c\ r2 \equiv |Func\ (Field\ r2)\ (Field\ r1)|$

**lemma** *Card-order-cexp*: *Card-order* $(r1\ \hat{}c\ r2)$
⟨*proof*⟩

**lemma** *cexp-mono'*:
**assumes** *1*: $p1 \leq o\ r1$ **and** *2*: $p2 \leq o\ r2$
**and** *n*: $Field\ p2 = \{\} \Longrightarrow Field\ r2 = \{\}$
**shows** $p1\ \hat{}c\ p2 \leq o\ r1\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-mono*:
**assumes** *1*: $p1 \leq o\ r1$ **and** *2*: $p2 \leq o\ r2$
**and** *n*: $p2 =o\ czero \Longrightarrow r2 =o\ czero$ **and** *card*: *Card-order p2*
**shows** $p1\ \hat{}c\ p2 \leq o\ r1\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-mono1*:
**assumes** *1*: $p1 \leq o\ r1$ **and** *q*: *Card-order q*
**shows** $p1\ \hat{}c\ q \leq o\ r1\ \hat{}c\ q$
⟨*proof*⟩

**lemma** *cexp-mono2'*:
**assumes** *2*: $p2 \leq o\ r2$ **and** *q*: *Card-order q*
**and** *n*: $Field\ p2 = \{\} \Longrightarrow Field\ r2 = \{\}$
**shows** $q\ \hat{}c\ p2 \leq o\ q\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-mono2*:
**assumes** *2*: $p2 \leq o\ r2$ **and** *q*: *Card-order q*
**and** *n*: $p2 =o\ czero \Longrightarrow r2 =o\ czero$ **and** *card*: *Card-order p2*
**shows** $q\ \hat{}c\ p2 \leq o\ q\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-mono2-Cnotzero*:
**assumes** $p2 \leq o\ r2$ *Card-order q Cnotzero p2*
**shows** $q\ \hat{}c\ p2 \leq o\ q\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-cong*:
**assumes** *1*: $p1 =o\ r1$ **and** *2*: $p2 =o\ r2$
**and** *Cr*: *Card-order r2*
**and** *Cp*: *Card-order p2*
**shows** $p1\ \hat{}c\ p2 =o\ r1\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-cong1*:
**assumes** *1*: $p1 =o\ r1$ **and** *q*: *Card-order q*
**shows** $p1\ \hat{}c\ q =o\ r1\ \hat{}c\ q$
⟨*proof*⟩

**lemma** *cexp-cong2*:
**assumes** *2*: $p2 =o\ r2$ **and** *q*: *Card-order q* **and** *p*: *Card-order p2*
**shows** $q\ \hat{}c\ p2 =o\ q\ \hat{}c\ r2$
⟨*proof*⟩

**lemma** *cexp-cone*:
**assumes** *Card-order r*
**shows** $r\ \hat{}c\ cone =o\ r$
⟨*proof*⟩

**lemma** *cexp-cprod*:
**assumes** *r1*: *Card-order r1*
**shows** $(r1\ \hat{}c\ r2)\ \hat{}c\ r3 =o\ r1\ \hat{}c\ (r2 *c\ r3)$ (**is** $?L =o\ ?R$)
⟨*proof*⟩

**lemma** *cprod-infinite1'*: $[\![Cinfinite\ r;\ Cnotzero\ p;\ p \leq o\ r]\!] \Longrightarrow r *c\ p =o\ r$
⟨*proof*⟩

**lemma** *cprod-infinite*: $Cinfinite\ r \Longrightarrow r *c\ r =o\ r$
⟨*proof*⟩

**lemma** *cexp-cprod-ordLeq*:
**assumes** *r1*: *Card-order r1* **and** *r2*: *Cinfinite r2*
**and** *r3*: *Cnotzero r3 r3* $\leq o$ *r2*
**shows** $(r1\ \hat{}c\ r2)\ \hat{}c\ r3 =o\ r1\ \hat{}c\ r2$ (**is** *?L =o ?R*)
⟨*proof*⟩

**lemma** *Cnotzero-UNIV*: *Cnotzero* $|UNIV|$
⟨*proof*⟩

**lemma** *ordLess-ctwo-cexp*:
**assumes** *Card-order r*
**shows** $r <o\ ctwo\ \hat{}c\ r$
⟨*proof*⟩

**lemma** *ordLeq-cexp1*:
**assumes** *Cnotzero r Card-order q*
**shows** $q \leq o\ q\ \hat{}c\ r$
⟨*proof*⟩

**lemma** *ordLeq-cexp2*:
**assumes** $ctwo \leq o\ q$ *Card-order r*
**shows** $r \leq o\ q\ \hat{}c\ r$
⟨*proof*⟩

**lemma** *cinfinite-cexp*: $[\![ctwo \leq o\ q;\ Cinfinite\ r]\!] \Longrightarrow cinfinite\ (q\ \hat{}c\ r)$
⟨*proof*⟩

**lemma** *Cinfinite-cexp*:
$[\![ctwo \leq o\ q;\ Cinfinite\ r]\!] \Longrightarrow Cinfinite\ (q\ \hat{}c\ r)$
⟨*proof*⟩

**lemma** *card-order-cexp*:
**assumes** *card-order r1 card-order r2*
**shows** *card-order* $(r1\ \hat{}c\ r2)$
⟨*proof*⟩

**lemma** *ctwo-ordLess-natLeq*: $ctwo <o\ natLeq$
⟨*proof*⟩

**lemma** *ctwo-ordLess-Cinfinite*: $Cinfinite\ r \Longrightarrow ctwo <o\ r$
⟨*proof*⟩

**lemma** *ctwo-ordLeq-Cinfinite*:

**assumes** *Cinfinite r*
**shows** $ctwo \le o\ r$
⟨*proof*⟩

**lemma** *Un-Cinfinite-bound*: $[\![|A| \le o\ r;\ |B| \le o\ r;\ Cinfinite\ r]\!] \Longrightarrow |A \cup B| \le o\ r$
⟨*proof*⟩

**lemma** *Un-Cinfinite-bound-strict*: $[\![|A| < o\ r;\ |B| < o\ r;\ Cinfinite\ r]\!] \Longrightarrow |A \cup B| < o\ r$
⟨*proof*⟩

**lemma** *UNION-Cinfinite-bound*: $[\![|I| \le o\ r;\ \forall\, i \in I.\ |A\ i| \le o\ r;\ Cinfinite\ r]\!] \Longrightarrow |\bigcup i \in I.\ A\ i| \le o\ r$
⟨*proof*⟩

**lemma** *csum-cinfinite-bound*:
**assumes** $p \le o\ r\ q \le o\ r$ *Card-order p Card-order q Cinfinite r*
**shows** $p +c\ q \le o\ r$
⟨*proof*⟩

**lemma** *cprod-cinfinite-bound*:
**assumes** $p \le o\ r\ q \le o\ r$ *Card-order p Card-order q Cinfinite r*
**shows** $p *c\ q \le o\ r$
⟨*proof*⟩

**lemma** *cprod-infinite2'*: $[\![Cnotzero\ r1;\ Cinfinite\ r2;\ r1 \le o\ r2]\!] \Longrightarrow r1 *c\ r2 =o\ r2$
⟨*proof*⟩

**lemma** *regularCard-cprod*:
**assumes** *Cinfinite r Cinfinite s regularCard r regularCard s*
**shows** *regularCard* $(r *c\ s)$
⟨*proof*⟩

**lemma** *cprod-csum-cexp*:
$r1 *c\ r2 \le o\ (r1 +c\ r2)\ \hat{}c\ ctwo$
⟨*proof*⟩

**lemma** *Cfinite-cprod-Cinfinite*: $[\![Cfinite\ r;\ Cinfinite\ s]\!] \Longrightarrow r *c\ s \le o\ s$
⟨*proof*⟩

**lemma** *cprod-cexp*: $(r *c\ s)\ \hat{}c\ t =o\ r\ \hat{}c\ t *c\ s\ \hat{}c\ t$
⟨*proof*⟩

**lemma** *cprod-cexp-csum-cexp-Cinfinite*:
**assumes** *t*: *Cinfinite t*
**shows** $(r *c\ s)\ \hat{}c\ t \le o\ (r +c\ s)\ \hat{}c\ t$
⟨*proof*⟩

**lemma** *Cfinite-cexp-Cinfinite*:
**assumes** *s*: *Cfinite s* **and** *t*: *Cinfinite t*
**shows** $s \mathbin{\hat{}c} t \leq o\ ctwo \mathbin{\hat{}c} t$
⟨*proof*⟩

**lemma** *csum-Cfinite-cexp-Cinfinite*:
**assumes** *r*: *Card-order r* **and** *s*: *Cfinite s* **and** *t*: *Cinfinite t*
**shows** $(r +c\ s) \mathbin{\hat{}c} t \leq o\ (r +c\ ctwo) \mathbin{\hat{}c} t$
⟨*proof*⟩

**lemma** *Cinfinite-cardSuc*: $Cinfinite\ r \Longrightarrow Cinfinite\ (cardSuc\ r)$
⟨*proof*⟩

**lemma** *cardSuc-UNION-Cinfinite*:
**assumes** $Cinfinite\ r\ relChain\ (cardSuc\ r)\ As\ B \leq (\bigcup i \in Field\ (cardSuc\ r).\ As\ i)\ |B| <= o\ r$
**shows** $\exists\, i \in Field\ (cardSuc\ r).\ B \leq As\ i$
⟨*proof*⟩

**lemma** *Cinfinite-card-suc*: $[\![\ Cinfinite\ r\ ;\ card\text{-}order\ r\ ]\!] \Longrightarrow Cinfinite\ (card\text{-}suc\ r)$
⟨*proof*⟩

**lemma** *card-suc-least*: $[\![card\text{-}order\ r;\ Card\text{-}order\ s;\ r <o\ s]\!] \Longrightarrow card\text{-}suc\ r \leq o\ s$
⟨*proof*⟩

**lemma** *regularCard-cardSuc*: $Cinfinite\ k \Longrightarrow regularCard\ (cardSuc\ k)$
⟨*proof*⟩

**lemma** *regularCard-card-suc*: $card\text{-}order\ r \Longrightarrow Cinfinite\ r \Longrightarrow regularCard\ (card\text{-}suc\ r)$
⟨*proof*⟩

**end**

# 32 Function Definition Base

**theory** *Fun-Def-Base*
**imports** *Ctr-Sugar Set Wellfounded*
**begin**

⟨*ML*⟩
**named-theorems** *termination-simp simplification rules for termination proofs*
⟨*ML*⟩

**end**

# 33 Definition of Bounded Natural Functors

**theory** *BNF-Def*
**imports** *BNF-Cardinal-Arithmetic Fun-Def-Base*
**keywords**
*print-bnfs* :: *diag* **and**
*bnf* :: *thy-goal-defn*
**begin**

**lemma** *Collect-case-prodD*: $x \in Collect\ (case\text{-}prod\ A) \Longrightarrow A\ (fst\ x)\ (snd\ x)$
$\langle proof \rangle$

**inductive**
$rel\text{-}sum :: ('a \Rightarrow 'c \Rightarrow bool) \Rightarrow ('b \Rightarrow 'd \Rightarrow bool) \Rightarrow 'a + 'b \Rightarrow 'c + 'd \Rightarrow bool$
**for** *R1 R2*
**where**
$R1\ a\ c \Longrightarrow rel\text{-}sum\ R1\ R2\ (Inl\ a)\ (Inl\ c)$
$|\ R2\ b\ d \Longrightarrow rel\text{-}sum\ R1\ R2\ (Inr\ b)\ (Inr\ d)$

**definition**
$rel\text{-}fun :: ('a \Rightarrow 'c \Rightarrow bool) \Rightarrow ('b \Rightarrow 'd \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'd) \Rightarrow bool$
**where**
$rel\text{-}fun\ A\ B = (\lambda f\ g.\ \forall\, x\ y.\ A\ x\ y \longrightarrow B\ (f\ x)\ (g\ y))$

**lemma** *rel-funI* [*intro*]:
**assumes** $\bigwedge x\ y.\ A\ x\ y \Longrightarrow B\ (f\ x)\ (g\ y)$
**shows** $rel\text{-}fun\ A\ B\ f\ g$
$\langle proof \rangle$

**lemma** *rel-funD*:
**assumes** $rel\text{-}fun\ A\ B\ f\ g$ **and** $A\ x\ y$
**shows** $B\ (f\ x)\ (g\ y)$
$\langle proof \rangle$

**lemma** *rel-fun-mono*:
$[\![\ rel\text{-}fun\ X\ A\ f\ g;\ \bigwedge x\ y.\ Y\ x\ y \longrightarrow X\ x\ y;\ \bigwedge x\ y.\ A\ x\ y \Longrightarrow B\ x\ y\ ]\!] \Longrightarrow rel\text{-}fun\ Y\ B\ f\ g$
$\langle proof \rangle$

**lemma** *rel-fun-mono'* [*mono*]:
$[\![\ \bigwedge x\ y.\ Y\ x\ y \longrightarrow X\ x\ y;\ \bigwedge x\ y.\ A\ x\ y \longrightarrow B\ x\ y\ ]\!] \Longrightarrow rel\text{-}fun\ X\ A\ f\ g \longrightarrow rel\text{-}fun\ Y\ B\ f\ g$
$\langle proof \rangle$

**definition** $rel\text{-}set :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow bool$
**where** $rel\text{-}set\ R = (\lambda A\ B.\ (\forall\, x{\in}A.\ \exists\, y{\in}B.\ R\ x\ y) \land (\forall\, y{\in}B.\ \exists\, x{\in}A.\ R\ x\ y))$

**lemma** *rel-setI*:
**assumes** $\bigwedge x.\ x \in A \Longrightarrow \exists\, y{\in}B.\ R\ x\ y$

**assumes** $\bigwedge y.\ y \in B \Longrightarrow \exists x{\in}A.\ R\ x\ y$
**shows** *rel-set* $R\ A\ B$
⟨*proof*⟩

**lemma** *predicate2-transferD*:
$[\![\textit{rel-fun}\ R1\ (\textit{rel-fun}\ R2\ (=))\ P\ Q;\ a \in A;\ b \in B;\ A \subseteq \{(x, y).\ R1\ x\ y\};\ B \subseteq \{(x, y).\ R2\ x\ y\}]\!] \Longrightarrow$
$P\ (\textit{fst}\ a)\ (\textit{fst}\ b) \longleftrightarrow Q\ (\textit{snd}\ a)\ (\textit{snd}\ b)$
⟨*proof*⟩

**definition** *collect* **where**
$\textit{collect}\ F\ x = (\bigcup f \in F.\ f\ x)$

**lemma** *fstI*: $x = (y, z) \Longrightarrow \textit{fst}\ x = y$
⟨*proof*⟩

**lemma** *sndI*: $x = (y, z) \Longrightarrow \textit{snd}\ x = z$
⟨*proof*⟩

**lemma** *bijI'*: $[\![\bigwedge x\ y.\ (f\ x = f\ y) = (x = y);\ \bigwedge y.\ \exists x.\ y = f\ x]\!] \Longrightarrow \textit{bij}\ f$
⟨*proof*⟩

**definition** $\textit{Gr}\ A\ f = \{(a, f\ a) \mid a.\ a \in A\}$

**definition** $\textit{Grp}\ A\ f = (\lambda a\ b.\ b = f\ a \wedge a \in A)$

**definition** *vimage2p* **where**
$\textit{vimage2p}\ f\ g\ R = (\lambda x\ y.\ R\ (f\ x)\ (g\ y))$

**lemma** *collect-comp*: $\textit{collect}\ F \circ g = \textit{collect}\ ((\lambda f.\ f \circ g)\ `\ F)$
⟨*proof*⟩

**definition** *convol* (‹(‹*indent=1 notation=*‹*mixfix convol*››⟨-,/ -⟩)›) **where**
$\langle f, g\rangle \equiv \lambda a.\ (f\ a, g\ a)$

**lemma** *fst-convol*: $\textit{fst} \circ \langle f, g\rangle = f$
⟨*proof*⟩

**lemma** *snd-convol*: $\textit{snd} \circ \langle f, g\rangle = g$
⟨*proof*⟩

**lemma** *convol-mem-GrpI*:
$x \in A \Longrightarrow \langle \textit{id}, g\rangle\ x \in (\textit{Collect}\ (\textit{case-prod}\ (\textit{Grp}\ A\ g)))$
⟨*proof*⟩

**definition** *csquare* **where**
$\textit{csquare}\ A\ f1\ f2\ p1\ p2 \longleftrightarrow (\forall\ a \in A.\ f1\ (p1\ a) = f2\ (p2\ a))$

**lemma** *eq-alt*: $(=) = Grp\ UNIV\ id$
⟨*proof*⟩

**lemma** *leq-conversepI*: $R = (=) \Longrightarrow R \leq R^{-1-1}$
⟨*proof*⟩

**lemma** *leq-OOI*: $R = (=) \Longrightarrow R \leq R\ OO\ R$
⟨*proof*⟩

**lemma** *OO-Grp-alt*: $(Grp\ A\ f)^{-1-1}\ OO\ Grp\ A\ g = (\lambda x\ y.\ \exists z.\ z \in A \wedge f\ z = x \wedge g\ z = y)$
⟨*proof*⟩

**lemma** *Grp-UNIV-id*: $f = id \Longrightarrow (Grp\ UNIV\ f)^{-1-1}\ OO\ Grp\ UNIV\ f = Grp\ UNIV\ f$
⟨*proof*⟩

**lemma** *Grp-UNIV-idI*: $x = y \Longrightarrow Grp\ UNIV\ id\ x\ y$
⟨*proof*⟩

**lemma** *Grp-mono*: $A \leq B \Longrightarrow Grp\ A\ f \leq Grp\ B\ f$
⟨*proof*⟩

**lemma** *GrpI*: $[\![f\ x = y;\ x \in A]\!] \Longrightarrow Grp\ A\ f\ x\ y$
⟨*proof*⟩

**lemma** *GrpE*: $Grp\ A\ f\ x\ y \Longrightarrow ([\![f\ x = y;\ x \in A]\!] \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *Collect-case-prod-Grp-eqD*: $z \in Collect\ (case\text{-}prod\ (Grp\ A\ f)) \Longrightarrow (f \circ fst)\ z = snd\ z$
⟨*proof*⟩

**lemma** *Collect-case-prod-Grp-in*: $z \in Collect\ (case\text{-}prod\ (Grp\ A\ f)) \Longrightarrow fst\ z \in A$
⟨*proof*⟩

**definition** *pick-middlep* $P\ Q\ a\ c = (SOME\ b.\ P\ a\ b \wedge Q\ b\ c)$

**lemma** *pick-middlep*:
$(P\ OO\ Q)\ a\ c \Longrightarrow P\ a\ (pick\text{-}middlep\ P\ Q\ a\ c) \wedge Q\ (pick\text{-}middlep\ P\ Q\ a\ c)\ c$
⟨*proof*⟩

**definition** *fstOp* **where**
$fstOp\ P\ Q\ ac = (fst\ ac,\ pick\text{-}middlep\ P\ Q\ (fst\ ac)\ (snd\ ac))$

**definition** *sndOp* **where**
$sndOp\ P\ Q\ ac = (pick\text{-}middlep\ P\ Q\ (fst\ ac)\ (snd\ ac),\ (snd\ ac))$

**lemma** *fstOp-in*: $ac \in Collect\ (case\text{-}prod\ (P\ OO\ Q)) \Longrightarrow fstOp\ P\ Q\ ac \in Collect$

$(case\text{-}prod\ P)$
⟨proof⟩

**lemma** *fst-fstOp*: $fst\ bc = (fst \circ fstOp\ P\ Q)\ bc$
⟨proof⟩

**lemma** *snd-sndOp*: $snd\ bc = (snd \circ sndOp\ P\ Q)\ bc$
⟨proof⟩

**lemma** *sndOp-in*: $ac \in Collect\ (case\text{-}prod\ (P\ OO\ Q)) \Longrightarrow sndOp\ P\ Q\ ac \in Collect\ (case\text{-}prod\ Q)$
⟨proof⟩

**lemma** *csquare-fstOp-sndOp*:
$csquare\ (Collect\ (f\ (P\ OO\ Q)))\ snd\ fst\ (fstOp\ P\ Q)\ (sndOp\ P\ Q)$
⟨proof⟩

**lemma** *snd-fst-flip*: $snd\ xy = (fst \circ (\%(x, y). (y, x)))\ xy$
⟨proof⟩

**lemma** *fst-snd-flip*: $fst\ xy = (snd \circ (\%(x, y). (y, x)))\ xy$
⟨proof⟩

**lemma** *flip-pred*: $A \subseteq Collect\ (case\text{-}prod\ (R\ ^{-1-1})) \Longrightarrow (\%(x, y). (y, x))\ `\ A \subseteq Collect\ (case\text{-}prod\ R)$
⟨proof⟩

**lemma** *predicate2-eqD*: $A = B \Longrightarrow A\ a\ b \longleftrightarrow B\ a\ b$
⟨proof⟩

**lemma** *case-sum-o-inj*: $case\text{-}sum\ f\ g \circ Inl = f\ case\text{-}sum\ f\ g \circ Inr = g$
⟨proof⟩

**lemma** *map-sum-o-inj*: $map\text{-}sum\ f\ g \circ Inl = Inl \circ f\ map\text{-}sum\ f\ g \circ Inr = Inr \circ g$
⟨proof⟩

**lemma** *card-order-csum-cone-cexp-def*:
$card\text{-}order\ r \Longrightarrow (\ |A1|\ +c\ cone)\ \hat{}c\ r = |Func\ UNIV\ (Inl\ `\ A1 \cup \{Inr\ ()\})|$
⟨proof⟩

**lemma** *If-the-inv-into-in-Func*:
$[\![inj\text{-}on\ g\ C;\ C \subseteq B \cup \{x\}]\!] \Longrightarrow$
$(\lambda i.\ if\ i \in g\ `\ C\ then\ the\text{-}inv\text{-}into\ C\ g\ i\ else\ x) \in Func\ UNIV\ (B \cup \{x\})$
⟨proof⟩

**lemma** *If-the-inv-into-f-f*:
$[\![i \in C;\ inj\text{-}on\ g\ C]\!] \Longrightarrow ((\lambda i.\ if\ i \in g\ `\ C\ then\ the\text{-}inv\text{-}into\ C\ g\ i\ else\ x) \circ g)\ i = id\ i$
⟨proof⟩

**lemma** *the-inv-f-o-f-id*: $inj\ f \Longrightarrow (the\text{-}inv\ f \circ f)\ z = id\ z$
⟨*proof*⟩

**lemma** *vimage2pI*: $R\ (f\ x)\ (g\ y) \Longrightarrow vimage2p\ f\ g\ R\ x\ y$
⟨*proof*⟩

**lemma** *rel-fun-iff-leq-vimage2p*: $(rel\text{-}fun\ R\ S)\ f\ g = (R \leq vimage2p\ f\ g\ S)$
⟨*proof*⟩

**lemma** *convol-image-vimage2p*: $\langle f \circ fst,\ g \circ snd\rangle\ `\ Collect\ (case\text{-}prod\ (vimage2p\ f\ g\ R)) \subseteq Collect\ (case\text{-}prod\ R)$
⟨*proof*⟩

**lemma** *vimage2p-Grp*: $vimage2p\ f\ g\ P = Grp\ UNIV\ f\ OO\ P\ OO\ (Grp\ UNIV\ g)^{-1-1}$
⟨*proof*⟩

**lemma** *subst-Pair*: $P\ x\ y \Longrightarrow a = (x,\ y) \Longrightarrow P\ (fst\ a)\ (snd\ a)$
⟨*proof*⟩

**lemma** *comp-apply-eq*: $f\ (g\ x) = h\ (k\ x) \Longrightarrow (f \circ g)\ x = (h \circ k)\ x$
⟨*proof*⟩

**lemma** *refl-ge-eq*: $(\bigwedge x.\ R\ x\ x) \Longrightarrow (=) \leq R$
⟨*proof*⟩

**lemma** *ge-eq-refl*: $(=) \leq R \Longrightarrow R\ x\ x$
⟨*proof*⟩

**lemma** *reflp-eq*: $reflp\ R = ((=) \leq R)$
⟨*proof*⟩

**lemma** *transp-relcompp*: $transp\ r \longleftrightarrow r\ OO\ r \leq r$
⟨*proof*⟩

**lemma** *symp-conversep*: $symp\ R = (R^{-1-1} \leq R)$
⟨*proof*⟩

**lemma** *diag-imp-eq-le*: $(\bigwedge x.\ x \in A \Longrightarrow R\ x\ x) \Longrightarrow \forall\, x\ y.\ x \in A \longrightarrow y \in A \longrightarrow x = y \longrightarrow R\ x\ y$
⟨*proof*⟩

**definition** *eq-onp* :: $('a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
**where** $eq\text{-}onp\ R = (\lambda x\ y.\ R\ x \wedge x = y)$

**lemma** *eq-onp-Grp*: $eq\text{-}onp\ P = BNF\text{-}Def.Grp\ (Collect\ P)\ id$
⟨*proof*⟩

**lemma** *eq-onp-to-eq*: *eq-onp* $P\ x\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *eq-onp-top-eq-eq*: *eq-onp top* $= (=)$
⟨*proof*⟩

**lemma** *eq-onp-same-args*: *eq-onp* $P\ x\ x = P\ x$
⟨*proof*⟩

**lemma** *eq-onp-eqD*: *eq-onp* $P = Q \Longrightarrow P\ x = Q\ x\ x$
⟨*proof*⟩

**lemma** *Ball-Collect*: *Ball* $A\ P = (A \subseteq (\mathit{Collect}\ P))$
⟨*proof*⟩

**lemma** *eq-onp-mono0*: $\forall x{\in}A.\ P\ x \longrightarrow Q\ x \Longrightarrow \forall x{\in}A.\ \forall y{\in}A.$ *eq-onp* $P\ x\ y \longrightarrow$ *eq-onp* $Q\ x\ y$
⟨*proof*⟩

**lemma** *eq-onp-True*: *eq-onp* $(\lambda\text{-}.\ \mathit{True}) = (=)$
⟨*proof*⟩

**lemma** *Ball-image-comp*: *Ball* $(f\ `\ A)\ g =$ *Ball* $A\ (g \circ f)$
⟨*proof*⟩

**lemma** *rel-fun-Collect-case-prodD*:
*rel-fun* $A\ B\ f\ g \Longrightarrow X \subseteq$ *Collect* (*case-prod* $A$) $\Longrightarrow x \in X \Longrightarrow B\ ((f \circ \mathit{fst})\ x)\ ((g \circ \mathit{snd})\ x)$
⟨*proof*⟩

**lemma** *eq-onp-mono-iff*: *eq-onp* $P \le$ *eq-onp* $Q \longleftrightarrow P \le Q$
⟨*proof*⟩

⟨*ML*⟩

**end**

# 34 Composition of Bounded Natural Functors

**theory** *BNF-Composition*
**imports** *BNF-Def*
**begin**

**lemma** *ssubst-mem*: $[\![t = s;\ s \in X]\!] \Longrightarrow t \in X$
⟨*proof*⟩

**lemma** *empty-natural*: $(\lambda\text{-}.\ \{\}) \circ f = \mathit{image}\ g \circ (\lambda\text{-}.\ \{\})$
⟨*proof*⟩

**lemma** *Cinfinite-gt-empty*: $Cinfinite\ r \Longrightarrow |\{\}| <o\ r$
⟨*proof*⟩

**lemma** *Union-natural*: $Union \circ image\ (image\ f) = image\ f \circ Union$
⟨*proof*⟩

**lemma** *in-Union-o-assoc*: $x \in (Union \circ gset \circ gmap)\ A \Longrightarrow x \in (Union \circ (gset \circ gmap))\ A$
⟨*proof*⟩

**lemma** *regularCard-UNION-bound*:
**assumes** $Cinfinite\ r\ regularCard\ r$ **and** $|I| <o\ r \bigwedge i.\ i \in I \Longrightarrow |A\ i| <o\ r$
**shows** $|\bigcup i{\in}I.\ A\ i| <o\ r$
⟨*proof*⟩

**lemma** *comp-single-set-bd-strict*:
**assumes** *fbd*: $Cinfinite\ fbd\ regularCard\ fbd$ **and**
*gbd*: $Cinfinite\ gbd\ regularCard\ gbd$ **and**
*fset-bd*: $\bigwedge x.\ |fset\ x| <o\ fbd$ **and**
*gset-bd*: $\bigwedge x.\ |gset\ x| <o\ gbd$
**shows** $|\bigcup(fset\ `\ gset\ x)| <o\ gbd *c\ fbd$
⟨*proof*⟩

**lemma** *comp-single-set-bd*:
**assumes** *fbd-Card-order*: $Card\text{-}order\ fbd$ **and**
*fset-bd*: $\bigwedge x.\ |fset\ x| \leq o\ fbd$ **and**
*gset-bd*: $\bigwedge x.\ |gset\ x| \leq o\ gbd$
**shows** $|\bigcup(fset\ `\ gset\ x)| \leq o\ gbd *c\ fbd$
⟨*proof*⟩

**lemma** *csum-dup*: $cinfinite\ r \Longrightarrow Card\text{-}order\ r \Longrightarrow p +c\ p' =o\ r +c\ r \Longrightarrow p +c\ p' =o\ r$
⟨*proof*⟩

**lemma** *cprod-dup*: $cinfinite\ r \Longrightarrow Card\text{-}order\ r \Longrightarrow p *c\ p' =o\ r *c\ r \Longrightarrow p *c\ p' =o\ r$
⟨*proof*⟩

**lemma** *Union-image-insert*: $\bigcup(f\ `\ insert\ a\ B) = f\ a \cup \bigcup(f\ `\ B)$
⟨*proof*⟩

**lemma** *Union-image-empty*: $A \cup \bigcup(f\ `\ \{\}) = A$
⟨*proof*⟩

**lemma** *image-o-collect*: $collect\ ((\lambda f.\ image\ g \circ f)\ `\ F) = image\ g \circ collect\ F$
⟨*proof*⟩

**lemma** *conj-subset-def*: $A \subseteq \{x.\ P\ x \wedge Q\ x\} = (A \subseteq \{x.\ P\ x\} \wedge A \subseteq \{x.\ Q\ x\})$
⟨*proof*⟩

**lemma** *UN-image-subset*: $\bigcup (f \text{ ` } g\ x) \subseteq X = (g\ x \subseteq \{x.\ f\ x \subseteq X\})$
⟨*proof*⟩

**lemma** *comp-set-bd-Union-o-collect*: $|\bigcup (\bigcup ((\lambda f.\ f\ x) \text{ ` } X))| \leq o\ hbd \Longrightarrow |(Union \circ collect\ X)\ x| \leq o\ hbd$
⟨*proof*⟩

**lemma** *comp-set-bd-Union-o-collect-strict*: $|\bigcup (\bigcup ((\lambda f.\ f\ x) \text{ ` } X))| <o\ hbd \Longrightarrow |(Union \circ collect\ X)\ x| <o\ hbd$
⟨*proof*⟩

**lemma** *Collect-inj*: $Collect\ P = Collect\ Q \Longrightarrow P = Q$
⟨*proof*⟩

**lemma** *Grp-fst-snd*: $(Grp\ (Collect\ (case\text{-}prod\ R))\ fst)^{-1-1}\ OO\ Grp\ (Collect\ (case\text{-}prod\ R))\ snd = R$
⟨*proof*⟩

**lemma** *OO-Grp-cong*: $A = B \Longrightarrow (Grp\ A\ f)^{-1-1}\ OO\ Grp\ A\ g = (Grp\ B\ f)^{-1-1}\ OO\ Grp\ B\ g$
⟨*proof*⟩

**lemma** *vimage2p-relcompp-mono*: $R\ OO\ S \leq T \Longrightarrow$
$vimage2p\ f\ g\ R\ OO\ vimage2p\ g\ h\ S \leq vimage2p\ f\ h\ T$
⟨*proof*⟩

**lemma** *type-copy-map-cong0*: $M\ (g\ x) = N\ (h\ x) \Longrightarrow (f \circ M \circ g)\ x = (f \circ N \circ h)\ x$
⟨*proof*⟩

**lemma** *type-copy-set-bd*: $(\bigwedge y.\ |S\ y| <o\ bd) \Longrightarrow |(S \circ Rep)\ x| <o\ bd$
⟨*proof*⟩

**lemma** *vimage2p-cong*: $R = S \Longrightarrow vimage2p\ f\ g\ R = vimage2p\ f\ g\ S$
⟨*proof*⟩

**lemma** *Ball-comp-iff*: $(\lambda x.\ Ball\ (A\ x)\ f) \circ g = (\lambda x.\ Ball\ ((A \circ g)\ x)\ f)$
⟨*proof*⟩

**lemma** *conj-comp-iff*: $(\lambda x.\ P\ x \wedge Q\ x) \circ g = (\lambda x.\ (P \circ g)\ x \wedge (Q \circ g)\ x)$
⟨*proof*⟩

**context**
**fixes** *Rep Abs*
**assumes** *type-copy*: *type-definition Rep Abs UNIV*
**begin**

**lemma** *type-copy-map-id0*: $M = id \Longrightarrow Abs \circ M \circ Rep = id$

⟨*proof*⟩

**lemma** *type-copy-map-comp0*: $M = M1 \circ M2 \Longrightarrow f \circ M \circ g = (f \circ M1 \circ Rep) \circ (Abs \circ M2 \circ g)$
⟨*proof*⟩

**lemma** *type-copy-set-map0*: $S \circ M = image\ f \circ S' \Longrightarrow (S \circ Rep) \circ (Abs \circ M \circ g) = image\ f \circ (S' \circ g)$
⟨*proof*⟩

**lemma** *type-copy-wit*: $x \in (S \circ Rep)\ (Abs\ y) \Longrightarrow x \in S\ y$
⟨*proof*⟩

**lemma** *type-copy-vimage2p-Grp-Rep*: $vimage2p\ f\ Rep\ (Grp\ (Collect\ P)\ h) = Grp\ (Collect\ (\lambda x.\ P\ (f\ x)))\ (Abs \circ h \circ f)$
⟨*proof*⟩

**lemma** *type-copy-vimage2p-Grp-Abs*:
$\bigwedge h.\ vimage2p\ g\ Abs\ (Grp\ (Collect\ P)\ h) = Grp\ (Collect\ (\lambda x.\ P\ (g\ x)))\ (Rep \circ h \circ g)$
⟨*proof*⟩

**lemma** *type-copy-ex-RepI*: $(\exists\, b.\ F\ b) = (\exists\, b.\ F\ (Rep\ b))$
⟨*proof*⟩

**lemma** *vimage2p-relcompp-converse*:
$vimage2p\ f\ g\ (R^{-1-1}\ OO\ S) = (vimage2p\ Rep\ f\ R)^{-1-1}\ OO\ vimage2p\ Rep\ g\ S$
⟨*proof*⟩

**end**

**bnf** *DEADID*: $'a$
*map*: $id :: 'a \Rightarrow 'a$
*bd*: *natLeq*
*rel*: $(=) :: 'a \Rightarrow 'a \Rightarrow bool$
⟨*proof*⟩

**definition** *id-bnf* $:: 'a \Rightarrow 'a$ **where**
*id-bnf* $\equiv (\lambda x.\ x)$

**lemma** *id-bnf-apply*: *id-bnf* $x = x$
⟨*proof*⟩

**bnf** *ID*: $'a$
*map*: *id-bnf* $:: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
*sets*: $\lambda x.\ \{x\}$
*bd*: *natLeq*
*rel*: *id-bnf* $:: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
*pred*: *id-bnf* $:: ('a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$

⟨*proof*⟩

**lemma** *type-definition-id-bnf-UNIV*: *type-definition id-bnf id-bnf UNIV*
⟨*proof*⟩

⟨*ML*⟩

**hide-fact**
*DEADID.inj-map DEADID.inj-map-strong DEADID.map-comp DEADID.map-cong DEADID.map-cong0*
*DEADID.map-cong-simp DEADID.map-id DEADID.map-id0 DEADID.map-ident DEADID.map-transfer*
*DEADID.rel-Grp DEADID.rel-compp DEADID.rel-compp-Grp DEADID.rel-conversep DEADID.rel-eq*
*DEADID.rel-flip DEADID.rel-map DEADID.rel-mono DEADID.rel-transfer*
*ID.inj-map ID.inj-map-strong ID.map-comp ID.map-cong ID.map-cong0 ID.map-cong-simp ID.map-id*
*ID.map-id0 ID.map-ident ID.map-transfer ID.rel-Grp ID.rel-compp ID.rel-compp-Grp ID.rel-conversep*
*ID.rel-eq ID.rel-flip ID.rel-map ID.rel-mono ID.rel-transfer ID.set-map ID.set-transfer*

**end**

# 35 Registration of Basic Types as Bounded Natural Functors

**theory** *Basic-BNFs*
**imports** *BNF-Def*
**begin**

**inductive-set** *setl* :: $'a + 'b \Rightarrow 'a$ *set* **for** $s$ :: $'a + 'b$ **where**
$s = Inl\ x \Longrightarrow x \in setl\ s$
**inductive-set** *setr* :: $'a + 'b \Rightarrow 'b$ *set* **for** $s$ :: $'a + 'b$ **where**
$s = Inr\ x \Longrightarrow x \in setr\ s$

**lemma** *sum-set-defs* [*code*]:
$setl = (\lambda x.\ case\ x\ of\ Inl\ z \Rightarrow \{z\} \mid \text{-} \Rightarrow \{\})$
$setr = (\lambda x.\ case\ x\ of\ Inr\ z \Rightarrow \{z\} \mid \text{-} \Rightarrow \{\})$
⟨*proof*⟩

**lemma** *rel-sum-simps* [*code*, *simp*]:
*rel-sum R1 R2* (*Inl a1*) (*Inl b1*) = *R1 a1 b1*
*rel-sum R1 R2* (*Inl a1*) (*Inr b2*) = *False*
*rel-sum R1 R2* (*Inr a2*) (*Inl b1*) = *False*
*rel-sum R1 R2* (*Inr a2*) (*Inr b2*) = *R2 a2 b2*
⟨*proof*⟩

**inductive**

*pred-sum* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a + 'b \Rightarrow bool$ **for** *P1 P2*
**where**
*P1 a* $\Longrightarrow$ *pred-sum P1 P2* (*Inl a*)
| *P2 b* $\Longrightarrow$ *pred-sum P1 P2* (*Inr b*)

**lemma** *pred-sum-inject* [*code*, *simp*]:
*pred-sum P1 P2* (*Inl a*) $\longleftrightarrow$ *P1 a*
*pred-sum P1 P2* (*Inr b*) $\longleftrightarrow$ *P2 b*
⟨*proof*⟩

**bnf** $'a + 'b$
*map*: *map-sum*
*sets*: *setl setr*
*bd*: *natLeq*
*wits*: *Inl Inr*
*rel*: *rel-sum*
*pred*: *pred-sum*
⟨*proof*⟩

**inductive-set** *fsts* :: $'a \times 'b \Rightarrow 'a$ *set* **for** *p* :: $'a \times 'b$ **where**
*fst p* $\in$ *fsts p*
**inductive-set** *snds* :: $'a \times 'b \Rightarrow 'b$ *set* **for** *p* :: $'a \times 'b$ **where**
*snd p* $\in$ *snds p*

**lemma** *prod-set-defs*[*code*]: *fsts* = ($\lambda p$. {*fst p*}) *snds* = ($\lambda p$. {*snd p*})
⟨*proof*⟩

**inductive**
*rel-prod* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('c \Rightarrow 'd \Rightarrow bool) \Rightarrow 'a \times 'c \Rightarrow 'b \times 'd \Rightarrow bool$
**for** *R1 R2*
**where**
⟦*R1 a b*; *R2 c d*⟧ $\Longrightarrow$ *rel-prod R1 R2* (*a*, *c*) (*b*, *d*)

**inductive**
*pred-prod* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a \times 'b \Rightarrow bool$ **for** *P1 P2*
**where**
⟦*P1 a*; *P2 b*⟧ $\Longrightarrow$ *pred-prod P1 P2* (*a*, *b*)

**lemma** *rel-prod-inject* [*code*, *simp*]:
*rel-prod R1 R2* (*a*, *b*) (*c*, *d*) $\longleftrightarrow$ *R1 a c* $\wedge$ *R2 b d*
⟨*proof*⟩

**lemma** *pred-prod-inject* [*code*, *simp*]:
*pred-prod P1 P2* (*a*, *b*) $\longleftrightarrow$ *P1 a* $\wedge$ *P2 b*
⟨*proof*⟩

**lemma** *rel-prod-conv*:
*rel-prod R1 R2* = ($\lambda$(*a*, *b*) (*c*, *d*). *R1 a c* $\wedge$ *R2 b d*)
⟨*proof*⟩

**definition**
*pred-fun* :: $('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$
**where**
*pred-fun* $A\ B = (\lambda f.\ \forall x.\ A\ x \longrightarrow B\ (f\ x))$

**lemma** *pred-funI*: $(\bigwedge x.\ A\ x \Longrightarrow B\ (f\ x)) \Longrightarrow$ *pred-fun* $A\ B\ f$
⟨*proof*⟩

**bnf** $'a \times 'b$
*map*: *map-prod*
*sets*: *fsts snds*
*bd*: *natLeq*
*rel*: *rel-prod*
*pred*: *pred-prod*
⟨*proof*⟩

**lemma** *card-order-bd-fun*: *card-order* (*natLeq* +*c* *card-suc* ( |*UNIV*| ))
⟨*proof*⟩

**lemma** *Cinfinite-bd-fun*: *Cinfinite* (*natLeq* +*c* *card-suc* ( |*UNIV*| ))
⟨*proof*⟩

**lemma** *regularCard-bd-fun*: *regularCard* (*natLeq* +*c* *card-suc* ( |*UNIV*| ))
(**is** *regularCard* (- +*c* *card-suc* *?U*))
⟨*proof*⟩

**lemma** *ordLess-bd-fun*: |*UNIV*::'*a set*| <*o* *natLeq* +*c* *card-suc* ( |*UNIV*::'*a set*| )
(**is** - <*o* (- +*c* *card-suc* (*?U* :: '*a rel*)))
⟨*proof*⟩

**bnf** $'a \Rightarrow 'b$
*map*: $(\circ)$
*sets*: *range*
*bd*: *natLeq* +*c* *card-suc* ( |*UNIV*::'*a set*| )
*rel*: *rel-fun* (=)
*pred*: *pred-fun* (λ-. *True*)
⟨*proof*⟩

**end**

# 36 Shared Fixpoint Operations on Bounded Natural Functors

**theory** *BNF-Fixpoint-Base*
**imports** *BNF-Composition Basic-BNFs*
**begin**

**lemma** *conj-imp-eq-imp-imp*: $(P \wedge Q \Longrightarrow PROP\ R) \equiv (P \Longrightarrow Q \Longrightarrow PROP\ R)$
⟨*proof*⟩

**lemma** *predicate2D-conj*: $P \leq Q \wedge R \Longrightarrow R \wedge (P\ x\ y \longrightarrow Q\ x\ y)$
⟨*proof*⟩

**lemma** *eq-sym-Unity-conv*: $(x = (() = ())) = x$
⟨*proof*⟩

**lemma** *case-unit-Unity*: $(case\ u\ of\ () \Rightarrow f) = f$
⟨*proof*⟩

**lemma** *case-prod-Pair-iden*: $(case\ p\ of\ (x,\ y) \Rightarrow (x,\ y)) = p$
⟨*proof*⟩

**lemma** *unit-all-impI*: $(P\ () \Longrightarrow Q\ ()) \Longrightarrow \forall\, x.\ P\ x \longrightarrow Q\ x$
⟨*proof*⟩

**lemma** *pointfree-idE*: $f \circ g = id \Longrightarrow f\ (g\ x) = x$
⟨*proof*⟩

**lemma** *o-bij*:
**assumes** *gf*: $g \circ f = id$ **and** *fg*: $f \circ g = id$
**shows** *bij f*
⟨*proof*⟩

**lemma** *case-sum-step*:
$case\text{-}sum\ (case\text{-}sum\ f'\ g')\ g\ (Inl\ p) = case\text{-}sum\ f'\ g'\ p$
$case\text{-}sum\ f\ (case\text{-}sum\ f'\ g')\ (Inr\ p) = case\text{-}sum\ f'\ g'\ p$
⟨*proof*⟩

**lemma** *obj-one-pointE*: $\forall\, x.\ s = x \longrightarrow P \Longrightarrow P$
⟨*proof*⟩

**lemma** *type-copy-obj-one-point-absE*:
**assumes** *type-definition Rep Abs UNIV* $\forall\, x.\ s = Abs\ x \longrightarrow P$ **shows** *P*
⟨*proof*⟩

**lemma** *obj-sumE-f*:
**assumes** $\forall\, x.\ s = f\ (Inl\ x) \longrightarrow P$ $\forall\, x.\ s = f\ (Inr\ x) \longrightarrow P$
**shows** $\forall\, x.\ s = f\ x \longrightarrow P$
⟨*proof*⟩

**lemma** *case-sum-if*:
$case\text{-}sum\ f\ g\ (if\ p\ then\ Inl\ x\ else\ Inr\ y) = (if\ p\ then\ f\ x\ else\ g\ y)$
⟨*proof*⟩

**lemma** *prod-set-simps*[*simp*]:
$fsts\ (x,\ y) = \{x\}$

*snds* $(x, y) = \{y\}$
⟨*proof*⟩

**lemma** *sum-set-simps*[*simp*]:
*setl* (*Inl* $x$) $= \{x\}$
*setl* (*Inr* $x$) $= \{\}$
*setr* (*Inl* $x$) $= \{\}$
*setr* (*Inr* $x$) $= \{x\}$
⟨*proof*⟩

**lemma** *Inl-Inr-False*: (*Inl* $x$ = *Inr* $y$) = *False*
⟨*proof*⟩

**lemma** *Inr-Inl-False*: (*Inr* $x$ = *Inl* $y$) = *False*
⟨*proof*⟩

**lemma** *spec2*: $\forall x\ y.\ P\ x\ y \Longrightarrow P\ x\ y$
⟨*proof*⟩

**lemma** *rewriteR-comp-comp*: $[\![g \circ h = r]\!] \Longrightarrow f \circ g \circ h = f \circ r$
⟨*proof*⟩

**lemma** *rewriteR-comp-comp2*: $[\![g \circ h = r1 \circ r2;\ f \circ r1 = l]\!] \Longrightarrow f \circ g \circ h = l \circ r2$
⟨*proof*⟩

**lemma** *rewriteL-comp-comp*: $[\![f \circ g = l]\!] \Longrightarrow f \circ (g \circ h) = l \circ h$
⟨*proof*⟩

**lemma** *rewriteL-comp-comp2*: $[\![f \circ g = l1 \circ l2;\ l2 \circ h = r]\!] \Longrightarrow f \circ (g \circ h) = l1 \circ r$
⟨*proof*⟩

**lemma** *convol-o*: $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$
⟨*proof*⟩

**lemma** *map-prod-o-convol*: *map-prod* $h1\ h2 \circ \langle f, g\rangle = \langle h1 \circ f, h2 \circ g\rangle$
⟨*proof*⟩

**lemma** *map-prod-o-convol-id*: (*map-prod* $f$ *id* $\circ \langle$*id*$, g\rangle)\ x = \langle$*id* $\circ f, g\rangle\ x$
⟨*proof*⟩

**lemma** *o-case-sum*: $h \circ$ *case-sum* $f\ g$ = *case-sum* $(h \circ f)\ (h \circ g)$
⟨*proof*⟩

**lemma** *case-sum-o-map-sum*: *case-sum* $f\ g \circ$ *map-sum* $h1\ h2$ = *case-sum* $(f \circ h1)\ (g \circ h2)$
⟨*proof*⟩

**lemma** *case-sum-o-map-sum-id*: $(\textit{case-sum id } g \circ \textit{map-sum } f \textit{ id})\ x = \textit{case-sum}\ (f \circ \textit{id})\ g\ x$
⟨*proof*⟩

**lemma** *rel-fun-def-butlast*:
$\textit{rel-fun } R\ (\textit{rel-fun } S\ T)\ f\ g = (\forall x\ y.\ R\ x\ y \longrightarrow (\textit{rel-fun } S\ T)\ (f\ x)\ (g\ y))$
⟨*proof*⟩

**lemma** *subst-eq-imp*: $(\forall a\ b.\ a = b \longrightarrow P\ a\ b) \equiv (\forall a.\ P\ a\ a)$
⟨*proof*⟩

**lemma** *eq-subset*: $(=) \leq (\lambda a\ b.\ P\ a\ b \vee a = b)$
⟨*proof*⟩

**lemma** *eq-le-Grp-id-iff*: $((=) \leq \textit{Grp}\ (\textit{Collect } R)\ \textit{id}) = (\textit{All } R)$
⟨*proof*⟩

**lemma** *Grp-id-mono-subst*: $(\bigwedge x\ y.\ \textit{Grp } P\ \textit{id}\ x\ y \Longrightarrow \textit{Grp } Q\ \textit{id}\ (f\ x)\ (f\ y)) \equiv$
$(\bigwedge x.\ x \in P \Longrightarrow f\ x \in Q)$
⟨*proof*⟩

**lemma** *vimage2p-mono*: $\textit{vimage2p } f\ g\ R\ x\ y \Longrightarrow R \leq S \Longrightarrow \textit{vimage2p } f\ g\ S\ x\ y$
⟨*proof*⟩

**lemma** *vimage2p-refl*: $(\bigwedge x.\ R\ x\ x) \Longrightarrow \textit{vimage2p } f\ f\ R\ x\ x$
⟨*proof*⟩

**lemma**
**assumes** *type-definition Rep Abs UNIV*
**shows** *type-copy-Rep-o-Abs*: $\textit{Rep} \circ \textit{Abs} = \textit{id}$ **and** *type-copy-Abs-o-Rep*: $\textit{Abs} \circ \textit{Rep} = \textit{id}$
⟨*proof*⟩

**lemma** *type-copy-map-comp0-undo*:
**assumes** *type-definition Rep Abs UNIV*
*type-definition Rep′ Abs′ UNIV*
*type-definition Rep″ Abs″ UNIV*
**shows** $\textit{Abs}' \circ M \circ \textit{Rep}'' = (\textit{Abs}' \circ M1 \circ \textit{Rep}) \circ (\textit{Abs} \circ M2 \circ \textit{Rep}'') \Longrightarrow M1 \circ M2 = M$
⟨*proof*⟩

**lemma** *vimage2p-id*: $\textit{vimage2p id id } R = R$
⟨*proof*⟩

**lemma** *vimage2p-comp*: $\textit{vimage2p}\ (f1 \circ f2)\ (g1 \circ g2) = \textit{vimage2p } f2\ g2 \circ \textit{vimage2p } f1\ g1$
⟨*proof*⟩

**lemma** *vimage2p-rel-fun*: $\textit{rel-fun}\ (\textit{vimage2p } f\ g\ R)\ R\ f\ g$

⟨*proof*⟩

**lemma** *fun-cong-unused-0*: $f = (\lambda x.\ g) \Longrightarrow f\ (\lambda x.\ 0) = g$
⟨*proof*⟩

**lemma** *inj-on-convol-ident*: $inj\text{-}on\ (\lambda x.\ (x, f\ x))\ X$
⟨*proof*⟩

**lemma** *map-sum-if-distrib-then*:
$\bigwedge f\ g\ e\ x\ y.\ map\text{-}sum\ f\ g\ (if\ e\ then\ Inl\ x\ else\ y) = (if\ e\ then\ Inl\ (f\ x)\ else\ map\text{-}sum\ f\ g\ y)$
$\bigwedge f\ g\ e\ x\ y.\ map\text{-}sum\ f\ g\ (if\ e\ then\ Inr\ x\ else\ y) = (if\ e\ then\ Inr\ (g\ x)\ else\ map\text{-}sum\ f\ g\ y)$
⟨*proof*⟩

**lemma** *map-sum-if-distrib-else*:
$\bigwedge f\ g\ e\ x\ y.\ map\text{-}sum\ f\ g\ (if\ e\ then\ x\ else\ Inl\ y) = (if\ e\ then\ map\text{-}sum\ f\ g\ x\ else\ Inl\ (f\ y))$
$\bigwedge f\ g\ e\ x\ y.\ map\text{-}sum\ f\ g\ (if\ e\ then\ x\ else\ Inr\ y) = (if\ e\ then\ map\text{-}sum\ f\ g\ x\ else\ Inr\ (g\ y))$
⟨*proof*⟩

**lemma** *case-prod-app*: $case\text{-}prod\ f\ x\ y = case\text{-}prod\ (\lambda l\ r.\ f\ l\ r\ y)\ x$
⟨*proof*⟩

**lemma** *case-sum-map-sum*: $case\text{-}sum\ l\ r\ (map\text{-}sum\ f\ g\ x) = case\text{-}sum\ (l \circ f)\ (r \circ g)\ x$
⟨*proof*⟩

**lemma** *case-sum-transfer*:
$rel\text{-}fun\ (rel\text{-}fun\ R\ T)\ (rel\text{-}fun\ (rel\text{-}fun\ S\ T)\ (rel\text{-}fun\ (rel\text{-}sum\ R\ S)\ T))\ case\text{-}sum\ case\text{-}sum$
⟨*proof*⟩

**lemma** *case-prod-map-prod*: $case\text{-}prod\ h\ (map\text{-}prod\ f\ g\ x) = case\text{-}prod\ (\lambda l\ r.\ h\ (f\ l)\ (g\ r))\ x$
⟨*proof*⟩

**lemma** *case-prod-o-map-prod*: $case\text{-}prod\ f \circ map\text{-}prod\ g1\ g2 = case\text{-}prod\ (\lambda l\ r.\ f\ (g1\ l)\ (g2\ r))$
⟨*proof*⟩

**lemma** *case-prod-transfer*:
$(rel\text{-}fun\ (rel\text{-}fun\ A\ (rel\text{-}fun\ B\ C))\ (rel\text{-}fun\ (rel\text{-}prod\ A\ B)\ C))\ case\text{-}prod\ case\text{-}prod$
⟨*proof*⟩

**lemma** *eq-ifI*: $(P \longrightarrow t = u1) \Longrightarrow (\neg\ P \longrightarrow t = u2) \Longrightarrow t = (if\ P\ then\ u1\ else\ u2)$
⟨*proof*⟩

**lemma** *comp-transfer*:
*rel-fun* (*rel-fun* $B$ $C$) (*rel-fun* (*rel-fun* $A$ $B$) (*rel-fun* $A$ $C$)) ($\circ$) ($\circ$)
⟨*proof*⟩

**lemma** *If-transfer*: *rel-fun* (=) (*rel-fun* $A$ (*rel-fun* $A$ $A$)) *If If*
⟨*proof*⟩

**lemma** *Abs-transfer*:
**assumes** *type-copy1*: *type-definition Rep1 Abs1 UNIV*
**assumes** *type-copy2*: *type-definition Rep2 Abs2 UNIV*
**shows** *rel-fun* $R$ (*vimage2p Rep1 Rep2* $R$) *Abs1 Abs2*
⟨*proof*⟩

**lemma** *Inl-transfer*:
*rel-fun* $S$ (*rel-sum* $S$ $T$) *Inl Inl*
⟨*proof*⟩

**lemma** *Inr-transfer*:
*rel-fun* $T$ (*rel-sum* $S$ $T$) *Inr Inr*
⟨*proof*⟩

**lemma** *Pair-transfer*: *rel-fun* $A$ (*rel-fun* $B$ (*rel-prod* $A$ $B$)) *Pair Pair*
⟨*proof*⟩

**lemma** *eq-onp-live-step*: $x = y \Longrightarrow$ *eq-onp* $P\ a\ a \wedge x \longleftrightarrow P\ a \wedge y$
⟨*proof*⟩

**lemma** *top-conj*: *top* $x \wedge P \longleftrightarrow P$ $P \wedge$ *top* $x \longleftrightarrow P$
⟨*proof*⟩

**lemma** *fst-convol'*: *fst* ($\langle f, g\rangle\ x$) $= f\ x$
⟨*proof*⟩

**lemma** *snd-convol'*: *snd* ($\langle f, g\rangle\ x$) $= g\ x$
⟨*proof*⟩

**lemma** *convol-expand-snd*: *fst* $\circ f = g \Longrightarrow \langle g,$ *snd* $\circ f\rangle = f$
⟨*proof*⟩

**lemma** *convol-expand-snd'*:
**assumes** (*fst* $\circ f = g$)
**shows** $h =$ *snd* $\circ f \longleftrightarrow \langle g, h\rangle = f$
⟨*proof*⟩

**lemma** *case-sum-expand-Inr-pointfree*: $f \circ$ *Inl* $= g \Longrightarrow$ *case-sum* $g$ ($f \circ$ *Inr*) $= f$
⟨*proof*⟩

**lemma** *case-sum-expand-Inr'*: $f \circ$ *Inl* $= g \Longrightarrow h = f \circ$ *Inr* $\longleftrightarrow$ *case-sum* $g\ h = f$

⟨*proof*⟩

**lemma** *case-sum-expand-Inr*: $f \circ Inl = g \Longrightarrow f\ x = case\text{-}sum\ g\ (f \circ Inr)\ x$
⟨*proof*⟩

**lemma** *id-transfer*: *rel-fun A A id id*
⟨*proof*⟩

**lemma** *fst-transfer*: *rel-fun (rel-prod A B) A fst fst*
⟨*proof*⟩

**lemma** *snd-transfer*: *rel-fun (rel-prod A B) B snd snd*
⟨*proof*⟩

**lemma** *convol-transfer*:
*rel-fun (rel-fun R S) (rel-fun (rel-fun R T) (rel-fun R (rel-prod S T))) BNF-Def.convol BNF-Def.convol*
⟨*proof*⟩

**lemma** *Let-const*: $Let\ x\ (\lambda\text{-}.\ c) = c$
⟨*proof*⟩

⟨*ML*⟩

**end**

# 37 Least Fixpoint (Datatype) Operation on Bounded Natural Functors

**theory** *BNF-Least-Fixpoint*
**imports** *BNF-Fixpoint-Base*
**keywords**
*datatype* :: *thy-defn* **and**
*datatype-compat* :: *thy-defn*
**begin**

**lemma** *subset-emptyI*: $(\bigwedge x.\ x \in A \Longrightarrow False) \Longrightarrow A \subseteq \{\}$
⟨*proof*⟩

**lemma** *image-Collect-subsetI*: $(\bigwedge x.\ P\ x \Longrightarrow f\ x \in B) \Longrightarrow f\ `\ \{x.\ P\ x\} \subseteq B$
⟨*proof*⟩

**lemma** *Collect-restrict*: $\{x.\ x \in X \wedge P\ x\} \subseteq X$
⟨*proof*⟩

**lemma** *prop-restrict*: $[\![x \in Z;\ Z \subseteq \{x.\ x \in X \wedge P\ x\}]\!] \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *underS-I*: $\llbracket i \neq j;\ (i, j) \in R\rrbracket \Longrightarrow i \in \mathit{underS}\ R\ j$
⟨*proof*⟩

**lemma** *underS-E*: $i \in \mathit{underS}\ R\ j \Longrightarrow i \neq j \wedge (i, j) \in R$
⟨*proof*⟩

**lemma** *underS-Field*: $i \in \mathit{underS}\ R\ j \Longrightarrow i \in \mathit{Field}\ R$
⟨*proof*⟩

**lemma** *ex-bij-betw*: $|A| \leq o\ (r :: 'b\ \mathit{rel}) \Longrightarrow \exists f\ B :: 'b\ \mathit{set}.\ \mathit{bij\text{-}betw}\ f\ B\ A$
⟨*proof*⟩

**lemma** *bij-betwI'*:
$\llbracket \bigwedge x\ y.\ \llbracket x \in X;\ y \in X\rrbracket \Longrightarrow (f\ x = f\ y) = (x = y);$
$\bigwedge x.\ x \in X \Longrightarrow f\ x \in Y;$
$\bigwedge y.\ y \in Y \Longrightarrow \exists x \in X.\ y = f\ x\rrbracket \Longrightarrow \mathit{bij\text{-}betw}\ f\ X\ Y$
⟨*proof*⟩

**lemma** *surj-fun-eq*:
**assumes** *surj-on*: $f\ `\ X = \mathit{UNIV}$ **and** *eq-on*: $\forall x \in X.\ (g1 \circ f)\ x = (g2 \circ f)\ x$
**shows** $g1 = g2$
⟨*proof*⟩

**lemma** *Card-order-wo-rel*: $\mathit{Card\text{-}order}\ r \Longrightarrow \mathit{wo\text{-}rel}\ r$
⟨*proof*⟩

**lemma** *Cinfinite-limit*: $\llbracket x \in \mathit{Field}\ r;\ \mathit{Cinfinite}\ r\rrbracket \Longrightarrow \exists y \in \mathit{Field}\ r.\ x \neq y \wedge (x, y) \in r$
⟨*proof*⟩

**lemma** *Card-order-trans*:
$\llbracket \mathit{Card\text{-}order}\ r;\ x \neq y;\ (x, y) \in r;\ y \neq z;\ (y, z) \in r\rrbracket \Longrightarrow x \neq z \wedge (x, z) \in r$
⟨*proof*⟩

**lemma** *Cinfinite-limit2*:
**assumes** *x1*: $x1 \in \mathit{Field}\ r$ **and** *x2*: $x2 \in \mathit{Field}\ r$ **and** *r*: $\mathit{Cinfinite}\ r$
**shows** $\exists y \in \mathit{Field}\ r.\ (x1 \neq y \wedge (x1, y) \in r) \wedge (x2 \neq y \wedge (x2, y) \in r)$
⟨*proof*⟩

**lemma** *Cinfinite-limit-finite*:
$\llbracket \mathit{finite}\ X;\ X \subseteq \mathit{Field}\ r;\ \mathit{Cinfinite}\ r\rrbracket \Longrightarrow \exists y \in \mathit{Field}\ r.\ \forall x \in X.\ (x \neq y \wedge (x, y) \in r)$
⟨*proof*⟩

**lemma** *insert-subsetI*: $\llbracket x \in A;\ X \subseteq A\rrbracket \Longrightarrow \mathit{insert}\ x\ X \subseteq A$
⟨*proof*⟩

**lemmas** *well-order-induct-imp* = *wo-rel.well-order-induct*[*of* $r\ \lambda x.\ x \in \mathit{Field}\ r \longrightarrow P\ x$ **for** $r\ P$]

**lemma** *meta-spec2*:
**assumes** $(\bigwedge x\ y.\ PROP\ P\ x\ y)$
**shows** $PROP\ P\ x\ y$
⟨*proof*⟩

**lemma** *nchotomy-relcomppE*:
**assumes** $\bigwedge y.\ \exists\, x.\ y = f\ x\ (r\ OO\ s)\ a\ c\ \bigwedge b.\ r\ a\ (f\ b) \Longrightarrow s\ (f\ b)\ c \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *predicate2D-vimage2p*: $[\![R \le vimage2p\ f\ g\ S;\ R\ x\ y]\!] \Longrightarrow S\ (f\ x)\ (g\ y)$
⟨*proof*⟩

**lemma** *ssubst-Pair-rhs*: $[\![(r,\ s) \in R;\ s' = s]\!] \Longrightarrow (r,\ s') \in R$
⟨*proof*⟩

**lemma** *all-mem-range1*:
$(\bigwedge y.\ y \in range\ f \Longrightarrow P\ y) \equiv (\bigwedge x.\ P\ (f\ x))$
⟨*proof*⟩

**lemma** *all-mem-range2*:
$(\bigwedge fa\ y.\ fa \in range\ f \Longrightarrow y \in range\ fa \Longrightarrow P\ y) \equiv (\bigwedge x\ xa.\ P\ (f\ x\ xa))$
⟨*proof*⟩

**lemma** *all-mem-range3*:
$(\bigwedge fa\ fb\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow y \in range\ fb \Longrightarrow P\ y) \equiv (\bigwedge x\ xa\ xb.\ P\ (f\ x\ xa\ xb))$
⟨*proof*⟩

**lemma** *all-mem-range4*:
$(\bigwedge fa\ fb\ fc\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow fc \in range\ fb \Longrightarrow y \in range\ fc \Longrightarrow P\ y) \equiv$
$(\bigwedge x\ xa\ xb\ xc.\ P\ (f\ x\ xa\ xb\ xc))$
⟨*proof*⟩

**lemma** *all-mem-range5*:
$(\bigwedge fa\ fb\ fc\ fd\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow fc \in range\ fb \Longrightarrow fd \in range\ fc \Longrightarrow$
$y \in range\ fd \Longrightarrow P\ y) \equiv$
$(\bigwedge x\ xa\ xb\ xc\ xd.\ P\ (f\ x\ xa\ xb\ xc\ xd))$
⟨*proof*⟩

**lemma** *all-mem-range6*:
$(\bigwedge fa\ fb\ fc\ fd\ fe\ ff\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow fc \in range\ fb \Longrightarrow fd \in range\ fc \Longrightarrow$
$fe \in range\ fd \Longrightarrow ff \in range\ fe \Longrightarrow y \in range\ ff \Longrightarrow P\ y) \equiv$
$(\bigwedge x\ xa\ xb\ xc\ xd\ xe\ xf.\ P\ (f\ x\ xa\ xb\ xc\ xd\ xe\ xf))$
⟨*proof*⟩

**lemma** *all-mem-range7*:
$(\bigwedge fa\ fb\ fc\ fd\ fe\ ff\ fg\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow fc \in range\ fb \Longrightarrow fd \in range\ fc \Longrightarrow$
$fe \in range\ fd \Longrightarrow ff \in range\ fe \Longrightarrow fg \in range\ ff \Longrightarrow y \in range\ fg \Longrightarrow P\ y) \equiv$
$(\bigwedge x\ xa\ xb\ xc\ xd\ xe\ xf\ xg.\ P\ (f\ x\ xa\ xb\ xc\ xd\ xe\ xf\ xg))$
⟨*proof*⟩

**lemma** *all-mem-range8*:
$(\bigwedge fa\ fb\ fc\ fd\ fe\ ff\ fg\ fh\ y.\ fa \in range\ f \Longrightarrow fb \in range\ fa \Longrightarrow fc \in range\ fb \Longrightarrow fd \in range\ fc \Longrightarrow$
$fe \in range\ fd \Longrightarrow ff \in range\ fe \Longrightarrow fg \in range\ ff \Longrightarrow fh \in range\ fg \Longrightarrow y \in range\ fh \Longrightarrow P\ y) \equiv$
$(\bigwedge x\ xa\ xb\ xc\ xd\ xe\ xf\ xg\ xh.\ P\ (f\ x\ xa\ xb\ xc\ xd\ xe\ xf\ xg\ xh))$
⟨*proof*⟩

**lemmas** *all-mem-range* = *all-mem-range1 all-mem-range2 all-mem-range3 all-mem-range4 all-mem-range5*
*all-mem-range6 all-mem-range7 all-mem-range8*

**lemma** *pred-fun-True-id*: *NO-MATCH id p* $\Longrightarrow$ *pred-fun* $(\lambda x.\ True)\ p\ f =$ *pred-fun* $(\lambda x.\ True)\ id\ (p \circ f)$
⟨*proof*⟩

⟨*ML*⟩

**end**

# 38 Equivalence Relations in Higher-Order Set Theory

**theory** *Equiv-Relations*
**imports** *BNF-Least-Fixpoint*
**begin**

## 38.1 Equivalence relations – set version

**definition** *equiv* :: $'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow bool$
**where** $equiv\ A\ r \longleftrightarrow r \subseteq A \times A \wedge$ *refl-on* $A\ r \wedge sym\ r \wedge trans\ r$

**lemma** *equivI*: $r \subseteq A \times A \Longrightarrow$ *refl-on* $A\ r \Longrightarrow sym\ r \Longrightarrow trans\ r \Longrightarrow equiv\ A\ r$
⟨*proof*⟩

**lemma** *equivE*:
**assumes** *equiv A r*
**obtains** $r \subseteq A \times A$ **and** *refl-on* $A\ r$ **and** *sym r* **and** *trans r*
⟨*proof*⟩

Suppes, Theorem 70: $r$ is an equiv relation iff $r^{-1}\ O\ r = r$.

First half: $equiv\ A\ r \Longrightarrow r^{-1}\ O\ r = r$.

**lemma** *sym-trans-comp-subset*:
**assumes** $r \subseteq A \times A$ **and** $sym\text{-}on\ A\ r$ **and** $trans\text{-}on\ A\ r$
**shows** $r^{-1}\ O\ r \subseteq r$
⟨*proof*⟩

**lemma** *refl-on-comp-subset*: $r \subseteq A \times A \Longrightarrow refl\text{-}on\ A\ r \Longrightarrow r \subseteq r^{-1}\ O\ r$
⟨*proof*⟩

**lemma** *equiv-comp-eq*: $equiv\ A\ r \Longrightarrow r^{-1}\ O\ r = r$
⟨*proof*⟩

Second half.

**lemma** *comp-equivI*:
**assumes** $r^{-1}\ O\ r = r$ $Domain\ r = A$
**shows** $equiv\ A\ r$
⟨*proof*⟩

## 38.2 Equivalence classes

**lemma** *equiv-class-subset*: $equiv\ A\ r \Longrightarrow (a,\ b) \in r \Longrightarrow r``\{a\} \subseteq r``\{b\}$
— lemma for the next result
⟨*proof*⟩

**theorem** *equiv-class-eq*:
**assumes** $equiv\ A\ r$ **and** $(a,\ b) \in r$
**shows** $r``\{a\} = r``\{b\}$
⟨*proof*⟩

**lemma** *equiv-class-self*: $equiv\ A\ r \Longrightarrow a \in A \Longrightarrow a \in r``\{a\}$
⟨*proof*⟩

**lemma** *subset-equiv-class*: $equiv\ A\ r \Longrightarrow r``\{b\} \subseteq r``\{a\} \Longrightarrow b \in A \Longrightarrow (a,\ b) \in r$
— lemma for the next result
⟨*proof*⟩

**lemma** *eq-equiv-class*: $r``\{a\} = r``\{b\} \Longrightarrow equiv\ A\ r \Longrightarrow b \in A \Longrightarrow (a,\ b) \in r$
⟨*proof*⟩

**lemma** *equiv-class-nondisjoint*: $equiv\ A\ r \Longrightarrow x \in (r``\{a\} \cap r``\{b\}) \Longrightarrow (a,\ b) \in r$
⟨*proof*⟩

**lemma** *equiv-type*: $equiv\ A\ r \Longrightarrow r \subseteq A \times A$
⟨*proof*⟩

**lemma** *equiv-class-eq-iff*: $equiv\ A\ r \Longrightarrow (x,\ y) \in r \longleftrightarrow r``\{x\} = r``\{y\} \wedge x \in A \wedge y \in A$
⟨*proof*⟩

**lemma** *eq-equiv-class-iff*: $equiv\ A\ r \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow r``\{x\} = r``\{y\} \longleftrightarrow (x, y) \in r$
⟨*proof*⟩

**lemma** *disjnt-equiv-class*: $equiv\ A\ r \Longrightarrow disjnt\ (r``\{a\})\ (r``\{b\}) \longleftrightarrow (a, b) \notin r$
⟨*proof*⟩

## 38.3 Quotients

**definition** *quotient* :: $'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow 'a\ set\ set$ (**infixl** ‹'/'/› *90*)
**where** $A//r = (\bigcup x \in A.\ \{r``\{x\}\})$ — set of equiv classes

**lemma** *quotientI*: $x \in A \Longrightarrow r``\{x\} \in A//r$
⟨*proof*⟩

**lemma** *quotientE*: $X \in A//r \Longrightarrow (\bigwedge x.\ X = r``\{x\} \Longrightarrow x \in A \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *Union-quotient*: $equiv\ A\ r \Longrightarrow \bigcup(A//r) = A$
⟨*proof*⟩

**lemma** *quotient-disj-strong*:
**assumes** $r \subseteq A \times A$ **and** *sym-on* $A\ r$ **and** *trans-on* $A\ r$ **and** $X \in A//r$ **and** $Y \in A//r$
**shows** $X = Y \vee X \cap Y = \{\}$
⟨*proof*⟩

**lemma** *quotient-disj*: $equiv\ A\ r \Longrightarrow X \in A//r \Longrightarrow Y \in A//r \Longrightarrow X = Y \vee X \cap Y = \{\}$
⟨*proof*⟩

**lemma** *quotient-eqI*:
**assumes** $equiv\ A\ r\ X \in A//r\ Y \in A//r$ **and** *xy*: $x \in X\ y \in Y\ (x, y) \in r$
**shows** $X = Y$
⟨*proof*⟩

**lemma** *quotient-eq-iff*:
**assumes** $equiv\ A\ r\ X \in A//r\ Y \in A//r$ **and** *xy*: $x \in X\ y \in Y$
**shows** $X = Y \longleftrightarrow (x, y) \in r$
⟨*proof*⟩

**lemma** *eq-equiv-class-iff2*: $equiv\ A\ r \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow \{x\}//r = \{y\}//r \longleftrightarrow (x, y) \in r$
⟨*proof*⟩

**lemma** *quotient-empty* [*simp*]: $\{\}//r = \{\}$
⟨*proof*⟩

**lemma** *quotient-is-empty* [*iff*]: $A//r = \{\} \longleftrightarrow A = \{\}$

⟨*proof*⟩

**lemma** *quotient-is-empty2* [*iff*]: $\{\} = A//r \longleftrightarrow A = \{\}$
⟨*proof*⟩

**lemma** *singleton-quotient*: $\{x\}//r = \{r \text{ `` } \{x\}\}$
⟨*proof*⟩

**lemma** *quotient-diff1*: *inj-on* $(\lambda a.\ \{a\}//r)\ A \Longrightarrow a \in A \Longrightarrow (A - \{a\})//r = A//r - \{a\}//r$
⟨*proof*⟩

## 38.4 Refinement of one equivalence relation WRT another

**lemma** *refines-equiv-class-eq*: $R \subseteq S \Longrightarrow$ *equiv* $A\ R \Longrightarrow$ *equiv* $A\ S \Longrightarrow R``(S``\{a\}) = S``\{a\}$
⟨*proof*⟩

**lemma** *refines-equiv-class-eq2*: $R \subseteq S \Longrightarrow$ *equiv* $A\ R \Longrightarrow$ *equiv* $A\ S \Longrightarrow S``(R``\{a\}) = S``\{a\}$
⟨*proof*⟩

**lemma** *refines-equiv-image-eq*: $R \subseteq S \Longrightarrow$ *equiv* $A\ R \Longrightarrow$ *equiv* $A\ S \Longrightarrow (\lambda X.\ S``X)\ `\ (A//R) = A//S$
⟨*proof*⟩

**lemma** *finite-refines-finite*:
*finite* $(A//R) \Longrightarrow R \subseteq S \Longrightarrow$ *equiv* $A\ R \Longrightarrow$ *equiv* $A\ S \Longrightarrow$ *finite* $(A//S)$
⟨*proof*⟩

**lemma** *finite-refines-card-le*:
*finite* $(A//R) \Longrightarrow R \subseteq S \Longrightarrow$ *equiv* $A\ R \Longrightarrow$ *equiv* $A\ S \Longrightarrow$ *card* $(A//S) \le$ *card* $(A//R)$
⟨*proof*⟩

## 38.5 Defining unary operations upon equivalence classes

A congruence-preserving function.

**definition** *congruent* :: $('a \times 'a)$ *set* $\Rightarrow ('a \Rightarrow 'b) \Rightarrow$ *bool*
**where** *congruent* $r\ f \longleftrightarrow (\forall (y, z) \in r.\ f\ y = f\ z)$

**lemma** *congruentI*: $(\bigwedge y\ z.\ (y, z) \in r \Longrightarrow f\ y = f\ z) \Longrightarrow$ *congruent* $r\ f$
⟨*proof*⟩

**lemma** *congruentD*: *congruent* $r\ f \Longrightarrow (y, z) \in r \Longrightarrow f\ y = f\ z$
⟨*proof*⟩

**abbreviation** *RESPECTS* :: $('a \Rightarrow 'b) \Rightarrow ('a \times 'a)$ *set* $\Rightarrow$ *bool* (**infixr** ‹*respects*› *80*)

**where** $f$ *respects* $r \equiv$ *congruent* $r\ f$

**lemma** *UN-constant-eq*: $a \in A \Longrightarrow \forall\, y \in A.\ f\ y = c \Longrightarrow (\bigcup y \in A.\ f\ y) = c$
— lemma required to prove *UN-equiv-class*
⟨*proof*⟩

**lemma** *UN-equiv-class*:
**assumes** *equiv* $A\ r\ f$ *respects* $r\ a \in A$
**shows** $(\bigcup x \in r``\{a\}.\ f\ x) = f\ a$
— Conversion rule
⟨*proof*⟩

**lemma** *UN-equiv-class-type*:
**assumes** $r$: *equiv* $A\ r\ f$ *respects* $r$ **and** $X$: $X \in A//r$ **and** $AB$: $\bigwedge x.\ x \in A \Longrightarrow f\ x \in B$
**shows** $(\bigcup x \in X.\ f\ x) \in B$
⟨*proof*⟩

Sufficient conditions for injectiveness. Could weaken premises! major premise could be an inclusion; *bcong* could be $\bigwedge y.\ y \in A \Longrightarrow f\ y \in B$.

**lemma** *UN-equiv-class-inject*:
**assumes** *equiv* $A\ r\ f$ *respects* $r$
**and** *eq*: $(\bigcup x \in X.\ f\ x) = (\bigcup y \in Y.\ f\ y)$
**and** $X$: $X \in A//r$ **and** $Y$: $Y \in A//r$
**and** *fr*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow f\ x = f\ y \Longrightarrow (x, y) \in r$
**shows** $X = Y$
⟨*proof*⟩

## 38.6 Defining binary operations upon equivalence classes

A congruence-preserving function of two arguments.

**definition** *congruent2* :: $('a \times 'a)$ *set* $\Rightarrow ('b \times 'b)$ *set* $\Rightarrow ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow$ *bool*
**where** *congruent2* $r1\ r2\ f \longleftrightarrow (\forall\,(y1, z1) \in r1.\ \forall\,(y2, z2) \in r2.\ f\ y1\ y2 = f\ z1\ z2)$

**lemma** *congruent2I'*:
**assumes** $\bigwedge y1\ z1\ y2\ z2.\ (y1, z1) \in r1 \Longrightarrow (y2, z2) \in r2 \Longrightarrow f\ y1\ y2 = f\ z1\ z2$
**shows** *congruent2* $r1\ r2\ f$
⟨*proof*⟩

**lemma** *congruent2D*: *congruent2* $r1\ r2\ f \Longrightarrow (y1, z1) \in r1 \Longrightarrow (y2, z2) \in r2 \Longrightarrow f\ y1\ y2 = f\ z1\ z2$
⟨*proof*⟩

Abbreviation for the common case where the relations are identical.

**abbreviation** *RESPECTS2*:: $('a \Rightarrow 'a \Rightarrow 'b) \Rightarrow ('a \times 'a)$ *set* $\Rightarrow$ *bool* (**infixr** ‹*respects2*› *80*)

**where** $f$ *respects2* $r \equiv$ *congruent2* $r\ r\ f$

**lemma** *congruent2-implies-congruent*:
*equiv* $A\ r1 \Longrightarrow$ *congruent2* $r1\ r2\ f \Longrightarrow a \in A \Longrightarrow$ *congruent* $r2\ (f\ a)$
⟨*proof*⟩

**lemma** *congruent2-implies-congruent-UN*:
**assumes** *equiv* $A1\ r1$ *equiv* $A2\ r2$ *congruent2* $r1\ r2\ f\ a \in A2$
**shows** *congruent* $r1\ (\lambda x1.\ \bigcup x2 \in r2``\{a\}.\ f\ x1\ x2)$
⟨*proof*⟩

**lemma** *UN-equiv-class2*:
*equiv* $A1\ r1 \Longrightarrow$ *equiv* $A2\ r2 \Longrightarrow$ *congruent2* $r1\ r2\ f \Longrightarrow a1 \in A1 \Longrightarrow a2 \in A2$
$\Longrightarrow$
$(\bigcup x1 \in r1``\{a1\}.\ \bigcup x2 \in r2``\{a2\}.\ f\ x1\ x2) = f\ a1\ a2$
⟨*proof*⟩

**lemma** *UN-equiv-class-type2*:
*equiv* $A1\ r1 \Longrightarrow$ *equiv* $A2\ r2 \Longrightarrow$ *congruent2* $r1\ r2\ f$
$\Longrightarrow X1 \in A1//r1 \Longrightarrow X2 \in A2//r2$
$\Longrightarrow (\bigwedge x1\ x2.\ x1 \in A1 \Longrightarrow x2 \in A2 \Longrightarrow f\ x1\ x2 \in B)$
$\Longrightarrow (\bigcup x1 \in X1.\ \bigcup x2 \in X2.\ f\ x1\ x2) \in B$
⟨*proof*⟩

**lemma** *UN-UN-split-split-eq*:
$(\bigcup (x1,\ x2) \in X.\ \bigcup (y1,\ y2) \in Y.\ A\ x1\ x2\ y1\ y2) =$
$(\bigcup x \in X.\ \bigcup y \in Y.\ (\lambda(x1,\ x2).\ (\lambda(y1,\ y2).\ A\ x1\ x2\ y1\ y2)\ y)\ x)$
— Allows a natural expression of binary operators,
— without explicit calls to *split*
⟨*proof*⟩

**lemma** *congruent2I*:
*equiv* $A1\ r1 \Longrightarrow$ *equiv* $A2\ r2$
$\Longrightarrow (\bigwedge y\ z\ w.\ w \in A2 \Longrightarrow (y,z) \in r1 \Longrightarrow f\ y\ w = f\ z\ w)$
$\Longrightarrow (\bigwedge y\ z\ w.\ w \in A1 \Longrightarrow (y,z) \in r2 \Longrightarrow f\ w\ y = f\ w\ z)$
$\Longrightarrow$ *congruent2* $r1\ r2\ f$
— Suggested by John Harrison – the two subproofs may be
— *much* simpler than the direct proof.
⟨*proof*⟩

**lemma** *congruent2-commuteI*:
**assumes** *equivA*: *equiv* $A\ r$
**and** *commute*: $\bigwedge y\ z.\ y \in A \Longrightarrow z \in A \Longrightarrow f\ y\ z = f\ z\ y$
**and** *congt*: $\bigwedge y\ z\ w.\ w \in A \Longrightarrow (y,z) \in r \Longrightarrow f\ w\ y = f\ w\ z$
**shows** $f$ *respects2* $r$
⟨*proof*⟩

## 38.7 Quotients and finiteness

Suggested by Florian Kammüller

**lemma** *finite-quotient*:
**assumes** *finite* $A$ $r \subseteq A \times A$
**shows** *finite* $(A//r)$
— recall *equiv* $?A$ $?r \Longrightarrow ?r \subseteq ?A \times ?A$
⟨*proof*⟩

**lemma** *finite-equiv-class*: *finite* $A \Longrightarrow r \subseteq A \times A \Longrightarrow X \in A//r \Longrightarrow$ *finite* $X$
⟨*proof*⟩

**lemma** *equiv-imp-dvd-card*:
**assumes** *finite* $A$ *equiv* $A$ $r$ $\bigwedge X.\ X \in A//r \Longrightarrow k$ *dvd card* $X$
**shows** $k$ *dvd card* $A$
⟨*proof*⟩

## 38.8 Kernel of a Function

**definition** *kernel* :: $('a \Rightarrow 'b) \Rightarrow ('a * 'a)$ *set* **where**
*kernel* $f = \{(x,y).\ f\ x = f\ y\}$

**lemma** *equiv-kernel*: *equiv UNIV* (*kernel* $f$)
⟨*proof*⟩

**lemma** *respects-kernel*: $f$ *respects* (*kernel* $f$)
⟨*proof*⟩

**lemma** *inj-on-vimage-image*: *inj-on* $(\lambda b.\ f\ -\text{`}\ \{b\})\ (f\ \text{`}\ A)$
⟨*proof*⟩

**lemma** *kernel-Image*: *kernel* $f\ \text{``}\ A = f\ -\text{`}\ (f\ \text{`}\ A)$
⟨*proof*⟩

**lemma** *quotient-kernel-eq-image*: $A$ // *kernel* $f = (\lambda b.\ f\ -\text{`}\ \{b\})\ \text{`}\ f\ \text{`}\ A$
⟨*proof*⟩

**lemma** *bij-betw-image-quotient-kernel*: *bij-betw* $(\lambda b.\ f\ -\text{`}\ \{b\})\ (f\ \text{`}\ A)$ $(A$ // *kernel* $f)$
⟨*proof*⟩

## 38.9 Projection

**definition** *proj* :: $('b \times 'a)$ *set* $\Rightarrow 'b \Rightarrow 'a$ *set*
**where** *proj* $r\ x = r\ \text{``}\ \{x\}$

**lemma** *proj-preserves*: $x \in A \Longrightarrow$ *proj* $r\ x \in A//r$
⟨*proof*⟩

**lemma** *proj-in-iff*:
**assumes** *equiv A r*
**shows** $proj\ r\ x \in A//r \longleftrightarrow x \in A$
(**is** $?lhs \longleftrightarrow ?rhs$)
⟨*proof*⟩

**lemma** *proj-iff*: $equiv\ A\ r \Longrightarrow \{x,\ y\} \subseteq A \Longrightarrow proj\ r\ x = proj\ r\ y \longleftrightarrow (x,\ y) \in r$
⟨*proof*⟩

**lemma** *proj-image*: $proj\ r\ `\ A = A//r$
⟨*proof*⟩

**lemma** *in-quotient-imp-non-empty*: $equiv\ A\ r \Longrightarrow X \in A//r \Longrightarrow X \neq \{\}$
⟨*proof*⟩

**lemma** *in-quotient-imp-in-rel*: $equiv\ A\ r \Longrightarrow X \in A//r \Longrightarrow \{x,\ y\} \subseteq X \Longrightarrow (x,\ y) \in r$
⟨*proof*⟩

**lemma** *in-quotient-imp-closed*: $equiv\ A\ r \Longrightarrow X \in A//r \Longrightarrow x \in X \Longrightarrow (x,\ y) \in r \Longrightarrow y \in X$
⟨*proof*⟩

**lemma** *in-quotient-imp-subset*: $equiv\ A\ r \Longrightarrow X \in A//r \Longrightarrow X \subseteq A$
⟨*proof*⟩

## 38.10 Equivalence relations – predicate version

Partial equivalences.

**definition** *part-equivp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
**where** $part\text{-}equivp\ R \longleftrightarrow (\exists x.\ R\ x\ x) \wedge (\forall x\ y.\ R\ x\ y \longleftrightarrow R\ x\ x \wedge R\ y\ y \wedge R\ x = R\ y)$
— John-Harrison-style characterization

**lemma** *part-equivpI*: $\exists x.\ R\ x\ x \Longrightarrow symp\ R \Longrightarrow transp\ R \Longrightarrow part\text{-}equivp\ R$
⟨*proof*⟩

**lemma** *part-equivpE*:
**assumes** *part-equivp R*
**obtains** *x* **where** *R x x* **and** *symp R* **and** *transp R*
⟨*proof*⟩

**lemma** *part-equivp-refl-symp-transp*: $part\text{-}equivp\ R \longleftrightarrow (\exists x.\ R\ x\ x) \wedge symp\ R \wedge transp\ R$
⟨*proof*⟩

**lemma** *part-equivp-symp*: $part\text{-}equivp\ R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$

⟨*proof*⟩

**lemma** *part-equivp-transp*: $\mathit{part\text{-}equivp}\ R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

**lemma** *part-equivp-typedef*: $\mathit{part\text{-}equivp}\ R \Longrightarrow \exists\, d.\ d \in \{c.\ \exists\, x.\ R\ x\ x \wedge c = \mathit{Collect}\ (R\ x)\}$
⟨*proof*⟩

Total equivalences.

**definition** *equivp* :: $('a \Rightarrow 'a \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool}$
**where** $\mathit{equivp}\ R \longleftrightarrow (\forall\, x\ y.\ R\ x\ y = (R\ x = R\ y))$ — John-Harrison-style characterization

**lemma** *equivpI*: $\mathit{reflp}\ R \Longrightarrow \mathit{symp}\ R \Longrightarrow \mathit{transp}\ R \Longrightarrow \mathit{equivp}\ R$
⟨*proof*⟩

**lemma** *equivpE*:
**assumes** *equivp R*
**obtains** *reflp R* **and** *symp R* **and** *transp R*
⟨*proof*⟩

**lemma** *equivp-implies-part-equivp*: $\mathit{equivp}\ R \Longrightarrow \mathit{part\text{-}equivp}\ R$
⟨*proof*⟩

**lemma** *equivp-equiv*: $\mathit{equiv}\ \mathit{UNIV}\ A \longleftrightarrow \mathit{equivp}\ (\lambda x\ y.\ (x,\ y) \in A)$
⟨*proof*⟩

**lemma** *equivp-reflp-symp-transp*: $\mathit{equivp}\ R \longleftrightarrow \mathit{reflp}\ R \wedge \mathit{symp}\ R \wedge \mathit{transp}\ R$
⟨*proof*⟩

**lemma** *identity-equivp*: *equivp* $(=)$
⟨*proof*⟩

**lemma** *equivp-reflp*: $\mathit{equivp}\ R \Longrightarrow R\ x\ x$
⟨*proof*⟩

**lemma** *equivp-symp*: $\mathit{equivp}\ R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ x$
⟨*proof*⟩

**lemma** *equivp-transp*: $\mathit{equivp}\ R \Longrightarrow R\ x\ y \Longrightarrow R\ y\ z \Longrightarrow R\ x\ z$
⟨*proof*⟩

**lemma** *equivp-rtranclp*: $\mathit{symp}\ r \Longrightarrow \mathit{equivp}\ r^{**}$
⟨*proof*⟩

**lemmas** *equivp-rtranclp-symclp* [*simp*] = *equivp-rtranclp*[*OF symp-on-symclp*]

**lemma** *equivp-vimage2p*: $\mathit{equivp}\ R \Longrightarrow \mathit{equivp}\ (\mathit{vimage2p}\ f\ f\ R)$

⟨*proof*⟩

**lemma** *equivp-imp-transp*: $equivp\ R \Longrightarrow transp\ R$
⟨*proof*⟩

## 38.11 Equivalence closure

**definition** *equivclp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ **where**
$equivclp\ r = (symclp\ r)^{**}$

**lemma** *transp-equivclp* [*simp*]: $transp\ (equivclp\ r)$
⟨*proof*⟩

**lemma** *reflp-equivclp* [*simp*]: $reflp\ (equivclp\ r)$
⟨*proof*⟩

**lemma** *symp-equivclp* [*simp*]: $symp\ (equivclp\ r)$
⟨*proof*⟩

**lemma** *equivp-evquivclp* [*simp*]: $equivp\ (equivclp\ r)$
⟨*proof*⟩

**lemma** *tranclp-equivclp* [*simp*]: $(equivclp\ r)^{++} = equivclp\ r$
⟨*proof*⟩

**lemma** *rtranclp-equivclp* [*simp*]: $(equivclp\ r)^{**} = equivclp\ r$
⟨*proof*⟩

**lemma** *symclp-equivclp* [*simp*]: $symclp\ (equivclp\ r) = equivclp\ r$
⟨*proof*⟩

**lemma** *equivclp-symclp* [*simp*]: $equivclp\ (symclp\ r) = equivclp\ r$
⟨*proof*⟩

**lemma** *equivclp-conversep* [*simp*]: $equivclp\ (conversep\ r) = equivclp\ r$
⟨*proof*⟩

**lemma** *equivclp-sym* [*sym*]: $equivclp\ r\ x\ y \Longrightarrow equivclp\ r\ y\ x$
⟨*proof*⟩

**lemma** *equivclp-OO-equivclp-le-equivclp*: $equivclp\ r\ OO\ equivclp\ r \leq equivclp\ r$
⟨*proof*⟩

**lemma** *rtranlcp-le-equivclp*: $r^{**} \leq equivclp\ r$
⟨*proof*⟩

**lemma** *rtranclp-conversep-le-equivclp*: $r^{-1-1**} \leq equivclp\ r$
⟨*proof*⟩

**lemma** *symclp-rtranclp-le-equivclp*: $symclp\ r^{**} \leq equivclp\ r$
⟨*proof*⟩

**lemma** *r-OO-conversep-into-equivclp*:
$r^{**}\ OO\ r^{-1-1**} \leq equivclp\ r$
⟨*proof*⟩

**lemma** *equivclp-induct* [*consumes 1*, *case-names base step*, *induct pred*: *equivclp*]:
**assumes** *a*: $equivclp\ r\ a\ b$
**and** *cases*: $P\ a\ \bigwedge y\ z.\ equivclp\ r\ a\ y \Longrightarrow r\ y\ z \vee r\ z\ y \Longrightarrow P\ y \Longrightarrow P\ z$
**shows** $P\ b$
⟨*proof*⟩

**lemma** *converse-equivclp-induct* [*consumes 1*, *case-names base step*]:
**assumes** *major*: $equivclp\ r\ a\ b$
**and** *cases*: $P\ b\ \bigwedge y\ z.\ r\ y\ z \vee r\ z\ y \Longrightarrow equivclp\ r\ z\ b \Longrightarrow P\ z \Longrightarrow P\ y$
**shows** $P\ a$
⟨*proof*⟩

**lemma** *equivclp-refl* [*simp*]: $equivclp\ r\ x\ x$
⟨*proof*⟩

**lemma** *r-into-equivclp* [*intro*]: $r\ x\ y \Longrightarrow equivclp\ r\ x\ y$
⟨*proof*⟩

**lemma** *converse-r-into-equivclp* [*intro*]: $r\ y\ x \Longrightarrow equivclp\ r\ x\ y$
⟨*proof*⟩

**lemma** *rtranclp-into-equivclp*: $r^{**}\ x\ y \Longrightarrow equivclp\ r\ x\ y$
⟨*proof*⟩

**lemma** *converse-rtranclp-into-equivclp*: $r^{**}\ y\ x \Longrightarrow equivclp\ r\ x\ y$
⟨*proof*⟩

**lemma** *equivclp-into-equivclp*: $[\![\ equivclp\ r\ a\ b;\ r\ b\ c \vee r\ c\ b\ ]\!] \Longrightarrow equivclp\ r\ a\ c$
⟨*proof*⟩

**lemma** *equivclp-trans* [*trans*]: $[\![\ equivclp\ r\ a\ b;\ equivclp\ r\ b\ c\ ]\!] \Longrightarrow equivclp\ r\ a\ c$
⟨*proof*⟩

**hide-const** (**open**) *proj*

**end**

**theory** *Basic-BNF-LFPs*
**imports** *BNF-Least-Fixpoint*
**begin**

**definition** *xtor* :: $'a \Rightarrow 'a$ **where**
*xtor* $x = x$

**lemma** *xtor-map*: $f$ (*xtor* $x$) = *xtor* ($f\,x$)
⟨*proof*⟩

**lemma** *xtor-map-unique*: $u \circ$ *xtor* = *xtor* $\circ f \Longrightarrow u = f$
⟨*proof*⟩

**lemma** *xtor-set*: $f$ (*xtor* $x$) = $f\,x$
⟨*proof*⟩

**lemma** *xtor-rel*: $R$ (*xtor* $x$) (*xtor* $y$) = $R\ x\ y$
⟨*proof*⟩

**lemma** *xtor-induct*: ($\bigwedge x$. $P$ (*xtor* $x$)) $\Longrightarrow P\ z$
⟨*proof*⟩

**lemma** *xtor-xtor*: *xtor* (*xtor* $x$) = $x$
⟨*proof*⟩

**lemmas** *xtor-inject* = *xtor-rel*[*of* (=)]

**lemma** *xtor-rel-induct*: ($\bigwedge x\ y$. *vimage2p id-bnf id-bnf* $R\ x\ y \Longrightarrow IR$ (*xtor* $x$) (*xtor* $y$)) $\Longrightarrow R \leq IR$
⟨*proof*⟩

**lemma** *xtor-iff-xtor*: $u$ = *xtor* $w \longleftrightarrow$ *xtor* $u = w$
⟨*proof*⟩

**lemma** *Inl-def-alt*: *Inl* ≡ ($\lambda a$. *xtor* (*id-bnf* (*Inl* $a$)))
⟨*proof*⟩

**lemma** *Inr-def-alt*: *Inr* ≡ ($\lambda a$. *xtor* (*id-bnf* (*Inr* $a$)))
⟨*proof*⟩

**lemma** *Pair-def-alt*: *Pair* ≡ ($\lambda a\ b$. *xtor* (*id-bnf* ($a$, $b$)))
⟨*proof*⟩

**definition** *ctor-rec* :: $'a \Rightarrow 'a$ **where**
*ctor-rec* $x = x$

**lemma** *ctor-rec*: $g$ = *id* $\Longrightarrow$ *ctor-rec* $f$ (*xtor* $x$) = $f$ ((*id-bnf* $\circ\ g \circ$ *id-bnf*) $x$)
⟨*proof*⟩

**lemma** *ctor-rec-unique*: $g$ = *id* $\Longrightarrow f \circ$ *xtor* = $s \circ$ (*id-bnf* $\circ\ g \circ$ *id-bnf*) $\Longrightarrow f$ = *ctor-rec* $s$
⟨*proof*⟩

**lemma** *ctor-rec-def-alt*: $f = ctor\text{-}rec\ (f \circ id\text{-}bnf)$
⟨*proof*⟩

**lemma** *ctor-rec-o-map*: $ctor\text{-}rec\ f \circ g = ctor\text{-}rec\ (f \circ (id\text{-}bnf \circ g \circ id\text{-}bnf))$
⟨*proof*⟩

**lemma** *ctor-rec-transfer*: $rel\text{-}fun\ (rel\text{-}fun\ (vimage2p\ id\text{-}bnf\ id\text{-}bnf\ R)\ S)\ (rel\text{-}fun\ R\ S)\ ctor\text{-}rec\ ctor\text{-}rec$
⟨*proof*⟩

**lemma** *eq-fst-iff*: $a = fst\ p \longleftrightarrow (\exists\, b.\ p = (a,\ b))$
⟨*proof*⟩

**lemma** *eq-snd-iff*: $b = snd\ p \longleftrightarrow (\exists\, a.\ p = (a,\ b))$
⟨*proof*⟩

**lemma** *ex-neg-all-pos*: $((\exists\, x.\ P\ x) \Longrightarrow Q) \equiv (\bigwedge x.\ P\ x \Longrightarrow Q)$
⟨*proof*⟩

**lemma** *hypsubst-in-prems*: $(\bigwedge x.\ y = x \Longrightarrow z = f\ x \Longrightarrow P) \equiv (z = f\ y \Longrightarrow P)$
⟨*proof*⟩

**lemma** *isl-map-sum*:
$isl\ (map\text{-}sum\ f\ g\ s) = isl\ s$
⟨*proof*⟩

**lemma** *map-sum-sel*:
$isl\ s \Longrightarrow projl\ (map\text{-}sum\ f\ g\ s) = f\ (projl\ s)$
$\neg\ isl\ s \Longrightarrow projr\ (map\text{-}sum\ f\ g\ s) = g\ (projr\ s)$
⟨*proof*⟩

**lemma** *set-sum-sel*:
$isl\ s \Longrightarrow projl\ s \in setl\ s$
$\neg\ isl\ s \Longrightarrow projr\ s \in setr\ s$
⟨*proof*⟩

**lemma** *rel-sum-sel*: $rel\text{-}sum\ R1\ R2\ a\ b = (isl\ a = isl\ b\ \wedge$
$(isl\ a \longrightarrow isl\ b \longrightarrow R1\ (projl\ a)\ (projl\ b))\ \wedge$
$(\neg\ isl\ a \longrightarrow \neg\ isl\ b \longrightarrow R2\ (projr\ a)\ (projr\ b)))$
⟨*proof*⟩

**lemma** *isl-transfer*: $rel\text{-}fun\ (rel\text{-}sum\ A\ B)\ (=)\ isl\ isl$
⟨*proof*⟩

**lemma** *rel-prod-sel*: $rel\text{-}prod\ R1\ R2\ p\ q = (R1\ (fst\ p)\ (fst\ q)\ \wedge\ R2\ (snd\ p)\ (snd\ q))$
⟨*proof*⟩

⟨*ML*⟩

**declare** *prod.size* [*no-atp*]

**hide-const** (**open**) *xtor ctor-rec*

**hide-fact** (**open**)
*xtor-def xtor-map xtor-set xtor-rel xtor-induct xtor-xtor xtor-inject ctor-rec-def ctor-rec*
*ctor-rec-def-alt ctor-rec-o-map xtor-rel-induct Inl-def-alt Inr-def-alt Pair-def-alt*

**end**

# 39 MESON Proof Method

**theory** *Meson*
**imports** *Nat*
**begin**

## 39.1 Negation Normal Form

de Morgan laws

**lemma** *not-conjD*: $\neg(P \wedge Q) \Longrightarrow \neg P \vee \neg Q$
**and** *not-disjD*: $\neg(P \vee Q) \Longrightarrow \neg P \wedge \neg Q$
**and** *not-notD*: $\neg\neg P \Longrightarrow P$
**and** *not-allD*: $\bigwedge P.\ \neg(\forall x.\ P(x)) \Longrightarrow \exists x.\ \neg P(x)$
**and** *not-exD*: $\bigwedge P.\ \neg(\exists x.\ P(x)) \Longrightarrow \forall x.\ \neg P(x)$
⟨*proof*⟩

Removal of $\longrightarrow$ and $\longleftrightarrow$ (positive and negative occurrences)

**lemma** *imp-to-disjD*: $P \longrightarrow Q \Longrightarrow \neg P \vee Q$
**and** *not-impD*: $\neg(P \longrightarrow Q) \Longrightarrow P \wedge \neg Q$
**and** *iff-to-disjD*: $P=Q \Longrightarrow (\neg P \vee Q) \wedge (\neg Q \vee P)$
**and** *not-iffD*: $\neg(P=Q) \Longrightarrow (P \vee Q) \wedge (\neg P \vee \neg Q)$
— Much more efficient than $P \wedge \neg\ Q \vee Q \wedge \neg\ P$ for computing CNF
**and** *not-refl-disj-D*: $x \neq x \vee P \Longrightarrow P$
⟨*proof*⟩

## 39.2 Pulling out the existential quantifiers

Conjunction

**lemma** *conj-exD1*: $\bigwedge P\ Q.\ (\exists x.\ P(x)) \wedge Q \Longrightarrow \exists x.\ P(x) \wedge Q$
**and** *conj-exD2*: $\bigwedge P\ Q.\ P \wedge (\exists x.\ Q(x)) \Longrightarrow \exists x.\ P \wedge Q(x)$
⟨*proof*⟩

Disjunction

**lemma** *disj-exD*: $\bigwedge P\ Q.\ (\exists x.\ P(x)) \vee (\exists x.\ Q(x)) \Longrightarrow \exists x.\ P(x) \vee Q(x)$
— DO NOT USE with forall-Skolemization: makes fewer schematic variables!!

— With ex-Skolemization, makes fewer Skolem constants
**and** *disj-exD1*: $\bigwedge P\ Q.\ (\exists\, x.\ P(x)) \lor Q \Longrightarrow \exists\, x.\ P(x) \lor Q$
**and** *disj-exD2*: $\bigwedge P\ Q.\ P \lor (\exists\, x.\ Q(x)) \Longrightarrow \exists\, x.\ P \lor Q(x)$
⟨*proof*⟩

**lemma** *disj-assoc*: $(P \lor Q) \lor R \Longrightarrow P \lor (Q \lor R)$
**and** *disj-comm*: $P \lor Q \Longrightarrow Q \lor P$
**and** *disj-FalseD1*: $\mathit{False} \lor P \Longrightarrow P$
**and** *disj-FalseD2*: $P \lor \mathit{False} \Longrightarrow P$
⟨*proof*⟩

Generation of contrapositives

Inserts negated disjunct after removing the negation; P is a literal. Model elimination requires assuming the negation of every attempted subgoal, hence the negated disjuncts.

**lemma** *make-neg-rule*: $\neg P \lor Q \Longrightarrow ((\neg P \Longrightarrow P) \Longrightarrow Q)$
⟨*proof*⟩

Version for Plaisted's "Postive refinement" of the Meson procedure

**lemma** *make-refined-neg-rule*: $\neg P \lor Q \Longrightarrow (P \Longrightarrow Q)$
⟨*proof*⟩

$P$ should be a literal

**lemma** *make-pos-rule*: $P \lor Q \Longrightarrow ((P \Longrightarrow \neg P) \Longrightarrow Q)$
⟨*proof*⟩

Versions of *make-neg-rule* and *make-pos-rule* that don't insert new assumptions, for ordinary resolution.

**lemmas** *make-neg-rule′* = *make-refined-neg-rule*

**lemma** *make-pos-rule′*: $[\![P \lor Q;\ \neg P]\!] \Longrightarrow Q$
⟨*proof*⟩

Generation of a goal clause – put away the final literal

**lemma** *make-neg-goal*: $\neg P \Longrightarrow ((\neg P \Longrightarrow P) \Longrightarrow \mathit{False})$
⟨*proof*⟩

**lemma** *make-pos-goal*: $P \Longrightarrow ((P \Longrightarrow \neg P) \Longrightarrow \mathit{False})$
⟨*proof*⟩

## 39.3 Lemmas for Forward Proof

There is a similarity to congruence rules. They are also useful in ordinary proofs.

**lemma** *conj-forward*: $[\![P' \land Q';\ \ P' \Longrightarrow P;\ \ Q' \Longrightarrow Q\ ]\!] \Longrightarrow P \land Q$
⟨*proof*⟩

**lemma** *disj-forward*: $[\![P' \vee Q';\ \ P' \Longrightarrow P;\ \ Q' \Longrightarrow Q\ ]\!] \Longrightarrow P \vee Q$
$\langle proof \rangle$

**lemma** *imp-forward*: $[\![P' \longrightarrow Q';\ \ P \Longrightarrow P';\ \ Q' \Longrightarrow Q\ ]\!] \Longrightarrow P \longrightarrow Q$
$\langle proof \rangle$

**lemma** *imp-forward2*: $[\![P' \longrightarrow Q';\ \ P \Longrightarrow P';\ \ P' \Longrightarrow Q' \Longrightarrow Q\ ]\!] \Longrightarrow P \longrightarrow Q$
$\langle proof \rangle$

**lemma** *disj-forward2*: $[\![\ P' \vee Q';\ \ P' \Longrightarrow P;\ \ [\![Q';\ P \Longrightarrow False]\!] \Longrightarrow Q]\!] \Longrightarrow P \vee Q$
$\langle proof \rangle$

**lemma** *all-forward*: $[|\ \forall x.\ P'(x);\ \ !!x.\ P'(x) ==> P(x)\ |] ==> \forall x.\ P(x)$
$\langle proof \rangle$

**lemma** *ex-forward*: $[|\ \exists x.\ P'(x);\ \ !!x.\ P'(x) ==> P(x)\ |] ==> \exists x.\ P(x)$
$\langle proof \rangle$

## 39.4 Clausification helper

**lemma** *TruepropI*: $P \equiv Q \Longrightarrow Trueprop\ P \equiv Trueprop\ Q$
$\langle proof \rangle$

**lemma** *ext-cong-neq*: $F\ g \neq F\ h \Longrightarrow F\ g \neq F\ h \wedge (\exists x.\ g\ x \neq h\ x)$
$\langle proof \rangle$

Combinator translation helpers

**definition** $COMBI :: 'a \Rightarrow 'a$ **where**
$COMBI\ P = P$

**definition** $COMBK :: 'a \Rightarrow 'b \Rightarrow 'a$ **where**
$COMBK\ P\ Q = P$

**definition** $COMBB :: ('b => 'c) \Rightarrow ('a => 'b) \Rightarrow 'a \Rightarrow 'c$ **where**
$COMBB\ P\ Q\ R = P\ (Q\ R)$

**definition** $COMBC :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'b \Rightarrow 'a \Rightarrow 'c$ **where**
$COMBC\ P\ Q\ R = P\ R\ Q$

**definition** $COMBS :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'c$ **where**
$COMBS\ P\ Q\ R = P\ R\ (Q\ R)$

**lemma** *abs-S*: $\lambda x.\ (f\ x)\ (g\ x) \equiv COMBS\ f\ g$
$\langle proof \rangle$

**lemma** *abs-I*: $\lambda x.\ x \equiv COMBI$
$\langle proof \rangle$

**lemma** *abs-K*: $\lambda x.\ y \equiv COMBK\ y$
⟨*proof*⟩

**lemma** *abs-B*: $\lambda x.\ a\ (g\ x) \equiv COMBB\ a\ g$
⟨*proof*⟩

**lemma** *abs-C*: $\lambda x.\ (f\ x)\ b \equiv COMBC\ f\ b$
⟨*proof*⟩

### 39.5 Skolemization helpers

**definition** *skolem* :: $'a \Rightarrow 'a$ **where**
$skolem = (\lambda x.\ x)$

**lemma** *skolem-COMBK-iff*: $P \longleftrightarrow skolem\ (COMBK\ P\ (i::nat))$
⟨*proof*⟩

**lemmas** *skolem-COMBK-I* = *iffD1* [*OF skolem-COMBK-iff*]

### 39.6 Meson package

⟨*ML*⟩

**hide-const** (**open**) *COMBI COMBK COMBB COMBC COMBS skolem*
**hide-fact** (**open**) *not-conjD not-disjD not-notD not-allD not-exD imp-to-disjD not-impD iff-to-disjD not-iffD not-refl-disj-D conj-exD1 conj-exD2 disj-exD disj-exD1 disj-exD2 disj-assoc disj-comm disj-FalseD1 disj-FalseD2 TruepropI ext-cong-neq COMBI-def COMBK-def COMBB-def COMBC-def COMBS-def abs-I abs-K abs-B abs-C abs-S skolem-def skolem-COMBK-iff skolem-COMBK-I*
**end**

## 40 Automatic Theorem Provers (ATPs)

**theory** *ATP*
**imports** *Meson Hilbert-Choice*
**begin**

### 40.1 ATP problems and proofs

⟨*ML*⟩

### 40.2 Higher-order reasoning helpers

**definition** *fFalse* :: *bool* **where**
*fFalse* $\longleftrightarrow$ *False*

**definition** *fTrue* :: *bool* **where**

*fTrue* $\longleftrightarrow$ *True*

**definition** *fNot* :: *bool* $\Rightarrow$ *bool* **where**
*fNot P* $\longleftrightarrow \neg$ *P*

**definition** *fComp* :: $('a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$ **where**
*fComp P* $= (\lambda x.\ \neg\ P\ x)$

**definition** *fconj* :: *bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool* **where**
*fconj P Q* $\longleftrightarrow P \wedge Q$

**definition** *fdisj* :: *bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool* **where**
*fdisj P Q* $\longleftrightarrow P \vee Q$

**definition** *fimplies* :: *bool* $\Rightarrow$ *bool* $\Rightarrow$ *bool* **where**
*fimplies P Q* $\longleftrightarrow (P \longrightarrow Q)$

**definition** *fAll* :: $('a \Rightarrow bool) \Rightarrow bool$ **where**
*fAll P* $\longleftrightarrow$ *All P*

**definition** *fEx* :: $('a \Rightarrow bool) \Rightarrow bool$ **where**
*fEx P* $\longleftrightarrow$ *Ex P*

**definition** *fequal* :: $'a \Rightarrow 'a \Rightarrow bool$ **where**
*fequal x y* $\longleftrightarrow (x = y)$

**definition** *fChoice* :: $('a \Rightarrow bool) \Rightarrow 'a$ **where**
*fChoice* $\equiv$ *Hilbert-Choice.Eps*

**lemma** *fTrue-ne-fFalse*: *fFalse* $\neq$ *fTrue*
$\langle proof \rangle$

**lemma** *fNot-table*:
*fNot fFalse* = *fTrue*
*fNot fTrue* = *fFalse*
$\langle proof \rangle$

**lemma** *fconj-table*:
*fconj fFalse P* = *fFalse*
*fconj P fFalse* = *fFalse*
*fconj fTrue fTrue* = *fTrue*
$\langle proof \rangle$

**lemma** *fdisj-table*:
*fdisj fTrue P* = *fTrue*
*fdisj P fTrue* = *fTrue*
*fdisj fFalse fFalse* = *fFalse*
$\langle proof \rangle$

**lemma** *fimplies-table*:
*fimplies P fTrue = fTrue*
*fimplies fFalse P = fTrue*
*fimplies fTrue fFalse = fFalse*
⟨*proof*⟩

**lemma** *fAll-table*:
*Ex (fComp P)* $\vee$ *fAll P = fTrue*
*All P* $\vee$ *fAll P = fFalse*
⟨*proof*⟩

**lemma** *fEx-table*:
*All (fComp P)* $\vee$ *fEx P = fTrue*
*Ex P* $\vee$ *fEx P = fFalse*
⟨*proof*⟩

**lemma** *fequal-table*:
*fequal x x = fTrue*
*x = y* $\vee$ *fequal x y = fFalse*
⟨*proof*⟩

**lemma** *fNot-law*:
*fNot P* $\neq$ *P*
⟨*proof*⟩

**lemma** *fComp-law*:
*fComp P x* $\longleftrightarrow$ $\neg$ *P x*
⟨*proof*⟩

**lemma** *fconj-laws*:
*fconj P P* $\longleftrightarrow$ *P*
*fconj P Q* $\longleftrightarrow$ *fconj Q P*
*fNot (fconj P Q)* $\longleftrightarrow$ *fdisj (fNot P) (fNot Q)*
⟨*proof*⟩

**lemma** *fdisj-laws*:
*fdisj P P* $\longleftrightarrow$ *P*
*fdisj P Q* $\longleftrightarrow$ *fdisj Q P*
*fNot (fdisj P Q)* $\longleftrightarrow$ *fconj (fNot P) (fNot Q)*
⟨*proof*⟩

**lemma** *fimplies-laws*:
*fimplies P Q* $\longleftrightarrow$ *fdisj* ($\neg$ *P*) *Q*
*fNot (fimplies P Q)* $\longleftrightarrow$ *fconj P (fNot Q)*
⟨*proof*⟩

**lemma** *fAll-law*:
*fNot (fAll R)* $\longleftrightarrow$ *fEx (fComp R)*
⟨*proof*⟩

**lemma** *fEx-law*:
*fNot* (*fEx R*) $\longleftrightarrow$ *fAll* (*fComp R*)
⟨*proof*⟩

**lemma** *fequal-laws*:
*fequal* $x$ $y$ = *fequal* $y$ $x$
*fequal* $x$ $y$ = *fFalse* $\vee$ *fequal* $y$ $z$ = *fFalse* $\vee$ *fequal* $x$ $z$ = *fTrue*
*fequal* $x$ $y$ = *fFalse* $\vee$ *fequal* ($f$ $x$) ($f$ $y$) = *fTrue*
⟨*proof*⟩

**lemma** *fChoice-iff-Ex*: $P$ (*fChoice* $P$) $\longleftrightarrow$ *HOL.Ex* $P$
⟨*proof*⟩

We use the *Ex* constant on the right-hand side of *fChoice-iff-Ex* because we want to use the TPTP-native version if fChoice is introduced in a logic that supports FOOL. In logics that don't support it, it gets replaced by fEx during processing. Notice that we cannot use $\exists\, x.\ P\ x$, as existentials are not skolimized by the metis proof method but *Ex P* is eta-expanded if FOOL is supported.

### 40.3 Basic connection between ATPs and HOL

⟨*ML*⟩

**end**

# 41 Metis Proof Method

**theory** *Metis*
**imports** *ATP*
**begin**

**context notes** [[*ML-catch-all*]]
**begin**
⟨*ML*⟩
**end**

### 41.1 Literal selection and lambda-lifting helpers

**definition** *select* :: $'a \Rightarrow 'a$ **where**
*select* = ($\lambda x.\ x$)

**lemma** *not-atomize*: ($\neg$ $A$ $\Longrightarrow$ *False*) $\equiv$ *Trueprop* $A$
⟨*proof*⟩

**lemma** *atomize-not-select*: ($A$ $\Longrightarrow$ *select False*) $\equiv$ *Trueprop* ($\neg$ $A$)
⟨*proof*⟩

**lemma** *not-atomize-select*: $(\neg\ A \Longrightarrow \mathit{select\ False}) \equiv \mathit{Trueprop}\ A$
⟨*proof*⟩

**lemma** *select-FalseI*: $\mathit{False} \Longrightarrow \mathit{select\ False}$
⟨*proof*⟩

**definition** *lambda* :: $'a \Rightarrow 'a$ **where**
$\mathit{lambda} = (\lambda x.\ x)$

**lemma** *eq-lambdaI*: $x \equiv y \Longrightarrow x \equiv \mathit{lambda}\ y$
⟨*proof*⟩

### 41.2 Metis package

⟨*ML*⟩

**hide-const** (**open**) *select fFalse fTrue fNot fComp fconj fdisj fimplies fAll fEx fequal lambda*
**hide-fact** (**open**) *select-def not-atomize atomize-not-select not-atomize-select select-FalseI*
*fFalse-def fTrue-def fNot-def fconj-def fdisj-def fimplies-def fAll-def fEx-def fequal-def*
*fTrue-ne-fFalse fNot-table fconj-table fdisj-table fimplies-table fAll-table fEx-table*
*fequal-table fAll-table fEx-table fNot-law fComp-law fconj-laws fdisj-laws fimplies-laws*
*fequal-laws fAll-law fEx-law lambda-def eq-lambdaI*

**end**

## 42 Generic theorem transfer using relations

**theory** *Transfer*
**imports** *Basic-BNF-LFPs Hilbert-Choice Metis*
**begin**

### 42.1 Relator for function space

**bundle** *lifting-syntax*
**begin**
**notation** *rel-fun* (**infixr** ‹===>› *55*)
**notation** *map-fun* (**infixr** ‹--->› *55*)
**end**

**context includes** *lifting-syntax*
**begin**

**lemma** *rel-funD2*:
**assumes** *rel-fun A B f g* **and** *A x x*

**shows** $B\ (f\ x)\ (g\ x)$
⟨*proof*⟩

**lemma** *rel-funE*:
**assumes** *rel-fun* $A\ B\ f\ g$ **and** $A\ x\ y$
**obtains** $B\ (f\ x)\ (g\ y)$
⟨*proof*⟩

**lemmas** *rel-fun-eq* = *fun.rel-eq*

**lemma** *rel-fun-eq-rel*:
**shows** *rel-fun* $(=)\ R = (\lambda f\ g.\ \forall x.\ R\ (f\ x)\ (g\ x))$
⟨*proof*⟩

## 42.2 Transfer method

Explicit tag for relation membership allows for backward proof methods.

**definition** *Rel* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$
**where** $Rel\ r \equiv r$

Handling of equality relations

**definition** *is-equality* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$
**where** *is-equality* $R \longleftrightarrow R = (=)$

**lemma** *is-equality-eq*: *is-equality* $(=)$
⟨*proof*⟩

Reverse implication for monotonicity rules

**definition** *rev-implies* **where**
*rev-implies* $x\ y \longleftrightarrow (y \longrightarrow x)$

Handling of meta-logic connectives

**definition** *transfer-forall* **where**
*transfer-forall* $\equiv$ *All*

**definition** *transfer-implies* **where**
*transfer-implies* $\equiv (\longrightarrow)$

**definition** *transfer-bforall* :: $('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$
**where** *transfer-bforall* $\equiv (\lambda P\ Q.\ \forall x.\ P\ x \longrightarrow Q\ x)$

**lemma** *transfer-forall-eq*: $(\bigwedge x.\ P\ x) \equiv Trueprop\ (transfer\text{-}forall\ (\lambda x.\ P\ x))$
⟨*proof*⟩

**lemma** *transfer-implies-eq*: $(A \Longrightarrow B) \equiv Trueprop\ (transfer\text{-}implies\ A\ B)$
⟨*proof*⟩

**lemma** *transfer-bforall-unfold*:

$Trueprop\ (transfer\text{-}bforall\ P\ (\lambda x.\ Q\ x)) \equiv (\bigwedge x.\ P\ x \Longrightarrow Q\ x)$
⟨*proof*⟩

**lemma** *transfer-start*: $[\![P;\ Rel\ (=)\ P\ Q]\!] \Longrightarrow Q$
⟨*proof*⟩

**lemma** *transfer-start'*: $[\![P;\ Rel\ (\longrightarrow)\ P\ Q]\!] \Longrightarrow Q$
⟨*proof*⟩

**lemma** *transfer-prover-start*: $[\![x = x';\ Rel\ R\ x'\ y]\!] \Longrightarrow Rel\ R\ x\ y$
⟨*proof*⟩

**lemma** *untransfer-start*: $[\![Q;\ Rel\ (=)\ P\ Q]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *Rel-eq-refl*: $Rel\ (=)\ x\ x$
⟨*proof*⟩

**lemma** *Rel-app*:
**assumes** $Rel\ (A\ {=}{=}{=}{>}\ B)\ f\ g$ **and** $Rel\ A\ x\ y$
**shows** $Rel\ B\ (f\ x)\ (g\ y)$
⟨*proof*⟩

**lemma** *Rel-abs*:
**assumes** $\bigwedge x\ y.\ Rel\ A\ x\ y \Longrightarrow Rel\ B\ (f\ x)\ (g\ y)$
**shows** $Rel\ (A\ {=}{=}{=}{>}\ B)\ (\lambda x.\ f\ x)\ (\lambda y.\ g\ y)$
⟨*proof*⟩

## 42.3 Predicates on relations, i.e. "class constraints"

See also *left-unique* and *right-unique*.

**definition** *left-total* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
**where** $left\text{-}total\ R \longleftrightarrow (\forall x.\ \exists y.\ R\ x\ y)$

**definition** *right-total* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
**where** $right\text{-}total\ R \longleftrightarrow (\forall y.\ \exists x.\ R\ x\ y)$

**definition** *bi-total* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
**where** $bi\text{-}total\ R \longleftrightarrow (\forall x.\ \exists y.\ R\ x\ y) \wedge (\forall y.\ \exists x.\ R\ x\ y)$

**definition** *bi-unique* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
**where** $bi\text{-}unique\ R \longleftrightarrow$
$(\forall x\ y\ z.\ R\ x\ y \longrightarrow R\ x\ z \longrightarrow y = z) \wedge$
$(\forall x\ y\ z.\ R\ x\ z \longrightarrow R\ y\ z \longrightarrow x = y)$

**lemma** *left-totalI*:
$(\bigwedge x.\ \exists y.\ R\ x\ y) \Longrightarrow left\text{-}total\ R$
⟨*proof*⟩

**lemma** *left-totalE*:
**assumes** *left-total R*
**obtains** $(\bigwedge x.\ \exists y.\ R\ x\ y)$
⟨*proof*⟩

**lemma** *bi-uniqueDr*: $[\![$ *bi-unique A*; $A\ x\ y$; $A\ x\ z$ $]\!] \Longrightarrow y = z$
⟨*proof*⟩

**lemma** *bi-uniqueDl*: $[\![$ *bi-unique A*; $A\ x\ y$; $A\ z\ y$ $]\!] \Longrightarrow x = z$
⟨*proof*⟩

**lemma** *bi-unique-iff*: *bi-unique R* $\longleftrightarrow (\forall z.\ \exists_{\leq 1}x.\ R\ x\ z) \wedge (\forall z.\ \exists_{\leq 1}x.\ R\ z\ x)$
⟨*proof*⟩

**lemma** *right-totalI*: $(\bigwedge y.\ \exists x.\ A\ x\ y) \Longrightarrow$ *right-total A*
⟨*proof*⟩

**lemma** *right-totalE*:
**assumes** *right-total A*
**obtains** $x$ **where** $A\ x\ y$
⟨*proof*⟩

**lemma** *right-total-alt-def2*:
*right-total R* $\longleftrightarrow ((R ===> (\longrightarrow)) ===> (\longrightarrow))$ *All All* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *right-unique-alt-def2*:
*right-unique R* $\longleftrightarrow (R ===> R ===> (\longrightarrow))\ (=)\ (=)$
⟨*proof*⟩

**lemma** *bi-total-alt-def2*:
*bi-total R* $\longleftrightarrow ((R ===> (=)) ===> (=))$ *All All* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *bi-unique-alt-def2*:
*bi-unique R* $\longleftrightarrow (R ===> R ===> (=))\ (=)\ (=)$
⟨*proof*⟩

**lemma** [*simp*]:
**shows** *left-total-conversep*: *left-total* $A^{-1-1} \longleftrightarrow$ *right-total A*
**and** *right-total-conversep*: *right-total* $A^{-1-1} \longleftrightarrow$ *left-total A*
⟨*proof*⟩

**lemma** *bi-unique-conversep* [*simp*]: *bi-unique* $R^{-1-1}$ = *bi-unique R*
⟨*proof*⟩

**lemma** *bi-total-conversep* [*simp*]: *bi-total* $R^{-1-1}$ = *bi-total R*
⟨*proof*⟩

**lemma** *right-unique-alt-def*: *right-unique* $R$ = (*conversep* $R$ *OO* $R \leq (=)$) ⟨*proof*⟩
**lemma** *left-unique-alt-def*: *left-unique* $R$ = ($R$ *OO* (*conversep* $R$) $\leq (=)$) ⟨*proof*⟩

**lemma** *right-total-alt-def*: *right-total* $R$ = (*conversep* $R$ *OO* $R \geq (=)$) ⟨*proof*⟩
**lemma** *left-total-alt-def*: *left-total* $R$ = ($R$ *OO* *conversep* $R \geq (=)$) ⟨*proof*⟩

**lemma** *bi-total-alt-def*: *bi-total* $A$ = (*left-total* $A$ $\wedge$ *right-total* $A$)
⟨*proof*⟩

**lemma** *bi-unique-alt-def*: *bi-unique* $A$ = (*left-unique* $A$ $\wedge$ *right-unique* $A$)
⟨*proof*⟩

**lemma** *bi-totalI*: *left-total* $R \Longrightarrow$ *right-total* $R \Longrightarrow$ *bi-total* $R$
⟨*proof*⟩

**lemma** *bi-uniqueI*: *left-unique* $R \Longrightarrow$ *right-unique* $R \Longrightarrow$ *bi-unique* $R$
⟨*proof*⟩

**end**

**lemma** *is-equality-lemma*: ($\bigwedge R$. *is-equality* $R \Longrightarrow$ *PROP* ($P$ $R$)) $\equiv$ *PROP* ($P$ $(=)$)
⟨*proof*⟩

**lemma** *Domainp-lemma*: ($\bigwedge R$. *Domainp* $T = R \Longrightarrow$ *PROP* ($P$ $R$)) $\equiv$ *PROP* ($P$ (*Domainp* $T$))
⟨*proof*⟩

⟨*ML*⟩
**declare** *refl* [*transfer-rule*]

**hide-const** (**open**) *Rel*

**context includes** *lifting-syntax*
**begin**

Handling of domains

**lemma** *Domainp-iff*: *Domainp* $T$ $x \longleftrightarrow (\exists y.\ T\ x\ y)$
⟨*proof*⟩

**lemma** *Domainp-refl*[*transfer-domain-rule*]:
*Domainp* $T$ = *Domainp* $T$ ⟨*proof*⟩

**lemma** *Domain-eq-top*[*transfer-domain-rule*]: *Domainp* $(=)$ = *top* ⟨*proof*⟩

**lemma** *Domainp-pred-fun-eq*[*relator-domain*]:
**assumes** *left-unique* $T$
**shows** *Domainp* ($T$ ===> $S$) = *pred-fun* (*Domainp* $T$) (*Domainp* $S$) (**is** *?lhs* = *?rhs*)

⟨*proof*⟩

Properties are preserved by relation composition.

**lemma** *OO-def*: $R\ OO\ S = (\lambda x\ z.\ \exists\, y.\ R\ x\ y \wedge S\ y\ z)$
⟨*proof*⟩

**lemma** *bi-total-OO*: $[\![$*bi-total* $A$; *bi-total* $B]\!] \Longrightarrow$ *bi-total* $(A\ OO\ B)$
⟨*proof*⟩

**lemma** *bi-unique-OO*: $[\![$*bi-unique* $A$; *bi-unique* $B]\!] \Longrightarrow$ *bi-unique* $(A\ OO\ B)$
⟨*proof*⟩

**lemma** *right-total-OO*:
$[\![$*right-total* $A$; *right-total* $B]\!] \Longrightarrow$ *right-total* $(A\ OO\ B)$
⟨*proof*⟩

**lemma** *right-unique-OO*:
$[\![$*right-unique* $A$; *right-unique* $B]\!] \Longrightarrow$ *right-unique* $(A\ OO\ B)$
⟨*proof*⟩

**lemma** *left-total-OO*: *left-total* $R \Longrightarrow$ *left-total* $S \Longrightarrow$ *left-total* $(R\ OO\ S)$
⟨*proof*⟩

**lemma** *left-unique-OO*: *left-unique* $R \Longrightarrow$ *left-unique* $S \Longrightarrow$ *left-unique* $(R\ OO\ S)$
⟨*proof*⟩

## 42.4 Properties of relators

**lemma** *left-total-eq*[*transfer-rule*]: *left-total* $(=)$
⟨*proof*⟩

**lemma** *left-unique-eq*[*transfer-rule*]: *left-unique* $(=)$
⟨*proof*⟩

**lemma** *right-total-eq* [*transfer-rule*]: *right-total* $(=)$
⟨*proof*⟩

**lemma** *right-unique-eq* [*transfer-rule*]: *right-unique* $(=)$
⟨*proof*⟩

**lemma** *bi-total-eq*[*transfer-rule*]: *bi-total* $(=)$
⟨*proof*⟩

**lemma** *bi-unique-eq*[*transfer-rule*]: *bi-unique* $(=)$
⟨*proof*⟩

**lemma** *left-total-fun*[*transfer-rule*]:
**assumes** *left-unique* $A$ *left-total* $B$
**shows** *left-total* $(A$ ===> $B)$

⟨*proof*⟩

**lemma** *left-unique-fun*[*transfer-rule*]:
$[\![$*left-total* $A$; *left-unique* $B]\!] \Longrightarrow$ *left-unique* $(A \text{ ===> } B)$
⟨*proof*⟩

**lemma** *right-total-fun* [*transfer-rule*]:
**assumes** *right-unique* $A$ *right-total* $B$
**shows** *right-total* $(A \text{ ===> } B)$
⟨*proof*⟩

**lemma** *right-unique-fun* [*transfer-rule*]:
$[\![$*right-total* $A$; *right-unique* $B]\!] \Longrightarrow$ *right-unique* $(A \text{ ===> } B)$
⟨*proof*⟩

**lemma** *bi-total-fun*[*transfer-rule*]:
$[\![$*bi-unique* $A$; *bi-total* $B]\!] \Longrightarrow$ *bi-total* $(A \text{ ===> } B)$
⟨*proof*⟩

**lemma** *bi-unique-fun*[*transfer-rule*]:
$[\![$*bi-total* $A$; *bi-unique* $B]\!] \Longrightarrow$ *bi-unique* $(A \text{ ===> } B)$
⟨*proof*⟩

**end**

**lemma** *if-conn*:
$(\textit{if } P \wedge Q \textit{ then } t \textit{ else } e) = (\textit{if } P \textit{ then if } Q \textit{ then } t \textit{ else } e \textit{ else } e)$
$(\textit{if } P \vee Q \textit{ then } t \textit{ else } e) = (\textit{if } P \textit{ then } t \textit{ else if } Q \textit{ then } t \textit{ else } e)$
$(\textit{if } P \longrightarrow Q \textit{ then } t \textit{ else } e) = (\textit{if } P \textit{ then if } Q \textit{ then } t \textit{ else } e \textit{ else } t)$
$(\textit{if } \neg P \textit{ then } t \textit{ else } e) = (\textit{if } P \textit{ then } e \textit{ else } t)$
⟨*proof*⟩

⟨*ML*⟩

**declare** *pred-fun-def* [*simp*]
**declare** *rel-fun-eq* [*relator-eq*]

**declare** *fun.Domainp-rel*[*relator-domain del*]

## 42.5 Transfer rules

**context includes** *lifting-syntax*
**begin**

**lemma** *Domainp-forall-transfer* [*transfer-rule*]:
**assumes** *right-total* $A$
**shows** $((A \text{ ===> } (=)) \text{ ===> } (=))$
(*transfer-bforall* (*Domainp* $A$)) *transfer-forall*

⟨*proof*⟩

Transfer rules using implication instead of equality on booleans.

**lemma** *transfer-forall-transfer* [*transfer-rule*]:
*bi-total A* ⟹ ((*A* ===> (=)) ===> (=)) *transfer-forall transfer-forall*
*right-total A* ⟹ ((*A* ===> (=)) ===> *implies*) *transfer-forall transfer-forall*
*right-total A* ⟹ ((*A* ===> *implies*) ===> *implies*) *transfer-forall transfer-forall*
*bi-total A* ⟹ ((*A* ===> (=)) ===> *rev-implies*) *transfer-forall transfer-forall*
*bi-total A* ⟹ ((*A* ===> *rev-implies*) ===> *rev-implies*) *transfer-forall transfer-forall*
⟨*proof*⟩

**lemma** *transfer-implies-transfer* [*transfer-rule*]:
((=) ===> (=) ===> (=) ) *transfer-implies transfer-implies*
(*rev-implies* ===> *implies* ===> *implies* ) *transfer-implies transfer-implies*
(*rev-implies* ===> (=) ===> *implies* ) *transfer-implies transfer-implies*
((=) ===> *implies* ===> *implies* ) *transfer-implies transfer-implies*
((=) ===> (=) ===> *implies* ) *transfer-implies transfer-implies*
(*implies* ===> *rev-implies* ===> *rev-implies*) *transfer-implies transfer-implies*
(*implies* ===> (=) ===> *rev-implies*) *transfer-implies transfer-implies*
((=) ===> *rev-implies* ===> *rev-implies*) *transfer-implies transfer-implies*
((=) ===> (=) ===> *rev-implies*) *transfer-implies transfer-implies*
⟨*proof*⟩

**lemma** *eq-imp-transfer* [*transfer-rule*]:
*right-unique A* ⟹ (*A* ===> *A* ===> (⟶)) (=) (=)
⟨*proof*⟩

Transfer rules using equality.

**lemma** *left-unique-transfer* [*transfer-rule*]:
**assumes** *right-total A*
**assumes** *right-total B*
**assumes** *bi-unique A*
**shows** ((*A* ===> *B* ===> (=)) ===> *implies*) *left-unique left-unique*
⟨*proof*⟩

**lemma** *eq-transfer* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*A* ===> *A* ===> (=)) (=) (=)
⟨*proof*⟩

**lemma** *right-total-Ex-transfer*[*transfer-rule*]:
**assumes** *right-total A*
**shows** ((*A* ===> (=)) ===> (=)) (*Bex* (*Collect* (*Domainp A*))) *Ex*
⟨*proof*⟩

**lemma** *right-total-All-transfer*[*transfer-rule*]:
**assumes** *right-total A*

**shows** $((A ===> (=)) ===> (=))$ $(Ball\ (Collect\ (Domainp\ A)))\ All$
⟨*proof*⟩

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *right-total-fun-eq-transfer*:
**assumes** [*transfer-rule*]: *right-total A bi-unique B*
**shows** $((A ===> B) ===> (A ===> B) ===> (=))$ $(\lambda f\, g.\ \forall x \in Collect(Domainp\ A).\ f\ x = g\ x)\ (=)$
⟨*proof*⟩

**end**

**lemma** *All-transfer* [*transfer-rule*]:
**assumes** *bi-total A*
**shows** $((A ===> (=)) ===> (=))$ *All All*
⟨*proof*⟩

**lemma** *Ex-transfer* [*transfer-rule*]:
**assumes** *bi-total A*
**shows** $((A ===> (=)) ===> (=))$ *Ex Ex*
⟨*proof*⟩

**lemma** *Ex1-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A bi-total A*
**shows** $((A ===> (=)) ===> (=))$ *Ex1 Ex1*
⟨*proof*⟩

**declare** *If-transfer* [*transfer-rule*]

**lemma** *Let-transfer* [*transfer-rule*]: $(A ===> (A ===> B) ===> B)$ *Let Let*
⟨*proof*⟩

**declare** *id-transfer* [*transfer-rule*]

**declare** *comp-transfer* [*transfer-rule*]

**lemma** *curry-transfer* [*transfer-rule*]:
$((rel\text{-}prod\ A\ B ===> C) ===> A ===> B ===> C)$ *curry curry*
⟨*proof*⟩

**lemma** *fun-upd-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** $((A ===> B) ===> A ===> B ===> A ===> B)$ *fun-upd fun-upd*
⟨*proof*⟩

**lemma** *case-nat-transfer* [*transfer-rule*]:

$(A$ ===> $((=)$ ===> $A)$ ===> $(=)$ ===> $A)$ *case-nat case-nat*
⟨*proof*⟩

**lemma** *rec-nat-transfer* [*transfer-rule*]:
$(A$ ===> $((=)$ ===> $A$ ===> $A)$ ===> $(=)$ ===> $A)$ *rec-nat rec-nat*
⟨*proof*⟩

**lemma** *funpow-transfer* [*transfer-rule*]:
$((=)$ ===> $(A$ ===> $A)$ ===> $(A$ ===> $A))$ *compow compow*
⟨*proof*⟩

**lemma** *mono-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $A$
**assumes** [*transfer-rule*]: $(A$ ===> $A$ ===> $(=))$ $(\leq)$ $(\leq)$
**assumes** [*transfer-rule*]: $(B$ ===> $B$ ===> $(=))$ $(\leq)$ $(\leq)$
**shows** $((A$ ===> $B)$ ===> $(=))$ *mono mono*
⟨*proof*⟩

**lemma** *right-total-relcompp-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *right-total* $B$
**shows** $((A$ ===> $B$ ===> $(=))$ ===> $(B$ ===> $C$ ===> $(=))$ ===> $A$
===> $C$ ===> $(=))$
$(\lambda R\ S\ x\ z.\ \exists y \in Collect\ (Domainp\ B).\ R\ x\ y \wedge S\ y\ z)$ $(OO)$
⟨*proof*⟩

**lemma** *relcompp-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $B$
**shows** $((A$ ===> $B$ ===> $(=))$ ===> $(B$ ===> $C$ ===> $(=))$ ===> $A$
===> $C$ ===> $(=))$ $(OO)$ $(OO)$
⟨*proof*⟩

**lemma** *right-total-Domainp-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *right-total* $B$
**shows** $((A$ ===> $B$ ===> $(=))$ ===> $A$ ===> $(=))$ $(\lambda T\ x.\ \exists y \in Collect(Domainp$
$B).\ T\ x\ y)$ *Domainp*
⟨*proof*⟩

**lemma** *Domainp-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $B$
**shows** $((A$ ===> $B$ ===> $(=))$ ===> $A$ ===> $(=))$ *Domainp Domainp*
⟨*proof*⟩

**lemma** *reflp-transfer*[*transfer-rule*]:
*bi-total* $A \Longrightarrow ((A$ ===> $A$ ===> $(=))$ ===> $(=))$ *reflp reflp*
*right-total* $A \Longrightarrow ((A$ ===> $A$ ===> *implies*$)$ ===> *implies*$)$ *reflp reflp*
*right-total* $A \Longrightarrow ((A$ ===> $A$ ===> $(=))$ ===> *implies*$)$ *reflp reflp*
*bi-total* $A \Longrightarrow ((A$ ===> $A$ ===> *rev-implies*$)$ ===> *rev-implies*$)$ *reflp reflp*
*bi-total* $A \Longrightarrow ((A$ ===> $A$ ===> $(=))$ ===> *rev-implies*$)$ *reflp reflp*

⟨*proof*⟩

**lemma** *right-unique-transfer* [*transfer-rule*]:
$\llbracket$ *right-total* $A$; *right-total* $B$; *bi-unique* $B$ $\rrbracket$
$\Longrightarrow$ (($A$ ===> $B$ ===> (=)) ===> *implies*) *right-unique right-unique*
⟨*proof*⟩

**lemma** *left-total-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $A$ *bi-total* $B$
**shows** (($A$ ===> $B$ ===> (=)) ===> (=)) *left-total left-total*
⟨*proof*⟩

**lemma** *right-total-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $A$ *bi-total* $B$
**shows** (($A$ ===> $B$ ===> (=)) ===> (=)) *right-total right-total*
⟨*proof*⟩

**lemma** *left-unique-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique* $A$ *bi-total* $A$ *bi-total* $B$
**shows** (($A$ ===> $B$ ===> (=)) ===> (=)) *left-unique left-unique*
⟨*proof*⟩

**lemma** *prod-pred-parametric* [*transfer-rule*]:
(($A$ ===> (=)) ===> ($B$ ===> (=)) ===> *rel-prod* $A$ $B$ ===> (=)) *pred-prod pred-prod*
⟨*proof*⟩

**lemma** *apfst-parametric* [*transfer-rule*]:
(($A$ ===> $B$) ===> *rel-prod* $A$ $C$ ===> *rel-prod* $B$ $C$) *apfst apfst*
⟨*proof*⟩

**lemma** *rel-fun-eq-eq-onp*: ((=) ===> *eq-onp* $P$) = *eq-onp* ($\lambda f.\ \forall x.\ P(f\ x)$)
⟨*proof*⟩

**lemma** *rel-fun-eq-onp-rel*:
**shows** ((*eq-onp* $R$) ===> $S$) = ($\lambda f\ g.\ \forall x.\ R\ x \longrightarrow S\ (f\ x)\ (g\ x)$)
⟨*proof*⟩

**lemma** *eq-onp-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique* $A$
**shows** (($A$ ===> (=)) ===> $A$ ===> $A$ ===> (=)) *eq-onp eq-onp*
⟨*proof*⟩

**lemma** *rtranclp-parametric* [*transfer-rule*]:
**assumes** *bi-unique* $A$ *bi-total* $A$
**shows** (($A$ ===> $A$ ===> (=)) ===> $A$ ===> $A$ ===> (=)) *rtranclp rtranclp*
⟨*proof*⟩

**lemma** *right-unique-parametric* [*transfer-rule*]:
  **assumes** [*transfer-rule*]: *bi-total A bi-unique B bi-total B*
  **shows** ((*A* ===> *B* ===> (=)) ===> (=)) *right-unique right-unique*
  ⟨*proof*⟩

**lemma** *map-fun-parametric* [*transfer-rule*]:
  ((*A* ===> *B*) ===> (*C* ===> *D*) ===> (*B* ===> *C*) ===> *A* ===> *D*) *map-fun map-fun*
  ⟨*proof*⟩

**end**

## 42.6 *of-bool* and *of-nat*

**context**
  **includes** *lifting-syntax*
**begin**

**lemma** *transfer-rule-of-bool*:
  ‹((⟷) ===> (≅)) *of-bool of-bool*›
    **if** [*transfer-rule*]: ‹*0* ≅ *0*› ‹*1* ≅ *1*›
    **for** *R* :: ‹'*a*::*zero-neq-one* ⇒ '*b*::*zero-neq-one* ⇒ *bool*›  (**infix** ‹≅› *50*)
  ⟨*proof*⟩

**lemma** *transfer-rule-of-nat*:
  ((=) ===> (≅)) *of-nat of-nat*
    **if** [*transfer-rule*]: ‹*0* ≅ *0*› ‹*1* ≅ *1*›
    ‹((≅) ===> (≅) ===> (≅)) (+) (+)›
  **for** *R* :: ‹'*a*::*semiring-1* ⇒ '*b*::*semiring-1* ⇒ *bool*›  (**infix** ‹≅› *50*)
  ⟨*proof*⟩

**end**

**end**

# 43 Lifting package

**theory** *Lifting*
**imports** *Equiv-Relations Transfer*
**keywords**
  *parametric* **and**
  *print-quot-maps print-quotients* :: *diag* **and**
  *lift-definition* :: *thy-goal-defn* **and**
  *setup-lifting lifting-forget lifting-update* :: *thy-decl*
**begin**

## 43.1 Function map

**context includes** *lifting-syntax*

**begin**

**lemma** *map-fun-id*:
$(id \; {-}{-}{-}{>} \; id) = id$
⟨*proof*⟩

## 43.2 Quotient Predicate

**definition** *Quotient* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$
**where**
$Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
$(\forall a.\ Abs\ (Rep\ a) = a) \wedge$
$(\forall a.\ R\ (Rep\ a)\ (Rep\ a)) \wedge$
$(\forall r\ s.\ R\ r\ s \longleftrightarrow R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s) \wedge$
$T = (\lambda x\ y.\ R\ x\ x \wedge Abs\ x = y)$

**lemma** *QuotientI*:
**assumes** $\bigwedge a.\ Abs\ (Rep\ a) = a$
**and** $\bigwedge a.\ R\ (Rep\ a)\ (Rep\ a)$
**and** $\bigwedge r\ s.\ R\ r\ s \longleftrightarrow R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s$
**and** $T = (\lambda x\ y.\ R\ x\ x \wedge Abs\ x = y)$
**shows** $Quotient\ R\ Abs\ Rep\ T$
⟨*proof*⟩

**context**
**fixes** $R\ Abs\ Rep\ T$
**assumes** *a*: $Quotient\ R\ Abs\ Rep\ T$
**begin**

**lemma** *Quotient-abs-rep*: $Abs\ (Rep\ a) = a$
⟨*proof*⟩

**lemma** *Quotient-rep-reflp*: $R\ (Rep\ a)\ (Rep\ a)$
⟨*proof*⟩

**lemma** *Quotient-rel*:
$R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s \longleftrightarrow R\ r\ s$ — orientation does not loop on rewriting
⟨*proof*⟩

**lemma** *Quotient-cr-rel*: $T = (\lambda x\ y.\ R\ x\ x \wedge Abs\ x = y)$
⟨*proof*⟩

**lemma** *Quotient-refl1*: $R\ r\ s \Longrightarrow R\ r\ r$
⟨*proof*⟩

**lemma** *Quotient-refl2*: $R\ r\ s \Longrightarrow R\ s\ s$
⟨*proof*⟩

**lemma** *Quotient-rel-rep*: $R\ (Rep\ a)\ (Rep\ b) \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *Quotient-rep-abs*: $R\ r\ r \Longrightarrow R\ (Rep\ (Abs\ r))\ r$
⟨*proof*⟩

**lemma** *Quotient-rep-abs-eq*: $R\ t\ t \Longrightarrow R \leq (=) \Longrightarrow Rep\ (Abs\ t) = t$
⟨*proof*⟩

**lemma** *Quotient-rep-abs-fold-unmap*:
**assumes** $x' \equiv Abs\ x$ **and** $R\ x\ x$ **and** $Rep\ x' \equiv Rep'\ x'$
**shows** $R\ (Rep'\ x')\ x$
⟨*proof*⟩

**lemma** *Quotient-Rep-eq*:
**assumes** $x' \equiv Abs\ x$
**shows** $Rep\ x' \equiv Rep\ x'$
⟨*proof*⟩

**lemma** *Quotient-rel-abs*: $R\ r\ s \Longrightarrow Abs\ r = Abs\ s$
⟨*proof*⟩

**lemma** *Quotient-rel-abs2*:
**assumes** $R\ (Rep\ x)\ y$
**shows** $x = Abs\ y$
⟨*proof*⟩

**lemma** *Quotient-symp*: *symp R*
⟨*proof*⟩

**lemma** *Quotient-transp*: *transp R*
⟨*proof*⟩

**lemma** *Quotient-part-equivp*: *part-equivp R*
⟨*proof*⟩

**end**

**lemma** *identity-quotient*: $Quotient\ (=)\ id\ id\ (=)$
⟨*proof*⟩

TODO: Use one of these alternatives as the real definition.

**lemma** *Quotient-alt-def*:
$Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
$(\forall a\ b.\ T\ a\ b \longrightarrow Abs\ a = b) \wedge$
$(\forall b.\ T\ (Rep\ b)\ b) \wedge$
$(\forall x\ y.\ R\ x\ y \longleftrightarrow T\ x\ (Abs\ x) \wedge T\ y\ (Abs\ y) \wedge Abs\ x = Abs\ y)$
⟨*proof*⟩

**lemma** *Quotient-alt-def2*:
  $Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
    $(\forall a\ b.\ T\ a\ b \longrightarrow Abs\ a = b) \wedge$
    $(\forall b.\ T\ (Rep\ b)\ b) \wedge$
    $(\forall x\ y.\ R\ x\ y \longleftrightarrow T\ x\ (Abs\ y) \wedge T\ y\ (Abs\ x))$
  ⟨*proof*⟩

**lemma** *Quotient-alt-def3*:
  $Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
    $(\forall a\ b.\ T\ a\ b \longrightarrow Abs\ a = b) \wedge (\forall b.\ T\ (Rep\ b)\ b) \wedge$
    $(\forall x\ y.\ R\ x\ y \longleftrightarrow (\exists z.\ T\ x\ z \wedge T\ y\ z))$
  ⟨*proof*⟩

**lemma** *Quotient-alt-def4*:
  $Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
    $(\forall a\ b.\ T\ a\ b \longrightarrow Abs\ a = b) \wedge (\forall b.\ T\ (Rep\ b)\ b) \wedge R = T\ OO\ conversep\ T$
  ⟨*proof*⟩

**lemma** *Quotient-alt-def5*:
  $Quotient\ R\ Abs\ Rep\ T \longleftrightarrow$
    $T \leq BNF\text{-}Def.Grp\ UNIV\ Abs \wedge BNF\text{-}Def.Grp\ UNIV\ Rep \leq T^{-1-1} \wedge R = T$ $OO\ T^{-1-1}$
  ⟨*proof*⟩

**lemma** *fun-quotient*:
  **assumes** *1*: $Quotient\ R1\ abs1\ rep1\ T1$
  **assumes** *2*: $Quotient\ R2\ abs2\ rep2\ T2$
  **shows** $Quotient\ (R1 ===> R2)\ (rep1 ---> abs2)\ (abs1 ---> rep2)\ (T1$ $===> T2)$
  ⟨*proof*⟩

**lemma** *apply-rsp*:
  **fixes** $f\ g::'a \Rightarrow 'c$
  **assumes** *q*: $Quotient\ R1\ Abs1\ Rep1\ T1$
  **and** *a*: $(R1 ===> R2)\ f\ g\ R1\ x\ y$
  **shows** $R2\ (f\ x)\ (g\ y)$
  ⟨*proof*⟩

**lemma** *apply-rsp'*:
  **assumes** *a*: $(R1 ===> R2)\ f\ g\ R1\ x\ y$
  **shows** $R2\ (f\ x)\ (g\ y)$
  ⟨*proof*⟩

**lemma** *apply-rsp''*:
  **assumes** $Quotient\ R\ Abs\ Rep\ T$
  **and** $(R ===> S)\ f\ f$
  **shows** $S\ (f\ (Rep\ x))\ (f\ (Rep\ x))$
⟨*proof*⟩

## 43.3 Quotient composition

**lemma** *Quotient-compose*:
**assumes** *1*: *Quotient R1 Abs1 Rep1 T1*
**assumes** *2*: *Quotient R2 Abs2 Rep2 T2*
**shows** *Quotient* (*T1 OO R2 OO conversep T1*) (*Abs2* $\circ$ *Abs1*) (*Rep1* $\circ$ *Rep2*) (*T1 OO T2*)
⟨*proof*⟩

**lemma** *equivp-reflp2*:
*equivp R* $\Longrightarrow$ *reflp R*
⟨*proof*⟩

## 43.4 Respects predicate

**definition** *Respects* :: ($'a \Rightarrow 'a \Rightarrow bool$) $\Rightarrow$ $'a$ *set*
**where** *Respects R* = $\{x.\ R\ x\ x\}$

**lemma** *in-respects*: $x \in$ *Respects R* $\longleftrightarrow$ *R x x*
⟨*proof*⟩

**lemma** *UNIV-typedef-to-Quotient*:
**assumes** *type-definition Rep Abs UNIV*
**and** *T-def*: $T \equiv (\lambda x\ y.\ x = Rep\ y)$
**shows** *Quotient* (=) *Abs Rep T*
⟨*proof*⟩

**lemma** *UNIV-typedef-to-equivp*:
**fixes** *Abs* :: $'a \Rightarrow 'b$
**and** *Rep* :: $'b \Rightarrow 'a$
**assumes** *type-definition Rep Abs* (*UNIV*::$'a$ *set*)
**shows** *equivp* ((=) ::$'a \Rightarrow 'a \Rightarrow bool$)
⟨*proof*⟩

**lemma** *typedef-to-Quotient*:
**assumes** *type-definition Rep Abs S*
**and** *T-def*: $T \equiv (\lambda x\ y.\ x = Rep\ y)$
**shows** *Quotient* (*eq-onp* ($\lambda x.\ x \in S$)) *Abs Rep T*
⟨*proof*⟩

**lemma** *typedef-to-part-equivp*:
**assumes** *type-definition Rep Abs S*
**shows** *part-equivp* (*eq-onp* ($\lambda x.\ x \in S$))
⟨*proof*⟩

**lemma** *open-typedef-to-Quotient*:
**assumes** *type-definition Rep Abs* $\{x.\ P\ x\}$
**and** *T-def*: $T \equiv (\lambda x\ y.\ x = Rep\ y)$
**shows** *Quotient* (*eq-onp P*) *Abs Rep T*
⟨*proof*⟩

**lemma** *open-typedef-to-part-equivp*:
**assumes** *type-definition Rep Abs* $\{x.\ P\ x\}$
**shows** *part-equivp* (*eq-onp* $P$)
⟨*proof*⟩

**lemma** *type-definition-Quotient-not-empty*: *Quotient* (*eq-onp* $P$) *Abs Rep* $T \Longrightarrow \exists\, x.\ P\ x$
⟨*proof*⟩

**lemma** *type-definition-Quotient-not-empty-witness*: *Quotient* (*eq-onp* $P$) *Abs Rep* $T \Longrightarrow P$ (*Rep undefined*)
⟨*proof*⟩

Generating transfer rules for quotients.

**context**
**fixes** $R$ *Abs Rep* $T$
**assumes** *1*: *Quotient* $R$ *Abs Rep* $T$
**begin**

**lemma** *Quotient-right-unique*: *right-unique* $T$
⟨*proof*⟩

**lemma** *Quotient-right-total*: *right-total* $T$
⟨*proof*⟩

**lemma** *Quotient-rel-eq-transfer*: $(T \mathrel{===>} T \mathrel{===>} (=))\ R\ (=)$
⟨*proof*⟩

**lemma** *Quotient-abs-induct*:
**assumes** $\bigwedge y.\ R\ y\ y \Longrightarrow P\ (Abs\ y)$ **shows** $P\ x$
⟨*proof*⟩

**end**

Generating transfer rules for total quotients.

**context**
**fixes** $R$ *Abs Rep* $T$
**assumes** *1*: *Quotient* $R$ *Abs Rep* $T$ **and** *2*: *reflp* $R$
**begin**

**lemma** *Quotient-left-total*: *left-total* $T$
⟨*proof*⟩

**lemma** *Quotient-bi-total*: *bi-total* $T$
⟨*proof*⟩

**lemma** *Quotient-id-abs-transfer*: $((=) \mathrel{===>} T)\ (\lambda x.\ x)\ Abs$
⟨*proof*⟩

**lemma** *Quotient-total-abs-induct*: $(\bigwedge y.\ P\ (Abs\ y)) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** *Quotient-total-abs-eq-iff*: $Abs\ x = Abs\ y \longleftrightarrow R\ x\ y$
⟨*proof*⟩

**end**

Generating transfer rules for a type defined with *typedef*.

**context**
**fixes** *Rep Abs A T*
**assumes** *type*: *type-definition Rep Abs A*
**assumes** *T-def*: $T \equiv (\lambda(x::'a)\ (y::'b).\ x = Rep\ y)$
**begin**

**lemma** *typedef-left-unique*: *left-unique T*
⟨*proof*⟩

**lemma** *typedef-bi-unique*: *bi-unique T*
⟨*proof*⟩

**lemma** *typedef-right-unique*: *right-unique T*
⟨*proof*⟩

**lemma** *typedef-right-total*: *right-total T*
⟨*proof*⟩

**lemma** *typedef-rep-transfer*: $(T\ {===>}\ (=))\ (\lambda x.\ x)\ Rep$
⟨*proof*⟩

**end**

Generating the correspondence rule for a constant defined with *lift-definition*.

**lemma** *Quotient-to-transfer*:
**assumes** *Quotient R Abs Rep T* **and** *R c c* **and** $c' \equiv Abs\ c$
**shows** $T\ c\ c'$
⟨*proof*⟩

Proving reflexivity

**lemma** *Quotient-to-left-total*:
**assumes** *q*: *Quotient R Abs Rep T*
**and** *r-R*: *reflp R*
**shows** *left-total T*
⟨*proof*⟩

**lemma** *Quotient-composition-ge-eq*:

**assumes** *left-total* $T$
**assumes** $R \geq (=)$
**shows** $(T\ OO\ R\ OO\ T^{-1-1}) \geq (=)$
⟨*proof*⟩

**lemma** *Quotient-composition-le-eq*:
**assumes** *left-unique* $T$
**assumes** $R \leq (=)$
**shows** $(T\ OO\ R\ OO\ T^{-1-1}) \leq (=)$
⟨*proof*⟩

**lemma** *eq-onp-le-eq*:
*eq-onp* $P \leq (=)$ ⟨*proof*⟩

**lemma** *reflp-ge-eq*:
*reflp* $R \Longrightarrow R \geq (=)$ ⟨*proof*⟩

Proving a parametrized correspondence relation

**definition** *POS* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$ **where**
$POS\ A\ B \equiv A \leq B$

**definition** *NEG* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow bool$ **where**
$NEG\ A\ B \equiv B \leq A$

**lemma** *pos-OO-eq*:
**shows** $POS\ (A\ OO\ (=))\ A$
⟨*proof*⟩

**lemma** *pos-eq-OO*:
**shows** $POS\ ((=)\ OO\ A)\ A$
⟨*proof*⟩

**lemma** *neg-OO-eq*:
**shows** $NEG\ (A\ OO\ (=))\ A$
⟨*proof*⟩

**lemma** *neg-eq-OO*:
**shows** $NEG\ ((=)\ OO\ A)\ A$
⟨*proof*⟩

**lemma** *POS-trans*:
**assumes** $POS\ A\ B$
**assumes** $POS\ B\ C$
**shows** $POS\ A\ C$
⟨*proof*⟩

**lemma** *NEG-trans*:
**assumes** $NEG\ A\ B$
**assumes** $NEG\ B\ C$

**shows** *NEG A C*
⟨*proof*⟩

**lemma** *POS-NEG*:
*POS A B* ≡ *NEG B A*
⟨*proof*⟩

**lemma** *NEG-POS*:
*NEG A B* ≡ *POS B A*
⟨*proof*⟩

**lemma** *POS-pcr-rule*:
**assumes** *POS* (*A OO B*) *C*
**shows** *POS* (*A OO B OO X*) (*C OO X*)
⟨*proof*⟩

**lemma** *NEG-pcr-rule*:
**assumes** *NEG* (*A OO B*) *C*
**shows** *NEG* (*A OO B OO X*) (*C OO X*)
⟨*proof*⟩

**lemma** *POS-apply*:
**assumes** *POS R R'*
**assumes** *R f g*
**shows** *R' f g*
⟨*proof*⟩

Proving a parametrized correspondence relation

**lemma** *fun-mono*:
**assumes** $A \geq C$
**assumes** $B \leq D$
**shows** (*A* ===> *B*) ≤ (*C* ===> *D*)
⟨*proof*⟩

**lemma** *pos-fun-distr*: ((*R* ===> *S*) *OO* (*R'* ===> *S'*)) ≤ ((*R OO R'*) ===> (*S OO S'*))
⟨*proof*⟩

**lemma** *functional-relation*: *right-unique R* ⟹ *left-total R* ⟹ ∀ *x*. ∃!*y*. *R x y*
⟨*proof*⟩

**lemma** *functional-converse-relation*: *left-unique R* ⟹ *right-total R* ⟹ ∀ *y*. ∃!*x*. *R x y*
⟨*proof*⟩

**lemma** *neg-fun-distr1*:
**assumes** *1*: *left-unique R right-total R*
**assumes** *2*: *right-unique R' left-total R'*
**shows** (*R OO R'* ===> *S OO S'*) ≤ ((*R* ===> *S*) *OO* (*R'* ===> *S'*))

⟨*proof*⟩

**lemma** *neg-fun-distr2*:
**assumes** *1*: *right-unique* $R'$ *left-total* $R'$
**assumes** *2*: *left-unique* $S'$ *right-total* $S'$
**shows** $(R\ OO\ R' ===> S\ OO\ S') \leq ((R ===> S)\ OO\ (R' ===> S'))$
⟨*proof*⟩

## 43.5 Domains

**lemma** *composed-equiv-rel-eq-onp*:
**assumes** *left-unique* $R$
**assumes** $(R ===> (=))\ P\ P'$
**assumes** *Domainp* $R = P''$
**shows** $(R\ OO\ \mathit{eq\text{-}onp}\ P'\ OO\ R^{-1-1}) = \mathit{eq\text{-}onp}\ (\mathit{inf}\ P''\ P)$
⟨*proof*⟩

**lemma** *composed-equiv-rel-eq-eq-onp*:
**assumes** *left-unique* $R$
**assumes** *Domainp* $R = P$
**shows** $(R\ OO\ (=)\ OO\ R^{-1-1}) = \mathit{eq\text{-}onp}\ P$
⟨*proof*⟩

**lemma** *pcr-Domainp-par-left-total*:
**assumes** *Domainp* $B = P$
**assumes** *left-total* $A$
**assumes** $(A ===> (=))\ P'\ P$
**shows** *Domainp* $(A\ OO\ B) = P'$
⟨*proof*⟩

**lemma** *pcr-Domainp-par*:
**assumes** *Domainp* $B = P2$
**assumes** *Domainp* $A = P1$
**assumes** $(A ===> (=))\ P2'\ P2$
**shows** *Domainp* $(A\ OO\ B) = (\mathit{inf}\ P1\ P2')$
⟨*proof*⟩

**definition** *rel-pred-comp* :: $('a => 'b => bool) => ('b => bool) => 'a => bool$
**where** *rel-pred-comp* $R\ P \equiv \lambda x.\ \exists y.\ R\ x\ y \wedge P\ y$

**lemma** *pcr-Domainp*:
**assumes** *Domainp* $B = P$
**shows** *Domainp* $(A\ OO\ B) = (\lambda x.\ \exists y.\ A\ x\ y \wedge P\ y)$
⟨*proof*⟩

**lemma** *pcr-Domainp-total*:
**assumes** *left-total* $B$
**assumes** *Domainp* $A = P$
**shows** *Domainp* $(A\ OO\ B) = P$

⟨*proof*⟩

**lemma** *Quotient-to-Domainp*:
  **assumes** *Quotient R Abs Rep T*
  **shows** *Domainp T* = (λ*x*. *R x x*)
⟨*proof*⟩

**lemma** *eq-onp-to-Domainp*:
  **assumes** *Quotient* (*eq-onp P*) *Abs Rep T*
  **shows** *Domainp T* = *P*
⟨*proof*⟩

**end**

**lemma** *right-total-UNIV-transfer*:
  **assumes** *right-total A*
  **shows** (*rel-set A*) (*Collect* (*Domainp A*)) *UNIV*
  ⟨*proof*⟩

## 43.6 ML setup

⟨*ML*⟩

**named-theorems** *relator-eq-onp*
  *theorems that a relator of an eq-onp is an eq-onp of the corresponding predicate*
⟨*ML*⟩

**declare** *fun-quotient*[*quot-map*]
**declare** *fun-mono*[*relator-mono*]
**lemmas** [*relator-distr*] = *pos-fun-distr neg-fun-distr1 neg-fun-distr2*

⟨*ML*⟩

**lemma** *pred-prod-beta*: *pred-prod P Q xy* ⟷ *P* (*fst xy*) ∧ *Q* (*snd xy*)
⟨*proof*⟩

**lemma** *pred-prod-split*: *P* (*pred-prod Q R xy*) ⟷ (∀ *x y*. *xy* = (*x*, *y*) ⟶ *P* (*Q x* ∧ *R y*))
⟨*proof*⟩

**hide-const** (**open**) *POS NEG*

**end**

# 44 Definition of Quotient Types

**theory** *Quotient*

**imports** *Lifting*
**keywords**
*print-quotmapsQ3 print-quotientsQ3 print-quotconsts* :: *diag* **and**
*quotient-type* :: *thy-goal-defn* **and** / **and**
*quotient-definition* :: *thy-goal-defn* **and**
*copy-bnf* :: *thy-defn* **and**
*lift-bnf* :: *thy-goal-defn*
**begin**

Basic definition for equivalence relations that are represented by predicates.

Composition of Relations

**abbreviation**
*rel-conj* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'b \Rightarrow bool$ (**infixr** ‹*OOO*› *75*)
**where**
$r1\ OOO\ r2 \equiv r1\ OO\ r2\ OO\ r1$

**lemma** *eq-comp-r*:
**shows** $((=)\ OOO\ R) = R$
⟨*proof*⟩

**context includes** *lifting-syntax*
**begin**

## 44.1 Quotient Predicate

**definition**
$Quotient3\ R\ Abs\ Rep \longleftrightarrow$
$(\forall a.\ Abs\ (Rep\ a) = a) \wedge (\forall a.\ R\ (Rep\ a)\ (Rep\ a)) \wedge$
$(\forall r\ s.\ R\ r\ s \longleftrightarrow R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s)$

**lemma** *Quotient3I*:
**assumes** $\bigwedge a.\ Abs\ (Rep\ a) = a$
**and** $\bigwedge a.\ R\ (Rep\ a)\ (Rep\ a)$
**and** $\bigwedge r\ s.\ R\ r\ s \longleftrightarrow R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s$
**shows** $Quotient3\ R\ Abs\ Rep$
⟨*proof*⟩

**context**
**fixes** $R\ Abs\ Rep$
**assumes** *a*: $Quotient3\ R\ Abs\ Rep$
**begin**

**lemma** *Quotient3-abs-rep*:
$Abs\ (Rep\ a) = a$
⟨*proof*⟩

**lemma** *Quotient3-rep-reflp*:
$R\ (Rep\ a)\ (Rep\ a)$

⟨*proof*⟩

**lemma** *Quotient3-rel*:
$R\ r\ r \wedge R\ s\ s \wedge Abs\ r = Abs\ s \longleftrightarrow R\ r\ s$ — orientation does not loop on rewriting
⟨*proof*⟩

**lemma** *Quotient3-refl1*:
$R\ r\ s \Longrightarrow R\ r\ r$
⟨*proof*⟩

**lemma** *Quotient3-refl2*:
$R\ r\ s \Longrightarrow R\ s\ s$
⟨*proof*⟩

**lemma** *Quotient3-rel-rep*:
$R\ (Rep\ a)\ (Rep\ b) \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *Quotient3-rep-abs*:
$R\ r\ r \Longrightarrow R\ (Rep\ (Abs\ r))\ r$
⟨*proof*⟩

**lemma** *Quotient3-rel-abs*:
$R\ r\ s \Longrightarrow Abs\ r = Abs\ s$
⟨*proof*⟩

**lemma** *Quotient3-symp*:
*symp R*
⟨*proof*⟩

**lemma** *Quotient3-transp*:
*transp R*
⟨*proof*⟩

**lemma** *Quotient3-part-equivp*:
*part-equivp R*
⟨*proof*⟩

**lemma** *abs-o-rep*:
$Abs \circ Rep = id$
⟨*proof*⟩

**lemma** *equals-rsp*:
**assumes** *b*: *R xa xb R ya yb*
**shows** $R\ xa\ ya = R\ xb\ yb$
⟨*proof*⟩

**lemma** *rep-abs-rsp*:
**assumes** *b*: *R x1 x2*

**shows** $R\ x1\ (Rep\ (Abs\ x2))$
⟨*proof*⟩

**lemma** *rep-abs-rsp-left*:
**assumes** $b$: $R\ x1\ x2$
**shows** $R\ (Rep\ (Abs\ x1))\ x2$
⟨*proof*⟩

**end**

**lemma** *identity-quotient3*:
$Quotient3\ (=)\ id\ id$
⟨*proof*⟩

**lemma** *fun-quotient3*:
**assumes** $q1$: $Quotient3\ R1\ abs1\ rep1$
**and** $q2$: $Quotient3\ R2\ abs2\ rep2$
**shows** $Quotient3\ (R1\ ===>\ R2)\ (rep1\ --->\ abs2)\ (abs1\ --->\ rep2)$
⟨*proof*⟩

**lemma** *lambda-prs*:
**assumes** $q1$: $Quotient3\ R1\ Abs1\ Rep1$
**and** $q2$: $Quotient3\ R2\ Abs2\ Rep2$
**shows** $(Rep1\ --->\ Abs2)\ (\lambda x.\ Rep2\ (f\ (Abs1\ x))) = (\lambda x.\ f\ x)$
⟨*proof*⟩

**lemma** *lambda-prs1*:
**assumes** $q1$: $Quotient3\ R1\ Abs1\ Rep1$
**and** $q2$: $Quotient3\ R2\ Abs2\ Rep2$
**shows** $(Rep1\ --->\ Abs2)\ (\lambda x.\ (Abs1\ --->\ Rep2)\ f\ x) = (\lambda x.\ f\ x)$
⟨*proof*⟩

In the following theorem R1 can be instantiated with anything, but we know some of the types of the Rep and Abs functions; so by solving Quotient assumptions we can get a unique R1 that will be provable; which is why we need to use *apply-rsp* and not the primed version

**lemma** *apply-rspQ3*:
**fixes** $f\ g{::}'a \Rightarrow 'c$
**assumes** $q$: $Quotient3\ R1\ Abs1\ Rep1$
**and** $a$: $(R1\ ===>\ R2)\ f\ g\ R1\ x\ y$
**shows** $R2\ (f\ x)\ (g\ y)$
⟨*proof*⟩

**lemma** *apply-rspQ3''*:
**assumes** $Quotient3\ R\ Abs\ Rep$
**and** $(R\ ===>\ S)\ f\ f$
**shows** $S\ (f\ (Rep\ x))\ (f\ (Rep\ x))$
⟨*proof*⟩

## 44.2 lemmas for regularisation of ball and bex

**lemma** *ball-reg-eqv*:
**fixes** $P :: 'a \Rightarrow bool$
**assumes** $a$: $equivp\ R$
**shows** $Ball\ (Respects\ R)\ P = (All\ P)$
⟨*proof*⟩

**lemma** *bex-reg-eqv*:
**fixes** $P :: 'a \Rightarrow bool$
**assumes** $a$: $equivp\ R$
**shows** $Bex\ (Respects\ R)\ P = (Ex\ P)$
⟨*proof*⟩

**lemma** *ball-reg-right*:
**assumes** $a$: $\bigwedge x.\ x \in R \Longrightarrow P\ x \longrightarrow Q\ x$
**shows** $All\ P \longrightarrow Ball\ R\ Q$
⟨*proof*⟩

**lemma** *bex-reg-left*:
**assumes** $a$: $\bigwedge x.\ x \in R \Longrightarrow Q\ x \longrightarrow P\ x$
**shows** $Bex\ R\ Q \longrightarrow Ex\ P$
⟨*proof*⟩

**lemma** *ball-reg-left*:
**assumes** $a$: $equivp\ R$
**shows** $(\bigwedge x.\ (Q\ x \longrightarrow P\ x)) \Longrightarrow Ball\ (Respects\ R)\ Q \longrightarrow All\ P$
⟨*proof*⟩

**lemma** *bex-reg-right*:
**assumes** $a$: $equivp\ R$
**shows** $(\bigwedge x.\ (Q\ x \longrightarrow P\ x)) \Longrightarrow Ex\ Q \longrightarrow Bex\ (Respects\ R)\ P$
⟨*proof*⟩

**lemma** *ball-reg-eqv-range*:
**fixes** $P::'a \Rightarrow bool$
**and** $x::'a$
**assumes** $a$: $equivp\ R2$
**shows** $(Ball\ (Respects\ (R1 ===> R2))\ (\lambda f.\ P\ (f\ x)) = All\ (\lambda f.\ P\ (f\ x)))$
⟨*proof*⟩

**lemma** *bex-reg-eqv-range*:
**assumes** $a$: $equivp\ R2$
**shows** $(Bex\ (Respects\ (R1 ===> R2))\ (\lambda f.\ P\ (f\ x)) = Ex\ (\lambda f.\ P\ (f\ x)))$
⟨*proof*⟩

**lemma** *all-reg*:
**assumes** $a$: $\forall x :: 'a.\ (P\ x \longrightarrow Q\ x)$
**and** $b$: $All\ P$

**shows** $All\ Q$
⟨*proof*⟩

**lemma** *ex-reg*:
**assumes** $a$: $\forall x :: 'a.\ (P\ x \longrightarrow Q\ x)$
**and** $b$: $Ex\ P$
**shows** $Ex\ Q$
⟨*proof*⟩

**lemma** *ball-reg*:
**assumes** $a$: $\forall x :: 'a.\ (x \in R \longrightarrow P\ x \longrightarrow Q\ x)$
**and** $b$: $Ball\ R\ P$
**shows** $Ball\ R\ Q$
⟨*proof*⟩

**lemma** *bex-reg*:
**assumes** $a$: $\forall x :: 'a.\ (x \in R \longrightarrow P\ x \longrightarrow Q\ x)$
**and** $b$: $Bex\ R\ P$
**shows** $Bex\ R\ Q$
⟨*proof*⟩

**lemma** *ball-all-comm*:
**assumes** $\bigwedge y.\ (\forall x \in P.\ A\ x\ y) \longrightarrow (\forall x.\ B\ x\ y)$
**shows** $(\forall x \in P.\ \forall y.\ A\ x\ y) \longrightarrow (\forall x.\ \forall y.\ B\ x\ y)$
⟨*proof*⟩

**lemma** *bex-ex-comm*:
**assumes** $(\exists y.\ \exists x.\ A\ x\ y) \longrightarrow (\exists y.\ \exists x \in P.\ B\ x\ y)$
**shows** $(\exists x.\ \exists y.\ A\ x\ y) \longrightarrow (\exists x \in P.\ \exists y.\ B\ x\ y)$
⟨*proof*⟩

## 44.3 Bounded abstraction

**definition**
$Babs :: 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$
**where**
$x \in p \Longrightarrow Babs\ p\ m\ x = m\ x$

**lemma** *babs-rsp*:
**assumes** $q$: $Quotient3\ R1\ Abs1\ Rep1$
**and** $a$: $(R1\ ===>\ R2)\ f\ g$
**shows** $(R1\ ===>\ R2)\ (Babs\ (Respects\ R1)\ f)\ (Babs\ (Respects\ R1)\ g)$
⟨*proof*⟩

**lemma** *babs-prs*:
**assumes** $q1$: $Quotient3\ R1\ Abs1\ Rep1$
**and** $q2$: $Quotient3\ R2\ Abs2\ Rep2$
**shows** $((Rep1\ --->\ Abs2)\ (Babs\ (Respects\ R1)\ ((Abs1\ --->\ Rep2)\ f))) = f$

⟨*proof*⟩

**lemma** *babs-simp*:
  **assumes** *q*: *Quotient3 R1 Abs Rep*
  **shows** ((*R1* ===> *R2*) (*Babs* (*Respects R1*) *f*) (*Babs* (*Respects R1*) *g*)) = ((*R1* ===> *R2*) *f g*)
    (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

If a user proves that a particular functional relation is an equivalence, this may be useful in regularising

**lemma** *babs-reg-eqv*:
  **shows** *equivp R* $\Longrightarrow$ *Babs* (*Respects R*) *P* = *P*
  ⟨*proof*⟩

**lemma** *ball-rsp*:
  **assumes** *a*: (*R* ===> (=)) *f g*
  **shows** *Ball* (*Respects R*) *f* = *Ball* (*Respects R*) *g*
  ⟨*proof*⟩

**lemma** *bex-rsp*:
  **assumes** *a*: (*R* ===> (=)) *f g*
  **shows** (*Bex* (*Respects R*) *f* = *Bex* (*Respects R*) *g*)
  ⟨*proof*⟩

**lemma** *bex1-rsp*:
  **assumes** *a*: (*R* ===> (=)) *f g*
  **shows** *Ex1* ($\lambda x.$ $x \in$ *Respects R* $\wedge$ *f x*) = *Ex1* ($\lambda x.$ $x \in$ *Respects R* $\wedge$ *g x*)
  ⟨*proof*⟩

Two lemmas needed for cleaning of quantifiers

**lemma** *all-prs*:
  **assumes** *a*: *Quotient3 R absf repf*
  **shows** *Ball* (*Respects R*) ((*absf* −−−> *id*) *f*) = *All f*
  ⟨*proof*⟩

**lemma** *ex-prs*:
  **assumes** *a*: *Quotient3 R absf repf*
  **shows** *Bex* (*Respects R*) ((*absf* −−−> *id*) *f*) = *Ex f*
  ⟨*proof*⟩

## 44.4 *Bex1-rel* quantifier

**definition**
  *Bex1-rel* :: ($'a \Rightarrow$ $'a \Rightarrow$ *bool*) $\Rightarrow$ ($'a \Rightarrow$ *bool*) $\Rightarrow$ *bool*
**where**

*Bex1-rel R P* $\longleftrightarrow$ ($\exists$ *x* $\in$ *Respects R. P x*) $\wedge$ ($\forall$ *x* $\in$ *Respects R.* $\forall$ *y* $\in$ *Respects R.* ((*P x* $\wedge$ *P y*) $\longrightarrow$ (*R x y*)))

**lemma** *bex1-rel-aux*:
$[\![\forall$ *xa ya. R xa ya* $\longrightarrow$ *x xa* = *y ya*; *Bex1-rel R x*$]\!] \Longrightarrow$ *Bex1-rel R y*
⟨*proof*⟩

**lemma** *bex1-rel-aux2*:
$[\![\forall$ *xa ya. R xa ya* $\longrightarrow$ *x xa* = *y ya*; *Bex1-rel R y*$]\!] \Longrightarrow$ *Bex1-rel R x*
⟨*proof*⟩

**lemma** *bex1-rel-rsp*:
**assumes** *a*: *Quotient3 R absf repf*
**shows** ((*R* ===> (=)) ===> (=)) (*Bex1-rel R*) (*Bex1-rel R*)
⟨*proof*⟩

**lemma** *ex1-prs*:
**assumes** *Quotient3 R absf repf*
**shows** ((*absf* −−−> *id*) −−−> *id*) (*Bex1-rel R*) *f* = *Ex1 f*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *bex1-bexeq-reg*:
**shows** ($\exists$!*x*$\in$*Respects R. P x*) $\longrightarrow$ (*Bex1-rel R* ($\lambda$*x. P x*))
⟨*proof*⟩

**lemma** *bex1-bexeq-reg-eqv*:
**assumes** *a*: *equivp R*
**shows** ($\exists$!*x. P x*) $\longrightarrow$ *Bex1-rel R P*
⟨*proof*⟩

## 44.5 Various respects and preserve lemmas

**lemma** *quot-rel-rsp*:
**assumes** *a*: *Quotient3 R Abs Rep*
**shows** (*R* ===> *R* ===> (=)) *R R*
⟨*proof*⟩

**lemma** *o-prs*:
**assumes** *q1*: *Quotient3 R1 Abs1 Rep1*
**and** *q2*: *Quotient3 R2 Abs2 Rep2*
**and** *q3*: *Quotient3 R3 Abs3 Rep3*
**shows** ((*Abs2* −−−> *Rep3*) −−−> (*Abs1* −−−> *Rep2*) −−−> (*Rep1* −−−> *Abs3*)) ($\circ$) = ($\circ$)
**and** (*id* −−−> (*Abs1* −−−> *id*) −−−> *Rep1* −−−> *id*) ($\circ$) = ($\circ$)
⟨*proof*⟩

**lemma** *o-rsp*:
((*R2* ===> *R3*) ===> (*R1* ===> *R2*) ===> (*R1* ===> *R3*)) ($\circ$) ($\circ$)

((=) ===> (*R1* ===> (=)) ===> *R1* ===> (=)) (∘) (∘)
⟨*proof*⟩

**lemma** *cond-prs*:
**assumes** *a*: *Quotient3 R absf repf*
**shows** *absf* (*if a then repf b else repf c*) = (*if a then b else c*)
⟨*proof*⟩

**lemma** *if-prs*:
**assumes** *q*: *Quotient3 R Abs Rep*
**shows** (*id* −−−> *Rep* −−−> *Rep* −−−> *Abs*) *If* = *If*
⟨*proof*⟩

**lemma** *if-rsp*:
**assumes** *q*: *Quotient3 R Abs Rep*
**shows** ((=) ===> *R* ===> *R* ===> *R*) *If If*
⟨*proof*⟩

**lemma** *let-prs*:
**assumes** *q1*: *Quotient3 R1 Abs1 Rep1*
**and** *q2*: *Quotient3 R2 Abs2 Rep2*
**shows** (*Rep2* −−−> (*Abs2* −−−> *Rep1*) −−−> *Abs1*) *Let* = *Let*
⟨*proof*⟩

**lemma** *let-rsp*:
**shows** (*R1* ===> (*R1* ===> *R2*) ===> *R2*) *Let Let*
⟨*proof*⟩

**lemma** *id-rsp*:
**shows** (*R* ===> *R*) *id id*
⟨*proof*⟩

**lemma** *id-prs*:
**assumes** *a*: *Quotient3 R Abs Rep*
**shows** (*Rep* −−−> *Abs*) *id* = *id*
⟨*proof*⟩

**end**

**locale** *quot-type* =
**fixes** $R :: 'a \Rightarrow 'a \Rightarrow bool$
**and** $Abs :: 'a\ set \Rightarrow 'b$
**and** $Rep :: 'b \Rightarrow 'a\ set$
**assumes** *equivp*: *part-equivp R*
**and** *rep-prop*: $\bigwedge y.\ \exists x.\ R\ x\ x \wedge Rep\ y = Collect\ (R\ x)$
**and** *rep-inverse*: $\bigwedge x.\ Abs\ (Rep\ x) = x$
**and** *abs-inverse*: $\bigwedge c.\ (\exists x.\ ((R\ x\ x) \wedge (c = Collect\ (R\ x)))) \Longrightarrow (Rep\ (Abs\ c)) = c$
**and** *rep-inject*: $\bigwedge x\ y.\ (Rep\ x = Rep\ y) = (x = y)$

**begin**

**definition**
  $abs :: 'a \Rightarrow 'b$
**where**
  $abs\ x = Abs\ (Collect\ (R\ x))$

**definition**
  $rep :: 'b \Rightarrow 'a$
**where**
  $rep\ a = (SOME\ x.\ x \in Rep\ a)$

**lemma** *some-collect*:
  **assumes** $R\ r\ r$
  **shows** $R\ (SOME\ x.\ x \in Collect\ (R\ r)) = R\ r$
  ⟨*proof*⟩

**lemma** *Quotient*: $Quotient3\ R\ abs\ rep$
  ⟨*proof*⟩

**end**

## 44.6 Quotient composition

**lemma** *OOO-quotient3*:
  **fixes** $R1 :: 'a \Rightarrow 'a \Rightarrow bool$
  **fixes** $Abs1 :: 'a \Rightarrow 'b$ **and** $Rep1 :: 'b \Rightarrow 'a$
  **fixes** $Abs2 :: 'b \Rightarrow 'c$ **and** $Rep2 :: 'c \Rightarrow 'b$
  **fixes** $R2' :: 'a \Rightarrow 'a \Rightarrow bool$
  **fixes** $R2 :: 'b \Rightarrow 'b \Rightarrow bool$
  **assumes** $R1$: $Quotient3\ R1\ Abs1\ Rep1$
  **assumes** $R2$: $Quotient3\ R2\ Abs2\ Rep2$
  **assumes** $Abs1$: $\bigwedge x\ y.\ R2'\ x\ y \Longrightarrow R1\ x\ x \Longrightarrow R1\ y\ y \Longrightarrow R2\ (Abs1\ x)\ (Abs1\ y)$
  **assumes** $Rep1$: $\bigwedge x\ y.\ R2\ x\ y \Longrightarrow R2'\ (Rep1\ x)\ (Rep1\ y)$
  **shows** $Quotient3\ (R1\ OO\ R2'\ OO\ R1)\ (Abs2 \circ Abs1)\ (Rep1 \circ Rep2)$
⟨*proof*⟩

**lemma** *OOO-eq-quotient3*:
  **fixes** $R1 :: 'a \Rightarrow 'a \Rightarrow bool$
  **fixes** $Abs1 :: 'a \Rightarrow 'b$ **and** $Rep1 :: 'b \Rightarrow 'a$
  **fixes** $Abs2 :: 'b \Rightarrow 'c$ **and** $Rep2 :: 'c \Rightarrow 'b$
  **assumes** $R1$: $Quotient3\ R1\ Abs1\ Rep1$
  **assumes** $R2$: $Quotient3\ (=)\ Abs2\ Rep2$
  **shows** $Quotient3\ (R1\ OOO\ (=))\ (Abs2 \circ Abs1)\ (Rep1 \circ Rep2)$
⟨*proof*⟩

## 44.7 Quotient3 to Quotient

**lemma** *Quotient3-to-Quotient*:
  **assumes** $Quotient3\ R\ Abs\ Rep$

  **and** $T \equiv \lambda x\ y.\ R\ x\ x \wedge Abs\ x = y$
 **shows** *Quotient R Abs Rep T*
 ⟨*proof*⟩

**lemma** *Quotient3-to-Quotient-equivp*:
 **assumes** *q*: *Quotient3 R Abs Rep*
  **and** *T-def*: $T \equiv \lambda x\ y.\ Abs\ x = y$
  **and** *eR*: *equivp R*
 **shows** *Quotient R Abs Rep T*
⟨*proof*⟩

## 44.8 ML setup

Auxiliary data for the quotient package

**named-theorems** *quot-equiv equivalence relation theorems*
 **and** *quot-respect respectfulness theorems*
 **and** *quot-preserve preservation theorems*
 **and** *id-simps identity simp rules for maps*
 **and** *quot-thm quotient theorems*
⟨*ML*⟩

**declare** [[*mapQ3 fun* = (*rel-fun*, *fun-quotient3*)]]

**lemmas** [*quot-thm*] = *fun-quotient3*
**lemmas** [*quot-respect*] = *quot-rel-rsp if-rsp o-rsp let-rsp id-rsp*
**lemmas** [*quot-preserve*] = *if-prs o-prs let-prs id-prs*
**lemmas** [*quot-equiv*] = *identity-equivp*

Lemmas about simplifying id's.

**lemmas** [*id-simps*] =
 *id-def*[*symmetric*]
 *map-fun-id*
 *id-apply*
 *id-o*
 *o-id*
 *eq-comp-r*
 *vimage-id*

Translation functions for the lifting process.

⟨*ML*⟩

Definitions of the quotient types.

⟨*ML*⟩

Definitions for quotient constants.

⟨*ML*⟩

An auxiliary constant for recording some information about the lifted theorem in a tactic.

**definition**
*Quot-True* :: $'a \Rightarrow bool$
**where**
*Quot-True* $x \longleftrightarrow$ *True*

**lemma**
**shows** *QT-all*: *Quot-True* (*All P*) $\Longrightarrow$ *Quot-True P*
**and** *QT-ex*: *Quot-True* (*Ex P*) $\Longrightarrow$ *Quot-True P*
**and** *QT-ex1*: *Quot-True* (*Ex1 P*) $\Longrightarrow$ *Quot-True P*
**and** *QT-lam*: *Quot-True* ($\lambda x.\ P\ x$) $\Longrightarrow$ ($\bigwedge x.$ *Quot-True* ($P\ x$))
**and** *QT-ext*: ($\bigwedge x.$ *Quot-True* ($a\ x$) $\Longrightarrow f\ x = g\ x$) $\Longrightarrow$ (*Quot-True* $a \Longrightarrow f = g$)
⟨*proof*⟩

**lemma** *QT-imp*: *Quot-True* $a \equiv$ *Quot-True* $b$
⟨*proof*⟩

**context includes** *lifting-syntax*
**begin**

Tactics for proving the lifted theorems

⟨*ML*⟩

**end**

## 44.9 Methods / Interface

⟨*ML*⟩

**no-notation** *rel-conj* (**infixr** ‹*OOO*› *75*)

# 45 Lifting of BNFs

**lemma** *sum-insert-Inl-unit*: $x \in A \Longrightarrow (\bigwedge y.\ x = Inr\ y \Longrightarrow Inr\ y \in B) \Longrightarrow x \in$ *insert* (*Inl* ()) *B*
⟨*proof*⟩

**lemma** *lift-sum-unit-vimage-commute*:
*insert* (*Inl* ()) (*Inr* ‘ *f* −‘ *A*) = *map-sum id f* −‘ *insert* (*Inl* ()) (*Inr* ‘ *A*)
⟨*proof*⟩

**lemma** *insert-Inl-int-map-sum-unit*: *insert* (*Inl* ()) *A* ∩ *range* (*map-sum id f*) ≠ {}
⟨*proof*⟩

**lemma** *image-map-sum-unit-subset*:
*A* ⊆ *insert* (*Inl* ()) (*Inr* ‘ *B*) $\Longrightarrow$ *map-sum id f* ‘ *A* ⊆ *insert* (*Inl* ()) (*Inr* ‘ *f* ‘ *B*)
⟨*proof*⟩

**lemma** *subset-lift-sum-unitD*: $A \subseteq insert\ (Inl\ ())\ (Inr\ `\ B) \Longrightarrow Inr\ x \in A \Longrightarrow x \in B$
⟨*proof*⟩

**lemma** *UNIV-sum-unit-conv*: $insert\ (Inl\ ())\ (range\ Inr) = UNIV$
⟨*proof*⟩

**lemma** *subset-vimage-image-subset*: $A \subseteq f\ -`\ B \Longrightarrow f\ `\ A \subseteq B$
⟨*proof*⟩

**lemma** *relcompp-mem-Grp-neq-bot*:
$A \cap range\ f \neq \{\} \Longrightarrow (\lambda x\ y.\ x \in A \wedge y \in A)\ OO\ (Grp\ UNIV\ f)^{-1-1} \neq bot$
⟨*proof*⟩

**lemma** *comp-projr-Inr*: $projr \circ Inr = id$
⟨*proof*⟩

**lemma** *in-rel-sum-in-image-projr*:
$B \subseteq \{(x,y).\ rel\text{-}sum\ ((=) :: unit \Rightarrow unit \Rightarrow bool)\ A\ x\ y\} \Longrightarrow$
$Inr\ `\ C = fst\ `\ B \Longrightarrow snd\ `\ B = Inr\ `\ D \Longrightarrow map\text{-}prod\ projr\ projr\ `\ B \subseteq \{(x,y).\ A\ x\ y\}$
⟨*proof*⟩

**lemma** *subset-rel-sumI*: $B \subseteq \{(x,y).\ A\ x\ y\} \Longrightarrow rel\text{-}sum\ ((=) :: unit => unit => bool)\ A$
$(if\ x \in B\ then\ Inr\ (fst\ x)\ else\ Inl\ ())$
$(if\ x \in B\ then\ Inr\ (snd\ x)\ else\ Inl\ ())$
⟨*proof*⟩

**lemma** *relcompp-eq-Grp-neq-bot*: $(=)\ OO\ (Grp\ UNIV\ f)^{-1-1} \neq bot$
⟨*proof*⟩

**lemma** *rel-fun-rel-OO1*: $(rel\text{-}fun\ Q\ (rel\text{-}fun\ R\ (=)))\ A\ B \Longrightarrow conversep\ Q\ OO\ A\ OO\ R \leq B$
⟨*proof*⟩

**lemma** *rel-fun-rel-OO2*: $(rel\text{-}fun\ Q\ (rel\text{-}fun\ R\ (=)))\ A\ B \Longrightarrow Q\ OO\ B\ OO\ conversep\ R \leq A$
⟨*proof*⟩

**lemma** *rel-sum-eq2-nonempty*: $rel\text{-}sum\ (=)\ A\ OO\ rel\text{-}sum\ (=)\ B \neq bot$
⟨*proof*⟩

**lemma** *rel-sum-eq3-nonempty*: $rel\text{-}sum\ (=)\ A\ OO\ (rel\text{-}sum\ (=)\ B\ OO\ rel\text{-}sum\ (=)\ C) \neq bot$
⟨*proof*⟩

**lemma** *hypsubst*: $A = B \Longrightarrow x \in B \Longrightarrow (x \in A \Longrightarrow P) \Longrightarrow P$ ⟨*proof*⟩

**lemma** *Quotient-crel-quotient*: $Quotient\ R\ Abs\ Rep\ T \Longrightarrow equivp\ R \Longrightarrow T \equiv (\lambda x\ y.\ Abs\ x = y)$
⟨*proof*⟩

**lemma** *Quotient-crel-typedef*: $Quotient\ (eq\text{-}onp\ P)\ Abs\ Rep\ T \Longrightarrow T \equiv (\lambda x\ y.\ x = Rep\ y)$
⟨*proof*⟩

**lemma** *Quotient-crel-typecopy*: $Quotient\ (=)\ Abs\ Rep\ T \Longrightarrow T \equiv (\lambda x\ y.\ x = Rep\ y)$
⟨*proof*⟩

**lemma** *equivp-add-relconj*:
**assumes** *equiv*: $equivp\ R\ equivp\ R'$ **and** *le*: $S\ OO\ T\ OO\ U \leq R\ OO\ STU\ OO\ R'$
**shows** $R\ OO\ S\ OO\ T\ OO\ U\ OO\ R' \leq R\ OO\ STU\ OO\ R'$
⟨*proof*⟩

**lemma** *Grp-conversep-eq-onp*: $((BNF\text{-}Def.Grp\ UNIV\ f)^{-1-1}\ OO\ BNF\text{-}Def.Grp\ UNIV\ f) = eq\text{-}onp\ (\lambda x.\ x \in range\ f)$
⟨*proof*⟩

**lemma** *Grp-conversep-nonempty*: $(BNF\text{-}Def.Grp\ UNIV\ f)^{-1-1}\ OO\ BNF\text{-}Def.Grp\ UNIV\ f \neq bot$
⟨*proof*⟩

**lemma** *relcomppI2*: $r\ a\ b \Longrightarrow s\ b\ c \Longrightarrow t\ c\ d \Longrightarrow (r\ OO\ s\ OO\ t)\ a\ d$
⟨*proof*⟩

**lemma** *rel-conj-eq-onp*: $equivp\ R \Longrightarrow rel\text{-}conj\ R\ (eq\text{-}onp\ P) \leq R$
⟨*proof*⟩

**lemma** *Quotient-Quotient3*: $Quotient\ R\ Abs\ Rep\ T \Longrightarrow Quotient3\ R\ Abs\ Rep$
⟨*proof*⟩

**lemma** *Quotient-reflp-imp-equivp*: $Quotient\ R\ Abs\ Rep\ T \Longrightarrow reflp\ R \Longrightarrow equivp\ R$
⟨*proof*⟩

**lemma** *Quotient-eq-onp-typedef*:
$Quotient\ (eq\text{-}onp\ P)\ Abs\ Rep\ cr \Longrightarrow type\text{-}definition\ Rep\ Abs\ \{x.\ P\ x\}$
⟨*proof*⟩

**lemma** *Quotient-eq-onp-type-copy*:
$Quotient\ (=)\ Abs\ Rep\ cr \Longrightarrow type\text{-}definition\ Rep\ Abs\ UNIV$
⟨*proof*⟩

⟨*ML*⟩

**hide-fact**

*sum-insert-Inl-unit lift-sum-unit-vimage-commute insert-Inl-int-map-sum-unit image-map-sum-unit-subset subset-lift-sum-unitD UNIV-sum-unit-conv subset-vimage-image-subset relcompp-mem-Grp-neq-bot comp-projr-Inr in-rel-sum-in-image-projr subset-rel-sumI relcompp-eq-Grp-neq-bot rel-fun-rel-OO1 rel-fun-rel-OO2 rel-sum-eq2-nonempty rel-sum-eq3-nonempty*
*hypsubst equivp-add-relconj Grp-conversep-eq-onp Grp-conversep-nonempty rel-comppI2 rel-conj-eq-onp*
*Quotient-reflp-imp-equivp Quotient-Quotient3*

**end**

# 46 Binary Numerals

**theory** *Num*
**imports** *BNF-Least-Fixpoint Transfer*
**begin**

## 46.1 The *num* type

**datatype** *num* = *One* | *Bit0 num* | *Bit1 num*

Increment function for type *num*

**primrec** *inc* :: ‹*num* ⇒ *num*›
**where**
‹*inc One* = *Bit0 One*›
| ‹*inc* (*Bit0 x*) = *Bit1 x*›
| ‹*inc* (*Bit1 x*) = *Bit0* (*inc x*)›

Converting between type *num* and type *nat*

**primrec** *nat-of-num* :: ‹*num* ⇒ *nat*›
**where**
‹*nat-of-num One* = *Suc 0*›
| ‹*nat-of-num* (*Bit0 x*) = *nat-of-num x* + *nat-of-num x*›
| ‹*nat-of-num* (*Bit1 x*) = *Suc* (*nat-of-num x* + *nat-of-num x*)›

**primrec** *num-of-nat* :: ‹*nat* ⇒ *num*›
**where**
‹*num-of-nat 0* = *One*›
| ‹*num-of-nat* (*Suc n*) = (*if 0* < *n then inc* (*num-of-nat n*) *else One*)›

**lemma** *nat-of-num-pos*: ‹*0* < *nat-of-num x*›
⟨*proof*⟩

**lemma** *nat-of-num-neq-0*: ‹ *nat-of-num x* ≠ *0*›
⟨*proof*⟩

**lemma** *nat-of-num-inc*: ‹*nat-of-num* (*inc x*) = *Suc* (*nat-of-num x*)›
⟨*proof*⟩

**lemma** *num-of-nat-double*: ‹$0 < n \Longrightarrow$ *num-of-nat* $(n + n) =$ *Bit0* (*num-of-nat* $n$)›
⟨*proof*⟩

Type *num* is isomorphic to the strictly positive natural numbers.

**lemma** *nat-of-num-inverse*: ‹*num-of-nat* (*nat-of-num* $x$) $= x$›
⟨*proof*⟩

**lemma** *num-of-nat-inverse*: ‹$0 < n \Longrightarrow$ *nat-of-num* (*num-of-nat* $n$) $= n$›
⟨*proof*⟩

**lemma** *num-eq-iff*: ‹$x = y \longleftrightarrow$ *nat-of-num* $x =$ *nat-of-num* $y$›
⟨*proof*⟩

**lemma** *num-induct* [*case-names One inc*]:
**fixes** $P$ :: ‹*num* $\Rightarrow$ *bool*›
**assumes** *One*: ‹$P$ *One*›
**and** *inc*: ‹$\bigwedge x.\ P\ x \Longrightarrow P$ (*inc* $x$)›
**shows** ‹$P\ x$›
⟨*proof*⟩

From now on, there are two possible models for *num*: as positive naturals (rule *num-induct*) and as digit representation (rules *num.induct*, *num.cases*).

## 46.2 Numeral operations

**instantiation** *num* :: ‹{*plus*,*times*,*linorder*}›
**begin**

**definition** [*code del*]: ‹$m + n =$ *num-of-nat* (*nat-of-num* $m$ + *nat-of-num* $n$)›

**definition** [*code del*]: ‹$m * n =$ *num-of-nat* (*nat-of-num* $m$ * *nat-of-num* $n$)›

**definition** [*code del*]: ‹$m \le n \longleftrightarrow$ *nat-of-num* $m \le$ *nat-of-num* $n$›

**definition** [*code del*]: ‹$m < n \longleftrightarrow$ *nat-of-num* $m <$ *nat-of-num* $n$›

**instance**
⟨*proof*⟩

**end**

**lemma** *nat-of-num-add*: ‹*nat-of-num* $(x + y) =$ *nat-of-num* $x$ + *nat-of-num* $y$›
⟨*proof*⟩

**lemma** *nat-of-num-mult*: ‹*nat-of-num* $(x * y) =$ *nat-of-num* $x$ * *nat-of-num* $y$›
⟨*proof*⟩

**lemma** *add-num-simps* [*simp*, *code*]:
‹$One + One = Bit0\ One$›
‹$One + Bit0\ n = Bit1\ n$›
‹$One + Bit1\ n = Bit0\ (n + One)$›
‹$Bit0\ m + One = Bit1\ m$›
‹$Bit0\ m + Bit0\ n = Bit0\ (m + n)$›
‹$Bit0\ m + Bit1\ n = Bit1\ (m + n)$›
‹$Bit1\ m + One = Bit0\ (m + One)$›
‹$Bit1\ m + Bit0\ n = Bit1\ (m + n)$›
‹$Bit1\ m + Bit1\ n = Bit0\ (m + n + One)$›
⟨*proof*⟩

**lemma** *mult-num-simps* [*simp*, *code*]:
‹$m * One = m$›
‹$One * n = n$›
‹$Bit0\ m * Bit0\ n = Bit0\ (Bit0\ (m * n))$›
‹$Bit0\ m * Bit1\ n = Bit0\ (m * Bit1\ n)$›
‹$Bit1\ m * Bit0\ n = Bit0\ (Bit1\ m * n)$›
‹$Bit1\ m * Bit1\ n = Bit1\ (m + n + Bit0\ (m * n))$›
⟨*proof*⟩

**lemma** *eq-num-simps*:
‹$One = One \longleftrightarrow True$›
‹$One = Bit0\ n \longleftrightarrow False$›
‹$One = Bit1\ n \longleftrightarrow False$›
‹$Bit0\ m = One \longleftrightarrow False$›
‹$Bit1\ m = One \longleftrightarrow False$›
‹$Bit0\ m = Bit0\ n \longleftrightarrow m = n$›
‹$Bit0\ m = Bit1\ n \longleftrightarrow False$›
‹$Bit1\ m = Bit0\ n \longleftrightarrow False$›
‹$Bit1\ m = Bit1\ n \longleftrightarrow m = n$›
⟨*proof*⟩

**lemma** *le-num-simps* [*simp*, *code*]:
‹$One \le n \longleftrightarrow True$›
‹$Bit0\ m \le One \longleftrightarrow False$›
‹$Bit1\ m \le One \longleftrightarrow False$›
‹$Bit0\ m \le Bit0\ n \longleftrightarrow m \le n$›
‹$Bit0\ m \le Bit1\ n \longleftrightarrow m \le n$›
‹$Bit1\ m \le Bit1\ n \longleftrightarrow m \le n$›
‹$Bit1\ m \le Bit0\ n \longleftrightarrow m < n$›
⟨*proof*⟩

**lemma** *less-num-simps* [*simp*, *code*]:
‹$m < One \longleftrightarrow False$›
‹$One < Bit0\ n \longleftrightarrow True$›
‹$One < Bit1\ n \longleftrightarrow True$›
‹$Bit0\ m < Bit0\ n \longleftrightarrow m < n$›
‹$Bit0\ m < Bit1\ n \longleftrightarrow m \le n$›

‹$Bit1\ m < Bit1\ n \longleftrightarrow m < n$›
‹$Bit1\ m < Bit0\ n \longleftrightarrow m < n$›
⟨*proof*⟩

**lemma** *le-num-One-iff*: ‹$x \leq One \longleftrightarrow x = One$›
⟨*proof*⟩

Rules using *One* and *inc* as constructors.

**lemma** *add-One*: ‹$x + One = inc\ x$›
⟨*proof*⟩

**lemma** *add-One-commute*: ‹$One + n = n + One$›
⟨*proof*⟩

**lemma** *add-inc*: ‹$x + inc\ y = inc\ (x + y)$›
⟨*proof*⟩

**lemma** *mult-inc*: ‹$x * inc\ y = x * y + x$›
⟨*proof*⟩

The *num-of-nat* conversion.

**lemma** *num-of-nat-One*: ‹$n \leq 1 \Longrightarrow num\text{-}of\text{-}nat\ n = One$›
⟨*proof*⟩

**lemma** *num-of-nat-plus-distrib*:
‹$0 < m \Longrightarrow 0 < n \Longrightarrow num\text{-}of\text{-}nat\ (m + n) = num\text{-}of\text{-}nat\ m + num\text{-}of\text{-}nat\ n$›
⟨*proof*⟩

A double-and-decrement function.

**primrec** *BitM* :: ‹$num \Rightarrow num$›
**where**
‹$BitM\ One = One$›
| ‹$BitM\ (Bit0\ n) = Bit1\ (BitM\ n)$›
| ‹$BitM\ (Bit1\ n) = Bit1\ (Bit0\ n)$›

**lemma** *BitM-plus-one*: ‹$BitM\ n + One = Bit0\ n$›
⟨*proof*⟩

**lemma** *one-plus-BitM*: ‹$One + BitM\ n = Bit0\ n$›
⟨*proof*⟩

**lemma** *BitM-inc-eq*:
‹$BitM\ (inc\ n) = Bit1\ n$›
⟨*proof*⟩

**lemma** *inc-BitM-eq*:
‹$inc\ (BitM\ n) = Bit0\ n$›
⟨*proof*⟩

Squaring and exponentiation.

**primrec** *sqr* :: ‹*num* ⇒ *num*›
  **where**
    ‹*sqr One* = *One*›
  | ‹*sqr* (*Bit0 n*) = *Bit0* (*Bit0* (*sqr n*))›
  | ‹*sqr* (*Bit1 n*) = *Bit1* (*Bit0* (*sqr n* + *n*))›

**primrec** *pow* :: ‹*num* ⇒ *num* ⇒ *num*›
  **where**
    ‹*pow x One* = *x*›
  | ‹*pow x* (*Bit0 y*) = *sqr* (*pow x y*)›
  | ‹*pow x* (*Bit1 y*) = *sqr* (*pow x y*) ∗ *x*›

**lemma** *nat-of-num-sqr*: ‹*nat-of-num* (*sqr x*) = *nat-of-num x* ∗ *nat-of-num x*›
  ⟨*proof*⟩

**lemma** *sqr-conv-mult*: ‹*sqr x* = *x* ∗ *x*›
  ⟨*proof*⟩

**lemma** *num-double* [*simp*]:
  ‹*Bit0 num.One* ∗ *n* = *Bit0 n*›
  ⟨*proof*⟩

## 46.3 Binary numerals

We embed binary representations into a generic algebraic structure using *numeral*.

**class** *numeral* = *one* + *semigroup-add*
**begin**

**primrec** *numeral* :: ‹*num* ⇒ ′*a*›
  **where**
    *numeral-One*: ‹*numeral One* = *1*›
  | *numeral-Bit0*: ‹*numeral* (*Bit0 n*) = *numeral n* + *numeral n*›
  | *numeral-Bit1*: ‹*numeral* (*Bit1 n*) = *numeral n* + *numeral n* + *1*›

**lemma** *numeral-code* [*code*]:
  ‹*numeral One* = *1*›
  ‹*numeral* (*Bit0 n*) = (*let m* = *numeral n in m* + *m*)›
  ‹*numeral* (*Bit1 n*) = (*let m* = *numeral n in m* + *m* + *1*)›
  ⟨*proof*⟩

**lemma** *one-plus-numeral-commute*: ‹*1* + *numeral x* = *numeral x* + *1*›
⟨*proof*⟩

**lemma** *numeral-inc*: ‹*numeral* (*inc x*) = *numeral x* + *1*›
⟨*proof*⟩

**declare** *numeral.simps* [*simp del*]

**abbreviation** ‹*Numeral1* ≡ *numeral One*›

**declare** *numeral-One* [*code-post*]

**end**

Numeral syntax.

**syntax**
*-Numeral* :: ‹*num-const* ⇒ *'a*› (‹(‹*open-block notation*=‹*literal number*››-)›)

⟨*ML*⟩

## 46.4 Class-specific numeral rules

*numeral* is a morphism.

### 46.4.1 Structures with addition: class *numeral*

**context** *numeral*
**begin**

**lemma** *numeral-add*: ‹*numeral* (*m* + *n*) = *numeral m* + *numeral n*›
⟨*proof*⟩

**lemma** *numeral-plus-numeral*: ‹*numeral m* + *numeral n* = *numeral* (*m* + *n*)›
⟨*proof*⟩

**lemma** *numeral-plus-one*: ‹*numeral n* + *1* = *numeral* (*n* + *One*)›
⟨*proof*⟩

**lemma** *one-plus-numeral*: ‹*1* + *numeral n* = *numeral* (*One* + *n*)›
⟨*proof*⟩

**lemma** *one-add-one*: ‹*1* + *1* = *2*›
⟨*proof*⟩

**lemmas** *add-numeral-special* =
*numeral-plus-one one-plus-numeral one-add-one*

**end**

### 46.4.2 Structures with negation: class *neg-numeral*

**class** *neg-numeral* = *numeral* + *group-add*
**begin**

**lemma** *uminus-numeral-One*: ‹− *Numeral1* = − *1*›

⟨*proof*⟩

Numerals form an abelian subgroup.

**inductive** *is-num* :: ‹$'a \Rightarrow bool$›
**where**
‹*is-num 1*›
| ‹*is-num* $x \Longrightarrow$ *is-num* $(- x)$›
| ‹*is-num* $x \Longrightarrow$ *is-num* $y \Longrightarrow$ *is-num* $(x + y)$›

**lemma** *is-num-numeral*: ‹*is-num* (*numeral k*)›
⟨*proof*⟩

**lemma** *is-num-add-commute*: ‹*is-num* $x \Longrightarrow$ *is-num* $y \Longrightarrow x + y = y + x$›
⟨*proof*⟩

**lemma** *is-num-add-left-commute*: ‹*is-num* $x \Longrightarrow$ *is-num* $y \Longrightarrow x + (y + z) = y + (x + z)$›
⟨*proof*⟩

**lemmas** *is-num-normalize* =
*add.assoc is-num-add-commute is-num-add-left-commute*
*is-num.intros is-num-numeral*
*minus-add*

**definition** *dbl* :: ‹$'a \Rightarrow 'a$›
**where** ‹*dbl* $x = x + x$›

**definition** *dbl-inc* :: ‹$'a \Rightarrow 'a$›
**where** ‹*dbl-inc* $x = x + x + 1$›

**definition** *dbl-dec* :: ‹$'a \Rightarrow 'a$›
**where** ‹*dbl-dec* $x = x + x - 1$›

**definition** *sub* :: ‹$num \Rightarrow num \Rightarrow 'a$›
**where** ‹*sub k l* = *numeral k* − *numeral l*›

**lemma** *numeral-BitM*: ‹*numeral* (*BitM n*) = *numeral* (*Bit0 n*) − *1*›
⟨*proof*⟩

**lemma** *sub-inc-One-eq*:
‹*sub* (*inc n*) *num.One* = *numeral n*›
⟨*proof*⟩

**lemma** *dbl-simps* [*simp*]:
‹*dbl* (− *numeral k*) = − *dbl* (*numeral k*)›
‹*dbl 0* = *0*›
‹*dbl 1* = *2*›
‹*dbl* (− *1*) = − *2*›
‹*dbl* (*numeral k*) = *numeral* (*Bit0 k*)›

⟨*proof*⟩

**lemma** *dbl-inc-simps* [*simp*]:
‹*dbl-inc* (− *numeral k*) = − *dbl-dec* (*numeral k*)›
‹*dbl-inc 0* = *1*›
‹*dbl-inc 1* = *3*›
‹*dbl-inc* (− *1*) = − *1*›
‹*dbl-inc* (*numeral k*) = *numeral* (*Bit1 k*)›
⟨*proof*⟩

**lemma** *dbl-dec-simps* [*simp*]:
‹*dbl-dec* (− *numeral k*) = − *dbl-inc* (*numeral k*)›
‹*dbl-dec 0* = − *1*›
‹*dbl-dec 1* = *1*›
‹*dbl-dec* (− *1*) = − *3*›
‹*dbl-dec* (*numeral k*) = *numeral* (*BitM k*)›
⟨*proof*⟩

**lemma** *sub-num-simps* [*simp*]:
‹*sub One One* = *0*›
‹*sub One* (*Bit0 l*) = − *numeral* (*BitM l*)›
‹*sub One* (*Bit1 l*) = − *numeral* (*Bit0 l*)›
‹*sub* (*Bit0 k*) *One* = *numeral* (*BitM k*)›
‹*sub* (*Bit1 k*) *One* = *numeral* (*Bit0 k*)›
‹*sub* (*Bit0 k*) (*Bit0 l*) = *dbl* (*sub k l*)›
‹*sub* (*Bit0 k*) (*Bit1 l*) = *dbl-dec* (*sub k l*)›
‹*sub* (*Bit1 k*) (*Bit0 l*) = *dbl-inc* (*sub k l*)›
‹*sub* (*Bit1 k*) (*Bit1 l*) = *dbl* (*sub k l*)›
⟨*proof*⟩

**lemma** *add-neg-numeral-simps*:
‹*numeral m* + − *numeral n* = *sub m n*›
‹− *numeral m* + *numeral n* = *sub n m*›
‹− *numeral m* + − *numeral n* = − (*numeral m* + *numeral n*)›
⟨*proof*⟩

**lemma** *add-neg-numeral-special*:
‹*1* + − *numeral m* = *sub One m*›
‹− *numeral m* + *1* = *sub One m*›
‹*numeral m* + − *1* = *sub m One*›
‹− *1* + *numeral n* = *sub n One*›
‹− *1* + − *numeral n* = − *numeral* (*inc n*)›
‹− *numeral m* + − *1* = − *numeral* (*inc m*)›
‹*1* + − *1* = *0*›
‹− *1* + *1* = *0*›
‹− *1* + − *1* = − *2*›
⟨*proof*⟩

**lemma** *diff-numeral-simps*:

‹$numeral\ m - numeral\ n = sub\ m\ n$›
‹$numeral\ m - - numeral\ n = numeral\ (m + n)$›
‹$- numeral\ m - numeral\ n = - numeral\ (m + n)$›
‹$- numeral\ m - - numeral\ n = sub\ n\ m$›
⟨*proof*⟩

**lemma** *diff-numeral-special*:
‹$1 - numeral\ n = sub\ One\ n$›
‹$numeral\ m - 1 = sub\ m\ One$›
‹$1 - - numeral\ n = numeral\ (One + n)$›
‹$- numeral\ m - 1 = - numeral\ (m + One)$›
‹$- 1 - numeral\ n = - numeral\ (inc\ n)$›
‹$numeral\ m - - 1 = numeral\ (inc\ m)$›
‹$- 1 - - numeral\ n = sub\ n\ One$›
‹$- numeral\ m - - 1 = sub\ One\ m$›
‹$1 - 1 = 0$›
‹$- 1 - 1 = - 2$›
‹$1 - - 1 = 2$›
‹$- 1 - - 1 = 0$›
⟨*proof*⟩

**end**

### 46.4.3 Structures with multiplication: class *semiring-numeral*

**class** *semiring-numeral* = *semiring* + *monoid-mult*
**begin**

**subclass** *numeral* ⟨*proof*⟩

**lemma** *numeral-mult*: ‹$numeral\ (m * n) = numeral\ m * numeral\ n$›
⟨*proof*⟩

**lemma** *numeral-times-numeral*: ‹$numeral\ m * numeral\ n = numeral\ (m * n)$›
⟨*proof*⟩

**lemma** *mult-2*: ‹$2 * z = z + z$›
⟨*proof*⟩

**lemma** *mult-2-right*: ‹$z * 2 = z + z$›
⟨*proof*⟩

**lemma** *left-add-twice*:
‹$a + (a + b) = 2 * a + b$›
⟨*proof*⟩

**lemma** *numeral-Bit0-eq-double*:
‹$numeral\ (Bit0\ n) = 2 * numeral\ n$›
⟨*proof*⟩

**lemma** *numeral-Bit1-eq-inc-double*:
‹*numeral* (*Bit1 n*) = *2* ∗ *numeral n* + *1*›
⟨*proof*⟩

**end**

### 46.4.4 Structures with a zero: class *semiring-1*

**context** *semiring-1*
**begin**

**subclass** *semiring-numeral* ⟨*proof*⟩

**lemma** *of-nat-numeral* [*simp*]: ‹*of-nat* (*numeral n*) = *numeral n*›
⟨*proof*⟩

**end**

**lemma** *nat-of-num-numeral* [*code-abbrev*]: ‹*nat-of-num* = *numeral*›
⟨*proof*⟩

**lemma** *nat-of-num-code* [*code*]:
‹*nat-of-num One* = *1*›
‹*nat-of-num* (*Bit0 n*) = (*let m* = *nat-of-num n in m* + *m*)›
‹*nat-of-num* (*Bit1 n*) = (*let m* = *nat-of-num n in Suc* (*m* + *m*))›
⟨*proof*⟩

### 46.4.5 Equality: class *semiring-char-0*

**context** *semiring-char-0*
**begin**

**lemma** *numeral-eq-iff*: ‹*numeral m* = *numeral n* ⟷ *m* = *n*›
⟨*proof*⟩

**lemma** *numeral-eq-one-iff*: ‹*numeral n* = *1* ⟷ *n* = *One*›
⟨*proof*⟩

**lemma** *one-eq-numeral-iff*: ‹*1* = *numeral n* ⟷ *One* = *n*›
⟨*proof*⟩

**lemma** *numeral-neq-zero*: ‹*numeral n* ≠ *0*›
⟨*proof*⟩

**lemma** *zero-neq-numeral*: ‹*0* ≠ *numeral n*›
⟨*proof*⟩

**lemmas** *eq-numeral-simps* [*simp*] =
*numeral-eq-iff*

*numeral-eq-one-iff*
*one-eq-numeral-iff*
*numeral-neq-zero*
*zero-neq-numeral*

**end**

### 46.4.6 Comparisons: class *linordered-nonzero-semiring*

**context** *linordered-nonzero-semiring*
**begin**

**lemma** *numeral-le-iff*: ‹$numeral\ m \le numeral\ n \longleftrightarrow m \le n$›
⟨*proof*⟩

**lemma** *one-le-numeral*: ‹$1 \le numeral\ n$›
⟨*proof*⟩

**lemma** *numeral-le-one-iff*: ‹$numeral\ n \le 1 \longleftrightarrow n \le One$›
⟨*proof*⟩

**lemma** *numeral-less-iff*: ‹$numeral\ m < numeral\ n \longleftrightarrow m < n$›
⟨*proof*⟩

**lemma** *not-numeral-less-one*: ‹$\neg\ numeral\ n < 1$›
⟨*proof*⟩

**lemma** *one-less-numeral-iff*: ‹$1 < numeral\ n \longleftrightarrow One < n$›
⟨*proof*⟩

**lemma** *zero-le-numeral*: ‹$0 \le numeral\ n$›
⟨*proof*⟩

**lemma** *zero-less-numeral*: ‹$0 < numeral\ n$›
⟨*proof*⟩

**lemma** *not-numeral-le-zero*: ‹$\neg\ numeral\ n \le 0$›
⟨*proof*⟩

**lemma** *not-numeral-less-zero*: ‹$\neg\ numeral\ n < 0$›
⟨*proof*⟩

**lemma** *one-of-nat-le-iff* [*simp*]: ‹$1 \le of\text{-}nat\ k \longleftrightarrow 1 \le k$›
⟨*proof*⟩

**lemma** *numeral-nat-le-iff* [*simp*]: ‹$numeral\ n \le of\text{-}nat\ k \longleftrightarrow numeral\ n \le k$›
⟨*proof*⟩

**lemma** *of-nat-le-1-iff* [*simp*]: ‹$of\text{-}nat\ k \le 1 \longleftrightarrow k \le 1$›

⟨proof⟩

**lemma** *of-nat-le-numeral-iff* [*simp*]: ‹$\mathit{of\text{-}nat}\ k \le \mathit{numeral}\ n \longleftrightarrow k \le \mathit{numeral}\ n$›
⟨proof⟩

**lemma** *one-of-nat-less-iff* [*simp*]: ‹$1 < \mathit{of\text{-}nat}\ k \longleftrightarrow 1 < k$›
⟨proof⟩

**lemma** *numeral-nat-less-iff* [*simp*]: ‹$\mathit{numeral}\ n < \mathit{of\text{-}nat}\ k \longleftrightarrow \mathit{numeral}\ n < k$›
⟨proof⟩

**lemma** *of-nat-less-1-iff* [*simp*]: ‹$\mathit{of\text{-}nat}\ k < 1 \longleftrightarrow k < 1$›
⟨proof⟩

**lemma** *of-nat-less-numeral-iff* [*simp*]: ‹$\mathit{of\text{-}nat}\ k < \mathit{numeral}\ n \longleftrightarrow k < \mathit{numeral}\ n$›
⟨proof⟩

**lemma** *of-nat-eq-numeral-iff* [*simp*]: ‹$\mathit{of\text{-}nat}\ k = \mathit{numeral}\ n \longleftrightarrow k = \mathit{numeral}\ n$›
⟨proof⟩

**lemmas** *le-numeral-extra* =
*zero-le-one not-one-le-zero*
*order-refl* [*of 0*] *order-refl* [*of 1*]

**lemmas** *less-numeral-extra* =
*zero-less-one not-one-less-zero*
*less-irrefl* [*of 0*] *less-irrefl* [*of 1*]

**lemmas** *le-numeral-simps* [*simp*] =
*numeral-le-iff*
*one-le-numeral*
*numeral-le-one-iff*
*zero-le-numeral*
*not-numeral-le-zero*

**lemmas** *less-numeral-simps* [*simp*] =
*numeral-less-iff*
*one-less-numeral-iff*
*not-numeral-less-one*
*zero-less-numeral*
*not-numeral-less-zero*

**lemma** *min-0-1* [*simp*]:
**fixes** *min'* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$›
**defines** ‹$\mathit{min}' \equiv \mathit{min}$›
**shows**
‹$\mathit{min}'\ 0\ 1 = 0$›
‹$\mathit{min}'\ 1\ 0 = 0$›
‹$\mathit{min}'\ 0\ (\mathit{numeral}\ x) = 0$›

‹*min′ (numeral x) 0 = 0*›
‹*min′ 1 (numeral x) = 1*›
‹*min′ (numeral x) 1 = 1*›
⟨*proof*⟩

**lemma** *max-0-1* [*simp*]:
**fixes** *max′* :: ‹*′a ⇒ ′a ⇒ ′a*›
**defines** ‹*max′ ≡ max*›
**shows**
‹*max′ 0 1 = 1*›
‹*max′ 1 0 = 1*›
‹*max′ 0 (numeral x) = numeral x*›
‹*max′ (numeral x) 0 = numeral x*›
‹*max′ 1 (numeral x) = numeral x*›
‹*max′ (numeral x) 1 = numeral x*›
⟨*proof*⟩

**end**

Unfold *min* and *max* on numerals.

**lemmas** *max-number-of* [*simp*] =
*max-def* [*of* ‹*numeral u*› ‹*numeral v*›]
*max-def* [*of* ‹*numeral u*› ‹*− numeral v*›]
*max-def* [*of* ‹*− numeral u*› ‹*numeral v*›]
*max-def* [*of* ‹*− numeral u*› ‹*− numeral v*›] **for** *u v*

**lemmas** *min-number-of* [*simp*] =
*min-def* [*of* ‹*numeral u*› ‹*numeral v*›]
*min-def* [*of* ‹*numeral u*› ‹*− numeral v*›]
*min-def* [*of* ‹*− numeral u*› ‹*numeral v*›]
*min-def* [*of* ‹*− numeral u*› ‹*− numeral v*›] **for** *u v*

### 46.4.7 Multiplication and negation: class *ring-1*

**context** *ring-1*
**begin**

**subclass** *neg-numeral* ⟨*proof*⟩

**lemma** *mult-neg-numeral-simps*:
‹*− numeral m ∗ − numeral n = numeral (m ∗ n)*›
‹*− numeral m ∗ numeral n = − numeral (m ∗ n)*›
‹*numeral m ∗ − numeral n = − numeral (m ∗ n)*›
⟨*proof*⟩

**lemma** *mult-minus1* [*simp*]: ‹*− 1 ∗ z = − z*›
⟨*proof*⟩

**lemma** *mult-minus1-right* [*simp*]: ‹*z ∗ − 1 = − z*›

⟨*proof*⟩

**lemma** *minus-sub-one-diff-one* [*simp*]:
‹$-$ *sub m One* $-$ *1* $=$ $-$ *numeral m*›
⟨*proof*⟩

**end**

### 46.4.8 Equality using *iszero* for rings with non-zero characteristic

**context** *ring-1*
**begin**

**definition** *iszero* :: ‹*'a* $\Rightarrow$ *bool*›
**where** ‹*iszero z* $\longleftrightarrow$ *z* $=$ *0*›

**lemma** *iszero-0* [*simp*]: ‹*iszero 0*›
⟨*proof*⟩

**lemma** *not-iszero-1* [*simp*]: ‹$\neg$ *iszero 1*›
⟨*proof*⟩

**lemma** *not-iszero-Numeral1*: ‹$\neg$ *iszero Numeral1*›
⟨*proof*⟩

**lemma** *not-iszero-neg-1* [*simp*]: ‹$\neg$ *iszero* ($-$ *1*)›
⟨*proof*⟩

**lemma** *not-iszero-neg-Numeral1*: ‹$\neg$ *iszero* ($-$ *Numeral1*)›
⟨*proof*⟩

**lemma** *iszero-neg-numeral* [*simp*]: ‹*iszero* ($-$ *numeral w*) $\longleftrightarrow$ *iszero* (*numeral w*)›
⟨*proof*⟩

**lemma** *eq-iff-iszero-diff*: ‹*x* $=$ *y* $\longleftrightarrow$ *iszero* (*x* $-$ *y*)›
⟨*proof*⟩

The *eq-numeral-iff-iszero* lemmas are not declared [*simp*] by default, because for rings of characteristic zero, better simp rules are possible. For a type like integers mod $n$, type-instantiated versions of these rules should be added to the simplifier, along with a type-specific rule for deciding propositions of the form *iszero* (*numeral w*).

bh: Maybe it would not be so bad to just declare these as simp rules anyway? I should test whether these rules take precedence over the *ring-char-0* rules in the simplifier.

**lemma** *eq-numeral-iff-iszero*:
‹*numeral x* $=$ *numeral y* $\longleftrightarrow$ *iszero* (*sub x y*)›
‹*numeral x* $=$ $-$ *numeral y* $\longleftrightarrow$ *iszero* (*numeral* (*x* $+$ *y*))›

‹$-$ *numeral x* $=$ *numeral y* $\longleftrightarrow$ *iszero* (*numeral* ($x$ + $y$))›
‹$-$ *numeral x* $= -$ *numeral y* $\longleftrightarrow$ *iszero* (*sub y x*)›
‹*numeral x* $=$ *1* $\longleftrightarrow$ *iszero* (*sub x One*)›
‹*1* $=$ *numeral y* $\longleftrightarrow$ *iszero* (*sub One y*)›
‹$-$ *numeral x* $=$ *1* $\longleftrightarrow$ *iszero* (*numeral* ($x$ + *One*))›
‹*1* $= -$ *numeral y* $\longleftrightarrow$ *iszero* (*numeral* (*One* + $y$))›
‹*numeral x* $=$ *0* $\longleftrightarrow$ *iszero* (*numeral x*)›
‹*0* $=$ *numeral y* $\longleftrightarrow$ *iszero* (*numeral y*)›
‹$-$ *numeral x* $=$ *0* $\longleftrightarrow$ *iszero* (*numeral x*)›
‹*0* $= -$ *numeral y* $\longleftrightarrow$ *iszero* (*numeral y*)›
⟨*proof*⟩

**end**

### 46.4.9 Equality and negation: class *ring-char-0*

**context** *ring-char-0*
**begin**

**lemma** *not-iszero-numeral* [*simp*]: ‹¬ *iszero* (*numeral w*)›
⟨*proof*⟩

**lemma** *neg-numeral-eq-iff*: ‹$-$ *numeral m* $= -$ *numeral n* $\longleftrightarrow$ $m = n$›
⟨*proof*⟩

**lemma** *numeral-neq-neg-numeral*: ‹*numeral m* $\neq -$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-neq-numeral*: ‹$-$ *numeral m* $\neq$ *numeral n*›
⟨*proof*⟩

**lemma** *zero-neq-neg-numeral*: ‹*0* $\neq -$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-neq-zero*: ‹$-$ *numeral n* $\neq$ *0*›
⟨*proof*⟩

**lemma** *one-neq-neg-numeral*: ‹*1* $\neq -$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-neq-one*: ‹$-$ *numeral n* $\neq$ *1*›
⟨*proof*⟩

**lemma** *neg-one-neq-numeral*: ‹$-$ *1* $\neq$ *numeral n*›
⟨*proof*⟩

**lemma** *numeral-neq-neg-one*: ‹*numeral n* $\neq -$ *1*›
⟨*proof*⟩

**lemma** *neg-one-eq-numeral-iff*: ‹$- 1 = -$ *numeral* $n \longleftrightarrow n =$ *One*›
⟨*proof*⟩

**lemma** *numeral-eq-neg-one-iff*: ‹$-$ *numeral* $n = - 1 \longleftrightarrow n =$ *One*›
⟨*proof*⟩

**lemma** *neg-one-neq-zero*: ‹$- 1 \neq 0$›
⟨*proof*⟩

**lemma** *zero-neq-neg-one*: ‹$0 \neq - 1$›
⟨*proof*⟩

**lemma** *neg-one-neq-one*: ‹$- 1 \neq 1$›
⟨*proof*⟩

**lemma** *one-neq-neg-one*: ‹$1 \neq - 1$›
⟨*proof*⟩

**lemmas** *eq-neg-numeral-simps* [*simp*] =
*neg-numeral-eq-iff*
*numeral-neq-neg-numeral neg-numeral-neq-numeral*
*one-neq-neg-numeral neg-numeral-neq-one*
*zero-neq-neg-numeral neg-numeral-neq-zero*
*neg-one-neq-numeral numeral-neq-neg-one*
*neg-one-eq-numeral-iff numeral-eq-neg-one-iff*
*neg-one-neq-zero zero-neq-neg-one*
*neg-one-neq-one one-neq-neg-one*

**end**

### 46.4.10 Structures with negation and order: class *linordered-idom*

**context** *linordered-idom*
**begin**

**subclass** *ring-char-0* ⟨*proof*⟩

**lemma** *neg-numeral-le-iff*: ‹$-$ *numeral* $m \leq -$ *numeral* $n \longleftrightarrow n \leq m$›
⟨*proof*⟩

**lemma** *neg-numeral-less-iff*: ‹$-$ *numeral* $m < -$ *numeral* $n \longleftrightarrow n < m$›
⟨*proof*⟩

**lemma** *neg-numeral-less-zero*: ‹$-$ *numeral* $n < 0$›
⟨*proof*⟩

**lemma** *neg-numeral-le-zero*: ‹$-$ *numeral* $n \leq 0$›
⟨*proof*⟩

**lemma** *not-zero-less-neg-numeral*: ‹$\neg$ *0* $<$ $-$ *numeral n*›
⟨*proof*⟩

**lemma** *not-zero-le-neg-numeral*: ‹$\neg$ *0* $\leq$ $-$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-less-numeral*: ‹$-$ *numeral m* $<$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-le-numeral*: ‹$-$ *numeral m* $\leq$ *numeral n*›
⟨*proof*⟩

**lemma** *not-numeral-less-neg-numeral*: ‹$\neg$ *numeral m* $<$ $-$ *numeral n*›
⟨*proof*⟩

**lemma** *not-numeral-le-neg-numeral*: ‹$\neg$ *numeral m* $\leq$ $-$ *numeral n*›
⟨*proof*⟩

**lemma** *neg-numeral-less-one*: ‹$-$ *numeral m* $<$ *1*›
⟨*proof*⟩

**lemma** *neg-numeral-le-one*: ‹$-$ *numeral m* $\leq$ *1*›
⟨*proof*⟩

**lemma** *not-one-less-neg-numeral*: ‹$\neg$ *1* $<$ $-$ *numeral m*›
⟨*proof*⟩

**lemma** *not-one-le-neg-numeral*: ‹$\neg$ *1* $\leq$ $-$ *numeral m*›
⟨*proof*⟩

**lemma** *not-numeral-less-neg-one*: ‹$\neg$ *numeral m* $<$ $-$ *1*›
⟨*proof*⟩

**lemma** *not-numeral-le-neg-one*: ‹$\neg$ *numeral m* $\leq$ $-$ *1*›
⟨*proof*⟩

**lemma** *neg-one-less-numeral*: ‹$-$ *1* $<$ *numeral m*›
⟨*proof*⟩

**lemma** *neg-one-le-numeral*: ‹$-$ *1* $\leq$ *numeral m*›
⟨*proof*⟩

**lemma** *neg-numeral-less-neg-one-iff*: ‹$-$ *numeral m* $<$ $-$ *1* $\longleftrightarrow$ *m* $\neq$ *One*›
⟨*proof*⟩

**lemma** *neg-numeral-le-neg-one*: ‹$-$ *numeral m* $\leq$ $-$ *1*›
⟨*proof*⟩

**lemma** *not-neg-one-less-neg-numeral*: ‹$\neg$ $-$ *1* $<$ $-$ *numeral m*›

⟨proof⟩

**lemma** *not-neg-one-le-neg-numeral-iff*: ‹$\neg - 1 \leq - numeral\ m \longleftrightarrow m \neq One$›
⟨proof⟩

**lemma** *sub-non-negative*: ‹$sub\ n\ m \geq 0 \longleftrightarrow n \geq m$›
⟨proof⟩

**lemma** *sub-positive*: ‹$sub\ n\ m > 0 \longleftrightarrow n > m$›
⟨proof⟩

**lemma** *sub-non-positive*: ‹$sub\ n\ m \leq 0 \longleftrightarrow n \leq m$›
⟨proof⟩

**lemma** *sub-negative*: ‹$sub\ n\ m < 0 \longleftrightarrow n < m$›
⟨proof⟩

**lemmas** *le-neg-numeral-simps* [*simp*] =
*neg-numeral-le-iff*
*neg-numeral-le-numeral not-numeral-le-neg-numeral*
*neg-numeral-le-zero not-zero-le-neg-numeral*
*neg-numeral-le-one not-one-le-neg-numeral*
*neg-one-le-numeral not-numeral-le-neg-one*
*neg-numeral-le-neg-one not-neg-one-le-neg-numeral-iff*

**lemma** *le-minus-one-simps* [*simp*]:
‹$- 1 \leq 0$›
‹$- 1 \leq 1$›
‹$\neg 0 \leq - 1$›
‹$\neg 1 \leq - 1$›
⟨proof⟩

**lemmas** *less-neg-numeral-simps* [*simp*] =
*neg-numeral-less-iff*
*neg-numeral-less-numeral not-numeral-less-neg-numeral*
*neg-numeral-less-zero not-zero-less-neg-numeral*
*neg-numeral-less-one not-one-less-neg-numeral*
*neg-one-less-numeral not-numeral-less-neg-one*
*neg-numeral-less-neg-one-iff not-neg-one-less-neg-numeral*

**lemma** *less-minus-one-simps* [*simp*]:
‹$- 1 < 0$›
‹$- 1 < 1$›
‹$\neg 0 < - 1$›
‹$\neg 1 < - 1$›
⟨proof⟩

**lemma** *abs-numeral* [*simp*]: ‹$|numeral\ n| = numeral\ n$›
⟨proof⟩

**lemma** *abs-neg-numeral* [*simp*]: ‹|− *numeral n*| = *numeral n*›
⟨*proof*⟩

**lemma** *abs-neg-one* [*simp*]: ‹|− *1*| = *1*›
⟨*proof*⟩

**end**

### 46.4.11 Natural numbers

**lemma** *numeral-num-of-nat*:
‹*numeral* (*num-of-nat n*) = *n*› **if** ‹*n* > *0*›
⟨*proof*⟩

**lemma** *Suc-1* [*simp*]: ‹*Suc 1* = *2*›
⟨*proof*⟩

**lemma** *Suc-numeral* [*simp*]: ‹*Suc* (*numeral n*) = *numeral* (*n* + *One*)›
⟨*proof*⟩

**definition** *pred-numeral* :: ‹*num* ⇒ *nat*›
**where** ‹*pred-numeral k* = *numeral k* − *1*›

**declare** [[*code drop*: *pred-numeral*]]

**lemma** *numeral-eq-Suc*: ‹*numeral k* = *Suc* (*pred-numeral k*)›
⟨*proof*⟩

**lemma** *eval-nat-numeral*:
‹*numeral One* = *Suc 0*›
‹*numeral* (*Bit0 n*) = *Suc* (*numeral* (*BitM n*))›
‹*numeral* (*Bit1 n*) = *Suc* (*numeral* (*Bit0 n*))›
⟨*proof*⟩

**lemma** *pred-numeral-simps* [*simp*]:
‹*pred-numeral One* = *0*›
‹*pred-numeral* (*Bit0 k*) = *numeral* (*BitM k*)›
‹*pred-numeral* (*Bit1 k*) = *numeral* (*Bit0 k*)›
⟨*proof*⟩

**lemma** *pred-numeral-inc* [*simp*]:
‹*pred-numeral* (*inc k*) = *numeral k*›
⟨*proof*⟩

**lemma** *numeral-2-eq-2*: ‹*2* = *Suc* (*Suc 0*)›
⟨*proof*⟩

**lemma** *numeral-3-eq-3*: ‹*3* = *Suc* (*Suc* (*Suc 0*))›

⟨proof⟩

**lemma** *numeral-1-eq-Suc-0*: ‹$Numeral1 = Suc\ 0$›
⟨proof⟩

**lemma** *Suc-nat-number-of-add*: ‹$Suc\ (numeral\ v + n) = numeral\ (v + One) + n$›
⟨proof⟩

**lemma** *numerals*: ‹$Numeral1 = (1::nat)$› ‹$2 = Suc\ (Suc\ 0)$›
⟨proof⟩

**lemmas** *numeral-nat = eval-nat-numeral BitM.simps One-nat-def*

Comparisons involving *Suc*.

**lemma** *eq-numeral-Suc* [*simp*]: ‹$numeral\ k = Suc\ n \longleftrightarrow pred\text{-}numeral\ k = n$›
⟨proof⟩

**lemma** *Suc-eq-numeral* [*simp*]: ‹$Suc\ n = numeral\ k \longleftrightarrow n = pred\text{-}numeral\ k$›
⟨proof⟩

**lemma** *less-numeral-Suc* [*simp*]: ‹$numeral\ k < Suc\ n \longleftrightarrow pred\text{-}numeral\ k < n$›
⟨proof⟩

**lemma** *less-Suc-numeral* [*simp*]: ‹$Suc\ n < numeral\ k \longleftrightarrow n < pred\text{-}numeral\ k$›
⟨proof⟩

**lemma** *le-numeral-Suc* [*simp*]: ‹$numeral\ k \leq Suc\ n \longleftrightarrow pred\text{-}numeral\ k \leq n$›
⟨proof⟩

**lemma** *le-Suc-numeral* [*simp*]: ‹$Suc\ n \leq numeral\ k \longleftrightarrow n \leq pred\text{-}numeral\ k$›
⟨proof⟩

**lemma** *diff-Suc-numeral* [*simp*]: ‹$Suc\ n - numeral\ k = n - pred\text{-}numeral\ k$›
⟨proof⟩

**lemma** *diff-numeral-Suc* [*simp*]: ‹$numeral\ k - Suc\ n = pred\text{-}numeral\ k - n$›
⟨proof⟩

**lemma** *max-Suc-numeral* [*simp*]: ‹$max\ (Suc\ n)\ (numeral\ k) = Suc\ (max\ n\ (pred\text{-}numeral\ k))$›
⟨proof⟩

**lemma** *max-numeral-Suc* [*simp*]: ‹$max\ (numeral\ k)\ (Suc\ n) = Suc\ (max\ (pred\text{-}numeral\ k)\ n)$›
⟨proof⟩

**lemma** *min-Suc-numeral* [*simp*]: ‹$min\ (Suc\ n)\ (numeral\ k) = Suc\ (min\ n\ (pred\text{-}numeral\ k))$›
⟨proof⟩

**lemma** *min-numeral-Suc* [*simp*]: ‹*min* (*numeral k*) (*Suc n*) = *Suc* (*min* (*pred-numeral k*) *n*)›
⟨*proof*⟩

For *case-nat* and *rec-nat*.

**lemma** *case-nat-numeral* [*simp*]: ‹*case-nat a f* (*numeral v*) = (*let pv* = *pred-numeral v in f pv*)›
⟨*proof*⟩

**lemma** *case-nat-add-eq-if* [*simp*]:
‹*case-nat a f* ((*numeral v*) + *n*) = (*let pv* = *pred-numeral v in f* (*pv* + *n*))›
⟨*proof*⟩

**lemma** *rec-nat-numeral* [*simp*]:
‹*rec-nat a f* (*numeral v*) = (*let pv* = *pred-numeral v in f pv* (*rec-nat a f pv*))›
⟨*proof*⟩

**lemma** *rec-nat-add-eq-if* [*simp*]:
‹*rec-nat a f* (*numeral v* + *n*) = (*let pv* = *pred-numeral v in f* (*pv* + *n*) (*rec-nat a f* (*pv* + *n*)))›
⟨*proof*⟩

Case analysis on $n < (2::'a)$.

**lemma** *less-2-cases*: ‹$n < 2 \Longrightarrow n = 0 \vee n = Suc\ 0$›
⟨*proof*⟩

**lemma** *less-2-cases-iff*: ‹$n < 2 \longleftrightarrow n = 0 \vee n = Suc\ 0$›
⟨*proof*⟩

Removal of Small Numerals: 0, 1 and (in additive positions) 2.

bh: Are these rules really a good idea? LCP: well, it already happens for 0 and 1!

**lemma** *add-2-eq-Suc* [*simp*]: ‹$2 + n = Suc\ (Suc\ n)$›
⟨*proof*⟩

**lemma** *add-2-eq-Suc'* [*simp*]: ‹$n + 2 = Suc\ (Suc\ n)$›
⟨*proof*⟩

Can be used to eliminate long strings of Sucs, but not by default.

**lemma** *Suc3-eq-add-3*: ‹$Suc\ (Suc\ (Suc\ n)) = 3 + n$›
⟨*proof*⟩

**lemmas** *nat-1-add-1* = *one-add-one* [**where** *'a*=*nat*]

**context** *semiring-numeral*
**begin**

**lemma** *numeral-add-unfold-funpow*:
‹*numeral k* + *a* = ((+) *1* ^^ *numeral k*) *a*›
⟨*proof*⟩

**end**

**context** *semiring-1*
**begin**

**lemma** *numeral-unfold-funpow*:
‹*numeral k* = ((+) *1* ^^ *numeral k*) *0*›
⟨*proof*⟩

**end**

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *transfer-rule-numeral*:
‹((=) ===> *R*) *numeral numeral*›
**if** [*transfer-rule*]: ‹*R 0 0*› ‹*R 1 1*›
‹(*R* ===> *R* ===> *R*) (+) (+)›
**for** *R* :: ‹'*a*::{*semiring-numeral*,*monoid-add*} ⇒ '*b*::{*semiring-numeral*,*monoid-add*} ⇒ *bool*›
⟨*proof*⟩

**end**

## 46.5 Particular lemmas concerning *2*::'*a*

**context** *linordered-field*
**begin**

**subclass** *field-char-0* ⟨*proof*⟩

**lemma** *half-gt-zero-iff*: ‹*0* < *a* / *2* ⟷ *0* < *a*›
⟨*proof*⟩

**lemma** *half-gt-zero* [*simp*]: ‹*0* < *a* ⟹ *0* < *a* / *2*›
⟨*proof*⟩

**end**

## 46.6 Numeral equations as default simplification rules

**declare** (**in** *numeral*) *numeral-One* [*simp*]
**declare** (**in** *numeral*) *numeral-plus-numeral* [*simp*]
**declare** (**in** *numeral*) *add-numeral-special* [*simp*]

**declare** (**in** *neg-numeral*) *add-neg-numeral-simps* [*simp*]
**declare** (**in** *neg-numeral*) *add-neg-numeral-special* [*simp*]
**declare** (**in** *neg-numeral*) *diff-numeral-simps* [*simp*]
**declare** (**in** *neg-numeral*) *diff-numeral-special* [*simp*]
**declare** (**in** *semiring-numeral*) *numeral-times-numeral* [*simp*]
**declare** (**in** *ring-1*) *mult-neg-numeral-simps* [*simp*]

### 46.6.1 Special Simplification for Constants

These distributive laws move literals inside sums and differences.

**lemmas** *distrib-right-numeral* [*simp*] = *distrib-right* [*of - - ‹numeral v›*] **for** *v*
**lemmas** *distrib-left-numeral* [*simp*] = *distrib-left* [*of ‹numeral v›*] **for** *v*
**lemmas** *left-diff-distrib-numeral* [*simp*] = *left-diff-distrib* [*of - - ‹numeral v›*] **for** *v*
**lemmas** *right-diff-distrib-numeral* [*simp*] = *right-diff-distrib* [*of ‹numeral v›*] **for** *v*

These are actually for fields, like real

**lemmas** *zero-less-divide-iff-numeral* [*simp*, *no-atp*] = *zero-less-divide-iff* [*of ‹numeral w›*] **for** *w*
**lemmas** *divide-less-0-iff-numeral* [*simp*, *no-atp*] = *divide-less-0-iff* [*of ‹numeral w›*] **for** *w*
**lemmas** *zero-le-divide-iff-numeral* [*simp*, *no-atp*] = *zero-le-divide-iff* [*of ‹numeral w›*] **for** *w*
**lemmas** *divide-le-0-iff-numeral* [*simp*, *no-atp*] = *divide-le-0-iff* [*of ‹numeral w›*] **for** *w*

Replaces *inverse #nn* by *1/#nn*. It looks strange, but then other simprocs simplify the quotient.

**lemmas** *inverse-eq-divide-numeral* [*simp*] =
*inverse-eq-divide* [*of ‹numeral w›*] **for** *w*

**lemmas** *inverse-eq-divide-neg-numeral* [*simp*] =
*inverse-eq-divide* [*of ‹− numeral w›*] **for** *w*

These laws simplify inequalities, moving unary minus from a term into the literal.

**lemmas** *equation-minus-iff-numeral* [*no-atp*] =
*equation-minus-iff* [*of ‹numeral v›*] **for** *v*

**lemmas** *minus-equation-iff-numeral* [*no-atp*] =
*minus-equation-iff* [*of - ‹numeral v›*] **for** *v*

**lemmas** *le-minus-iff-numeral* [*no-atp*] =
*le-minus-iff* [*of ‹numeral v›*] **for** *v*

**lemmas** *minus-le-iff-numeral* [*no-atp*] =
*minus-le-iff* [*of - ‹numeral v›*] **for** *v*

**lemmas** *less-minus-iff-numeral* [*no-atp*] =
*less-minus-iff* [*of* ‹*numeral v*›] **for** *v*

**lemmas** *minus-less-iff-numeral* [*no-atp*] =
*minus-less-iff* [*of* - ‹*numeral v*›] **for** *v*

Cancellation of constant factors in comparisons ($<$ and $\leq$)

**lemmas** *mult-less-cancel-left-numeral* [*simp*, *no-atp*] = *mult-less-cancel-left* [*of* ‹*numeral v*›] **for** *v*
**lemmas** *mult-less-cancel-right-numeral* [*simp*, *no-atp*] = *mult-less-cancel-right* [*of* - ‹*numeral v*›] **for** *v*
**lemmas** *mult-le-cancel-left-numeral* [*simp*, *no-atp*] = *mult-le-cancel-left* [*of* ‹*numeral v*›] **for** *v*
**lemmas** *mult-le-cancel-right-numeral* [*simp*, *no-atp*] = *mult-le-cancel-right* [*of* - ‹*numeral v*›] **for** *v*

Multiplying out constant divisors in comparisons ($<$, $\leq$ and $=$)

**named-theorems** *divide-const-simps* ‹*simplification rules to simplify comparisons involving constant divisors*›

**lemmas** *le-divide-eq-numeral1* [*simp*,*divide-const-simps*] =
*pos-le-divide-eq* [*of* ‹*numeral w*›, *OF zero-less-numeral*]
*neg-le-divide-eq* [*of* ‹− *numeral w*›, *OF neg-numeral-less-zero*] **for** *w*

**lemmas** *divide-le-eq-numeral1* [*simp*,*divide-const-simps*] =
*pos-divide-le-eq* [*of* ‹*numeral w*›, *OF zero-less-numeral*]
*neg-divide-le-eq* [*of* ‹− *numeral w*›, *OF neg-numeral-less-zero*] **for** *w*

**lemmas** *less-divide-eq-numeral1* [*simp*,*divide-const-simps*] =
*pos-less-divide-eq* [*of* ‹*numeral w*›, *OF zero-less-numeral*]
*neg-less-divide-eq* [*of* ‹− *numeral w*›, *OF neg-numeral-less-zero*] **for** *w*

**lemmas** *divide-less-eq-numeral1* [*simp*,*divide-const-simps*] =
*pos-divide-less-eq* [*of* ‹*numeral w*›, *OF zero-less-numeral*]
*neg-divide-less-eq* [*of* ‹− *numeral w*›, *OF neg-numeral-less-zero*] **for** *w*

**lemmas** *eq-divide-eq-numeral1* [*simp*,*divide-const-simps*] =
*eq-divide-eq* [*of* - - ‹*numeral w*›]
*eq-divide-eq* [*of* - - ‹− *numeral w*›] **for** *w*

**lemmas** *divide-eq-eq-numeral1* [*simp*,*divide-const-simps*] =
*divide-eq-eq* [*of* - ‹*numeral w*›]
*divide-eq-eq* [*of* - ‹− *numeral w*›] **for** *w*

### 46.6.2 Optional Simplification Rules Involving Constants

Simplify quotients that are compared with a literal constant.

**lemmas** *le-divide-eq-numeral* [*divide-const-simps*] =

*le-divide-eq* [*of* ‹*numeral w*›]
*le-divide-eq* [*of* ‹− *numeral w*›] **for** *w*

**lemmas** *divide-le-eq-numeral* [*divide-const-simps*] =
*divide-le-eq* [*of* - - ‹*numeral w*›]
*divide-le-eq* [*of* - - ‹− *numeral w*›] **for** *w*

**lemmas** *less-divide-eq-numeral* [*divide-const-simps*] =
*less-divide-eq* [*of* ‹*numeral w*›]
*less-divide-eq* [*of* ‹− *numeral w*›] **for** *w*

**lemmas** *divide-less-eq-numeral* [*divide-const-simps*] =
*divide-less-eq* [*of* - - ‹*numeral w*›]
*divide-less-eq* [*of* - - ‹− *numeral w*›] **for** *w*

**lemmas** *eq-divide-eq-numeral* [*divide-const-simps*] =
*eq-divide-eq* [*of* ‹*numeral w*›]
*eq-divide-eq* [*of* ‹− *numeral w*›] **for** *w*

**lemmas** *divide-eq-eq-numeral* [*divide-const-simps*] =
*divide-eq-eq* [*of* - - ‹*numeral w*›]
*divide-eq-eq* [*of* - - ‹− *numeral w*›] **for** *w*

Not good as automatic simprules because they cause case splits.

**lemmas** [*divide-const-simps*] =
*le-divide-eq-1 divide-le-eq-1 less-divide-eq-1 divide-less-eq-1*

## 46.7 Setting up simprocs

**lemma** *mult-numeral-1*: ‹*Numeral1* ∗ *a* = *a*›
**for** *a* :: ‹'*a*::*semiring-numeral*›
⟨*proof*⟩

**lemma** *mult-numeral-1-right*: ‹*a* ∗ *Numeral1* = *a*›
**for** *a* :: ‹'*a*::*semiring-numeral*›
⟨*proof*⟩

**lemma** *divide-numeral-1*: ‹*a* / *Numeral1* = *a*›
**for** *a* :: ‹'*a*::*field*›
⟨*proof*⟩

**lemma** *inverse-numeral-1*: ‹*inverse Numeral1* = (*Numeral1*::'*a*::*division-ring*)›
⟨*proof*⟩

Theorem lists for the cancellation simprocs. The use of a binary numeral for 1 reduces the number of special cases.

**lemma** *mult-1s-semiring-numeral*:
‹*Numeral1* ∗ *a* = *a*›
‹*a* ∗ *Numeral1* = *a*›

**for** *a* :: ‹*'a*::*semiring-numeral*›
⟨*proof*⟩

**lemma** *mult-1s-ring-1*:
‹$-$ *Numeral1* $*$ *b* $= -$ *b*›
‹*b* $* -$ *Numeral1* $= -$ *b*›
**for** *b* :: ‹*'a*::*ring-1*›
⟨*proof*⟩

**lemmas** *mult-1s* = *mult-1s-semiring-numeral mult-1s-ring-1*

⟨*ML*⟩

### 46.7.1 Simplification of arithmetic operations on integer constants

**lemmas** *arith-special* =
*add-numeral-special add-neg-numeral-special*
*diff-numeral-special*

**lemmas** *arith-extra-simps* =
*numeral-plus-numeral add-neg-numeral-simps add-0-left add-0-right*
*minus-zero*
*diff-numeral-simps diff-0 diff-0-right*
*numeral-times-numeral mult-neg-numeral-simps*
*mult-zero-left mult-zero-right*
*abs-numeral abs-neg-numeral*

For making a minimal simpset, one must include these default simprules. Also include *simp-thms*.

**lemmas** *arith-simps* =
*add-num-simps mult-num-simps sub-num-simps*
*BitM.simps dbl-simps dbl-inc-simps dbl-dec-simps*
*abs-zero abs-one arith-extra-simps*

**lemmas** *more-arith-simps* =
*neg-le-iff-le*
*minus-zero left-minus right-minus*
*mult-1-left mult-1-right*
*mult-minus-left mult-minus-right*
*minus-add-distrib minus-minus mult.assoc*

**lemmas** *of-nat-simps* =
*of-nat-0 of-nat-1 of-nat-Suc of-nat-add of-nat-mult*

Simplification of relational operations.

**lemmas** *eq-numeral-extra* =
*zero-neq-one one-neq-zero*

**lemmas** *rel-simps* =
*le-num-simps less-num-simps eq-num-simps*
*le-numeral-simps le-neg-numeral-simps le-minus-one-simps le-numeral-extra*
*less-numeral-simps less-neg-numeral-simps less-minus-one-simps less-numeral-extra*
*eq-numeral-simps eq-neg-numeral-simps eq-numeral-extra*

**lemma** *Let-numeral* [*simp*]: ‹$Let\ (numeral\ v)\ f = f\ (numeral\ v)$›
— Unfold all *let*s involving constants
⟨*proof*⟩

**lemma** *Let-neg-numeral* [*simp*]: ‹$Let\ (-\ numeral\ v)\ f = f\ (-\ numeral\ v)$›
— Unfold all *let*s involving constants
⟨*proof*⟩

⟨*ML*⟩

### 46.7.2 Simplification of arithmetic when nested to the right

**lemma** *add-numeral-left* [*simp*]: ‹$numeral\ v + (numeral\ w + z) = (numeral(v + w) + z)$›
⟨*proof*⟩

**lemma** *add-neg-numeral-left* [*simp*]:
‹$numeral\ v + (-\ numeral\ w + y) = (sub\ v\ w + y)$›
‹$-\ numeral\ v + (numeral\ w + y) = (sub\ w\ v + y)$›
‹$-\ numeral\ v + (-\ numeral\ w + y) = (-\ numeral(v + w) + y)$›
⟨*proof*⟩

**lemma** *mult-numeral-left-semiring-numeral*:
‹$numeral\ v * (numeral\ w * z) = (numeral(v * w) * z :: 'a::semiring\text{-}numeral)$›
⟨*proof*⟩

**lemma** *mult-numeral-left-ring-1*:
‹$-\ numeral\ v * (numeral\ w * y) = (-\ numeral(v * w) * y :: 'a::ring\text{-}1)$›
‹$numeral\ v * (-\ numeral\ w * y) = (-\ numeral(v * w) * y :: 'a::ring\text{-}1)$›
‹$-\ numeral\ v * (-\ numeral\ w * y) = (numeral(v * w) * y :: 'a::ring\text{-}1)$›
⟨*proof*⟩

**lemmas** *mult-numeral-left* [*simp*] =
*mult-numeral-left-semiring-numeral*
*mult-numeral-left-ring-1*

## 46.8 Code module namespace

**code-identifier**
**code-module** *Num* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

## 46.9 Printing of evaluated natural numbers as numerals

**lemma** [*code-post*]:

‹*Suc 0 = 1*›
‹*Suc 1 = 2*›
‹*Suc* (*numeral n*) = *numeral* (*inc n*)›
⟨*proof*⟩

**lemmas** [*code-post*] = *inc.simps*

### 46.10 More on auxiliary conversion

**context** *semiring-1*
**begin**

**lemma** *num-of-nat-numeral-eq* [*simp*]:
‹*num-of-nat* (*numeral q*) = *q*›
⟨*proof*⟩

**lemma** *numeral-num-of-nat-unfold*:
‹*numeral* (*num-of-nat n*) = (*if n = 0 then 1 else of-nat n*)›
⟨*proof*⟩

**end**

**hide-const** (**open**) *One Bit0 Bit1 BitM inc pow sqr sub dbl dbl-inc dbl-dec*

**end**

# 47 Exponentiation

**theory** *Power*
**imports** *Num*
**begin**

### 47.1 Powers for Arbitrary Monoids

**class** *power* = *one* + *times*
**begin**

**primrec** *power* :: *'a* ⇒ *nat* ⇒ *'a* (**infixr** ‹^› *80*)
**where**
*power-0*: *a* ^ *0* = *1*
| *power-Suc*: *a* ^ *Suc n* = *a* ∗ *a* ^ *n*

**notation** (*latex* **output**)
*power* (‹(-$^{-}$)› [*1000*] *1000*)

Special syntax for squares.

**abbreviation** *power2* :: *'a* ⇒ *'a* (‹(‹*notation*=‹*postfix 2*››-$^{2}$)› [*1000*] *999*)
**where** $x^2$ ≡ *x* ^ *2*

**end**

**context**
  **includes** *lifting-syntax*
**begin**

**lemma** *power-transfer* [*transfer-rule*]:
  ‹($R$ ===> (=) ===> $R$) (^) (^)›
    **if** [*transfer-rule*]: ‹$R$ *1 1*›
      ‹($R$ ===> $R$ ===> $R$) (∗) (∗)›
    **for** $R$ :: ‹′*a*::*power* ⇒ ′*b*::*power* ⇒ *bool*›
  ⟨*proof*⟩

**end**

**context** *monoid-mult*
**begin**

**subclass** *power* ⟨*proof*⟩

**lemma** *power-one* [*simp*]: *1* ^ *n* = *1*
  ⟨*proof*⟩

**lemma** *power-one-right* [*simp*]: *a* ^ *1* = *a*
  ⟨*proof*⟩

**lemma** *power-Suc0-right* [*simp*]: *a* ^ *Suc 0* = *a*
  ⟨*proof*⟩

**lemma** *power-commutes*: *a* ^ *n* ∗ *a* = *a* ∗ *a* ^ *n*
  ⟨*proof*⟩

**lemma** *power-Suc2*: *a* ^ *Suc n* = *a* ^ *n* ∗ *a*
  ⟨*proof*⟩

**lemma** *power-add*: *a* ^ (*m* + *n*) = *a* ^ *m* ∗ *a* ^ *n*
  ⟨*proof*⟩

**lemma** *power-mult*: *a* ^ (*m* ∗ *n*) = (*a* ^ *m*) ^ *n*
  ⟨*proof*⟩

**lemma** *power-even-eq*: *a* ^ (*2* ∗ *n*) = $(a$ ^ $n)^2$
  ⟨*proof*⟩

**lemma** *power-odd-eq*: *a* ^ *Suc* (*2*∗*n*) = *a* ∗ $(a$ ^ $n)^2$
  ⟨*proof*⟩

**lemma** *power-numeral-even*: *z* ^ *numeral* (*Num.Bit0 w*) = (*let w* = *z* ^ (*numeral*

*w*) *in* $w * w$)
⟨*proof*⟩

**lemma** *power-numeral-odd*: $z$ ^ *numeral* (*Num.Bit1* $w$) = (*let* $w = z$ ^ (*numeral* $w$) *in* $z * w * w$)
⟨*proof*⟩

**lemma** *power2-eq-square*: $a^2 = a * a$
⟨*proof*⟩

**lemma** *power3-eq-cube*: $a$ ^ $3 = a * a * a$
⟨*proof*⟩

**lemma** *power4-eq-xxxx*: $x$^$4 = x * x * x * x$
⟨*proof*⟩

**lemma** *power-numeral-reduce*: $x$ ^ *numeral* $n = x * x$ ^ *pred-numeral* $n$
⟨*proof*⟩

**lemma** *funpow-times-power*: (*times* $x$ ^^ $f\ x$) = *times* ($x$ ^ $f\ x$)
⟨*proof*⟩

**lemma** *power-commuting-commutes*:
**assumes** $x * y = y * x$
**shows** $x$ ^ $n * y = y * x$ ^ $n$
⟨*proof*⟩

**lemma** *power-minus-mult*: $0 < n \Longrightarrow a$ ^ $(n - 1) * a = a$ ^ $n$
⟨*proof*⟩

**lemma** *left-right-inverse-power*:
**assumes** $x * y = 1$
**shows** $x$ ^ $n * y$ ^ $n = 1$
⟨*proof*⟩

**end**

**context** *comm-monoid-mult*
**begin**

**lemma** *power-mult-distrib* [*algebra-simps*, *algebra-split-simps*, *field-simps*, *field-split-simps*, *divide-simps*]:
$(a * b)$ ^ $n = (a$ ^ $n) * (b$ ^ $n)$
⟨*proof*⟩

**end**

Extract constant factors from powers.

**declare** *power-mult-distrib* [**where** $a$ = *numeral* $w$ **for** $w$, *simp*]

**declare** *power-mult-distrib* [**where** $b$ = *numeral* $w$ **for** $w$, *simp*]

**lemma** *power-add-numeral* [*simp*]: $a$^*numeral* $m$ ∗ $a$^*numeral* $n$ = $a$^*numeral* ($m$ + $n$)
**for** $a$ :: ′$a$::*monoid-mult*
⟨*proof*⟩

**lemma** *power-add-numeral2* [*simp*]: $a$^*numeral* $m$ ∗ ($a$^*numeral* $n$ ∗ $b$) = $a$^*numeral* ($m$ + $n$) ∗ $b$
**for** $a$ :: ′$a$::*monoid-mult*
⟨*proof*⟩

**lemma** *power-mult-numeral* [*simp*]: ($a$^*numeral* $m$)^*numeral* $n$ = $a$^*numeral* ($m$ ∗ $n$)
**for** $a$ :: ′$a$::*monoid-mult*
⟨*proof*⟩

**context** *semiring-numeral*
**begin**

**lemma** *numeral-sqr*: *numeral* (*Num.sqr* $k$) = *numeral* $k$ ∗ *numeral* $k$
⟨*proof*⟩

**lemma** *numeral-pow*: *numeral* (*Num.pow* $k$ $l$) = *numeral* $k$ ^ *numeral* $l$
⟨*proof*⟩

**lemma** *power-numeral* [*simp*]: *numeral* $k$ ^ *numeral* $l$ = *numeral* (*Num.pow* $k$ $l$)
⟨*proof*⟩

**end**

**context** *semiring-1*
**begin**

**lemma** *of-nat-power* [*simp*]: *of-nat* ($m$ ^ $n$) = *of-nat* $m$ ^ $n$
⟨*proof*⟩

**lemma** *zero-power*: $0 < n \Longrightarrow 0$ ^ $n$ = $0$
⟨*proof*⟩

**lemma** *power-zero-numeral* [*simp*]: $0$ ^ *numeral* $k$ = $0$
⟨*proof*⟩

**lemma** *zero-power2*: $0^2 = 0$
⟨*proof*⟩

**lemma** *one-power2*: $1^2 = 1$
⟨*proof*⟩

**lemma** *power-0-Suc* [*simp*]: *0* ^ *Suc n* = *0*
⟨*proof*⟩

It looks plausible as a simprule, but its effect can be strange.

**lemma** *power-0-left*: *0* ^ *n* = (*if n* = *0 then 1 else 0*)
⟨*proof*⟩

**end**

**context** *semiring-char-0* **begin**

**lemma** *numeral-power-eq-of-nat-cancel-iff* [*simp*]:
*numeral x* ^ *n* = *of-nat y* ⟷ *numeral x* ^ *n* = *y*
⟨*proof*⟩

**lemma** *real-of-nat-eq-numeral-power-cancel-iff* [*simp*]:
*of-nat y* = *numeral x* ^ *n* ⟷ *y* = *numeral x* ^ *n*
⟨*proof*⟩

**lemma** *of-nat-eq-of-nat-power-cancel-iff*[*simp*]: (*of-nat b*) ^ *w* = *of-nat x* ⟷ *b* ^ *w* = *x*
⟨*proof*⟩

**lemma** *of-nat-power-eq-of-nat-cancel-iff*[*simp*]: *of-nat x* = (*of-nat b*) ^ *w* ⟷ *x* = *b* ^ *w*
⟨*proof*⟩

**end**

**context** *comm-semiring-1*
**begin**

The divides relation.

**lemma** *le-imp-power-dvd*:
**assumes** *m* ≤ *n*
**shows** *a* ^ *m dvd a* ^ *n*
⟨*proof*⟩

**lemma** *power-le-dvd*: *a* ^ *n dvd b* ⟹ *m* ≤ *n* ⟹ *a* ^ *m dvd b*
⟨*proof*⟩

**lemma** *dvd-power-same*: *x dvd y* ⟹ *x* ^ *n dvd y* ^ *n*
⟨*proof*⟩

**lemma** *dvd-power-le*: *x dvd y* ⟹ *m* ≥ *n* ⟹ *x* ^ *n dvd y* ^ *m*
⟨*proof*⟩

**lemma** *dvd-power* [*simp*]:
**fixes** *n* :: *nat*

**assumes** $n > 0 \vee x = 1$
**shows** $x\ dvd\ (x \,\hat{}\, n)$
⟨*proof*⟩

**end**

**context** *semiring-1-no-zero-divisors*
**begin**

**subclass** *power* ⟨*proof*⟩

**lemma** *power-eq-0-iff* [*simp*]: $a \,\hat{}\, n = 0 \longleftrightarrow a = 0 \wedge n > 0$
⟨*proof*⟩

**lemma** *power-not-zero*: $a \neq 0 \Longrightarrow a \,\hat{}\, n \neq 0$
⟨*proof*⟩

**lemma** *zero-eq-power2* [*simp*]: $a^2 = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**end**

**context** *ring-1*
**begin**

**lemma** *power-minus*: $(-\ a) \,\hat{}\, n = (-\ 1) \,\hat{}\, n * a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-minus′*: $NO\text{-}MATCH\ 1\ x \Longrightarrow (-x) \,\hat{}\, n = (-1)\hat{}n * x \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-minus-Bit0*: $(-\ x) \,\hat{}\, numeral\ (Num.Bit0\ k) = x \,\hat{}\, numeral\ (Num.Bit0\ k)$
⟨*proof*⟩

**lemma** *power-minus-Bit1*: $(-\ x) \,\hat{}\, numeral\ (Num.Bit1\ k) = -\ (x \,\hat{}\, numeral\ (Num.Bit1\ k))$
⟨*proof*⟩

**lemma** *power2-minus* [*simp*]: $(-\ a)^2 = a^2$
⟨*proof*⟩

**lemma** *power-minus1-even* [*simp*]: $(-\ 1) \,\hat{}\, (2*n) = 1$
⟨*proof*⟩

**lemma** *power-minus1-odd*: $(-\ 1) \,\hat{}\, Suc\ (2*n) = -1$
⟨*proof*⟩

**lemma** *power-minus-even* [*simp*]: $(-a) \,\hat{}\, (2*n) = a \,\hat{}\, (2*n)$

⟨*proof*⟩

**end**

**context** *ring-1-no-zero-divisors*
**begin**

**lemma** *power2-eq-1-iff*: $a^2 = 1 \longleftrightarrow a = 1 \vee a = -1$
⟨*proof*⟩

**end**

**context** *idom*
**begin**

**lemma** *power2-eq-iff*: $x^2 = y^2 \longleftrightarrow x = y \vee x = -y$
⟨*proof*⟩

**end**

**context** *semidom-divide*
**begin**

**lemma** *power-diff*:
$a \mathbin{\hat{}} (m - n) = (a \mathbin{\hat{}} m)\ div\ (a \mathbin{\hat{}} n)$ **if** $a \neq 0$ **and** $n \leq m$
⟨*proof*⟩

**lemma** *power-diff-if*:
$a \mathbin{\hat{}} (m - n) = (if\ n \leq m\ then\ (a \mathbin{\hat{}} m)\ div\ (a \mathbin{\hat{}} n)\ else\ 1)$ **if** $a \neq 0$
⟨*proof*⟩

**end**

**context** *algebraic-semidom*
**begin**

**lemma** *div-power*: $b\ dvd\ a \Longrightarrow (a\ div\ b) \mathbin{\hat{}} n = a \mathbin{\hat{}} n\ div\ b \mathbin{\hat{}} n$
⟨*proof*⟩

**lemma** *is-unit-power-iff*: $is\text{-}unit\ (a \mathbin{\hat{}} n) \longleftrightarrow is\text{-}unit\ a \vee n = 0$
⟨*proof*⟩

**lemma** *dvd-power-iff*:
**assumes** $x \neq 0$
**shows** $x \mathbin{\hat{}} m\ dvd\ x \mathbin{\hat{}} n \longleftrightarrow is\text{-}unit\ x \vee m \leq n$
⟨*proof*⟩

**end**

**context** *normalization-semidom-multiplicative*
**begin**

**lemma** *normalize-power*: *normalize* $(a \,\hat{}\, n) = $ *normalize* $a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *unit-factor-power*: *unit-factor* $(a \,\hat{}\, n) = $ *unit-factor* $a \,\hat{}\, n$
⟨*proof*⟩

**end**

**context** *division-ring*
**begin**

Perhaps these should be simprules.

**lemma** *power-inverse* [*field-simps*, *field-split-simps*, *divide-simps*]: *inverse* $a \,\hat{}\, n =$ *inverse* $(a \,\hat{}\, n)$
⟨*proof*⟩

**lemma** *power-one-over* [*field-simps*, *field-split-simps*, *divide-simps*]: $(1 \,/\, a) \,\hat{}\, n = 1 \,/\, a \,\hat{}\, n$
⟨*proof*⟩

**end**

**context** *field*
**begin**

**lemma** *power-divide* [*field-simps*, *field-split-simps*, *divide-simps*]: $(a \,/\, b) \,\hat{}\, n = a \,\hat{}\, n \,/\, b \,\hat{}\, n$
⟨*proof*⟩

**end**

## 47.2 Exponentiation on ordered types

**context** *ordered-semiring-1*
**begin**

**lemma** *zero-le-power* [*simp*]: $0 \le a \Longrightarrow 0 \le a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-mono*: $a \le b \Longrightarrow 0 \le a \Longrightarrow a \,\hat{}\, n \le b \,\hat{}\, n$
⟨*proof*⟩

**lemma** *one-le-power* [*simp*]: $1 \le a \Longrightarrow 1 \le a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-le-one*: $0 \leq a \Longrightarrow a \leq 1 \Longrightarrow a \widehat{\ } n \leq 1$
⟨*proof*⟩

**lemma** *power-gt1-lemma*:
**assumes** *gt1*: $1 < a$
**shows** $1 < a * a \widehat{\ } n$
⟨*proof*⟩

**lemma** *power-gt1*: $1 < a \Longrightarrow 1 < a \widehat{\ } Suc\ n$
⟨*proof*⟩

**lemma** *one-less-power* [*simp*]: $1 < a \Longrightarrow 0 < n \Longrightarrow 1 < a \widehat{\ } n$
⟨*proof*⟩

Proof resembles that of *power-strict-decreasing*.

**lemma** *power-increasing*: $n \leq N \Longrightarrow 1 \leq a \Longrightarrow a \widehat{\ } n \leq a \widehat{\ } N$
⟨*proof*⟩

Proof resembles that of *power-strict-decreasing*.

**lemma** *power-decreasing*: $n \leq N \Longrightarrow 0 \leq a \Longrightarrow a \leq 1 \Longrightarrow a \widehat{\ } N \leq a \widehat{\ } n$
⟨*proof*⟩

**lemma** *power-Suc-le-self*: $0 \leq a \Longrightarrow a \leq 1 \Longrightarrow a \widehat{\ } Suc\ n \leq a$
⟨*proof*⟩

**end**

**context** *linordered-semidom*
**begin**

**lemma** *zero-less-power* [*simp*]: $0 < a \Longrightarrow 0 < a \widehat{\ } n$
⟨*proof*⟩

**lemma** *power-le-imp-le-exp*:
**assumes** *gt1*: $1 < a$
**shows** $a \widehat{\ } m \leq a \widehat{\ } n \Longrightarrow m \leq n$
⟨*proof*⟩

**lemma** *of-nat-zero-less-power-iff* [*simp*]: $of\text{-}nat\ x \widehat{\ } n > 0 \longleftrightarrow x > 0 \vee n = 0$
⟨*proof*⟩

**lemma** *power-strict-mono*: $a < b \Longrightarrow 0 \leq a \Longrightarrow 0 < n \Longrightarrow a \widehat{\ } n < b \widehat{\ } n$
⟨*proof*⟩

**lemma** *power-mono-iff* [*simp*]:
**shows** $[\![a \geq 0;\ b \geq 0;\ n>0]\!] \Longrightarrow a \widehat{\ } n \leq b \widehat{\ } n \longleftrightarrow a \leq b$
⟨*proof*⟩

Lemma for *power-strict-decreasing*

**lemma** *power-Suc-less*: $0 < a \Longrightarrow a < 1 \Longrightarrow a * a \,\hat{}\, n < a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-strict-decreasing*: $n < N \Longrightarrow 0 < a \Longrightarrow a < 1 \Longrightarrow a \,\hat{}\, N < a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-decreasing-iff* [*simp*]: $[\![0 < b;\ b < 1]\!] \Longrightarrow b \,\hat{}\, m \le b \,\hat{}\, n \longleftrightarrow n \le m$
⟨*proof*⟩

**lemma** *power-strict-decreasing-iff* [*simp*]: $[\![0 < b;\ b < 1]\!] \Longrightarrow b \,\hat{}\, m < b \,\hat{}\, n \longleftrightarrow n < m$
⟨*proof*⟩

**lemma** *power-Suc-less-one*: $0 < a \Longrightarrow a < 1 \Longrightarrow a \,\hat{}\, Suc\ n < 1$
⟨*proof*⟩

Lemma for *power-strict-increasing*.

**lemma** *power-less-power-Suc*: $1 < a \Longrightarrow a \,\hat{}\, n < a * a \,\hat{}\, n$
⟨*proof*⟩

**lemma** *power-strict-increasing*: $n < N \Longrightarrow 1 < a \Longrightarrow a \,\hat{}\, n < a \,\hat{}\, N$
⟨*proof*⟩

Surely we can strengthen this? It holds for $0<a<1$ too.

**lemma** *power-inject-exp* [*simp*]:
‹$a \,\hat{}\, m = a \,\hat{}\, n \longleftrightarrow m = n$› **if** ‹$1 < a$›
⟨*proof*⟩

**lemma** *power-inject-exp′*:
‹$a \,\hat{}\, m = a \,\hat{}\, n \longleftrightarrow m = n$› **if** ‹$a\neq 1$› ‹$a>0$›
⟨*proof*⟩

Can relax the first premise to $0 < a$ in the case of the natural numbers.

**lemma** *power-less-imp-less-exp*: $1 < a \Longrightarrow a \,\hat{}\, m < a \,\hat{}\, n \Longrightarrow m < n$
⟨*proof*⟩

**lemma** *power-increasing-iff* [*simp*]: $1 < b \Longrightarrow b \,\hat{}\, x \le b \,\hat{}\, y \longleftrightarrow x \le y$
⟨*proof*⟩

**lemma** *power-strict-increasing-iff* [*simp*]: $1 < b \Longrightarrow b \,\hat{}\, x < b \,\hat{}\, y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *power-le-imp-le-base*:
**assumes** *le*: $a \,\hat{}\, Suc\ n \le b \,\hat{}\, Suc\ n$
**and** $0 \le b$
**shows** $a \le b$
⟨*proof*⟩

**lemma** *power-less-imp-less-base*:
**assumes** *less*: $a \hat{}\ n < b \hat{}\ n$
**assumes** *nonneg*: $0 \leq b$
**shows** $a < b$
⟨*proof*⟩

**lemma** *power-inject-base*: $a \hat{}\ Suc\ n = b \hat{}\ Suc\ n \Longrightarrow 0 \leq a \Longrightarrow 0 \leq b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *power-eq-imp-eq-base*: $a \hat{}\ n = b \hat{}\ n \Longrightarrow 0 \leq a \Longrightarrow 0 \leq b \Longrightarrow 0 < n \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *power-eq-iff-eq-base*: $0 < n \Longrightarrow 0 \leq a \Longrightarrow 0 \leq b \Longrightarrow a \hat{}\ n = b \hat{}\ n \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *power2-le-imp-le*: $x^2 \leq y^2 \Longrightarrow 0 \leq y \Longrightarrow x \leq y$
⟨*proof*⟩

**lemma** *power2-less-imp-less*: $x^2 < y^2 \Longrightarrow 0 \leq y \Longrightarrow x < y$
⟨*proof*⟩

**lemma** *power2-eq-imp-eq*: $x^2 = y^2 \Longrightarrow 0 \leq x \Longrightarrow 0 \leq y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *power2-eq-iff-nonneg* [*simp*]:
**assumes** $0 \leq x\ 0 \leq y$
**shows** $(x \hat{}\ 2 = y \hat{}\ 2) \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *of-nat-less-numeral-power-cancel-iff* [*simp*]:
$of\text{-}nat\ x < numeral\ i \hat{}\ n \longleftrightarrow x < numeral\ i \hat{}\ n$
⟨*proof*⟩

**lemma** *of-nat-le-numeral-power-cancel-iff* [*simp*]:
$of\text{-}nat\ x \leq numeral\ i \hat{}\ n \longleftrightarrow x \leq numeral\ i \hat{}\ n$
⟨*proof*⟩

**lemma** *numeral-power-less-of-nat-cancel-iff* [*simp*]:
$numeral\ i \hat{}\ n < of\text{-}nat\ x \longleftrightarrow numeral\ i \hat{}\ n < x$
⟨*proof*⟩

**lemma** *numeral-power-le-of-nat-cancel-iff* [*simp*]:
$numeral\ i \hat{}\ n \leq of\text{-}nat\ x \longleftrightarrow numeral\ i \hat{}\ n \leq x$
⟨*proof*⟩

**lemma** *of-nat-le-of-nat-power-cancel-iff* [*simp*]: $(of\text{-}nat\ b) \hat{}\ w \leq of\text{-}nat\ x \longleftrightarrow b \hat{}\ w \leq x$

⟨*proof*⟩

**lemma** *of-nat-power-le-of-nat-cancel-iff*[*simp*]: $of\text{-}nat\ x \leq (of\text{-}nat\ b)\ \hat{}\ w \longleftrightarrow x \leq b\ \hat{}\ w$
⟨*proof*⟩

**lemma** *of-nat-less-of-nat-power-cancel-iff*[*simp*]: $(of\text{-}nat\ b)\ \hat{}\ w < of\text{-}nat\ x \longleftrightarrow b\ \hat{}\ w < x$
⟨*proof*⟩

**lemma** *of-nat-power-less-of-nat-cancel-iff*[*simp*]: $of\text{-}nat\ x < (of\text{-}nat\ b)\ \hat{}\ w \longleftrightarrow x < b\ \hat{}\ w$
⟨*proof*⟩

**lemma** *power2-nonneg-ge-1-iff*:
**assumes** $x \geq 0$
**shows** $x\ \hat{}\ 2 \geq 1 \longleftrightarrow x \geq 1$
⟨*proof*⟩

**lemma** *power2-nonneg-gt-1-iff*:
**assumes** $x \geq 0$
**shows** $x\ \hat{}\ 2 > 1 \longleftrightarrow x > 1$
⟨*proof*⟩

**end**

Some *nat*-specific lemmas:

**lemma** *mono-ge2-power-minus-self*:
**assumes** $k \geq 2$ **shows** $mono\ (\lambda m.\ k\ \hat{}\ m - m)$
⟨*proof*⟩

**lemma** *self-le-ge2-pow*[*simp*]:
**assumes** $k \geq 2$ **shows** $m \leq k\ \hat{}\ m$
⟨*proof*⟩

**lemma** *diff-le-diff-pow*[*simp*]:
**assumes** $k \geq 2$ **shows** $m - n \leq k\ \hat{}\ m - k\ \hat{}\ n$
⟨*proof*⟩

**context** *linordered-ring-strict*
**begin**

**lemma** *sum-squares-eq-zero-iff*: $x * x + y * y = 0 \longleftrightarrow x = 0 \land y = 0$
⟨*proof*⟩

**lemma** *sum-squares-le-zero-iff*: $x * x + y * y \leq 0 \longleftrightarrow x = 0 \land y = 0$
⟨*proof*⟩

**lemma** *sum-squares-gt-zero-iff*: $0 < x * x + y * y \longleftrightarrow x \neq 0 \vee y \neq 0$
⟨*proof*⟩

**end**

**context** *linordered-idom*
**begin**

**lemma** *zero-le-power2* [*simp*]: $0 \leq a^2$
⟨*proof*⟩

**lemma** *zero-less-power2* [*simp*]: $0 < a^2 \longleftrightarrow a \neq 0$
⟨*proof*⟩

**lemma** *power2-less-0* [*simp*]: $\neg\ a^2 < 0$
⟨*proof*⟩

**lemma** *power-abs*: $|a \mathbin{\hat{}} n| = |a| \mathbin{\hat{}} n$ — FIXME simp?
⟨*proof*⟩

**lemma** *power-sgn* [*simp*]: $sgn\ (a \mathbin{\hat{}} n) = sgn\ a \mathbin{\hat{}} n$
⟨*proof*⟩

**lemma** *abs-power-minus* [*simp*]: $|(-\ a) \mathbin{\hat{}} n| = |a \mathbin{\hat{}} n|$
⟨*proof*⟩

**lemma** *zero-less-power-abs-iff* [*simp*]: $0 < |a| \mathbin{\hat{}} n \longleftrightarrow a \neq 0 \vee n = 0$
⟨*proof*⟩

**lemma** *zero-le-power-abs* [*simp*]: $0 \leq |a| \mathbin{\hat{}} n$
⟨*proof*⟩

**lemma** *power2-less-eq-zero-iff* [*simp*]: $a^2 \leq 0 \longleftrightarrow a = 0$
⟨*proof*⟩

**lemma** *abs-power2* [*simp*]: $|a^2| = a^2$
⟨*proof*⟩

**lemma** *power2-abs* [*simp*]: $|a|^2 = a^2$
⟨*proof*⟩

**lemma** *odd-power-less-zero*: $a < 0 \Longrightarrow a \mathbin{\hat{}} Suc\ (2 * n) < 0$
⟨*proof*⟩

**lemma** *odd-0-le-power-imp-0-le*: $0 \leq a \mathbin{\hat{}} Suc\ (2 * n) \Longrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *zero-le-even-power'*[*simp*]: $0 \leq a \mathbin{\hat{}} (2 * n)$
⟨*proof*⟩

**lemma** *sum-power2-ge-zero*: $0 \leq x^2 + y^2$
⟨*proof*⟩

**lemma** *not-sum-power2-lt-zero*: $\neg\ x^2 + y^2 < 0$
⟨*proof*⟩

**lemma** *sum-power2-eq-zero-iff*: $x^2 + y^2 = 0 \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *sum-power2-le-zero-iff*: $x^2 + y^2 \leq 0 \longleftrightarrow x = 0 \wedge y = 0$
⟨*proof*⟩

**lemma** *sum-power2-gt-zero-iff*: $0 < x^2 + y^2 \longleftrightarrow x \neq 0 \vee y \neq 0$
⟨*proof*⟩

**lemma** *abs-le-square-iff*: $|x| \leq |y| \longleftrightarrow x^2 \leq y^2$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *power2-le-iff-abs-le*:
$y \geq 0 \Longrightarrow x^2 \leq y^2 \longleftrightarrow |x| \leq y$
⟨*proof*⟩

**lemma** *abs-square-le-1*:$x^2 \leq 1 \longleftrightarrow |x| \leq 1$
⟨*proof*⟩

**lemma** *abs-square-eq-1*: $x^2 = 1 \longleftrightarrow |x| = 1$
⟨*proof*⟩

**lemma** *abs-square-less-1*: $x^2 < 1 \longleftrightarrow |x| < 1$
⟨*proof*⟩

**lemma** *square-le-1*:
**assumes** $-\ 1 \leq x\ x \leq 1$
**shows** $x^2 \leq 1$
⟨*proof*⟩

**lemma** *power2-mono*: $|x| \leq |y| \Longrightarrow x\ \hat{}\ 2 \leq y\ \hat{}\ 2$
⟨*proof*⟩

**lemma** *power2-strict-mono*:
**assumes** $|x| < |y|$
**shows** $x\ \hat{}\ 2 < y\ \hat{}\ 2$
⟨*proof*⟩

**end**

## 47.3 Miscellaneous rules

**context** *linordered-semidom*
**begin**

**lemma** *self-le-power*: $1 \leq a \Longrightarrow 0 < n \Longrightarrow a \leq a \hat{\ } n$
⟨*proof*⟩

**lemma** *power-le-one-iff*: $0 \leq a \Longrightarrow a \hat{\ } n \leq 1 \longleftrightarrow (n = 0 \vee a \leq 1)$
⟨*proof*⟩

**lemma** *power-less1-D*: $a\hat{\ }n < 1 \Longrightarrow a < 1$
⟨*proof*⟩

**lemma** *power-less-one-iff*: $0 \leq a \Longrightarrow a \hat{\ } n < 1 \longleftrightarrow (n > 0 \wedge a < 1)$
⟨*proof*⟩

**end**

**lemma** *power2-ge-1-iff*: $x \hat{\ } 2 \geq 1 \longleftrightarrow x \geq 1 \vee x \leq (-1 :: 'a :: linordered\text{-}idom)$
⟨*proof*⟩

**lemma** *power2-less-1-iff*: $x^2 < 1 \longleftrightarrow (-1 :: 'a :: linordered\text{-}idom) < x \wedge x < 1$
⟨*proof*⟩

**lemma** *power2-gt-1-iff*: $x^2 > 1 \longleftrightarrow x < (-1 :: 'a :: linordered\text{-}idom) \vee x > 1$
⟨*proof*⟩

**lemma** (**in** *power*) *power-eq-if*: $p \hat{\ } m = (if\ m{=}0\ then\ 1\ else\ p * (p \hat{\ } (m - 1)))$
⟨*proof*⟩

**lemma** (**in** *comm-semiring-1*) *power2-sum*: $(x + y)^2 = x^2 + y^2 + 2 * x * y$
⟨*proof*⟩

**context** *comm-ring-1*
**begin**

**lemma** *power2-diff*: $(x - y)^2 = x^2 + y^2 - 2 * x * y$
⟨*proof*⟩

**lemma** *power2-commute*: $(x - y)^2 = (y - x)^2$
⟨*proof*⟩

**lemma** *minus-power-mult-self*: $(- a) \hat{\ } n * (- a) \hat{\ } n = a \hat{\ } (2 * n)$
⟨*proof*⟩

**lemma** *minus-one-mult-self* [*simp*]: $(- 1) \hat{\ } n * (- 1) \hat{\ } n = 1$
⟨*proof*⟩

**lemma** *left-minus-one-mult-self* [*simp*]: $(- 1) \hat{\ } n * ((- 1) \hat{\ } n * a) = a$

$\langle proof \rangle$

**end**

Simprules for comparisons where common factors can be cancelled.

**lemmas** *zero-compare-simps* =
*add-strict-increasing add-strict-increasing2 add-increasing*
*zero-le-mult-iff zero-le-divide-iff*
*zero-less-mult-iff zero-less-divide-iff*
*mult-le-0-iff divide-le-0-iff*
*mult-less-0-iff divide-less-0-iff*
*zero-le-power2 power2-less-0*

## 47.4 Exponentiation for the Natural Numbers

**lemma** *nat-one-le-power* [*simp*]: $Suc\ 0 \leq i \Longrightarrow Suc\ 0 \leq i \mathbin{\hat{}} n$
$\langle proof \rangle$

**lemma** *nat-zero-less-power-iff* [*simp*]: $x \mathbin{\hat{}} n > 0 \longleftrightarrow x > 0 \vee n = 0$
**for** $x :: nat$
$\langle proof \rangle$

**lemma** *nat-power-eq-Suc-0-iff* [*simp*]: $x \mathbin{\hat{}} m = Suc\ 0 \longleftrightarrow m = 0 \vee x = Suc\ 0$
$\langle proof \rangle$

**lemma** *power-Suc-0* [*simp*]: $Suc\ 0 \mathbin{\hat{}} n = Suc\ 0$
$\langle proof \rangle$

Valid for the naturals, but what if $0 < i < 1$? Premises cannot be weakened: consider the case where $i = 0$, $m = 1$ and $n = 0$.

**lemma** *nat-power-less-imp-less*:
**fixes** $i :: nat$
**assumes** *nonneg*: $0 < i$
**assumes** *less*: $i \mathbin{\hat{}} m < i \mathbin{\hat{}} n$
**shows** $m < n$
$\langle proof \rangle$

**lemma** *power-gt-expt*: $n > Suc\ 0 \Longrightarrow n\hat{}k > k$
$\langle proof \rangle$

**lemma** *less-exp* [*simp*]:
‹$n < 2 \mathbin{\hat{}} n$›
$\langle proof \rangle$

**lemma** *power-dvd-imp-le*:
**fixes** $i :: nat$
**assumes** $i \mathbin{\hat{}} m\ dvd\ i \mathbin{\hat{}} n\ 1 < i$
**shows** $m \leq n$
$\langle proof \rangle$

**lemma** *dvd-power-iff-le*:
**fixes** $k$::*nat*
**shows** $2 \leq k \Longrightarrow ((k \mathbin{\char`\^} m)\ dvd\ (k \mathbin{\char`\^} n) \longleftrightarrow m \leq n)$
⟨*proof*⟩

**lemma** *power2-nat-le-eq-le*: $m^2 \leq n^2 \longleftrightarrow m \leq n$
**for** $m\ n$ :: *nat*
⟨*proof*⟩

**lemma** *power2-nat-le-imp-le*:
**fixes** $m\ n$ :: *nat*
**assumes** $m^2 \leq n$
**shows** $m \leq n$
⟨*proof*⟩

**lemma** *ex-power-ivl1*: **fixes** $b\ k$ :: *nat* **assumes** $b \geq 2$
**shows** $k \geq 1 \Longrightarrow \exists n.\ b\char`\^n \leq k \wedge k < b\char`\^(n+1)$ (**is** - $\Longrightarrow \exists n.\ ?P\ k\ n$)
⟨*proof*⟩

**lemma** *ex-power-ivl2*: **fixes** $b\ k$ :: *nat* **assumes** $b \geq 2\ k \geq 2$
**shows** $\exists n.\ b\char`\^n < k \wedge k \leq b\char`\^(n+1)$
⟨*proof*⟩

### 47.4.1 Cardinality of the Powerset

**lemma** *card-UNIV-bool* [*simp*]: *card* (*UNIV* :: *bool set*) = *2*
⟨*proof*⟩

**lemma** *card-Pow*: *finite A* $\Longrightarrow$ *card* (*Pow A*) = *2* ^ *card A*
⟨*proof*⟩

## 47.5 Code generator tweak

**code-identifier**
**code-module** *Power* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 48 Big sum and product over finite (non-empty) sets

**theory** *Groups-Big*
**imports** *Power Equiv-Relations*
**begin**

## 48.1 Generic monoid operation over a set

**locale** *comm-monoid-set* = *comm-monoid*

**begin**

### 48.1.1 Standard sum or product indexed by a finite set

**interpretation** *comp-fun-commute f*
⟨*proof*⟩

**interpretation** *comp?*: *comp-fun-commute* $f \circ g$
⟨*proof*⟩

**definition** $F :: ('b \Rightarrow 'a) \Rightarrow 'b\ set \Rightarrow 'a$
**where** *eq-fold*: $F\ g\ A = Finite\text{-}Set.fold\ (f \circ g)\ \mathbf{1}\ A$

**lemma** *infinite* [*simp*]: $\neg$ *finite* $A \Longrightarrow F\ g\ A = \mathbf{1}$
⟨*proof*⟩

**lemma** *empty* [*simp*]: $F\ g\ \{\} = \mathbf{1}$
⟨*proof*⟩

**lemma** *insert* [*simp*]: *finite* $A \Longrightarrow x \notin A \Longrightarrow F\ g\ (insert\ x\ A) = g\ x * F\ g\ A$
⟨*proof*⟩

**lemma** *remove*:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $F\ g\ A = g\ x * F\ g\ (A - \{x\})$
⟨*proof*⟩

**lemma** *insert-remove*: *finite* $A \Longrightarrow F\ g\ (insert\ x\ A) = g\ x * F\ g\ (A - \{x\})$
⟨*proof*⟩

**lemma** *insert-if*: *finite* $A \Longrightarrow F\ g\ (insert\ x\ A) = (if\ x \in A\ then\ F\ g\ A\ else\ g\ x *$
$F\ g\ A)$
⟨*proof*⟩

**lemma** *neutral*: $\forall x{\in}A.\ g\ x = \mathbf{1} \Longrightarrow F\ g\ A = \mathbf{1}$
⟨*proof*⟩

**lemma** *neutral-const* [*simp*]: $F\ (\lambda\text{-}.\ \mathbf{1})\ A = \mathbf{1}$
⟨*proof*⟩

**lemma** *union-inter*:
**assumes** *finite* $A$ **and** *finite* $B$
**shows** $F\ g\ (A \cup B) * F\ g\ (A \cap B) = F\ g\ A * F\ g\ B$
— The reversed orientation looks more natural, but LOOPS as a simprule!
⟨*proof*⟩

**corollary** *union-inter-neutral*:
**assumes** *finite* $A$ **and** *finite* $B$
**and** $\forall x \in A \cap B.\ g\ x = \mathbf{1}$

**shows** $F\ g\ (A \cup B) = F\ g\ A * F\ g\ B$
⟨*proof*⟩

**corollary** *union-disjoint*:
**assumes** *finite* $A$ **and** *finite* $B$
**assumes** $A \cap B = \{\}$
**shows** $F\ g\ (A \cup B) = F\ g\ A * F\ g\ B$
⟨*proof*⟩

**lemma** *union-diff2*:
**assumes** *finite* $A$ **and** *finite* $B$
**shows** $F\ g\ (A \cup B) = F\ g\ (A - B) * F\ g\ (B - A) * F\ g\ (A \cap B)$
⟨*proof*⟩

**lemma** *subset-diff*:
**assumes** $B \subseteq A$ **and** *finite* $A$
**shows** $F\ g\ A = F\ g\ (A - B) * F\ g\ B$
⟨*proof*⟩

**lemma** *Int-Diff*:
**assumes** *finite* $A$
**shows** $F\ g\ A = F\ g\ (A \cap B) * F\ g\ (A - B)$
⟨*proof*⟩

**lemma** *setdiff-irrelevant*:
**assumes** *finite* $A$
**shows** $F\ g\ (A - \{x.\ g\ x = z\}) = F\ g\ A$
⟨*proof*⟩

**lemma** *not-neutral-contains-not-neutral*:
**assumes** $F\ g\ A \neq \mathbf{1}$
**obtains** $a$ **where** $a \in A$ **and** $g\ a \neq \mathbf{1}$
⟨*proof*⟩

**lemma** *reindex*:
**assumes** *inj-on* $h\ A$
**shows** $F\ g\ (h\ \text{‘}\ A) = F\ (g \circ h)\ A$
⟨*proof*⟩

**lemma** *cong* [*fundef-cong*]:
**assumes** $A = B$
**assumes** *g-h*: $\bigwedge x.\ x \in B \Longrightarrow g\ x = h\ x$
**shows** $F\ g\ A = F\ h\ B$
⟨*proof*⟩

**lemma** *cong-simp* [*cong*]:
$[\![\ A = B;\ \bigwedge x.\ x \in B =simp=> g\ x = h\ x\ ]\!] \Longrightarrow F\ (\lambda x.\ g\ x)\ A = F\ (\lambda x.\ h\ x)\ B$
⟨*proof*⟩

**lemma** *reindex-cong*:
**assumes** $inj\text{-}on\ l\ B$
**assumes** $A = l\ `\ B$
**assumes** $\bigwedge x.\ x \in B \Longrightarrow g\ (l\ x) = h\ x$
**shows** $F\ g\ A = F\ h\ B$
⟨*proof*⟩

**lemma** *image-eq*:
**assumes** $inj\text{-}on\ g\ A$
**shows** $F\ (\lambda x.\ x)\ (g\ `\ A) = F\ g\ A$
⟨*proof*⟩

**lemma** *UNION-disjoint*:
**assumes** $finite\ I$ **and** $\forall i{\in}I.\ finite\ (A\ i)$
**and** $\forall i{\in}I.\ \forall j{\in}I.\ i \neq j \longrightarrow A\ i \cap A\ j = \{\}$
**shows** $F\ g\ (\bigcup (A\ `\ I)) = F\ (\lambda x.\ F\ g\ (A\ x))\ I$
⟨*proof*⟩

**lemma** *Union-disjoint*:
**assumes** $\forall A{\in}C.\ finite\ A\ \forall A{\in}C.\ \forall B{\in}C.\ A \neq B \longrightarrow A \cap B = \{\}$
**shows** $F\ g\ (\bigcup C) = (F \circ F)\ g\ C$
⟨*proof*⟩

**lemma** *distrib*: $F\ (\lambda x.\ g\ x * h\ x)\ A = F\ g\ A * F\ h\ A$
⟨*proof*⟩

**lemma** *Sigma*:
**assumes** $finite\ A\ \forall x{\in}A.\ finite\ (B\ x)$
**shows** $F\ (\lambda x.\ F\ (g\ x)\ (B\ x))\ A = F\ (case\text{-}prod\ g)\ (SIGMA\ x{:}A.\ B\ x)$
⟨*proof*⟩

**lemma** *related*:
**assumes** *Re*: $R\ \mathbf{1}\ \mathbf{1}$
**and** *Rop*: $\forall x1\ y1\ x2\ y2.\ R\ x1\ x2 \wedge R\ y1\ y2 \longrightarrow R\ (x1 * y1)\ (x2 * y2)$
**and** *fin*: $finite\ S$
**and** *R-h-g*: $\forall x{\in}S.\ R\ (h\ x)\ (g\ x)$
**shows** $R\ (F\ h\ S)\ (F\ g\ S)$
⟨*proof*⟩

**lemma** *mono-neutral-cong-left*:
**assumes** $finite\ T$
**and** $S \subseteq T$
**and** $\forall i \in T - S.\ h\ i = \mathbf{1}$
**and** $\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x$
**shows** $F\ g\ S = F\ h\ T$
⟨*proof*⟩

**lemma** *mono-neutral-cong-right*:
$finite\ T \Longrightarrow S \subseteq T \Longrightarrow \forall i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x)$

$\Longrightarrow$
$F\ g\ T = F\ h\ S$
$\langle proof \rangle$

**lemma** *mono-neutral-left*: *finite* $T \Longrightarrow S \subseteq T \Longrightarrow \forall\, i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow F\ g\ S = F\ g\ T$
$\langle proof \rangle$

**lemma** *mono-neutral-right*: *finite* $T \Longrightarrow S \subseteq T \Longrightarrow \forall\, i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow F\ g\ T = F\ g\ S$
$\langle proof \rangle$

**lemma** *mono-neutral-cong*:
**assumes** [*simp*]: *finite* $T$ *finite* $S$
**and** $*$: $\bigwedge i.\ i \in T - S \Longrightarrow h\ i = \mathbf{1}$ $\bigwedge i.\ i \in S - T \Longrightarrow g\ i = \mathbf{1}$
**and** *gh*: $\bigwedge x.\ x \in S \cap T \Longrightarrow g\ x = h\ x$
**shows** $F\ g\ S = F\ h\ T$
$\langle proof \rangle$

**lemma** *reindex-bij-betw*: *bij-betw* $h\ S\ T \Longrightarrow F\ (\lambda x.\ g\ (h\ x))\ S = F\ g\ T$
$\langle proof \rangle$

**lemma** *reindex-bij-witness*:
**assumes** *witness*:
$\bigwedge a.\ a \in S \Longrightarrow i\ (j\ a) = a$
$\bigwedge a.\ a \in S \Longrightarrow j\ a \in T$
$\bigwedge b.\ b \in T \Longrightarrow j\ (i\ b) = b$
$\bigwedge b.\ b \in T \Longrightarrow i\ b \in S$
**assumes** *eq*:
$\bigwedge a.\ a \in S \Longrightarrow h\ (j\ a) = g\ a$
**shows** $F\ g\ S = F\ h\ T$
$\langle proof \rangle$

**lemma** *reindex-bij-betw-not-neutral*:
**assumes** *fin*: *finite* $S'$ *finite* $T'$
**assumes** *bij*: *bij-betw* $h\ (S - S')\ (T - T')$
**assumes** *nn*:
$\bigwedge a.\ a \in S' \Longrightarrow g\ (h\ a) = z$
$\bigwedge b.\ b \in T' \Longrightarrow g\ b = z$
**shows** $F\ (\lambda x.\ g\ (h\ x))\ S = F\ g\ T$
$\langle proof \rangle$

**lemma** *reindex-nontrivial*:
**assumes** *finite* $A$
**and** *nz*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \neq y \Longrightarrow h\ x = h\ y \Longrightarrow g\ (h\ x) = \mathbf{1}$
**shows** $F\ g\ (h\ `\ A) = F\ (g \circ h)\ A$
$\langle proof \rangle$

**lemma** *reindex-bij-witness-not-neutral*:

**assumes** *fin*: *finite* $S'$ *finite* $T'$
**assumes** *witness*:
$\bigwedge a.\ a \in S - S' \Longrightarrow i\ (j\ a) = a$
$\bigwedge a.\ a \in S - S' \Longrightarrow j\ a \in T - T'$
$\bigwedge b.\ b \in T - T' \Longrightarrow j\ (i\ b) = b$
$\bigwedge b.\ b \in T - T' \Longrightarrow i\ b \in S - S'$
**assumes** *nn*:
$\bigwedge a.\ a \in S' \Longrightarrow g\ a = z$
$\bigwedge b.\ b \in T' \Longrightarrow h\ b = z$
**assumes** *eq*:
$\bigwedge a.\ a \in S \Longrightarrow h\ (j\ a) = g\ a$
**shows** $F\ g\ S = F\ h\ T$
⟨*proof*⟩

**lemma** *delta-remove*:
**assumes** *fS*: *finite* $S$
**shows** $F\ (\lambda k.\ \textit{if}\ k = a\ \textit{then}\ b\ k\ \textit{else}\ c\ k)\ S = (\textit{if}\ a \in S\ \textit{then}\ b\ a * F\ c\ (S - \{a\})\ \textit{else}\ F\ c\ (S - \{a\}))$
⟨*proof*⟩

**lemma** *delta* [*simp*]:
**assumes** *fS*: *finite* $S$
**shows** $F\ (\lambda k.\ \textit{if}\ k = a\ \textit{then}\ b\ k\ \textit{else}\ \mathbf{1})\ S = (\textit{if}\ a \in S\ \textit{then}\ b\ a\ \textit{else}\ \mathbf{1})$
⟨*proof*⟩

**lemma** *delta'* [*simp*]:
**assumes** *fin*: *finite* $S$
**shows** $F\ (\lambda k.\ \textit{if}\ a = k\ \textit{then}\ b\ k\ \textit{else}\ \mathbf{1})\ S = (\textit{if}\ a \in S\ \textit{then}\ b\ a\ \textit{else}\ \mathbf{1})$
⟨*proof*⟩

**lemma** *If-cases*:
**fixes** $P :: 'b \Rightarrow \textit{bool}$ **and** $g\ h :: 'b \Rightarrow 'a$
**assumes** *fin*: *finite* $A$
**shows** $F\ (\lambda x.\ \textit{if}\ P\ x\ \textit{then}\ h\ x\ \textit{else}\ g\ x)\ A = F\ h\ (A \cap \{x.\ P\ x\}) * F\ g\ (A \cap - \{x.\ P\ x\})$
⟨*proof*⟩

**lemma** *cartesian-product*: $F\ (\lambda x.\ F\ (g\ x)\ B)\ A = F\ (\textit{case-prod}\ g)\ (A \times B)$
⟨*proof*⟩

**lemma** *cartesian-product'*:
$F\ g\ (A \times B) = F\ (\lambda x.\ F\ (\lambda y.\ g\ (x,y))\ B)\ A$
⟨*proof*⟩

**lemma** *inter-restrict*:
**assumes** *finite* $A$
**shows** $F\ g\ (A \cap B) = F\ (\lambda x.\ \textit{if}\ x \in B\ \textit{then}\ g\ x\ \textit{else}\ \mathbf{1})\ A$
⟨*proof*⟩

**lemma** *inter-filter*:
*finite* $A \Longrightarrow F\ g\ \{x \in A.\ P\ x\} = F\ (\lambda x.\ \mathit{if}\ P\ x\ \mathit{then}\ g\ x\ \mathit{else}\ \mathbf{1})\ A$
⟨*proof*⟩

**lemma** *Union-comp*:
**assumes** $\forall A \in B.$ *finite* $A$
**and** $\bigwedge A1\ A2\ x.\ A1 \in B \Longrightarrow A2 \in B \Longrightarrow A1 \neq A2 \Longrightarrow x \in A1 \Longrightarrow x \in A2 \Longrightarrow g\ x = \mathbf{1}$
**shows** $F\ g\ (\bigcup B) = (F \circ F)\ g\ B$
⟨*proof*⟩

**lemma** *swap*: $F\ (\lambda i.\ F\ (g\ i)\ B)\ A = F\ (\lambda j.\ F\ (\lambda i.\ g\ i\ j)\ A)\ B$
⟨*proof*⟩

**lemma** *swap-restrict*:
*finite* $A \Longrightarrow$ *finite* $B \Longrightarrow$
$F\ (\lambda x.\ F\ (g\ x)\ \{y.\ y \in B \wedge R\ x\ y\})\ A = F\ (\lambda y.\ F\ (\lambda x.\ g\ x\ y)\ \{x.\ x \in A \wedge R\ x\ y\})\ B$
⟨*proof*⟩

**lemma** *image-gen*:
**assumes** *fin*: *finite* $S$
**shows** $F\ h\ S = F\ (\lambda y.\ F\ h\ \{x.\ x \in S \wedge g\ x = y\})\ (g\ `\ S)$
⟨*proof*⟩

**lemma** *group*:
**assumes** *fS*: *finite* $S$ **and** *fT*: *finite* $T$ **and** *fST*: $g\ `\ S \subseteq T$
**shows** $F\ (\lambda y.\ F\ h\ \{x.\ x \in S \wedge g\ x = y\})\ T = F\ h\ S$
⟨*proof*⟩

**lemma** *Plus*:
**fixes** $A :: 'b\ set$ **and** $B :: 'c\ set$
**assumes** *fin*: *finite* $A$ *finite* $B$
**shows** $F\ g\ (A <+> B) = F\ (g \circ Inl)\ A * F\ (g \circ Inr)\ B$
⟨*proof*⟩

**lemma** *same-carrier*:
**assumes** *finite* $C$
**assumes** *subset*: $A \subseteq C\ B \subseteq C$
**assumes** *trivial*: $\bigwedge a.\ a \in C - A \Longrightarrow g\ a = \mathbf{1}\ \bigwedge b.\ b \in C - B \Longrightarrow h\ b = \mathbf{1}$
**shows** $F\ g\ A = F\ h\ B \longleftrightarrow F\ g\ C = F\ h\ C$
⟨*proof*⟩

**lemma** *same-carrierI*:
**assumes** *finite* $C$
**assumes** *subset*: $A \subseteq C\ B \subseteq C$
**assumes** *trivial*: $\bigwedge a.\ a \in C - A \Longrightarrow g\ a = \mathbf{1}\ \bigwedge b.\ b \in C - B \Longrightarrow h\ b = \mathbf{1}$
**assumes** $F\ g\ C = F\ h\ C$

**shows** $F\ g\ A = F\ h\ B$
⟨*proof*⟩

**lemma** *eq-general*:
**assumes** $B$: $\bigwedge y.\ y \in B \Longrightarrow \exists!x.\ x \in A \wedge h\ x = y$ **and** $A$: $\bigwedge x.\ x \in A \Longrightarrow h\ x \in B \wedge \gamma(h\ x) = \varphi\ x$
**shows** $F\ \varphi\ A = F\ \gamma\ B$
⟨*proof*⟩

**lemma** *eq-general-inverses*:
**assumes** $B$: $\bigwedge y.\ y \in B \Longrightarrow k\ y \in A \wedge h(k\ y) = y$ **and** $A$: $\bigwedge x.\ x \in A \Longrightarrow h\ x \in B \wedge k(h\ x) = x \wedge \gamma(h\ x) = \varphi\ x$
**shows** $F\ \varphi\ A = F\ \gamma\ B$
⟨*proof*⟩

### 48.1.2 HOL Light variant: sum/product indexed by the non-neutral subset

NB only a subset of the properties above are proved

**definition** $G :: [{'b} \Rightarrow {'a},{'b}\ set] \Rightarrow {'a}$
**where** $G\ p\ I \equiv$ *if finite* $\{x \in I.\ p\ x \neq \mathbf{1}\}$ *then* $F\ p\ \{x \in I.\ p\ x \neq \mathbf{1}\}$ *else* $\mathbf{1}$

**lemma** *finite-Collect-op*:
**shows** $[\![$*finite* $\{i \in I.\ x\ i \neq \mathbf{1}\}$; *finite* $\{i \in I.\ y\ i \neq \mathbf{1}\}]\!] \Longrightarrow$ *finite* $\{i \in I.\ x\ i * y\ i \neq \mathbf{1}\}$
⟨*proof*⟩

**lemma** *empty'* [*simp*]: $G\ p\ \{\} = \mathbf{1}$
⟨*proof*⟩

**lemma** *eq-sum* [*simp*]: *finite* $I \Longrightarrow G\ p\ I = F\ p\ I$
⟨*proof*⟩

**lemma** *insert'* [*simp*]:
**assumes** *finite* $\{x \in I.\ p\ x \neq \mathbf{1}\}$
**shows** $G\ p\ (insert\ i\ I) = ($*if* $i \in I$ *then* $G\ p\ I$ *else* $p\ i * G\ p\ I)$
⟨*proof*⟩

**lemma** *distrib-triv'*:
**assumes** *finite* $I$
**shows** $G\ (\lambda i.\ g\ i * h\ i)\ I = G\ g\ I * G\ h\ I$
⟨*proof*⟩

**lemma** *non-neutral'*: $G\ g\ \{x \in I.\ g\ x \neq \mathbf{1}\} = G\ g\ I$
⟨*proof*⟩

**lemma** *distrib'*:
**assumes** *finite* $\{x \in I.\ g\ x \neq \mathbf{1}\}$ *finite* $\{x \in I.\ h\ x \neq \mathbf{1}\}$
**shows** $G\ (\lambda i.\ g\ i * h\ i)\ I = G\ g\ I * G\ h\ I$

⟨*proof*⟩

**lemma** *cong'*:
  **assumes** $A = B$
  **assumes** *g-h*: $\bigwedge x.\ x \in B \Longrightarrow g\ x = h\ x$
  **shows** $G\ g\ A = G\ h\ B$
  ⟨*proof*⟩

**lemma** *mono-neutral-cong-left'*:
  **assumes** $S \subseteq T$
    **and** $\bigwedge i.\ i \in T - S \Longrightarrow h\ i = \mathbf{1}$
    **and** $\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x$
  **shows** $G\ g\ S = G\ h\ T$
⟨*proof*⟩

**lemma** *mono-neutral-cong-right'*:
  $S \subseteq T \Longrightarrow \forall\, i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow g\ x = h\ x) \Longrightarrow$
    $G\ g\ T = G\ h\ S$
  ⟨*proof*⟩

**lemma** *mono-neutral-left'*: $S \subseteq T \Longrightarrow \forall\, i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow G\ g\ S = G\ g\ T$
  ⟨*proof*⟩

**lemma** *mono-neutral-right'*: $S \subseteq T \Longrightarrow \forall\, i \in T - S.\ g\ i = \mathbf{1} \Longrightarrow G\ g\ T = G\ g\ S$
  ⟨*proof*⟩

**end**

## 48.2 Generalized summation over a set

**context** *comm-monoid-add*
**begin**

**sublocale** *sum*: *comm-monoid-set plus 0*
  **defines** *sum* = *sum.F* **and** *sum'* = *sum.G* ⟨*proof*⟩

**abbreviation** *Sum* (‹$\sum$›)
  **where** $\sum\ \equiv$ *sum* $(\lambda x.\ x)$

**end**

Now: lots of fancy syntax. First, *sum* $(\lambda x.\ e)\ A$ is written $\sum x{\in}A.\ e$.

**syntax** (*ASCII*)
  *-sum* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*::*comm-monoid-add* (‹(‹*indent=3 notation=*‹*binder SUM*››*SUM* (-/:-)./ -)› [*0*, *51*, *10*] *10*)
**syntax**
  *-sum* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b*::*comm-monoid-add* (‹(‹*indent=2 notation=*‹*binder* $\sum$››$\sum$ (-/∈-)./ -)› [*0*, *51*, *10*] *10*)

**syntax-consts**
*-sum* $\rightleftharpoons$ *sum*

**translations** — Beware of argument permutation!
$\sum i{\in}A.\ b \rightleftharpoons CONST\ sum\ (\lambda i.\ b)\ A$

Instead of $\sum x{\in}\{x.\ P\}.\ e$ we introduce the shorter $\sum x|P.\ e$.

**syntax** (*ASCII*)
*-qsum* :: *pttrn* $\Rightarrow$ *bool* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (‹(‹indent=3 notation=‹binder SUM Collect››SUM - |/ -./ -)› [0, 0, 10] 10)
**syntax**
*-qsum* :: *pttrn* $\Rightarrow$ *bool* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (‹(‹indent=2 notation=‹binder $\sum$ Collect››$\sum$ -| (-)./ -)› [0, 0, 10] 10)

**syntax-consts**
*-qsum* == *sum*
**translations**
$\sum x|P.\ t$ => $CONST\ sum\ (\lambda x.\ t)\ \{x.\ P\}$
⟨*ML*⟩

### 48.2.1 Properties in more restricted classes of structures

**lemma** *sum-Un*:
*finite A* $\Longrightarrow$ *finite B* $\Longrightarrow$ *sum f* $(A \cup B)$ = *sum f A* + *sum f B* − *sum f* $(A \cap B)$
**for** *f* :: *'b* $\Rightarrow$ *'a::ab-group-add*
⟨*proof*⟩

**lemma** *sum-Un2*:
**assumes** *finite* $(A \cup B)$
**shows** *sum f* $(A \cup B)$ = *sum f* $(A - B)$ + *sum f* $(B - A)$ + *sum f* $(A \cap B)$
⟨*proof*⟩

**lemma** *sum-diff*:
**fixes** *f* :: *'b* $\Rightarrow$ *'a::ab-group-add*
**assumes** *finite A* $B \subseteq A$
**shows** *sum f* $(A - B)$ = *sum f A* − *sum f B*
⟨*proof*⟩

**lemma** *sum-diff1*:
**fixes** *f* :: *'b* $\Rightarrow$ *'a::ab-group-add*
**assumes** *finite A*
**shows** *sum f* $(A - \{a\})$ = (*if* $a \in A$ *then sum f A* − *f a else sum f A*)
⟨*proof*⟩

**lemma** *sum-diff1'-aux*:
**fixes** *f* :: *'a* $\Rightarrow$ *'b::ab-group-add*
**assumes** *finite F* $\{i \in I.\ f\ i \neq 0\} \subseteq F$

**shows** $sum'\ f\ (I - \{i\}) = (if\ i \in I\ then\ sum'\ f\ I - f\ i\ else\ sum'\ f\ I)$
⟨*proof*⟩

**lemma** *sum-diff1'*:
**fixes** $f :: 'a \Rightarrow 'b::ab\text{-}group\text{-}add$
**assumes** $finite\ \{i \in I.\ f\ i \neq 0\}$
**shows** $sum'\ f\ (I - \{i\}) = (if\ i \in I\ then\ sum'\ f\ I - f\ i\ else\ sum'\ f\ I)$
⟨*proof*⟩

**lemma** (**in** *ordered-comm-monoid-add*) *sum-mono*:
$(\bigwedge i.\ i{\in}K \Longrightarrow f\ i \leq g\ i) \Longrightarrow (\sum i{\in}K.\ f\ i) \leq (\sum i{\in}K.\ g\ i)$
⟨*proof*⟩

**lemma** (**in** *ordered-cancel-comm-monoid-add*) *sum-strict-mono-strong*:
**assumes** $finite\ A\ \ a \in A\ \ f\ a < g\ a$
**and** $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$
**shows** $sum\ f\ A < sum\ g\ A$
⟨*proof*⟩

**lemma** (**in** *strict-ordered-comm-monoid-add*) *sum-strict-mono*:
**assumes** $finite\ A\ \ A \neq \{\}$
**and** $\bigwedge x.\ x \in A \Longrightarrow f\ x < g\ x$
**shows** $sum\ f\ A < sum\ g\ A$
⟨*proof*⟩

**lemma** *sum-strict-mono-ex1*:
**fixes** $f\ g :: 'i \Rightarrow 'a::ordered\text{-}cancel\text{-}comm\text{-}monoid\text{-}add$
**assumes** $finite\ A$
**and** $\forall x{\in}A.\ f\ x \leq g\ x$
**and** $\exists a{\in}A.\ f\ a < g\ a$
**shows** $sum\ f\ A < sum\ g\ A$
⟨*proof*⟩

**lemma** *sum-mono-inv*:
**fixes** $f\ g :: 'i \Rightarrow 'a :: ordered\text{-}cancel\text{-}comm\text{-}monoid\text{-}add$
**assumes** *eq*: $sum\ f\ I = sum\ g\ I$
**assumes** *le*: $\bigwedge i.\ i \in I \Longrightarrow f\ i \leq g\ i$
**assumes** *i*: $i \in I$
**assumes** *I*: $finite\ I$
**shows** $f\ i = g\ i$
⟨*proof*⟩

**lemma** *member-le-sum*:
**fixes** $f :: \text{-} \Rightarrow 'b::\{semiring\text{-}1,\ ordered\text{-}comm\text{-}monoid\text{-}add\}$
**assumes** $i \in A$
**and** *le*: $\bigwedge x.\ x \in A - \{i\} \Longrightarrow 0 \leq f\ x$
**and** $finite\ A$
**shows** $f\ i \leq sum\ f\ A$
⟨*proof*⟩

**lemma** *sum-negf*: $(\sum x{\in}A.\ -\ f\ x) = -\ (\sum x{\in}A.\ f\ x)$
**for** $f$ :: $'b \Rightarrow\ 'a$::*ab-group-add*
⟨*proof*⟩

**lemma** *sum-subtractf*: $(\sum x{\in}A.\ f\ x\ -\ g\ x) = (\sum x{\in}A.\ f\ x)\ -\ (\sum x{\in}A.\ g\ x)$
**for** $f\ g$ :: $'b \Rightarrow 'a$::*ab-group-add*
⟨*proof*⟩

**lemma** *sum-subtractf-nat*:
$(\bigwedge x.\ x \in A \Longrightarrow g\ x \le f\ x) \Longrightarrow (\sum x{\in}A.\ f\ x\ -\ g\ x) = (\sum x{\in}A.\ f\ x)\ -\ (\sum x{\in}A.\ g\ x)$
**for** $f\ g$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**context** *ordered-comm-monoid-add*
**begin**

**lemma** *sum-nonneg*: $(\bigwedge x.\ x \in A \Longrightarrow 0 \le f\ x) \Longrightarrow 0 \le sum\ f\ A$
⟨*proof*⟩

**lemma** *sum-nonpos*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \le 0) \Longrightarrow sum\ f\ A \le 0$
⟨*proof*⟩

**lemma** *sum-nonneg-eq-0-iff*:
$finite\ A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow 0 \le f\ x) \Longrightarrow sum\ f\ A = 0 \longleftrightarrow (\forall x{\in}A.\ f\ x = 0)$
⟨*proof*⟩

**lemma** *sum-nonneg-0*:
$finite\ s \Longrightarrow (\bigwedge i.\ i \in s \Longrightarrow f\ i \ge 0) \Longrightarrow (\sum\ i \in s.\ f\ i) = 0 \Longrightarrow i \in s \Longrightarrow f\ i = 0$
⟨*proof*⟩

**lemma** *sum-nonneg-leq-bound*:
**assumes** $finite\ s\ \bigwedge i.\ i \in s \Longrightarrow f\ i \ge 0\ (\sum i \in s.\ f\ i) = B\ i \in s$
**shows** $f\ i \le B$
⟨*proof*⟩

**lemma** *sum-mono2*:
**assumes** *fin*: *finite B*
**and** *sub*: $A \subseteq B$
**and** *nn*: $\bigwedge b.\ b \in B{-}A \Longrightarrow 0 \le f\ b$
**shows** $sum\ f\ A \le sum\ f\ B$
⟨*proof*⟩

**lemma** *sum-le-included*:
**assumes** *finite s finite t*
**and** $\forall y{\in}t.\ 0 \le g\ y\ (\forall x{\in}s.\ \exists y{\in}t.\ i\ y = x \land f\ x \le g\ y)$
**shows** $sum\ f\ s \le sum\ g\ t$
⟨*proof*⟩

**end**

**lemma** (**in** *canonically-ordered-monoid-add*) *sum-eq-0-iff* [*simp*]:
$finite\ F \Longrightarrow (sum\ f\ F = 0) = (\forall a{\in}F.\ f\ a = 0)$
⟨*proof*⟩

**context** *semiring-0*
**begin**

**lemma** *sum-distrib-left*: $r * sum\ f\ A = (\sum n{\in}A.\ r * f\ n)$
⟨*proof*⟩

**lemma** *sum-distrib-right*: $sum\ f\ A * r = (\sum n{\in}A.\ f\ n * r)$
⟨*proof*⟩

**end**

**lemma** *sum-divide-distrib*: $sum\ f\ A\ /\ r = (\sum n{\in}A.\ f\ n\ /\ r)$
**for** $r :: 'a::field$
⟨*proof*⟩

**lemma** *sum-abs*[*iff*]: $|sum\ f\ A| \leq sum\ (\lambda i.\ |f\ i|)\ A$
**for** $f :: 'a \Rightarrow 'b::ordered\text{-}ab\text{-}group\text{-}add\text{-}abs$
⟨*proof*⟩

**lemma** *sum-abs-ge-zero*[*iff*]: $0 \leq sum\ (\lambda i.\ |f\ i|)\ A$
**for** $f :: 'a \Rightarrow 'b::ordered\text{-}ab\text{-}group\text{-}add\text{-}abs$
⟨*proof*⟩

**lemma** *abs-sum-abs*[*simp*]: $|\sum a{\in}A.\ |f\ a|| = (\sum a{\in}A.\ |f\ a|)$
**for** $f :: 'a \Rightarrow 'b::ordered\text{-}ab\text{-}group\text{-}add\text{-}abs$
⟨*proof*⟩

**lemma** *sum-product*:
**fixes** $f :: 'a \Rightarrow 'b::semiring\text{-}0$
**shows** $sum\ f\ A * sum\ g\ B = (\sum i{\in}A.\ \sum j{\in}B.\ f\ i * g\ j)$
⟨*proof*⟩

**lemma** *sum-mult-sum-if-inj*:
**fixes** $f :: 'a \Rightarrow 'b::semiring\text{-}0$
**shows** $inj\text{-}on\ (\lambda(a,\ b).\ f\ a * g\ b)\ (A \times B) \Longrightarrow$
$sum\ f\ A * sum\ g\ B = sum\ id\ \{f\ a * g\ b\ |a\ b.\ a \in A \wedge b \in B\}$
⟨*proof*⟩

**lemma** *sum-SucD*: $sum\ f\ A = Suc\ n \Longrightarrow \exists a{\in}A.\ 0 < f\ a$
⟨*proof*⟩

**lemma** *sum-eq-Suc0-iff*:

$finite\ A \Longrightarrow sum\ f\ A = Suc\ 0 \longleftrightarrow (\exists\, a{\in}A.\ f\ a = Suc\ 0 \wedge (\forall\, b{\in}A.\ a \neq b \longrightarrow f\ b = 0))$
⟨*proof*⟩

**lemmas** *sum-eq-1-iff* = *sum-eq-Suc0-iff*[*simplified One-nat-def*[*symmetric*]]

**lemma** *sum-Un-nat*:
$finite\ A \Longrightarrow finite\ B \Longrightarrow sum\ f\ (A \cup B) = sum\ f\ A + sum\ f\ B - sum\ f\ (A \cap B)$
**for** $f :: 'a \Rightarrow nat$
— For the natural numbers, we have subtraction.
⟨*proof*⟩

**lemma** *sum-diff1-nat*: $sum\ f\ (A - \{a\}) = (if\ a \in A\ then\ sum\ f\ A - f\ a\ else\ sum\ f\ A)$
**for** $f :: 'a \Rightarrow nat$
⟨*proof*⟩

**lemma** *sum-diff-nat*:
**fixes** $f :: 'a \Rightarrow nat$
**assumes** *finite B* **and** $B \subseteq A$
**shows** $sum\ f\ (A - B) = sum\ f\ A - sum\ f\ B$
⟨*proof*⟩

**lemma** *sum-comp-morphism*:
$h\ 0 = 0 \Longrightarrow (\bigwedge x\ y.\ h\ (x + y) = h\ x + h\ y) \Longrightarrow sum\ (h \circ g)\ A = h\ (sum\ g\ A)$
⟨*proof*⟩

**lemma** (**in** *comm-semiring-1*) *dvd-sum*: $(\bigwedge a.\ a \in A \Longrightarrow d\ dvd\ f\ a) \Longrightarrow d\ dvd\ sum\ f\ A$
⟨*proof*⟩

**lemma** (**in** *ordered-comm-monoid-add*) *sum-pos*:
$finite\ I \Longrightarrow I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow 0 < f\ i) \Longrightarrow 0 < sum\ f\ I$
⟨*proof*⟩

**lemma** (**in** *ordered-comm-monoid-add*) *sum-pos2*:
**assumes** $I$: $finite\ I\ i \in I\ 0 < f\ i\ \bigwedge i.\ i \in I \Longrightarrow 0 \leq f\ i$
**shows** $0 < sum\ f\ I$
⟨*proof*⟩

**lemma** *sum-strict-mono2*:
**fixes** $f :: 'a \Rightarrow 'b$::*ordered-cancel-comm-monoid-add*
**assumes** $finite\ B\ A \subseteq B\ b \in B{-}A\ f\ b > 0$ **and** $\bigwedge x.\ x \in B \Longrightarrow f\ x \geq 0$
**shows** $sum\ f\ A < sum\ f\ B$
⟨*proof*⟩

**lemma** *sum-cong-Suc*:
**assumes** $0 \notin A\ \bigwedge x.\ Suc\ x \in A \Longrightarrow f\ (Suc\ x) = g\ (Suc\ x)$
**shows** $sum\ f\ A = sum\ g\ A$

⟨*proof*⟩

### 48.2.2 Cardinality as special case of *sum*

**lemma** *card-eq-sum*: $card\ A = sum\ (\lambda x.\ 1)\ A$
⟨*proof*⟩

**context** *semiring-1*
**begin**

**lemma** *sum-constant* [*simp*]:
$(\sum x \in A.\ y) = of\text{-}nat\ (card\ A) * y$
⟨*proof*⟩

**context**
**fixes** $A$
**assumes** ‹*finite A*›
**begin**

**lemma** *sum-of-bool-eq* [*simp*]:
‹$(\sum x \in A.\ of\text{-}bool\ (P\ x)) = of\text{-}nat\ (card\ (A \cap \{x.\ P\ x\}))$› **if** ‹*finite A*›
⟨*proof*⟩

**lemma** *sum-mult-of-bool-eq* [*simp*]:
‹$(\sum x \in A.\ f\ x * of\text{-}bool\ (P\ x)) = (\sum x \in (A \cap \{x.\ P\ x\}).\ f\ x)$›
⟨*proof*⟩

**lemma** *sum-of-bool-mult-eq* [*simp*]:
‹$(\sum x \in A.\ of\text{-}bool\ (P\ x) * f\ x) = (\sum x \in (A \cap \{x.\ P\ x\}).\ f\ x)$›
⟨*proof*⟩

**end**

**end**

**lemma** *sum-Suc*: $sum\ (\lambda x.\ Suc(f\ x))\ A = sum\ f\ A + card\ A$
⟨*proof*⟩

**lemma** *sum-bounded-above*:
**fixes** $K$ :: $'a$::{*semiring-1*,*ordered-comm-monoid-add*}
**assumes** *le*: $\bigwedge i.\ i \in A \Longrightarrow f\ i \leq K$
**shows** $sum\ f\ A \leq of\text{-}nat\ (card\ A) * K$
⟨*proof*⟩

**lemma** *sum-bounded-above-divide*:
**fixes** $K$ :: $'a$::*linordered-field*
**assumes** *le*: $\bigwedge i.\ i \in A \Longrightarrow f\ i \leq K\ /\ of\text{-}nat\ (card\ A)$ **and** *fin*: $finite\ A\ A \neq \{\}$
**shows** $sum\ f\ A \leq K$
⟨*proof*⟩

**lemma** *sum-bounded-above-strict*:
**fixes** $K$ :: $'a$::{*ordered-cancel-comm-monoid-add*,*semiring-1*}
**assumes** $\bigwedge i.\ i{\in}A \Longrightarrow f\ i < K\ card\ A > 0$
**shows** $sum\ f\ A < of\text{-}nat\ (card\ A) * K$
⟨*proof*⟩

**lemma** *sum-bounded-below*:
**fixes** $K$ :: $'a$::{*semiring-1*,*ordered-comm-monoid-add*}
**assumes** *le*: $\bigwedge i.\ i{\in}A \Longrightarrow K \le f\ i$
**shows** $of\text{-}nat\ (card\ A) * K \le sum\ f\ A$
⟨*proof*⟩

**lemma** *convex-sum-bound-le*:
**fixes** $x$ :: $'a \Rightarrow 'b$::*linordered-idom*
**assumes** *0*: $\bigwedge i.\ i \in I \Longrightarrow 0 \le x\ i$ **and** *1*: $sum\ x\ I = 1$
**and** $\delta$: $\bigwedge i.\ i \in I \Longrightarrow |a\ i - b| \le \delta$
**shows** $|(\sum i{\in}I.\ a\ i * x\ i) - b| \le \delta$
⟨*proof*⟩

**lemma** *card-UN-disjoint*:
**assumes** *finite* $I$ **and** $\forall i{\in}I.$ *finite* $(A\ i)$
**and** $\forall i{\in}I.\ \forall j{\in}I.\ i \ne j \longrightarrow A\ i \cap A\ j = \{\}$
**shows** $card\ (\bigcup (A\ `\ I)) = (\sum i{\in}I.\ card(A\ i))$
⟨*proof*⟩

**lemma** *card-Union-disjoint*:
**assumes** *pairwise disjnt* $C$ **and** *fin*: $\bigwedge A.\ A \in C \Longrightarrow$ *finite* $A$
**shows** $card\ (\bigcup C) = sum\ card\ C$
⟨*proof*⟩

**lemma** *card-Union-le-sum-card-weak*:
**fixes** $U$ :: $'a\ set\ set$
**assumes** $\forall u \in U.$ *finite* $u$
**shows** $card\ (\bigcup U) \le sum\ card\ U$
⟨*proof*⟩

**lemma** *card-Union-le-sum-card*:
**fixes** $U$ :: $'a\ set\ set$
**shows** $card\ (\bigcup U) \le sum\ card\ U$
⟨*proof*⟩

**lemma** *card-UN-le*:
**assumes** *finite* $I$
**shows** $card(\bigcup i{\in}I.\ A\ i) \le (\sum i{\in}I.\ card(A\ i))$
⟨*proof*⟩

**lemma** *card-quotient-disjoint*:
**assumes** *finite* $A$ *inj-on* $(\lambda x.\ \{x\}\ //\ r)\ A$

**shows** $card\ (A//r) = card\ A$
⟨*proof*⟩

**lemma** *sum-multicount-gen*:
**assumes** $finite\ s\ finite\ t\ \forall j\in t.\ (card\ \{i\in s.\ R\ i\ j\} = k\ j)$
**shows** $sum\ (\lambda i.\ (card\ \{j\in t.\ R\ i\ j\}))\ s = sum\ k\ t$
(**is** $?l = ?r$)
⟨*proof*⟩

**lemma** *sum-multicount*:
**assumes** $finite\ S\ finite\ T\ \forall j\in T.\ (card\ \{i\in S.\ R\ i\ j\} = k)$
**shows** $sum\ (\lambda i.\ card\ \{j\in T.\ R\ i\ j\})\ S = k * card\ T$ (**is** $?l = ?r$)
⟨*proof*⟩

**lemma** *sum-card-image*:
**assumes** $finite\ A$
**assumes** $pairwise\ (\lambda s\ t.\ disjnt\ (f\ s)\ (f\ t))\ A$
**shows** $sum\ card\ (f\ `\ A) = sum\ (\lambda a.\ card\ (f\ a))\ A$
⟨*proof*⟩

By Jakub Kdzioka:

**lemma** *sum-fun-comp*:
**assumes** $finite\ S\ finite\ R\ g\ `\ S \subseteq R$
**shows** $(\sum x \in S.\ f\ (g\ x)) = (\sum y \in R.\ of\text{-}nat\ (card\ \{x \in S.\ g\ x = y\}) * f\ y)$
⟨*proof*⟩

### 48.2.3 Cardinality of products

**lemma** *card-SigmaI* [*simp*]:
$finite\ A \Longrightarrow \forall a\in A.\ finite\ (B\ a) \Longrightarrow card\ (SIGMA\ x{:}\ A.\ B\ x) = (\sum a\in A.\ card\ (B\ a))$
⟨*proof*⟩

**lemma** *card-cartesian-product*: $card\ (A \times B) = card\ A * card\ B$
⟨*proof*⟩

**lemma** *card-cartesian-product-singleton*: $card\ (\{x\} \times A) = card\ A$
⟨*proof*⟩

## 48.3 Generalized product over a set

**context** *comm-monoid-mult*
**begin**

**sublocale** *prod*: *comm-monoid-set times 1*
**defines** $prod = prod.F$ **and** $prod' = prod.G$ ⟨*proof*⟩

**abbreviation** *Prod* (‹∏ ›)
**where** ∏ ≡ *prod* ($\lambda x.\ x$)

**end**

**syntax** (*ASCII*)
*-prod* :: *pttrn* => *'a set* => *'b* => *'b*::*comm-monoid-mult* (‹(‹*indent=4 notation=*‹*binder PROD*››*PROD* (-/:-)./ -)› [*0*, *51*, *10*] *10*)
**syntax**
*-prod* :: *pttrn* => *'a set* => *'b* => *'b*::*comm-monoid-mult* (‹(‹*indent=2 notation=*‹*binder* ∏ ››∏ (-/∈-)./ -)› [*0*, *51*, *10*] *10*)

**syntax-consts**
*-prod* == *prod*

**translations** — Beware of argument permutation!
∏ *i*∈*A*. *b* == *CONST prod* ($\lambda i.\ b$) *A*

Instead of ∏ $x \in \{x.\ P\}.\ e$ we introduce the shorter ∏ $x|P.\ e$.

**syntax** (*ASCII*)
*-qprod* :: *pttrn* ⇒ *bool* ⇒ *'a* ⇒ *'a* (‹(‹*indent=4 notation=*‹*binder PROD Collect*››*PROD* - |/ -./ -)› [*0*, *0*, *10*] *10*)
**syntax**
*-qprod* :: *pttrn* ⇒ *bool* ⇒ *'a* ⇒ *'a* (‹(‹*indent=2 notation=*‹*binder* ∏ *Collect*››∏ - | (-)./ -)› [*0*, *0*, *10*] *10*)

**syntax-consts**
*-qprod* == *prod*
**translations**
∏ *x*|*P*. *t* => *CONST prod* ($\lambda x.\ t$) {*x*. *P*}
⟨*ML*⟩

**context** *comm-monoid-mult*
**begin**

**lemma** *prod-dvd-prod*: ($\bigwedge a.\ a \in A \Longrightarrow f\ a\ dvd\ g\ a) \Longrightarrow prod\ f\ A\ dvd\ prod\ g\ A$
⟨*proof*⟩

**lemma** *prod-dvd-prod-subset*: $finite\ B \Longrightarrow A \subseteq B \Longrightarrow prod\ f\ A\ dvd\ prod\ f\ B$
⟨*proof*⟩

**end**

### 48.3.1 Properties in more restricted classes of structures

**context** *linordered-nonzero-semiring*
**begin**

**lemma** *prod-ge-1*: ($\bigwedge x.\ x \in A \Longrightarrow 1 \leq f\ x) \Longrightarrow 1 \leq prod\ f\ A$

⟨*proof*⟩

**lemma** *prod-le-1*:
**fixes** $f :: 'b \Rightarrow 'a$
**assumes** $\bigwedge x.\ x \in A \Longrightarrow 0 \leq f\ x \wedge f\ x \leq 1$
**shows** $prod\ f\ A \leq 1$
⟨*proof*⟩

**end**

**context** *comm-semiring-1*
**begin**

**lemma** *dvd-prod-eqI* [*intro*]:
**assumes** *finite A* **and** $a \in A$ **and** $b = f\ a$
**shows** *b dvd prod f A*
⟨*proof*⟩

**lemma** *dvd-prodI* [*intro*]: *finite A* $\Longrightarrow a \in A \Longrightarrow$ *f a dvd prod f A*
⟨*proof*⟩

**lemma** *prod-zero*:
**assumes** *finite A* **and** $\exists\, a{\in}A.\ f\ a = 0$
**shows** $prod\ f\ A = 0$
⟨*proof*⟩

**lemma** *prod-dvd-prod-subset2*:
**assumes** *finite B* **and** $A \subseteq B$ **and** $\bigwedge a.\ a \in A \Longrightarrow f\ a\ dvd\ g\ a$
**shows** *prod f A dvd prod g B*
⟨*proof*⟩

**end**

**lemma** (**in** *semidom*) *prod-zero-iff* [*simp*]:
**fixes** $f :: 'b \Rightarrow 'a$
**assumes** *finite A*
**shows** $prod\ f\ A = 0 \longleftrightarrow (\exists\, a{\in}A.\ f\ a = 0)$
⟨*proof*⟩

**lemma** (**in** *semidom-divide*) *prod-diff1*:
**assumes** *finite A* **and** $f\ a \neq 0$
**shows** $prod\ f\ (A - \{a\}) = (if\ a \in A\ then\ prod\ f\ A\ div\ f\ a\ else\ prod\ f\ A)$
⟨*proof*⟩

**lemma** *prod-uminus*: $(\prod x{\in}A.\ -f\ x :: 'a :: comm\text{-}ring\text{-}1) = (-1)\ \hat{}\ card\ A *$ $(\prod x{\in}A.\ f\ x)$
⟨*proof*⟩

**lemma** *prod-diff*:

**fixes** $f$ :: $'a \Rightarrow 'b$ :: *field*
**assumes** *finite* $A$ $B \subseteq A$ $\bigwedge x.\ x \in B \Longrightarrow f\ x \neq 0$
**shows** $prod\ f\ (A - B) = prod\ f\ A\ /\ prod\ f\ B$
⟨*proof*⟩

**lemma** *sum-zero-power* [*simp*]: $(\sum i{\in}A.\ c\ i * 0\hat{\ }i) = (if\ finite\ A \wedge 0 \in A\ then\ c\ 0\ else\ 0)$
**for** $c$ :: $nat \Rightarrow 'a$::*division-ring*
⟨*proof*⟩

**lemma** *sum-zero-power'* [*simp*]:
$(\sum i{\in}A.\ c\ i * 0\hat{\ }i\ /\ d\ i) = (if\ finite\ A \wedge 0 \in A\ then\ c\ 0\ /\ d\ 0\ else\ 0)$
**for** $c$ :: $nat \Rightarrow 'a$::*field*
⟨*proof*⟩

**lemma** (**in** *field*) *prod-inversef*: $prod\ (inverse \circ f)\ A = inverse\ (prod\ f\ A)$
⟨*proof*⟩

**lemma** (**in** *field*) *prod-dividef*: $(\prod x{\in}A.\ f\ x\ /\ g\ x) = prod\ f\ A\ /\ prod\ g\ A$
⟨*proof*⟩

**lemma** *prod-Un*:
**fixes** $f$ :: $'b \Rightarrow 'a$ :: *field*
**assumes** *finite* $A$ **and** *finite* $B$
**and** $\forall x{\in}A \cap B.\ f\ x \neq 0$
**shows** $prod\ f\ (A \cup B) = prod\ f\ A * prod\ f\ B\ /\ prod\ f\ (A \cap B)$
⟨*proof*⟩

**context** *linordered-semidom*
**begin**

**lemma** *prod-nonneg*: $(\bigwedge a.\ a{\in}A \Longrightarrow 0 \leq f\ a) \Longrightarrow 0 \leq prod\ f\ A$
⟨*proof*⟩

**lemma** *prod-pos*: $(\bigwedge a.\ a{\in}A \Longrightarrow 0 < f\ a) \Longrightarrow 0 < prod\ f\ A$
⟨*proof*⟩

**lemma** *prod-mono*:
$(\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i \wedge f\ i \leq g\ i) \Longrightarrow prod\ f\ A \leq prod\ g\ A$
⟨*proof*⟩

Only one needs to be strict

**lemma** *prod-mono-strict*:
**assumes** $i \in A$ $f\ i < g\ i$
**assumes** *finite* $A$
**assumes** $\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i \wedge f\ i \leq g\ i$
**assumes** $\bigwedge i.\ i \in A \Longrightarrow 0 < g\ i$
**shows** $prod\ f\ A < prod\ g\ A$
⟨*proof*⟩

**lemma** *prod-le-power*:
**assumes** $A$: $\bigwedge i.\ i \in A \Longrightarrow 0 \leq f\ i \wedge f\ i \leq n$ $card\ A \leq k$ **and** $n \geq 1$
**shows** $prod\ f\ A \leq n\ \hat{}\ k$
⟨*proof*⟩

**end**

**lemma** *prod-mono2*:
**fixes** $f$ :: $'a \Rightarrow\ 'b$ :: *linordered-idom*
**assumes** *fin*: *finite* $B$
**and** *sub*: $A \subseteq B$
**and** *nn*: $\bigwedge b.\ b \in B-A \Longrightarrow 1 \leq f\ b$
**and** $A$: $\bigwedge a.\ a \in A \Longrightarrow 0 \leq f\ a$
**shows** $prod\ f\ A \leq prod\ f\ B$
⟨*proof*⟩

**lemma** *less-1-prod*:
**fixes** $f$ :: $'a \Rightarrow\ 'b$::*linordered-idom*
**shows** *finite* $I \Longrightarrow I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow 1 < f\ i) \Longrightarrow 1 < prod\ f\ I$
⟨*proof*⟩

**lemma** *less-1-prod2*:
**fixes** $f$ :: $'a \Rightarrow\ 'b$::*linordered-idom*
**assumes** $I$: *finite* $I$ $i \in I$ $1 < f\ i$ $\bigwedge i.\ i \in I \Longrightarrow 1 \leq f\ i$
**shows** $1 < prod\ f\ I$
⟨*proof*⟩

**lemma** (**in** *linordered-field*) *abs-prod*: $|prod\ f\ A| = (\prod x{\in}A.\ |f\ x|)$
⟨*proof*⟩

**lemma** *prod-eq-1-iff* [*simp*]: *finite* $A \Longrightarrow prod\ f\ A = 1 \longleftrightarrow (\forall\ a{\in}A.\ f\ a = 1)$
**for** $f$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**lemma** *prod-pos-nat-iff* [*simp*]: *finite* $A \Longrightarrow prod\ f\ A > 0 \longleftrightarrow (\forall\ a{\in}A.\ f\ a > 0)$
**for** $f$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**lemma** *prod-constant* [*simp*]: $(\prod x{\in}\ A.\ y) = y\ \hat{}\ card\ A$
**for** $y$ :: $'a$::*comm-monoid-mult*
⟨*proof*⟩

**lemma** *prod-diff-swap*:
**fixes** $f$ :: $'a \Rightarrow\ 'b$ :: *comm-ring-1*
**shows** $prod\ (\lambda x.\ f\ x - g\ x)\ A = (-1)\ \hat{}\ card\ A * prod\ (\lambda x.\ g\ x - f\ x)\ A$
⟨*proof*⟩

**lemma** *prod-power-distrib*: $prod\ f\ A\ \hat{}\ n = prod\ (\lambda x.\ (f\ x)\ \hat{}\ n)\ A$

**for** $f :: 'a \Rightarrow 'b::comm\text{-}semiring\text{-}1$
$\langle proof \rangle$

**lemma** *power-inject-exp'*:
**assumes** $a \neq 1\ a > (0 :: 'a :: linordered\text{-}semidom)$
**shows** $a\ \hat{}\ m = a\ \hat{}\ n \longleftrightarrow m = n$
$\langle proof \rangle$

**lemma** *power-sum*: $c\ \hat{}\ (\sum a{\in}A.\ f\ a) = (\prod a{\in}A.\ c\ \hat{}\ f\ a)$
$\langle proof \rangle$

**lemma** *prod-gen-delta*:
**fixes** $b :: 'b \Rightarrow 'a::comm\text{-}monoid\text{-}mult$
**assumes** *fin*: *finite S*
**shows** $prod\ (\lambda k.\ if\ k = a\ then\ b\ k\ else\ c)\ S =$
$(if\ a \in S\ then\ b\ a * c\ \hat{}\ (card\ S - 1)\ else\ c\ \hat{}\ card\ S)$
$\langle proof \rangle$

**lemma** *sum-image-le*:
**fixes** $g :: 'a \Rightarrow 'b::ordered\text{-}comm\text{-}monoid\text{-}add$
**assumes** $finite\ I\ \bigwedge i.\ i \in I \Longrightarrow 0 \leq g(f\ i)$
**shows** $sum\ g\ (f\ `\ I) \leq sum\ (g \circ f)\ I$
$\langle proof \rangle$

**lemma** *prod-add*:
**fixes** $f1\ f2 :: 'a \Rightarrow 'c :: comm\text{-}semiring\text{-}1$
**assumes** *finite*: *finite A*
**shows** $(\prod x{\in}A.\ f1\ x + f2\ x) = (\sum X{\in}Pow\ A.\ (\prod x{\in}X.\ f1\ x) * (\prod x{\in}A{-}X.\ f2\ x))$
$\langle proof \rangle$

**lemma** *prod-diff-conv-sum*:
**fixes** $f1\ f2 :: 'a \Rightarrow 'c :: comm\text{-}ring\text{-}1$
**assumes** *finite*: *finite A*
**shows** $(\prod x{\in}A.\ f1\ x - f2\ x) = (\sum X{\in}Pow\ A.\ (-1)\ \hat{}\ card\ X * (\prod x{\in}X.\ f2\ x) * (\prod x{\in}A{-}X.\ f1\ x))$
$\langle proof \rangle$

**lemma** *prod-diff-conv-sum'*:
**fixes** $f1\ f2 :: 'a \Rightarrow 'c :: comm\text{-}ring\text{-}1$
**assumes** *finite*: *finite A*
**shows** $(\prod x{\in}A.\ f1\ x - f2\ x) = (\sum X{\in}Pow\ A.\ (-1)\ \hat{}\ (card\ A - card\ X) * (\prod x{\in}X.\ f1\ x) * (\prod x{\in}A{-}X.\ f2\ x))$
$\langle proof \rangle$

**end**

# 49 Chain-complete partial orders and their fixpoints

**theory** *Complete-Partial-Order*
**imports** *Product-Type*
**begin**

## 49.1 Chains

A chain is a totally-ordered set. Chains are parameterized over the order for maximal flexibility, since type classes are not enough.

**definition** $chain :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
**where** $chain\ ord\ S \longleftrightarrow (\forall x{\in}S.\ \forall y{\in}S.\ ord\ x\ y \vee ord\ y\ x)$

**lemma** *chainI*:
**assumes** $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow ord\ x\ y \vee ord\ y\ x$
**shows** $chain\ ord\ S$
$\langle proof \rangle$

**lemma** *chainD*:
**assumes** $chain\ ord\ S$ **and** $x \in S$ **and** $y \in S$
**shows** $ord\ x\ y \vee ord\ y\ x$
$\langle proof \rangle$

**lemma** *chainE*:
**assumes** $chain\ ord\ S$ **and** $x \in S$ **and** $y \in S$
**obtains** $ord\ x\ y \mid ord\ y\ x$
$\langle proof \rangle$

**lemma** *chain-empty*: $chain\ ord\ \{\}$
$\langle proof \rangle$

**lemma** *chain-equality*: $chain\ (=)\ A \longleftrightarrow (\forall x{\in}A.\ \forall y{\in}A.\ x = y)$
$\langle proof \rangle$

**lemma** *chain-subset*: $chain\ ord\ A \Longrightarrow B \subseteq A \Longrightarrow chain\ ord\ B$
$\langle proof \rangle$

**lemma** *chain-imageI*:
**assumes** *chain*: $chain\ le\text{-}a\ Y$
**and** *mono*: $\bigwedge x\ y.\ x \in Y \Longrightarrow y \in Y \Longrightarrow le\text{-}a\ x\ y \Longrightarrow le\text{-}b\ (f\ x)\ (f\ y)$
**shows** $chain\ le\text{-}b\ (f\ `\ Y)$
$\langle proof \rangle$

## 49.2 Chain-complete partial orders

A *ccpo* has a least upper bound for any chain. In particular, the empty set is a chain, so every *ccpo* must have a bottom element.

**class** *ccpo* = *order* + *Sup* +
  **assumes** *ccpo-Sup-upper*: $chain\ (\leq)\ A \Longrightarrow x \in A \Longrightarrow x \leq Sup\ A$
  **assumes** *ccpo-Sup-least*: $chain\ (\leq)\ A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow x \leq z) \Longrightarrow Sup\ A \leq z$
**begin**

**lemma** *chain-singleton*: *Complete-Partial-Order.chain* $(\leq)\ \{x\}$
  ⟨*proof*⟩

**lemma** *ccpo-Sup-singleton* [*simp*]: $\bigsqcup\{x\} = x$
  ⟨*proof*⟩

## 49.3 Transfinite iteration of a function

**context notes** [[*inductive-internals*]]
**begin**

**inductive-set** *iterates* :: $('a \Rightarrow 'a) \Rightarrow 'a\ set$
  **for** $f :: 'a \Rightarrow 'a$
  **where**
    *step*: $x \in iterates\ f \Longrightarrow f\ x \in iterates\ f$
  | *Sup*: $chain\ (\leq)\ M \Longrightarrow \forall x{\in}M.\ x \in iterates\ f \Longrightarrow Sup\ M \in iterates\ f$

**end**

**lemma** *iterates-le-f*: $x \in iterates\ f \Longrightarrow monotone\ (\leq)\ (\leq)\ f \Longrightarrow x \leq f\ x$
  ⟨*proof*⟩

**lemma** *chain-iterates*:
  **assumes** *f*: $monotone\ (\leq)\ (\leq)\ f$
  **shows** $chain\ (\leq)\ (iterates\ f)$ (**is** *chain - ?C*)
⟨*proof*⟩

**lemma** *bot-in-iterates*: $Sup\ \{\} \in iterates\ f$
  ⟨*proof*⟩

## 49.4 Fixpoint combinator

**definition** *fixp* :: $('a \Rightarrow 'a) \Rightarrow 'a$
  **where** $fixp\ f = Sup\ (iterates\ f)$

**lemma** *iterates-fixp*:
  **assumes** *f*: $monotone\ (\leq)\ (\leq)\ f$
  **shows** $fixp\ f \in iterates\ f$
  ⟨*proof*⟩

**lemma** *fixp-unfold*:
  **assumes** *f*: $monotone\ (\leq)\ (\leq)\ f$
  **shows** $fixp\ f = f\ (fixp\ f)$
⟨*proof*⟩

**lemma** *fixp-lowerbound*:
**assumes** *f*: *monotone* $(\leq)$ $(\leq)$ *f*
**and** *z*: *f z* $\leq$ *z*
**shows** *fixp f* $\leq$ *z*
$\langle proof \rangle$

**end**

## 49.5 Fixpoint induction

$\langle ML \rangle$

**definition** *admissible* :: $('a\ set \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$
**where** *admissible lub ord P* $\longleftrightarrow$ $(\forall A.\ chain\ ord\ A \longrightarrow A \neq \{\} \longrightarrow (\forall x{\in}A.\ P\ x) \longrightarrow P\ (lub\ A))$

**lemma** *admissibleI*:
**assumes** $\bigwedge A.\ chain\ ord\ A \Longrightarrow A \neq \{\} \Longrightarrow \forall x{\in}A.\ P\ x \Longrightarrow P\ (lub\ A)$
**shows** *ccpo.admissible lub ord P*
$\langle proof \rangle$

**lemma** *admissibleD*:
**assumes** *ccpo.admissible lub ord P*
**assumes** *chain ord A*
**assumes** $A \neq \{\}$
**assumes** $\bigwedge x.\ x \in A \Longrightarrow P\ x$
**shows** $P\ (lub\ A)$
$\langle proof \rangle$

$\langle ML \rangle$

**lemma** (**in** *ccpo*) *fixp-induct*:
**assumes** *adm*: *ccpo.admissible Sup* $(\leq)$ *P*
**assumes** *mono*: *monotone* $(\leq)$ $(\leq)$ *f*
**assumes** *bot*: $P\ (Sup\ \{\})$
**assumes** *step*: $\bigwedge x.\ P\ x \Longrightarrow P\ (f\ x)$
**shows** $P\ (fixp\ f)$
$\langle proof \rangle$

**lemma** *admissible-True*: *ccpo.admissible lub ord* $(\lambda x.\ True)$
$\langle proof \rangle$

**lemma** *admissible-const*: *ccpo.admissible lub ord* $(\lambda x.\ t)$
$\langle proof \rangle$

**lemma** *admissible-conj*:
**assumes** *ccpo.admissible lub ord* $(\lambda x.\ P\ x)$

**assumes** *ccpo.admissible lub ord* $(\lambda x.\ Q\ x)$
**shows** *ccpo.admissible lub ord* $(\lambda x.\ P\ x \wedge Q\ x)$
⟨*proof*⟩

**lemma** *admissible-all*:
**assumes** $\bigwedge y.$ *ccpo.admissible lub ord* $(\lambda x.\ P\ x\ y)$
**shows** *ccpo.admissible lub ord* $(\lambda x.\ \forall\, y.\ P\ x\ y)$
⟨*proof*⟩

**lemma** *admissible-ball*:
**assumes** $\bigwedge y.\ y \in A \Longrightarrow$ *ccpo.admissible lub ord* $(\lambda x.\ P\ x\ y)$
**shows** *ccpo.admissible lub ord* $(\lambda x.\ \forall\, y{\in}A.\ P\ x\ y)$
⟨*proof*⟩

**lemma** *chain-compr*: *chain ord A* $\Longrightarrow$ *chain ord* $\{x \in A.\ P\ x\}$
⟨*proof*⟩

**context** *ccpo*
**begin**

**lemma** *admissible-disj*:
**fixes** $P\ Q :: 'a \Rightarrow bool$
**assumes** *P*: *ccpo.admissible Sup* $(\leq)$ $(\lambda x.\ P\ x)$
**assumes** *Q*: *ccpo.admissible Sup* $(\leq)$ $(\lambda x.\ Q\ x)$
**shows** *ccpo.admissible Sup* $(\leq)$ $(\lambda x.\ P\ x \vee Q\ x)$
⟨*proof*⟩

**end**

**instance** *complete-lattice* $\subseteq$ *ccpo*
⟨*proof*⟩

**lemma** *lfp-eq-fixp*:
**assumes** *mono*: *mono f*
**shows** *lfp f* = *fixp f*
⟨*proof*⟩

**hide-const** (**open**) *iterates fixp*

**end**

# 50 Datatype option

**theory** *Option*
**imports** *Lifting*
**begin**

**datatype** *'a option* =
*None*

| *Some* (*the*: $'a$)

**datatype-compat** *option*

**lemma** [*case-names None Some*, *cases type*: *option*]:
— for backward compatibility – names of variables differ
$(y = None \Longrightarrow P) \Longrightarrow (\bigwedge a.\ y = Some\ a \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** [*case-names None Some*, *induct type*: *option*]:
— for backward compatibility – names of variables differ
$P\ None \Longrightarrow (\bigwedge option.\ P\ (Some\ option)) \Longrightarrow P\ option$
⟨*proof*⟩

Compatibility:

⟨*ML*⟩
**lemmas** *inducts* = *option.induct*
**lemmas** *cases* = *option.case*
⟨*ML*⟩

**lemma** *not-None-eq* [*iff*]: $x \neq None \longleftrightarrow (\exists\, y.\ x = Some\ y)$
⟨*proof*⟩

**lemma** *not-Some-eq* [*iff*]: $(\forall\, y.\ x \neq Some\ y) \longleftrightarrow x = None$
⟨*proof*⟩

**lemma** *comp-the-Some*[*simp*]: *the o Some* = *id*
⟨*proof*⟩

Although it may appear that both of these equalities are helpful only when applied to assumptions, in practice it seems better to give them the uniform iff attribute.

**lemma** *inj-Some* [*simp*]: *inj-on Some A*
⟨*proof*⟩

**lemma** *case-optionE*:
**assumes** *c*: (*case x of None* ⇒ *P* | *Some y* ⇒ *Q y*)
**obtains**
(*None*) *x* = *None* **and** *P*
| (*Some*) *y* **where** *x* = *Some y* **and** *Q y*
⟨*proof*⟩

**lemma** *split-option-all*: $(\forall\, x.\ P\ x) \longleftrightarrow P\ None \wedge (\forall\, x.\ P\ (Some\ x))$
⟨*proof*⟩

**lemma** *split-option-ex*: $(\exists\, x.\ P\ x) \longleftrightarrow P\ None \vee (\exists\, x.\ P\ (Some\ x))$
⟨*proof*⟩

**lemma** *UNIV-option-conv*: *UNIV* = *insert None* (*range Some*)

⟨proof⟩

**lemma** *rel-option-None1* [*simp*]: *rel-option P None x* $\longleftrightarrow$ *x* = *None*
⟨proof⟩

**lemma** *rel-option-None2* [*simp*]: *rel-option P x None* $\longleftrightarrow$ *x* = *None*
⟨proof⟩

**lemma** *option-rel-Some1*: *rel-option A* (*Some x*) *y* $\longleftrightarrow$ ($\exists$ *y'*. *y* = *Some y'* $\wedge$ *A x y'*)
⟨proof⟩

**lemma** *option-rel-Some2*: *rel-option A x* (*Some y*) $\longleftrightarrow$ ($\exists$ *x'*. *x* = *Some x'* $\wedge$ *A x' y*)
⟨proof⟩

**lemma** *rel-option-inf*: *inf* (*rel-option A*) (*rel-option B*) = *rel-option* (*inf A B*)
(**is** *?lhs* = *?rhs*)
⟨proof⟩

**lemma** *rel-option-reflI*:
($\bigwedge$*x*. *x* $\in$ *set-option y* $\Longrightarrow$ *P x x*) $\Longrightarrow$ *rel-option P y y*
⟨proof⟩

### 50.0.1 Operations

**lemma** *ospec* [*dest*]: ($\forall$ *x*$\in$*set-option A*. *P x*) $\Longrightarrow$ *A* = *Some x* $\Longrightarrow$ *P x*
⟨proof⟩

⟨ML⟩

**lemma** *elem-set* [*iff*]: (*x* $\in$ *set-option xo*) = (*xo* = *Some x*)
⟨proof⟩

**lemma** *set-empty-eq* [*simp*]: (*set-option xo* = {}) = (*xo* = *None*)
⟨proof⟩

**lemma** *map-option-case*: *map-option f y* = (*case y of None* $\Rightarrow$ *None* | *Some x* $\Rightarrow$ *Some* (*f x*))
⟨proof⟩

**lemma** *map-option-is-None* [*iff*]: (*map-option f opt* = *None*) = (*opt* = *None*)
⟨proof⟩

**lemma** *None-eq-map-option-iff* [*iff*]: *None* = *map-option f x* $\longleftrightarrow$ *x* = *None*
⟨proof⟩

**lemma** *map-option-eq-Some* [*iff*]: (*map-option f xo* = *Some y*) = ($\exists$ *z*. *xo* = *Some z* $\wedge$ *f z* = *y*)

$\langle proof \rangle$

**lemma** *map-option-o-case-sum* [*simp*]:
$map\text{-}option\ f \circ case\text{-}sum\ g\ h = case\text{-}sum\ (map\text{-}option\ f \circ g)\ (map\text{-}option\ f \circ h)$
$\langle proof \rangle$

**lemma** *map-option-cong*: $x = y \Longrightarrow (\bigwedge a.\ y = Some\ a \Longrightarrow f\ a = g\ a) \Longrightarrow map\text{-}option\ f\ x = map\text{-}option\ g\ y$
$\langle proof \rangle$

**lemma** *map-option-idI*: $(\bigwedge y.\ y \in set\text{-}option\ x \Longrightarrow f\ y = y) \Longrightarrow map\text{-}option\ f\ x = x$
$\langle proof \rangle$

**functor** *map-option*: *map-option*
$\langle proof \rangle$

**lemma** *case-map-option* [*simp*]: $case\text{-}option\ g\ h\ (map\text{-}option\ f\ x) = case\text{-}option\ g\ (h \circ f)\ x$
$\langle proof \rangle$

**lemma** *None-notin-image-Some* [*simp*]: $None \notin Some\ `\ A$
$\langle proof \rangle$

**lemma** *notin-range-Some*: $x \notin range\ Some \longleftrightarrow x = None$
$\langle proof \rangle$

**lemma** *rel-option-iff*:
$rel\text{-}option\ R\ x\ y = (case\ (x,\ y)\ of\ (None,\ None) \Rightarrow True$
$\mid (Some\ x,\ Some\ y) \Rightarrow R\ x\ y$
$\mid \text{-} \Rightarrow False)$
$\langle proof \rangle$

**definition** *combine-options* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ option \Rightarrow 'a\ option \Rightarrow 'a\ option$
**where** $combine\text{-}options\ f\ x\ y =$
$(case\ x\ of\ None \Rightarrow y \mid Some\ x \Rightarrow (case\ y\ of\ None \Rightarrow Some\ x \mid Some\ y \Rightarrow Some\ (f\ x\ y)))$

**lemma** *combine-options-simps* [*simp*]:
$combine\text{-}options\ f\ None\ y = y$
$combine\text{-}options\ f\ x\ None = x$
$combine\text{-}options\ f\ (Some\ a)\ (Some\ b) = Some\ (f\ a\ b)$
$\langle proof \rangle$

**lemma** *combine-options-cases* [*case-names None1 None2 Some*]:
$(x = None \Longrightarrow P\ x\ y) \Longrightarrow (y = None \Longrightarrow P\ x\ y) \Longrightarrow$
$(\bigwedge a\ b.\ x = Some\ a \Longrightarrow y = Some\ b \Longrightarrow P\ x\ y) \Longrightarrow P\ x\ y$

⟨*proof*⟩

**lemma** *combine-options-commute*:
$(\bigwedge x\ y.\ f\ x\ y = f\ y\ x) \Longrightarrow combine\text{-}options\ f\ x\ y = combine\text{-}options\ f\ y\ x$
⟨*proof*⟩

**lemma** *combine-options-assoc*:
$(\bigwedge x\ y\ z.\ f\ (f\ x\ y)\ z = f\ x\ (f\ y\ z)) \Longrightarrow$
$combine\text{-}options\ f\ (combine\text{-}options\ f\ x\ y)\ z =$
$combine\text{-}options\ f\ x\ (combine\text{-}options\ f\ y\ z)$
⟨*proof*⟩

**lemma** *combine-options-left-commute*:
$(\bigwedge x\ y.\ f\ x\ y = f\ y\ x) \Longrightarrow (\bigwedge x\ y\ z.\ f\ (f\ x\ y)\ z = f\ x\ (f\ y\ z)) \Longrightarrow$
$combine\text{-}options\ f\ y\ (combine\text{-}options\ f\ x\ z) =$
$combine\text{-}options\ f\ x\ (combine\text{-}options\ f\ y\ z)$
⟨*proof*⟩

**lemmas** *combine-options-ac* =
*combine-options-commute combine-options-assoc combine-options-left-commute*

**context**
**begin**

**qualified definition** *is-none* :: *'a option* ⇒ *bool*
**where** [*code-post*]: $is\text{-}none\ x \longleftrightarrow x = None$

**lemma** *is-none-simps* [*simp*]:
*is-none None*
$\neg\ is\text{-}none\ (Some\ x)$
⟨*proof*⟩

**lemma** *is-none-code* [*code*]:
$is\text{-}none\ None \longleftrightarrow True$
$is\text{-}none\ (Some\ x) \longleftrightarrow False$
⟨*proof*⟩

**lemma** *rel-option-unfold*:
$rel\text{-}option\ R\ x\ y \longleftrightarrow$
$(is\text{-}none\ x \longleftrightarrow is\text{-}none\ y) \wedge (\neg\ is\text{-}none\ x \longrightarrow \neg\ is\text{-}none\ y \longrightarrow R\ (the\ x)\ (the\ y))$
⟨*proof*⟩

**lemma** *rel-optionI*:
$[\![\ is\text{-}none\ x \longleftrightarrow is\text{-}none\ y;\ [\![\ \neg\ is\text{-}none\ x;\ \neg\ is\text{-}none\ y\ ]\!] \Longrightarrow P\ (the\ x)\ (the\ y)\ ]\!]$
$\Longrightarrow rel\text{-}option\ P\ x\ y$
⟨*proof*⟩

**lemma** *is-none-map-option* [*simp*]: $\textit{is-none}\ (\textit{map-option}\ f\ x) \longleftrightarrow \textit{is-none}\ x$
⟨*proof*⟩

**lemma** *the-map-option*: $\neg\ \textit{is-none}\ x \Longrightarrow \textit{the}\ (\textit{map-option}\ f\ x) = f\ (\textit{the}\ x)$
⟨*proof*⟩ **primrec** *bind* :: $'a\ \textit{option} \Rightarrow ('a \Rightarrow 'b\ \textit{option}) \Rightarrow 'b\ \textit{option}$
**where**
*bind-lzero*: $\textit{bind None}\ f = \textit{None}$
| *bind-lunit*: $\textit{bind}\ (\textit{Some}\ x)\ f = f\ x$

**lemma** *is-none-bind*: $\textit{is-none}\ (\textit{bind}\ f\ g) \longleftrightarrow \textit{is-none}\ f \lor \textit{is-none}\ (g\ (\textit{the}\ f))$
⟨*proof*⟩

**lemma** *bind-runit*[*simp*]: $\textit{bind}\ x\ \textit{Some} = x$
⟨*proof*⟩

**lemma** *bind-assoc*[*simp*]: $\textit{bind}\ (\textit{bind}\ x\ f)\ g = \textit{bind}\ x\ (\lambda y.\ \textit{bind}\ (f\ y)\ g)$
⟨*proof*⟩

**lemma** *bind-rzero*[*simp*]: $\textit{bind}\ x\ (\lambda x.\ \textit{None}) = \textit{None}$
⟨*proof*⟩ **lemma** *bind-cong*: $x = y \Longrightarrow (\bigwedge a.\ y = \textit{Some}\ a \Longrightarrow f\ a = g\ a) \Longrightarrow \textit{bind}\ x\ f = \textit{bind}\ y\ g$
⟨*proof*⟩

**lemma** *bind-split*: $P\ (\textit{bind}\ m\ f) \longleftrightarrow (m = \textit{None} \longrightarrow P\ \textit{None}) \land (\forall v.\ m = \textit{Some}\ v \longrightarrow P\ (f\ v))$
⟨*proof*⟩

**lemma** *bind-split-asm*: $P\ (\textit{bind}\ m\ f) \longleftrightarrow \neg\ (m = \textit{None} \land \neg\ P\ \textit{None} \lor (\exists x.\ m = \textit{Some}\ x \land \neg\ P\ (f\ x)))$
⟨*proof*⟩

**lemmas** *bind-splits* = *bind-split bind-split-asm*

**lemma** *bind-eq-Some-conv*: $\textit{bind}\ f\ g = \textit{Some}\ x \longleftrightarrow (\exists y.\ f = \textit{Some}\ y \land g\ y = \textit{Some}\ x)$
⟨*proof*⟩

**lemma** *bind-eq-None-conv*: $\textit{Option.bind}\ a\ f = \textit{None} \longleftrightarrow a = \textit{None} \lor f\ (\textit{the}\ a) = \textit{None}$
⟨*proof*⟩

**lemma** *map-option-bind*: $\textit{map-option}\ f\ (\textit{bind}\ x\ g) = \textit{bind}\ x\ (\textit{map-option}\ f \circ g)$
⟨*proof*⟩

**lemma** *bind-option-cong*:
$[\![\ x = y;\ \bigwedge z.\ z \in \textit{set-option}\ y \Longrightarrow f\ z = g\ z\ ]\!] \Longrightarrow \textit{bind}\ x\ f = \textit{bind}\ y\ g$
⟨*proof*⟩

**lemma** *bind-option-cong-simp*:

$\llbracket x = y; \bigwedge z.\ z \in$ *set-option* $y$ =simp=> $f\ z = g\ z \rrbracket \Longrightarrow$ *bind* $x\ f =$ *bind* $y\ g$
⟨*proof*⟩

**lemma** *bind-option-cong-code*: $x = y \Longrightarrow$ *bind* $x\ f =$ *bind* $y\ f$
⟨*proof*⟩

**lemma** *bind-map-option*: *bind* (*map-option* $f\ x$) $g$ = *bind* $x$ ($g \circ f$)
⟨*proof*⟩

**lemma** *set-bind-option* [*simp*]: *set-option* (*bind* $x\ f$) = ($\bigcup$ ((*set-option* $\circ f$) ` *set-option* $x$))
⟨*proof*⟩

**lemma** *map-conv-bind-option*: *map-option* $f\ x$ = *Option.bind* $x$ (*Some* $\circ f$)
⟨*proof*⟩

**end**

⟨*ML*⟩

**context**
**begin**

**qualified definition** *these* :: *'a option set* $\Rightarrow$ *'a set*
**where** *these* $A$ = *the* ` $\{x \in A.\ x \neq$ *None*$\}$

**qualified lemma** *these-eq* [*code*]:
‹*these* $A$ = *the* ` (*Set.filter* (*Not* $\circ$ *Option.is-none*) $A$)›
⟨*proof*⟩ **lemma** *these-unfold*:
‹*these* $A = \{x.\ \exists\, y \in A.\ y =$ *Some* $x\}$›
⟨*proof*⟩

**lemma** *these-empty* [*simp*]: *these* {} = {}
⟨*proof*⟩

**lemma** *these-insert-None* [*simp*]: *these* (*insert None* $A$) = *these* $A$
⟨*proof*⟩

**lemma** *these-insert-Some* [*simp*]: *these* (*insert* (*Some* $x$) $A$) = *insert* $x$ (*these* $A$)
⟨*proof*⟩

**lemma** *in-these-eq*: $x \in$ *these* $A \longleftrightarrow$ *Some* $x \in A$
⟨*proof*⟩

**lemma** *these-image-Some-eq* [*simp*]: *these* (*Some* ` $A$) = $A$
⟨*proof*⟩

**lemma** *Some-image-these-eq*: *Some* ` *these* $A = \{x{\in}A.\ x \neq$ *None*$\}$

⟨*proof*⟩

**lemma** *these-empty-eq*: *these* $B = \{\} \longleftrightarrow B = \{\} \vee B = \{None\}$
⟨*proof*⟩

**lemma** *these-not-empty-eq*: *these* $B \neq \{\} \longleftrightarrow B \neq \{\} \wedge B \neq \{None\}$
⟨*proof*⟩ **definition** *image-filter* :: $('a \Rightarrow 'b\ option) \Rightarrow 'a\ set \Rightarrow 'b\ set$
**where** *image-filter-eq*: *image-filter f A* = *these* (*f* ‘ *A*)

**end**

**lemma** *finite-range-Some*: *finite* (*range* (*Some* :: $'a \Rightarrow 'a$ *option*)) = *finite* (*UNIV* :: $'a$ *set*)
⟨*proof*⟩

## 50.1 Transfer rules for the Transfer package

**context includes** *lifting-syntax*
**begin**

**lemma** *option-bind-transfer* [*transfer-rule*]:
(*rel-option A* ===> (*A* ===> *rel-option B*) ===> *rel-option B*)
*Option.bind Option.bind*
⟨*proof*⟩

**lemma** *pred-option-parametric* [*transfer-rule*]:
((*A* ===> (=)) ===> *rel-option A* ===> (=)) *pred-option pred-option*
⟨*proof*⟩

**end**

### 50.1.1 Interaction with finite sets

**lemma** *finite-option-UNIV* [*simp*]:
*finite* (*UNIV* :: $'a$ *option set*) = *finite* (*UNIV* :: $'a$ *set*)
⟨*proof*⟩

**instance** *option* :: (*finite*) *finite*
⟨*proof*⟩

### 50.1.2 Code generator setup

**lemma** *equal-None-code-unfold* [*code-unfold*]:
*HOL.equal x None* ⟷ *Option.is-none x*
*HOL.equal None* = *Option.is-none*
⟨*proof*⟩

**code-printing**
**type-constructor** *option* ⇀
(*SML*) - *option*

**and** (*OCaml*) - *option*
**and** (*Haskell*) *Maybe* -
**and** (*Scala*) !*Option*[(-)]
| **constant** *None* $\rightharpoonup$
(*SML*) *NONE*
**and** (*OCaml*) *None*
**and** (*Haskell*) *Nothing*
**and** (*Scala*) !*None*
| **constant** *Some* $\rightharpoonup$
(*SML*) *SOME*
**and** (*OCaml*) *Some* -
**and** (*Haskell*) *Just*
**and** (*Scala*) *Some*
| **class-instance** *option* :: *equal* $\rightharpoonup$
(*Haskell*) $-$
| **constant** *HOL.equal* :: $'a$ *option* $\Rightarrow$ $'a$ *option* $\Rightarrow$ *bool* $\rightharpoonup$
(*Haskell*) **infix** *4* ==

**code-reserved**
(*SML*) *option NONE SOME*
**and** (*OCaml*) *option None Some*
**and** (*Scala*) *Option None Some*

**end**

# 51 Partial Function Definitions

**theory** *Partial-Function*
**imports** *Complete-Partial-Order Option*
**keywords** *partial-function* :: *thy-defn*
**begin**

**named-theorems** *partial-function-mono monotonicity rules for partial function definitions*
⟨*ML*⟩

**lemma** (**in** *ccpo*) *in-chain-finite*:
**assumes** *Complete-Partial-Order.chain* $(\leq)$ $A$ *finite* $A$ $A \neq \{\}$
**shows** $\bigsqcup A \in A$
⟨*proof*⟩

**lemma** (**in** *ccpo*) *admissible-chfin*:
($\forall S.$ *Complete-Partial-Order.chain* $(\leq)$ $S \longrightarrow$ *finite* $S$)
$\Longrightarrow$ *ccpo.admissible Sup* $(\leq)$ $P$
⟨*proof*⟩

## 51.1 Axiomatic setup

This techical locale constains the requirements for function definitions with ccpo fixed points.

**definition** *fun-ord ord f g* $\longleftrightarrow$ ($\forall$ *x*. *ord* (*f x*) (*g x*))
**definition** *fun-lub L A* = ($\lambda$*x*. *L* {*y*. $\exists$*f*$\in$*A*. *y* = *f x*})
**definition** *img-ord f ord* = ($\lambda$*x y*. *ord* (*f x*) (*f y*))
**definition** *img-lub f g Lub* = ($\lambda$*A*. *g* (*Lub* (*f* ‘ *A*)))

**lemma** *chain-fun*:
  **assumes** *A*: *chain* (*fun-ord ord*) *A*
  **shows** *chain ord* {*y*. $\exists$*f*$\in$*A*. *y* = *f a*} (**is** *chain ord ?C*)
$\langle$*proof*$\rangle$

**lemma** *call-mono*[*partial-function-mono*]: *monotone* (*fun-ord ord*) *ord* ($\lambda$*f*. *f t*)
$\langle$*proof*$\rangle$

**lemma** *let-mono*[*partial-function-mono*]:
  ($\bigwedge$*x*. *monotone orda ordb* ($\lambda$*f*. *b f x*))
  $\Longrightarrow$ *monotone orda ordb* ($\lambda$*f*. *Let t* (*b f*))
$\langle$*proof*$\rangle$

**lemma** *if-mono*[*partial-function-mono*]: *monotone orda ordb F*
$\Longrightarrow$ *monotone orda ordb G*
$\Longrightarrow$ *monotone orda ordb* ($\lambda$*f*. *if c then F f else G f*)
$\langle$*proof*$\rangle$

**definition** *mk-less R* = ($\lambda$*x y*. *R x y* $\wedge$ $\neg$ *R y x*)

**locale** *partial-function-definitions* =
  **fixes** *leq* :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  **fixes** *lub* :: *'a set* $\Rightarrow$ *'a*
  **assumes** *leq-refl*: *leq x x*
  **assumes** *leq-trans*: *leq x y* $\Longrightarrow$ *leq y z* $\Longrightarrow$ *leq x z*
  **assumes** *leq-antisym*: *leq x y* $\Longrightarrow$ *leq y x* $\Longrightarrow$ *x* = *y*
  **assumes** *lub-upper*: *chain leq A* $\Longrightarrow$ *x* $\in$ *A* $\Longrightarrow$ *leq x* (*lub A*)
  **assumes** *lub-least*: *chain leq A* $\Longrightarrow$ ($\bigwedge$*x*. *x* $\in$ *A* $\Longrightarrow$ *leq x z*) $\Longrightarrow$ *leq* (*lub A*) *z*

**lemma** *partial-function-lift*:
  **assumes** *partial-function-definitions ord lb*
  **shows** *partial-function-definitions* (*fun-ord ord*) (*fun-lub lb*) (**is** *partial-function-definitions ?ordf ?lubf*)
$\langle$*proof*$\rangle$

**lemma** *ccpo*: **assumes** *partial-function-definitions ord lb*
  **shows** *class.ccpo lb ord* (*mk-less ord*)
$\langle$*proof*$\rangle$

**lemma** *partial-function-image*:

**assumes** *partial-function-definitions ord Lub*
**assumes** *inj*: $\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y$
**assumes** *inv*: $\bigwedge x.\ f\ (g\ x) = x$
**shows** *partial-function-definitions* (*img-ord f ord*) (*img-lub f g Lub*)
⟨*proof*⟩

**context** *partial-function-definitions*
**begin**

**abbreviation** *le-fun* ≡ *fun-ord leq*
**abbreviation** *lub-fun* ≡ *fun-lub lub*
**abbreviation** *fixp-fun* ≡ *ccpo.fixp lub-fun le-fun*
**abbreviation** *mono-body* ≡ *monotone le-fun leq*
**abbreviation** *admissible* ≡ *ccpo.admissible lub-fun le-fun*

Interpret manually, to avoid flooding everything with facts about orders

**lemma** *ccpo*: *class.ccpo lub-fun le-fun* (*mk-less le-fun*)
⟨*proof*⟩

The crucial fixed-point theorem

**lemma** *mono-body-fixp*:
$(\bigwedge x.\ \textit{mono-body}\ (\lambda f.\ F\ f\ x)) \Longrightarrow \textit{fixp-fun}\ F = F\ (\textit{fixp-fun}\ F)$
⟨*proof*⟩

Version with curry/uncurry combinators, to be used by package

**lemma** *fixp-rule-uc*:
**fixes** $F :: 'c \Rightarrow 'c$ **and**
$U :: 'c \Rightarrow 'b \Rightarrow 'a$ **and**
$C :: ('b \Rightarrow 'a) \Rightarrow 'c$
**assumes** *mono*: $\bigwedge x.\ \textit{mono-body}\ (\lambda f.\ U\ (F\ (C\ f))\ x)$
**assumes** *eq*: $f \equiv C\ (\textit{fixp-fun}\ (\lambda f.\ U\ (F\ (C\ f))))$
**assumes** *inverse*: $\bigwedge f.\ C\ (U\ f) = f$
**shows** $f = F\ f$
⟨*proof*⟩

Fixpoint induction rule

**lemma** *fixp-induct-uc*:
**fixes** $F :: 'c \Rightarrow 'c$
**and** $U :: 'c \Rightarrow 'b \Rightarrow 'a$
**and** $C :: ('b \Rightarrow 'a) \Rightarrow 'c$
**and** $P :: ('b \Rightarrow 'a) \Rightarrow bool$
**assumes** *mono*: $\bigwedge x.\ \textit{mono-body}\ (\lambda f.\ U\ (F\ (C\ f))\ x)$
**and** *eq*: $f \equiv C\ (\textit{fixp-fun}\ (\lambda f.\ U\ (F\ (C\ f))))$
**and** *inverse*: $\bigwedge f.\ U\ (C\ f) = f$
**and** *adm*: *ccpo.admissible lub-fun le-fun P*
**and** *bot*: $P\ (\lambda\text{-}.\ lub\ \{\})$
**and** *step*: $\bigwedge f.\ P\ (U\ f) \Longrightarrow P\ (U\ (F\ f))$
**shows** $P\ (U\ f)$

⟨*proof*⟩

Rules for *monotone le-fun leq*:

**lemma** *const-mono*[*partial-function-mono*]: *monotone ord leq* $(\lambda f.\ c)$
⟨*proof*⟩

**end**

## 51.2 Flat interpretation: tailrec and option

**definition**
*flat-ord* $b\ x\ y \longleftrightarrow x = b \lor x = y$

**definition**
*flat-lub* $b\ A = (\mathit{if}\ A \subseteq \{b\}\ \mathit{then}\ b\ \mathit{else}\ (\mathit{THE}\ x.\ x \in A - \{b\}))$

**lemma** *flat-interpretation*:
*partial-function-definitions* (*flat-ord* $b$) (*flat-lub* $b$)
⟨*proof*⟩

**lemma** *flat-ordI*: $(x \neq a \Longrightarrow x = y) \Longrightarrow$ *flat-ord* $a\ x\ y$
⟨*proof*⟩

**lemma** *flat-ord-antisym*: $[\![$ *flat-ord* $a\ x\ y$; *flat-ord* $a\ y\ x$ $]\!] \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *antisymp-flat-ord*: *antisymp* (*flat-ord* $a$)
⟨*proof*⟩

**interpretation** *tailrec*:
*partial-function-definitions flat-ord undefined flat-lub undefined*
**rewrites** *flat-lub undefined* $\{\} \equiv$ *undefined*
⟨*proof*⟩

**interpretation** *option*:
*partial-function-definitions flat-ord None flat-lub None*
**rewrites** *flat-lub None* $\{\} \equiv$ *None*
⟨*proof*⟩

**abbreviation** *tailrec-ord* $\equiv$ *flat-ord undefined*
**abbreviation** *mono-tailrec* $\equiv$ *monotone* (*fun-ord tailrec-ord*) *tailrec-ord*

**lemma** *tailrec-admissible*:
*ccpo.admissible* (*fun-lub* (*flat-lub* $c$)) (*fun-ord* (*flat-ord* $c$))
$(\lambda a.\ \forall x.\ a\ x \neq c \longrightarrow P\ x\ (a\ x))$
⟨*proof*⟩

**lemma** *fixp-induct-tailrec*:

**fixes** $F :: 'c \Rightarrow 'c$ **and**
$U :: 'c \Rightarrow 'b \Rightarrow 'a$ **and**
$C :: ('b \Rightarrow 'a) \Rightarrow 'c$ **and**
$P :: 'b \Rightarrow 'a \Rightarrow bool$ **and**
$x :: 'b$
**assumes** *mono*: $\bigwedge x.$ *monotone* (*fun-ord* (*flat-ord* $c$)) (*flat-ord* $c$) ($\lambda f.\ U\ (F\ (C\ f))\ x$)
**assumes** *eq*: $f \equiv C$ (*ccpo.fixp* (*fun-lub* (*flat-lub* $c$)) (*fun-ord* (*flat-ord* $c$)) ($\lambda f.\ U\ (F\ (C\ f))$))
**assumes** *inverse2*: $\bigwedge f.\ U\ (C\ f) = f$
**assumes** *step*: $\bigwedge f\ x\ y.\ (\bigwedge x\ y.\ U\ f\ x = y \Longrightarrow y \neq c \Longrightarrow P\ x\ y) \Longrightarrow U\ (F\ f)\ x = y \Longrightarrow y \neq c \Longrightarrow P\ x\ y$
**assumes** *result*: $U\ f\ x = y$
**assumes** *defined*: $y \neq c$
**shows** $P\ x\ y$
⟨*proof*⟩

**lemma** *admissible-image*:
**assumes** *pfun*: *partial-function-definitions le lub*
**assumes** *adm*: *ccpo.admissible lub le* ($P \circ g$)
**assumes** *inj*: $\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y$
**assumes** *inv*: $\bigwedge x.\ f\ (g\ x) = x$
**shows** *ccpo.admissible* (*img-lub f g lub*) (*img-ord f le*) $P$
⟨*proof*⟩

**lemma** *admissible-fun*:
**assumes** *pfun*: *partial-function-definitions le lub*
**assumes** *adm*: $\bigwedge x.$ *ccpo.admissible lub le* ($Q\ x$)
**shows** *ccpo.admissible* (*fun-lub lub*) (*fun-ord le*) ($\lambda f.\ \forall x.\ Q\ x\ (f\ x)$)
⟨*proof*⟩

**abbreviation** *option-ord* $\equiv$ *flat-ord None*
**abbreviation** *mono-option* $\equiv$ *monotone* (*fun-ord option-ord*) *option-ord*

**lemma** *bind-mono*[*partial-function-mono*]:
**assumes** *mf*: *mono-option* $B$ **and** *mg*: $\bigwedge y.$ *mono-option* ($\lambda f.\ C\ y\ f$)
**shows** *mono-option* ($\lambda f.$ *Option.bind* ($B\ f$) ($\lambda y.\ C\ y\ f$))
⟨*proof*⟩

**lemma** *flat-lub-in-chain*:
**assumes** *ch*: *chain* (*flat-ord* $b$) $A$
**assumes** *lub*: *flat-lub* $b\ A = a$
**shows** $a = b \lor a \in A$
⟨*proof*⟩

**lemma** *option-admissible*: *option.admissible* (%($f$::$'a \Rightarrow 'b$ *option*).
($\forall x\ y.\ f\ x = Some\ y \longrightarrow P\ x\ y$))
⟨*proof*⟩

**lemma** *fixp-induct-option*:
**fixes** $F :: 'c \Rightarrow 'c$ **and**
$U :: 'c \Rightarrow 'b \Rightarrow 'a\ option$ **and**
$C :: ('b \Rightarrow 'a\ option) \Rightarrow 'c$ **and**
$P :: 'b \Rightarrow 'a \Rightarrow bool$
**assumes** *mono*: $\bigwedge x.\ mono\text{-}option\ (\lambda f.\ U\ (F\ (C\ f))\ x)$
**assumes** *eq*: $f \equiv C\ (ccpo.fixp\ (fun\text{-}lub\ (flat\text{-}lub\ None))\ (fun\text{-}ord\ option\text{-}ord)\ (\lambda f.\ U\ (F\ (C\ f))))$
**assumes** *inverse2*: $\bigwedge f.\ U\ (C\ f) = f$
**assumes** *step*: $\bigwedge f\ x\ y.\ (\bigwedge x\ y.\ U\ f\ x = Some\ y \Longrightarrow P\ x\ y) \Longrightarrow U\ (F\ f)\ x = Some\ y \Longrightarrow P\ x\ y$
**assumes** *defined*: $U\ f\ x = Some\ y$
**shows** $P\ x\ y$
⟨*proof*⟩

⟨*ML*⟩

**hide-const** (**open**) *chain*

**end**

**theory** *Argo*
**imports** *HOL*
**begin**

⟨*ML*⟩

**end**

# 52 Reconstructing external resolution proofs for propositional logic

**theory** *SAT*
**imports** *Argo*
**begin**

⟨*ML*⟩

**end**

# 53 Function Definitions and Termination Proofs

**theory** *Fun-Def*
**imports** *Basic-BNF-LFPs Partial-Function SAT*
**keywords**
*function termination* :: *thy-goal-defn* **and**

*fun fun-cases* :: *thy-defn*
**begin**

## 53.1 Definitions with default value

**definition** *THE-default* :: $'a \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a$
**where** *THE-default* $d\ P = (if\ (\exists!x.\ P\ x)\ then\ (THE\ x.\ P\ x)\ else\ d)$

**lemma** *THE-defaultI'*: $\exists!x.\ P\ x \Longrightarrow P\ (THE\text{-}default\ d\ P)$
⟨*proof*⟩

**lemma** *THE-default1-equality*: $\exists!x.\ P\ x \Longrightarrow P\ a \Longrightarrow THE\text{-}default\ d\ P = a$
⟨*proof*⟩

**lemma** *THE-default-none*: $\neg\ (\exists!x.\ P\ x) \Longrightarrow THE\text{-}default\ d\ P = d$
⟨*proof*⟩

**lemma** *fundef-ex1-existence*:
**assumes** *f-def*: $f \equiv (\lambda x::'a.\ THE\text{-}default\ (d\ x)\ (\lambda y.\ G\ x\ y))$
**assumes** *ex1*: $\exists!y.\ G\ x\ y$
**shows** $G\ x\ (f\ x)$
⟨*proof*⟩

**lemma** *fundef-ex1-uniqueness*:
**assumes** *f-def*: $f \equiv (\lambda x::'a.\ THE\text{-}default\ (d\ x)\ (\lambda y.\ G\ x\ y))$
**assumes** *ex1*: $\exists!y.\ G\ x\ y$
**assumes** *elm*: $G\ x\ (h\ x)$
**shows** $h\ x = f\ x$
⟨*proof*⟩

**lemma** *fundef-ex1-iff*:
**assumes** *f-def*: $f \equiv (\lambda x::'a.\ THE\text{-}default\ (d\ x)\ (\lambda y.\ G\ x\ y))$
**assumes** *ex1*: $\exists!y.\ G\ x\ y$
**shows** $(G\ x\ y) = (f\ x = y)$
⟨*proof*⟩

**lemma** *fundef-default-value*:
**assumes** *f-def*: $f \equiv (\lambda x::'a.\ THE\text{-}default\ (d\ x)\ (\lambda y.\ G\ x\ y))$
**assumes** *graph*: $\bigwedge x\ y.\ G\ x\ y \Longrightarrow D\ x$
**assumes** $\neg\ D\ x$
**shows** $f\ x = d\ x$
⟨*proof*⟩

**definition** *in-rel-def*[*simp*]: *in-rel* $R\ x\ y \equiv (x,\ y) \in R$

**lemma** *wf-in-rel*: *wf* $R \Longrightarrow$ *wfp* (*in-rel* $R$)
⟨*proof*⟩

⟨*ML*⟩

## 53.2 Measure functions

**inductive** *is-measure* :: $('a \Rightarrow nat) \Rightarrow bool$
**where** *is-measure-trivial*: *is-measure f*

**named-theorems** *measure-function rules that guide the heuristic generation of measure functions*
⟨*ML*⟩

**lemma** *measure-size*[*measure-function*]: *is-measure size*
⟨*proof*⟩

**lemma** *measure-fst*[*measure-function*]: *is-measure f* $\Longrightarrow$ *is-measure* $(\lambda p.\ f\ (fst\ p))$
⟨*proof*⟩

**lemma** *measure-snd*[*measure-function*]: *is-measure f* $\Longrightarrow$ *is-measure* $(\lambda p.\ f\ (snd\ p))$
⟨*proof*⟩

⟨*ML*⟩

## 53.3 Congruence rules

**lemma** *let-cong* [*fundef-cong*]: $M = N \Longrightarrow (\bigwedge x.\ x = N \Longrightarrow f\ x = g\ x) \Longrightarrow Let\ M\ f = Let\ N\ g$
⟨*proof*⟩

**lemmas** [*fundef-cong*] =
*if-cong image-cong*
*bex-cong ball-cong imp-cong map-option-cong Option.bind-cong*

**lemma** *split-cong* [*fundef-cong*]:
$(\bigwedge x\ y.\ (x, y) = q \Longrightarrow f\ x\ y = g\ x\ y) \Longrightarrow p = q \Longrightarrow case\text{-}prod\ f\ p = case\text{-}prod\ g\ q$
⟨*proof*⟩

**lemma** *comp-cong* [*fundef-cong*]: $f\ (g\ x) = f'\ (g'\ x') \Longrightarrow (f \circ g)\ x = (f' \circ g')\ x'$
⟨*proof*⟩

## 53.4 Simp rules for termination proofs

**declare**
*trans-less-add1*[*termination-simp*]
*trans-less-add2*[*termination-simp*]
*trans-le-add1*[*termination-simp*]
*trans-le-add2*[*termination-simp*]
*less-imp-le-nat*[*termination-simp*]
*le-imp-less-Suc*[*termination-simp*]

**lemma** *size-prod-simp*[*termination-simp*]: *size-prod f g p* = *f* (*fst p*) + *g* (*snd p*) + *Suc 0*
⟨*proof*⟩

## 53.5 Decomposition

**lemma** *less-by-empty*: $A = \{\} \Longrightarrow A \subseteq B$
**and** *union-comp-emptyL*: $A\ O\ C = \{\} \Longrightarrow B\ O\ C = \{\} \Longrightarrow (A \cup B)\ O\ C = \{\}$
**and** *union-comp-emptyR*: $A\ O\ B = \{\} \Longrightarrow A\ O\ C = \{\} \Longrightarrow A\ O\ (B \cup C) = \{\}$
**and** *wf-no-loop*: $R\ O\ R = \{\} \Longrightarrow$ *wf R*
⟨*proof*⟩

## 53.6 Reduction pairs

**definition** *reduction-pair P* ⟷ *wf* (*fst P*) ∧ *fst P O snd P* ⊆ *fst P*

**lemma** *reduction-pairI*[*intro*]: *wf R* ⟹ *R O S* ⊆ *R* ⟹ *reduction-pair* (*R*, *S*)
⟨*proof*⟩

**lemma** *reduction-pair-lemma*:
**assumes** *rp*: *reduction-pair P*
**assumes** *R* ⊆ *fst P*
**assumes** *S* ⊆ *snd P*
**assumes** *wf S*
**shows** *wf* (*R* ∪ *S*)
⟨*proof*⟩

**definition** *rp-inv-image* = (λ(*R*,*S*) *f*. (*inv-image R f*, *inv-image S f*))

**lemma** *rp-inv-image-rp*: *reduction-pair P* ⟹ *reduction-pair* (*rp-inv-image P f*)
⟨*proof*⟩

## 53.7 Concrete orders for SCNP termination proofs

**definition** *pair-less* = *less-than* <∗*lex*∗> *less-than*
**definition** *pair-leq* = *pair-less*$^=$
**definition** *max-strict* = *max-ext pair-less*
**definition** *max-weak* = *max-ext pair-leq* ∪ {({}, {})}
**definition** *min-strict* = *min-ext pair-less*
**definition** *min-weak* = *min-ext pair-leq* ∪ {({}, {})}

**lemma** *wf-pair-less*[*simp*]: *wf pair-less*
⟨*proof*⟩

**lemma** *total-pair-less* [*iff*]: *total-on A pair-less* **and** *trans-pair-less* [*iff*]: *trans pair-less*
⟨*proof*⟩

Introduction rules for *pair-less*/*pair-leq*

**lemma** *pair-leqI1*: $a < b \Longrightarrow ((a, s), (b, t)) \in$ *pair-leq*
**and** *pair-leqI2*: $a \leq b \Longrightarrow s \leq t \Longrightarrow ((a, s), (b, t)) \in$ *pair-leq*
**and** *pair-lessI1*: $a < b \Longrightarrow ((a, s), (b, t)) \in$ *pair-less*
**and** *pair-lessI2*: $a \leq b \Longrightarrow s < t \Longrightarrow ((a, s), (b, t)) \in$ *pair-less*
⟨*proof*⟩

**lemma** *pair-less-iff1* [*simp*]: $((x,y), (x,z)) \in$ *pair-less* $\longleftrightarrow y<z$
⟨*proof*⟩

Introduction rules for max

**lemma** *smax-emptyI*: *finite* $Y \Longrightarrow Y \neq \{\} \Longrightarrow (\{\}, Y) \in$ *max-strict*
**and** *smax-insertI*:
$y \in Y \Longrightarrow (x, y) \in$ *pair-less* $\Longrightarrow (X, Y) \in$ *max-strict* $\Longrightarrow$ (*insert* $x$ $X$, $Y$) $\in$ *max-strict*
**and** *wmax-emptyI*: *finite* $X \Longrightarrow (\{\}, X) \in$ *max-weak*
**and** *wmax-insertI*:
$y \in YS \Longrightarrow (x, y) \in$ *pair-leq* $\Longrightarrow (XS, YS) \in$ *max-weak* $\Longrightarrow$ (*insert* $x$ $XS$, $YS$) $\in$ *max-weak*
⟨*proof*⟩

Introduction rules for min

**lemma** *smin-emptyI*: $X \neq \{\} \Longrightarrow (X, \{\}) \in$ *min-strict*
**and** *smin-insertI*:
$x \in XS \Longrightarrow (x, y) \in$ *pair-less* $\Longrightarrow (XS, YS) \in$ *min-strict* $\Longrightarrow$ ($XS$, *insert* $y$ $YS$) $\in$ *min-strict*
**and** *wmin-emptyI*: $(X, \{\}) \in$ *min-weak*
**and** *wmin-insertI*:
$x \in XS \Longrightarrow (x, y) \in$ *pair-leq* $\Longrightarrow (XS, YS) \in$ *min-weak* $\Longrightarrow$ ($XS$, *insert* $y$ $YS$) $\in$ *min-weak*
⟨*proof*⟩

Reduction Pairs.

**lemma** *max-ext-compat*:
**assumes** $R$ $O$ $S \subseteq R$
**shows** *max-ext* $R$ $O$ (*max-ext* $S \cup \{(\{\}, \{\})\}) \subseteq$ *max-ext* $R$
⟨*proof*⟩

**lemma** *max-rpair-set*: *reduction-pair* (*max-strict*, *max-weak*)
⟨*proof*⟩

**lemma** *min-ext-compat*:
**assumes** $R$ $O$ $S \subseteq R$
**shows** *min-ext* $R$ $O$ (*min-ext* $S \cup \{(\{\},\{\})\}) \subseteq$ *min-ext* $R$
⟨*proof*⟩

**lemma** *min-rpair-set*: *reduction-pair* (*min-strict*, *min-weak*)
⟨*proof*⟩

## 53.8 Yet more induction principles on the natural numbers

**lemma** *nat-descend-induct* [*case-names base descend*]:
  **fixes** $P$ :: $nat \Rightarrow bool$
  **assumes** $H1$: $\bigwedge k.\ k > n \Longrightarrow P\ k$
  **assumes** $H2$: $\bigwedge k.\ k \leq n \Longrightarrow (\bigwedge i.\ i > k \Longrightarrow P\ i) \Longrightarrow P\ k$
  **shows** $P\ m$
  ⟨*proof*⟩

**lemma** *induct-nat-012*[*case-names 0 1 ge2*]:
  $P\ 0 \Longrightarrow P\ (Suc\ 0) \Longrightarrow (\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ n) \Longrightarrow P\ (Suc\ (Suc\ n))) \Longrightarrow P\ n$
⟨*proof*⟩

## 53.9 Tool setup

⟨*ML*⟩

**end**

# 54 The Integers as Equivalence Classes over Pairs of Natural Numbers

**theory** *Int*
  **imports** *Quotient Groups-Big Fun-Def*
**begin**

## 54.1 Definition of integers as a quotient type

**definition** *intrel* :: $(nat \times nat) \Rightarrow (nat \times nat) \Rightarrow bool$
  **where** $intrel = (\lambda(x,\ y)\ (u,\ v).\ x + v = u + y)$

**lemma** *intrel-iff* [*simp*]: $intrel\ (x,\ y)\ (u,\ v) \longleftrightarrow x + v = u + y$
  ⟨*proof*⟩

**quotient-type** $int = nat \times nat\ /\ intrel$
  **morphisms** *Rep-Integ Abs-Integ*
⟨*proof*⟩

## 54.2 Integers form a commutative ring

**instantiation** *int* :: *comm-ring-1*
**begin**

**lift-definition** *zero-int* :: *int* **is** $(0,\ 0)$ ⟨*proof*⟩

**lift-definition** *one-int* :: *int* **is** $(1,\ 0)$ ⟨*proof*⟩

**lift-definition** *plus-int* :: $int \Rightarrow int \Rightarrow int$
  **is** $\lambda(x,\ y)\ (u,\ v).\ (x + u,\ y + v)$

⟨*proof*⟩

**lift-definition** *uminus-int* :: *int* ⇒ *int*
  **is** $\lambda(x, y). (y, x)$
  ⟨*proof*⟩

**lift-definition** *minus-int* :: *int* ⇒ *int* ⇒ *int*
  **is** $\lambda(x, y)\ (u, v). (x + v, y + u)$
  ⟨*proof*⟩

**lift-definition** *times-int* :: *int* ⇒ *int* ⇒ *int*
  **is** $\lambda(x, y)\ (u, v). (x*u + y*v, x*v + y*u)$
⟨*proof*⟩

**instance**
  ⟨*proof*⟩

**end**

**abbreviation** *int* :: *nat* ⇒ *int*
  **where** *int* ≡ *of-nat*

**lemma** *int-def*: *int n* = *Abs-Integ* (*n*, *0*)
  ⟨*proof*⟩

**lemma** *int-transfer* [*transfer-rule*]:
  **includes** *lifting-syntax*
  **shows** *rel-fun* (=) *pcr-int* (λ*n*. (*n*, *0*)) *int*
  ⟨*proof*⟩

**lemma** *int-diff-cases*: **obtains** (*diff*) *m n* **where** *z* = *int m* − *int n*
  ⟨*proof*⟩

## 54.3 Integers are totally ordered

**instantiation** *int* :: *linorder*
**begin**

**lift-definition** *less-eq-int* :: *int* ⇒ *int* ⇒ *bool*
  **is** $\lambda(x, y)\ (u, v).\ x + v \le u + y$
  ⟨*proof*⟩

**lift-definition** *less-int* :: *int* ⇒ *int* ⇒ *bool*
  **is** $\lambda(x, y)\ (u, v).\ x + v < u + y$
  ⟨*proof*⟩

**instance**
  ⟨*proof*⟩

**end**

**instantiation** *int* :: *distrib-lattice*
**begin**

**definition** $(inf :: int \Rightarrow int \Rightarrow int) = min$

**definition** $(sup :: int \Rightarrow int \Rightarrow int) = max$

**instance**
  ⟨*proof*⟩

**end**

## 54.4 Ordering properties of arithmetic operations

**instance** *int* :: *ordered-cancel-ab-semigroup-add*
⟨*proof*⟩

Strict Monotonicity of Multiplication.

Strict, in 1st argument; proof is by induction on $k > 0$.

**lemma** *zmult-zless-mono2-lemma*: $i < j \Longrightarrow 0 < k \Longrightarrow int\ k * i < int\ k * j$
  **for** $i\ j :: int$
⟨*proof*⟩

**lemma** *zero-le-imp-eq-int*:
  **assumes** $k \geq (0::int)$ **shows** $\exists\, n.\ k = int\ n$
⟨*proof*⟩

**lemma** *zero-less-imp-eq-int*:
  **assumes** $k > (0::int)$ **shows** $\exists\, n{>}0.\ k = int\ n$
⟨*proof*⟩

**lemma** *zmult-zless-mono2*: $i < j \Longrightarrow 0 < k \Longrightarrow k * i < k * j$
  **for** $i\ j\ k :: int$
  ⟨*proof*⟩

The integers form an ordered integral domain.

**instantiation** *int* :: *linordered-idom*
**begin**

**definition** *zabs-def*: $|i::int| = (if\ i < 0\ then\ -\ i\ else\ i)$

**definition** *zsgn-def*: $sgn\ (i::int) = (if\ i = 0\ then\ 0\ else\ if\ 0 < i\ then\ 1\ else\ -\ 1)$

**instance**
⟨*proof*⟩

**end**

**instance** *int* :: *discrete-linordered-semidom*
⟨*proof*⟩

**lemma** *zless-imp-add1-zle*: $w < z \Longrightarrow w + 1 \leq z$
**for** $w\ z$ :: *int*
⟨*proof*⟩

**lemma** *zless-iff-Suc-zadd*: $w < z \longleftrightarrow (\exists n.\ z = w + int\ (Suc\ n))$
**for** $w\ z$ :: *int*
⟨*proof*⟩

**lemma** *zabs-less-one-iff* [*simp*]: $|z| < 1 \longleftrightarrow z = 0$ (**is** *?lhs* ⟷ *?rhs*)
**for** $z$ :: *int*
⟨*proof*⟩

## 54.5 Embedding of the Integers into any *ring-1*: *of-int*

**context** *ring-1*
**begin**

**lift-definition** *of-int* :: *int* ⇒ *'a*
**is** $\lambda(i, j).\ of\text{-}nat\ i - of\text{-}nat\ j$
⟨*proof*⟩

**lemma** *of-int-0* [*simp*]: *of-int* $0 = 0$
⟨*proof*⟩

**lemma** *of-int-1* [*simp*]: *of-int* $1 = 1$
⟨*proof*⟩

**lemma** *of-int-add* [*simp*]: *of-int* $(w + z)$ = *of-int* $w$ + *of-int* $z$
⟨*proof*⟩

**lemma** *of-int-minus* [*simp*]: *of-int* $(- z)$ = $-$ (*of-int* $z$)
⟨*proof*⟩

**lemma** *of-int-diff* [*simp*]: *of-int* $(w - z)$ = *of-int* $w$ $-$ *of-int* $z$
⟨*proof*⟩

**lemma** *of-int-mult* [*simp*]: *of-int* $(w*z)$ = *of-int* $w$ $*$ *of-int* $z$
⟨*proof*⟩

**lemma** *mult-of-int-commute*: *of-int* $x * y = y *$ *of-int* $x$
⟨*proof*⟩

Collapse nested embeddings.

**lemma** *of-int-of-nat-eq* [*simp*]: *of-int* (*int* $n$) = *of-nat* $n$

⟨*proof*⟩

**lemma** *of-int-numeral* [*simp*, *code-post*]: *of-int* (*numeral k*) = *numeral k*
⟨*proof*⟩

**lemma** *of-int-neg-numeral* [*code-post*]: *of-int* (− *numeral k*) = − *numeral k*
⟨*proof*⟩

**lemma** *of-int-power* [*simp*]: *of-int* (*z* ^ *n*) = *of-int z* ^ *n*
⟨*proof*⟩

**lemma** *of-int-of-bool* [*simp*]:
*of-int* (*of-bool P*) = *of-bool P*
⟨*proof*⟩

**end**

**context** *ring-char-0*
**begin**

**lemma** *of-int-eq-iff* [*simp*]: *of-int w* = *of-int z* ⟷ *w* = *z*
⟨*proof*⟩

Special cases where either operand is zero.

**lemma** *of-int-eq-0-iff* [*simp*]: *of-int z* = *0* ⟷ *z* = *0*
⟨*proof*⟩

**lemma** *of-int-0-eq-iff* [*simp*]: *0* = *of-int z* ⟷ *z* = *0*
⟨*proof*⟩

**lemma** *of-int-eq-1-iff* [*iff*]: *of-int z* = *1* ⟷ *z* = *1*
⟨*proof*⟩

**lemma** *of-nat-of-int-iff*: *of-int i* = *of-nat n* ⟷ *i* = *of-nat n* *of-nat n* = *of-int i* ⟷ *i* = *of-nat n*
⟨*proof*⟩

**lemma** *numeral-power-eq-of-int-cancel-iff* [*simp*]:
*numeral x* ^ *n* = *of-int y* ⟷ *numeral x* ^ *n* = *y*
⟨*proof*⟩

**lemma** *of-int-eq-numeral-power-cancel-iff* [*simp*]:
*of-int y* = *numeral x* ^ *n* ⟷ *y* = *numeral x* ^ *n*
⟨*proof*⟩

**lemma** *neg-numeral-power-eq-of-int-cancel-iff* [*simp*]:
(− *numeral x*) ^ *n* = *of-int y* ⟷ (− *numeral x*) ^ *n* = *y*
⟨*proof*⟩

**lemma** *of-int-eq-neg-numeral-power-cancel-iff* [*simp*]:
*of-int* $y = (-$ *numeral* $x)$ ^ $n \longleftrightarrow y = (-$ *numeral* $x)$ ^ $n$
⟨*proof*⟩

**lemma** *of-int-eq-of-int-power-cancel-iff*[*simp*]: (*of-int* $b$) ^ $w$ = *of-int* $x \longleftrightarrow b$ ^ $w$ = $x$
⟨*proof*⟩

**lemma** *of-int-power-eq-of-int-cancel-iff*[*simp*]: *of-int* $x$ = (*of-int* $b$) ^ $w \longleftrightarrow x$ = $b$ ^ $w$
⟨*proof*⟩

**end**

**context** *linordered-idom*
**begin**

Every *linordered-idom* has characteristic zero.

**subclass** *ring-char-0* ⟨*proof*⟩

**lemma** *of-int-le-iff* [*simp*]: *of-int* $w \leq$ *of-int* $z \longleftrightarrow w \leq z$
⟨*proof*⟩

**lemma** *of-int-less-iff* [*simp*]: *of-int* $w <$ *of-int* $z \longleftrightarrow w < z$
⟨*proof*⟩

**lemma** *of-int-0-le-iff* [*simp*]: $0 \leq$ *of-int* $z \longleftrightarrow 0 \leq z$
⟨*proof*⟩

**lemma** *of-int-le-0-iff* [*simp*]: *of-int* $z \leq 0 \longleftrightarrow z \leq 0$
⟨*proof*⟩

**lemma** *of-int-0-less-iff* [*simp*]: $0 <$ *of-int* $z \longleftrightarrow 0 < z$
⟨*proof*⟩

**lemma** *of-int-less-0-iff* [*simp*]: *of-int* $z < 0 \longleftrightarrow z < 0$
⟨*proof*⟩

**lemma** *of-int-1-le-iff* [*simp*]: $1 \leq$ *of-int* $z \longleftrightarrow 1 \leq z$
⟨*proof*⟩

**lemma** *of-int-le-1-iff* [*simp*]: *of-int* $z \leq 1 \longleftrightarrow z \leq 1$
⟨*proof*⟩

**lemma** *of-int-1-less-iff* [*simp*]: $1 <$ *of-int* $z \longleftrightarrow 1 < z$
⟨*proof*⟩

**lemma** *of-int-less-1-iff* [*simp*]: *of-int* $z < 1 \longleftrightarrow z < 1$
⟨*proof*⟩

**lemma** *of-int-pos*: $z > 0 \Longrightarrow$ *of-int* $z > 0$
⟨*proof*⟩

**lemma** *of-int-nonneg*: $z \geq 0 \Longrightarrow$ *of-int* $z \geq 0$
⟨*proof*⟩

**lemma** *of-int-abs* [*simp*]: *of-int* $|x|$ = $|$*of-int* $x|$
⟨*proof*⟩

**lemma** *of-int-lessD*:
**assumes** $|$*of-int* $n| < x$
**shows** $n = 0 \vee x > 1$
⟨*proof*⟩

**lemma** *of-int-leD*:
**assumes** $|$*of-int* $n| \leq x$
**shows** $n = 0 \vee 1 \leq x$
⟨*proof*⟩

**lemma** *numeral-power-le-of-int-cancel-iff* [*simp*]:
*numeral* $x$ ^ $n \leq$ *of-int* $a \longleftrightarrow$ *numeral* $x$ ^ $n \leq a$
⟨*proof*⟩

**lemma** *of-int-le-numeral-power-cancel-iff* [*simp*]:
*of-int* $a \leq$ *numeral* $x$ ^ $n \longleftrightarrow a \leq$ *numeral* $x$ ^ $n$
⟨*proof*⟩

**lemma** *numeral-power-less-of-int-cancel-iff* [*simp*]:
*numeral* $x$ ^ $n <$ *of-int* $a \longleftrightarrow$ *numeral* $x$ ^ $n < a$
⟨*proof*⟩

**lemma** *of-int-less-numeral-power-cancel-iff* [*simp*]:
*of-int* $a <$ *numeral* $x$ ^ $n \longleftrightarrow a <$ *numeral* $x$ ^ $n$
⟨*proof*⟩

**lemma** *neg-numeral-power-le-of-int-cancel-iff* [*simp*]:
$(-$ *numeral* $x)$ ^ $n \leq$ *of-int* $a \longleftrightarrow (-$ *numeral* $x)$ ^ $n \leq a$
⟨*proof*⟩

**lemma** *of-int-le-neg-numeral-power-cancel-iff* [*simp*]:
*of-int* $a \leq (-$ *numeral* $x)$ ^ $n \longleftrightarrow a \leq (-$ *numeral* $x)$ ^ $n$
⟨*proof*⟩

**lemma** *neg-numeral-power-less-of-int-cancel-iff* [*simp*]:
$(-$ *numeral* $x)$ ^ $n <$ *of-int* $a \longleftrightarrow (-$ *numeral* $x)$ ^ $n < a$
⟨*proof*⟩

**lemma** *of-int-less-neg-numeral-power-cancel-iff* [*simp*]:

*of-int* $a < (-$ *numeral* $x)$ ^ $n \longleftrightarrow a < (-$ *numeral* $x$::*int*$)$ ^ $n$
⟨*proof*⟩

**lemma** *of-int-le-of-int-power-cancel-iff*[*simp*]: (*of-int* $b$) ^ $w \leq$ *of-int* $x \longleftrightarrow b$ ^ $w \leq x$
⟨*proof*⟩

**lemma** *of-int-power-le-of-int-cancel-iff*[*simp*]: *of-int* $x \leq$ (*of-int* $b$) ^ $w \longleftrightarrow x \leq b$ ^ $w$
⟨*proof*⟩

**lemma** *of-int-less-of-int-power-cancel-iff*[*simp*]: (*of-int* $b$) ^ $w <$ *of-int* $x \longleftrightarrow b$ ^ $w < x$
⟨*proof*⟩

**lemma** *of-int-power-less-of-int-cancel-iff*[*simp*]: *of-int* $x <$ (*of-int* $b$) ^ $w \longleftrightarrow x < b$ ^ $w$
⟨*proof*⟩

**lemma** *of-int-max*: *of-int* (*max* $x$ $y$) = *max* (*of-int* $x$) (*of-int* $y$)
⟨*proof*⟩

**lemma** *of-int-min*: *of-int* (*min* $x$ $y$) = *min* (*of-int* $x$) (*of-int* $y$)
⟨*proof*⟩

**end**

**context** *division-ring*
**begin**

**lemmas** *mult-inverse-of-int-commute* =
*mult-commute-imp-mult-inverse-commute*[*OF mult-of-int-commute*]

**end**

Comparisons involving *of-int*.

**lemma** *of-int-eq-numeral-iff* [*iff*]: *of-int* $z$ = (*numeral* $n$ :: *'a*::*ring-char-0*) $\longleftrightarrow z$ = *numeral* $n$
⟨*proof*⟩

**lemma** *of-int-le-numeral-iff* [*simp*]:
*of-int* $z \leq$ (*numeral* $n$ :: *'a*::*linordered-idom*) $\longleftrightarrow z \leq$ *numeral* $n$
⟨*proof*⟩

**lemma** *of-int-numeral-le-iff* [*simp*]:
(*numeral* $n$ :: *'a*::*linordered-idom*) $\leq$ *of-int* $z \longleftrightarrow$ *numeral* $n \leq z$
⟨*proof*⟩

**lemma** *of-int-less-numeral-iff* [*simp*]:

$of\text{-}int\ z < (numeral\ n :: 'a::linordered\text{-}idom) \longleftrightarrow z < numeral\ n$
⟨proof⟩

**lemma** *of-int-numeral-less-iff* [*simp*]:
$(numeral\ n :: 'a::linordered\text{-}idom) < of\text{-}int\ z \longleftrightarrow numeral\ n < z$
⟨proof⟩

**lemma** *of-nat-less-of-int-iff*: $(of\text{-}nat\ n::'a::linordered\text{-}idom) < of\text{-}int\ x \longleftrightarrow int\ n < x$
⟨proof⟩

**lemma** *of-int-eq-id* [*simp*]: $of\text{-}int = id$
⟨proof⟩

**instance** *int* :: *no-top*
⟨proof⟩

**instance** *int* :: *no-bot*
⟨proof⟩

## 54.6 Magnitude of an Integer, as a Natural Number: *nat*

**lift-definition** $nat :: int \Rightarrow nat$ **is** $\lambda(x, y).\ x - y$
⟨proof⟩

**lemma** *nat-int* [*simp*]: $nat\ (int\ n) = n$
⟨proof⟩

**lemma** *int-nat-eq* [*simp*]: $int\ (nat\ z) = (if\ 0 \leq z\ then\ z\ else\ 0)$
⟨proof⟩

**lemma** *nat-0-le*: $0 \leq z \Longrightarrow int\ (nat\ z) = z$
⟨proof⟩

**lemma** *nat-le-0* [*simp*]: $z \leq 0 \Longrightarrow nat\ z = 0$
⟨proof⟩

**lemma** *nat-le-eq-zle*: $0 < w \vee 0 \leq z \Longrightarrow nat\ w \leq nat\ z \longleftrightarrow w \leq z$
⟨proof⟩

An alternative condition is $0 \leq w$.

**lemma** *nat-mono-iff*: $0 < z \Longrightarrow nat\ w < nat\ z \longleftrightarrow w < z$
⟨proof⟩

**lemma** *nat-less-eq-zless*: $0 \leq w \Longrightarrow nat\ w < nat\ z \longleftrightarrow w < z$
⟨proof⟩

**lemma** *zless-nat-conj* [*simp*]: $nat\ w < nat\ z \longleftrightarrow 0 < z \wedge w < z$
⟨proof⟩

**lemma** *nonneg-int-cases*:
**assumes** $0 \leq k$
**obtains** $n$ **where** $k = int\ n$
⟨*proof*⟩

**lemma** *pos-int-cases*:
**assumes** $0 < k$
**obtains** $n$ **where** $k = int\ n$ **and** $n > 0$
⟨*proof*⟩

**lemma** *nonpos-int-cases*:
**assumes** $k \leq 0$
**obtains** $n$ **where** $k = -\ int\ n$
⟨*proof*⟩

**lemma** *neg-int-cases*:
**assumes** $k < 0$
**obtains** $n$ **where** $k = -\ int\ n$ **and** $n > 0$
⟨*proof*⟩

**lemma** *nat-eq-iff*: $nat\ w = m \longleftrightarrow (if\ 0 \leq w\ then\ w = int\ m\ else\ m = 0)$
⟨*proof*⟩

**lemma** *nat-eq-iff2*: $m = nat\ w \longleftrightarrow (if\ 0 \leq w\ then\ w = int\ m\ else\ m = 0)$
⟨*proof*⟩

**lemma** *nat-0* [*simp*]: $nat\ 0 = 0$
⟨*proof*⟩

**lemma** *nat-1* [*simp*]: $nat\ 1 = Suc\ 0$
⟨*proof*⟩

**lemma** *nat-numeral* [*simp*]: $nat\ (numeral\ k) = numeral\ k$
⟨*proof*⟩

**lemma** *nat-neg-numeral* [*simp*]: $nat\ (-\ numeral\ k) = 0$
⟨*proof*⟩

**lemma** *nat-2*: $nat\ 2 = Suc\ (Suc\ 0)$
⟨*proof*⟩

**lemma** *nat-less-iff*: $0 \leq w \Longrightarrow nat\ w < m \longleftrightarrow w < of\text{-}nat\ m$
⟨*proof*⟩

**lemma** *nat-le-iff*: $nat\ x \leq n \longleftrightarrow x \leq int\ n$
⟨*proof*⟩

**lemma** *nat-mono*: $x \leq y \Longrightarrow nat\ x \leq nat\ y$

⟨proof⟩

**lemma** *nat-0-iff* [*simp*]: $nat\ i = 0 \longleftrightarrow i \leq 0$
**for** $i :: int$
⟨proof⟩

**lemma** *int-eq-iff*: $of\text{-}nat\ m = z \longleftrightarrow m = nat\ z \wedge 0 \leq z$
⟨proof⟩

**lemma** *zero-less-nat-eq* [*simp*]: $0 < nat\ z \longleftrightarrow 0 < z$
⟨proof⟩

**lemma** *nat-add-distrib*: $0 \leq z \Longrightarrow 0 \leq z' \Longrightarrow nat\ (z + z') = nat\ z + nat\ z'$
⟨proof⟩

**lemma** *nat-diff-distrib'*: $0 \leq x \Longrightarrow 0 \leq y \Longrightarrow nat\ (x - y) = nat\ x - nat\ y$
⟨proof⟩

**lemma** *nat-diff-distrib*: $0 \leq z' \Longrightarrow z' \leq z \Longrightarrow nat\ (z - z') = nat\ z - nat\ z'$
⟨proof⟩

**lemma** *nat-zminus-int* [*simp*]: $nat\ (- int\ n) = 0$
⟨proof⟩

**lemma** *le-nat-iff*: $k \geq 0 \Longrightarrow n \leq nat\ k \longleftrightarrow int\ n \leq k$
⟨proof⟩

**lemma** *zless-nat-eq-int-zless*: $m < nat\ z \longleftrightarrow int\ m < z$
⟨proof⟩

**lemma** (**in** *ring-1*) *of-nat-nat* [*simp*]: $0 \leq z \Longrightarrow of\text{-}nat\ (nat\ z) = of\text{-}int\ z$
⟨proof⟩

**lemma** *diff-nat-numeral* [*simp*]: $(numeral\ v :: nat) - numeral\ v' = nat\ (numeral\ v - numeral\ v')$
⟨proof⟩

**lemma** *nat-abs-triangle-ineq*:
$nat\ |k + l| \leq nat\ |k| + nat\ |l|$
⟨proof⟩

**lemma** *nat-of-bool* [*simp*]:
$nat\ (of\text{-}bool\ P) = of\text{-}bool\ P$
⟨proof⟩

**lemma** *split-nat* [*linarith-split*]: $P\ (nat\ i) \longleftrightarrow ((\forall n.\ i = int\ n \longrightarrow P\ n) \wedge (i < 0 \longrightarrow P\ 0))$
(**is** $?P = (?L \wedge ?R)$)
**for** $i :: int$

⟨*proof*⟩

**lemma** *all-nat*: $(\forall x.\ P\ x) \longleftrightarrow (\forall x \geq 0.\ P\ (nat\ x))$
⟨*proof*⟩

**lemma** *ex-nat*: $(\exists x.\ P\ x) \longleftrightarrow (\exists x.\ 0 \leq x \land P\ (nat\ x))$
⟨*proof*⟩

For termination proofs:

**lemma** *measure-function-int*[*measure-function*]: *is-measure* $(nat \circ abs)$ ⟨*proof*⟩

## 54.7 Lemmas about the Function *of-nat* and Orderings

**lemma** *negative-zless-0*: $-\ (int\ (Suc\ n)) < (0 :: int)$
⟨*proof*⟩

**lemma** *negative-zless* [*iff*]: $-\ (int\ (Suc\ n)) < int\ m$
⟨*proof*⟩

**lemma** *negative-zle-0*: $-\ int\ n \leq 0$
⟨*proof*⟩

**lemma** *negative-zle* [*iff*]: $-\ int\ n \leq int\ m$
⟨*proof*⟩

**lemma** *not-zle-0-negative* [*simp*]: $\neg\ 0 \leq -\ int\ (Suc\ n)$
⟨*proof*⟩

**lemma** *int-zle-neg*: $int\ n \leq -\ int\ m \longleftrightarrow n = 0 \land m = 0$
⟨*proof*⟩

**lemma** *not-int-zless-negative* [*simp*]: $\neg\ int\ n < -\ int\ m$
⟨*proof*⟩

**lemma** *negative-eq-positive* [*simp*]: $-\ int\ n = of\text{-}nat\ m \longleftrightarrow n = 0 \land m = 0$
⟨*proof*⟩

**lemma** *zle-iff-zadd*: $w \leq z \longleftrightarrow (\exists n.\ z = w + int\ n)$
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

**lemma** *zadd-int-left*: $int\ m + (int\ n + z) = int\ (m + n) + z$
⟨*proof*⟩

**lemma** *negD*:
**assumes** $x < 0$ **shows** $\exists n.\ x = -\ (int\ (Suc\ n))$
⟨*proof*⟩

## 54.8 Cases and induction

Now we replace the case analysis rule by a more conventional one: whether an integer is negative or not.

This version is symmetric in the two subgoals.

**lemma** *int-cases2* [*case-names nonneg nonpos*, *cases type*: *int*]:
$(\bigwedge n.\ z = int\ n \Longrightarrow P) \Longrightarrow (\bigwedge n.\ z = -\ (int\ n) \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

This is the default, with a negative case.

**lemma** *int-cases* [*case-names nonneg neg*, *cases type*: *int*]:
**assumes** *pos*: $\bigwedge n.\ z = int\ n \Longrightarrow P$ **and** *neg*: $\bigwedge n.\ z = -\ (int\ (Suc\ n)) \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *int-cases3* [*case-names zero pos neg*]:
**fixes** $k :: int$
**assumes** $k = 0 \Longrightarrow P$ **and** $\bigwedge n.\ k = int\ n \Longrightarrow n > 0 \Longrightarrow P$
**and** $\bigwedge n.\ k = -\ int\ n \Longrightarrow n > 0 \Longrightarrow P$
**shows** $P$
⟨*proof*⟩

**lemma** *int-of-nat-induct* [*case-names nonneg neg*, *induct type*: *int*]:
$(\bigwedge n.\ P\ (int\ n)) \Longrightarrow (\bigwedge n.\ P\ (-\ (int\ (Suc\ n)))) \Longrightarrow P\ z$
⟨*proof*⟩

**lemma** *sgn-mult-dvd-iff* [*simp*]:
$sgn\ r * l\ dvd\ k \longleftrightarrow l\ dvd\ k \wedge (r = 0 \longrightarrow k = 0)$ **for** $k\ l\ r :: int$
⟨*proof*⟩

**lemma** *mult-sgn-dvd-iff* [*simp*]:
$l * sgn\ r\ dvd\ k \longleftrightarrow l\ dvd\ k \wedge (r = 0 \longrightarrow k = 0)$ **for** $k\ l\ r :: int$
⟨*proof*⟩

**lemma** *dvd-sgn-mult-iff* [*simp*]:
$l\ dvd\ sgn\ r * k \longleftrightarrow l\ dvd\ k \vee r = 0$ **for** $k\ l\ r :: int$
⟨*proof*⟩

**lemma** *dvd-mult-sgn-iff* [*simp*]:
$l\ dvd\ k * sgn\ r \longleftrightarrow l\ dvd\ k \vee r = 0$ **for** $k\ l\ r :: int$
⟨*proof*⟩

**lemma** *int-sgnE*:
**fixes** $k :: int$
**obtains** $n$ **and** $l$ **where** $k = sgn\ l * int\ n$
⟨*proof*⟩

### 54.8.1 Binary comparisons

Preliminaries

**lemma** *le-imp-0-less*:
**fixes** $z$ :: *int*
**assumes** *le*: $0 \le z$
**shows** $0 < 1 + z$
⟨*proof*⟩

**lemma** *odd-less-0-iff*: $1 + z + z < 0 \longleftrightarrow z < 0$
**for** $z$ :: *int*
⟨*proof*⟩

### 54.8.2 Comparisons, for Ordered Rings

**lemma** *odd-nonzero*: $1 + z + z \neq 0$
**for** $z$ :: *int*
⟨*proof*⟩

## 54.9 The Set of Integers

**context** *ring-1*
**begin**

**definition** *Ints* :: *'a set* (‹$\mathbb{Z}$›)
**where** $\mathbb{Z}$ = *range of-int*

**lemma** *Ints-of-int* [*simp*]: *of-int* $z \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-of-nat* [*simp*]: *of-nat* $n \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-0* [*simp*]: $0 \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-1* [*simp*]: $1 \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-numeral* [*simp*]: *numeral* $n \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-add* [*simp*]: $a \in \mathbb{Z} \Longrightarrow b \in \mathbb{Z} \Longrightarrow a + b \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-minus* [*simp*]: $a \in \mathbb{Z} \Longrightarrow -a \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *minus-in-Ints-iff* [*simp*]: $-x \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$

⟨*proof*⟩

**lemma** *Ints-diff* [*simp*]: $a \in \mathbb{Z} \Longrightarrow b \in \mathbb{Z} \Longrightarrow a - b \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-mult* [*simp*]: $a \in \mathbb{Z} \Longrightarrow b \in \mathbb{Z} \Longrightarrow a * b \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-power* [*simp*]: $a \in \mathbb{Z} \Longrightarrow a \,\widehat{}\, n \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-cases* [*cases set*: *Ints*]:
**assumes** $q \in \mathbb{Z}$
**obtains** (*of-int*) $z$ **where** $q = \textit{of-int}\ z$
⟨*proof*⟩

**lemma** *Ints-induct* [*case-names of-int*, *induct set*: *Ints*]:
$q \in \mathbb{Z} \Longrightarrow (\bigwedge z.\ P\ (\textit{of-int}\ z)) \Longrightarrow P\ q$
⟨*proof*⟩

**lemma** *Nats-subset-Ints*: $\mathbb{N} \subseteq \mathbb{Z}$
⟨*proof*⟩

**lemma** *Nats-altdef1*: $\mathbb{N} = \{\textit{of-int}\ n\ |n.\ n \geq 0\}$
⟨*proof*⟩

**end**

**lemma** *Ints-sum* [*intro*]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Z}) \Longrightarrow \textit{sum}\ f\ A \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-prod* [*intro*]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Z}) \Longrightarrow \textit{prod}\ f\ A \in \mathbb{Z}$
⟨*proof*⟩

**lemma** (**in** *linordered-idom*) *Ints-abs* [*simp*]:
**shows** $a \in \mathbb{Z} \Longrightarrow \textit{abs}\ a \in \mathbb{Z}$
⟨*proof*⟩

**lemma** (**in** *linordered-idom*) *Nats-altdef2*: $\mathbb{N} = \{n \in \mathbb{Z}.\ n \geq 0\}$
⟨*proof*⟩

**lemma** (**in** *idom-divide*) *of-int-divide-in-Ints*:
$\textit{of-int}\ a\ \textit{div}\ \textit{of-int}\ b \in \mathbb{Z}$ **if** $b\ \textit{dvd}\ a$
⟨*proof*⟩

The premise involving $\mathbb{Z}$ prevents $a = 1\ /\ (2::'a)$.

**lemma** *Ints-double-eq-0-iff*:
**fixes** $a :: 'a::\textit{ring-char-0}$
**assumes** *in-Ints*: $a \in \mathbb{Z}$

**shows** $a + a = 0 \longleftrightarrow a = 0$
(**is** $?lhs \longleftrightarrow ?rhs$)
⟨*proof*⟩

**lemma** *Ints-odd-nonzero*:
**fixes** $a :: 'a::ring\text{-}char\text{-}0$
**assumes** *in-Ints*: $a \in \mathbb{Z}$
**shows** $1 + a + a \neq 0$
⟨*proof*⟩

**lemma** *Nats-numeral* [*simp*]: *numeral* $w \in \mathbb{N}$
⟨*proof*⟩

**lemma** *Ints-odd-less-0*:
**fixes** $a :: 'a::linordered\text{-}idom$
**assumes** *in-Ints*: $a \in \mathbb{Z}$
**shows** $1 + a + a < 0 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *add-in-Ints-iff-left* [*simp*]: $x \in \mathbb{Z} \Longrightarrow x + y \in \mathbb{Z} \longleftrightarrow y \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *add-in-Ints-iff-right* [*simp*]: $y \in \mathbb{Z} \Longrightarrow x + y \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *diff-in-Ints-iff-left* [*simp*]: $x \in \mathbb{Z} \Longrightarrow x - y \in \mathbb{Z} \longleftrightarrow y \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *diff-in-Ints-iff-right* [*simp*]: $y \in \mathbb{Z} \Longrightarrow x - y \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemmas** [*simp*] = *minus-in-Ints-iff*

**lemma** *fraction-not-in-Ints*:
**assumes** $\neg(n\ dvd\ m)$ $n \neq 0$
**shows** *of-int* $m$ / *of-int* $n \notin (\mathbb{Z} :: 'a :: \{division\text{-}ring,ring\text{-}char\text{-}0\}\ set)$
⟨*proof*⟩

**lemma** *of-int-div-of-int-in-Ints-iff*:
(*of-int* $n$ / *of-int* $m :: 'a :: \{division\text{-}ring,ring\text{-}char\text{-}0\}) \in \mathbb{Z} \longleftrightarrow m = 0 \lor m\ dvd$
$n$
⟨*proof*⟩

**lemma** *fraction-numeral-not-in-Ints* [*simp*]:
**assumes** $\neg$(*numeral* $b$ :: *int*) *dvd numeral* $a$
**shows** *numeral* $a$ / *numeral* $b \notin (\mathbb{Z} :: 'a :: \{division\text{-}ring,\ ring\text{-}char\text{-}0\}\ set)$
⟨*proof*⟩

## 54.10 *sum* and *prod*

**context** *semiring-1*
**begin**

**lemma** *of-nat-sum* [*simp*]:
$of\text{-}nat\ (sum\ f\ A) = (\sum x{\in}A.\ of\text{-}nat\ (f\ x))$
⟨*proof*⟩

**end**

**context** *ring-1*
**begin**

**lemma** *of-int-sum* [*simp*]:
$of\text{-}int\ (sum\ f\ A) = (\sum x{\in}A.\ of\text{-}int\ (f\ x))$
⟨*proof*⟩

**end**

**context** *comm-semiring-1*
**begin**

**lemma** *of-nat-prod* [*simp*]:
$of\text{-}nat\ (prod\ f\ A) = (\prod x{\in}A.\ of\text{-}nat\ (f\ x))$
⟨*proof*⟩

**end**

**context** *comm-ring-1*
**begin**

**lemma** *of-int-prod* [*simp*]:
$of\text{-}int\ (prod\ f\ A) = (\prod x{\in}A.\ of\text{-}int\ (f\ x))$
⟨*proof*⟩

**end**

## 54.11 Setting up simplification procedures

⟨*ML*⟩

## 54.12 More Inequality Reasoning

**lemma** *zless-add1-eq*: $w < z + 1 \longleftrightarrow w < z \lor w = z$
**for** $w\ z :: int$
⟨*proof*⟩

**lemma** *add1-zle-eq*: $w + 1 \le z \longleftrightarrow w < z$
**for** $w\ z :: int$

⟨*proof*⟩

**lemma** *zle-diff1-eq* [*simp*]: $w \leq z - 1 \longleftrightarrow w < z$
**for** $w$ $z$ :: *int*
⟨*proof*⟩

**lemma** *zle-add1-eq-le* [*simp*]: $w < z + 1 \longleftrightarrow w \leq z$
**for** $w$ $z$ :: *int*
⟨*proof*⟩

**lemma** *int-one-le-iff-zero-less*: $1 \leq z \longleftrightarrow 0 < z$
**for** $z$ :: *int*
⟨*proof*⟩

**lemma** *Ints-nonzero-abs-ge1*:
**fixes** $x$:: $'a$ :: *linordered-idom*
**assumes** $x \in \mathit{Ints}$ $x \neq 0$
**shows** $1 \leq \mathit{abs}\ x$
⟨*proof*⟩

**lemma** *Ints-nonzero-abs-less1*:
**fixes** $x$:: $'a$ :: *linordered-idom*
**shows** $[\![x \in \mathit{Ints};\ \mathit{abs}\ x < 1]\!] \Longrightarrow x = 0$
⟨*proof*⟩

**lemma** *Ints-eq-abs-less1*:
**fixes** $x$:: $'a$ :: *linordered-idom*
**shows** $[\![x \in \mathit{Ints};\ y \in \mathit{Ints}]\!] \Longrightarrow x = y \longleftrightarrow \mathit{abs}\ (x-y) < 1$
⟨*proof*⟩

## 54.13 The functions *nat* and *int*

Simplify the term $w + - z$.

**lemma** *one-less-nat-eq* [*simp*]: $\mathit{Suc}\ 0 < \mathit{nat}\ z \longleftrightarrow 1 < z$
⟨*proof*⟩

**lemma** *int-eq-iff-numeral* [*simp*]:
$\mathit{int}\ m = \mathit{numeral}\ v \longleftrightarrow m = \mathit{numeral}\ v$
⟨*proof*⟩

**lemma** *nat-abs-int-diff*:
$\mathit{nat}\ |\mathit{int}\ a - \mathit{int}\ b| = (\mathit{if}\ a \leq b\ \mathit{then}\ b - a\ \mathit{else}\ a - b)$
⟨*proof*⟩

**lemma** *nat-int-add*: $\mathit{nat}\ (\mathit{int}\ a + \mathit{int}\ b) = a + b$
⟨*proof*⟩

**context** *ring-1*
**begin**

**lemma** *of-int-of-nat* [*nitpick-simp*]:
*of-int* $k$ = (*if* $k < 0$ *then* − *of-nat* (*nat* (− $k$)) *else of-nat* (*nat* $k$))
⟨*proof*⟩

**end**

**lemma** *transfer-rule-of-int*:
**includes** *lifting-syntax*
**fixes** $R$ :: ′*a*::*ring-1* ⇒ ′*b*::*ring-1* ⇒ *bool*
**assumes** [*transfer-rule*]: $R$ *0 0* $R$ *1 1*
($R$ ===> $R$ ===> $R$) (+) (+)
($R$ ===> $R$) *uminus uminus*
**shows** ((=) ===> $R$) *of-int of-int*
⟨*proof*⟩

**lemma** *nat-mult-distrib*:
**fixes** $z$ $z'$ :: *int*
**assumes** $0 \leq z$
**shows** *nat* ($z * z'$) = *nat* $z$ ∗ *nat* $z'$
⟨*proof*⟩

**lemma** *nat-mult-distrib-neg*:
**assumes** $z \leq$ (*0*::*int*) **shows** *nat* ($z * z'$) = *nat* (− $z$) ∗ *nat* (− $z'$) (**is** *?L* = *?R*)
⟨*proof*⟩

**lemma** *nat-abs-mult-distrib*: *nat* $|w * z|$ = *nat* $|w|$ ∗ *nat* $|z|$
⟨*proof*⟩

**lemma** *int-in-range-abs* [*simp*]: *int* $n$ ∈ *range abs*
⟨*proof*⟩

**lemma** *range-abs-Nats* [*simp*]: *range abs* = (ℕ :: *int set*)
⟨*proof*⟩

**lemma** *Suc-nat-eq-nat-zadd1*: $0 \leq z \Longrightarrow$ *Suc* (*nat* $z$) = *nat* ($1 + z$)
**for** $z$ :: *int*
⟨*proof*⟩

**lemma** *diff-nat-eq-if*:
*nat* $z$ − *nat* $z'$ =
(*if* $z' < 0$ *then nat* $z$
*else*
*let* $d = z - z'$
*in if* $d < 0$ *then 0 else nat* $d$)
⟨*proof*⟩

**lemma** *nat-numeral-diff-1* [*simp*]: *numeral* $v$ − (*1*::*nat*) = *nat* (*numeral* $v$ − *1*)
⟨*proof*⟩

## 54.14 Induction principles for int

Well-founded segments of the integers.

**definition** *int-ge-less-than* :: $int \Rightarrow (int \times int)\ set$
 **where** *int-ge-less-than* $d = \{(z', z).\ d \leq z' \wedge z' < z\}$

**lemma** *wf-int-ge-less-than*: $wf\ (int\text{-}ge\text{-}less\text{-}than\ d)$
⟨*proof*⟩

This variant looks odd, but is typical of the relations suggested by Rank-Finder.

**definition** *int-ge-less-than2* :: $int \Rightarrow (int \times int)\ set$
 **where** *int-ge-less-than2* $d = \{(z',z).\ d \leq z \wedge z' < z\}$

**lemma** *wf-int-ge-less-than2*: $wf\ (int\text{-}ge\text{-}less\text{-}than2\ d)$
⟨*proof*⟩

**theorem** *int-ge-induct* [*case-names base step*, *induct set*: *int*]:
 **fixes** $i :: int$
 **assumes** *ge*: $k \leq i$
  **and** *base*: $P\ k$
  **and** *step*: $\bigwedge i.\ k \leq i \Longrightarrow P\ i \Longrightarrow P\ (i + 1)$
 **shows** $P\ i$
⟨*proof*⟩

**theorem** *int-gr-induct* [*case-names base step*, *induct set*: *int*]:
 **fixes** $i\ k :: int$
 **assumes** $k < i\ P\ (k + 1)\ \bigwedge i.\ k < i \Longrightarrow P\ i \Longrightarrow P\ (i + 1)$
 **shows** $P\ i$
⟨*proof*⟩

**theorem** *int-le-induct* [*consumes 1*, *case-names base step*]:
 **fixes** $i\ k :: int$
 **assumes** *le*: $i \leq k$
  **and** *base*: $P\ k$
  **and** *step*: $\bigwedge i.\ i \leq k \Longrightarrow P\ i \Longrightarrow P\ (i - 1)$
 **shows** $P\ i$
⟨*proof*⟩

**theorem** *int-less-induct* [*consumes 1*, *case-names base step*]:
 **fixes** $i\ k :: int$
 **assumes** $i < k\ P\ (k - 1)\ \bigwedge i.\ i < k \Longrightarrow P\ i \Longrightarrow P\ (i - 1)$
 **shows** $P\ i$
⟨*proof*⟩

**theorem** *int-induct* [*case-names base step1 step2*]:
 **fixes** $k :: int$

**assumes** *base*: $P\ k$
**and** *step1*: $\bigwedge i.\ k \le i \Longrightarrow P\ i \Longrightarrow P\ (i + 1)$
**and** *step2*: $\bigwedge i.\ k \ge i \Longrightarrow P\ i \Longrightarrow P\ (i - 1)$
**shows** $P\ i$
⟨*proof*⟩

## 54.15 Intermediate value theorems

**lemma** *nat-ivt-aux*:
$[\![\forall i<n.\ |f\ (Suc\ i) - f\ i| \le 1;\ f\ 0 \le k;\ k \le f\ n]\!] \Longrightarrow \exists i \le n.\ f\ i = k$
**for** $m\ n :: nat$ **and** $k :: int$
⟨*proof*⟩

**lemma** *nat-intermed-int-val*:
**fixes** $m\ n :: nat$ **and** $k :: int$
**assumes** $\forall i.\ m \le i \wedge i < n \longrightarrow |f\ (Suc\ i) - f\ i| \le 1\ m \le n\ f\ m \le k\ k \le f\ n$
**shows** $\exists i.\ m \le i \wedge i \le n \wedge f\ i = k$
⟨*proof*⟩

**lemma** *nat0-intermed-int-val*:
$\exists i\le n.\ f\ i = k$
**if** $\forall i<n.\ |f\ (i + 1) - f\ i| \le 1\ f\ 0 \le k\ k \le f\ n$
**for** $n :: nat$ **and** $k :: int$
⟨*proof*⟩

## 54.16 Products and 1, by T. M. Rasmussen

**lemma** *abs-zmult-eq-1*:
**fixes** $m\ n :: int$
**assumes** *mn*: $|m * n| = 1$
**shows** $|m| = 1$
⟨*proof*⟩

**lemma** *pos-zmult-eq-1-iff-lemma*: $m * n = 1 \Longrightarrow m = 1 \vee m = -\ 1$
**for** $m\ n :: int$
⟨*proof*⟩

**lemma** *pos-zmult-eq-1-iff*:
**fixes** $m\ n :: int$
**assumes** $0 < m$
**shows** $m * n = 1 \longleftrightarrow m = 1 \wedge n = 1$
⟨*proof*⟩

**lemma** *zmult-eq-1-iff*: $m * n = 1 \longleftrightarrow (m = 1 \wedge n = 1) \vee (m = -\ 1 \wedge n = -\ 1)$ (**is** $?L = ?R$)
**for** $m\ n :: int$
⟨*proof*⟩

**lemma** *zmult-eq-neg1-iff*: $a * b = (-1 :: int) \longleftrightarrow a = 1 \wedge b = -1 \vee a = -1 \wedge b = 1$

⟨*proof*⟩

**lemma** *infinite-UNIV-int* [*simp*]: $\neg$ *finite* (*UNIV*::*int set*)
⟨*proof*⟩

## 54.17 The divides relation

**lemma** *zdvd-antisym-nonneg*: $0 \leq m \Longrightarrow 0 \leq n \Longrightarrow m\ dvd\ n \Longrightarrow n\ dvd\ m \Longrightarrow m = n$
**for** $m\ n$ :: *int*
⟨*proof*⟩

**lemma** *zdvd-antisym-abs*:
**fixes** $a\ b$ :: *int*
**assumes** $a\ dvd\ b$ **and** $b\ dvd\ a$
**shows** $|a| = |b|$
⟨*proof*⟩

**lemma** *zdvd-zdiffD*: $k\ dvd\ m - n \Longrightarrow k\ dvd\ n \Longrightarrow k\ dvd\ m$
**for** $k\ m\ n$ :: *int*
⟨*proof*⟩

**lemma** *zdvd-reduce*: $k\ dvd\ n + k * m \longleftrightarrow k\ dvd\ n$
**for** $k\ m\ n$ :: *int*
⟨*proof*⟩

**lemma** *dvd-imp-le-int*:
**fixes** $d\ i$ :: *int*
**assumes** $i \neq 0$ **and** $d\ dvd\ i$
**shows** $|d| \leq |i|$
⟨*proof*⟩

**lemma** *zdvd-not-zless*:
**fixes** $m\ n$ :: *int*
**assumes** $0 < m$ **and** $m < n$
**shows** $\neg\ n\ dvd\ m$
⟨*proof*⟩

**lemma** *zdvd-mult-cancel*:
**fixes** $k\ m\ n$ :: *int*
**assumes** *d*: $k * m\ dvd\ k * n$
**and** $k \neq 0$
**shows** $m\ dvd\ n$
⟨*proof*⟩

**lemma** *int-dvd-int-iff* [*simp*]:
*int* $m$ *dvd int* $n \longleftrightarrow m\ dvd\ n$
⟨*proof*⟩

**lemma** *dvd-nat-abs-iff* [*simp*]:
$n\ dvd\ nat\ |k| \longleftrightarrow int\ n\ dvd\ k$
⟨*proof*⟩

**lemma** *nat-abs-dvd-iff* [*simp*]:
$nat\ |k|\ dvd\ n \longleftrightarrow k\ dvd\ int\ n$
⟨*proof*⟩

**lemma** *zdvd1-eq* [*simp*]: $x\ dvd\ 1 \longleftrightarrow |x| = 1$ (**is** *?lhs* ⟷ *?rhs*)
**for** $x :: int$
⟨*proof*⟩

**lemma** *zdvd-mult-cancel1*:
**fixes** $m :: int$
**assumes** *mp*: $m \neq 0$
**shows** $m * n\ dvd\ m \longleftrightarrow |n| = 1$
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

**lemma** *nat-dvd-iff*: $nat\ z\ dvd\ m \longleftrightarrow (if\ 0 \leq z\ then\ z\ dvd\ int\ m\ else\ m = 0)$
⟨*proof*⟩

**lemma** *eq-nat-nat-iff*: $0 \leq z \Longrightarrow 0 \leq z' \Longrightarrow nat\ z = nat\ z' \longleftrightarrow z = z'$
⟨*proof*⟩

**lemma** *nat-power-eq*: $0 \leq z \Longrightarrow nat\ (z\ \widehat{}\ n) = nat\ z\ \widehat{}\ n$
⟨*proof*⟩

**lemma** *numeral-power-eq-nat-cancel-iff* [*simp*]:
$numeral\ x\ \widehat{}\ n = nat\ y \longleftrightarrow numeral\ x\ \widehat{}\ n = y$
⟨*proof*⟩

**lemma** *nat-eq-numeral-power-cancel-iff* [*simp*]:
$nat\ y = numeral\ x\ \widehat{}\ n \longleftrightarrow y = numeral\ x\ \widehat{}\ n$
⟨*proof*⟩

**lemma** *numeral-power-le-nat-cancel-iff* [*simp*]:
$numeral\ x\ \widehat{}\ n \leq nat\ a \longleftrightarrow numeral\ x\ \widehat{}\ n \leq a$
⟨*proof*⟩

**lemma** *nat-le-numeral-power-cancel-iff* [*simp*]:
$nat\ a \leq numeral\ x\ \widehat{}\ n \longleftrightarrow a \leq numeral\ x\ \widehat{}\ n$
⟨*proof*⟩

**lemma** *numeral-power-less-nat-cancel-iff* [*simp*]:
$numeral\ x\ \widehat{}\ n < nat\ a \longleftrightarrow numeral\ x\ \widehat{}\ n < a$
⟨*proof*⟩

**lemma** *nat-less-numeral-power-cancel-iff* [*simp*]:

*nat a* $<$ *numeral x* ^ *n* $\longleftrightarrow$ *a* $<$ *numeral x* ^ *n*
⟨*proof*⟩

**lemma** *zdvd-imp-le*: $z \leq n$ **if** *z dvd n* $0 < n$ **for** *n z* :: *int*
⟨*proof*⟩

**lemma** *zdvd-period*:
**fixes** *a d* :: *int*
**assumes** *a dvd d*
**shows** *a dvd* $(x + t)$ $\longleftrightarrow$ *a dvd* $((x + c * d) + t)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *fraction-numeral-not-in-Ints'* [*simp*]:
**assumes** $b \neq$ *Num.One*
**shows** *1* / *numeral b* $\notin$ ($\mathbb{Z}$ :: *'a* :: {*division-ring*, *ring-char-0*} *set*)
⟨*proof*⟩

## 54.18 Powers with integer exponents

The following allows writing powers with an integer exponent. While the type signature is very generic, most theorems will assume that the underlying type is a division ring or a field.

The notation 'powi' is inspired by the 'powr' notation for real/complex exponentiation.

**definition** *power-int* :: *'a* :: {*inverse*, *power*} $\Rightarrow$ *int* $\Rightarrow$ *'a* (**infixr** ‹*powi*› *80*) **where**
*power-int x n* = (*if* $n \geq 0$ *then x* ^ *nat n else inverse x* ^ (*nat* $(-n)$))

**lemma** *power-int-0-right* [*simp*]: *power-int x 0* = *1*
**and** *power-int-1-right* [*simp*]:
*power-int* (*y* :: *'a* :: {*power*, *inverse*, *monoid-mult*}) *1* = *y*
**and** *power-int-minus1-right* [*simp*]:
*power-int* (*y* :: *'a* :: {*power*, *inverse*, *monoid-mult*}) $(-1)$ = *inverse y*
⟨*proof*⟩

**lemma** *power-int-of-nat* [*simp*]: *power-int x* (*int n*) = *x* ^ *n*
⟨*proof*⟩

**lemma** *power-int-numeral* [*simp*]: *power-int x* (*numeral n*) = *x* ^ *numeral n*
⟨*proof*⟩

**lemma** *powi-numeral-reduce*: *x powi numeral n* = *x* $*$ *x powi int* (*pred-numeral n*)
⟨*proof*⟩

**lemma** *powi-minus-numeral-reduce*: *x powi* $-$ (*numeral n*) = *inverse x* $*$ *x powi* $-$ *int*(*pred-numeral n*)
⟨*proof*⟩

**lemma** *int-cases4* [*case-names nonneg neg*]:
**fixes** $m$ :: *int*
**obtains** $n$ **where** $m = int\ n \mid n$ **where** $n > 0\ m = -int\ n$
⟨*proof*⟩

**context**
**assumes** *SORT-CONSTRAINT*('*a*::*division-ring*)
**begin**

**lemma** *power-int-minus*: *power-int* $(x{::}'a)\ (-n) = inverse\ (power\text{-}int\ x\ n)$
⟨*proof*⟩

**lemma** *power-int-minus-divide*: *power-int* $(x{::}'a)\ (-n) = 1\ /\ (power\text{-}int\ x\ n)$
⟨*proof*⟩

**lemma** *power-int-eq-0-iff* [*simp*]: *power-int* $(x{::}'a)\ n = 0 \longleftrightarrow x = 0 \wedge n \neq 0$
⟨*proof*⟩

**lemma** *power-int-0-left-if*: *power-int* $(0 :: 'a)\ m = (if\ m = 0\ then\ 1\ else\ 0)$
⟨*proof*⟩

**lemma** *power-int-0-left* [*simp*]: $m \neq 0 \Longrightarrow power\text{-}int\ (0 :: 'a)\ m = 0$
⟨*proof*⟩

**lemma** *power-int-1-left* [*simp*]: *power-int* $1\ n = (1 :: 'a :: division\text{-}ring)$
⟨*proof*⟩

**lemma** *power-diff-conv-inverse*: $x \neq 0 \Longrightarrow m \leq n \Longrightarrow (x :: 'a)\ \widehat{}\ (n - m) = x\ \widehat{}\ n * inverse\ x\ \widehat{}\ m$
⟨*proof*⟩

**lemma** *power-mult-inverse-distrib*: $x\ \widehat{}\ m * inverse\ (x :: 'a) = inverse\ x * x\ \widehat{}\ m$
⟨*proof*⟩

**lemma** *power-mult-power-inverse-commute*:
$x\ \widehat{}\ m * inverse\ (x :: 'a)\ \widehat{}\ n = inverse\ x\ \widehat{}\ n * x\ \widehat{}\ m$
⟨*proof*⟩

**lemma** *power-int-add*:
**assumes** $x \neq 0 \vee m + n \neq 0$
**shows** *power-int* $(x{::}'a)\ (m + n) = power\text{-}int\ x\ m * power\text{-}int\ x\ n$
⟨*proof*⟩

**lemma** *power-int-add-1*:
**assumes** $x \neq 0 \vee m \neq -1$
**shows** *power-int* $(x{::}'a)\ (m + 1) = power\text{-}int\ x\ m * x$
⟨*proof*⟩

**lemma** *power-int-add-1'*:
**assumes** $x \neq 0 \vee m \neq -1$
**shows** *power-int* $(x::'a)$ $(m + 1) = x * $ *power-int* $x$ $m$
⟨*proof*⟩

**lemma** *power-int-commutes*: *power-int* $(x :: 'a)$ $n * x = x *$ *power-int* $x$ $n$
⟨*proof*⟩

**lemma** *power-int-inverse* [*field-simps*, *field-split-simps*, *divide-simps*]:
*power-int* (*inverse* $(x :: 'a)$) $n$ = *inverse* (*power-int* $x$ $n$)
⟨*proof*⟩

**lemma** *power-int-mult*: *power-int* $(x :: 'a)$ $(m * n)$ = *power-int* (*power-int* $x$ $m$) $n$
⟨*proof*⟩

**lemma** *power-int-power*: $(a$ ^ $b :: 'a ::$ *division-ring*) *powi* $c = a$ *powi* (*int* $b * c$)
⟨*proof*⟩

**lemma** *power-int-power'*: $(a$ *powi* $b :: 'a ::$ *division-ring*) ^ $c = a$ *powi* $(b *$ *int* $c)$
⟨*proof*⟩

**lemma** *power-int-nonneg-exp*: $n \geq 0 \Longrightarrow x$ *powi* $n = x$ ^ *nat* $n$
⟨*proof*⟩

**end**

**context**
**assumes** *SORT-CONSTRAINT*('*a*::*field*)
**begin**

**lemma** *power-int-diff*:
**assumes** $x \neq 0 \vee m \neq n$
**shows** *power-int* $(x::'a)$ $(m - n)$ = *power-int* $x$ $m$ / *power-int* $x$ $n$
⟨*proof*⟩

**lemma** *power-int-minus-mult*: $x \neq 0 \vee n \neq 0 \Longrightarrow$ *power-int* $(x :: 'a)$ $(n - 1) *$ $x$ = *power-int* $x$ $n$
⟨*proof*⟩

**lemma** *power-int-mult-distrib*: *power-int* $(x * y :: 'a)$ $m$ = *power-int* $x$ $m *$ *power-int* $y$ $m$
⟨*proof*⟩

**lemmas** *power-int-mult-distrib-numeral1* = *power-int-mult-distrib* [**where** $x$ = *numeral* $w$ **for** $w$, *simp*]
**lemmas** *power-int-mult-distrib-numeral2* = *power-int-mult-distrib* [**where** $y$ = *numeral* $w$ **for** $w$, *simp*]

**lemma** *power-int-divide-distrib*: *power-int* $(x / y :: 'a)$ $m$ = *power-int* $x$ $m$ /

*power-int y m*
⟨*proof*⟩

**end**

**lemma** *power-int-add-numeral* [*simp*]:
*power-int x* (*numeral m*) $*$ *power-int x* (*numeral n*) = *power-int x* (*numeral* (*m* + *n*))
**for** *x* :: *'a* :: *division-ring*
⟨*proof*⟩

**lemma** *power-int-add-numeral2* [*simp*]:
*power-int x* (*numeral m*) $*$ (*power-int x* (*numeral n*) $*$ *b*) = *power-int x* (*numeral* (*m* + *n*)) $*$ *b*
**for** *x* :: *'a* :: *division-ring*
⟨*proof*⟩

**lemma** *power-int-mult-numeral* [*simp*]:
*power-int* (*power-int x* (*numeral m*)) (*numeral n*) = *power-int x* (*numeral* (*m* $*$ *n*))
**for** *x* :: *'a* :: *division-ring*
⟨*proof*⟩

**lemma** *power-int-not-zero*: (*x* :: *'a* :: *division-ring*) ≠ *0* ∨ *n* = *0* ⟹ *power-int x n* ≠ *0*
⟨*proof*⟩

**lemma** *power-int-one-over* [*field-simps*, *field-split-simps*, *divide-simps*]:
*power-int* (*1* / *x* :: *'a* :: *division-ring*) *n* = *1* / *power-int x n*
⟨*proof*⟩

**context**
**assumes** *SORT-CONSTRAINT*(*'a* :: *linordered-field*)
**begin**

**lemma** *power-int-numeral-neg-numeral* [*simp*]:
*power-int* (*numeral m*) (−*numeral n*) = (*inverse* (*numeral* (*Num.pow m n*)) :: *'a*)
⟨*proof*⟩

**lemma** *zero-less-power-int* [*simp*]: *0* < (*x* :: *'a*) ⟹ *0* < *power-int x n*
⟨*proof*⟩

**lemma** *zero-le-power-int* [*simp*]: *0* ≤ (*x* :: *'a*) ⟹ *0* ≤ *power-int x n*
⟨*proof*⟩

**lemma** *power-int-mono*: (*x* :: *'a*) ≤ *y* ⟹ *n* ≥ *0* ⟹ *0* ≤ *x* ⟹ *power-int x n* ≤

*power-int y n*
  ⟨*proof*⟩

**lemma** *one-le-power-int* [*simp*]: $1 \leq (x :: 'a) \Longrightarrow n \geq 0 \Longrightarrow 1 \leq$ *power-int* $x$ $n$
  ⟨*proof*⟩

**lemma** *power-int-le-one*: $0 \leq (x :: 'a) \Longrightarrow n \geq 0 \Longrightarrow x \leq 1 \Longrightarrow$ *power-int* $x$ $n \leq 1$
  ⟨*proof*⟩

**lemma** *power-int-le-imp-le-exp*:
  **assumes** *gt1*: $1 < (x :: 'a ::$ *linordered-field*$)$
  **assumes** *power-int* $x$ $m \leq$ *power-int* $x$ $n$ $n \geq 0$
  **shows** $m \leq n$
⟨*proof*⟩

**lemma** *power-int-le-imp-less-exp*:
  **assumes** *gt1*: $1 < (x :: 'a ::$ *linordered-field*$)$
  **assumes** *power-int* $x$ $m <$ *power-int* $x$ $n$ $n \geq 0$
  **shows** $m < n$
⟨*proof*⟩

**lemma** *power-int-strict-mono*:
  $(a :: 'a ::$ *linordered-field*$) < b \Longrightarrow 0 \leq a \Longrightarrow 0 < n \Longrightarrow$ *power-int* $a$ $n <$ *power-int* $b$ $n$
  ⟨*proof*⟩

**lemma** *power-int-mono-iff* [*simp*]:
  **fixes** $a$ $b :: 'a ::$ *linordered-field*
  **shows** $[\![a \geq 0; b \geq 0; n > 0]\!] \Longrightarrow$ *power-int* $a$ $n \leq$ *power-int* $b$ $n \longleftrightarrow a \leq b$
  ⟨*proof*⟩

**lemma** *power-int-strict-increasing*:
  **fixes** $a :: 'a ::$ *linordered-field*
  **assumes** $n < N$ $1 < a$
  **shows** *power-int* $a$ $N >$ *power-int* $a$ $n$
⟨*proof*⟩

**lemma** *power-int-increasing*:
  **fixes** $a :: 'a ::$ *linordered-field*
  **assumes** $n \leq N$ $a \geq 1$
  **shows** *power-int* $a$ $N \geq$ *power-int* $a$ $n$
⟨*proof*⟩

**lemma** *power-int-strict-decreasing*:
  **fixes** $a :: 'a ::$ *linordered-field*
  **assumes** $n < N$ $0 < a$ $a < 1$
  **shows** *power-int* $a$ $N <$ *power-int* $a$ $n$
⟨*proof*⟩

**lemma** *power-int-decreasing*:
**fixes** $a :: 'a :: \mathit{linordered\text{-}field}$
**assumes** $n \leq N$ $0 \leq a$ $a \leq 1$ $a \neq 0 \vee N \neq 0 \vee n = 0$
**shows** $\mathit{power\text{-}int}\ a\ N \leq \mathit{power\text{-}int}\ a\ n$
⟨*proof*⟩

**lemma** *one-less-power-int*: $1 < (a :: 'a) \Longrightarrow 0 < n \Longrightarrow 1 < \mathit{power\text{-}int}\ a\ n$
⟨*proof*⟩

**lemma** *power-int-abs*: $|\mathit{power\text{-}int}\ a\ n :: 'a| = \mathit{power\text{-}int}\ |a|\ n$
⟨*proof*⟩

**lemma** *power-int-sgn* [*simp*]: $\mathit{sgn}\ (\mathit{power\text{-}int}\ a\ n :: 'a) = \mathit{power\text{-}int}\ (\mathit{sgn}\ a)\ n$
⟨*proof*⟩

**lemma** *abs-power-int-minus* [*simp*]: $|\mathit{power\text{-}int}\ (-\ a)\ n :: 'a| = |\mathit{power\text{-}int}\ a\ n|$
⟨*proof*⟩

**lemma** *power-int-strict-antimono*:
**assumes** $(a :: 'a :: \mathit{linordered\text{-}field}) < b$ $0 < a$ $n < 0$
**shows** $\mathit{power\text{-}int}\ a\ n > \mathit{power\text{-}int}\ b\ n$
⟨*proof*⟩

**lemma** *power-int-antimono*:
**assumes** $(a :: 'a :: \mathit{linordered\text{-}field}) \leq b$ $0 < a$ $n < 0$
**shows** $\mathit{power\text{-}int}\ a\ n \geq \mathit{power\text{-}int}\ b\ n$
⟨*proof*⟩

**end**

## 54.19 Finiteness of intervals

**lemma** *finite-interval-int1* [*iff*]: $\mathit{finite}\ \{i :: \mathit{int}.\ a \leq i \wedge i \leq b\}$
⟨*proof*⟩

**lemma** *finite-interval-int2* [*iff*]: $\mathit{finite}\ \{i :: \mathit{int}.\ a \leq i \wedge i < b\}$
⟨*proof*⟩

**lemma** *finite-interval-int3* [*iff*]: $\mathit{finite}\ \{i :: \mathit{int}.\ a < i \wedge i \leq b\}$
⟨*proof*⟩

**lemma** *finite-interval-int4* [*iff*]: $\mathit{finite}\ \{i :: \mathit{int}.\ a < i \wedge i < b\}$
⟨*proof*⟩

## 54.20 Configuration of the code generator

Constructors

**definition** $\mathit{Pos} :: \mathit{num} \Rightarrow \mathit{int}$

**where** [*simp*, *code-abbrev*]: $Pos = numeral$

**definition** $Neg :: num \Rightarrow int$
**where** [*simp*, *code-abbrev*]: $Neg\ n = -\ (Pos\ n)$

**code-datatype** $0::int\ Pos\ Neg$

Auxiliary operations.

**definition** $dup :: int \Rightarrow int$
**where** [*simp*]: $dup\ k = k + k$

**lemma** *dup-code* [*code*]:
$dup\ 0 = 0$
$dup\ (Pos\ n) = Pos\ (Num.Bit0\ n)$
$dup\ (Neg\ n) = Neg\ (Num.Bit0\ n)$
$\langle proof \rangle$

**definition** $sub :: num \Rightarrow num \Rightarrow int$
**where** [*simp*]: $sub\ m\ n = numeral\ m - numeral\ n$

**lemma** *sub-code* [*code*]:
$sub\ Num.One\ Num.One = 0$
$sub\ (Num.Bit0\ m)\ Num.One = Pos\ (Num.BitM\ m)$
$sub\ (Num.Bit1\ m)\ Num.One = Pos\ (Num.Bit0\ m)$
$sub\ Num.One\ (Num.Bit0\ n) = Neg\ (Num.BitM\ n)$
$sub\ Num.One\ (Num.Bit1\ n) = Neg\ (Num.Bit0\ n)$
$sub\ (Num.Bit0\ m)\ (Num.Bit0\ n) = dup\ (sub\ m\ n)$
$sub\ (Num.Bit1\ m)\ (Num.Bit1\ n) = dup\ (sub\ m\ n)$
$sub\ (Num.Bit1\ m)\ (Num.Bit0\ n) = dup\ (sub\ m\ n) + 1$
$sub\ (Num.Bit0\ m)\ (Num.Bit1\ n) = dup\ (sub\ m\ n) - 1$
$\langle proof \rangle$

**lemma** *sub-BitM-One-eq*:
$\langle (Num.sub\ (Num.BitM\ n)\ num.One) = 2 * (Num.sub\ n\ Num.One :: int) \rangle$
$\langle proof \rangle$

Implementations.

**lemma** *one-int-code* [*code*]: $1 = Pos\ Num.One$
$\langle proof \rangle$

**lemma** *plus-int-code* [*code*]:
$k + 0 = k$
$0 + l = l$
$Pos\ m + Pos\ n = Pos\ (m + n)$
$Pos\ m + Neg\ n = sub\ m\ n$
$Neg\ m + Pos\ n = sub\ n\ m$
$Neg\ m + Neg\ n = Neg\ (m + n)$
**for** $k\ l :: int$
$\langle proof \rangle$

**lemma** *uminus-int-code* [*code*]:
$uminus\ 0 = (0::int)$
$uminus\ (Pos\ m) = Neg\ m$
$uminus\ (Neg\ m) = Pos\ m$
⟨*proof*⟩

**lemma** *minus-int-code* [*code*]:
$k - 0 = k$
$0 - l = uminus\ l$
$Pos\ m - Pos\ n = sub\ m\ n$
$Pos\ m - Neg\ n = Pos\ (m + n)$
$Neg\ m - Pos\ n = Neg\ (m + n)$
$Neg\ m - Neg\ n = sub\ n\ m$
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *times-int-code* [*code*]:
$k * 0 = 0$
$0 * l = 0$
$Pos\ m * Pos\ n = Pos\ (m * n)$
$Pos\ m * Neg\ n = Neg\ (m * n)$
$Neg\ m * Pos\ n = Neg\ (m * n)$
$Neg\ m * Neg\ n = Pos\ (m * n)$
**for** $k\ l :: int$
⟨*proof*⟩

**instantiation** *int* :: *equal*
**begin**

**definition** $HOL.equal\ k\ l \longleftrightarrow k = (l::int)$

**instance**
⟨*proof*⟩

**end**

**lemma** *equal-int-code* [*code*]:
$HOL.equal\ 0\ (0::int) \longleftrightarrow True$
$HOL.equal\ 0\ (Pos\ l) \longleftrightarrow False$
$HOL.equal\ 0\ (Neg\ l) \longleftrightarrow False$
$HOL.equal\ (Pos\ k)\ 0 \longleftrightarrow False$
$HOL.equal\ (Pos\ k)\ (Pos\ l) \longleftrightarrow HOL.equal\ k\ l$
$HOL.equal\ (Pos\ k)\ (Neg\ l) \longleftrightarrow False$
$HOL.equal\ (Neg\ k)\ 0 \longleftrightarrow False$
$HOL.equal\ (Neg\ k)\ (Pos\ l) \longleftrightarrow False$
$HOL.equal\ (Neg\ k)\ (Neg\ l) \longleftrightarrow HOL.equal\ k\ l$
⟨*proof*⟩

**lemma** *equal-int-refl* [*code nbe*]: $HOL.equal\ k\ k \longleftrightarrow True$
**for** $k :: int$
⟨*proof*⟩

**lemma** *less-eq-int-code* [*code*]:
$0 \leq (0::int) \longleftrightarrow True$
$0 \leq Pos\ l \longleftrightarrow True$
$0 \leq Neg\ l \longleftrightarrow False$
$Pos\ k \leq 0 \longleftrightarrow False$
$Pos\ k \leq Pos\ l \longleftrightarrow k \leq l$
$Pos\ k \leq Neg\ l \longleftrightarrow False$
$Neg\ k \leq 0 \longleftrightarrow True$
$Neg\ k \leq Pos\ l \longleftrightarrow True$
$Neg\ k \leq Neg\ l \longleftrightarrow l \leq k$
⟨*proof*⟩

**lemma** *less-int-code* [*code*]:
$0 < (0::int) \longleftrightarrow False$
$0 < Pos\ l \longleftrightarrow True$
$0 < Neg\ l \longleftrightarrow False$
$Pos\ k < 0 \longleftrightarrow False$
$Pos\ k < Pos\ l \longleftrightarrow k < l$
$Pos\ k < Neg\ l \longleftrightarrow False$
$Neg\ k < 0 \longleftrightarrow True$
$Neg\ k < Pos\ l \longleftrightarrow True$
$Neg\ k < Neg\ l \longleftrightarrow l < k$
⟨*proof*⟩

**lemma** *nat-code* [*code*]:
$nat\ (Int.Neg\ k) = 0$
$nat\ 0 = 0$
$nat\ (Int.Pos\ k) = nat\text{-}of\text{-}num\ k$
⟨*proof*⟩

**lemma** (**in** *ring-1*) *of-int-code* [*code*]:
$of\text{-}int\ (Int.Neg\ k) = -\ numeral\ k$
$of\text{-}int\ 0 = 0$
$of\text{-}int\ (Int.Pos\ k) = numeral\ k$
⟨*proof*⟩

Serializer setup.

**code-identifier**
**code-module** *Int* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**quickcheck-params** [*default-type* = *int*]

**hide-const** (**open**) *Pos Neg sub dup*

De-register *int* as a quotient type:

**lifting-update** *int.lifting*
**lifting-forget** *int.lifting*

### 54.21 Duplicates

**lemmas** *int-sum* = *of-nat-sum* [**where** $'a$=*int*]
**lemmas** *int-prod* = *of-nat-prod* [**where** $'a$=*int*]
**lemmas** *zle-int* = *of-nat-le-iff* [**where** $'a$=*int*]
**lemmas** *int-int-eq* = *of-nat-eq-iff* [**where** $'a$=*int*]
**lemmas** *nonneg-eq-int* = *nonneg-int-cases*
**lemmas** *double-eq-0-iff* = *double-zero*

**lemmas** *int-distrib* =
*distrib-right* [*of z1 z2 w*]
*distrib-left* [*of w z1 z2*]
*left-diff-distrib* [*of z1 z2 w*]
*right-diff-distrib* [*of w z1 z2*]
**for** *z1 z2 w* :: *int*

**end**

# 55 Big infimum (minimum) and supremum (maximum) over finite (non-empty) sets

**theory** *Lattices-Big*
**imports** *Groups-Big Option*
**begin**

## 55.1 Generic lattice operations over a set

### 55.1.1 Without neutral element

**locale** *semilattice-set* = *semilattice*
**begin**

**interpretation** *comp-fun-idem f*
⟨*proof*⟩

**definition** $F$ :: $'a$ *set* ⇒ $'a$
**where**
*eq-fold'*: $F$ $A$ = *the* (*Finite-Set.fold* (λ*x y*. *Some* (*case y of None* ⇒ *x* | *Some z* ⇒ *f x z*)) *None A*)

**lemma** *eq-fold*:
**assumes** *finite A*
**shows** $F$ (*insert x A*) = *Finite-Set.fold f x A*
⟨*proof*⟩

**lemma** *singleton* [*simp*]:

$F\ \{x\} = x$
⟨*proof*⟩

**lemma** *insert-not-elem*:
**assumes** *finite* $A$ **and** $x \notin A$ **and** $A \neq \{\}$
**shows** $F\ (insert\ x\ A) = x * F\ A$
⟨*proof*⟩

**lemma** *in-idem*:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $x * F\ A = F\ A$
⟨*proof*⟩

**lemma** *insert* [*simp*]:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $F\ (insert\ x\ A) = x * F\ A$
⟨*proof*⟩

**lemma** *union*:
**assumes** *finite* $A$ $A \neq \{\}$ **and** *finite* $B$ $B \neq \{\}$
**shows** $F\ (A \cup B) = F\ A * F\ B$
⟨*proof*⟩

**lemma** *remove*:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $F\ A = (if\ A - \{x\} = \{\}\ then\ x\ else\ x * F\ (A - \{x\}))$
⟨*proof*⟩

**lemma** *insert-remove*:
**assumes** *finite* $A$
**shows** $F\ (insert\ x\ A) = (if\ A - \{x\} = \{\}\ then\ x\ else\ x * F\ (A - \{x\}))$
⟨*proof*⟩

**lemma** *subset*:
**assumes** *finite* $A$ $B \neq \{\}$ **and** $B \subseteq A$
**shows** $F\ B * F\ A = F\ A$
⟨*proof*⟩

**lemma** *closed*:
**assumes** *finite* $A$ $A \neq \{\}$ **and** *elem*: $\bigwedge x\ y.\ x * y \in \{x, y\}$
**shows** $F\ A \in A$
⟨*proof*⟩

**lemma** *hom-commute*:
**assumes** *hom*: $\bigwedge x\ y.\ h\ (x * y) = h\ x * h\ y$
**and** $N$: *finite* $N$ $N \neq \{\}$
**shows** $h\ (F\ N) = F\ (h\ `\ N)$
⟨*proof*⟩

**lemma** *infinite*: $\neg$ *finite* $A \Longrightarrow F\ A = \mathit{the\ None}$
⟨*proof*⟩

**end**

**locale** *semilattice-order-set* = *binary?*: *semilattice-order* + *semilattice-set*
**begin**

**lemma** *bounded-iff*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $x \leq F\ A \longleftrightarrow (\forall\, a{\in}A.\ x \leq a)$
⟨*proof*⟩

**lemma** *boundedI*:
**assumes** *finite* $A$
**assumes** $A \neq \{\}$
**assumes** $\bigwedge a.\ a \in A \Longrightarrow x \leq a$
**shows** $x \leq F\ A$
⟨*proof*⟩

**lemma** *boundedE*:
**assumes** *finite* $A$ **and** $A \neq \{\}$ **and** $x \leq F\ A$
**obtains** $\bigwedge a.\ a \in A \Longrightarrow x \leq a$
⟨*proof*⟩

**lemma** *coboundedI*:
**assumes** *finite* $A$
**and** $a \in A$
**shows** $F\ A \leq a$
⟨*proof*⟩

**lemma** *subset-imp*:
**assumes** $A \subseteq B$ **and** $A \neq \{\}$ **and** *finite* $B$
**shows** $F\ B \leq F\ A$
⟨*proof*⟩

**end**

### 55.1.2 With neutral element

**locale** *semilattice-neutr-set* = *semilattice-neutr*
**begin**

**interpretation** *comp-fun-idem f*
⟨*proof*⟩

**definition** $F$ :: *'a set* $\Rightarrow$ *'a*
**where**
*eq-fold*: $F\ A = \mathit{Finite\text{-}Set.fold}\ f\ \mathbf{1}\ A$

**lemma** *infinite* [*simp*]:
$\neg$ *finite* $A \Longrightarrow F\ A = \mathbf{1}$
⟨*proof*⟩

**lemma** *empty* [*simp*]:
$F\ \{\} = \mathbf{1}$
⟨*proof*⟩

**lemma** *insert* [*simp*]:
**assumes** *finite* $A$
**shows** $F\ (insert\ x\ A) = x * F\ A$
⟨*proof*⟩

**lemma** *in-idem*:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $x * F\ A = F\ A$
⟨*proof*⟩

**lemma** *union*:
**assumes** *finite* $A$ **and** *finite* $B$
**shows** $F\ (A \cup B) = F\ A * F\ B$
⟨*proof*⟩

**lemma** *remove*:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $F\ A = x * F\ (A - \{x\})$
⟨*proof*⟩

**lemma** *insert-remove*:
**assumes** *finite* $A$
**shows** $F\ (insert\ x\ A) = x * F\ (A - \{x\})$
⟨*proof*⟩

**lemma** *subset*:
**assumes** *finite* $A$ **and** $B \subseteq A$
**shows** $F\ B * F\ A = F\ A$
⟨*proof*⟩

**lemma** *closed*:
**assumes** *finite* $A$ $A \neq \{\}$ **and** *elem*: $\bigwedge x\ y.\ x * y \in \{x,\ y\}$
**shows** $F\ A \in A$
⟨*proof*⟩

**end**

**locale** *semilattice-order-neutr-set* = *binary?*: *semilattice-neutr-order* + *semilattice-neutr-set*
**begin**

**lemma** *bounded-iff*:
**assumes** *finite* $A$
**shows** $x \leq F\ A \longleftrightarrow (\forall\, a \in A.\ x \leq a)$
⟨*proof*⟩

**lemma** *boundedI*:
**assumes** *finite* $A$
**assumes** $\bigwedge a.\ a \in A \Longrightarrow x \leq a$
**shows** $x \leq F\ A$
⟨*proof*⟩

**lemma** *boundedE*:
**assumes** *finite* $A$ **and** $x \leq F\ A$
**obtains** $\bigwedge a.\ a \in A \Longrightarrow x \leq a$
⟨*proof*⟩

**lemma** *coboundedI*:
**assumes** *finite* $A$
**and** $a \in A$
**shows** $F\ A \leq a$
⟨*proof*⟩

**lemma** *subset-imp*:
**assumes** $A \subseteq B$ **and** *finite* $B$
**shows** $F\ B \leq F\ A$
⟨*proof*⟩

**end**

## 55.2 Lattice operations on finite sets

**context** *semilattice-inf*
**begin**

**sublocale** *Inf-fin*: *semilattice-order-set inf less-eq less*
**defines**
*Inf-fin* (‹$\bigsqcap_{fin}$› [900] 900) = *Inf-fin.F* ⟨*proof*⟩

**end**

**context** *semilattice-sup*
**begin**

**sublocale** *Sup-fin*: *semilattice-order-set sup greater-eq greater*
**defines**
*Sup-fin* (‹$\bigsqcup_{fin}$› [900] 900) = *Sup-fin.F* ⟨*proof*⟩

**end**

## 55.3 Infimum and Supremum over non-empty sets

**context** *lattice*
**begin**

**lemma** *Inf-fin-le-Sup-fin* [*simp*]:
 **assumes** *finite* $A$ **and** $A \neq \{\}$
 **shows** $\bigsqcap_{fin} A \leq \bigsqcup_{fin} A$
⟨*proof*⟩

**lemma** *sup-Inf-absorb* [*simp*]:
 *finite* $A \Longrightarrow a \in A \Longrightarrow \bigsqcap_{fin} A \sqcup a = a$
 ⟨*proof*⟩

**lemma** *inf-Sup-absorb* [*simp*]:
 *finite* $A \Longrightarrow a \in A \Longrightarrow a \sqcap \bigsqcup_{fin} A = a$
 ⟨*proof*⟩

**end**

**context** *distrib-lattice*
**begin**

**lemma** *sup-Inf1-distrib*:
 **assumes** *finite* $A$
  **and** $A \neq \{\}$
 **shows** *sup* $x$ $(\bigsqcap_{fin} A) = \bigsqcap_{fin}\{sup\ x\ a|a.\ a \in A\}$
⟨*proof*⟩

**lemma** *sup-Inf2-distrib*:
 **assumes** $A$: *finite* $A$ $A \neq \{\}$ **and** $B$: *finite* $B$ $B \neq \{\}$
 **shows** *sup* $(\bigsqcap_{fin} A)$ $(\bigsqcap_{fin} B) = \bigsqcap_{fin}\{sup\ a\ b|a\ b.\ a \in A \wedge b \in B\}$
⟨*proof*⟩

**lemma** *inf-Sup1-distrib*:
 **assumes** *finite* $A$ **and** $A \neq \{\}$
 **shows** *inf* $x$ $(\bigsqcup_{fin} A) = \bigsqcup_{fin}\{inf\ x\ a|a.\ a \in A\}$
⟨*proof*⟩

**lemma** *inf-Sup2-distrib*:
 **assumes** $A$: *finite* $A$ $A \neq \{\}$ **and** $B$: *finite* $B$ $B \neq \{\}$
 **shows** *inf* $(\bigsqcup_{fin} A)$ $(\bigsqcup_{fin} B) = \bigsqcup_{fin}\{inf\ a\ b|a\ b.\ a \in A \wedge b \in B\}$
⟨*proof*⟩

**end**

**context** *complete-lattice*
**begin**

**lemma** *Inf-fin-Inf*:

**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $\bigsqcap_{fin} A = \bigsqcap A$
$\langle proof \rangle$

**lemma** *Sup-fin-Sup*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $\bigsqcup_{fin} A = \bigsqcup A$
$\langle proof \rangle$

**end**

## 55.4 Minimum and Maximum over non-empty sets

**context** *linorder*
**begin**

**sublocale** *Min*: *semilattice-order-set min less-eq less*
+ *Max*: *semilattice-order-set max greater-eq greater*
**defines**
*Min* = *Min.F* **and** *Max* = *Max.F* $\langle proof \rangle$

**end**

**syntax**
*-MIN1* :: *pttrns* ⇒ ′*b* ⇒ ′*b* (‹(‹indent=3 notation=‹binder MIN››MIN -./ -)› [0, 10] 10)
*-MIN* :: *pttrn* ⇒ ′*a set* ⇒ ′*b* ⇒ ′*b* (‹(‹indent=3 notation=‹binder MIN››MIN -∈-./ -)› [0, 0, 10] 10)
*-MAX1* :: *pttrns* ⇒ ′*b* ⇒ ′*b* (‹(‹indent=3 notation=‹binder MAX››MAX -./ -)› [0, 10] 10)
*-MAX* :: *pttrn* ⇒ ′*a set* ⇒ ′*b* ⇒ ′*b* (‹(‹indent=3 notation=‹binder MAX››MAX -∈-./ -)› [0, 0, 10] 10)

**syntax-consts**
*-MIN1 -MIN* ⇌ *Min* **and**
*-MAX1 -MAX* ⇌ *Max*

**translations**
*MIN x y. f* ⇌ *MIN x. MIN y. f*
*MIN x. f* ⇌ *CONST Min* (*CONST range* (λ*x. f*))
*MIN x*∈*A. f* ⇌ *CONST Min* ((λ*x. f*) ‘ *A*)
*MAX x y. f* ⇌ *MAX x. MAX y. f*
*MAX x. f* ⇌ *CONST Max* (*CONST range* (λ*x. f*))
*MAX x*∈*A. f* ⇌ *CONST Max* ((λ*x. f*) ‘ *A*)

An aside: *Min*/*Max* on linear orders as special case of *Inf-fin*/*Sup-fin*

**lemma** *Inf-fin-Min*:
*Inf-fin* = (*Min* :: ′*a*::{*semilattice-inf*, *linorder*} *set* ⇒ ′*a*)
$\langle proof \rangle$

**lemma** *Sup-fin-Max*:
*Sup-fin* = (*Max* :: *'a*::{*semilattice-sup*, *linorder*} *set* ⇒ *'a*)
⟨*proof*⟩

**context** *linorder*
**begin**

**lemma** *dual-min*:
*ord.min greater-eq* = *max*
⟨*proof*⟩

**lemma** *dual-max*:
*ord.max greater-eq* = *min*
⟨*proof*⟩

**lemma** *dual-Min*:
*linorder.Min greater-eq* = *Max*
⟨*proof*⟩

**lemma** *dual-Max*:
*linorder.Max greater-eq* = *Min*
⟨*proof*⟩

**lemmas** *Min-singleton* = *Min.singleton*
**lemmas** *Max-singleton* = *Max.singleton*
**lemmas** *Min-insert* = *Min.insert*
**lemmas** *Max-insert* = *Max.insert*
**lemmas** *Min-Un* = *Min.union*
**lemmas** *Max-Un* = *Max.union*
**lemmas** *hom-Min-commute* = *Min.hom-commute*
**lemmas** *hom-Max-commute* = *Max.hom-commute*

**lemma** *Min-in* [*simp*]:
**assumes** *finite A* **and** $A \neq \{\}$
**shows** *Min A* ∈ *A*
⟨*proof*⟩

**lemma** *Max-in* [*simp*]:
**assumes** *finite A* **and** $A \neq \{\}$
**shows** *Max A* ∈ *A*
⟨*proof*⟩

**lemma** *Min-insert2*:
**assumes** *finite A* **and** *min*: $\bigwedge b.\ b \in A \Longrightarrow a \leq b$
**shows** *Min* (*insert a A*) = *a*
⟨*proof*⟩

**lemma** *Max-insert2*:

**assumes** *finite* $A$ **and** *max*: $\bigwedge b.\ b \in A \Longrightarrow b \leq a$
**shows** *Max* (*insert* $a$ $A$) $= a$
⟨*proof*⟩

**lemma** *Max-const*[*simp*]: $[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow$ *Max* $((\lambda\text{-}.\ c)$ ‘ $A) = c$
⟨*proof*⟩

**lemma** *Min-const*[*simp*]: $[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow$ *Min* $((\lambda\text{-}.\ c)$ ‘ $A) = c$
⟨*proof*⟩

**lemma** *Min-le* [*simp*]:
**assumes** *finite* $A$ **and** $x \in A$
**shows** *Min* $A \leq x$
⟨*proof*⟩

**lemma** *Max-ge* [*simp*]:
**assumes** *finite* $A$ **and** $x \in A$
**shows** $x \leq$ *Max* $A$
⟨*proof*⟩

**lemma** *Min-eqI*:
**assumes** *finite* $A$
**assumes** $\bigwedge y.\ y \in A \Longrightarrow y \geq x$
**and** $x \in A$
**shows** *Min* $A = x$
⟨*proof*⟩

**lemma** *Max-eqI*:
**assumes** *finite* $A$
**assumes** $\bigwedge y.\ y \in A \Longrightarrow y \leq x$
**and** $x \in A$
**shows** *Max* $A = x$
⟨*proof*⟩

**lemma** *eq-Min-iff*:
$[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow m =$ *Min* $A \longleftrightarrow m \in A \wedge (\forall a \in A.\ m \leq a)$
⟨*proof*⟩

**lemma** *Min-eq-iff*:
$[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow$ *Min* $A = m \longleftrightarrow m \in A \wedge (\forall a \in A.\ m \leq a)$
⟨*proof*⟩

**lemma** *eq-Max-iff*:
$[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow m =$ *Max* $A \longleftrightarrow m \in A \wedge (\forall a \in A.\ a \leq m)$
⟨*proof*⟩

**lemma** *Max-eq-iff*:
$[\![$ *finite* $A$; $A \neq \{\}$ $]\!] \Longrightarrow$ *Max* $A = m \longleftrightarrow m \in A \wedge (\forall a \in A.\ a \leq m)$
⟨*proof*⟩

**context**
**fixes** $A$ :: $'a$ $set$
**assumes** *fin-nonempty*: *finite* $A$ $A \neq \{\}$
**begin**

**lemma** *Min-ge-iff* [*simp*]:
$x \leq Min\ A \longleftrightarrow (\forall\, a{\in}A.\ x \leq a)$
⟨*proof*⟩

**lemma** *Max-le-iff* [*simp*]:
$Max\ A \leq x \longleftrightarrow (\forall\, a{\in}A.\ a \leq x)$
⟨*proof*⟩

**lemma** *Min-gr-iff* [*simp*]:
$x < Min\ A \longleftrightarrow (\forall\, a{\in}A.\ x < a)$
⟨*proof*⟩

**lemma** *Max-less-iff* [*simp*]:
$Max\ A < x \longleftrightarrow (\forall\, a{\in}A.\ a < x)$
⟨*proof*⟩

**lemma** *Min-le-iff*:
$Min\ A \leq x \longleftrightarrow (\exists\, a{\in}A.\ a \leq x)$
⟨*proof*⟩

**lemma** *Max-ge-iff*:
$x \leq Max\ A \longleftrightarrow (\exists\, a{\in}A.\ x \leq a)$
⟨*proof*⟩

**lemma** *Min-less-iff*:
$Min\ A < x \longleftrightarrow (\exists\, a{\in}A.\ a < x)$
⟨*proof*⟩

**lemma** *Max-gr-iff*:
$x < Max\ A \longleftrightarrow (\exists\, a{\in}A.\ x < a)$
⟨*proof*⟩

**end**

Handy results about *Max* and *Min* by Chelsea Edmonds

**lemma** *obtains-Max*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**obtains** $x$ **where** $x \in A$ **and** $Max\ A = x$
⟨*proof*⟩

**lemma** *obtains-MAX*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**obtains** $x$ **where** $x \in A$ **and** $Max\ (f\ `\ A) = f\ x$

⟨*proof*⟩

**lemma** *obtains-Min*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**obtains** $x$ **where** $x \in A$ **and** $Min\ A = x$
⟨*proof*⟩

**lemma** *obtains-MIN*:
**assumes** *finite* $A$ **and** $A \neq \{\}$
**obtains** $x$ **where** $x \in A$ **and** $Min\ (f\ `\ A) = f\ x$
⟨*proof*⟩

**lemma** *Max-eq-if*:
**assumes** *finite* $A$ *finite* $B$ $\forall a \in A.\ \exists b \in B.\ a \leq b$ $\forall b \in B.\ \exists a \in A.\ b \leq a$
**shows** $Max\ A = Max\ B$
⟨*proof*⟩

**lemma** *Min-antimono*:
**assumes** $M \subseteq N$ **and** $M \neq \{\}$ **and** *finite* $N$
**shows** $Min\ N \leq Min\ M$
⟨*proof*⟩

**lemma** *Max-mono*:
**assumes** $M \subseteq N$ **and** $M \neq \{\}$ **and** *finite* $N$
**shows** $Max\ M \leq Max\ N$
⟨*proof*⟩

**lemma** *mono-Min-commute*:
**assumes** *mono* $f$
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $f\ (Min\ A) = Min\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *mono-Max-commute*:
**assumes** *mono* $f$
**assumes** *finite* $A$ **and** $A \neq \{\}$
**shows** $f\ (Max\ A) = Max\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *finite-linorder-max-induct* [*consumes 1*, *case-names empty insert*]:
**assumes** *fin*: *finite* $A$
**and** *empty*: $P\ \{\}$
**and** *insert*: $\bigwedge b\ A.\ finite\ A \Longrightarrow \forall a \in A.\ a < b \Longrightarrow P\ A \Longrightarrow P\ (insert\ b\ A)$
**shows** $P\ A$
⟨*proof*⟩

**lemma** *finite-linorder-min-induct* [*consumes 1*, *case-names empty insert*]:
$[\![finite\ A;\ P\ \{\};\ \bigwedge b\ A.\ [\![finite\ A;\ \forall a \in A.\ b < a;\ P\ A]\!] \Longrightarrow P\ (insert\ b\ A)]\!] \Longrightarrow P\ A$

⟨*proof*⟩

**lemma** *finite-ranking-induct*[*consumes 1*, *case-names empty insert*]:
**fixes** $f :: 'b \Rightarrow 'a$
**assumes** *finite* $S$
**assumes** $P$ {}
**assumes** $\bigwedge x\ S.\ \mathit{finite}\ S \Longrightarrow (\bigwedge y.\ y \in S \Longrightarrow f\ y \leq f\ x) \Longrightarrow P\ S \Longrightarrow P\ (\mathit{insert}\ x\ S)$
**shows** $P\ S$
⟨*proof*⟩

**lemma** *Least-Min*:
**assumes** *finite* $\{a.\ P\ a\}$ **and** $\exists\, a.\ P\ a$
**shows** $(\mathit{LEAST}\ a.\ P\ a) = \mathit{Min}\ \{a.\ P\ a\}$
⟨*proof*⟩

**lemma** *Greatest-Max*:
**assumes** *finite* $\{a.\ P\ a\}$ **and** $\exists\, a.\ P\ a$
**shows** $(\mathit{GREATEST}\ a.\ P\ a) = \mathit{Max}\ \{a.\ P\ a\}$
⟨*proof*⟩

**lemma** *infinite-growing*:
**assumes** $X \neq \{\}$
**assumes** $*$: $\bigwedge x.\ x \in X \Longrightarrow \exists\, y{\in}X.\ y > x$
**shows** $\neg$ *finite* $X$
⟨*proof*⟩

**end**

**lemma** *sum-le-card-Max*: $\mathit{finite}\ A \Longrightarrow \mathit{sum}\ f\ A \leq \mathit{card}\ A * \mathit{Max}\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *card-Min-le-sum*: $\mathit{finite}\ A \Longrightarrow \mathit{card}\ A * \mathit{Min}\ (f\ `\ A) \leq \mathit{sum}\ f\ A$
⟨*proof*⟩

**context** *linordered-ab-semigroup-add*
**begin**

**lemma** *Min-add-commute*:
**fixes** $k$
**assumes** *finite* $S$ **and** $S \neq \{\}$
**shows** $\mathit{Min}\ ((\lambda x.\ f\ x + k)\ `\ S) = \mathit{Min}(f\ `\ S) + k$
⟨*proof*⟩

**lemma** *Max-add-commute*:
**fixes** $k$
**assumes** *finite* $S$ **and** $S \neq \{\}$
**shows** $\mathit{Max}\ ((\lambda x.\ f\ x + k)\ `\ S) = \mathit{Max}(f\ `\ S) + k$
⟨*proof*⟩

**end**

**context** *linordered-ab-group-add*
**begin**

**lemma** *minus-Max-eq-Min* [*simp*]:
$\mathit{finite}\ S \Longrightarrow S \neq \{\} \Longrightarrow -\ \mathit{Max}\ S = \mathit{Min}\ (\mathit{uminus}\ `\ S)$
⟨*proof*⟩

**lemma** *minus-Min-eq-Max* [*simp*]:
$\mathit{finite}\ S \Longrightarrow S \neq \{\} \Longrightarrow -\ \mathit{Min}\ S = \mathit{Max}\ (\mathit{uminus}\ `\ S)$
⟨*proof*⟩

**end**

**context** *complete-linorder*
**begin**

**lemma** *Min-Inf*:
**assumes** *finite A* **and** $A \neq \{\}$
**shows** $\mathit{Min}\ A = \mathit{Inf}\ A$
⟨*proof*⟩

**lemma** *Max-Sup*:
**assumes** *finite A* **and** $A \neq \{\}$
**shows** $\mathit{Max}\ A = \mathit{Sup}\ A$
⟨*proof*⟩

**end**

**lemma** *disjnt-ge-max*:
‹*disjnt X Y*› **if** ‹*finite Y*› ‹$\bigwedge x.\ x \in X \Longrightarrow x > \mathit{Max}\ Y$›
⟨*proof*⟩

## 55.5 An aside: code generation for *LEAST* and *GREATEST*

**context**
**begin**

**qualified definition** *Least* :: ‹$'a::\mathit{linorder}\ \mathit{set} \Rightarrow 'a$› — only for code generation
**where** *Least-eq* [*code-abbrev*, *simp*]: ‹$\mathit{Least}\ S = (\mathit{LEAST}\ x.\ x \in S)$›

**qualified lemma** *Least-filter-eq* [*code-abbrev*]:
‹$\mathit{Least}\ (\mathit{Set.filter}\ P\ S) = (\mathit{LEAST}\ x.\ x \in S \wedge P\ x)$›
⟨*proof*⟩ **definition** *Least-abort* :: ‹$'a\ \mathit{set} \Rightarrow 'a::\mathit{linorder}$›
**where** ‹$\mathit{Least\text{-}abort} = \mathit{Least}$›

**qualified lemma** *Least-code* [*code abort*: *Lattices-Big.Least-abort*, *code*]:

‹*Least A* = (*if finite A* ⟶ *Set.is-empty A then Least-abort A else Min A*)›
⟨*proof*⟩ **definition** *Greatest* :: ‹′*a*::*linorder set* ⇒ ′*a*› — only for code generation
**where** *Greatest-eq* [*code-abbrev*, *simp*]: ‹*Greatest S* = (*GREATEST x*. *x* ∈ *S*)›

**qualified lemma** *Greatest-filter-eq* [*code-abbrev*]:
‹*Greatest* (*Set.filter P S*) = (*GREATEST x*. *x* ∈ *S* ∧ *P x*)›
⟨*proof*⟩ **definition** *Greatest-abort* :: ‹′*a set* ⇒ ′*a*::*linorder*›
**where** ‹*Greatest-abort* = *Greatest*›

**qualified lemma** *Greatest-code* [*code abort*: *Lattices-Big.Greatest-abort*, *code*]:
‹*Greatest A* = (*if finite A* ⟶ *Set.is-empty A then Greatest-abort A else Max A*)›
⟨*proof*⟩

**end**

## 55.6 Arg Min

**context** *ord*
**begin**

**definition** *is-arg-min* :: (′*b* ⇒ ′*a*) ⇒ (′*b* ⇒ *bool*) ⇒ ′*b* ⇒ *bool* **where**
*is-arg-min f P x* = (*P x* ∧ ¬(∃ *y*. *P y* ∧ *f y* < *f x*))

**definition** *arg-min* :: (′*b* ⇒ ′*a*) ⇒ (′*b* ⇒ *bool*) ⇒ ′*b* **where**
*arg-min f P* = (*SOME x*. *is-arg-min f P x*)

**definition** *arg-min-on* :: (′*b* ⇒ ′*a*) ⇒ ′*b set* ⇒ ′*b* **where**
*arg-min-on f S* = *arg-min f* (λ*x*. *x* ∈ *S*)

**end**

**syntax**
*-arg-min* :: (′*b* ⇒ ′*a*) ⇒ *pttrn* ⇒ *bool* ⇒ ′*b*
(‹(‹*indent=3 notation=*‹*binder ARG-MIN*››*ARG′-MIN - -./ -*)› [*1000*, *0*, *10*] *10*)
**syntax-consts**
*-arg-min* ⇌ *arg-min*
**translations**
*ARG-MIN f x*. *P* ⇌ *CONST arg-min f* (λ*x*. *P*)

**lemma** *is-arg-min-linorder*: **fixes** *f* :: ′*a* ⇒ ′*b* :: *linorder*
**shows** *is-arg-min f P x* = (*P x* ∧ (∀ *y*. *P y* ⟶ *f x* ≤ *f y*))
⟨*proof*⟩

**lemma** *is-arg-min-antimono*: **fixes** *f* :: ′*a* ⇒ (′*b*::*order*)
**shows** ⟦ *is-arg-min f P x*; *f y* ≤ *f x*; *P y* ⟧ ⟹ *is-arg-min f P y*
⟨*proof*⟩

**lemma** *arg-minI*:

$\llbracket$ $P\ x$;
$\bigwedge y.\ P\ y \Longrightarrow \neg\ f\ y < f\ x$;
$\bigwedge x.\ \llbracket\ P\ x;\ \forall\, y.\ P\ y \longrightarrow \neg\ f\ y < f\ x\ \rrbracket \Longrightarrow Q\ x\ \rrbracket$
$\Longrightarrow Q\ (\mathit{arg\text{-}min}\ f\ P)$
⟨*proof*⟩

**lemma** *arg-min-equality*:
$\llbracket\ P\ k;\ \bigwedge x.\ P\ x \Longrightarrow f\ k \le f\ x\ \rrbracket \Longrightarrow f\ (\mathit{arg\text{-}min}\ f\ P) = f\ k$
**for** $f$ :: - $\Rightarrow$ *'a::order*
⟨*proof*⟩

**lemma** *wf-linord-ex-has-least*:
$\llbracket\ \mathit{wf}\ r;\ \forall\, x\ y.\ (x,\ y) \in r^{+} \longleftrightarrow (y,\ x) \notin r^{*};\ P\ k\ \rrbracket$
$\Longrightarrow \exists\, x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow (m\ x,\ m\ y) \in r^{*})$
⟨*proof*⟩

**lemma** *ex-has-least-nat*: $P\ k \Longrightarrow \exists\, x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow m\ x \le m\ y)$
**for** $m$ :: *'a* $\Rightarrow$ *nat*
⟨*proof*⟩

**lemma** *arg-min-nat-lemma*:
$P\ k \Longrightarrow P(\mathit{arg\text{-}min}\ m\ P) \wedge (\forall\, y.\ P\ y \longrightarrow m\ (\mathit{arg\text{-}min}\ m\ P) \le m\ y)$
**for** $m$ :: *'a* $\Rightarrow$ *nat*
⟨*proof*⟩

**lemmas** *arg-min-natI* = *arg-min-nat-lemma* [*THEN conjunct1*]

**lemma** *is-arg-min-arg-min-nat*: **fixes** $m$ :: *'a* $\Rightarrow$ *nat*
**shows** $P\ x \Longrightarrow \mathit{is\text{-}arg\text{-}min}\ m\ P\ (\mathit{arg\text{-}min}\ m\ P)$
⟨*proof*⟩

**lemma** *arg-min-nat-le*: $P\ x \Longrightarrow m\ (\mathit{arg\text{-}min}\ m\ P) \le m\ x$
**for** $m$ :: *'a* $\Rightarrow$ *nat*
⟨*proof*⟩

**lemma** *ex-min-if-finite*:
$\llbracket\ \mathit{finite}\ S;\ S \neq \{\}\ \rrbracket \Longrightarrow \exists\, m{\in}S.\ \neg(\exists\, x{\in}S.\ x < (m{::}'a{::}\mathit{order}))$
⟨*proof*⟩

**lemma** *ex-is-arg-min-if-finite*: **fixes** $f$ :: *'a* $\Rightarrow$ *'b* :: *order*
**shows** $\llbracket\ \mathit{finite}\ S;\ S \neq \{\}\ \rrbracket \Longrightarrow \exists\, x.\ \mathit{is\text{-}arg\text{-}min}\ f\ (\lambda x.\ x \in S)\ x$
⟨*proof*⟩

**lemma** *arg-min-SOME-Min*:
$\mathit{finite}\ S \Longrightarrow \mathit{arg\text{-}min\text{-}on}\ f\ S = (\mathit{SOME}\ y.\ y \in S \wedge f\ y = \mathit{Min}(f\ `\ S))$
⟨*proof*⟩

**lemma** *arg-min-if-finite*: **fixes** $f$ :: *'a* $\Rightarrow$ *'b* :: *order*
**assumes** *finite S* $S \neq \{\}$

**shows** $\textit{arg-min-on } f\ S \in S$ **and** $\neg(\exists\, x{\in}S.\ f\ x < f\ (\textit{arg-min-on } f\ S))$
$\langle \textit{proof} \rangle$

**lemma** *arg-min-least*: **fixes** $f :: {}'a \Rightarrow {}'b :: \textit{linorder}$
**shows** $[\![\ \textit{finite } S;\ \ S \neq \{\};\ \ y \in S\ ]\!] \Longrightarrow f(\textit{arg-min-on } f\ S) \leq f\ y$
$\langle \textit{proof} \rangle$

**lemma** *arg-min-inj-eq*: **fixes** $f :: {}'a \Rightarrow {}'b :: \textit{order}$
**shows** $[\![\ \textit{inj-on } f\ \{x.\ P\ x\};\ P\ a;\ \forall\, y.\ P\ y \longrightarrow f\ a \leq f\ y\ ]\!] \Longrightarrow \textit{arg-min } f\ P = a$
$\langle \textit{proof} \rangle$

## 55.7 Arg Max

**context** *ord*
**begin**

**definition** $\textit{is-arg-max} :: ({}'b \Rightarrow {}'a) \Rightarrow ({}'b \Rightarrow \textit{bool}) \Rightarrow {}'b \Rightarrow \textit{bool}$ **where**
$\textit{is-arg-max } f\ P\ x = (P\ x \wedge \neg(\exists\, y.\ P\ y \wedge f\ y > f\ x))$

**definition** $\textit{arg-max} :: ({}'b \Rightarrow {}'a) \Rightarrow ({}'b \Rightarrow \textit{bool}) \Rightarrow {}'b$ **where**
$\textit{arg-max } f\ P = (\textit{SOME } x.\ \textit{is-arg-max } f\ P\ x)$

**definition** $\textit{arg-max-on} :: ({}'b \Rightarrow {}'a) \Rightarrow {}'b\ \textit{set} \Rightarrow {}'b$ **where**
$\textit{arg-max-on } f\ S = \textit{arg-max } f\ (\lambda x.\ x \in S)$

**end**

**syntax**
$\textit{-arg-max} :: ({}'b \Rightarrow {}'a) \Rightarrow \textit{pttrn} \Rightarrow \textit{bool} \Rightarrow {}'a$
(‹(‹indent=3 notation=‹binder ARG-MAX››ARG′-MAX - -./ -)› [1000, 0, 10] 10)
**syntax-consts**
$\textit{-arg-max} \rightleftharpoons \textit{arg-max}$
**translations**
$\textit{ARG-MAX } f\ x.\ P \rightleftharpoons \textit{CONST arg-max } f\ (\lambda x.\ P)$

**lemma** *is-arg-max-linorder*: **fixes** $f :: {}'a \Rightarrow {}'b :: \textit{linorder}$
**shows** $\textit{is-arg-max } f\ P\ x = (P\ x \wedge (\forall\, y.\ P\ y \longrightarrow f\ x \geq f\ y))$
$\langle \textit{proof} \rangle$

**lemma** *arg-maxI*:
$P\ x \Longrightarrow$
$(\bigwedge y.\ P\ y \Longrightarrow \neg\ f\ y > f\ x) \Longrightarrow$
$(\bigwedge x.\ P\ x \Longrightarrow \forall\, y.\ P\ y \longrightarrow \neg\ f\ y > f\ x \Longrightarrow Q\ x) \Longrightarrow$
$Q\ (\textit{arg-max } f\ P)$
$\langle \textit{proof} \rangle$

**lemma** *arg-max-equality*:
$[\![\ P\ k;\ \bigwedge x.\ P\ x \Longrightarrow f\ x \leq f\ k\ ]\!] \Longrightarrow f\ (\textit{arg-max } f\ P) = f\ k$

**for** $f$ :: - $\Rightarrow$ $'a$::*order*
⟨*proof*⟩

**lemma** *ex-has-greatest-nat-lemma*:
$P\ k \Longrightarrow \forall x.\ P\ x \longrightarrow (\exists y.\ P\ y \wedge \neg\ f\ y \leq f\ x) \Longrightarrow \exists y.\ P\ y \wedge \neg\ f\ y < f\ k + n$
**for** $f$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**lemma** *ex-has-greatest-nat*:
**assumes** $P\ k$
**and** $\forall y.\ P\ y \longrightarrow (f{::}\ 'a \Rightarrow nat)\ y < b$
**shows** $\exists x.\ P\ x \wedge (\forall y.\ P\ y \longrightarrow f\ y \leq f\ x)$
⟨*proof*⟩

**lemma** *arg-max-nat-lemma*:
$[\![\ P\ k;\ \ \forall y.\ P\ y \longrightarrow f\ y < b\ ]\!]$
$\Longrightarrow P\ (\textit{arg-max}\ f\ P) \wedge (\forall y.\ P\ y \longrightarrow f\ y \leq f\ (\textit{arg-max}\ f\ P))$
**for** $f$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**lemmas** *arg-max-natI* = *arg-max-nat-lemma* [*THEN conjunct1*]

**lemma** *arg-max-nat-le*: $P\ x \Longrightarrow \forall y.\ P\ y \longrightarrow f\ y < b \Longrightarrow f\ x \leq f\ (\textit{arg-max}\ f\ P)$
**for** $f$ :: $'a \Rightarrow nat$
⟨*proof*⟩

**end**

# 56 Division in euclidean (semi)rings

**theory** *Euclidean-Rings*
**imports** *Int Lattices-Big*
**begin**

## 56.1 Euclidean (semi)rings with explicit division and remainder

**class** *euclidean-semiring* = *semidom-modulo* +
**fixes** *euclidean-size* :: $'a \Rightarrow nat$
**assumes** *size-0* [*simp*]: *euclidean-size* $0 = 0$
**assumes** *mod-size-less*:
$b \neq 0 \Longrightarrow \textit{euclidean-size}\ (a\ mod\ b) < \textit{euclidean-size}\ b$
**assumes** *size-mult-mono*:
$b \neq 0 \Longrightarrow \textit{euclidean-size}\ a \leq \textit{euclidean-size}\ (a * b)$
**begin**

**lemma** *euclidean-size-eq-0-iff* [*simp*]:
$\textit{euclidean-size}\ b = 0 \longleftrightarrow b = 0$
⟨*proof*⟩

**lemma** *euclidean-size-greater-0-iff* [*simp*]:
  $euclidean\text{-}size\ b > 0 \longleftrightarrow b \neq 0$
  ⟨*proof*⟩

**lemma** *size-mult-mono'*: $b \neq 0 \Longrightarrow euclidean\text{-}size\ a \leq euclidean\text{-}size\ (b * a)$
  ⟨*proof*⟩

**lemma** *dvd-euclidean-size-eq-imp-dvd*:
  **assumes** $a \neq 0$ **and** $euclidean\text{-}size\ a = euclidean\text{-}size\ b$
    **and** $b\ dvd\ a$
  **shows** $a\ dvd\ b$
⟨*proof*⟩

**lemma** *euclidean-size-times-unit*:
  **assumes** $is\text{-}unit\ a$
  **shows** $euclidean\text{-}size\ (a * b) = euclidean\text{-}size\ b$
⟨*proof*⟩

**lemma** *euclidean-size-unit*:
  $is\text{-}unit\ a \Longrightarrow euclidean\text{-}size\ a = euclidean\text{-}size\ 1$
  ⟨*proof*⟩

**lemma** *unit-iff-euclidean-size*:
  $is\text{-}unit\ a \longleftrightarrow euclidean\text{-}size\ a = euclidean\text{-}size\ 1 \wedge a \neq 0$
⟨*proof*⟩

**lemma** *euclidean-size-times-nonunit*:
  **assumes** $a \neq 0$ $b \neq 0$ $\neg\ is\text{-}unit\ a$
  **shows** $euclidean\text{-}size\ b < euclidean\text{-}size\ (a * b)$
⟨*proof*⟩

**lemma** *dvd-imp-size-le*:
  **assumes** $a\ dvd\ b$ $b \neq 0$
  **shows** $euclidean\text{-}size\ a \leq euclidean\text{-}size\ b$
  ⟨*proof*⟩

**lemma** *dvd-proper-imp-size-less*:
  **assumes** $a\ dvd\ b$ $\neg\ b\ dvd\ a$ $b \neq 0$
  **shows** $euclidean\text{-}size\ a < euclidean\text{-}size\ b$
⟨*proof*⟩

**lemma** *unit-imp-mod-eq-0*:
  $a\ mod\ b = 0$ **if** $is\text{-}unit\ b$
  ⟨*proof*⟩

**lemma** *mod-eq-self-iff-div-eq-0*:
  $a\ mod\ b = a \longleftrightarrow a\ div\ b = 0$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *coprime-mod-left-iff* [*simp*]:
  *coprime* (*a mod b*) *b* $\longleftrightarrow$ *coprime a b* **if** $b \neq 0$
  ⟨*proof*⟩

**lemma** *coprime-mod-right-iff* [*simp*]:
  *coprime a* (*b mod a*) $\longleftrightarrow$ *coprime a b* **if** $a \neq 0$
  ⟨*proof*⟩

**end**

**class** *euclidean-ring* = *idom-modulo* + *euclidean-semiring*
**begin**

**lemma** *dvd-diff-commute* [*ac-simps*]:
  *a dvd c* − *b* $\longleftrightarrow$ *a dvd b* − *c*
⟨*proof*⟩

**end**

## 56.2 Euclidean (semi)rings with cancel rules

**class** *euclidean-semiring-cancel* = *euclidean-semiring* +
  **assumes** *div-mult-self1* [*simp*]: $b \neq 0 \Longrightarrow$ (*a* + *c* ∗ *b*) *div b* = *c* + *a div b*
  **and** *div-mult-mult1* [*simp*]: $c \neq 0 \Longrightarrow$ (*c* ∗ *a*) *div* (*c* ∗ *b*) = *a div b*
**begin**

**lemma** *div-mult-self2* [*simp*]:
  **assumes** $b \neq 0$
  **shows** (*a* + *b* ∗ *c*) *div b* = *c* + *a div b*
  ⟨*proof*⟩

**lemma** *div-mult-self3* [*simp*]:
  **assumes** $b \neq 0$
  **shows** (*c* ∗ *b* + *a*) *div b* = *c* + *a div b*
  ⟨*proof*⟩

**lemma** *div-mult-self4* [*simp*]:
  **assumes** $b \neq 0$
  **shows** (*b* ∗ *c* + *a*) *div b* = *c* + *a div b*
  ⟨*proof*⟩

**lemma** *mod-mult-self1* [*simp*]: (*a* + *c* ∗ *b*) *mod b* = *a mod b*
⟨*proof*⟩

**lemma** *mod-mult-self2* [*simp*]:
  (*a* + *b* ∗ *c*) *mod b* = *a mod b*
  ⟨*proof*⟩

**lemma** *mod-mult-self3* [*simp*]:
$(c * b + a)$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-mult-self4* [*simp*]:
$(b * c + a)$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-mult-self1-is-0* [*simp*]:
$b * a$ *mod* $b = 0$
⟨*proof*⟩

**lemma** *mod-mult-self2-is-0* [*simp*]:
$a * b$ *mod* $b = 0$
⟨*proof*⟩

**lemma** *div-add-self1*:
**assumes** $b \neq 0$
**shows** $(b + a)$ *div* $b = a$ *div* $b + 1$
⟨*proof*⟩

**lemma** *div-add-self2*:
**assumes** $b \neq 0$
**shows** $(a + b)$ *div* $b = a$ *div* $b + 1$
⟨*proof*⟩

**lemma** *mod-add-self1* [*simp*]:
$(b + a)$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-add-self2* [*simp*]:
$(a + b)$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-div-trivial* [*simp*]:
$a$ *mod* $b$ *div* $b = 0$
⟨*proof*⟩

**lemma** *mod-mod-trivial* [*simp*]:
$a$ *mod* $b$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-mod-cancel*:
**assumes** $c$ *dvd* $b$
**shows** $a$ *mod* $b$ *mod* $c = a$ *mod* $c$
⟨*proof*⟩

**lemma** *div-mult-mult2* [*simp*]:
$c \neq 0 \Longrightarrow (a * c)$ *div* $(b * c) = a$ *div* $b$

⟨*proof*⟩

**lemma** *div-mult-mult1-if* [*simp*]:
$(c * a)$ *div* $(c * b) = ($*if* $c = 0$ *then* $0$ *else* $a$ *div* $b)$
⟨*proof*⟩

**lemma** *mod-mult-mult1*:
$(c * a)$ *mod* $(c * b) = c * (a$ *mod* $b)$
⟨*proof*⟩

**lemma** *mod-mult-mult2*:
$(a * c)$ *mod* $(b * c) = (a$ *mod* $b) * c$
⟨*proof*⟩

**lemma** *mult-mod-left*: $(a$ *mod* $b) * c = (a * c)$ *mod* $(b * c)$
⟨*proof*⟩

**lemma** *mult-mod-right*: $c * (a$ *mod* $b) = (c * a)$ *mod* $(c * b)$
⟨*proof*⟩

**lemma** *dvd-mod*: $k$ *dvd* $m \Longrightarrow k$ *dvd* $n \Longrightarrow k$ *dvd* $(m$ *mod* $n)$
⟨*proof*⟩

**lemma** *div-plus-div-distrib-dvd-left*:
$c$ *dvd* $a \Longrightarrow (a + b)$ *div* $c = a$ *div* $c + b$ *div* $c$
⟨*proof*⟩

**lemma** *div-plus-div-distrib-dvd-right*:
$c$ *dvd* $b \Longrightarrow (a + b)$ *div* $c = a$ *div* $c + b$ *div* $c$
⟨*proof*⟩

**lemma** *sum-div-partition*:
‹$(\sum a\in A.\ f\ a)$ *div* $b = (\sum a\in A \cap \{a.\ b$ *dvd* $f\ a\}.\ f\ a$ *div* $b) + (\sum a\in A \cap \{a.\ \neg\ b$ *dvd* $f\ a\}.\ f\ a)$ *div* $b$›
**if** ‹*finite* $A$›
⟨*proof*⟩

**named-theorems** *mod-simps*

Addition respects modular equivalence.

**lemma** *mod-add-left-eq* [*mod-simps*]:
$(a$ *mod* $c + b)$ *mod* $c = (a + b)$ *mod* $c$
⟨*proof*⟩

**lemma** *mod-add-right-eq* [*mod-simps*]:
$(a + b$ *mod* $c)$ *mod* $c = (a + b)$ *mod* $c$
⟨*proof*⟩

**lemma** *mod-add-eq*:

$(a\ mod\ c + b\ mod\ c)\ mod\ c = (a + b)\ mod\ c$
⟨*proof*⟩

**lemma** *mod-sum-eq* [*mod-simps*]:
$(\sum i{\in}A.\ f\ i\ mod\ a)\ mod\ a = sum\ f\ A\ mod\ a$
⟨*proof*⟩

**lemma** *mod-add-cong*:
**assumes** $a\ mod\ c = a'\ mod\ c$
**assumes** $b\ mod\ c = b'\ mod\ c$
**shows** $(a + b)\ mod\ c = (a' + b')\ mod\ c$
⟨*proof*⟩

Multiplication respects modular equivalence.

**lemma** *mod-mult-left-eq* [*mod-simps*]:
$((a\ mod\ c) * b)\ mod\ c = (a * b)\ mod\ c$
⟨*proof*⟩

**lemma** *mod-mult-right-eq* [*mod-simps*]:
$(a * (b\ mod\ c))\ mod\ c = (a * b)\ mod\ c$
⟨*proof*⟩

**lemma** *mod-mult-eq*:
$((a\ mod\ c) * (b\ mod\ c))\ mod\ c = (a * b)\ mod\ c$
⟨*proof*⟩

**lemma** *mod-prod-eq* [*mod-simps*]:
$(\prod i{\in}A.\ f\ i\ mod\ a)\ mod\ a = prod\ f\ A\ mod\ a$
⟨*proof*⟩

**lemma** *mod-mult-cong*:
**assumes** $a\ mod\ c = a'\ mod\ c$
**assumes** $b\ mod\ c = b'\ mod\ c$
**shows** $(a * b)\ mod\ c = (a' * b')\ mod\ c$
⟨*proof*⟩

Exponentiation respects modular equivalence.

**lemma** *power-mod* [*mod-simps*]:
$((a\ mod\ b)\ \hat{}\ n)\ mod\ b = (a\ \hat{}\ n)\ mod\ b$
⟨*proof*⟩

**lemma** *power-diff-power-eq*:
‹$a\ \hat{}\ m\ div\ a\ \hat{}\ n = (if\ n \le m\ then\ a\ \hat{}\ (m - n)\ else\ 1\ div\ a\ \hat{}\ (n - m))$›
**if** ‹$a \neq 0$›
⟨*proof*⟩

**end**

**class** *euclidean-ring-cancel* = *euclidean-ring* + *euclidean-semiring-cancel*
**begin**

**subclass** *idom-divide* ⟨*proof*⟩

**lemma** *div-minus-minus* [*simp*]: $(- a)$ *div* $(- b) = a$ *div* $b$
⟨*proof*⟩

**lemma** *mod-minus-minus* [*simp*]: $(- a)$ *mod* $(- b) = - (a$ *mod* $b)$
⟨*proof*⟩

**lemma** *div-minus-right*: $a$ *div* $(- b) = (- a)$ *div* $b$
⟨*proof*⟩

**lemma** *mod-minus-right*: $a$ *mod* $(- b) = - ((- a)$ *mod* $b)$
⟨*proof*⟩

**lemma** *div-minus1-right* [*simp*]: $a$ *div* $(- 1) = - a$
⟨*proof*⟩

**lemma** *mod-minus1-right* [*simp*]: $a$ *mod* $(- 1) = 0$
⟨*proof*⟩

Negation respects modular equivalence.

**lemma** *mod-minus-eq* [*mod-simps*]:
$(- (a$ *mod* $b))$ *mod* $b = (- a)$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-minus-cong*:
**assumes** $a$ *mod* $b = a'$ *mod* $b$
**shows** $(- a)$ *mod* $b = (- a')$ *mod* $b$
⟨*proof*⟩

Subtraction respects modular equivalence.

**lemma** *mod-diff-left-eq* [*mod-simps*]:
$(a$ *mod* $c - b)$ *mod* $c = (a - b)$ *mod* $c$
⟨*proof*⟩

**lemma** *mod-diff-right-eq* [*mod-simps*]:
$(a - b$ *mod* $c)$ *mod* $c = (a - b)$ *mod* $c$
⟨*proof*⟩

**lemma** *mod-diff-eq*:
$(a$ *mod* $c - b$ *mod* $c)$ *mod* $c = (a - b)$ *mod* $c$
⟨*proof*⟩

**lemma** *mod-diff-cong*:
**assumes** $a$ *mod* $c = a'$ *mod* $c$
**assumes** $b$ *mod* $c = b'$ *mod* $c$

**shows** $(a - b)$ *mod* $c = (a' - b')$ *mod* $c$
⟨*proof*⟩

**lemma** *minus-mod-self2* [*simp*]:
$(a - b)$ *mod* $b = a$ *mod* $b$
⟨*proof*⟩

**lemma** *minus-mod-self1* [*simp*]:
$(b - a)$ *mod* $b = - a$ *mod* $b$
⟨*proof*⟩

**lemma** *mod-eq-dvd-iff*:
$a$ *mod* $c = b$ *mod* $c \longleftrightarrow c$ *dvd* $a - b$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *mod-eqE*:
**assumes** $a$ *mod* $c = b$ *mod* $c$
**obtains** $d$ **where** $b = a + c * d$
⟨*proof*⟩

**lemma** *invertible-coprime*:
*coprime* $a$ $c$ **if** $a * b$ *mod* $c = 1$
⟨*proof*⟩

**end**

## 56.3 Uniquely determined division

**class** *unique-euclidean-semiring* = *euclidean-semiring* +
**assumes** *euclidean-size-mult*: ‹*euclidean-size* $(a * b) =$ *euclidean-size* $a *$ *euclidean-size* $b$›
**fixes** *division-segment* :: ‹$'a \Rightarrow 'a$›
**assumes** *is-unit-division-segment* [*simp*]: ‹*is-unit* (*division-segment* $a$)›
**and** *division-segment-mult*:
‹$a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow$ *division-segment* $(a * b) =$ *division-segment* $a *$ *division-segment* $b$›
**and** *division-segment-mod*:
‹$b \neq 0 \Longrightarrow \neg\, b$ *dvd* $a \Longrightarrow$ *division-segment* $(a$ *mod* $b) =$ *division-segment* $b$›
**assumes** *div-bounded*:
‹$b \neq 0 \Longrightarrow$ *division-segment* $r =$ *division-segment* $b$
$\Longrightarrow$ *euclidean-size* $r <$ *euclidean-size* $b$
$\Longrightarrow (q * b + r)$ *div* $b = q$›
**begin**

**lemma** *division-segment-not-0* [*simp*]:
‹*division-segment* $a \neq 0$›
⟨*proof*⟩

**lemma** *euclidean-relationI* [*case-names by0 divides euclidean-relation*]:

‹(*a div b*, *a mod b*) = (*q*, *r*)›
**if** *by0*: ‹*b* = *0* ⟹ *q* = *0* ∧ *r* = *a*›
**and** *divides*: ‹*b* ≠ *0* ⟹ *b dvd a* ⟹ *r* = *0* ∧ *a* = *q* ∗ *b*›
**and** *euclidean-relation*: ‹*b* ≠ *0* ⟹ ¬ *b dvd a* ⟹ *division-segment r* = *division-segment b*
∧ *euclidean-size r* < *euclidean-size b* ∧ *a* = *q* ∗ *b* + *r*›
⟨*proof*⟩

**subclass** *euclidean-semiring-cancel*
⟨*proof*⟩

**lemma** *div-eq-0-iff*:
‹*a div b* = *0* ⟷ *euclidean-size a* < *euclidean-size b* ∨ *b* = *0*› (**is** - ⟷ *?P*)
**if** ‹*division-segment a* = *division-segment b*›
⟨*proof*⟩

**lemma** *div-mult1-eq*:
‹(*a* ∗ *b*) *div c* = *a* ∗ (*b div c*) + *a* ∗ (*b mod c*) *div c*›
⟨*proof*⟩

**lemma** *div-add1-eq*:
‹(*a* + *b*) *div c* = *a div c* + *b div c* + (*a mod c* + *b mod c*) *div c*›
⟨*proof*⟩

**end**

**class** *unique-euclidean-ring* = *euclidean-ring* + *unique-euclidean-semiring*
**begin**

**subclass** *euclidean-ring-cancel* ⟨*proof*⟩

**end**

## 56.4 Division on *nat*

**instantiation** *nat* :: *normalization-semidom*
**begin**

**definition** *normalize-nat* :: ‹*nat* ⇒ *nat*›
**where** [*simp*]: ‹*normalize* = (*id* :: *nat* ⇒ *nat*)›

**definition** *unit-factor-nat* :: ‹*nat* ⇒ *nat*›
**where** ‹*unit-factor n* = *of-bool* (*n* > *0*)› **for** *n* :: *nat*

**lemma** *unit-factor-simps* [*simp*]:
‹*unit-factor 0* = (*0*::*nat*)›
‹*unit-factor* (*Suc n*) = *1*›
⟨*proof*⟩

**definition** *divide-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
  **where** ‹$m$ *div* $n = ($*if* $n = 0$ *then* $0$ *else Max* $\{k.\ k * n \leq m\})$› **for** $m\ n$ :: *nat*

**instance**
  ⟨*proof*⟩

**end**

**lemma** *coprime-Suc-0-left* [*simp*]:
  *coprime* (*Suc 0*) *n*
  ⟨*proof*⟩

**lemma** *coprime-Suc-0-right* [*simp*]:
  *coprime n* (*Suc 0*)
  ⟨*proof*⟩

**lemma** *coprime-common-divisor-nat*: *coprime a b* $\Longrightarrow$ *x dvd a* $\Longrightarrow$ *x dvd b* $\Longrightarrow$ $x = 1$
  **for** $a\ b$ :: *nat*
  ⟨*proof*⟩

**instantiation** *nat* :: *unique-euclidean-semiring*
**begin**

**definition** *euclidean-size-nat* :: ‹*nat* ⇒ *nat*›
  **where** [*simp*]: ‹*euclidean-size-nat* = *id*›

**definition** *division-segment-nat* :: ‹*nat* ⇒ *nat*›
  **where** [*simp*]: ‹*division-segment* $n = 1$› **for** $n$ :: *nat*

**definition** *modulo-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
  **where** ‹$m$ *mod* $n = m - (m$ *div* $n * n)$› **for** $m\ n$ :: *nat*

**instance** ⟨*proof*⟩

**end**

**lemma** *euclidean-relation-natI* [*case-names by0 divides euclidean-relation*]:
  ‹($m$ *div* $n$, $m$ *mod* $n$) = ($q$, $r$)›
    **if** *by0*: ‹$n = 0 \Longrightarrow q = 0 \wedge r = m$›
    **and** *divides*: ‹$n > 0 \Longrightarrow n$ *dvd* $m \Longrightarrow r = 0 \wedge m = q * n$›
    **and** *euclidean-relation*: ‹$n > 0 \Longrightarrow \neg\ n$ *dvd* $m \Longrightarrow r < n \wedge m = q * n + r$›
**for** $m\ n\ q\ r$ :: *nat*
  ⟨*proof*⟩

**lemma** *div-nat-eqI*:
  ‹$m$ *div* $n = q$› **if** ‹$n * q \leq m$› **and** ‹$m < n *$ *Suc* $q$› **for** $m\ n\ q$ :: *nat*
⟨*proof*⟩

**lemma** *mod-nat-eqI*:
$\langle m\ mod\ n = r\rangle$ **if** $\langle r < n\rangle$ **and** $\langle r \leq m\rangle$ **and** $\langle n\ dvd\ m - r\rangle$ **for** $m\ n\ r :: nat$
$\langle proof \rangle$

Tool support

$\langle ML \rangle$

**lemma** *div-mult-self-is-m* [*simp*]:
$m * n\ div\ n = m$ **if** $n > 0$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *div-mult-self1-is-m* [*simp*]:
$n * m\ div\ n = m$ **if** $n > 0$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *mod-less-divisor* [*simp*]:
$m\ mod\ n < n$ **if** $n > 0$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *mod-le-divisor* [*simp*]:
$m\ mod\ n \leq n$ **if** $n > 0$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *div-times-less-eq-dividend* [*simp*]:
$m\ div\ n * n \leq m$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *times-div-less-eq-dividend* [*simp*]:
$n * (m\ div\ n) \leq m$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *dividend-less-div-times*:
$m < n + (m\ div\ n) * n$ **if** $0 < n$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *dividend-less-times-div*:
$m < n + n * (m\ div\ n)$ **if** $0 < n$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *mod-Suc-le-divisor* [*simp*]:
$m\ mod\ Suc\ n \leq n$
$\langle proof \rangle$

**lemma** *mod-less-eq-dividend* [*simp*]:
$m\ mod\ n \leq m$ **for** $m\ n :: nat$
$\langle proof \rangle$

**lemma**
*div-less* [*simp*]: $m\ div\ n = 0$

**and** *mod-less* [*simp*]: $m \ mod \ n = m$
**if** $m < n$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *split-div*:
‹$P \ (m \ div \ n) \longleftrightarrow$
$(n = 0 \longrightarrow P \ 0) \wedge$
$(n \neq 0 \longrightarrow (\forall i \ j. \ j < n \wedge m = n * i + j \longrightarrow P \ i))$› (**is** *?div*)
**and** *split-mod*:
‹$Q \ (m \ mod \ n) \longleftrightarrow$
$(n = 0 \longrightarrow Q \ m) \wedge$
$(n \neq 0 \longrightarrow (\forall i \ j. \ j < n \wedge m = n * i + j \longrightarrow Q \ j))$› (**is** *?mod*)
**for** $m \ n :: nat$
⟨*proof*⟩

**declare** *split-div* [*of - -* ‹*numeral n*›, *linarith-split*] **for** *n*
**declare** *split-mod* [*of - -* ‹*numeral n*›, *linarith-split*] **for** *n*

**lemma** *split-div'*:
$P \ (m \ div \ n) \longleftrightarrow n = 0 \wedge P \ 0 \vee (\exists q. \ (n * q \leq m \wedge m < n * Suc \ q) \wedge P \ q)$
⟨*proof*⟩

**lemma** *le-div-geq*:
$m \ div \ n = Suc \ ((m - n) \ div \ n)$ **if** $0 < n$ **and** $n \leq m$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *le-mod-geq*:
$m \ mod \ n = (m - n) \ mod \ n$ **if** $n \leq m$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *div-if*:
$m \ div \ n = (if \ m < n \vee n = 0 \ then \ 0 \ else \ Suc \ ((m - n) \ div \ n))$
⟨*proof*⟩

**lemma** *mod-if*:
$m \ mod \ n = (if \ m < n \ then \ m \ else \ (m - n) \ mod \ n)$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *div-eq-0-iff*:
$m \ div \ n = 0 \longleftrightarrow m < n \vee n = 0$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *div-greater-zero-iff*:
$m \ div \ n > 0 \longleftrightarrow n \leq m \wedge n > 0$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *mod-greater-zero-iff-not-dvd*:
$m \ mod \ n > 0 \longleftrightarrow \neg \ n \ dvd \ m$ **for** $m \ n :: nat$
⟨*proof*⟩

**lemma** *div-by-Suc-0* [*simp*]:
$m\ div\ Suc\ 0 = m$
⟨*proof*⟩

**lemma** *mod-by-Suc-0* [*simp*]:
$m\ mod\ Suc\ 0 = 0$
⟨*proof*⟩

**lemma** *div2-Suc-Suc* [*simp*]:
$Suc\ (Suc\ m)\ div\ 2 = Suc\ (m\ div\ 2)$
⟨*proof*⟩

**lemma** *Suc-n-div-2-gt-zero* [*simp*]:
$0 < Suc\ n\ div\ 2$ **if** $n > 0$ **for** $n :: nat$
⟨*proof*⟩

**lemma** *div-2-gt-zero* [*simp*]:
$0 < n\ div\ 2$ **if** $Suc\ 0 < n$ **for** $n :: nat$
⟨*proof*⟩

**lemma** *mod2-Suc-Suc* [*simp*]:
$Suc\ (Suc\ m)\ mod\ 2 = m\ mod\ 2$
⟨*proof*⟩

**lemma** *add-self-div-2* [*simp*]:
$(m + m)\ div\ 2 = m$ **for** $m :: nat$
⟨*proof*⟩

**lemma** *add-self-mod-2* [*simp*]:
$(m + m)\ mod\ 2 = 0$ **for** $m :: nat$
⟨*proof*⟩

**lemma** *mod2-gr-0* [*simp*]:
$0 < m\ mod\ 2 \longleftrightarrow m\ mod\ 2 = 1$ **for** $m :: nat$
⟨*proof*⟩

**lemma** *mod-Suc-eq* [*mod-simps*]:
$Suc\ (m\ mod\ n)\ mod\ n = Suc\ m\ mod\ n$
⟨*proof*⟩

**lemma** *mod-Suc-Suc-eq* [*mod-simps*]:
$Suc\ (Suc\ (m\ mod\ n))\ mod\ n = Suc\ (Suc\ m)\ mod\ n$
⟨*proof*⟩

**lemma**
*Suc-mod-mult-self1* [*simp*]: $Suc\ (m + k * n)\ mod\ n = Suc\ m\ mod\ n$
**and** *Suc-mod-mult-self2* [*simp*]: $Suc\ (m + n * k)\ mod\ n = Suc\ m\ mod\ n$
**and** *Suc-mod-mult-self3* [*simp*]: $Suc\ (k * n + m)\ mod\ n = Suc\ m\ mod\ n$

**and** *Suc-mod-mult-self4* [*simp*]: $Suc\ (n * k + m)\ mod\ n = Suc\ m\ mod\ n$
⟨*proof*⟩

**lemma** *Suc-0-mod-eq* [*simp*]:
$Suc\ 0\ mod\ n = of\text{-}bool\ (n \neq Suc\ 0)$
⟨*proof*⟩

**lemma** *div-mult2-eq*:
‹$m\ div\ (n * q) = (m\ div\ n)\ div\ q$› (**is** *?Q*)
**and** *mod-mult2-eq*:
‹$m\ mod\ (n * q) = n * (m\ div\ n\ mod\ q) + m\ mod\ n$› (**is** *?R*)
**for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *div-le-mono*:
$m\ div\ k \leq n\ div\ k$ **if** $m \leq n$ **for** $m\ n\ k :: nat$
⟨*proof*⟩

Antimonotonicity of (*div*) in second argument

**lemma** *div-le-mono2*:
$k\ div\ n \leq k\ div\ m$ **if** $0 < m$ **and** $m \leq n$ **for** $m\ n\ k :: nat$
⟨*proof*⟩

**lemma** *div-le-dividend* [*simp*]:
$m\ div\ n \leq m$ **for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *div-less-dividend* [*simp*]:
$m\ div\ n < m$ **if** $1 < n$ **and** $0 < m$ **for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *div-eq-dividend-iff*:
$m\ div\ n = m \longleftrightarrow n = 1$ **if** $m > 0$ **for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *less-mult-imp-div-less*:
$m\ div\ n < i$ **if** $m < i * n$ **for** $m\ n\ i :: nat$
⟨*proof*⟩

**lemma** *div-less-iff-less-mult*:
‹$m\ div\ q < n \longleftrightarrow m < n * q$› (**is** ‹$?P \longleftrightarrow ?Q$›)
**if** ‹$q > 0$› **for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *less-eq-div-iff-mult-less-eq*:
‹$m \leq n\ div\ q \longleftrightarrow m * q \leq n$› **if** ‹$q > 0$› **for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *div-Suc*:

‹$Suc\ m\ div\ n = (if\ Suc\ m\ mod\ n = 0\ then\ Suc\ (m\ div\ n)\ else\ m\ div\ n)$›
⟨*proof*⟩

**lemma** *mod-Suc*:
‹$Suc\ m\ mod\ n = (if\ Suc\ (m\ mod\ n) = n\ then\ 0\ else\ Suc\ (m\ mod\ n))$›
⟨*proof*⟩

**lemma** *Suc-times-mod-eq*:
$Suc\ (m * n)\ mod\ m = 1$ **if** $Suc\ 0 < m$
⟨*proof*⟩

**lemma** *Suc-times-numeral-mod-eq* [*simp*]:
$Suc\ (numeral\ k * n)\ mod\ numeral\ k = 1$ **if** $numeral\ k \neq (1::nat)$
⟨*proof*⟩

**lemma** *Suc-div-le-mono* [*simp*]:
$m\ div\ n \leq Suc\ m\ div\ n$
⟨*proof*⟩

These lemmas collapse some needless occurrences of Suc: at least three Sucs, since two and fewer are rewritten back to Suc again! We already have some rules to simplify operands smaller than 3.

**lemma** *div-Suc-eq-div-add3* [*simp*]:
$m\ div\ Suc\ (Suc\ (Suc\ n)) = m\ div\ (3 + n)$
⟨*proof*⟩

**lemma** *mod-Suc-eq-mod-add3* [*simp*]:
$m\ mod\ Suc\ (Suc\ (Suc\ n)) = m\ mod\ (3 + n)$
⟨*proof*⟩

**lemma** *Suc-div-eq-add3-div*:
$Suc\ (Suc\ (Suc\ m))\ div\ n = (3 + m)\ div\ n$
⟨*proof*⟩

**lemma** *Suc-mod-eq-add3-mod*:
$Suc\ (Suc\ (Suc\ m))\ mod\ n = (3 + m)\ mod\ n$
⟨*proof*⟩

**lemmas** *Suc-div-eq-add3-div-numeral* [*simp*] =
*Suc-div-eq-add3-div* [*of - numeral v*] **for** *v*

**lemmas** *Suc-mod-eq-add3-mod-numeral* [*simp*] =
*Suc-mod-eq-add3-mod* [*of - numeral v*] **for** *v*

**lemma** (**in** *field-char-0*) *of-nat-div*:
$of\text{-}nat\ (m\ div\ n) = ((of\text{-}nat\ m - of\text{-}nat\ (m\ mod\ n)) / of\text{-}nat\ n)$
⟨*proof*⟩

An "induction" law for modulus arithmetic.

**lemma** *mod-induct* [*consumes 3*, *case-names step*]:
$P\ m$ **if** $P\ n$ **and** $n < p$ **and** $m < p$
**and** *step*: $\bigwedge n.\ n < p \Longrightarrow P\ n \Longrightarrow P\ (Suc\ n\ mod\ p)$
⟨*proof*⟩

**lemma** *funpow-mod-eq*:
‹$(f \ \widehat{}\widehat{}\ (m\ mod\ n))\ x = (f \ \widehat{}\widehat{}\ m)\ x$› **if** ‹$(f \ \widehat{}\widehat{}\ n)\ x = x$›
⟨*proof*⟩

**lemma** *mod-eq-dvd-iff-nat*:
‹$m\ mod\ q = n\ mod\ q \longleftrightarrow q\ dvd\ m - n$› (**is** ‹$?P \longleftrightarrow ?Q$›)
**if** ‹$m \geq n$› **for** $m\ n\ q :: nat$
⟨*proof*⟩

**lemma** *mod-eq-iff-dvd-symdiff-nat*:
‹$m\ mod\ q = n\ mod\ q \longleftrightarrow q\ dvd\ nat\ |int\ m - int\ n|$›
⟨*proof*⟩

**lemma** *mod-eq-nat1E*:
**fixes** $m\ n\ q :: nat$
**assumes** $m\ mod\ q = n\ mod\ q$ **and** $m \geq n$
**obtains** $s$ **where** $m = n + q * s$
⟨*proof*⟩

**lemma** *mod-eq-nat2E*:
**fixes** $m\ n\ q :: nat$
**assumes** $m\ mod\ q = n\ mod\ q$ **and** $n \geq m$
**obtains** $s$ **where** $n = m + q * s$
⟨*proof*⟩

**lemma** *nat-mod-eq-iff*:
$(x::nat)\ mod\ n = y\ mod\ n \longleftrightarrow (\exists\, q1\ q2.\ x + n * q1 = y + n * q2)$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

## 56.5 Division on *int*

The following specification of integer division rounds towards minus infinity and is advocated by Donald Knuth. See [5] for an overview and terminology of different possibilities to specify integer division; there division rounding towards minus infinitiy is named "F-division".

### 56.5.1 Basic instantiation

**instantiation** *int* :: {*normalization-semidom*, *idom-modulo*}
**begin**

**definition** *normalize-int* :: ‹$int \Rightarrow int$›
**where** [*simp*]: ‹$normalize = (abs :: int \Rightarrow int)$›

**definition** *unit-factor-int* :: ‹$int \Rightarrow int$›
**where** [*simp*]: ‹$unit\text{-}factor = (sgn :: int \Rightarrow int)$›

**definition** *divide-int* :: ‹$int \Rightarrow int \Rightarrow int$›
**where** ‹$k\ div\ l = (sgn\ k * sgn\ l * int\ (nat\ |k|\ div\ nat\ |l|)$
$-\ of\text{-}bool\ (l \neq 0 \wedge sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k))$›

**lemma** *divide-int-unfold*:
‹$(sgn\ k * int\ m)\ div\ (sgn\ l * int\ n) = (sgn\ k * sgn\ l * int\ (m\ div\ n)$
$-\ of\text{-}bool\ ((k = 0 \longleftrightarrow m = 0) \wedge l \neq 0 \wedge n \neq 0 \wedge sgn\ k \neq sgn\ l \wedge \neg\ n\ dvd\ m))$›
⟨*proof*⟩

**definition** *modulo-int* :: ‹$int \Rightarrow int \Rightarrow int$›
**where** ‹$k\ mod\ l = sgn\ k * int\ (nat\ |k|\ mod\ nat\ |l|) + l * of\text{-}bool\ (sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k)$›

**lemma** *modulo-int-unfold*:
‹$(sgn\ k * int\ m)\ mod\ (sgn\ l * int\ n) =$
$sgn\ k * int\ (m\ mod\ (of\text{-}bool\ (l \neq 0) * n)) + (sgn\ l * int\ n) * of\text{-}bool\ ((k = 0 \longleftrightarrow m = 0) \wedge sgn\ k \neq sgn\ l \wedge \neg\ n\ dvd\ m)$›
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**lemma** *of-int-div*: $b\ dvd\ a \Longrightarrow of\text{-}int\ (a\ div\ b) = (of\text{-}int\ a\ /\ of\text{-}int\ b :: 'a :: field\text{-}char\text{-}0)$
⟨*proof*⟩

### 56.5.2 Algebraic foundations

**lemma** *coprime-int-iff* [*simp*]:
$coprime\ (int\ m)\ (int\ n) \longleftrightarrow coprime\ m\ n$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *coprime-abs-left-iff* [*simp*]:
$coprime\ |k|\ l \longleftrightarrow coprime\ k\ l$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *coprime-abs-right-iff* [*simp*]:
$coprime\ k\ |l| \longleftrightarrow coprime\ k\ l$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *coprime-nat-abs-left-iff* [*simp*]:
$coprime\ (nat\ |k|)\ n \longleftrightarrow coprime\ k\ (int\ n)$
⟨*proof*⟩

**lemma** *coprime-nat-abs-right-iff* [*simp*]:
$coprime\ n\ (nat\ |k|) \longleftrightarrow coprime\ (int\ n)\ k$
⟨*proof*⟩

**lemma** *coprime-common-divisor-int*: $coprime\ a\ b \Longrightarrow x\ dvd\ a \Longrightarrow x\ dvd\ b \Longrightarrow |x| = 1$
**for** $a\ b :: int$
⟨*proof*⟩

### 56.5.3 Basic conversions

**lemma** *div-abs-eq-div-nat*:
$|k|\ div\ |l| = int\ (nat\ |k|\ div\ nat\ |l|)$
⟨*proof*⟩

**lemma** *div-eq-div-abs*:
‹$k\ div\ l = sgn\ k * sgn\ l * (|k|\ div\ |l|)$
$- of\text{-}bool\ (l \neq 0 \wedge sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k)$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *div-abs-eq*:
‹$|k|\ div\ |l| = sgn\ k * sgn\ l * (k\ div\ l + of\text{-}bool\ (sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k))$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *mod-abs-eq-div-nat*:
$|k|\ mod\ |l| = int\ (nat\ |k|\ mod\ nat\ |l|)$
⟨*proof*⟩

**lemma** *mod-eq-mod-abs*:
‹$k\ mod\ l = sgn\ k * (|k|\ mod\ |l|) + l * of\text{-}bool\ (sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k)$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *mod-abs-eq*:
‹$|k|\ mod\ |l| = sgn\ k * (k\ mod\ l - l * of\text{-}bool\ (sgn\ k \neq sgn\ l \wedge \neg\ l\ dvd\ k))$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *div-sgn-abs-cancel*:
**fixes** $k\ l\ v :: int$
**assumes** $v \neq 0$
**shows** $(sgn\ v * |k|)\ div\ (sgn\ v * |l|) = |k|\ div\ |l|$
⟨*proof*⟩

**lemma** *div-eq-sgn-abs*:
**fixes** $k\ l\ v :: int$

**assumes** *sgn k = sgn l*
**shows** *k div l* = $|k|$ *div* $|l|$
⟨*proof*⟩

**lemma** *div-dvd-sgn-abs*:
**fixes** *k l* :: *int*
**assumes** *l dvd k*
**shows** *k div l* = (*sgn k* ∗ *sgn l*) ∗ ($|k|$ *div* $|l|$)
⟨*proof*⟩

**lemma** *div-noneq-sgn-abs*:
**fixes** *k l* :: *int*
**assumes** $l \neq 0$
**assumes** *sgn k* ≠ *sgn l*
**shows** *k div l* = − ($|k|$ *div* $|l|$) − *of-bool* (¬ *l dvd k*)
⟨*proof*⟩

### 56.5.4 Euclidean division

**instantiation** *int* :: *unique-euclidean-ring*
**begin**

**definition** *euclidean-size-int* :: *int* ⇒ *nat*
**where** [*simp*]: *euclidean-size-int* = (*nat* ∘ *abs* :: *int* ⇒ *nat*)

**definition** *division-segment-int* :: *int* ⇒ *int*
**where** *division-segment-int k* = (*if* $k \geq 0$ *then 1 else* − *1*)

**lemma** *division-segment-eq-sgn*:
*division-segment k* = *sgn k* **if** $k \neq 0$ **for** *k* :: *int*
⟨*proof*⟩

**lemma** *abs-division-segment* [*simp*]:
|*division-segment k*| = *1* **for** *k* :: *int*
⟨*proof*⟩

**lemma** *abs-mod-less*:
|*k mod l*| < $|l|$ **if** $l \neq 0$ **for** *k l* :: *int*
⟨*proof*⟩

**lemma** *sgn-mod*:
*sgn* (*k mod l*) = *sgn l* **if** $l \neq 0$ ¬ *l dvd k* **for** *k l* :: *int*
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**lemma** *euclidean-relation-intI* [*case-names by0 divides euclidean-relation*]:

‹$(k\ div\ l,\ k\ mod\ l) = (q,\ r)$›
**if** *by0'*: ‹$l = 0 \Longrightarrow q = 0 \wedge r = k$›
**and** *divides'*: ‹$l \neq 0 \Longrightarrow l\ dvd\ k \Longrightarrow r = 0 \wedge k = q * l$›
**and** *euclidean-relation'*: ‹$l \neq 0 \Longrightarrow \neg\ l\ dvd\ k \Longrightarrow sgn\ r = sgn\ l$
$\wedge |r| < |l| \wedge k = q * l + r$› **for** $k\ l :: int$
⟨*proof*⟩

### 56.5.5 Trivial reduction steps

**lemma** *div-pos-pos-trivial* [*simp*]:
$k\ div\ l = 0$ **if** $k \geq 0$ **and** $k < l$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *mod-pos-pos-trivial* [*simp*]:
$k\ mod\ l = k$ **if** $k \geq 0$ **and** $k < l$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *div-neg-neg-trivial* [*simp*]:
$k\ div\ l = 0$ **if** $k \leq 0$ **and** $l < k$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *mod-neg-neg-trivial* [*simp*]:
$k\ mod\ l = k$ **if** $k \leq 0$ **and** $l < k$ **for** $k\ l :: int$
⟨*proof*⟩

**lemma**
*div-pos-neg-trivial*: ‹$k\ div\ l = -\ 1$› (**is** *?Q*)
**and** *mod-pos-neg-trivial*: ‹$k\ mod\ l = k + l$› (**is** *?R*)
**if** ‹$0 < k$› **and** ‹$k + l \leq 0$› **for** $k\ l :: int$
⟨*proof*⟩

There is neither *div-neg-pos-trivial* nor *mod-neg-pos-trivial* because $0\ div\ l = 0$ would supersede it.

### 56.5.6 More uniqueness rules

**lemma**
**fixes** $a\ b\ q\ r :: int$
**assumes** ‹$a = b * q + r$› ‹$0 \leq r$› ‹$r < b$›
**shows** *int-div-pos-eq*:
‹$a\ div\ b = q$› (**is** *?Q*)
**and** *int-mod-pos-eq*:
‹$a\ mod\ b = r$› (**is** *?R*)
⟨*proof*⟩

**lemma** *int-div-neg-eq*:
‹$a\ div\ b = q$› **if** ‹$a = b * q + r$› ‹$r \leq 0$› ‹$b < r$› **for** $a\ b\ q\ r :: int$
⟨*proof*⟩

**lemma** *int-mod-neg-eq*:
‹$a$ *mod* $b = r$› **if** ‹$a = b * q + r$› ‹$r \leq 0$› ‹$b < r$› **for** $a$ $b$ $q$ $r$ :: *int*
⟨*proof*⟩

### 56.5.7 Laws for unary minus

**lemma** *zmod-zminus1-not-zero*:
**fixes** $k$ $l$ :: *int*
**shows** $-$ $k$ *mod* $l \neq 0 \Longrightarrow k$ *mod* $l \neq 0$
⟨*proof*⟩

**lemma** *zmod-zminus2-not-zero*:
**fixes** $k$ $l$ :: *int*
**shows** $k$ *mod* $-$ $l \neq 0 \Longrightarrow k$ *mod* $l \neq 0$
⟨*proof*⟩

**lemma** *zdiv-zminus1-eq-if*:
‹$(-\ a)$ *div* $b =$ (*if* $a$ *mod* $b = 0$ *then* $-$ ($a$ *div* $b$) *else* $-$ ($a$ *div* $b$) $-$ $1$)›
**if** ‹$b \neq 0$› **for** $a$ $b$ :: *int*
⟨*proof*⟩

**lemma** *zdiv-zminus2-eq-if*:
‹$a$ *div* $(-\ b) =$ (*if* $a$ *mod* $b = 0$ *then* $-$ ($a$ *div* $b$) *else* $-$ ($a$ *div* $b$) $-$ $1$)›
**if** ‹$b \neq 0$› **for** $a$ $b$ :: *int*
⟨*proof*⟩

**lemma** *zmod-zminus1-eq-if*:
‹$(-\ a)$ *mod* $b =$ (*if* $a$ *mod* $b = 0$ *then* $0$ *else* $b$ $-$ ($a$ *mod* $b$))›
**for** $a$ $b$ :: *int*
⟨*proof*⟩

**lemma** *zmod-zminus2-eq-if*:
‹$a$ *mod* $(-\ b) =$ (*if* $a$ *mod* $b = 0$ *then* $0$ *else* ($a$ *mod* $b$) $-$ $b$)›
**for** $a$ $b$ :: *int*
⟨*proof*⟩

### 56.5.8 Borders

**lemma** *pos-mod-bound* [*simp*]:
$k$ *mod* $l < l$ **if** $l > 0$ **for** $k$ $l$ :: *int*
⟨*proof*⟩

**lemma** *neg-mod-bound* [*simp*]:
$l < k$ *mod* $l$ **if** $l < 0$ **for** $k$ $l$ :: *int*
⟨*proof*⟩

**lemma** *pos-mod-sign* [*simp*]:
$0 \leq k$ *mod* $l$ **if** $l > 0$ **for** $k$ $l$ :: *int*
⟨*proof*⟩

**lemma** *neg-mod-sign* [*simp*]:
$k$ *mod* $l \leq 0$ **if** $l < 0$ **for** $k\ l :: int$
⟨*proof*⟩

### 56.5.9 Splitting Rules for div and mod

**lemma** *split-zdiv*:
‹$P\ (n\ div\ k) \longleftrightarrow$
$(k = 0 \longrightarrow P\ 0) \wedge$
$(0 < k \longrightarrow (\forall i\ j.\ 0 \leq j \wedge j < k \wedge n = k * i + j \longrightarrow P\ i)) \wedge$
$(k < 0 \longrightarrow (\forall i\ j.\ k < j \wedge j \leq 0 \wedge n = k * i + j \longrightarrow P\ i))$› (**is** *?div*)
**and** *split-zmod*:
‹$Q\ (n\ mod\ k) \longleftrightarrow$
$(k = 0 \longrightarrow Q\ n) \wedge$
$(0 < k \longrightarrow (\forall i\ j.\ 0 \leq j \wedge j < k \wedge n = k * i + j \longrightarrow Q\ j)) \wedge$
$(k < 0 \longrightarrow (\forall i\ j.\ k < j \wedge j \leq 0 \wedge n = k * i + j \longrightarrow Q\ j))$› (**is** *?mod*)
**for** $n\ k :: int$
⟨*proof*⟩

Enable (lin)arith to deal with (*div*) and (*mod*) when these are applied to some constant that is of the form *numeral k*:

**declare** *split-zdiv* [*of* - - ‹*numeral n*›, *linarith-split*] **for** *n*
**declare** *split-zdiv* [*of* - - ‹− *numeral n*›, *linarith-split*] **for** *n*
**declare** *split-zmod* [*of* - - ‹*numeral n*›, *linarith-split*] **for** *n*
**declare** *split-zmod* [*of* - - ‹− *numeral n*›, *linarith-split*] **for** *n*

**lemma** *zdiv-eq-0-iff*:
$i\ div\ k = 0 \longleftrightarrow k = 0 \vee 0 \leq i \wedge i < k \vee i \leq 0 \wedge k < i$ (**is** $?L = ?R$)
**for** $i\ k :: int$
⟨*proof*⟩

**lemma** *zmod-trivial-iff*:
**fixes** $i\ k :: int$
**shows** $i\ mod\ k = i \longleftrightarrow k = 0 \vee 0 \leq i \wedge i < k \vee i \leq 0 \wedge k < i$
⟨*proof*⟩

### 56.5.10 Algebraic rewrites

**lemma** *zdiv-zmult2-eq*: ‹$a\ div\ (b * c) = (a\ div\ b)\ div\ c$› (**is** *?Q*)
**and** *zmod-zmult2-eq*: ‹$a\ mod\ (b * c) = b * (a\ div\ b\ mod\ c) + a\ mod\ b$› (**is** *?P*)
**if** ‹$c \geq 0$› **for** $a\ b\ c :: int$
⟨*proof*⟩

**lemma** *zdiv-zmult2-eq'*:
‹$k\ div\ (l * j) = ((sgn\ j * k)\ div\ l)\ div\ |j|$› **for** $k\ l\ j :: int$
⟨*proof*⟩

**lemma** *half-nonnegative-int-iff* [*simp*]:
‹$k\ div\ 2 \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$

⟨*proof*⟩

**lemma** *half-negative-int-iff* [*simp*]:
‹$k$ *div* $2 < 0 \longleftrightarrow k < 0$› **for** $k$ :: *int*
⟨*proof*⟩

### 56.5.11 Distributive laws for conversions.

**lemma** *zdiv-int*:
‹*int* ($m$ *div* $n$) = *int* $m$ *div* *int* $n$›
⟨*proof*⟩

**lemma** *zmod-int*:
‹*int* ($m$ *mod* $n$) = *int* $m$ *mod* *int* $n$›
⟨*proof*⟩

**lemma** *nat-div-distrib*:
‹*nat* ($x$ *div* $y$) = *nat* $x$ *div* *nat* $y$› **if** ‹$0 \leq x$›
⟨*proof*⟩

**lemma** *nat-div-distrib'*:
‹*nat* ($x$ *div* $y$) = *nat* $x$ *div* *nat* $y$› **if** ‹$0 \leq y$›
⟨*proof*⟩

**lemma** *nat-mod-distrib*: — Fails if y<0: the LHS collapses to (nat z) but the RHS doesn't
‹*nat* ($x$ *mod* $y$) = *nat* $x$ *mod* *nat* $y$› **if** ‹$0 \leq x$› ‹$0 \leq y$›
⟨*proof*⟩

### 56.5.12 Monotonicity in the First Argument (Dividend)

**lemma** *zdiv-mono1*:
‹$a$ *div* $b \leq a'$ *div* $b$›
**if** ‹$a \leq a'$› ‹$0 < b$›
**for** $a$ $b$ $b'$ :: *int*
⟨*proof*⟩

**lemma** *zdiv-mono1-neg*:
‹$a'$ *div* $b \leq a$ *div* $b$›
**if** ‹$a \leq a'$› ‹$b < 0$›
**for** $a$ $a'$ $b$ :: *int*
⟨*proof*⟩

### 56.5.13 Monotonicity in the Second Argument (Divisor)

**lemma** *zdiv-mono2*:
‹$a$ *div* $b \leq a$ *div* $b'$› **if** ‹$0 \leq a$› ‹$0 < b'$› ‹$b' \leq b$› **for** $a$ $b$ $b'$ :: *int*
⟨*proof*⟩

**lemma** *zdiv-mono2-neg*:

‹$a$ *div* $b' \leq a$ *div* $b$› **if** ‹$a < 0$› ‹$0 < b'$› ‹$b' \leq b$› **for** $a\ b\ b'$ :: *int*
⟨*proof*⟩

### 56.5.14 Quotients of Signs

**lemma** *div-eq-minus1*:
‹$0 < b \Longrightarrow - 1$ *div* $b = - 1$› **for** $b$ :: *int*
⟨*proof*⟩

**lemma** *zmod-minus1*:
‹$0 < b \Longrightarrow - 1$ *mod* $b = b - 1$› **for** $b$ :: *int*
⟨*proof*⟩

**lemma** *minus-mod-int-eq*:
‹$- k$ *mod* $l = l - 1 - (k - 1)$ *mod* $l$› **if** ‹$l \geq 0$› **for** $k\ l$ :: *int*
⟨*proof*⟩

**lemma** *div-neg-pos-less0*:
‹$a$ *div* $b < 0$› **if** ‹$a < 0$› ‹$0 < b$› **for** $a\ b$ :: *int*
⟨*proof*⟩

**lemma** *div-nonneg-neg-le0*:
‹$a$ *div* $b \leq 0$› **if** ‹$0 \leq a$› ‹$b < 0$› **for** $a\ b$ :: *int*
⟨*proof*⟩

**lemma** *div-nonpos-pos-le0*:
‹$a$ *div* $b \leq 0$› **if** ‹$a \leq 0$› ‹$0 < b$› **for** $a\ b$ :: *int*
⟨*proof*⟩

Now for some equivalences of the form $a$ *div* $b$ >=< $0 \longleftrightarrow \ldots$ conditional upon the sign of $a$ or $b$. There are many more. They should all be simp rules unless that causes too much search.

**lemma** *pos-imp-zdiv-nonneg-iff*:
‹$0 \leq a$ *div* $b \longleftrightarrow 0 \leq a$›
**if** ‹$0 < b$› **for** $a\ b$ :: *int*
⟨*proof*⟩

**lemma** *neg-imp-zdiv-nonneg-iff*:
‹$0 \leq a$ *div* $b \longleftrightarrow a \leq 0$› **if** ‹$b < 0$› **for** $a\ b$ :: *int*
⟨*proof*⟩

**lemma** *pos-imp-zdiv-pos-iff*:
‹$0 < (i{::}int)$ *div* $k \longleftrightarrow k \leq i$› **if** ‹$0 < k$› **for** $i\ k$ :: *int*
⟨*proof*⟩

**lemma** *pos-imp-zdiv-neg-iff*:
‹$a$ *div* $b < 0 \longleftrightarrow a < 0$› **if** ‹$0 < b$› **for** $a\ b$ :: *int*
— But not ($a$ *div* $b \leq 0$) = ($a \leq 0$); consider $a = 1$, $b = 2$ when $a$ *div* $b = 0$.
⟨*proof*⟩

**lemma** *neg-imp-zdiv-neg-iff*:
— But not $(a\ div\ b \leq 0) = (0 \leq a)$; consider $a = -1$, $b = -2$ when $a\ div\ b = 0$.
‹$a\ div\ b < 0 \longleftrightarrow 0 < a$› **if** ‹$b < 0$› **for** $a\ b :: int$
⟨*proof*⟩

**lemma** *nonneg1-imp-zdiv-pos-iff*:
‹$a\ div\ b > 0 \longleftrightarrow a \geq b \wedge b > 0$› **if** ‹$0 \leq a$› **for** $a\ b :: int$
⟨*proof*⟩

**lemma** *zmod-le-nonneg-dividend*:
‹$m\ mod\ k \leq m$› **if** ‹$(m::int) \geq 0$› **for** $m\ k :: int$
⟨*proof*⟩

**lemma** *sgn-div-eq-sgn-mult*:
‹$sgn\ (k\ div\ l) = of\text{-}bool\ (k\ div\ l \neq 0) * sgn\ (k * l)$›
**for** $k\ l :: int$
⟨*proof*⟩

### 56.5.15 Further properties

**lemma** *div-int-pos-iff*:
$k\ div\ l \geq 0 \longleftrightarrow k = 0 \vee l = 0 \vee k \geq 0 \wedge l \geq 0$
$\vee\ k < 0 \wedge l < 0$
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *mod-int-pos-iff*:
‹$k\ mod\ l \geq 0 \longleftrightarrow l\ dvd\ k \vee l = 0 \wedge k \geq 0 \vee l > 0$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *abs-div*:
‹$|x\ div\ y| = |x|\ div\ |y|$› **if** ‹$y\ dvd\ x$› **for** $x\ y :: int$
⟨*proof*⟩

**lemma** *int-power-div-base*:
‹$k\ \hat{}\ m\ div\ k = k\ \hat{}\ (m - Suc\ 0)$› **if** ‹$0 < m$› ‹$0 < k$› **for** $k :: int$
⟨*proof*⟩

**lemma** *int-div-less-self*:
‹$x\ div\ k < x$› **if** ‹$0 < x$› ‹$1 < k$› **for** $x\ k :: int$
⟨*proof*⟩

**lemma** *int-div-le-self*:
‹$x\ div\ k \leq x$› **if** ‹$0 < x$› **for** $x\ k :: int$
⟨*proof*⟩

### 56.5.16 Computing *div* and *mod* by shifting

**lemma** *div-pos-geq*:
‹$k$ *div* $l = (k - l)$ *div* $l + 1$› **if** ‹$0 < l$› ‹$l \leq k$› **for** $k$ $l$ :: *int*
⟨*proof*⟩

**lemma** *mod-pos-geq*:
‹$k$ *mod* $l = (k - l)$ *mod* $l$› **if** ‹$0 < l$› ‹$l \leq k$› **for** $k$ $l$ :: *int*
⟨*proof*⟩

**lemma** *pos-zdiv-mult-2*: ‹$(1 + 2 * b)$ *div* $(2 * a) = b$ *div* $a$› (**is** *?Q*)
**and** *pos-zmod-mult-2*: ‹$(1 + 2 * b)$ *mod* $(2 * a) = 1 + 2 * (b$ *mod* $a)$› (**is** *?R*)
**if** ‹$0 \leq a$› **for** $a$ $b$ :: *int*
⟨*proof*⟩

**lemma** *neg-zdiv-mult-2*: ‹$(1 + 2 * b)$ *div* $(2 * a) = (b + 1)$ *div* $a$› (**is** *?Q*)
**and** *neg-zmod-mult-2*: ‹$(1 + 2 * b)$ *mod* $(2 * a) = 2 * ((b + 1)$ *mod* $a) - 1$›
(**is** *?R*)
**if** ‹$a \leq 0$› **for** $a$ $b$ :: *int*
⟨*proof*⟩

**lemma** *zdiv-numeral-Bit0* [*simp*]:
‹*numeral* (*Num.Bit0 v*) *div numeral* (*Num.Bit0 w*) =
*numeral v div* (*numeral w* :: *int*)›
⟨*proof*⟩

**lemma** *zdiv-numeral-Bit1* [*simp*]:
‹*numeral* (*Num.Bit1 v*) *div numeral* (*Num.Bit0 w*) =
(*numeral v div* (*numeral w* :: *int*))›
⟨*proof*⟩

**lemma** *zmod-numeral-Bit0* [*simp*]:
‹*numeral* (*Num.Bit0 v*) *mod numeral* (*Num.Bit0 w*) =
(*2*::*int*) ∗ (*numeral v mod numeral w*)›
⟨*proof*⟩

**lemma** *zmod-numeral-Bit1* [*simp*]:
‹*numeral* (*Num.Bit1 v*) *mod numeral* (*Num.Bit0 w*) =
*2* ∗ (*numeral v mod numeral w*) + (*1*::*int*)›
⟨*proof*⟩

## 56.6 Code generation

**context**
**begin**

**qualified definition** *divmod-nat* :: $nat \Rightarrow nat \Rightarrow nat \times nat$
**where** *divmod-nat* $m$ $n$ = ($m$ *div* $n$, $m$ *mod* $n$)

**qualified lemma** *divmod-nat-if* [*code*]:

*divmod-nat m n* = (*if n* = *0* ∨ *m* < *n then* (*0*, *m*) *else*
*let* (*q*, *r*) = *divmod-nat* (*m* − *n*) *n in* (*Suc q*, *r*))
⟨*proof*⟩ **lemma** [*code*]:
*m div n* = *fst* (*divmod-nat m n*)
*m mod n* = *snd* (*divmod-nat m n*)
⟨*proof*⟩

**end**

**code-identifier**
**code-module** *Euclidean-Rings* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 57 Parity in rings and semirings

**theory** *Parity*
**imports** *Euclidean-Rings*
**begin**

## 57.1 Ring structures with parity and *even*/*odd* predicates

**class** *semiring-parity* = *comm-semiring-1* + *semiring-modulo* +
**assumes** *mod-2-eq-odd*: ‹*a mod 2* = *of-bool* (¬ *2 dvd a*)›
**and** *odd-one* [*simp*]: ‹¬ *2 dvd 1*›
**and** *even-half-succ-eq* [*simp*]: ‹*2 dvd a* ⟹ (*1* + *a*) *div 2* = *a div 2*›
**begin**

**abbreviation** *even* :: *'a* ⇒ *bool*
**where** ‹*even a* ≡ *2 dvd a*›

**abbreviation** *odd* :: *'a* ⇒ *bool*
**where** ‹*odd a* ≡ ¬ *2 dvd a*›

**end**

**class** *ring-parity* = *ring* + *semiring-parity*
**begin**

**subclass** *comm-ring-1* ⟨*proof*⟩

**end**

**instance** *nat* :: *semiring-parity*
⟨*proof*⟩

**instance** *int* :: *ring-parity*
⟨*proof*⟩

**context** *semiring-parity*
**begin**

**lemma** *evenE* [*elim?*]:
**assumes** ‹*even a*›
**obtains** *b* **where** ‹$a = 2 * b$›
⟨*proof*⟩

**lemma** *oddE* [*elim?*]:
**assumes** ‹*odd a*›
**obtains** *b* **where** ‹$a = 2 * b + 1$›
⟨*proof*⟩

**lemma** *of-bool-odd-eq-mod-2*:
‹*of-bool* (*odd a*) = *a mod 2*›
⟨*proof*⟩

**lemma** *odd-of-bool-self* [*simp*]:
‹*odd* (*of-bool p*) $\longleftrightarrow$ *p*›
⟨*proof*⟩

**lemma** *not-mod-2-eq-0-eq-1* [*simp*]:
‹*a mod 2* $\neq$ *0* $\longleftrightarrow$ *a mod 2* = *1*›
⟨*proof*⟩

**lemma** *not-mod-2-eq-1-eq-0* [*simp*]:
‹*a mod 2* $\neq$ *1* $\longleftrightarrow$ *a mod 2* = *0*›
⟨*proof*⟩

**lemma** *even-iff-mod-2-eq-zero*:
‹*2 dvd a* $\longleftrightarrow$ *a mod 2* = *0*›
⟨*proof*⟩

**lemma** *odd-iff-mod-2-eq-one*:
‹$\neg$ *2 dvd a* $\longleftrightarrow$ *a mod 2* = *1*›
⟨*proof*⟩

**lemma** *even-mod-2-iff* [*simp*]:
‹*even* (*a mod 2*) $\longleftrightarrow$ *even a*›
⟨*proof*⟩

**lemma** *mod2-eq-if*:
*a mod 2* = (*if even a then 0 else 1*)
⟨*proof*⟩

**lemma** *zero-mod-two-eq-zero* [*simp*]:
‹*0 mod 2* = *0*›
⟨*proof*⟩

**lemma** *one-mod-two-eq-one* [*simp*]:
‹$1 \bmod 2 = 1$›
⟨*proof*⟩

**lemma** *parity-cases* [*case-names even odd*]:
**assumes** ‹$even\ a \Longrightarrow a \bmod 2 = 0 \Longrightarrow P$›
**assumes** ‹$odd\ a \Longrightarrow a \bmod 2 = 1 \Longrightarrow P$›
**shows** $P$
⟨*proof*⟩

**lemma** *even-zero* [*simp*]:
‹$even\ 0$›
⟨*proof*⟩

**lemma** *odd-even-add*:
$even\ (a + b)$ **if** $odd\ a$ **and** $odd\ b$
⟨*proof*⟩

**lemma** *even-add* [*simp*]:
$even\ (a + b) \longleftrightarrow (even\ a \longleftrightarrow even\ b)$
⟨*proof*⟩

**lemma** *odd-add* [*simp*]:
$odd\ (a + b) \longleftrightarrow \neg\ (odd\ a \longleftrightarrow odd\ b)$
⟨*proof*⟩

**lemma** *even-plus-one-iff* [*simp*]:
$even\ (a + 1) \longleftrightarrow odd\ a$
⟨*proof*⟩

**lemma** *even-mult-iff* [*simp*]:
$even\ (a * b) \longleftrightarrow even\ a \lor even\ b$ (**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *even-numeral* [*simp*]: $even\ (numeral\ (Num.Bit0\ n))$
⟨*proof*⟩

**lemma** *odd-numeral* [*simp*]: $odd\ (numeral\ (Num.Bit1\ n))$
⟨*proof*⟩

**lemma** *odd-numeral-BitM* [*simp*]:
‹$odd\ (numeral\ (Num.BitM\ w))$›
⟨*proof*⟩

**lemma** *even-power* [*simp*]: $even\ (a\ \hat{}\ n) \longleftrightarrow even\ a \land n > 0$
⟨*proof*⟩

**lemma** *even-prod-iff*:

‹*even* (*prod f A*) $\longleftrightarrow$ ($\exists$ *a*∈*A*. *even* (*f a*))› **if** ‹*finite A*›
⟨*proof*⟩

**lemma** *even-half-maybe-succ-eq* [*simp*]:
‹*even a* $\Longrightarrow$ (*of-bool b* + *a*) *div 2* = *a div 2*›
⟨*proof*⟩

**lemma** *even-half-maybe-succ′-eq* [*simp*]:
‹*even a* $\Longrightarrow$ (*b mod 2* + *a*) *div 2* = *a div 2*›
⟨*proof*⟩

**lemma** *mask-eq-sum-exp*:
‹*2* ^ *n* − *1* = ($\sum$ *m*∈{*q*. *q* < *n*}. *2* ^ *m*)›
⟨*proof*⟩

**lemma** (**in** −) *mask-eq-sum-exp-nat*:
‹*2* ^ *n* − *Suc 0* = ($\sum$ *m*∈{*q*. *q* < *n*}. *2* ^ *m*)›
⟨*proof*⟩

**end**

**context** *ring-parity*
**begin**

**lemma** *even-minus*:
*even* (− *a*) $\longleftrightarrow$ *even a*
⟨*proof*⟩

**lemma** *even-diff* [*simp*]:
*even* (*a* − *b*) $\longleftrightarrow$ *even* (*a* + *b*)
⟨*proof*⟩

**end**

## 57.2 Instance for *nat*

**lemma** *even-Suc-Suc-iff* [*simp*]:
*even* (*Suc* (*Suc n*)) $\longleftrightarrow$ *even n*
⟨*proof*⟩

**lemma** *even-Suc* [*simp*]: *even* (*Suc n*) $\longleftrightarrow$ *odd n*
⟨*proof*⟩

**lemma** *even-diff-nat* [*simp*]:
*even* (*m* − *n*) $\longleftrightarrow$ *m* < *n* ∨ *even* (*m* + *n*) **for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *odd-pos*:
*odd n* $\Longrightarrow$ *0* < *n* **for** *n* :: *nat*

⟨*proof*⟩

**lemma** *Suc-double-not-eq-double*:
$Suc\ (2 * m) \neq 2 * n$
⟨*proof*⟩

**lemma** *double-not-eq-Suc-double*:
$2 * m \neq Suc\ (2 * n)$
⟨*proof*⟩

**lemma** *odd-Suc-minus-one* [*simp*]: $odd\ n \Longrightarrow Suc\ (n - Suc\ 0) = n$
⟨*proof*⟩

**lemma** *even-Suc-div-two* [*simp*]:
$even\ n \Longrightarrow Suc\ n\ div\ 2 = n\ div\ 2$
⟨*proof*⟩

**lemma** *odd-Suc-div-two* [*simp*]:
$odd\ n \Longrightarrow Suc\ n\ div\ 2 = Suc\ (n\ div\ 2)$
⟨*proof*⟩

**lemma** *odd-two-times-div-two-nat* [*simp*]:
**assumes** *odd n*
**shows** $2 * (n\ div\ 2) = n - (1 :: nat)$
⟨*proof*⟩

**lemma** *not-mod2-eq-Suc-0-eq-0* [*simp*]:
$n\ mod\ 2 \neq Suc\ 0 \longleftrightarrow n\ mod\ 2 = 0$
⟨*proof*⟩

**lemma** *odd-card-imp-not-empty*:
‹$A \neq \{\}$› **if** ‹*odd* (*card A*)›
⟨*proof*⟩

**lemma** *nat-induct2* [*case-names 0 1 step*]:
**assumes** *P 0 P 1* **and** *step*: $\bigwedge n{::}nat.\ P\ n \Longrightarrow P\ (n + 2)$
**shows** *P n*
⟨*proof*⟩

**lemma** *mod-double-nat*:
‹$n\ mod\ (2 * m) = n\ mod\ m \lor n\ mod\ (2 * m) = n\ mod\ m + m$›
**for** *m n* :: *nat*
⟨*proof*⟩

**context** *semiring-parity*
**begin**

**lemma** *even-sum-iff*:
‹$even\ (sum\ f\ A) \longleftrightarrow even\ (card\ \{a{\in}A.\ odd\ (f\ a)\})$› **if** ‹*finite A*›

$\langle proof \rangle$

**lemma** *even-mask-iff* [*simp*]:
‹$even\ (2\ \hat{}\ n - 1) \longleftrightarrow n = 0$›
$\langle proof \rangle$

**lemma** *even-of-nat-iff* [*simp*]:
$even\ (of\text{-}nat\ n) \longleftrightarrow even\ n$
$\langle proof \rangle$

**end**

## 57.3 Parity and powers

**context** *ring-1*
**begin**

**lemma** *power-minus-even* [*simp*]: $even\ n \Longrightarrow (-\ a)\ \hat{}\ n = a\ \hat{}\ n$
$\langle proof \rangle$

**lemma** *power-minus-odd* [*simp*]: $odd\ n \Longrightarrow (-\ a)\ \hat{}\ n = -\ (a\ \hat{}\ n)$
$\langle proof \rangle$

**lemma** *uminus-power-if*:
$(-\ a)\ \hat{}\ n = (if\ even\ n\ then\ a\ \hat{}\ n\ else\ -\ (a\ \hat{}\ n))$
$\langle proof \rangle$

**lemma** *neg-one-even-power* [*simp*]: $even\ n \Longrightarrow (-\ 1)\ \hat{}\ n = 1$
$\langle proof \rangle$

**lemma** *neg-one-odd-power* [*simp*]: $odd\ n \Longrightarrow (-\ 1)\ \hat{}\ n = -\ 1$
$\langle proof \rangle$

**lemma** *neg-one-power-add-eq-neg-one-power-diff*: $k \leq n \Longrightarrow (-\ 1)\ \hat{}\ (n + k) = (-\ 1)\ \hat{}\ (n - k)$
$\langle proof \rangle$

**lemma** *minus-one-power-iff*: $(-\ 1)\ \hat{}\ n = (if\ even\ n\ then\ 1\ else\ -\ 1)$
$\langle proof \rangle$

**end**

**context** *linordered-idom*
**begin**

**lemma** *zero-le-even-power*: $even\ n \Longrightarrow 0 \leq a\ \hat{}\ n$
$\langle proof \rangle$

**lemma** *zero-le-odd-power*: $odd\ n \Longrightarrow 0 \leq a\ \hat{}\ n \longleftrightarrow 0 \leq a$

$\langle proof \rangle$

**lemma** *zero-le-power-eq*: $0 \leq a \,\hat{}\, n \longleftrightarrow even\ n \vee odd\ n \wedge 0 \leq a$
$\langle proof \rangle$

**lemma** *zero-less-power-eq*: $0 < a \,\hat{}\, n \longleftrightarrow n = 0 \vee even\ n \wedge a \neq 0 \vee odd\ n \wedge 0 < a$
$\langle proof \rangle$

**lemma** *power-less-zero-eq* [*simp*]: $a \,\hat{}\, n < 0 \longleftrightarrow odd\ n \wedge a < 0$
$\langle proof \rangle$

**lemma** *power-le-zero-eq*: $a \,\hat{}\, n \leq 0 \longleftrightarrow n > 0 \wedge (odd\ n \wedge a \leq 0 \vee even\ n \wedge a = 0)$
$\langle proof \rangle$

**lemma** *power-even-abs*: $even\ n \Longrightarrow |a| \,\hat{}\, n = a \,\hat{}\, n$
$\langle proof \rangle$

**lemma** *power-mono-even*:
**assumes** $even\ n$ **and** $|a| \leq |b|$
**shows** $a \,\hat{}\, n \leq b \,\hat{}\, n$
$\langle proof \rangle$

**lemma** *power-mono-odd*:
**assumes** $odd\ n$ **and** $a \leq b$
**shows** $a \,\hat{}\, n \leq b \,\hat{}\, n$
$\langle proof \rangle$

Simplify, when the exponent is a numeral

**lemma** *zero-le-power-eq-numeral* [*simp*]:
$0 \leq a \,\hat{}\, numeral\ w \longleftrightarrow even\ (numeral\ w :: nat) \vee odd\ (numeral\ w :: nat) \wedge 0 \leq a$
$\langle proof \rangle$

**lemma** *zero-less-power-eq-numeral* [*simp*]:
$0 < a \,\hat{}\, numeral\ w \longleftrightarrow$
$numeral\ w = (0 :: nat) \vee$
$even\ (numeral\ w :: nat) \wedge a \neq 0 \vee$
$odd\ (numeral\ w :: nat) \wedge 0 < a$
$\langle proof \rangle$

**lemma** *power-le-zero-eq-numeral* [*simp*]:
$a \,\hat{}\, numeral\ w \leq 0 \longleftrightarrow$
$(0 :: nat) < numeral\ w \wedge$
$(odd\ (numeral\ w :: nat) \wedge a \leq 0 \vee even\ (numeral\ w :: nat) \wedge a = 0)$
$\langle proof \rangle$

**lemma** *power-less-zero-eq-numeral* [*simp*]:

$a \mathbin{\hat{}} numeral\ w < 0 \longleftrightarrow odd\ (numeral\ w :: nat) \wedge a < 0$
⟨proof⟩

**lemma** *power-even-abs-numeral* [*simp*]:
$even\ (numeral\ w :: nat) \Longrightarrow |a| \mathbin{\hat{}} numeral\ w = a \mathbin{\hat{}} numeral\ w$
⟨proof⟩

**end**

## 57.4 Instance for *int*

**lemma** *even-diff-iff*:
$even\ (k - l) \longleftrightarrow even\ (k + l)$ **for** $k\ l :: int$
⟨proof⟩

**lemma** *even-abs-add-iff*:
$even\ (|k| + l) \longleftrightarrow even\ (k + l)$ **for** $k\ l :: int$
⟨proof⟩

**lemma** *even-add-abs-iff*:
$even\ (k + |l|) \longleftrightarrow even\ (k + l)$ **for** $k\ l :: int$
⟨proof⟩

**lemma** *even-nat-iff*: $0 \leq k \Longrightarrow even\ (nat\ k) \longleftrightarrow even\ k$
⟨proof⟩

**context**
**assumes** *SORT-CONSTRAINT*('*a*::*division-ring*)
**begin**

**lemma** *power-int-minus-left*:
$power\text{-}int\ (-a :: 'a)\ n = (if\ even\ n\ then\ power\text{-}int\ a\ n\ else\ -power\text{-}int\ a\ n)$
⟨proof⟩

**lemma** *power-int-minus-left-even* [*simp*]: $even\ n \Longrightarrow power\text{-}int\ (-a :: 'a)\ n = power\text{-}int\ a\ n$
⟨proof⟩

**lemma** *power-int-minus-left-odd* [*simp*]: $odd\ n \Longrightarrow power\text{-}int\ (-a :: 'a)\ n = -power\text{-}int\ a\ n$
⟨proof⟩

**lemma** *power-int-minus-left-distrib*:
$NO\text{-}MATCH\ (-1)\ x \Longrightarrow power\text{-}int\ (-a :: 'a)\ n = power\text{-}int\ (-1)\ n * power\text{-}int\ a\ n$
⟨proof⟩

**lemma** *power-int-minus-one-minus*: $power\text{-}int\ (-1 :: 'a)\ (-n) = power\text{-}int\ (-1)\ n$

⟨*proof*⟩

**lemma** *power-int-minus-one-diff-commute*: *power-int* $(-1 :: 'a)$ $(a - b) =$ *power-int* $(-1)$ $(b - a)$
⟨*proof*⟩

**lemma** *power-int-minus-one-mult-self* [*simp*]:
*power-int* $(-1 :: 'a)$ $m *$ *power-int* $(-1)$ $m = 1$
⟨*proof*⟩

**lemma** *power-int-minus-one-mult-self'* [*simp*]:
*power-int* $(-1 :: 'a)$ $m *$ (*power-int* $(-1)$ $m * b) = b$
⟨*proof*⟩

**end**

## 57.5 Special case: euclidean rings structurally containing the natural numbers

**class** *linordered-euclidean-semiring* = *discrete-linordered-semidom* + *unique-euclidean-semiring* +
**assumes** *of-nat-div*: *of-nat* ($m$ *div* $n$) = *of-nat* $m$ *div* *of-nat* $n$
**and** *division-segment-of-nat* [*simp*]: *division-segment* (*of-nat* $n$) = *1*
**and** *division-segment-euclidean-size* [*simp*]: *division-segment* $a$ ∗ *of-nat* (*euclidean-size* $a$) = $a$
**begin**

**lemma** *division-segment-eq-iff*:
$a = b$ **if** *division-segment* $a$ = *division-segment* $b$
**and** *euclidean-size* $a$ = *euclidean-size* $b$
⟨*proof*⟩

**lemma** *euclidean-size-of-nat* [*simp*]:
*euclidean-size* (*of-nat* $n$) = $n$
⟨*proof*⟩

**lemma** *of-nat-euclidean-size*:
*of-nat* (*euclidean-size* $a$) = $a$ *div* *division-segment* $a$
⟨*proof*⟩

**lemma** *division-segment-1* [*simp*]:
*division-segment* *1* = *1*
⟨*proof*⟩

**lemma** *division-segment-numeral* [*simp*]:
*division-segment* (*numeral* $k$) = *1*
⟨*proof*⟩

**lemma** *euclidean-size-1* [*simp*]:

*euclidean-size 1 = 1*
⟨*proof*⟩

**lemma** *euclidean-size-numeral* [*simp*]:
*euclidean-size* (*numeral k*) = *numeral k*
⟨*proof*⟩

**lemma** *of-nat-dvd-iff*:
*of-nat m dvd of-nat n* ⟷ *m dvd n* (**is** *?P* ⟷ *?Q*)
⟨*proof*⟩

**lemma** *of-nat-mod*:
*of-nat* (*m mod n*) = *of-nat m mod of-nat n*
⟨*proof*⟩

**lemma** *one-div-two-eq-zero* [*simp*]:
*1 div 2 = 0*
⟨*proof*⟩

**lemma** *one-mod-2-pow-eq* [*simp*]:
*1 mod* (*2 ^ n*) = *of-bool* (*n > 0*)
⟨*proof*⟩

**lemma** *one-div-2-pow-eq* [*simp*]:
*1 div* (*2 ^ n*) = *of-bool* (*n = 0*)
⟨*proof*⟩

**lemma** *div-mult2-eq'*:
‹*a div* (*of-nat m* ∗ *of-nat n*) = *a div of-nat m div of-nat n*›
⟨*proof*⟩

**lemma** *mod-mult2-eq'*:
*a mod* (*of-nat m* ∗ *of-nat n*) = *of-nat m* ∗ (*a div of-nat m mod of-nat n*) + *a mod of-nat m*
⟨*proof*⟩

**lemma** *div-mult2-numeral-eq*:
*a div numeral k div numeral l = a div numeral* (*k* ∗ *l*) (**is** *?A = ?B*)
⟨*proof*⟩

**lemma** *numeral-Bit0-div-2*:
*numeral* (*num.Bit0 n*) *div 2 = numeral n*
⟨*proof*⟩

**lemma** *numeral-Bit1-div-2*:
*numeral* (*num.Bit1 n*) *div 2 = numeral n*
⟨*proof*⟩

**lemma** *exp-mod-exp*:

‹$2$ ^ $m$ *mod* $2$ ^ $n$ = *of-bool* ($m < n$) ∗ $2$ ^ $m$›
⟨*proof*⟩

**lemma** *mask-mod-exp*:
‹($2$ ^ $n$ − $1$) *mod* $2$ ^ $m$ = $2$ ^ *min* $m$ $n$ − $1$›
⟨*proof*⟩

**lemma** *of-bool-half-eq-0* [*simp*]:
‹*of-bool* $b$ *div* $2$ = $0$›
⟨*proof*⟩

**lemma** *of-nat-mod-double*:
‹*of-nat* $n$ *mod* ($2$ ∗ *of-nat* $m$) = *of-nat* $n$ *mod* *of-nat* $m$ ∨ *of-nat* $n$ *mod* ($2$ ∗ *of-nat* $m$) = *of-nat* $n$ *mod* *of-nat* $m$ + *of-nat* $m$›
⟨*proof*⟩

**end**

**instance** *nat* :: *linordered-euclidean-semiring*
⟨*proof*⟩

**instance** *int* :: *linordered-euclidean-semiring*
⟨*proof*⟩

**context** *linordered-euclidean-semiring*
**begin**

**subclass** *semiring-parity*
⟨*proof*⟩

**lemma** *even-succ-div-two* [*simp*]:
*even* $a \Longrightarrow (a + 1)$ *div* $2 = a$ *div* $2$
⟨*proof*⟩

**lemma** *odd-succ-div-two* [*simp*]:
*odd* $a \Longrightarrow (a + 1)$ *div* $2 = a$ *div* $2 + 1$
⟨*proof*⟩

**lemma** *even-two-times-div-two*:
*even* $a \Longrightarrow 2 * (a$ *div* $2) = a$
⟨*proof*⟩

**lemma** *odd-two-times-div-two-succ* [*simp*]:
*odd* $a \Longrightarrow 2 * (a$ *div* $2) + 1 = a$
⟨*proof*⟩

**lemma** *coprime-left-2-iff-odd* [*simp*]:
*coprime* $2$ $a \longleftrightarrow$ *odd* $a$
⟨*proof*⟩

**lemma** *coprime-right-2-iff-odd* [*simp*]:
*coprime a 2* $\longleftrightarrow$ *odd a*
⟨*proof*⟩

**end**

**lemma** *minus-1-mod-2-eq* [*simp*]:
‹$-$ *1 mod 2* = (*1*::*int*)›
⟨*proof*⟩

**lemma** *minus-1-div-2-eq* [*simp*]:
$-$ *1 div 2* = $-$ (*1*::*int*)
⟨*proof*⟩

**context** *linordered-euclidean-semiring*
**begin**

**lemma** *even-decr-exp-div-exp-iff'*:
‹*even* ((*2* ^ *m* $-$ *1*) *div 2* ^ *n*) $\longleftrightarrow$ *m* $\leq$ *n*›
⟨*proof*⟩

**end**

## 57.6 Generic symbolic computations

The following type class contains everything necessary to formulate a division algorithm in ring structures with numerals, restricted to its positive segments.

**class** *linordered-euclidean-semiring-division* = *linordered-euclidean-semiring* +
**fixes** *divmod* :: ‹*num* $\Rightarrow$ *num* $\Rightarrow$ *'a* $\times$ *'a*›
**and** *divmod-step* :: ‹*'a* $\Rightarrow$ *'a* $\times$ *'a* $\Rightarrow$ *'a* $\times$ *'a*› — These are conceptually definitions but force generated code to be monomorphic wrt. particular instances of this class which yields a significant speedup.
**assumes** *divmod-def*: ‹*divmod m n* = (*numeral m div numeral n*, *numeral m mod numeral n*)›
**and** *divmod-step-def* [*simp*]: ‹*divmod-step l* (*q*, *r*) =
(*if euclidean-size l* $\leq$ *euclidean-size r then* (*2* $*$ *q* + *1*, *r* $-$ *l*)
*else* (*2* $*$ *q*, *r*))› — This is a formulation of one step (referring to one digit position) in school-method division: compare the dividend at the current digit position with the remainder from previous division steps and evaluate accordingly.
**begin**

**lemma** *fst-divmod*:
‹*fst* (*divmod m n*) = *numeral m div numeral n*›
⟨*proof*⟩

**lemma** *snd-divmod*:
‹*snd* (*divmod m n*) = *numeral m mod numeral n*›

⟨*proof*⟩

Following a formulation of school-method division. If the divisor is smaller than the dividend, terminate. If not, shift the dividend to the right until termination occurs and then reiterate single division steps in the opposite direction.

**lemma** *divmod-divmod-step*:
‹*divmod m n* = (*if m* < *n then* (*0*, *numeral m*)
*else divmod-step* (*numeral n*) (*divmod m* (*Num.Bit0 n*)))›
⟨*proof*⟩

The division rewrite proper – first, trivial results involving *1*

**lemma** *divmod-trivial* [*simp*]:
*divmod m Num.One* = (*numeral m*, *0*)
*divmod num.One* (*num.Bit0 n*) = (*0*, *Numeral1*)
*divmod num.One* (*num.Bit1 n*) = (*0*, *Numeral1*)
⟨*proof*⟩

Division by an even number is a right-shift

**lemma** *divmod-cancel* [*simp*]:
‹*divmod* (*Num.Bit0 m*) (*Num.Bit0 n*) = (*case divmod m n of* (*q*, *r*) ⇒ (*q*, *2* ∗ *r*))› (**is** *?P*)
‹*divmod* (*Num.Bit1 m*) (*Num.Bit0 n*) = (*case divmod m n of* (*q*, *r*) ⇒ (*q*, *2* ∗ *r* + *1*))› (**is** *?Q*)
⟨*proof*⟩

The really hard work

**lemma** *divmod-steps* [*simp*]:
*divmod* (*num.Bit0 m*) (*num.Bit1 n*) =
(*if m* ≤ *n then* (*0*, *numeral* (*num.Bit0 m*))
*else divmod-step* (*numeral* (*num.Bit1 n*))
(*divmod* (*num.Bit0 m*)
(*num.Bit0* (*num.Bit1 n*))))
*divmod* (*num.Bit1 m*) (*num.Bit1 n*) =
(*if m* < *n then* (*0*, *numeral* (*num.Bit1 m*))
*else divmod-step* (*numeral* (*num.Bit1 n*))
(*divmod* (*num.Bit1 m*)
(*num.Bit0* (*num.Bit1 n*))))
⟨*proof*⟩

**lemmas** *divmod-algorithm-code* = *divmod-trivial divmod-cancel divmod-steps*

Special case: divisibility

**definition** *divides-aux* :: *'a* × *'a* ⇒ *bool*
**where**
*divides-aux qr* ⟷ *snd qr* = *0*

**lemma** *divides-aux-eq* [*simp*]:

*divides-aux* $(q, r) \longleftrightarrow r = 0$
⟨*proof*⟩

**lemma** *dvd-numeral-simp* [*simp*]:
*numeral m dvd numeral n* $\longleftrightarrow$ *divides-aux* (*divmod n m*)
⟨*proof*⟩

Generic computation of quotient and remainder

**lemma** *numeral-div-numeral* [*simp*]:
*numeral k div numeral l* = *fst* (*divmod k l*)
⟨*proof*⟩

**lemma** *numeral-mod-numeral* [*simp*]:
*numeral k mod numeral l* = *snd* (*divmod k l*)
⟨*proof*⟩

**lemma** *one-div-numeral* [*simp*]:
*1 div numeral n* = *fst* (*divmod num.One n*)
⟨*proof*⟩

**lemma** *one-mod-numeral* [*simp*]:
*1 mod numeral n* = *snd* (*divmod num.One n*)
⟨*proof*⟩

**end**

**instantiation** *nat* :: *linordered-euclidean-semiring-division*
**begin**

**definition** *divmod-nat* :: *num* ⇒ *num* ⇒ *nat* × *nat*
**where**
*divmod′-nat-def*: *divmod-nat m n* = (*numeral m div numeral n*, *numeral m mod numeral n*)

**definition** *divmod-step-nat* :: *nat* ⇒ *nat* × *nat* ⇒ *nat* × *nat*
**where**
*divmod-step-nat l qr* = (*let* (*q*, *r*) = *qr*
*in if* $r \geq l$ *then* $(2 * q + 1, r - l)$
*else* $(2 * q, r)$)

**instance**
⟨*proof*⟩

**end**

**declare** *divmod-algorithm-code* [**where** *?′a* = *nat*, *code*]

**lemma** *Suc-0-div-numeral* [*simp*]:
‹*Suc 0 div numeral Num.One* = *1*›

‹*Suc 0 div numeral* (*Num.Bit0 n*) = *0*›
‹*Suc 0 div numeral* (*Num.Bit1 n*) = *0*›
⟨*proof*⟩

**lemma** *Suc-0-mod-numeral* [*simp*]:
‹*Suc 0 mod numeral Num.One* = *0*›
‹*Suc 0 mod numeral* (*Num.Bit0 n*) = *1*›
‹*Suc 0 mod numeral* (*Num.Bit1 n*) = *1*›
⟨*proof*⟩

**instantiation** *int* :: *linordered-euclidean-semiring-division*
**begin**

**definition** *divmod-int* :: *num* ⇒ *num* ⇒ *int* × *int*
**where**
*divmod-int m n* = (*numeral m div numeral n*, *numeral m mod numeral n*)

**definition** *divmod-step-int* :: *int* ⇒ *int* × *int* ⇒ *int* × *int*
**where**
*divmod-step-int l qr* = (*let* (*q*, *r*) = *qr*
*in if* |*l*| ≤ |*r*| *then* (*2* ∗ *q* + *1*, *r* − *l*)
*else* (*2* ∗ *q*, *r*))

**instance**
⟨*proof*⟩

**end**

**declare** *divmod-algorithm-code* [**where** *?'a* = *int*, *code*]

**context**
**begin**

**qualified definition** *adjust-div* :: *int* × *int* ⇒ *int*
**where**
*adjust-div qr* = (*let* (*q*, *r*) = *qr in q* + *of-bool* (*r* ≠ *0*))

**qualified lemma** *adjust-div-eq* [*simp*, *code*]:
*adjust-div* (*q*, *r*) = *q* + *of-bool* (*r* ≠ *0*)
⟨*proof*⟩ **definition** *adjust-mod* :: *num* ⇒ *int* ⇒ *int*
**where**
[*simp*]: *adjust-mod l r* = (*if r* = *0 then 0 else numeral l* − *r*)

**lemma** *minus-numeral-div-numeral* [*simp*]:
− *numeral m div numeral n* = − (*adjust-div* (*divmod m n*) :: *int*)
⟨*proof*⟩

**lemma** *minus-numeral-mod-numeral* [*simp*]:
− *numeral m mod numeral n* = *adjust-mod n* (*snd* (*divmod m n*) :: *int*)

⟨*proof*⟩

**lemma** *numeral-div-minus-numeral* [*simp*]:
*numeral m div* − *numeral n* = − (*adjust-div* (*divmod m n*) :: *int*)
⟨*proof*⟩

**lemma** *numeral-mod-minus-numeral* [*simp*]:
*numeral m mod* − *numeral n* = − *adjust-mod n* (*snd* (*divmod m n*) :: *int*)
⟨*proof*⟩

**lemma** *minus-one-div-numeral* [*simp*]:
− *1 div numeral n* = − (*adjust-div* (*divmod Num.One n*) :: *int*)
⟨*proof*⟩

**lemma** *minus-one-mod-numeral* [*simp*]:
− *1 mod numeral n* = *adjust-mod n* (*snd* (*divmod Num.One n*) :: *int*)
⟨*proof*⟩

**lemma** *one-div-minus-numeral* [*simp*]:
*1 div* − *numeral n* = − (*adjust-div* (*divmod Num.One n*) :: *int*)
⟨*proof*⟩

**lemma** *one-mod-minus-numeral* [*simp*]:
*1 mod* − *numeral n* = − *adjust-mod n* (*snd* (*divmod Num.One n*) :: *int*)
⟨*proof*⟩

**lemma** [*code*]:
**fixes** *k* :: *int*
**shows**
*k div 0* = *0*
*k mod 0* = *k*
*0 div k* = *0*
*0 mod k* = *0*
*k div Int.Pos Num.One* = *k*
*k mod Int.Pos Num.One* = *0*
*k div Int.Neg Num.One* = − *k*
*k mod Int.Neg Num.One* = *0*
*Int.Pos m div Int.Pos n* = (*fst* (*divmod m n*) :: *int*)
*Int.Pos m mod Int.Pos n* = (*snd* (*divmod m n*) :: *int*)
*Int.Neg m div Int.Pos n* = − (*adjust-div* (*divmod m n*) :: *int*)
*Int.Neg m mod Int.Pos n* = *adjust-mod n* (*snd* (*divmod m n*) :: *int*)
*Int.Pos m div Int.Neg n* = − (*adjust-div* (*divmod m n*) :: *int*)
*Int.Pos m mod Int.Neg n* = − *adjust-mod n* (*snd* (*divmod m n*) :: *int*)
*Int.Neg m div Int.Neg n* = (*fst* (*divmod m n*) :: *int*)
*Int.Neg m mod Int.Neg n* = − (*snd* (*divmod m n*) :: *int*)
⟨*proof*⟩

**end**

**lemma** *divmod-BitM-2-eq* [*simp*]:
‹*divmod* (*Num.BitM m*) (*Num.Bit0 Num.One*) = (*numeral m* − *1*, (*1* :: *int*))›
⟨*proof*⟩

### 57.6.1 Computation by simplification

**lemma** *euclidean-size-nat-less-eq-iff*:
‹*euclidean-size m* ≤ *euclidean-size n* ⟷ *m* ≤ *n*› **for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *euclidean-size-int-less-eq-iff*:
‹*euclidean-size k* ≤ *euclidean-size l* ⟷ |*k*| ≤ |*l*|› **for** *k l* :: *int*
⟨*proof*⟩

⟨*ML*⟩

## 57.7 Computing congruences modulo *2* ^ *q*

**context** *linordered-euclidean-semiring-division*
**begin**

**lemma** *cong-exp-iff-simps*:
*numeral n mod numeral Num.One* = *0*
⟷ *True*
*numeral* (*Num.Bit0 n*) *mod numeral* (*Num.Bit0 q*) = *0*
⟷ *numeral n mod numeral q* = *0*
*numeral* (*Num.Bit1 n*) *mod numeral* (*Num.Bit0 q*) = *0*
⟷ *False*
*numeral m mod numeral Num.One* = (*numeral n mod numeral Num.One*)
⟷ *True*
*numeral Num.One mod numeral* (*Num.Bit0 q*) = (*numeral Num.One mod numeral* (*Num.Bit0 q*))
⟷ *True*
*numeral Num.One mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit0 n*) *mod numeral* (*Num.Bit0 q*))
⟷ *False*
*numeral Num.One mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit1 n*) *mod numeral* (*Num.Bit0 q*))
⟷ (*numeral n mod numeral q*) = *0*
*numeral* (*Num.Bit0 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral Num.One mod numeral* (*Num.Bit0 q*))
⟷ *False*
*numeral* (*Num.Bit0 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit0 n*) *mod numeral* (*Num.Bit0 q*))
⟷ *numeral m mod numeral q* = (*numeral n mod numeral q*)
*numeral* (*Num.Bit0 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit1 n*) *mod numeral* (*Num.Bit0 q*))
⟷ *False*
*numeral* (*Num.Bit1 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral Num.One mod numeral* (*Num.Bit0 q*))

$\longleftrightarrow$ (*numeral m mod numeral q*) = *0*
*numeral* (*Num.Bit1 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit0 n*) *mod numeral* (*Num.Bit0 q*))
$\longleftrightarrow$ *False*
*numeral* (*Num.Bit1 m*) *mod numeral* (*Num.Bit0 q*) = (*numeral* (*Num.Bit1 n*) *mod numeral* (*Num.Bit0 q*))
$\longleftrightarrow$ *numeral m mod numeral q* = (*numeral n mod numeral q*)
⟨*proof*⟩

**end**

**code-identifier**
**code-module** *Parity* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**lemmas** *even-of-nat* = *even-of-nat-iff*

**end**

# 58 Combination and Cancellation Simprocs for Numeral Expressions

**theory** *Numeral-Simprocs*
**imports** *Parity*
**begin**

⟨*ML*⟩

**lemmas** *semiring-norm* =
*Let-def arith-simps diff-nat-numeral rel-simps*
*if-False if-True*
*add-Suc add-numeral-left*
*add-neg-numeral-left mult-numeral-left*
*numeral-One* [*symmetric*] *uminus-numeral-One* [*symmetric*] *Suc-eq-plus1*
*eq-numeral-iff-iszero not-iszero-Numeral1*

For *combine-numerals*

**lemma** *left-add-mult-distrib*: $i*u + (j*u + k) = (i+j)*u + (k::nat)$
⟨*proof*⟩

For *cancel-numerals*

**lemma** *nat-diff-add-eq1*:
$j <= (i::nat) ==> ((i*u + m) - (j*u + n)) = (((i-j)*u + m) - n)$
⟨*proof*⟩

**lemma** *nat-diff-add-eq2*:
$i <= (j::nat) ==> ((i*u + m) - (j*u + n)) = (m - ((j-i)*u + n))$
⟨*proof*⟩

**lemma** *nat-eq-add-iff1*:
$j <= (i::nat) ==> (i*u + m = j*u + n) = ((i-j)*u + m = n)$
⟨*proof*⟩

**lemma** *nat-eq-add-iff2*:
$i <= (j::nat) ==> (i*u + m = j*u + n) = (m = (j-i)*u + n)$
⟨*proof*⟩

**lemma** *nat-less-add-iff1*:
$j <= (i::nat) ==> (i*u + m < j*u + n) = ((i-j)*u + m < n)$
⟨*proof*⟩

**lemma** *nat-less-add-iff2*:
$i <= (j::nat) ==> (i*u + m < j*u + n) = (m < (j-i)*u + n)$
⟨*proof*⟩

**lemma** *nat-le-add-iff1*:
$j <= (i::nat) ==> (i*u + m <= j*u + n) = ((i-j)*u + m <= n)$
⟨*proof*⟩

**lemma** *nat-le-add-iff2*:
$i <= (j::nat) ==> (i*u + m <= j*u + n) = (m <= (j-i)*u + n)$
⟨*proof*⟩

For *cancel-numeral-factors*

**lemma** *nat-mult-le-cancel1*: $(0::nat) < k ==> (k*m <= k*n) = (m<=n)$
⟨*proof*⟩

**lemma** *nat-mult-less-cancel1*: $(0::nat) < k ==> (k*m < k*n) = (m<n)$
⟨*proof*⟩

**lemma** *nat-mult-eq-cancel1*: $(0::nat) < k ==> (k*m = k*n) = (m=n)$
⟨*proof*⟩

**lemma** *nat-mult-div-cancel1*: $(0::nat) < k ==> (k*m)\ div\ (k*n) = (m\ div\ n)$
⟨*proof*⟩

**lemma** *nat-mult-dvd-cancel-disj*[*simp*]:
$(k*m)\ dvd\ (k*n) = (k=0 \lor m\ dvd\ (n::nat))$
⟨*proof*⟩

**lemma** *nat-mult-dvd-cancel1*: $0 < k \Longrightarrow (k*m)\ dvd\ (k*n::nat) = (m\ dvd\ n)$
⟨*proof*⟩

For *cancel-factor*

**lemmas** *nat-mult-le-cancel-disj* = *mult-le-cancel1*

**lemmas** *nat-mult-less-cancel-disj* = *mult-less-cancel1*

**lemma** *nat-mult-eq-cancel-disj*:
**fixes** $k\ m\ n :: nat$
**shows** $k * m = k * n \longleftrightarrow k = 0 \lor m = n$
⟨*proof*⟩

**lemma** *nat-mult-div-cancel-disj*:
**fixes** $k\ m\ n :: nat$
**shows** $(k * m)\ div\ (k * n) = (if\ k = 0\ then\ 0\ else\ m\ div\ n)$
⟨*proof*⟩

**lemma** *numeral-times-minus-swap*:
**fixes** $x{::}\ 'a{::}comm\text{-}ring\text{-}1$ **shows** $numeral\ w * -x = x * -\ numeral\ w$
⟨*proof*⟩

⟨*ML*⟩

**end**

# 59 Semiring normalization

**theory** *Semiring-Normalization*
**imports** *Numeral-Simprocs*
**begin**

Prelude

**class** *comm-semiring-1-cancel-crossproduct* = *comm-semiring-1-cancel* +
**assumes** *crossproduct-eq*: $w * y + x * z = w * z + x * y \longleftrightarrow w = x \lor y = z$
**begin**

**lemma** *crossproduct-noteq*:
$a \neq b \land c \neq d \longleftrightarrow a * c + b * d \neq a * d + b * c$
⟨*proof*⟩

**lemma** *add-scale-eq-noteq*:
$r \neq 0 \Longrightarrow a = b \land c \neq d \Longrightarrow a + r * c \neq b + r * d$
⟨*proof*⟩

**lemma** *add-0-iff*:
$b = b + a \longleftrightarrow a = 0$
⟨*proof*⟩

**end**

**subclass** (**in** *idom*) *comm-semiring-1-cancel-crossproduct*
⟨*proof*⟩

**instance** *nat* :: *comm-semiring-1-cancel-crossproduct*
⟨*proof*⟩

Semiring normalization proper

⟨ML⟩

**context** *comm-semiring-1*
**begin**

**lemma** *semiring-normalization-rules* [*no-atp*]:
$(a * m) + (b * m) = (a + b) * m$
$(a * m) + m = (a + 1) * m$
$m + (a * m) = (a + 1) * m$
$m + m = (1 + 1) * m$
$0 + a = a$
$a + 0 = a$
$a * b = b * a$
$(a + b) * c = (a * c) + (b * c)$
$0 * a = 0$
$a * 0 = 0$
$1 * a = a$
$a * 1 = a$
$(lx * ly) * (rx * ry) = (lx * rx) * (ly * ry)$
$(lx * ly) * (rx * ry) = lx * (ly * (rx * ry))$
$(lx * ly) * (rx * ry) = rx * ((lx * ly) * ry)$
$(lx * ly) * rx = (lx * rx) * ly$
$(lx * ly) * rx = lx * (ly * rx)$
$lx * (rx * ry) = (lx * rx) * ry$
$lx * (rx * ry) = rx * (lx * ry)$
$(a + b) + (c + d) = (a + c) + (b + d)$
$(a + b) + c = a + (b + c)$
$a + (c + d) = c + (a + d)$
$(a + b) + c = (a + c) + b$
$a + c = c + a$
$a + (c + d) = (a + c) + d$
$(x \hat{} p) * (x \hat{} q) = x \hat{} (p + q)$
$x * (x \hat{} q) = x \hat{} (Suc\ q)$
$(x \hat{} q) * x = x \hat{} (Suc\ q)$
$x * x = x^2$
$(x * y) \hat{} q = (x \hat{} q) * (y \hat{} q)$
$(x \hat{} p) \hat{} q = x \hat{} (p * q)$
$x \hat{} 0 = 1$
$x \hat{} 1 = x$
$x * (y + z) = (x * y) + (x * z)$
$x \hat{} (Suc\ q) = x * (x \hat{} q)$
$x \hat{} (2*n) = (x \hat{} n) * (x \hat{} n)$
⟨*proof*⟩

⟨ML⟩

**end**

**context** *comm-ring-1*
**begin**

**lemma** *ring-normalization-rules* [*no-atp*]:
$- x = (- 1) * x$
$x - y = x + (- y)$
⟨*proof*⟩

⟨*ML*⟩

**end**

**context** *comm-semiring-1-cancel-crossproduct*
**begin**

⟨*ML*⟩

**end**

**context** *idom*
**begin**

⟨*ML*⟩

**end**

**context** *field*
**begin**

⟨*ML*⟩

**end**

**code-identifier**
**code-module** *Semiring-Normalization* ⇀ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

**end**

# 60 Groebner bases

**theory** *Groebner-Basis*
**imports** *Semiring-Normalization Parity*
**begin**

## 60.1 Groebner Bases

**lemmas** *bool-simps* = *simp-thms*(*1*−*34*) — FIXME move to *HOL.HOL*

**lemma** *nnf-simps*: — FIXME shadows fact binding in *HOL.HOL*
$(\neg(P \wedge Q)) = (\neg P \vee \neg Q)$ $(\neg(P \vee Q)) = (\neg P \wedge \neg Q)$
$(P \longrightarrow Q) = (\neg P \vee Q)$
$(P = Q) = ((P \wedge Q) \vee (\neg P \wedge \neg\ Q))$ $(\neg\ \neg(P)) = P$
⟨*proof*⟩

**lemma** *dnf*:
$(P \wedge (Q \vee R)) = ((P{\wedge}Q) \vee (P{\wedge}R))$
$((Q \vee R) \wedge P) = ((Q{\wedge}P) \vee (R{\wedge}P))$
$(P \wedge Q) = (Q \wedge P)$
$(P \vee Q) = (Q \vee P)$
⟨*proof*⟩

**lemmas** *weak-dnf-simps* = *dnf bool-simps*

**lemma** *PFalse*:
$P \equiv \mathit{False} \Longrightarrow \neg\ P$
$\neg\ P \Longrightarrow (P \equiv \mathit{False})$
⟨*proof*⟩

**named-theorems** *algebra pre−simplification rules for algebraic methods*
⟨*ML*⟩

**declare** *dvd-def*[*algebra*]
**declare** *mod-eq-0-iff-dvd*[*algebra*]
**declare** *mod-div-trivial*[*algebra*]
**declare** *mod-mod-trivial*[*algebra*]
**declare** *div-by-0*[*algebra*]
**declare** *mod-by-0*[*algebra*]
**declare** *mult-div-mod-eq*[*algebra*]
**declare** *div-minus-minus*[*algebra*]
**declare** *mod-minus-minus*[*algebra*]
**declare** *div-minus-right*[*algebra*]
**declare** *mod-minus-right*[*algebra*]
**declare** *div-0*[*algebra*]
**declare** *mod-0*[*algebra*]
**declare** *mod-by-1*[*algebra*]
**declare** *div-by-1*[*algebra*]
**declare** *mod-minus1-right*[*algebra*]
**declare** *div-minus1-right*[*algebra*]
**declare** *mod-mult-self2-is-0*[*algebra*]
**declare** *mod-mult-self1-is-0*[*algebra*]

**lemma** *zmod-eq-0-iff* [*algebra*]:
‹$m\ \mathit{mod}\ d = 0 \longleftrightarrow (\exists\, q.\ m = d * q)$› **for** $m\ d$ :: *int*
⟨*proof*⟩

**declare** *dvd-0-left-iff*[*algebra*]
**declare** *zdvd1-eq*[*algebra*]

**declare** *mod-eq-dvd-iff* [*algebra*]
**declare** *nat-mod-eq-iff* [*algebra*]

**context** *semiring-parity*
**begin**

**declare** *even-mult-iff* [*algebra*]
**declare** *even-power* [*algebra*]

**end**

**context** *ring-parity*
**begin**

**declare** *even-minus* [*algebra*]

**end**

**declare** *even-Suc* [*algebra*]
**declare** *even-diff-nat* [*algebra*]

**end**

# 61 Set intervals

**theory** *Set-Interval*
**imports** *Parity*
**begin**

**lemma** *card-2-iff*: $card\ S = 2 \longleftrightarrow (\exists\, x\ y.\ S = \{x,y\} \wedge x \neq y)$
⟨*proof*⟩

**lemma** *card-2-iff'*: $card\ S = 2 \longleftrightarrow (\exists\, x{\in}S.\ \exists\, y{\in}S.\ x \neq y \wedge (\forall\, z{\in}S.\ z = x \vee z = y))$
⟨*proof*⟩

**lemma** *card-3-iff*: $card\ S = 3 \longleftrightarrow (\exists\, x\ y\ z.\ S = \{x,y,z\} \wedge x \neq y \wedge y \neq z \wedge x \neq z)$
⟨*proof*⟩

**context** *ord*
**begin**

**definition**
*lessThan* :: $'a \Rightarrow 'a\ set$ (‹(‹indent=1 notation=‹mixfix set interval››{..<-})›)
**where**
$\{..{<}u\} == \{x.\ x < u\}$

**definition**
*atMost* :: *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{..-})›)
**where**
{..*u*} == {*x*. *x* ≤ *u*}

**definition**
*greaterThan* :: *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-<..})›)
**where**
{*l*<..} == {*x*. *l*<*x*}

**definition**
*atLeast* :: *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-..})›)
**where**
{*l*..} == {*x*. *l*≤*x*}

**definition**
*greaterThanLessThan* :: *'a* ⇒ *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-/<..<-})›) **where**
{*l*<..<*u*} == {*l*<..} ∩ {..<*u*}

**definition**
*atLeastLessThan* :: *'a* ⇒ *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-/..<-})›) **where**
{*l*..<*u*} == {*l*..} ∩ {..<*u*}

**definition**
*greaterThanAtMost* :: *'a* ⇒ *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-/<..-})›) **where**
{*l*<..*u*} == {*l*<..} ∩ {..*u*}

**definition**
*atLeastAtMost* :: *'a* ⇒ *'a* ⇒ *'a set* (‹(‹*indent=1 notation=*‹*mixfix set interval*››{-/..-})›) **where**
{*l*..*u*} == {*l*..} ∩ {..*u*}

**end**

A note of warning when using {..<*n*} on type *nat*: it is equivalent to {*0*..<*n*} but some lemmas involving {*m*..<*n*} may not exist in {..<*n*}-form as well.

**syntax** (*ASCII*)
*-UNION-le* :: *'a* ⇒ *'a* ⇒ *'b set* ⇒ *'b set* (‹(‹*indent=3 notation=*‹*binder UN*››*UN* -<=-./ -)› [*0*, *0*, *10*] *10*)
*-UNION-less* :: *'a* ⇒ *'a* ⇒ *'b set* ⇒ *'b set* (‹(‹*indent=3 notation=*‹*binder UN*››*UN* -<-./ -)› [*0*, *0*, *10*] *10*)
*-INTER-le* :: *'a* ⇒ *'a* ⇒ *'b set* ⇒ *'b set* (‹(‹*indent=3 notation=*‹*binder INT*››*INT* -<=-./ -)› [*0*, *0*, *10*] *10*)
*-INTER-less* :: *'a* ⇒ *'a* ⇒ *'b set* ⇒ *'b set* (‹(‹*indent=3 notation=*‹*binder INT*››*INT* -<-./ -)› [*0*, *0*, *10*] *10*)

**syntax** (*latex* **output**)
  *-UNION-le* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(3$\bigcup$(‹unbreakable›- $\leq$ -)/ -)› [0, 0, 10] 10)
  *-UNION-less* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(3$\bigcup$(‹unbreakable›- $<$ -)/ -)› [0, 0, 10] 10)
  *-INTER-le* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(3$\bigcap$(‹unbreakable›- $\leq$ -)/ -)› [0, 0, 10] 10)
  *-INTER-less* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(3$\bigcap$(‹unbreakable›- $<$ -)/ -)› [0, 0, 10] 10)

**syntax**
  *-UNION-le* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(‹indent=3 notation=‹binder $\bigcup$››$\bigcup$-$\leq$-./ -)› [0, 0, 10] 10)
  *-UNION-less* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(‹indent=3 notation=‹binder $\bigcup$››$\bigcup$-$<$-./ -)› [0, 0, 10] 10)
  *-INTER-le* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(‹indent=3 notation=‹binder $\bigcap$››$\bigcap$-$\leq$-./ -)› [0, 0, 10] 10)
  *-INTER-less* :: $'a \Rightarrow 'a \Rightarrow 'b\ set \Rightarrow 'b\ set$ (‹(‹indent=3 notation=‹binder $\bigcap$››$\bigcap$-$<$-./ -)› [0, 0, 10] 10)

**syntax-consts**
  *-UNION-le -UNION-less* $\rightleftharpoons$ *Union* **and**
  *-INTER-le -INTER-less* $\rightleftharpoons$ *Inter*

**translations**
  $\bigcup i\leq n.\ A \rightleftharpoons \bigcup i\in\{..n\}.\ A$
  $\bigcup i<n.\ A \rightleftharpoons \bigcup i\in\{..<n\}.\ A$
  $\bigcap i\leq n.\ A \rightleftharpoons \bigcap i\in\{..n\}.\ A$
  $\bigcap i<n.\ A \rightleftharpoons \bigcap i\in\{..<n\}.\ A$

## 61.1 Various equivalences

**lemma** (**in** *ord*) *lessThan-iff* [*iff*]: $(i \in lessThan\ k) = (i<k)$
⟨*proof*⟩

**lemma** *Compl-lessThan* [*simp*]:
  $!!k::\ 'a::linorder.\ -lessThan\ k = atLeast\ k$
⟨*proof*⟩

**lemma** *single-Diff-lessThan* [*simp*]: $!!k::\ 'a::preorder.\ \{k\} - lessThan\ k = \{k\}$
⟨*proof*⟩

**lemma** (**in** *ord*) *greaterThan-iff* [*iff*]: $(i \in greaterThan\ k) = (k<i)$
⟨*proof*⟩

**lemma** *Compl-greaterThan* [*simp*]:
  $!!k::\ 'a::linorder.\ -greaterThan\ k = atMost\ k$
⟨*proof*⟩

**lemma** *Compl-atMost* [*simp*]: $!!k:: 'a::linorder.\ -atMost\ k = greaterThan\ k$
⟨*proof*⟩

**lemma** (**in** *ord*) *atLeast-iff* [*iff*]: $(i \in atLeast\ k) = (k<=i)$
⟨*proof*⟩

**lemma** *Compl-atLeast* [*simp*]: $!!k:: 'a::linorder.\ -atLeast\ k = lessThan\ k$
⟨*proof*⟩

**lemma** (**in** *ord*) *atMost-iff* [*iff*]: $(i \in atMost\ k) = (i<=k)$
⟨*proof*⟩

**lemma** *atMost-Int-atLeast*: $!!n:: 'a::order.\ atMost\ n \cap atLeast\ n = \{n\}$
⟨*proof*⟩

**lemma** (**in** *linorder*) *lessThan-Int-lessThan*: $\{\ a <..\} \cap \{\ b <..\} = \{\ max\ a\ b <..\}$
⟨*proof*⟩

**lemma** (**in** *linorder*) *greaterThan-Int-greaterThan*: $\{..<\ a\} \cap \{..<\ b\} = \{..<\ min\ a\ b\}$
⟨*proof*⟩

## 61.2 Logical Equivalences for Set Inclusion and Equality

**lemma** *atLeast-empty-triv* [*simp*]: $\{\{\}..\} = UNIV$
⟨*proof*⟩

**lemma** *atMost-UNIV-triv* [*simp*]: $\{..UNIV\} = UNIV$
⟨*proof*⟩

**lemma** *atLeast-subset-iff* [*iff*]:
$(atLeast\ x \subseteq atLeast\ y) = (y \le (x::'a::preorder))$
⟨*proof*⟩

**lemma** *atLeast-eq-iff* [*iff*]:
$(atLeast\ x = atLeast\ y) = (x = (y::'a::order))$
⟨*proof*⟩

**lemma** *greaterThan-subset-iff* [*iff*]:
$(greaterThan\ x \subseteq greaterThan\ y) = (y \le (x::'a::linorder))$
⟨*proof*⟩

**lemma** *greaterThan-eq-iff* [*iff*]:
$(greaterThan\ x = greaterThan\ y) = (x = (y::'a::linorder))$
⟨*proof*⟩

**lemma** *atMost-subset-iff* [*iff*]: $(atMost\ x \subseteq atMost\ y) = (x \le (y::'a::preorder))$
⟨*proof*⟩

**lemma** *atMost-eq-iff* [*iff*]: $(atMost\ x = atMost\ y) = (x = (y::'a::order))$
⟨*proof*⟩

**lemma** *lessThan-subset-iff* [*iff*]:
$(lessThan\ x \subseteq lessThan\ y) = (x \leq (y::'a::linorder))$
⟨*proof*⟩

**lemma** *lessThan-eq-iff* [*iff*]:
$(lessThan\ x = lessThan\ y) = (x = (y::'a::linorder))$
⟨*proof*⟩

**lemma** *lessThan-strict-subset-iff*:
**fixes** $m\ n :: 'a::linorder$
**shows** $\{..<m\} < \{..<n\} \longleftrightarrow m < n$
⟨*proof*⟩

**lemma** (**in** *linorder*) *Ici-subset-Ioi-iff*: $\{a\ ..\} \subseteq \{b <..\} \longleftrightarrow b < a$
⟨*proof*⟩

**lemma** (**in** *linorder*) *Iic-subset-Iio-iff*: $\{..\ a\} \subseteq \{..< b\} \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** (**in** *preorder*) *Ioi-le-Ico*: $\{a <..\} \subseteq \{a\ ..\}$
⟨*proof*⟩

## 61.3 Two-sided intervals

**context** *ord*
**begin**

**lemma** *greaterThanLessThan-iff* [*simp*]: $(i \in \{l<..<u\}) = (l < i \wedge i < u)$
⟨*proof*⟩

**lemma** *atLeastLessThan-iff* [*simp*]: $(i \in \{l..<u\}) = (l \leq i \wedge i < u)$
⟨*proof*⟩

**lemma** *greaterThanAtMost-iff* [*simp*]: $(i \in \{l<..u\}) = (l < i \wedge i \leq u)$
⟨*proof*⟩

**lemma** *atLeastAtMost-iff* [*simp*]: $(i \in \{l..u\}) = (l \leq i \wedge i \leq u)$
⟨*proof*⟩

The above four lemmas could be declared as iffs. Unfortunately this breaks many proofs. Since it only helps blast, it is better to leave them alone.

**lemma** *greaterThanLessThan-eq*: $\{\ a <..< b\} = \{\ a <..\} \cap \{..< b\ \}$
⟨*proof*⟩

**lemma** (**in** *order*) *atLeastLessThan-eq-atLeastAtMost-diff*:
$\{a..<b\} = \{a..b\} - \{b\}$

⟨*proof*⟩

**lemma** (**in** *order*) *greaterThanAtMost-eq-atLeastAtMost-diff*:
$\{a<..b\} = \{a..b\} - \{a\}$
⟨*proof*⟩

**end**

### 61.3.1 Emptyness, singletons, subset

**context** *preorder*
**begin**

**lemma** *atLeastatMost-empty-iff* [*simp*]:
$\{a..b\} = \{\} \longleftrightarrow (\neg\ a \leq b)$
⟨*proof*⟩

**lemma** *atLeastatMost-empty-iff2* [*simp*]:
$\{\} = \{a..b\} \longleftrightarrow (\neg\ a \leq b)$
⟨*proof*⟩

**lemma** *atLeastLessThan-empty-iff* [*simp*]:
$\{a..<b\} = \{\} \longleftrightarrow (\neg\ a < b)$
⟨*proof*⟩

**lemma** *atLeastLessThan-empty-iff2* [*simp*]:
$\{\} = \{a..<b\} \longleftrightarrow (\neg\ a < b)$
⟨*proof*⟩

**lemma** *greaterThanAtMost-empty-iff* [*simp*]: $\{k<..l\} = \{\} \longleftrightarrow \neg\ k < l$
⟨*proof*⟩

**lemma** *greaterThanAtMost-empty-iff2* [*simp*]: $\{\} = \{k<..l\} \longleftrightarrow \neg\ k < l$
⟨*proof*⟩

**lemma** *atLeastatMost-subset-iff* [*simp*]:
$\{a..b\} \leq \{c..d\} \longleftrightarrow (\neg\ a \leq b) \vee c \leq a \wedge b \leq d$
⟨*proof*⟩

**lemma** *atLeastatMost-psubset-iff*:
$\{a..b\} < \{c..d\} \longleftrightarrow$
$((\neg\ a \leq b) \vee c \leq a \wedge b \leq d \wedge (c < a \vee b < d)) \wedge c \leq d$
⟨*proof*⟩

**lemma** *atLeastAtMost-subseteq-atLeastLessThan-iff*:
$\{a..b\} \subseteq \{c\ ..<\ d\} \longleftrightarrow (a \leq b \longrightarrow c \leq a \wedge b < d)$
⟨*proof*⟩

**lemma** *Icc-subset-Ici-iff* [*simp*]:

$\{l..h\} \subseteq \{l'..\} = (\neg\ l \leq h \lor l \geq l')$
⟨*proof*⟩

**lemma** *Icc-subset-Iic-iff* [*simp*]:
$\{l..h\} \subseteq \{..h'\} = (\neg\ l \leq h \lor h \leq h')$
⟨*proof*⟩

**lemma** *not-Ici-eq-empty*[*simp*]: $\{l..\} \neq \{\}$
⟨*proof*⟩

**lemma** *not-Iic-eq-empty*[*simp*]: $\{..h\} \neq \{\}$
⟨*proof*⟩

**lemmas** *not-empty-eq-Ici-eq-empty*[*simp*] = *not-Ici-eq-empty*[*symmetric*]
**lemmas** *not-empty-eq-Iic-eq-empty*[*simp*] = *not-Iic-eq-empty*[*symmetric*]

**end**

**context** *order*
**begin**

**lemma** *atLeastatMost-empty*[*simp*]: $b < a \Longrightarrow \{a..b\} = \{\}$
**and** *atLeastatMost-empty'*[*simp*]: $\neg\ a \leq b \Longrightarrow \{a..b\} = \{\}$
⟨*proof*⟩

**lemma** *atLeastLessThan-empty*[*simp*]:
$b \leq a \Longrightarrow \{a..<b\} = \{\}$
⟨*proof*⟩

**lemma** *greaterThanAtMost-empty*[*simp*]: $l \leq k \Longrightarrow \{k<..l\} = \{\}$
⟨*proof*⟩

**lemma** *greaterThanLessThan-empty*[*simp*]:$l \leq k \Longrightarrow \{k<..<l\} = \{\}$
⟨*proof*⟩

**lemma** *atLeastAtMost-singleton* [*simp*]: $\{a..a\} = \{a\}$
⟨*proof*⟩

**lemma** *atLeastAtMost-singleton'*: $a = b \Longrightarrow \{a\ ..\ b\} = \{a\}$ ⟨*proof*⟩

**lemma** *Icc-eq-Icc*[*simp*]:
$\{l..h\} = \{l'..h'\} = (l=l' \land h=h' \lor \neg\ l \leq h \land \neg\ l' \leq h')$
⟨*proof*⟩

**lemma** (**in** *linorder*) *Ico-eq-Ico*:
$\{l..<h\} = \{l'..<h'\} = (l=l' \land h=h' \lor \neg\ l<h \land \neg\ l'<h')$
⟨*proof*⟩

**lemma** *atLeastAtMost-singleton-iff* [*simp*]:

$\{a .. b\} = \{c\} \longleftrightarrow a = b \wedge b = c$
$\langle proof \rangle$

Quantifiers

**lemma** *ex-interval-simps*:
$(\exists x \in \{..<u\}. P x) \longleftrightarrow (\exists x<u. P x)$
$(\exists x \in \{..u\}. P x) \longleftrightarrow (\exists x\leq u. P x)$
$(\exists x \in \{l<..\}. P x) \longleftrightarrow (\exists x>l. P x)$
$(\exists x \in \{l..\}. P x) \longleftrightarrow (\exists x\geq l. P x)$
$(\exists x \in \{l<..<u\}. P x) \longleftrightarrow (\exists x. l<x \wedge x<u \wedge P x)$
$(\exists x \in \{l..<u\}. P x) \longleftrightarrow (\exists x. l\leq x \wedge x<u \wedge P x)$
$(\exists x \in \{l<..u\}. P x) \longleftrightarrow (\exists x. l<x \wedge x\leq u \wedge P x)$
$(\exists x \in \{l..u\}. P x) \longleftrightarrow (\exists x. l\leq x \wedge x\leq u \wedge P x)$
$\langle proof \rangle$

**lemma** *all-interval-simps*:
$(\forall x \in \{..<u\}. P x) \longleftrightarrow (\forall x<u. P x)$
$(\forall x \in \{..u\}. P x) \longleftrightarrow (\forall x\leq u. P x)$
$(\forall x \in \{l<..\}. P x) \longleftrightarrow (\forall x>l. P x)$
$(\forall x \in \{l..\}. P x) \longleftrightarrow (\forall x\geq l. P x)$
$(\forall x \in \{l<..<u\}. P x) \longleftrightarrow (\forall x. l<x \longrightarrow x<u \longrightarrow P x)$
$(\forall x \in \{l..<u\}. P x) \longleftrightarrow (\forall x. l\leq x \longrightarrow x<u \longrightarrow P x)$
$(\forall x \in \{l<..u\}. P x) \longleftrightarrow (\forall x. l<x \longrightarrow x\leq u \longrightarrow P x)$
$(\forall x \in \{l..u\}. P x) \longleftrightarrow (\forall x. l\leq x \longrightarrow x\leq u \longrightarrow P x)$
$\langle proof \rangle$

The following results generalise their namesakes in *HOL.Nat* to intervals

**lemma** *lift-Suc-mono-le-ivl*:
**assumes** *mono*: $\bigwedge n.\ n\in N \Longrightarrow f\ n \leq f\ (Suc\ n)$
**and** $n \leq n'$ **and** *subN*: $\{n..<n'\} \subseteq N$
**shows** $f\ n \leq f\ n'$
$\langle proof \rangle$

**lemma** *lift-Suc-antimono-le-ivl*:
**assumes** *mono*: $\bigwedge n.\ n\in N \Longrightarrow f\ n \geq f\ (Suc\ n)$
**and** $n \leq n'$ **and** *subN*: $\{n..<n'\} \subseteq N$
**shows** $f\ n \geq f\ n'$
$\langle proof \rangle$

**lemma** *lift-Suc-mono-less-ivl*:
**assumes** *mono*: $\bigwedge n.\ n\in N \Longrightarrow f\ n < f\ (Suc\ n)$
**and** $n < n'$ **and** *subN*: $\{n..<n'\} \subseteq N$
**shows** $f\ n < f\ n'$
$\langle proof \rangle$

**end**

**context** *no-top*
**begin**

**lemma** *not-UNIV-le-Icc*[*simp*]: $\neg$ *UNIV* $\subseteq \{l..h\}$
⟨*proof*⟩

**lemma** *not-UNIV-le-Iic*[*simp*]: $\neg$ *UNIV* $\subseteq \{..h\}$
⟨*proof*⟩

**lemma** *not-Ici-le-Icc*[*simp*]: $\neg \{l..\} \subseteq \{l'..h'\}$
⟨*proof*⟩

**lemma** *not-Ici-le-Iic*[*simp*]: $\neg \{l..\} \subseteq \{..h'\}$
⟨*proof*⟩

**end**

**context** *no-bot*
**begin**

**lemma** *not-UNIV-le-Ici*[*simp*]: $\neg$ *UNIV* $\subseteq \{l..\}$
⟨*proof*⟩

**lemma** *not-Iic-le-Icc*[*simp*]: $\neg \{..h\} \subseteq \{l'..h'\}$
⟨*proof*⟩

**lemma** *not-Iic-le-Ici*[*simp*]: $\neg \{..h\} \subseteq \{l'..\}$
⟨*proof*⟩

**end**

**context** *no-top*
**begin**

**lemma** *not-UNIV-eq-Icc*[*simp*]: $\neg$ *UNIV* $= \{l'..h'\}$
⟨*proof*⟩

**lemmas** *not-Icc-eq-UNIV*[*simp*] = *not-UNIV-eq-Icc*[*symmetric*]

**lemma** *not-UNIV-eq-Iic*[*simp*]: $\neg$ *UNIV* $= \{..h'\}$
⟨*proof*⟩

**lemmas** *not-Iic-eq-UNIV*[*simp*] = *not-UNIV-eq-Iic*[*symmetric*]

**lemma** *not-Icc-eq-Ici*[*simp*]: $\neg \{l..h\} = \{l'..\}$
⟨*proof*⟩

**lemmas** *not-Ici-eq-Icc*[*simp*] = *not-Icc-eq-Ici*[*symmetric*]

**lemma** *not-Iic-eq-Ici*[*simp*]: $\neg$ $\{..h\} = \{l'..\}$
⟨*proof*⟩

**lemmas** *not-Ici-eq-Iic*[*simp*] = *not-Iic-eq-Ici*[*symmetric*]

**end**

**context** *no-bot*
**begin**

**lemma** *not-UNIV-eq-Ici*[*simp*]: $\neg$ *UNIV* = $\{l'..\}$
⟨*proof*⟩

**lemmas** *not-Ici-eq-UNIV*[*simp*] = *not-UNIV-eq-Ici*[*symmetric*]

**lemma** *not-Icc-eq-Iic*[*simp*]: $\neg$ $\{l..h\} = \{..h'\}$
⟨*proof*⟩

**lemmas** *not-Iic-eq-Icc*[*simp*] = *not-Icc-eq-Iic*[*symmetric*]

**end**

**context** *dense-linorder*
**begin**

**lemma** *greaterThanLessThan-empty-iff*[*simp*]:
  $\{\ a\ {<}..{<}\ b\ \} = \{\} \longleftrightarrow b \le a$
  ⟨*proof*⟩

**lemma** *greaterThanLessThan-empty-iff2*[*simp*]:
  $\{\} = \{\ a\ {<}..{<}\ b\ \} \longleftrightarrow b \le a$
  ⟨*proof*⟩

**lemma** *atLeastLessThan-subseteq-atLeastAtMost-iff*:
  $\{a\ ..{<}\ b\} \subseteq \{\ c\ ..\ d\ \} \longleftrightarrow (a < b \longrightarrow c \le a \wedge b \le d)$
  ⟨*proof*⟩

**lemma** *greaterThanAtMost-subseteq-atLeastAtMost-iff*:
  $\{a\ {<}..\ b\} \subseteq \{\ c\ ..\ d\ \} \longleftrightarrow (a < b \longrightarrow c \le a \wedge b \le d)$
  ⟨*proof*⟩

**lemma** *greaterThanLessThan-subseteq-atLeastAtMost-iff*:
  $\{a\ {<}..{<}\ b\} \subseteq \{\ c\ ..\ d\ \} \longleftrightarrow (a < b \longrightarrow c \le a \wedge b \le d)$
  ⟨*proof*⟩

**lemma** *greaterThanLessThan-subseteq-greaterThanLessThan*:

$\{a <..< b\} \subseteq \{c <..< d\} \longleftrightarrow (a < b \longrightarrow a \geq c \land b \leq d)$
⟨*proof*⟩

**lemma** *greaterThanAtMost-subseteq-atLeastLessThan-iff*:
$\{a <.. b\} \subseteq \{ c ..< d \} \longleftrightarrow (a < b \longrightarrow c \leq a \land b < d)$
⟨*proof*⟩

**lemma** *greaterThanLessThan-subseteq-atLeastLessThan-iff*:
$\{a <..< b\} \subseteq \{ c ..< d \} \longleftrightarrow (a < b \longrightarrow c \leq a \land b \leq d)$
⟨*proof*⟩

**lemma** *greaterThanLessThan-subseteq-greaterThanAtMost-iff*:
$\{a <..< b\} \subseteq \{ c <.. d \} \longleftrightarrow (a < b \longrightarrow c \leq a \land b \leq d)$
⟨*proof*⟩

**end**

**context** *no-top*
**begin**

**lemma** *greaterThan-non-empty*[*simp*]: $\{x <..\} \neq \{\}$
⟨*proof*⟩

**end**

**context** *no-bot*
**begin**

**lemma** *lessThan-non-empty*[*simp*]: $\{..< x\} \neq \{\}$
⟨*proof*⟩

**end**

**lemma** (**in** *linorder*) *atLeastLessThan-subset-iff*:
$\{a..<b\} \subseteq \{c..<d\} \Longrightarrow b \leq a \lor c \leq a \land b \leq d$
⟨*proof*⟩

**lemma** *atLeastLessThan-inj*:
**fixes** $a\ b\ c\ d$ :: $'a$::*linorder*
**assumes** *eq*: $\{a ..< b\} = \{c ..< d\}$ **and** $a < b\ c < d$
**shows** $a = c\ b = d$
⟨*proof*⟩

**lemma** *atLeastLessThan-eq-iff*:
**fixes** $a\ b\ c\ d$ :: $'a$::*linorder*
**assumes** $a < b\ c < d$
**shows** $\{a ..< b\} = \{c ..< d\} \longleftrightarrow a = c \land b = d$
⟨*proof*⟩

**lemma** (**in** *linorder*) *Ioc-inj*:
‹$\{a <.. b\} = \{c <.. d\} \longleftrightarrow (b \leq a \wedge d \leq c) \vee a = c \wedge b = d$› (**is** ‹$?P \longleftrightarrow ?Q$›)
⟨*proof*⟩

**lemma** (**in** *order*) *Iio-Int-singleton*: $\{..<k\} \cap \{x\} = (\mathit{if}\ x < k\ \mathit{then}\ \{x\}\ \mathit{else}\ \{\})$
⟨*proof*⟩

**lemma** (**in** *linorder*) *Ioc-subset-iff*: $\{a<..b\} \subseteq \{c<..d\} \longleftrightarrow (b \leq a \vee c \leq a \wedge b \leq d)$
⟨*proof*⟩

**lemma** (**in** *order-bot*) *atLeast-eq-UNIV-iff*: $\{x..\} = \mathit{UNIV} \longleftrightarrow x = \mathit{bot}$
⟨*proof*⟩

**lemma** (**in** *order-top*) *atMost-eq-UNIV-iff*: $\{..x\} = \mathit{UNIV} \longleftrightarrow x = \mathit{top}$
⟨*proof*⟩

**lemma** (**in** *bounded-lattice*) *atLeastAtMost-eq-UNIV-iff*:
$\{x..y\} = \mathit{UNIV} \longleftrightarrow (x = \mathit{bot} \wedge y = \mathit{top})$
⟨*proof*⟩

**lemma** *Iio-eq-empty-iff*: $\{..< n::'a::\{\mathit{linorder}, \mathit{order\text{-}bot}\}\} = \{\} \longleftrightarrow n = \mathit{bot}$
⟨*proof*⟩

**lemma** *lessThan-empty-iff*: $\{..< n::\mathit{nat}\} = \{\} \longleftrightarrow n = 0$
⟨*proof*⟩

**lemma** *mono-image-least*:
**assumes** *f-mono*: *mono f* **and** *f-img*: $f\ `\ \{m ..< n\} = \{m' ..< n'\}$ $m < n$
**shows** $f\ m = m'$
⟨*proof*⟩

## 61.4 Infinite intervals

**context** *dense-linorder*
**begin**

**lemma** *infinite-Ioo*:
**assumes** $a < b$
**shows** $\neg$ *finite* $\{a<..<b\}$
⟨*proof*⟩

**lemma** *infinite-Icc*: $a < b \Longrightarrow \neg$ *finite* $\{a .. b\}$
⟨*proof*⟩

**lemma** *infinite-Ico*: $a < b \Longrightarrow \neg$ *finite* $\{a ..< b\}$
⟨*proof*⟩

**lemma** *infinite-Ioc*: $a < b \Longrightarrow \neg$ *finite* $\{a <.. b\}$

⟨*proof*⟩

**lemma** *infinite-Ioo-iff* [*simp*]: *infinite* $\{a<..<b\} \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *infinite-Icc-iff* [*simp*]: *infinite* $\{a .. b\} \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *infinite-Ico-iff* [*simp*]: *infinite* $\{a..<b\} \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *infinite-Ioc-iff* [*simp*]: *infinite* $\{a<..b\} \longleftrightarrow a < b$
⟨*proof*⟩

**end**

**lemma** *infinite-Iio*: $\neg$ *finite* $\{..< a :: 'a :: \{no\text{-}bot, linorder\}\}$
⟨*proof*⟩

**lemma** *infinite-Iic*: $\neg$ *finite* $\{.. a :: 'a :: \{no\text{-}bot, linorder\}\}$
⟨*proof*⟩

**lemma** *infinite-Ioi*: $\neg$ *finite* $\{a :: 'a :: \{no\text{-}top, linorder\} <..\}$
⟨*proof*⟩

**lemma** *infinite-Ici*: $\neg$ *finite* $\{a :: 'a :: \{no\text{-}top, linorder\} ..\}$
⟨*proof*⟩

### 61.4.1 Intersection

**context** *linorder*
**begin**

**lemma** *Icc-minus-Ico* [*simp*]:
**assumes** $a \leq b$
**shows** $\{a..b'\} - \{a..<b\} = \{b..b'\}$
⟨*proof*⟩

**lemma** *Icc-minus-Ioc* [*simp*]:
**assumes** $a \leq b$
**shows** $\{a'..b\} - \{a<..b\} = \{a'..a\}$
⟨*proof*⟩

**lemma** *Icc-minus-Ioo* [*simp*]:
**assumes** $a \leq b$
**shows** $\{a..b\} - \{a<..<b\} = \{a,b\}$
⟨*proof*⟩

**lemma** *Int-atLeastAtMost*[*simp*]: $\{a..b\} \cap \{c..d\} = \{max\ a\ c\ ..\ min\ b\ d\}$

⟨*proof*⟩

**lemma** *Int-atLeastAtMostR1*[*simp*]: $\{..b\} \cap \{c..d\} = \{c \;..\; min\ b\ d\}$
⟨*proof*⟩

**lemma** *Int-atLeastAtMostR2*[*simp*]: $\{a..\} \cap \{c..d\} = \{max\ a\ c \;..\; d\}$
⟨*proof*⟩

**lemma** *Int-atLeastAtMostL1*[*simp*]: $\{a..b\} \cap \{..d\} = \{a \;..\; min\ b\ d\}$
⟨*proof*⟩

**lemma** *Int-atLeastAtMostL2*[*simp*]: $\{a..b\} \cap \{c..\} = \{max\ a\ c \;..\; b\}$
⟨*proof*⟩

**lemma** *Int-atLeastLessThan*[*simp*]: $\{a..<b\} \cap \{c..<d\} = \{max\ a\ c \;..<\; min\ b\ d\}$
⟨*proof*⟩

**lemma** *Int-greaterThanAtMost*[*simp*]: $\{a<..b\} \cap \{c<..d\} = \{max\ a\ c \;<..\; min\ b\ d\}$
⟨*proof*⟩

**lemma** *Int-greaterThanLessThan*[*simp*]: $\{a<..<b\} \cap \{c<..<d\} = \{max\ a\ c \;<..<\; min\ b\ d\}$
⟨*proof*⟩

**lemma** *Int-atMost*[*simp*]: $\{..a\} \cap \{..b\} = \{..\ min\ a\ b\}$
⟨*proof*⟩

**lemma** *Ioc-disjoint*: $\{a<..b\} \cap \{c<..d\} = \{\} \longleftrightarrow b \le a \lor d \le c \lor b \le c \lor d \le a$
⟨*proof*⟩

**end**

**context** *complete-lattice*
**begin**

**lemma**
**shows** *Sup-atLeast*[*simp*]: $Sup\ \{x\ ..\} = top$
**and** *Sup-greaterThanAtLeast*[*simp*]: $x < top \Longrightarrow Sup\ \{x\ <..\} = top$
**and** *Sup-atMost*[*simp*]: $Sup\ \{..\ y\} = y$
**and** *Sup-atLeastAtMost*[*simp*]: $x \le y \Longrightarrow Sup\ \{\ x\ ..\ y\} = y$
**and** *Sup-greaterThanAtMost*[*simp*]: $x < y \Longrightarrow Sup\ \{\ x\ <..\ y\} = y$
⟨*proof*⟩

**lemma**
**shows** *Inf-atMost*[*simp*]: $Inf\ \{..\ x\} = bot$
**and** *Inf-atMostLessThan*[*simp*]: $top < x \Longrightarrow Inf\ \{..<\ x\} = bot$
**and** *Inf-atLeast*[*simp*]: $Inf\ \{x\ ..\} = x$

**and** *Inf-atLeastAtMost*[*simp*]: $x \le y \Longrightarrow Inf\ \{x\ ..\ y\} = x$
**and** *Inf-atLeastLessThan*[*simp*]: $x < y \Longrightarrow Inf\ \{x\ ..< y\} = x$
⟨*proof*⟩

**end**

**lemma**
**fixes** $x\ y$ :: $'a$ :: {*complete-lattice*, *dense-linorder*}
**shows** *Sup-lessThan*[*simp*]: $Sup\ \{..< y\} = y$
**and** *Sup-atLeastLessThan*[*simp*]: $x < y \Longrightarrow Sup\ \{x\ ..< y\} = y$
**and** *Sup-greaterThanLessThan*[*simp*]: $x < y \Longrightarrow Sup\ \{x <..< y\} = y$
**and** *Inf-greaterThan*[*simp*]: $Inf\ \{x <..\} = x$
**and** *Inf-greaterThanAtMost*[*simp*]: $x < y \Longrightarrow Inf\ \{x <..\ y\} = x$
**and** *Inf-greaterThanLessThan*[*simp*]: $x < y \Longrightarrow Inf\ \{x <..< y\} = x$
⟨*proof*⟩

## 61.5 Intervals of natural numbers

### 61.5.1 The Constant *lessThan*

**lemma** *lessThan-0* [*simp*]: *lessThan* (*0*::*nat*) = {}
⟨*proof*⟩

**lemma** *lessThan-Suc*: *lessThan* (*Suc k*) = *insert k* (*lessThan k*)
⟨*proof*⟩

The following proof is convenient in induction proofs where new elements get indices at the beginning. So it is used to transform $\{..<Suc\ n\}$ to *0* and $\{..<n\}$.

**lemma** *zero-notin-Suc-image* [*simp*]: $0 \notin Suc$ ‘ $A$
⟨*proof*⟩

**lemma** *lessThan-Suc-eq-insert-0*: $\{..<Suc\ n\} = insert\ 0\ (Suc$ ‘ $\{..<n\})$
⟨*proof*⟩

**lemma** *lessThan-Suc-atMost*: *lessThan* (*Suc k*) = *atMost k*
⟨*proof*⟩

**lemma** *atMost-Suc-eq-insert-0*: $\{..\ Suc\ n\} = insert\ 0\ (Suc$ ‘ $\{..\ n\})$
⟨*proof*⟩

**lemma** *UN-lessThan-UNIV*: $(\bigcup m::nat.\ lessThan\ m) = UNIV$
⟨*proof*⟩

### 61.5.2 The Constant *greaterThan*

**lemma** *greaterThan-0*: *greaterThan 0* = *range Suc*
⟨*proof*⟩

**lemma** *greaterThan-Suc*: *greaterThan* (*Suc k*) = *greaterThan k* − {*Suc k*}

⟨*proof*⟩

**lemma** *INT-greaterThan-UNIV*: $(\bigcap m::nat.\ greaterThan\ m) = \{\}$
⟨*proof*⟩

### 61.5.3 The Constant *atLeast*

**lemma** *atLeast-0* [*simp*]: $atLeast\ (0::nat) = UNIV$
⟨*proof*⟩

**lemma** *atLeast-Suc*: $atLeast\ (Suc\ k) = atLeast\ k - \{k\}$
⟨*proof*⟩

**lemma** *atLeast-Suc-greaterThan*: $atLeast\ (Suc\ k) = greaterThan\ k$
⟨*proof*⟩

**lemma** *UN-atLeast-UNIV*: $(\bigcup m::nat.\ atLeast\ m) = UNIV$
⟨*proof*⟩

### 61.5.4 The Constant *atMost*

**lemma** *atMost-0* [*simp*]: $atMost\ (0::nat) = \{0\}$
⟨*proof*⟩

**lemma** *atMost-Suc*: $atMost\ (Suc\ k) = insert\ (Suc\ k)\ (atMost\ k)$
⟨*proof*⟩

**lemma** *UN-atMost-UNIV*: $(\bigcup m::nat.\ atMost\ m) = UNIV$
⟨*proof*⟩

### 61.5.5 The Constant *atLeastLessThan*

The orientation of the following 2 rules is tricky. The lhs is defined in terms of the rhs. Hence the chosen orientation makes sense in this theory — the reverse orientation complicates proofs (eg nontermination). But outside, when the definition of the lhs is rarely used, the opposite orientation seems preferable because it reduces a specific concept to a more general one.

**lemma** *atLeast0LessThan* [*code-abbrev*]: $\{0::nat..<n\} = \{..<n\}$
⟨*proof*⟩

**lemma** *atLeast0AtMost* [*code-abbrev*]: $\{0..n::nat\} = \{..n\}$
⟨*proof*⟩

**lemma** *lessThan-atLeast0*: $\{..<n\} = \{0::nat..<n\}$
⟨*proof*⟩

**lemma** *atMost-atLeast0*: $\{..n\} = \{0::nat..n\}$
⟨*proof*⟩

**lemma** *atLeastLessThan0*: $\{m..<0::nat\} = \{\}$
⟨*proof*⟩

**lemma** *atLeast0-lessThan-Suc*: $\{0..<Suc\ n\} = insert\ n\ \{0..<n\}$
⟨*proof*⟩

**lemma** *atLeast0-lessThan-Suc-eq-insert-0*: $\{0..<Suc\ n\} = insert\ 0\ (Suc\ `\ \{0..<n\})$
⟨*proof*⟩

### 61.5.6 The Constant *atLeastAtMost*

**lemma** *Icc-eq-insert-lb-nat*: $m \leq n \Longrightarrow \{m..n\} = insert\ m\ \{Suc\ m..n\}$
⟨*proof*⟩

**lemma** *atLeast0-atMost-Suc*:
$\{0..Suc\ n\} = insert\ (Suc\ n)\ \{0..n\}$
⟨*proof*⟩

**lemma** *atLeast0-atMost-Suc-eq-insert-0*:
$\{0..Suc\ n\} = insert\ 0\ (Suc\ `\ \{0..n\})$
⟨*proof*⟩

### 61.5.7 Intervals of nats with *Suc*

Not a simprule because the RHS is too messy.

**lemma** *atLeastLessThanSuc*:
$\{m..<Suc\ n\} = (if\ m \leq n\ then\ insert\ n\ \{m..<n\}\ else\ \{\})$
⟨*proof*⟩

**lemma** *atLeastLessThan-singleton* [*simp*]: $\{m..<Suc\ m\} = \{m\}$
⟨*proof*⟩

**lemma** *atLeastLessThanSuc-atLeastAtMost*: $\{l..<Suc\ u\} = \{l..u\}$
⟨*proof*⟩

**lemma** *atLeastSucAtMost-greaterThanAtMost*: $\{Suc\ l..u\} = \{l<..u\}$
⟨*proof*⟩

**lemma** *atLeastSucLessThan-greaterThanLessThan*: $\{Suc\ l..<u\} = \{l<..<u\}$
⟨*proof*⟩

**lemma** *atLeastAtMostSuc-conv*: $m \leq Suc\ n \Longrightarrow \{m..Suc\ n\} = insert\ (Suc\ n)\ \{m..n\}$
⟨*proof*⟩

**lemma** *atLeastAtMost-insertL*: $m \leq n \Longrightarrow insert\ m\ \{Suc\ m..n\} = \{m\ ..n\}$
⟨*proof*⟩

The analogous result is useful on *int*:

**lemma** *atLeastAtMostPlus1-int-conv*:
$m \le 1+n \Longrightarrow \{m..1+n\} = insert\ (1+n)\ \{m..n::int\}$
⟨*proof*⟩

**lemma** *atLeastLessThan-add-Un*: $i \le j \Longrightarrow \{i..<j+k\} = \{i..<j\} \cup \{j..<j+k::nat\}$
⟨*proof*⟩

### 61.5.8 Intervals and numerals

**lemma** *lessThan-nat-numeral*: — Evaluation for specific numerals
$lessThan\ (numeral\ k :: nat) = insert\ (pred\text{-}numeral\ k)\ (lessThan\ (pred\text{-}numeral\ k))$
⟨*proof*⟩

**lemma** *atMost-nat-numeral*: — Evaluation for specific numerals
$atMost\ (numeral\ k :: nat) = insert\ (numeral\ k)\ (atMost\ (pred\text{-}numeral\ k))$
⟨*proof*⟩

**lemma** *atLeastLessThan-nat-numeral*: — Evaluation for specific numerals
$atLeastLessThan\ m\ (numeral\ k :: nat) =$
$(if\ m \le (pred\text{-}numeral\ k)\ then\ insert\ (pred\text{-}numeral\ k)\ (atLeastLessThan\ m\ (pred\text{-}numeral\ k))$
$else\ \{\})$
⟨*proof*⟩

### 61.5.9 Image

**context** *linordered-semidom*
**begin**

**lemma** *image-add-atLeast*[*simp*]: $plus\ k\ `\ \{i..\} = \{k + i..\}$
⟨*proof*⟩

**lemma** *image-add-atLeastAtMost* [*simp*]:
$plus\ k\ `\ \{i..j\} = \{i + k..j + k\}$ (**is** $?A = ?B$)
⟨*proof*⟩

**lemma** *image-add-atLeastAtMost'* [*simp*]:
$(\lambda n.\ n + k)\ `\ \{i..j\} = \{i + k..j + k\}$
⟨*proof*⟩

**lemma** *image-add-atLeastLessThan* [*simp*]:
$plus\ k\ `\ \{i..<j\} = \{i + k..<j + k\}$
⟨*proof*⟩

**lemma** *image-add-atLeastLessThan'* [*simp*]:
$(\lambda n.\ n + k)\ `\ \{i..<j\} = \{i + k..<j + k\}$
⟨*proof*⟩

**lemma** *image-add-greaterThanAtMost*[*simp*]: $(+)\ c\ `\ \{a<..b\} = \{c + a<..c + b\}$

⟨*proof*⟩

**end**

**context** *ordered-ab-group-add*
**begin**

**lemma**
**fixes** $x :: 'a$
**shows** *image-uminus-greaterThan*[*simp*]: $uminus$ ‘ $\{x<..\} = \{..<-x\}$
**and** *image-uminus-atLeast*[*simp*]: $uminus$ ‘ $\{x..\} = \{..-x\}$
⟨*proof*⟩

**lemma**
**fixes** $x :: 'a$
**shows** *image-uminus-lessThan*[*simp*]: $uminus$ ‘ $\{..<x\} = \{-x<..\}$
**and** *image-uminus-atMost*[*simp*]: $uminus$ ‘ $\{..x\} = \{-x..\}$
⟨*proof*⟩

**lemma**
**fixes** $x :: 'a$
**shows** *image-uminus-atLeastAtMost*[*simp*]: $uminus$ ‘ $\{x..y\} = \{-y..-x\}$
**and** *image-uminus-greaterThanAtMost*[*simp*]: $uminus$ ‘ $\{x<..y\} = \{-y..<-x\}$
**and** *image-uminus-atLeastLessThan*[*simp*]: $uminus$ ‘ $\{x..<y\} = \{-y<..-x\}$
**and** *image-uminus-greaterThanLessThan*[*simp*]: $uminus$ ‘ $\{x<..<y\} = \{-y<..<-x\}$
⟨*proof*⟩

**lemma** *image-add-atMost*[*simp*]: $(+)\ c$ ‘ $\{..a\} = \{..c + a\}$
⟨*proof*⟩

**end**

**lemma** *image-Suc-atLeastAtMost* [*simp*]:
$Suc$ ‘ $\{i..j\} = \{Suc\ i..Suc\ j\}$
⟨*proof*⟩

**lemma** *image-Suc-atLeastLessThan* [*simp*]:
$Suc$ ‘ $\{i..<j\} = \{Suc\ i..<Suc\ j\}$
⟨*proof*⟩

**corollary** *image-Suc-atMost*:
$Suc$ ‘ $\{..n\} = \{1..Suc\ n\}$
⟨*proof*⟩

**corollary** *image-Suc-lessThan*:
$Suc$ ‘ $\{..<n\} = \{1..n\}$
⟨*proof*⟩

**lemma** *image-diff-atLeastAtMost* [*simp*]:

**fixes** $d::'a::linordered\text{-}idom$ **shows** $((-)\ d\ `\ \{a..b\}) = \{d-b..d-a\}$
⟨proof⟩

**lemma** *image-diff-atLeastLessThan* [*simp*]:
**fixes** $a\ b\ c::'a::linordered\text{-}idom$
**shows** $(-)\ c\ `\ \{a..<b\} = \{c - b<..c - a\}$
⟨proof⟩

**lemma** *image-minus-const-greaterThanAtMost*[*simp*]:
**fixes** $a\ b\ c::'a::linordered\text{-}idom$
**shows** $(-)\ c\ `\ \{a<..b\} = \{c - b..<c - a\}$
⟨proof⟩

**lemma** *image-minus-const-atLeast*[*simp*]:
**fixes** $a\ c::'a::linordered\text{-}idom$
**shows** $(-)\ c\ `\ \{a..\} = \{..c - a\}$
⟨proof⟩

**lemma** *image-minus-const-AtMost*[*simp*]:
**fixes** $b\ c::'a::linordered\text{-}idom$
**shows** $(-)\ c\ `\ \{..b\} = \{c - b..\}$
⟨proof⟩

**lemma** *image-minus-const-atLeastAtMost'* [*simp*]:
$(\lambda t.\ t-d)`\{a..b\} = \{a-d..b-d\}$ **for** $d::'a::linordered\text{-}idom$
⟨proof⟩

**context** *linordered-field*
**begin**

**lemma** *image-mult-atLeastAtMost* [*simp*]:
$((*)\ d\ `\ \{a..b\}) = \{d*a..d*b\}$ **if** $d>0$
⟨proof⟩

**lemma** *image-divide-atLeastAtMost* [*simp*]:
$((\lambda c.\ c\ /\ d)\ `\ \{a..b\}) = \{a/d..b/d\}$ **if** $d>0$
⟨proof⟩

**lemma** *image-mult-atLeastAtMost-if*:
$(*)\ c\ `\ \{x\ ..\ y\} =$
$(\textit{if}\ c > 0\ \textit{then}\ \{c * x\ ..\ c * y\}\ \textit{else if}\ x \le y\ \textit{then}\ \{c * y\ ..\ c * x\}\ \textit{else}\ \{\})$
⟨proof⟩

**lemma** *image-mult-atLeastAtMost-if'*:
$(\lambda x.\ x * c)\ `\ \{x..y\} =$
$(\textit{if}\ x \le y\ \textit{then if}\ c > 0\ \textit{then}\ \{x * c\ ..\ y * c\}\ \textit{else}\ \{y * c\ ..\ x * c\}\ \textit{else}\ \{\})$
⟨proof⟩

**lemma** *image-affinity-atLeastAtMost*:

$((\lambda x.\ m * x + c)\ `\ \{a..b\}) = (if\ \{a..b\} = \{\}\ then\ \{\}$
$\quad else\ if\ 0 \leq m\ then\ \{m * a + c\ ..\ m * b + c\}$
$\quad else\ \{m * b + c\ ..\ m * a + c\})$
$\langle proof \rangle$

**lemma** *image-affinity-atLeastAtMost-diff*:
$((\lambda x.\ m{*}x - c)\ `\ \{a..b\}) = (if\ \{a..b\}{=}\{\}\ then\ \{\}$
$\quad else\ if\ 0 \leq m\ then\ \{m{*}a - c\ ..\ m{*}b - c\}$
$\quad else\ \{m{*}b - c\ ..\ m{*}a - c\})$
$\langle proof \rangle$

**lemma** *image-affinity-atLeastAtMost-div*:
$((\lambda x.\ x/m + c)\ `\ \{a..b\}) = (if\ \{a..b\}{=}\{\}\ then\ \{\}$
$\quad else\ if\ 0 \leq m\ then\ \{a/m + c\ ..\ b/m + c\}$
$\quad else\ \{b/m + c\ ..\ a/m + c\})$
$\langle proof \rangle$

**lemma** *image-affinity-atLeastAtMost-div-diff*:
$((\lambda x.\ x/m - c)\ `\ \{a..b\}) = (if\ \{a..b\}{=}\{\}\ then\ \{\}$
$\quad else\ if\ 0 \leq m\ then\ \{a/m - c\ ..\ b/m - c\}$
$\quad else\ \{b/m - c\ ..\ a/m - c\})$
$\langle proof \rangle$

**end**

**lemma** *atLeast1-lessThan-eq-remove0*:
$\{Suc\ 0..{<}n\} = \{..{<}n\} - \{0\}$
$\langle proof \rangle$

**lemma** *atLeast1-atMost-eq-remove0*:
$\{Suc\ 0..n\} = \{..n\} - \{0\}$
$\langle proof \rangle$

**lemma** *image-add-int-atLeastLessThan*:
$(\lambda x.\ x + (l{::}int))\ `\ \{0..{<}u{-}l\} = \{l..{<}u\}$
$\langle proof \rangle$

**lemma** *image-minus-const-atLeastLessThan-nat*:
**fixes** $c :: nat$
**shows** $(\lambda i.\ i - c)\ `\ \{x\ ..{<}\ y\} =$
$\quad (if\ c < y\ then\ \{x - c\ ..{<}\ y - c\}\ else\ if\ x < y\ then\ \{0\}\ else\ \{\})$
(**is** - = *?right*)
$\langle proof \rangle$

**lemma** *image-int-atLeastLessThan*:
$int\ `\ \{a..{<}b\} = \{int\ a..{<}int\ b\}$
$\langle proof \rangle$

**lemma** *image-int-atLeastAtMost*:

$int$ ‘ $\{a..b\} = \{int\ a..int\ b\}$
⟨proof⟩

### 61.5.10 Finiteness

**lemma** *finite-lessThan* [*iff*]: **fixes** $k$ :: *nat* **shows** *finite* $\{..<k\}$
⟨proof⟩

**lemma** *finite-atMost* [*iff*]: **fixes** $k$ :: *nat* **shows** *finite* $\{..k\}$
⟨proof⟩

**lemma** *finite-greaterThanLessThan* [*iff*]:
**fixes** $l$ :: *nat* **shows** *finite* $\{l<..<u\}$
⟨proof⟩

**lemma** *finite-atLeastLessThan* [*iff*]:
**fixes** $l$ :: *nat* **shows** *finite* $\{l..<u\}$
⟨proof⟩

**lemma** *finite-greaterThanAtMost* [*iff*]:
**fixes** $l$ :: *nat* **shows** *finite* $\{l<..u\}$
⟨proof⟩

**lemma** *finite-atLeastAtMost* [*iff*]:
**fixes** $l$ :: *nat* **shows** *finite* $\{l..u\}$
⟨proof⟩

A bounded set of natural numbers is finite.

**lemma** *bounded-nat-set-is-finite*: $(\forall\, i{\in}N.\ i < (n{::}nat)) \Longrightarrow$ *finite* $N$
⟨proof⟩

A set of natural numbers is finite iff it is bounded.

**lemma** *finite-nat-set-iff-bounded*:
*finite*$(N{::}nat\ set) = (\exists\, m.\ \forall\, n{\in}N.\ n{<}m)$ (**is** $?F = ?B$)
⟨proof⟩

**lemma** *finite-nat-set-iff-bounded-le*: *finite*$(N{::}nat\ set) = (\exists\, m.\ \forall\, n{\in}N.\ n{\le}m)$
⟨proof⟩

**lemma** *finite-less-ub*:
$\bigwedge f{::}nat{\Rightarrow}nat.\ (!!n.\ n \le f\ n) \Longrightarrow$ *finite* $\{n.\ f\ n \le u\}$
⟨proof⟩

**lemma** *bounded-Max-nat*:
**fixes** $P$ :: $nat \Rightarrow bool$
**assumes** $x$: $P\ x$ **and** $M$: $\bigwedge x.\ P\ x \Longrightarrow x \le M$
**obtains** $m$ **where** $P\ m$ $\bigwedge x.\ P\ x \Longrightarrow x \le m$
⟨proof⟩

Any subset of an interval of natural numbers the size of the subset is exactly that interval.

**lemma** *subset-card-intvl-is-intvl*:
**assumes** $A \subseteq \{k..<k + card\ A\}$
**shows** $A = \{k..<k + card\ A\}$
⟨*proof*⟩

### 61.5.11 Proving Inclusions and Equalities between Unions

**lemma** *UN-le-eq-Un0*:
$(\bigcup i \le n::nat.\ M\ i) = (\bigcup i \in \{1..n\}.\ M\ i) \cup M\ 0$ (**is** $?A = ?B$)
⟨*proof*⟩

**lemma** *UN-le-add-shift*:
$(\bigcup i \le n::nat.\ M(i+k)) = (\bigcup i \in \{k..n+k\}.\ M\ i)$ (**is** $?A = ?B$)
⟨*proof*⟩

**lemma** *UN-le-add-shift-strict*:
$(\bigcup i < n::nat.\ M(i+k)) = (\bigcup i \in \{k..<n+k\}.\ M\ i)$ (**is** $?A = ?B$)
⟨*proof*⟩

**lemma** *UN-UN-finite-eq*: $(\bigcup n::nat.\ \bigcup i \in \{0..<n\}.\ A\ i) = (\bigcup n.\ A\ n)$
⟨*proof*⟩

**lemma** *UN-finite-subset*:
$(\bigwedge n::nat.\ (\bigcup i \in \{0..<n\}.\ A\ i) \subseteq C) \Longrightarrow (\bigcup n.\ A\ n) \subseteq C$
⟨*proof*⟩

**lemma** *UN-finite2-subset*:
**assumes** $\bigwedge n::nat.\ (\bigcup i \in \{0..<n\}.\ A\ i) \subseteq (\bigcup i \in \{0..<n + k\}.\ B\ i)$
**shows** $(\bigcup n.\ A\ n) \subseteq (\bigcup n.\ B\ n)$
⟨*proof*⟩

**lemma** *UN-finite2-eq*:
**assumes** $(\bigwedge n::nat.\ (\bigcup i \in \{0..<n\}.\ A\ i) = (\bigcup i \in \{0..<n + k\}.\ B\ i))$
**shows** $(\bigcup n.\ A\ n) = (\bigcup n.\ B\ n)$
⟨*proof*⟩

### 61.5.12 Cardinality

**lemma** *card-lessThan* [*simp*]: $card\ \{..<u\} = u$
⟨*proof*⟩

**lemma** *card-atMost* [*simp*]: $card\ \{..u\} = Suc\ u$
⟨*proof*⟩

**lemma** *card-atLeastLessThan* [*simp*]: $card\ \{l..<u\} = u - l$
⟨*proof*⟩

**lemma** *card-atLeastAtMost* [*simp*]: $card\ \{l..u\} = Suc\ u - l$
⟨*proof*⟩

**lemma** *card-greaterThanAtMost* [*simp*]: $card\ \{l<..u\} = u - l$
⟨*proof*⟩

**lemma** *card-greaterThanLessThan* [*simp*]: $card\ \{l<..<u\} = u - Suc\ l$
⟨*proof*⟩

**lemma** *subset-eq-atLeast0-lessThan-finite*:
**fixes** $n :: nat$
**assumes** $N \subseteq \{0..<n\}$
**shows** *finite N*
⟨*proof*⟩

**lemma** *subset-eq-atLeast0-atMost-finite*:
**fixes** $n :: nat$
**assumes** $N \subseteq \{0..n\}$
**shows** *finite N*
⟨*proof*⟩

**lemma** *ex-bij-betw-nat-finite*:
$finite\ M \Longrightarrow \exists h.\ bij\text{-}betw\ h\ \{0..<card\ M\}\ M$
⟨*proof*⟩

**lemma** *ex-bij-betw-finite-nat*:
$finite\ M \Longrightarrow \exists h.\ bij\text{-}betw\ h\ M\ \{0..<card\ M\}$
⟨*proof*⟩

**lemma** *finite-same-card-bij*:
$finite\ A \Longrightarrow finite\ B \Longrightarrow card\ A = card\ B \Longrightarrow \exists h.\ bij\text{-}betw\ h\ A\ B$
⟨*proof*⟩

**lemma** *ex-bij-betw-nat-finite-1*:
$finite\ M \Longrightarrow \exists h.\ bij\text{-}betw\ h\ \{1\ ..\ card\ M\}\ M$
⟨*proof*⟩

**lemma** *bij-betw-iff-card*:
**assumes** *finite A finite B*
**shows** $(\exists f.\ bij\text{-}betw\ f\ A\ B) \longleftrightarrow (card\ A = card\ B)$
⟨*proof*⟩

**lemma** *subset-eq-atLeast0-lessThan-card*:
**fixes** $n :: nat$
**assumes** $N \subseteq \{0..<n\}$
**shows** $card\ N \le n$
⟨*proof*⟩

Relational version of *card-inj-on-le*:

**lemma** *card-le-if-inj-on-rel*:
**assumes** *finite B*
$\bigwedge a.\ a \in A \Longrightarrow \exists\, b.\ b{\in}B \wedge r\ a\ b$
$\bigwedge a1\ a2\ b.\ [\![\ a1 \in A;\ \ a2 \in A;\ \ b \in B;\ \ r\ a1\ b;\ \ r\ a2\ b\ ]\!] \Longrightarrow a1 = a2$
**shows** $card\ A \leq card\ B$
⟨*proof*⟩

**lemma** *inj-on-funpow-least*:
‹$inj\text{-}on\ (\lambda k.\ (f\ \widehat{}\widehat{}\ k)\ s)\ \{0..{<}n\}$›
**if** ‹$(f\ \widehat{}\widehat{}\ n)\ s = s$› ‹$\bigwedge m.\ 0 < m \Longrightarrow m < n \Longrightarrow (f\ \widehat{}\widehat{}\ m)\ s \neq s$›
⟨*proof*⟩

## 61.6 Intervals of integers

**lemma** *atLeastLessThanPlusOne-atLeastAtMost-int*: $\{l..{<}u{+}1\} = \{l..(u{::}int)\}$
⟨*proof*⟩

**lemma** *atLeastPlusOneAtMost-greaterThanAtMost-int*: $\{l{+}1..u\} = \{l{<}..(u{::}int)\}$
⟨*proof*⟩

**lemma** *atLeastPlusOneLessThan-greaterThanLessThan-int*:
$\{l{+}1..{<}u\} = \{l{<}..{<}u{::}int\}$
⟨*proof*⟩

### 61.6.1 Finiteness

**lemma** *image-atLeastZeroLessThan-int*:
**assumes** $0 \leq u$
**shows** $\{(0{::}int)..{<}u\} = int\ `\ \{..{<}nat\ u\}$
⟨*proof*⟩

**lemma** *finite-atLeastZeroLessThan-int*: *finite* $\{(0{::}int)..{<}u\}$
⟨*proof*⟩

**lemma** *finite-atLeastLessThan-int* [*iff*]: *finite* $\{l..{<}u{::}int\}$
⟨*proof*⟩

**lemma** *finite-atLeastAtMost-int* [*iff*]: *finite* $\{l..(u{::}int)\}$
⟨*proof*⟩

**lemma** *finite-greaterThanAtMost-int* [*iff*]: *finite* $\{l{<}..(u{::}int)\}$
⟨*proof*⟩

**lemma** *finite-greaterThanLessThan-int* [*iff*]: *finite* $\{l{<}..{<}u{::}int\}$
⟨*proof*⟩

### 61.6.2 Cardinality

**lemma** *card-atLeastZeroLessThan-int*: $card\ \{(0{::}int)..{<}u\} = nat\ u$

⟨*proof*⟩

**lemma** *card-atLeastLessThan-int* [*simp*]: $card\ \{l..<u\} = nat\ (u - l)$
⟨*proof*⟩

**lemma** *card-atLeastAtMost-int* [*simp*]: $card\ \{l..u\} = nat\ (u - l + 1)$
⟨*proof*⟩

**lemma** *card-greaterThanAtMost-int* [*simp*]: $card\ \{l<..u\} = nat\ (u - l)$
⟨*proof*⟩

**lemma** *card-greaterThanLessThan-int* [*simp*]: $card\ \{l<..<u\} = nat\ (u - (l + 1))$
⟨*proof*⟩

**lemma** *finite-M-bounded-by-nat*: $finite\ \{k.\ P\ k \wedge k < (i{::}nat)\}$
⟨*proof*⟩

**lemma** *card-less*:
**assumes** *zero-in-M*: $0 \in M$
**shows** $card\ \{k \in M.\ k < Suc\ i\} \neq 0$
⟨*proof*⟩

**lemma** *card-less-Suc2*:
**assumes** $0 \notin M$ **shows** $card\ \{k.\ Suc\ k \in M \wedge k < i\} = card\ \{k \in M.\ k < Suc\ i\}$
⟨*proof*⟩

**lemma** *card-less-Suc*:
**assumes** $0 \in M$
**shows** $Suc\ (card\ \{k.\ Suc\ k \in M \wedge k < i\}) = card\ \{k \in M.\ k < Suc\ i\}$
⟨*proof*⟩

**lemma** *card-le-Suc-Max*: $finite\ S \Longrightarrow card\ S \leq Suc\ (Max\ S)$
⟨*proof*⟩

**lemma** *finite-countable-subset*:
**assumes** $finite\ A$ **and** $A$: $A \subseteq (\bigcup i{::}nat.\ B\ i)$
**obtains** $n$ **where** $A \subseteq (\bigcup i<n.\ B\ i)$
⟨*proof*⟩

**lemma** *finite-countable-equals*:
**assumes** $finite\ A\ A = (\bigcup i{::}nat.\ B\ i)$
**obtains** $n$ **where** $A = (\bigcup i<n.\ B\ i)$
⟨*proof*⟩

## 61.7 Lemmas useful with the summation operator sum

For examples, see Algebra/poly/UnivPoly2.thy

### 61.7.1 Disjoint Unions

Singletons and open intervals

**lemma** *ivl-disj-un-singleton*:
$\{l{::}'a{::}linorder\} \cup \{l{<}..\} = \{l..\}$
$\{..{<}u\} \cup \{u{::}'a{::}linorder\} = \{..u\}$
$(l{::}'a{::}linorder) < u \Longrightarrow \{l\} \cup \{l{<}..{<}u\} = \{l..{<}u\}$
$(l{::}'a{::}linorder) < u \Longrightarrow \{l{<}..{<}u\} \cup \{u\} = \{l{<}..u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{l\} \cup \{l{<}..u\} = \{l..u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{l..{<}u\} \cup \{u\} = \{l..u\}$
$\langle proof \rangle$

One- and two-sided intervals

**lemma** *ivl-disj-un-one*:
$(l{::}'a{::}linorder) < u \Longrightarrow \{..l\} \cup \{l{<}..{<}u\} = \{..{<}u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{..{<}l\} \cup \{l..{<}u\} = \{..{<}u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{..l\} \cup \{l{<}..u\} = \{..u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{..{<}l\} \cup \{l..u\} = \{..u\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{l{<}..u\} \cup \{u{<}..\} = \{l{<}..\}$
$(l{::}'a{::}linorder) < u \Longrightarrow \{l{<}..{<}u\} \cup \{u..\} = \{l{<}..\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{l..u\} \cup \{u{<}..\} = \{l..\}$
$(l{::}'a{::}linorder) \leq u \Longrightarrow \{l..{<}u\} \cup \{u..\} = \{l..\}$
$\langle proof \rangle$

Two- and two-sided intervals

**lemma** *ivl-disj-un-two*:
$[\![(l{::}'a{::}linorder) < m;\ m \leq u]\!] \Longrightarrow \{l{<}..{<}m\} \cup \{m..{<}u\} = \{l{<}..{<}u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m < u]\!] \Longrightarrow \{l{<}..m\} \cup \{m{<}..{<}u\} = \{l{<}..{<}u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m \leq u]\!] \Longrightarrow \{l..{<}m\} \cup \{m..{<}u\} = \{l..{<}u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m < u]\!] \Longrightarrow \{l..m\} \cup \{m{<}..{<}u\} = \{l..{<}u\}$
$[\![(l{::}'a{::}linorder) < m;\ m \leq u]\!] \Longrightarrow \{l{<}..{<}m\} \cup \{m..u\} = \{l{<}..u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m \leq u]\!] \Longrightarrow \{l{<}..m\} \cup \{m{<}..u\} = \{l{<}..u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m \leq u]\!] \Longrightarrow \{l..{<}m\} \cup \{m..u\} = \{l..u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m \leq u]\!] \Longrightarrow \{l..m\} \cup \{m{<}..u\} = \{l..u\}$
$\langle proof \rangle$

**lemma** *ivl-disj-un-two-touch*:
$[\![(l{::}'a{::}linorder) < m;\ m < u]\!] \Longrightarrow \{l{<}..m\} \cup \{m..{<}u\} = \{l{<}..{<}u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m < u]\!] \Longrightarrow \{l..m\} \cup \{m..{<}u\} = \{l..{<}u\}$
$[\![(l{::}'a{::}linorder) < m;\ m \leq u]\!] \Longrightarrow \{l{<}..m\} \cup \{m..u\} = \{l{<}..u\}$
$[\![(l{::}'a{::}linorder) \leq m;\ m \leq u]\!] \Longrightarrow \{l..m\} \cup \{m..u\} = \{l..u\}$
$\langle proof \rangle$

**lemmas** *ivl-disj-un* = *ivl-disj-un-singleton ivl-disj-un-one ivl-disj-un-two ivl-disj-un-two-touch*

### 61.7.2 Disjoint Intersections

One- and two-sided intervals

**lemma** *ivl-disj-int-one*:
$\{..l::'a::order\} \cap \{l<..<u\} = \{\}$
$\{..<l\} \cap \{l..<u\} = \{\}$
$\{..l\} \cap \{l<..u\} = \{\}$
$\{..<l\} \cap \{l..u\} = \{\}$
$\{l<..u\} \cap \{u<..\} = \{\}$
$\{l<..<u\} \cap \{u..\} = \{\}$
$\{l..u\} \cap \{u<..\} = \{\}$
$\{l..<u\} \cap \{u..\} = \{\}$
$\langle proof \rangle$

Two- and two-sided intervals

**lemma** *ivl-disj-int-two*:
$\{l::'a::order<..<m\} \cap \{m..<u\} = \{\}$
$\{l<..m\} \cap \{m<..<u\} = \{\}$
$\{l..<m\} \cap \{m..<u\} = \{\}$
$\{l..m\} \cap \{m<..<u\} = \{\}$
$\{l<..<m\} \cap \{m..u\} = \{\}$
$\{l<..m\} \cap \{m<..u\} = \{\}$
$\{l..<m\} \cap \{m..u\} = \{\}$
$\{l..m\} \cap \{m<..u\} = \{\}$
$\langle proof \rangle$

**lemmas** *ivl-disj-int* = *ivl-disj-int-one ivl-disj-int-two*

### 61.7.3 Some Differences

**lemma** *ivl-diff* [*simp*]:
$i \le n \Longrightarrow \{i..<m\} - \{i..<n\} = \{n..<(m::'a::linorder)\}$
$\langle proof \rangle$

**lemma** (**in** *linorder*) *lessThan-minus-lessThan* [*simp*]:
$\{..< n\} - \{..< m\} = \{m ..< n\}$
$\langle proof \rangle$

**lemma** (**in** *linorder*) *atLeastAtMost-diff-ends*:
$\{a..b\} - \{a, b\} = \{a<..<b\}$
$\langle proof \rangle$

### 61.7.4 Some Subset Conditions

**lemma** *ivl-subset* [*simp*]: $(\{i..<j\} \subseteq \{m..<n\}) = (j \le i \lor m \le i \land j \le (n::'a::linorder))$
$\langle proof \rangle$

## 61.8 Generic big monoid operation over intervals

**context** *semiring-char-0*
**begin**

**lemma** *inj-on-of-nat* [*simp*]:

*inj-on of-nat N*
⟨*proof*⟩

**lemma** *bij-betw-of-nat* [*simp*]:
*bij-betw of-nat N A* $\longleftrightarrow$ *of-nat ' N = A*
⟨*proof*⟩

**lemma** *Nats-infinite*: *infinite* ($\mathbb{N}$ :: *'a set*)
⟨*proof*⟩

**end**

**context** *comm-monoid-set*
**begin**

**lemma** *atLeastLessThan-reindex*:
*F g* {*h m..<h n*} = *F* (*g* ∘ *h*) {*m..<n*}
**if** *bij-betw h* {*m..<n*} {*h m..<h n*} **for** *m n* ::*nat*
⟨*proof*⟩

**lemma** *atLeastAtMost-reindex*:
*F g* {*h m..h n*} = *F* (*g* ∘ *h*) {*m..n*}
**if** *bij-betw h* {*m..n*} {*h m..h n*} **for** *m n* ::*nat*
⟨*proof*⟩

**lemma** *atLeastLessThan-shift-bounds*:
*F g* {*m* + *k..<n* + *k*} = *F* (*g* ∘ *plus k*) {*m..<n*}
**for** *m n k* :: *nat*
⟨*proof*⟩

**lemma** *atLeastAtMost-shift-bounds*:
*F g* {*m* + *k..n* + *k*} = *F* (*g* ∘ *plus k*) {*m..n*}
**for** *m n k* :: *nat*
⟨*proof*⟩

**lemma** *atLeast-Suc-lessThan-Suc-shift*:
*F g* {*Suc m..<Suc n*} = *F* (*g* ∘ *Suc*) {*m..<n*}
⟨*proof*⟩

**lemma** *atLeast-Suc-atMost-Suc-shift*:
*F g* {*Suc m..Suc n*} = *F* (*g* ∘ *Suc*) {*m..n*}
⟨*proof*⟩

**lemma** *atLeast-atMost-pred-shift*:
*F* (*g* ∘ (λ*n*. *n* − *Suc 0*)) {*Suc m..Suc n*} = *F g* {*m..n*}
⟨*proof*⟩

**lemma** *atLeast-lessThan-pred-shift*:
*F* (*g* ∘ (λ*n*. *n* − *Suc 0*)) {*Suc m..<Suc n*} = *F g* {*m..<n*}

⟨proof⟩

**lemma** *atLeast-int-lessThan-int-shift*:
$F\ g\ \{int\ m..{<}int\ n\} = F\ (g \circ int)\ \{m..{<}n\}$
⟨proof⟩

**lemma** *atLeast-int-atMost-int-shift*:
$F\ g\ \{int\ m..int\ n\} = F\ (g \circ int)\ \{m..n\}$
⟨proof⟩

**lemma** *atLeast0-lessThan-Suc*:
$F\ g\ \{0..{<}Suc\ n\} = F\ g\ \{0..{<}n\} * g\ n$
⟨proof⟩

**lemma** *atLeast0-atMost-Suc*:
$F\ g\ \{0..Suc\ n\} = F\ g\ \{0..n\} * g\ (Suc\ n)$
⟨proof⟩

**lemma** *atLeast0-lessThan-Suc-shift*:
$F\ g\ \{0..{<}Suc\ n\} = g\ 0 * F\ (g \circ Suc)\ \{0..{<}n\}$
⟨proof⟩

**lemma** *atLeast0-atMost-Suc-shift*:
$F\ g\ \{0..Suc\ n\} = g\ 0 * F\ (g \circ Suc)\ \{0..n\}$
⟨proof⟩

**lemma** *atLeast-Suc-lessThan*:
$F\ g\ \{m..{<}n\} = g\ m * F\ g\ \{Suc\ m..{<}n\}$ **if** $m < n$
⟨proof⟩

**lemma** *atLeast-Suc-atMost*:
$F\ g\ \{m..n\} = g\ m * F\ g\ \{Suc\ m..n\}$ **if** $m \le n$
⟨proof⟩

**lemma** *ivl-cong*:
$a = c \Longrightarrow b = d \Longrightarrow (\bigwedge x.\ c \le x \Longrightarrow x < d \Longrightarrow g\ x = h\ x)$
$\Longrightarrow F\ g\ \{a..{<}b\} = F\ h\ \{c..{<}d\}$
⟨proof⟩

**lemma** *atLeastLessThan-shift-0*:
**fixes** $m\ n\ p :: nat$
**shows** $F\ g\ \{m..{<}n\} = F\ (g \circ plus\ m)\ \{0..{<}n - m\}$
⟨proof⟩

**lemma** *atLeastAtMost-shift-0*:
**fixes** $m\ n\ p :: nat$
**assumes** $m \le n$
**shows** $F\ g\ \{m..n\} = F\ (g \circ plus\ m)\ \{0..n - m\}$
⟨proof⟩

**lemma** *atLeastLessThan-concat*:
**fixes** $m$ $n$ $p$ :: *nat*
**shows** $m \leq n \Longrightarrow n \leq p \Longrightarrow F\ g\ \{m..<n\} * F\ g\ \{n..<p\} = F\ g\ \{m..<p\}$
⟨*proof*⟩

**lemma** *atLeastLessThan-rev*:
$F\ g\ \{n..<m\} = F\ (\lambda i.\ g\ (m + n - Suc\ i))\ \{n..<m\}$
⟨*proof*⟩

**lemma** *atLeastAtMost-rev*:
**fixes** $n$ $m$ :: *nat*
**shows** $F\ g\ \{n..m\} = F\ (\lambda i.\ g\ (m + n - i))\ \{n..m\}$
⟨*proof*⟩

**lemma** *atLeastLessThan-rev-at-least-Suc-atMost*:
$F\ g\ \{n..<m\} = F\ (\lambda i.\ g\ (m + n - i))\ \{Suc\ n..m\}$
⟨*proof*⟩

**end**

## 61.9 Summation indexed over intervals

**syntax** (*ASCII*)
*-from-to-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹notation=‹binder SUM››SUM - = -..-./ -)› [0,0,0,10] 10)
*-from-upto-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹notation=‹binder SUM››SUM - = -..<-./ -)› [0,0,0,10] 10)
*-upt-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹notation=‹binder SUM››SUM -<-./ -)› [0,0,10] 10)
*-upto-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹notation=‹binder SUM››SUM -<=-./ -)› [0,0,10] 10)

**syntax** (*latex-sum* **output**)
*-from-to-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b*
(‹(3$\sum_{- = -}^{-}$ -)› [0,0,0,10] 10)
*-from-upto-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b*
(‹(3$\sum_{- = -}^{<-}$ -)› [0,0,0,10] 10)
*-upt-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b*
(‹(3$\sum_{- < -}$ -)› [0,0,10] 10)
*-upto-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b*
(‹(3$\sum_{- \leq -}$ -)› [0,0,10] 10)

**syntax**
*-from-to-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder $\sum$››$\sum$ - = -..-./ -)› [0,0,0,10] 10)
*-from-upto-sum* :: *idt* ⇒ *'a* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder $\sum$››$\sum$ - = -..<-./ -)› [0,0,0,10] 10)
*-upt-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder $\sum$››$\sum$ -<-./ -)›

*[0,0,10] 10)*
*-upto-sum* :: *idt* ⇒ *'a* ⇒ *'b* ⇒ *'b* (‹(‹*indent=3 notation=*‹*binder* $\sum$ ››$\sum$ *-≤-./ -)*› *[0,0,10] 10)*

**syntax-consts**
*-from-to-sum -from-upto-sum -upt-sum -upto-sum* == *sum*

**translations**
$\sum x{=}a..b.\ t == CONST\ sum\ (\lambda x.\ t)\ \{a..b\}$
$\sum x{=}a..{<}b.\ t == CONST\ sum\ (\lambda x.\ t)\ \{a..{<}b\}$
$\sum i{\le}n.\ t == CONST\ sum\ (\lambda i.\ t)\ \{..n\}$
$\sum i{<}n.\ t == CONST\ sum\ (\lambda i.\ t)\ \{..{<}n\}$

The above introduces some pretty alternative syntaxes for summation over intervals:

| Old | New | LaTeX |
|---|---|---|
| $\sum x{\in}\{a..b\}.\ e$ | $\sum x = a..b.\ e$ | $\sum_{x\ =\ a}^{b}\ e$ |
| $\sum x{\in}\{a..{<}b\}.\ e$ | $\sum x = a..{<}b.\ e$ | $\sum_{x\ =\ a}^{<b}\ e$ |
| $\sum x{\in}\{..b\}.\ e$ | $\sum x{\le}b.\ e$ | $\sum_{x\ \le\ b}\ e$ |
| $\sum x{\in}\{..{<}b\}.\ e$ | $\sum x{<}b.\ e$ | $\sum_{x\ <\ b}\ e$ |

The left column shows the term before introduction of the new syntax, the middle column shows the new (default) syntax, and the right column shows a special syntax. The latter is only meaningful for latex output and has to be activated explicitly by setting the print mode to *latex-sum* (e.g. via *mode = latex-sum* in antiquotations). It is not the default LaTeX output because it only works well with italic-style formulae, not tt-style.

Note that for uniformity on *nat* it is better to use $\sum x = 0..{<}n.\ e$ rather than $\sum x{<}n.\ e$: *sum* may not provide all lemmas available for $\{m..{<}n\}$ also in the special form for $\{..{<}n\}$.

This congruence rule should be used for sums over intervals as the standard theorem *sum.cong* does not work well with the simplifier who adds the unsimplified premise $x \in B$ to the context.

**context** *comm-monoid-set*
**begin**

**lemma** *zero-middle*:
**assumes** $1 \le p$ $k \le p$
**shows** $F\ (\lambda j.\ if\ j < k\ then\ g\ j\ else\ if\ j = k\ then\ \mathbf{1}\ else\ h\ (j - Suc\ 0))\ \{..p\}$
$= F\ (\lambda j.\ if\ j < k\ then\ g\ j\ else\ h\ j)\ \{..p - Suc\ 0\}$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *atMost-Suc* [*simp*]:
$F\ g\ \{..Suc\ n\} = F\ g\ \{..n\} * g\ (Suc\ n)$
⟨*proof*⟩

**lemma** *lessThan-Suc* [*simp*]:
$F\ g\ \{..<Suc\ n\} = F\ g\ \{..<n\} * g\ n$
⟨*proof*⟩

**lemma** *cl-ivl-Suc* [*simp*]:
$F\ g\ \{m..Suc\ n\} = (\mathit{if}\ Suc\ n < m\ \mathit{then}\ \mathbf{1}\ \mathit{else}\ F\ g\ \{m..n\} * g(Suc\ n))$
⟨*proof*⟩

**lemma** *op-ivl-Suc* [*simp*]:
$F\ g\ \{m..<Suc\ n\} = (\mathit{if}\ n < m\ \mathit{then}\ \mathbf{1}\ \mathit{else}\ F\ g\ \{m..<n\} * g(n))$
⟨*proof*⟩

**lemma** *head*:
**fixes** $n :: nat$
**assumes** *mn*: $m \leq n$
**shows** $F\ g\ \{m..n\} = g\ m * F\ g\ \{m<..n\}$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

**lemma** *last-plus*:
**fixes** $n::nat$ **shows** $m \leq n \Longrightarrow F\ g\ \{m..n\} = g\ n * F\ g\ \{m..<n\}$
⟨*proof*⟩

**lemma** *head-if*:
**fixes** $n :: nat$
**shows** $F\ g\ \{m..n\} = (\mathit{if}\ n < m\ \mathit{then}\ \mathbf{1}\ \mathit{else}\ \ F\ g\ \{m..<n\} * g(n))$
⟨*proof*⟩

**lemma** *ub-add-nat*:
**assumes** $(m::nat) \leq n + 1$
**shows** $F\ g\ \{m..n + p\} = F\ g\ \{m..n\} * F\ g\ \{n + 1..n + p\}$
⟨*proof*⟩

**lemma** *nat-group*:
**fixes** $k::nat$ **shows** $F\ (\lambda m.\ F\ g\ \{m * k\ ..<\ m*k + k\})\ \{..<n\} = F\ g\ \{..<\ n * k\}$
⟨*proof*⟩

**lemma** *triangle-reindex*:
**fixes** $n :: nat$
**shows** $F\ (\lambda(i,j).\ g\ i\ j)\ \{(i,j).\ i+j < n\} = F\ (\lambda k.\ F\ (\lambda i.\ g\ i\ (k - i))\ \{..k\})$ $\{..<n\}$
⟨*proof*⟩

**lemma** *triangle-reindex-eq*:
**fixes** $n :: nat$
**shows** $F\ (\lambda(i,j).\ g\ i\ j)\ \{(i,j).\ i+j \leq n\} = F\ (\lambda k.\ F\ (\lambda i.\ g\ i\ (k - i))\ \{..k\})\ \{..n\}$
⟨*proof*⟩

**lemma** *nat-diff-reindex*: $F\ (\lambda i.\ g\ (n - Suc\ i))\ \{..<n\} = F\ g\ \{..<n\}$

⟨proof⟩

**lemma** *shift-bounds-nat-ivl*:
$F\ g\ \{m+k..<n+k\} = F\ (\lambda i.\ g(i + k))\{m..<n::nat\}$
⟨proof⟩

**lemma** *shift-bounds-cl-nat-ivl*:
$F\ g\ \{m+k..n+k\} = F\ (\lambda i.\ g(i + k))\{m..n::nat\}$
⟨proof⟩

**corollary** *shift-bounds-cl-Suc-ivl*:
$F\ g\ \{Suc\ m..Suc\ n\} = F\ (\lambda i.\ g(Suc\ i))\{m..n\}$
⟨proof⟩

**corollary** *Suc-reindex-ivl*: $m \leq n \Longrightarrow F\ g\ \{m..n\} * g\ (Suc\ n) = g\ m * F\ (\lambda i.\ g\ (Suc\ i))\ \{m..n\}$
⟨proof⟩

**corollary** *shift-bounds-Suc-ivl*:
$F\ g\ \{Suc\ m..<Suc\ n\} = F\ (\lambda i.\ g(Suc\ i))\{m..<n\}$
⟨proof⟩

**lemma** *atMost-Suc-shift*:
**shows** $F\ g\ \{..Suc\ n\} = g\ 0 * F\ (\lambda i.\ g\ (Suc\ i))\ \{..n\}$
⟨proof⟩

**lemma** *lessThan-Suc-shift*:
$F\ g\ \{..<Suc\ n\} = g\ 0 * F\ (\lambda i.\ g\ (Suc\ i))\ \{..<n\}$
⟨proof⟩

**lemma** *atMost-shift*:
$F\ g\ \{..n\} = g\ 0 * F\ (\lambda i.\ g\ (Suc\ i))\ \{..<n\}$
⟨proof⟩

**lemma** *nested-swap*:
$F\ (\lambda i.\ F\ (\lambda j.\ a\ i\ j)\ \{0..<i\})\ \{0..n\} = F\ (\lambda j.\ F\ (\lambda i.\ a\ i\ j)\ \{Suc\ j..n\})\ \{0..<n\}$
⟨proof⟩

**lemma** *nested-swap'*:
$F\ (\lambda i.\ F\ (\lambda j.\ a\ i\ j)\ \{..<i\})\ \{..n\} = F\ (\lambda j.\ F\ (\lambda i.\ a\ i\ j)\ \{Suc\ j..n\})\ \{..<n\}$
⟨proof⟩

**lemma** *atLeast1-atMost-eq*:
$F\ g\ \{Suc\ 0..n\} = F\ (\lambda k.\ g\ (Suc\ k))\ \{..<n\}$
⟨proof⟩

**lemma** *atLeastLessThan-Suc*: $a \leq b \Longrightarrow F\ g\ \{a..<Suc\ b\} = F\ g\ \{a..<b\} * g\ b$
⟨proof⟩

**lemma** *nat-ivl-Suc'*:
**assumes** $m \leq Suc\ n$
**shows** $F\ g\ \{m..Suc\ n\} = g\ (Suc\ n) * F\ g\ \{m..n\}$
⟨*proof*⟩

**lemma** *in-pairs*: $F\ g\ \{2{*}m..Suc(2{*}n)\} = F\ (\lambda i.\ g(2{*}i) * g(Suc(2{*}i)))\ \{m..n\}$
⟨*proof*⟩

**lemma** *in-pairs-0*: $F\ g\ \{..Suc(2{*}n)\} = F\ (\lambda i.\ g(2{*}i) * g(Suc(2{*}i)))\ \{..n\}$
⟨*proof*⟩

**end**

**lemma** *card-sum-le-nat-sum*: $\sum\ \{0..<card\ S\} \leq \sum\ S$
⟨*proof*⟩

**lemma** *sum-natinterval-diff*:
**fixes** $f{::}\ nat \Rightarrow ('a{::}ab\text{-}group\text{-}add)$
**shows** $sum\ (\lambda k.\ f\ k - f(k + 1))\ \{(m{::}nat)\ ..\ n\} =$
$(if\ m \leq n\ then\ f\ m - f(n + 1)\ else\ 0)$
⟨*proof*⟩

**lemma** *sum-diff-nat-ivl*:
**fixes** $f :: nat \Rightarrow 'a{::}ab\text{-}group\text{-}add$
**shows** $[\![\ m \leq n;\ n \leq p\ ]\!] \Longrightarrow sum\ f\ \{m..<p\} - sum\ f\ \{m..<n\} = sum\ f\ \{n..<p\}$
⟨*proof*⟩

**lemma** *sum-diff-distrib*: $\forall x.\ Q\ x \leq P\ x \Longrightarrow (\sum x{<}n.\ P\ x) - (\sum x{<}n.\ Q\ x) = (\sum x{<}n.\ P\ x - Q\ x :: nat)$
⟨*proof*⟩

### 61.9.1 Shifting bounds

**context** *comm-monoid-add*
**begin**

**context**
**fixes** $f :: nat \Rightarrow 'a$
**assumes** $f\ 0 = 0$
**begin**

**lemma** *sum-shift-lb-Suc0-0-upt*:
$sum\ f\ \{Suc\ 0..<k\} = sum\ f\ \{0..<k\}$
⟨*proof*⟩

**lemma** *sum-shift-lb-Suc0-0*: $sum\ f\ \{Suc\ 0..k\} = sum\ f\ \{0..k\}$
⟨*proof*⟩

**end**

**end**

**lemma** *sum-Suc-diff*:
**fixes** $f :: nat \Rightarrow 'a::ab\text{-}group\text{-}add$
**assumes** $m \leq Suc\ n$
**shows** $(\sum i = m..n.\ f(Suc\ i) - f\ i) = f\ (Suc\ n) - f\ m$
⟨*proof*⟩

**lemma** *sum-Suc-diff'*:
**fixes** $f :: nat \Rightarrow 'a::ab\text{-}group\text{-}add$
**assumes** $m \leq n$
**shows** $(\sum i = m..<n.\ f\ (Suc\ i) - f\ i) = f\ n - f\ m$
⟨*proof*⟩

**lemma** *sum-diff-split*:
**fixes** $f:: nat \Rightarrow 'a::ab\text{-}group\text{-}add$
**assumes** $m \leq n$
**shows** $(\sum i{\leq}n.\ f\ i) - (\sum i{<}m.\ f\ i) = (\sum i{\leq}n - m.\ f(n - i))$
⟨*proof*⟩

**lemma** *prod-divide-nat-ivl*:
**fixes** $f :: nat \Rightarrow 'a::idom\text{-}divide$
**shows** $[\![\ m \leq n;\ n \leq p;\ prod\ f\ \{m..<n\} \neq 0]\!] \Longrightarrow prod\ f\ \{m..<p\}\ div\ prod\ f\ \{m..<n\} = prod\ f\ \{n..<p\}$
⟨*proof*⟩

**lemma** *prod-divide-split*:
**fixes** $f:: nat \Rightarrow 'a::idom\text{-}divide$
**assumes** $m \leq n$ $prod\ f\ \{..<m\} \neq 0$
**shows** $(prod\ f\ \{..n\})\ div\ (prod\ f\ \{..<m\}) = prod\ (\lambda i.\ f(n - i))\ \{..n - m\}$
⟨*proof*⟩

### 61.9.2 Telescoping sums

**lemma** *sum-telescope*:
**fixes** $f::nat \Rightarrow 'a::ab\text{-}group\text{-}add$
**shows** $sum\ (\lambda i.\ f\ i - f\ (Suc\ i))\ \{..\ i\} = f\ 0 - f\ (Suc\ i)$
⟨*proof*⟩

**lemma** *sum-telescope''*:
**assumes** $m \leq n$
**shows** $(\sum k{\in}\{Suc\ m..n\}.\ f\ k - f\ (k - 1)) = f\ n - (f\ m :: 'a :: ab\text{-}group\text{-}add)$
⟨*proof*⟩

**lemma** *sum-lessThan-telescope*:
$(\sum n{<}m.\ f\ (Suc\ n) - f\ n :: 'a :: ab\text{-}group\text{-}add) = f\ m - f\ 0$
⟨*proof*⟩

**lemma** *sum-lessThan-telescope'*:
$(\sum n<m.\ f\ n - f\ (Suc\ n) :: 'a :: ab\text{-}group\text{-}add) = f\ 0 - f\ m$
⟨*proof*⟩

### 61.9.3 The formula for geometric sums

**lemma** *sum-power2*: $(\sum i=0..<k.\ (2::nat)\hat{}i) = 2\hat{}k-1$
⟨*proof*⟩

**lemma** *geometric-sum*:
**assumes** $x \neq 1$
**shows** $(\sum i<n.\ x \hat{}\ i) = (x \hat{}\ n - 1) / (x - 1::'a::field)$
⟨*proof*⟩

**lemma** *geometric-sum-less*:
**assumes** $0 < x\ x < 1$ *finite S*
**shows** $(\sum i\in S.\ x \hat{}\ i) < 1 / (1 - x::'a::linordered\text{-}field)$
⟨*proof*⟩

**lemma** *diff-power-eq-sum*:
**fixes** $y :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows**
$x \hat{}(Suc\ n) - y \hat{}(Suc\ n) =$
$(x - y) * (\sum p<Suc\ n.\ (x \hat{}\ p) * y \hat{}(n - p))$
⟨*proof*⟩

**corollary** *power-diff-sumr2*: — *COMPLEX-POLYFUN* in HOL Light
**fixes** $x :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows** $x\hat{}n - y\hat{}n = (x - y) * (\sum i<n.\ y\hat{}(n - Suc\ i) * x\hat{}i)$
⟨*proof*⟩

**lemma** *power-diff-1-eq*:
**fixes** $x :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows** $x\hat{}n - 1 = (x - 1) * (\sum i<n.\ (x\hat{}i))$
⟨*proof*⟩

**lemma** *one-diff-power-eq'*:
**fixes** $x :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows** $1 - x\hat{}n = (1 - x) * (\sum i<n.\ x\hat{}(n - Suc\ i))$
⟨*proof*⟩

**lemma** *one-diff-power-eq*:
**fixes** $x :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows** $1 - x\hat{}n = (1 - x) * (\sum i<n.\ x\hat{}i)$
⟨*proof*⟩

**lemma** *sum-gp-basic*:
**fixes** $x :: 'a::\{comm\text{-}ring,monoid\text{-}mult\}$
**shows** $(1 - x) * (\sum i\leq n.\ x\hat{}i) = 1 - x\hat{}Suc\ n$

⟨*proof*⟩

**lemma** *sum-power-shift*:
**fixes** $x$ :: $'a$::{*comm-ring*,*monoid-mult*}
**assumes** $m \leq n$
**shows** $(\sum i{=}m..n.\ x\hat{}i) = x\hat{}m * (\sum i{\leq}n{-}m.\ x\hat{}i)$
⟨*proof*⟩

**lemma** *sum-gp-multiplied*:
**fixes** $x$ :: $'a$::{*comm-ring*,*monoid-mult*}
**assumes** $m \leq n$
**shows** $(1 - x) * (\sum i{=}m..n.\ x\hat{}i) = x\hat{}m - x\hat{}Suc\ n$
⟨*proof*⟩

**lemma** *sum-gp*:
**fixes** $x$ :: $'a$::{*comm-ring*,*division-ring*}
**shows** $(\sum i{=}m..n.\ x\hat{}i) =$
$(\textit{if}\ n < m\ \textit{then}\ 0$
$\textit{else if}\ x = 1\ \textit{then of-nat}((n + 1) - m)$
$\textit{else}\ (x\hat{}m - x\hat{}Suc\ n) / (1 - x))$
⟨*proof*⟩

### 61.9.4 Geometric progressions

**lemma** *sum-gp0*:
**fixes** $x$ :: $'a$::{*comm-ring*,*division-ring*}
**shows** $(\sum i{\leq}n.\ x\hat{}i) = (\textit{if}\ x = 1\ \textit{then of-nat}(n + 1)\ \textit{else}\ (1 - x\hat{}Suc\ n) / (1 - x))$
⟨*proof*⟩

**lemma** *sum-power-add*:
**fixes** $x$ :: $'a$::{*comm-ring*,*monoid-mult*}
**shows** $(\sum i{\in}I.\ x\hat{}(m+i)) = x\hat{}m * (\sum i{\in}I.\ x\hat{}i)$
⟨*proof*⟩

**lemma** *sum-gp-offset*:
**fixes** $x$ :: $'a$::{*comm-ring*,*division-ring*}
**shows** $(\sum i{=}m..m{+}n.\ x\hat{}i) =$
$(\textit{if}\ x = 1\ \textit{then of-nat}\ n + 1\ \textit{else}\ x\hat{}m * (1 - x\hat{}Suc\ n) / (1 - x))$
⟨*proof*⟩

**lemma** *sum-gp-strict*:
**fixes** $x$ :: $'a$::{*comm-ring*,*division-ring*}
**shows** $(\sum i{<}n.\ x\hat{}i) = (\textit{if}\ x = 1\ \textit{then of-nat}\ n\ \textit{else}\ (1 - x\hat{}n) / (1 - x))$
⟨*proof*⟩

### 61.9.5 The formulae for arithmetic sums

**context** *comm-semiring-1*
**begin**

**lemma** *double-gauss-sum*:
$2 * (\sum i = 0..n.\ of\text{-}nat\ i) = of\text{-}nat\ n * (of\text{-}nat\ n + 1)$
⟨*proof*⟩

**lemma** *double-gauss-sum-from-Suc-0*:
$2 * (\sum i = Suc\ 0..n.\ of\text{-}nat\ i) = of\text{-}nat\ n * (of\text{-}nat\ n + 1)$
⟨*proof*⟩

**lemma** *double-arith-series*:
$2 * (\sum i = 0..n.\ a + of\text{-}nat\ i * d) = (of\text{-}nat\ n + 1) * (2 * a + of\text{-}nat\ n * d)$
⟨*proof*⟩

**end**

**context** *linordered-euclidean-semiring*
**begin**

**lemma** *gauss-sum*:
$(\sum i = 0..n.\ of\text{-}nat\ i) = of\text{-}nat\ n * (of\text{-}nat\ n + 1)\ div\ 2$
⟨*proof*⟩

**lemma** *gauss-sum-from-Suc-0*:
$(\sum i = Suc\ 0..n.\ of\text{-}nat\ i) = of\text{-}nat\ n * (of\text{-}nat\ n + 1)\ div\ 2$
⟨*proof*⟩

**lemma** *arith-series*:
$(\sum i = 0..n.\ a + of\text{-}nat\ i * d) = (of\text{-}nat\ n + 1) * (2 * a + of\text{-}nat\ n * d)\ div\ 2$
⟨*proof*⟩

**end**

**lemma** *gauss-sum-nat*:
$\sum \{0..n\} = (n * Suc\ n)\ div\ 2$
⟨*proof*⟩

**lemma** *arith-series-nat*:
$(\sum i = 0..n.\ a + i * d) = Suc\ n * (2 * a + n * d)\ div\ 2$
⟨*proof*⟩

**lemma** *Sum-Icc-int*:
$\sum \{m..n\} = (n * (n + 1) - m * (m - 1))\ div\ 2$
**if** $m \leq n$ **for** $m\ n :: int$
⟨*proof*⟩

**lemma** *Sum-Icc-nat*:
$\sum \{m..n\} = (n * (n + 1) - m * (m - 1))\ div\ 2$ **for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *Sum-Ico-nat*:
$\sum\{m..<n\} = (n * (n - 1) - m * (m - 1))$ *div 2* **for** *m n* :: *nat*
⟨*proof*⟩

### 61.9.6 Division remainder

**lemma** *range-mod*:
**fixes** *n* :: *nat*
**assumes** $n > 0$
**shows** *range* ($\lambda m.$ *m mod n*) $= \{0..<n\}$ (**is** *?A = ?B*)
⟨*proof*⟩

## 61.10 Products indexed over intervals

**syntax** (*ASCII*)
-from-to-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹notation=‹binder PROD››PROD - = -..-./ -)› [0,0,0,10] 10)
-from-upto-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹notation=‹binder PROD››PROD - = -..<-./ -)› [0,0,0,10] 10)
-upt-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹notation=‹binder PROD››PROD -<-./ -)› [0,0,10] 10)
-upto-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹notation=‹binder PROD››PROD -<=-./ -)› [0,0,10] 10)

**syntax** (*latex-prod* **output**)
-from-to-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b
(‹(3$\prod_{\text{-} = \text{-}}^{\text{-}}$ -)› [0,0,0,10] 10)
-from-upto-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b
(‹(3$\prod_{\text{-} = \text{-}}^{<\text{-}}$ -)› [0,0,0,10] 10)
-upt-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b
(‹(3$\prod_{\text{-} < \text{-}}$ -)› [0,0,10] 10)
-upto-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b
(‹(3$\prod_{\text{-} \leq \text{-}}$ -)› [0,0,10] 10)

**syntax**
-from-to-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ∏››∏ - = -..-./ -)› [0,0,0,10] 10)
-from-upto-prod :: idt ⇒ 'a ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ∏››∏ - = -..<-./ -)› [0,0,0,10] 10)
-upt-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ∏››∏ -<-./ -)› [0,0,10] 10)
-upto-prod :: idt ⇒ 'a ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ∏››∏ -≤-./ -)› [0,0,10] 10)

**syntax-consts**
-from-to-prod -from-upto-prod -upt-prod -upto-prod ⇌ prod

**translations**
∏x=a..b. t ⇌ CONST prod (λx. t) {a..b}
∏x=a..<b. t ⇌ CONST prod (λx. t) {a..<b}

$\prod i{\le}n.\ t \rightleftharpoons CONST\ prod\ (\lambda i.\ t)\ \{..n\}$
$\prod i{<}n.\ t \rightleftharpoons CONST\ prod\ (\lambda i.\ t)\ \{..{<}n\}$

**lemma** *prod-int-plus-eq*: $prod\ int\ \{i..i+j\} = \prod\{int\ i..int\ (i+j)\}$
⟨*proof*⟩

**lemma** *prod-int-eq*: $prod\ int\ \{i..j\} = \prod\{int\ i..int\ j\}$
⟨*proof*⟩

### 61.10.1 Telescoping products

**lemma** *prod-telescope*:
**fixes** $f::nat \Rightarrow 'a::field$
**assumes** $\bigwedge i.\ i{\le}n \Longrightarrow f\ (Suc\ i) \neq 0$
**shows** $(\prod i{\le}n.\ f\ i\ /\ f\ (Suc\ i)) = f\ 0\ /\ f\ (Suc\ n)$
⟨*proof*⟩

**lemma** *prod-telescope''*:
**fixes** $f::nat \Rightarrow 'a::field$
**assumes** $m \le n$
**assumes** $\bigwedge i.\ i \in \{m..n\} \Longrightarrow f\ i \neq 0$
**shows** $(\prod i = Suc\ m..n.\ f\ i\ /\ f\ (i - 1)) = f\ n\ /\ f\ m$
⟨*proof*⟩

**lemma** *prod-lessThan-telescope*:
**fixes** $f::nat \Rightarrow 'a::field$
**assumes** $\bigwedge i.\ i{\le}n \Longrightarrow f\ i \neq 0$
**shows** $(\prod i{<}n.\ f\ (Suc\ i)\ /\ f\ i) = f\ n\ /\ f\ 0$
⟨*proof*⟩

**lemma** *prod-lessThan-telescope'*:
**fixes** $f::nat \Rightarrow 'a::field$
**assumes** $\bigwedge i.\ i{\le}n \Longrightarrow f\ i \neq 0$
**shows** $(\prod i{<}n.\ f\ i\ /\ f\ (Suc\ i)) = f\ 0\ /\ f\ n$
⟨*proof*⟩

## 61.11 Efficient folding over intervals

**function** *fold-atLeastAtMost-nat* **where**
[*simp del*]: *fold-atLeastAtMost-nat f a* $(b::nat)$ *acc* =
(*if* $a > b$ *then acc else fold-atLeastAtMost-nat f* $(a+1)$ *b* (*f a acc*))
⟨*proof*⟩
**termination** ⟨*proof*⟩

**lemma** *fold-atLeastAtMost-nat*:
**assumes** *comp-fun-commute f*
**shows** *fold-atLeastAtMost-nat f a b acc = Finite-Set.fold f acc* $\{a..b\}$
⟨*proof*⟩

**lemma** *sum-atLeastAtMost-code*:

$sum\ f\ \{a..b\} = fold\text{-}atLeastAtMost\text{-}nat\ (\lambda a\ acc.\ f\ a + acc)\ a\ b\ 0$
⟨proof⟩

**lemma** *prod-atLeastAtMost-code*:
$prod\ f\ \{a..b\} = fold\text{-}atLeastAtMost\text{-}nat\ (\lambda a\ acc.\ f\ a * acc)\ a\ b\ 1$
⟨proof⟩

**lemma** *pairs-le-eq-Sigma*: $\{(i, j).\ i + j \le m\} = Sigma\ (atMost\ m)\ (\lambda r.\ atMost\ (m - r))$
**for** $m :: nat$
⟨proof⟩

**lemma** *sum-up-index-split*: $(\sum k \le m + n.\ f\ k) = (\sum k \le m.\ f\ k) + (\sum k = Suc\ m..m + n.\ f\ k)$
⟨proof⟩

**lemma** *Sigma-interval-disjoint*: $(SIGMA\ i{:}A.\ \{..v\ i\}) \cap (SIGMA\ i{:}A.\{v\ i{<}..w\}) = \{\}$
**for** $w :: 'a{::}order$
⟨proof⟩

**lemma** *product-atMost-eq-Un*: $A \times \{..m\} = (SIGMA\ i{:}A.\{..m - i\}) \cup (SIGMA\ i{:}A.\{m - i{<}..m\})$
**for** $m :: nat$
⟨proof⟩

**lemma** *polynomial-product*:
**fixes** $x :: 'a{::}idom$
**assumes** $m$: $\bigwedge i.\ i > m \Longrightarrow a\ i = 0$
**and** $n$: $\bigwedge j.\ j > n \Longrightarrow b\ j = 0$
**shows** $(\sum i \le m.\ (a\ i) * x \mathbin{\hat{}} i) * (\sum j \le n.\ (b\ j) * x \mathbin{\hat{}} j) =$
$(\sum r \le m + n.\ (\sum k \le r.\ (a\ k) * (b\ (r - k))) * x \mathbin{\hat{}} r)$
⟨proof⟩

**end**

# 62 Decision Procedure for Presburger Arithmetic

**theory** *Presburger*
**imports** *Groebner-Basis Set-Interval*
**begin**

⟨ML⟩

## 62.1 The $-\infty$ and $+\infty$ Properties

**lemma** *minf*:

$[\![\exists (z ::'a::linorder).\forall x<z.\ P\ x = P'\ x;\ \exists z.\forall x<z.\ Q\ x = Q'\ x]\!]$
$\Longrightarrow \exists z.\forall x<z.\ (P\ x \wedge Q\ x) = (P'\ x \wedge Q'\ x)$
$[\![\exists (z ::'a::linorder).\forall x<z.\ P\ x = P'\ x;\ \exists z.\forall x<z.\ Q\ x = Q'\ x]\!]$
$\Longrightarrow \exists z.\forall x<z.\ (P\ x \vee Q\ x) = (P'\ x \vee Q'\ x)$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x = t) = False$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x \neq t) = True$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x < t) = True$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x \leq t) = True$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x > t) = False$
$\exists (z ::'a::\{linorder\}).\forall x<z.(x \geq t) = False$
$\exists z.\forall (x::'b::\{linorder,plus,Rings.dvd\})<z.\ (d\ dvd\ x + s) = (d\ dvd\ x + s)$
$\exists z.\forall (x::'b::\{linorder,plus,Rings.dvd\})<z.\ (\neg\ d\ dvd\ x + s) = (\neg\ d\ dvd\ x + s)$
$\exists z.\forall x<z.\ F = F$
$\langle proof \rangle$

**lemma** *pinf*:
$[\![\exists (z ::'a::linorder).\forall x>z.\ P\ x = P'\ x;\ \exists z.\forall x>z.\ Q\ x = Q'\ x]\!]$
$\Longrightarrow \exists z.\forall x>z.\ (P\ x \wedge Q\ x) = (P'\ x \wedge Q'\ x)$
$[\![\exists (z ::'a::linorder).\forall x>z.\ P\ x = P'\ x;\ \exists z.\forall x>z.\ Q\ x = Q'\ x]\!]$
$\Longrightarrow \exists z.\forall x>z.\ (P\ x \vee Q\ x) = (P'\ x \vee Q'\ x)$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x = t) = False$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x \neq t) = True$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x < t) = False$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x \leq t) = False$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x > t) = True$
$\exists (z ::'a::\{linorder\}).\forall x>z.(x \geq t) = True$
$\exists z.\forall (x::'b::\{linorder,plus,Rings.dvd\})>z.\ (d\ dvd\ x + s) = (d\ dvd\ x + s)$
$\exists z.\forall (x::'b::\{linorder,plus,Rings.dvd\})>z.\ (\neg\ d\ dvd\ x + s) = (\neg\ d\ dvd\ x + s)$
$\exists z.\forall x>z.\ F = F$
$\langle proof \rangle$

**lemma** *inf-period*:
$[\![\forall x\ k.\ P\ x = P\ (x - k*D);\ \forall x\ k.\ Q\ x = Q\ (x - k*D)]\!]$
$\Longrightarrow \forall x\ k.\ (P\ x \wedge Q\ x) = (P\ (x - k*D) \wedge Q\ (x - k*D))$
$[\![\forall x\ k.\ P\ x = P\ (x - k*D);\ \forall x\ k.\ Q\ x = Q\ (x - k*D)]\!]$
$\Longrightarrow \forall x\ k.\ (P\ x \vee Q\ x) = (P\ (x - k*D) \vee Q\ (x - k*D))$
$(d::'a::\{comm\text{-}ring,Rings.dvd\})\ dvd\ D \Longrightarrow \forall x\ k.\ (d\ dvd\ x + t) = (d\ dvd\ (x - k*D) + t)$
$(d::'a::\{comm\text{-}ring,Rings.dvd\})\ dvd\ D \Longrightarrow \forall x\ k.\ (\neg d\ dvd\ x + t) = (\neg d\ dvd\ (x - k*D) + t)$
$\forall x\ k.\ F = F$
$\langle proof \rangle$

## 62.2 The A and B sets

**lemma** *bset*:
$[\![\forall x.(\forall j \in \{1\ ..\ D\}.\ \forall b\in B.\ x \neq b + j) \longrightarrow P\ x \longrightarrow P(x - D)\ ;$
$\forall x.(\forall j\in\{1\ ..\ D\}.\ \forall b\in B.\ x \neq b + j) \longrightarrow Q\ x \longrightarrow Q(x - D)]\!] \Longrightarrow$
$\forall x.(\forall j\in\{1\ ..\ D\}.\ \forall b\in B.\ x \neq b + j) \longrightarrow (P\ x \wedge Q\ x) \longrightarrow (P(x - D) \wedge Q\ (x -$

$D))$
$[\![\forall x.(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow P\ x \longrightarrow P(x - D)$ ;
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow Q\ x \longrightarrow Q(x - D)]\!] \Longrightarrow$
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (P\ x \vee Q\ x) \longrightarrow (P(x - D) \vee Q\ (x - D))$
$[\![D>0;\ t - 1 \in B]\!] \Longrightarrow (\forall x.(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x = t) \longrightarrow (x - D = t))$
$[\![D>0\ ;\ t \in B]\!] \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \neq t) \longrightarrow (x - D \neq t))$
$D>0 \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x < t) \longrightarrow (x - D < t))$
$D>0 \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \leq t) \longrightarrow (x - D \leq t))$
$[\![D>0\ ;\ t \in B]\!] \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x > t) \longrightarrow (x - D > t))$
$[\![D>0\ ;\ t - 1 \in B]\!] \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (x \geq t) \longrightarrow (x - D \geq t))$
$d\ dvd\ D \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (d\ dvd\ x+t) \longrightarrow (d\ dvd\ (x - D) + t))$
$d\ dvd\ D \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow (\neg d\ dvd\ x+t) \longrightarrow (\neg d\ dvd\ (x - D) + t))$
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in B. x \neq b + j) \longrightarrow F \longrightarrow F$
$\langle proof \rangle$

**lemma** *aset*:
$[\![\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow P\ x \longrightarrow P(x + D)$ ;
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow Q\ x \longrightarrow Q(x + D)]\!] \Longrightarrow$
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (P\ x \wedge Q\ x) \longrightarrow (P(x + D) \wedge Q\ (x + D))$
$[\![\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow P\ x \longrightarrow P(x + D)$ ;
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow Q\ x \longrightarrow Q(x + D)]\!] \Longrightarrow$
$\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (P\ x \vee Q\ x) \longrightarrow (P(x + D) \vee Q\ (x + D))$
$[\![D>0;\ t + 1 \in A]\!] \Longrightarrow (\forall x.(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x = t) \longrightarrow (x + D = t))$
$[\![D>0\ ;\ t \in A]\!] \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \neq t) \longrightarrow (x + D \neq t))$
$[\![D>0;\ t \in A]\!] \Longrightarrow (\forall (x::int).\ (\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x < t) \longrightarrow (x + D < t))$
$[\![D>0;\ t + 1 \in A]\!] \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \leq t) \longrightarrow (x + D \leq t))$
$D>0 \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x > t) \longrightarrow (x + D > t))$
$D>0 \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (x \geq t) \longrightarrow (x + D \geq t))$
$d\ dvd\ D \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (d\ dvd\ x+t) \longrightarrow (d\ dvd\ (x + D) + t))$
$d\ dvd\ D \Longrightarrow (\forall (x::int).(\forall j \in \{1 .. D\}. \forall b \in A. x \neq b - j) \longrightarrow (\neg d\ dvd\ x+t) \longrightarrow (\neg d\ dvd\ (x + D) + t))$

$\forall x.(\forall j \in \{1 \,..\, D\}.\ \forall b \in A.\ x \neq b - j) \longrightarrow F \longrightarrow F$
⟨*proof*⟩

## 62.3 Cooper's Theorem $-\infty$ and $+\infty$ Version

### 62.3.1 First some trivial facts about periodic sets or predicates

**lemma** *periodic-finite-ex*:
**assumes** *dpos*: $(0::int) < d$ **and** *modd*: $\forall x\ k.\ P\ x = P(x - k*d)$
**shows** $(\exists x.\ P\ x) = (\exists j \in \{1..d\}.\ P\ j)$
(**is** *?LHS* = *?RHS*)
⟨*proof*⟩

### 62.3.2 The $-\infty$ Version

**lemma** *decr-lemma*: $0 < (d::int) \Longrightarrow x - (|x - z| + 1) * d < z$
⟨*proof*⟩

**lemma** *incr-lemma*: $0 < (d::int) \Longrightarrow z < x + (|x - z| + 1) * d$
⟨*proof*⟩

**lemma** *decr-mult-lemma*:
**assumes** *dpos*: $(0::int) < d$ **and** *minus*: $\forall x.\ P\ x \longrightarrow P(x - d)$ **and** *knneg*: $0 <= k$
**shows** $\forall x.\ P\ x \longrightarrow P(x - k*d)$
⟨*proof*⟩

**lemma** *minusinfinity*:
**assumes** *dpos*: $0 < d$ **and**
*P1eqP1*: $\forall x\ k.\ P1\ x = P1(x - k*d)$ **and** *ePeqP1*: $\exists z::int.\ \forall x.\ x < z \longrightarrow (P\ x = P1\ x)$
**shows** $(\exists x.\ P1\ x) \longrightarrow (\exists x.\ P\ x)$
⟨*proof*⟩

**lemma** *cpmi*:
**assumes** *dp*: $0 < D$ **and** *p1*: $\exists z.\ \forall\ x< z.\ P\ x = P'\ x$
**and** *nb*: $\forall x.(\forall\ j \in \{1..D\}.\ \forall(b::int) \in B.\ x \neq b+j) \longrightarrow P\ (x) \longrightarrow P\ (x - D)$
**and** *pd*: $\forall\ x\ k.\ P'\ x = P'\ (x-k*D)$
**shows** $(\exists x.\ P\ x) = ((\exists j \in \{1..D\}\ .\ P'\ j) \vee (\exists j \in \{1..D\}.\ \exists\ b \in B.\ P\ (b+j)))$
(**is** *?L* = (*?R1* ∨ *?R2*))
⟨*proof*⟩

### 62.3.3 The $+\infty$ Version

**lemma** *plusinfinity*:
**assumes** *dpos*: $(0::int) < d$ **and**
*P1eqP1*: $\forall x\ k.\ P'\ x = P'(x - k*d)$ **and** *ePeqP1*: $\exists\ z.\ \forall\ x>z.\ P\ x = P'\ x$
**shows** $(\exists\ x.\ P'\ x) \longrightarrow (\exists\ x.\ P\ x)$
⟨*proof*⟩

**lemma** *incr-mult-lemma*:
**assumes** *dpos*: $(0::int) < d$ **and** *plus*: $\forall x::int.\ P\ x \longrightarrow P(x + d)$ **and** *knneg*: $0 <= k$
**shows** $\forall x.\ P\ x \longrightarrow P(x + k*d)$
⟨*proof*⟩

**lemma** *cppi*:
**assumes** *dp*: $0 < D$ **and** *p1*:$\exists z.\ \forall\ x> z.\ P\ x = P'\ x$
**and** *nb*:$\forall x.(\forall\ j\in \{1..D\}.\ \forall (b::int) \in A.\ x \neq b - j) \longrightarrow P\ (x) \longrightarrow P\ (x + D)$
**and** *pd*: $\forall\ x\ k.\ P'\ x= P'\ (x-k*D)$
**shows** $(\exists x.\ P\ x) = ((\exists j \in \{1..D\}\ .\ P'\ j) \vee (\exists\ j \in \{1..D\}.\ \exists\ b\in A.\ P\ (b - j)))$ (**is** $?L = (?R1 \vee ?R2)$)
⟨*proof*⟩

**lemma** *simp-from-to*: $\{i..j::int\} = (if\ j < i\ then\ \{\}\ else\ insert\ i\ \{i+1..j\})$
⟨*proof*⟩

**theorem** *unity-coeff-ex*: $(\exists (x::'a::\{semiring\text{-}0,Rings.dvd\}).\ P\ (l * x)) \equiv (\exists x.\ l\ dvd\ (x + 0) \wedge P\ x)$
⟨*proof*⟩

**lemma** *zdvd-mono*:
**fixes** $k\ m\ t :: int$
**assumes** $k \neq 0$
**shows** $m\ dvd\ t \equiv k * m\ dvd\ k * t$
⟨*proof*⟩

**lemma** *uminus-dvd-conv*:
**fixes** $d\ t :: int$
**shows** $d\ dvd\ t \equiv -\ d\ dvd\ t$ **and** $d\ dvd\ t \equiv d\ dvd - t$
⟨*proof*⟩

Theorems for transforming predicates on nat to predicates on *int*

**lemma** *zdiff-int-split*: $P\ (int\ (x - y)) =$
$((y \leq x \longrightarrow P\ (int\ x - int\ y)) \wedge (x < y \longrightarrow P\ 0))$
⟨*proof*⟩

Specific instances of congruence rules, to prevent simplifier from looping.

**theorem** *imp-le-cong*:
$[\![x = x';\ 0 \leq x' \Longrightarrow P = P']\!] \Longrightarrow (0 \leq (x::int) \longrightarrow P) = (0 \leq x' \longrightarrow P')$
⟨*proof*⟩

**theorem** *conj-le-cong*:
$[\![x = x';\ 0 \leq x' \Longrightarrow P = P']\!] \Longrightarrow (0 \leq (x::int) \wedge P) = (0 \leq x' \wedge P')$
⟨*proof*⟩

⟨*ML*⟩

**declare** *mod-eq-0-iff-dvd* [*presburger*]
**declare** *mod-by-Suc-0* [*presburger*]
**declare** *mod-0* [*presburger*]
**declare** *mod-by-1* [*presburger*]
**declare** *mod-self* [*presburger*]
**declare** *div-by-0* [*presburger*]
**declare** *mod-by-0* [*presburger*]
**declare** *mod-div-trivial* [*presburger*]
**declare** *mult-div-mod-eq* [*presburger*]
**declare** *div-mult-mod-eq* [*presburger*]
**declare** *mod-mult-self1* [*presburger*]
**declare** *mod-mult-self2* [*presburger*]
**declare** *mod2-Suc-Suc* [*presburger*]
**declare** *not-mod-2-eq-0-eq-1* [*presburger*]
**declare** *nat-zero-less-power-iff* [*presburger*]

**lemma** [*presburger*, *algebra*]: $m\ mod\ 2 = (1::nat) \longleftrightarrow \neg\ 2\ dvd\ m$ ⟨*proof*⟩
**lemma** [*presburger*, *algebra*]: $m\ mod\ 2 = Suc\ 0 \longleftrightarrow \neg\ 2\ dvd\ m$ ⟨*proof*⟩
**lemma** [*presburger*, *algebra*]: $m\ mod\ (Suc\ (Suc\ 0)) = (1::nat) \longleftrightarrow \neg\ 2\ dvd\ m$ ⟨*proof*⟩
**lemma** [*presburger*, *algebra*]: $m\ mod\ (Suc\ (Suc\ 0)) = Suc\ 0 \longleftrightarrow \neg\ 2\ dvd\ m$ ⟨*proof*⟩
**lemma** [*presburger*, *algebra*]: $m\ mod\ 2 = (1::int) \longleftrightarrow \neg\ 2\ dvd\ m$ ⟨*proof*⟩

**context** *semiring-parity*
**begin**

**declare** *even-mult-iff* [*presburger*]

**declare** *even-power* [*presburger*]

**lemma** [*presburger*]:
$even\ (a + b) \longleftrightarrow even\ a \wedge even\ b \vee odd\ a \wedge odd\ b$
⟨*proof*⟩

**end**

**context** *ring-parity*
**begin**

**declare** *even-minus* [*presburger*]

**end**

**context** *linordered-idom*
**begin**

**declare** *zero-le-power-eq* [*presburger*]

**declare** *zero-less-power-eq* [*presburger*]

**declare** *power-less-zero-eq* [*presburger*]

**declare** *power-le-zero-eq* [*presburger*]

**end**

**declare** *even-Suc* [*presburger*]

**lemma** [*presburger*]:
$Suc\ n\ div\ Suc\ (Suc\ 0) = n\ div\ Suc\ (Suc\ 0) \longleftrightarrow even\ n$
⟨*proof*⟩

**declare** *even-diff-nat* [*presburger*]

**lemma** [*presburger*]:
**fixes** $k :: int$
**shows** $(k + 1)\ div\ 2 = k\ div\ 2 \longleftrightarrow even\ k$
⟨*proof*⟩

**lemma** [*presburger*]:
**fixes** $k :: int$
**shows** $(k + 1)\ div\ 2 = k\ div\ 2 + 1 \longleftrightarrow odd\ k$
⟨*proof*⟩

**lemma** [*presburger*]:
$even\ n \longleftrightarrow even\ (int\ n)$
⟨*proof*⟩

## 62.4 Nice facts about division by $4::'a$

**lemma** *even-even-mod-4-iff*:
$even\ (n::nat) \longleftrightarrow even\ (n\ mod\ 4)$
⟨*proof*⟩

**lemma** *odd-mod-4-div-2*:
$n\ mod\ 4 = (3::nat) \Longrightarrow odd\ ((n - Suc\ 0)\ div\ 2)$
⟨*proof*⟩

**lemma** *even-mod-4-div-2*:
$n\ mod\ 4 = Suc\ 0 \Longrightarrow even\ ((n - Suc\ 0)\ div\ 2)$
⟨*proof*⟩

**end**

**theory** *Try0-HOL*
**imports** *Try0 Presburger*
**begin**

⟨*ML*⟩

**declare** [[*try0-schedule* =
*satx metis*
*order presburger linarith algebra argo*
*simp auto blast fast fastforce force meson*
]]

**end**

# 63 Bindings to Satisfiability Modulo Theories (SMT) solvers based on SMT-LIB 2

**theory** *SMT*
**imports** *Numeral-Simprocs*
**keywords** *smt-status* :: *diag*
**begin**

## 63.1 A skolemization tactic and proof method

**lemma** *ex-iff-push*: $(\exists y.\ P \longleftrightarrow Q\ y) \longleftrightarrow (P \longrightarrow (\exists y.\ Q\ y)) \wedge ((\forall y.\ Q\ y) \longrightarrow P)$
⟨*proof*⟩

⟨*ML*⟩

**hide-fact** (**open**) *ex-iff-push*

## 63.2 Triggers for quantifier instantiation

Some SMT solvers support patterns as a quantifier instantiation heuristics. Patterns may either be positive terms (tagged by "pat") triggering quantifier instantiations – when the solver finds a term matching a positive pattern, it instantiates the corresponding quantifier accordingly – or negative terms (tagged by "nopat") inhibiting quantifier instantiations. A list of patterns of the same kind is called a multipattern, and all patterns in a multipattern are considered conjunctively for quantifier instantiation. A list of multipatterns is called a trigger, and their multipatterns act disjunctively during quantifier instantiation. Each multipattern should mention at least all quantified variables of the preceding quantifier block.

**typedecl** *'a symb-list*

**consts**
*Symb-Nil* :: *'a symb-list*
*Symb-Cons* :: $'a \Rightarrow$ *'a symb-list* $\Rightarrow$ *'a symb-list*

**typedecl** *pattern*

**consts**
*pat* :: $'a \Rightarrow$ *pattern*
*nopat* :: $'a \Rightarrow$ *pattern*

**definition** *trigger* :: *pattern symb-list symb-list* $\Rightarrow$ *bool* $\Rightarrow$ *bool* **where**
*trigger* - $P = P$

## 63.3 Higher-order encoding

Application is made explicit for constants occurring with varying numbers of arguments. This is achieved by the introduction of the following constant.

**definition** *fun-app* :: $'a \Rightarrow 'a$ **where** *fun-app* $f = f$

Some solvers support a theory of arrays which can be used to encode higher-order functions. The following set of lemmas specifies the properties of such (extensional) arrays.

**lemmas** *array-rules* = *ext fun-upd-apply fun-upd-same fun-upd-other fun-upd-upd fun-app-def*

## 63.4 Normalization

**lemma** *case-bool-if*[*abs-def*]: *case-bool* $x\ y\ P$ = (*if* $P$ *then* $x$ *else* $y$)
⟨*proof*⟩

**lemmas** *Ex1-def-raw* = *Ex1-def*[*abs-def*]
**lemmas** *Ball-def-raw* = *Ball-def*[*abs-def*]
**lemmas** *Bex-def-raw* = *Bex-def*[*abs-def*]
**lemmas** *abs-if-raw* = *abs-if*[*abs-def*]
**lemmas** *min-def-raw* = *min-def*[*abs-def*]
**lemmas** *max-def-raw* = *max-def*[*abs-def*]

**lemma** *nat-zero-as-int*:
*0* = *nat 0*
⟨*proof*⟩

**lemma** *nat-one-as-int*:
*1* = *nat 1*
⟨*proof*⟩

**lemma** *nat-numeral-as-int*: *numeral* = ($\lambda i$. *nat* (*numeral* $i$)) ⟨*proof*⟩
**lemma** *nat-less-as-int*: $(<) = (\lambda a\ b.\ int\ a < int\ b)$ ⟨*proof*⟩
**lemma** *nat-leq-as-int*: $(\leq) = (\lambda a\ b.\ int\ a \leq int\ b)$ ⟨*proof*⟩
**lemma** *Suc-as-int*: $Suc = (\lambda a.\ nat\ (int\ a + 1))$ ⟨*proof*⟩
**lemma** *nat-plus-as-int*: $(+) = (\lambda a\ b.\ nat\ (int\ a + int\ b))$ ⟨*proof*⟩
**lemma** *nat-minus-as-int*: $(-) = (\lambda a\ b.\ nat\ (int\ a - int\ b))$ ⟨*proof*⟩
**lemma** *nat-times-as-int*: $(*) = (\lambda a\ b.\ nat\ (int\ a * int\ b))$ ⟨*proof*⟩
**lemma** *nat-div-as-int*: $(div) = (\lambda a\ b.\ nat\ (int\ a\ div\ int\ b))$ ⟨*proof*⟩

**lemma** *nat-mod-as-int*: $(mod) = (\lambda a\ b.\ nat\ (int\ a\ mod\ int\ b))$ ⟨*proof*⟩

**lemma** *int-Suc*: $int\ (Suc\ n) = int\ n + 1$ ⟨*proof*⟩
**lemma** *int-plus*: $int\ (n + m) = int\ n + int\ m$ ⟨*proof*⟩
**lemma** *int-minus*: $int\ (n - m) = int\ (nat\ (int\ n - int\ m))$ ⟨*proof*⟩

**lemma** *nat-int-comparison*:
  **fixes** $a\ b :: nat$
  **shows** $(a = b) = (int\ a = int\ b)$
    **and** $(a < b) = (int\ a < int\ b)$
    **and** $(a \leq b) = (int\ a \leq int\ b)$
  ⟨*proof*⟩

**lemma** *int-ops*:
  **fixes** $a\ b :: nat$
  **shows** $int\ 0 = 0$
    **and** $int\ 1 = 1$
    **and** $int\ (numeral\ n) = numeral\ n$
    **and** $int\ (Suc\ a) = int\ a + 1$
    **and** $int\ (a + b) = int\ a + int\ b$
    **and** $int\ (a - b) = (if\ int\ a < int\ b\ then\ 0\ else\ int\ a - int\ b)$
    **and** $int\ (a * b) = int\ a * int\ b$
    **and** $int\ (a\ div\ b) = int\ a\ div\ int\ b$
    **and** $int\ (a\ mod\ b) = int\ a\ mod\ int\ b$
  ⟨*proof*⟩

**lemma** *int-if*:
  **fixes** $a\ b :: nat$
  **shows** $int\ (if\ P\ then\ a\ else\ b) = (if\ P\ then\ int\ a\ else\ int\ b)$
  ⟨*proof*⟩

## 63.5 Integer division and modulo for Z3

The following Z3-inspired definitions are overspecified for the case where $l = 0$. This Schönheitsfehler is corrected in the *div-as-z3div* and *mod-as-z3mod* theorems.

**definition** *z3div* :: $int \Rightarrow int \Rightarrow int$ **where**
  $z3div\ k\ l = (if\ l \geq 0\ then\ k\ div\ l\ else\ -\ (k\ div\ -\ l))$

**definition** *z3mod* :: $int \Rightarrow int \Rightarrow int$ **where**
  $z3mod\ k\ l = k\ mod\ (if\ l \geq 0\ then\ l\ else\ -\ l)$

**lemma** *div-as-z3div*:
  $\forall k\ l.\ k\ div\ l = (if\ l = 0\ then\ 0\ else\ if\ l > 0\ then\ z3div\ k\ l\ else\ z3div\ (-\ k)\ (-\ l))$
  ⟨*proof*⟩

**lemma** *mod-as-z3mod*:
  $\forall k\ l.\ k\ mod\ l = (if\ l = 0\ then\ k\ else\ if\ l > 0\ then\ z3mod\ k\ l\ else\ -\ z3mod\ (-\ k)\ (-\ l))$

⟨*proof*⟩

## 63.6 Extra theorems for veriT reconstruction

**lemma** *verit-sko-forall*: ‹$(\forall x.\ P\ x) \longleftrightarrow P\ (SOME\ x.\ \neg P\ x)$›
⟨*proof*⟩

**lemma** *verit-sko-forall'*: ‹$P\ (SOME\ x.\ \neg P\ x) = A \Longrightarrow (\forall x.\ P\ x) = A$›
⟨*proof*⟩

**lemma** *verit-sko-forall''*: ‹$B = A \Longrightarrow (SOME\ x.\ P\ x) = A \equiv (SOME\ x.\ P\ x) = B$›
⟨*proof*⟩

**lemma** *verit-sko-forall-indirect*: ‹$x = (SOME\ x.\ \neg P\ x) \Longrightarrow (\forall x.\ P\ x) \longleftrightarrow P\ x$›
⟨*proof*⟩

**lemma** *verit-sko-forall-indirect2*:
‹$x = (SOME\ x.\ \neg P\ x) \Longrightarrow (\bigwedge x :: 'a.\ (P\ x = P'\ x)) \Longrightarrow (\forall x.\ P'\ x) \longleftrightarrow P\ x$›
⟨*proof*⟩

**lemma** *verit-sko-ex*: ‹$(\exists x.\ P\ x) \longleftrightarrow P\ (SOME\ x.\ P\ x)$›
⟨*proof*⟩

**lemma** *verit-sko-ex'*: ‹$P\ (SOME\ x.\ P\ x) = A \Longrightarrow (\exists x.\ P\ x) = A$›
⟨*proof*⟩

**lemma** *verit-sko-ex-indirect*: ‹$x = (SOME\ x.\ P\ x) \Longrightarrow (\exists x.\ P\ x) \longleftrightarrow P\ x$›
⟨*proof*⟩

**lemma** *verit-sko-ex-indirect2*: ‹$x = (SOME\ x.\ P\ x) \Longrightarrow (\bigwedge x.\ P\ x = P'\ x) \Longrightarrow (\exists x.\ P'\ x) \longleftrightarrow P\ x$›
⟨*proof*⟩

**lemma** *verit-Pure-trans*:
‹$P \equiv Q \Longrightarrow Q \Longrightarrow P$›
⟨*proof*⟩

**lemma** *verit-if-cong*:
**assumes** ‹$b \equiv c$›
**and** ‹$c \Longrightarrow x \equiv u$›
**and** ‹$\neg\ c \Longrightarrow y \equiv v$›
**shows** ‹$(\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y) \equiv (\mathit{if}\ c\ \mathit{then}\ u\ \mathit{else}\ v)$›
⟨*proof*⟩

**lemma** *verit-if-weak-cong'*:
‹$b \equiv c \Longrightarrow (\mathit{if}\ b\ \mathit{then}\ x\ \mathit{else}\ y) \equiv (\mathit{if}\ c\ \mathit{then}\ x\ \mathit{else}\ y)$›
⟨*proof*⟩

**lemma** *verit-or-neg*:
‹$(A \Longrightarrow B) \Longrightarrow B \vee \neg A$›
‹$(\neg A \Longrightarrow B) \Longrightarrow B \vee A$›
⟨*proof*⟩

**lemma** *verit-subst-bool*: ‹$P \Longrightarrow f\ True \Longrightarrow f\ P$›
⟨*proof*⟩

**lemma** *verit-and-pos*:
‹$(a \Longrightarrow \neg(b \wedge c) \vee A) \Longrightarrow \neg(a \wedge b \wedge c) \vee A$›
‹$(a \Longrightarrow b \Longrightarrow A) \Longrightarrow \neg(a \wedge b) \vee A$›
⟨*proof*⟩

**lemma** *verit-farkas*:
‹$(a \Longrightarrow A) \Longrightarrow \neg a \vee A$›
‹$(\neg a \Longrightarrow A) \Longrightarrow a \vee A$›
⟨*proof*⟩

**lemma** *verit-or-pos*:
‹$A \wedge A' \Longrightarrow (c \wedge A) \vee (\neg c \wedge A')$›
‹$A \wedge A' \Longrightarrow (\neg c \wedge A) \vee (c \wedge A')$›
⟨*proof*⟩

**lemma** *verit-la-generic*:
‹$(a{::}int) \leq x \vee a = x \vee a \geq x$›
⟨*proof*⟩

**lemma** *verit-bfun-elim*:
‹$(if\ b\ then\ P\ True\ else\ P\ False) = P\ b$›
‹$(\forall b.\ P'\ b) = (P'\ False \wedge P'\ True)$›
‹$(\exists b.\ P'\ b) = (P'\ False \vee P'\ True)$›
⟨*proof*⟩

**lemma** *verit-eq-true-simplify*:
‹$(P = True) \equiv P$›
⟨*proof*⟩

**lemma** *verit-and-neg*:
‹$(a \Longrightarrow \neg b \vee A) \Longrightarrow \neg(a \wedge b) \vee A$›
‹$(a \Longrightarrow A) \Longrightarrow \neg a \vee A$›
‹$(\neg a \Longrightarrow A) \Longrightarrow a \vee A$›
⟨*proof*⟩

**lemma** *verit-forall-inst*:
‹$A \longleftrightarrow B \Longrightarrow \neg A \vee B$›
‹$\neg A \longleftrightarrow B \Longrightarrow A \vee B$›
‹$A \longleftrightarrow B \Longrightarrow \neg B \vee A$›
‹$A \longleftrightarrow \neg B \Longrightarrow B \vee A$›

$\langle A \longrightarrow B \Longrightarrow \neg A \vee B\rangle$
$\langle \neg A \longrightarrow B \Longrightarrow A \vee B\rangle$
$\langle proof\rangle$

**lemma** *verit-eq-transitive*:
$\langle A = B \Longrightarrow B = C \Longrightarrow A = C\rangle$
$\langle A = B \Longrightarrow C = B \Longrightarrow A = C\rangle$
$\langle B = A \Longrightarrow B = C \Longrightarrow A = C\rangle$
$\langle B = A \Longrightarrow C = B \Longrightarrow A = C\rangle$
$\langle proof\rangle$

**lemma** *verit-bool-simplify*:
$\langle \neg(P \longrightarrow Q) \longleftrightarrow P \wedge \neg Q\rangle$
$\langle \neg(P \vee Q) \longleftrightarrow \neg P \wedge \neg Q\rangle$
$\langle \neg(P \wedge Q) \longleftrightarrow \neg P \vee \neg Q\rangle$
$\langle (P \longrightarrow (Q \longrightarrow R)) \longleftrightarrow ((P \wedge Q) \longrightarrow R)\rangle$
$\langle ((P \longrightarrow Q) \longrightarrow Q) \longleftrightarrow P \vee Q\rangle$
$\langle (Q \longleftrightarrow (P \vee Q)) \longleftrightarrow (P \longrightarrow Q)\rangle$ — This rule was inverted
$\langle P \wedge (P \longrightarrow Q) \longleftrightarrow P \wedge Q\rangle$
$\langle (P \longrightarrow Q) \wedge P \longleftrightarrow P \wedge Q\rangle$

$\langle proof\rangle$

We need the last equation for $\neg\ (\forall\, a\ b.\ \neg\ P\ a\ b)$

**lemma** *verit-connective-def*: — the definition of XOR is missing as the operator is not generated by Isabelle
$\langle (A = B) \longleftrightarrow ((A \longrightarrow B) \wedge (B \longrightarrow A))\rangle$
$\langle (If\ A\ B\ C) = ((A \longrightarrow B) \wedge (\neg A \longrightarrow C))\rangle$
$\langle (\exists\, x.\ P\ x) \longleftrightarrow \neg(\forall\, x.\ \neg P\ x)\rangle$
$\langle \neg(\exists\, x.\ P\ x) \longleftrightarrow (\forall\, x.\ \neg P\ x)\rangle$
$\langle proof\rangle$

**lemma** *verit-ite-simplify*:
$\langle (If\ True\ B\ C) = B\rangle$
$\langle (If\ False\ B\ C) = C\rangle$
$\langle (If\ A'\ B\ B) = B\rangle$
$\langle (If\ (\neg A')\ B\ C) = (If\ A'\ C\ B)\rangle$
$\langle (If\ c\ (If\ c\ A\ B)\ C) = (If\ c\ A\ C)\rangle$
$\langle (If\ c\ C\ (If\ c\ A\ B)) = (If\ c\ C\ B)\rangle$
$\langle (If\ A'\ True\ False) = A'\rangle$
$\langle (If\ A'\ False\ True) \longleftrightarrow \neg A'\rangle$
$\langle (If\ A'\ True\ B') \longleftrightarrow A' \vee B'\rangle$
$\langle (If\ A'\ B'\ False) \longleftrightarrow A' \wedge B'\rangle$
$\langle (If\ A'\ False\ B') \longleftrightarrow \neg A' \wedge B'\rangle$
$\langle (If\ A'\ B'\ True) \longleftrightarrow \neg A' \vee B'\rangle$
$\langle x \wedge True \longleftrightarrow x\rangle$
$\langle x \vee False \longleftrightarrow x\rangle$
**for** $B\ C :: 'a$ **and** $A'\ B'\ C' :: bool$
$\langle proof\rangle$

**lemma** *verit-and-simplify1*:
‹$True \wedge b \longleftrightarrow b$› ‹$b \wedge True \longleftrightarrow b$›
‹$False \wedge b \longleftrightarrow False$› ‹$b \wedge False \longleftrightarrow False$›
‹$(c \wedge \neg c) \longleftrightarrow False$› ‹$(\neg c \wedge c) \longleftrightarrow False$›
‹$\neg\neg a = a$›
⟨*proof*⟩

**lemmas** *verit-and-simplify* = *conj-ac de-Morgan-conj disj-not1*

**lemma** *verit-or-simplify-1*:
‹$False \vee b \longleftrightarrow b$› ‹$b \vee False \longleftrightarrow b$›
‹$b \vee \neg b$›
‹$\neg b \vee b$›
⟨*proof*⟩

**lemmas** *verit-or-simplify* = *disj-ac*

**lemma** *verit-not-simplify*:
‹$\neg \neg b \longleftrightarrow b$› ‹$\neg True \longleftrightarrow False$› ‹$\neg False \longleftrightarrow True$›
⟨*proof*⟩

**lemma** *verit-implies-simplify*:
‹$(\neg a \longrightarrow \neg b) \longleftrightarrow (b \longrightarrow a)$›
‹$(False \longrightarrow a) \longleftrightarrow True$›
‹$(a \longrightarrow True) \longleftrightarrow True$›
‹$(True \longrightarrow a) \longleftrightarrow a$›
‹$(a \longrightarrow False) \longleftrightarrow \neg a$›
‹$(a \longrightarrow a) \longleftrightarrow True$›
‹$(\neg a \longrightarrow a) \longleftrightarrow a$›
‹$(a \longrightarrow \neg a) \longleftrightarrow \neg a$›
‹$((a \longrightarrow b) \longrightarrow b) \longleftrightarrow a \vee b$›
⟨*proof*⟩

**lemma** *verit-equiv-simplify*:
‹$((\neg a) = (\neg b)) \longleftrightarrow (a = b)$›
‹$(a = a) \longleftrightarrow True$›
‹$(a = (\neg a)) \longleftrightarrow False$›
‹$((\neg a) = a) \longleftrightarrow False$›
‹$(True = a) \longleftrightarrow a$›
‹$(a = True) \longleftrightarrow a$›
‹$(False = a) \longleftrightarrow \neg a$›
‹$(a = False) \longleftrightarrow \neg a$›
‹$\neg\neg a \longleftrightarrow a$›
‹$(\neg\ False) = True$›
**for** $a\ b$ :: *bool*
⟨*proof*⟩

**lemmas** *verit-eq-simplify* =
*semiring-char-0-class.eq-numeral-simps eq-refl zero-neq-one num.simps*
*neg-equal-zero equal-neg-zero one-neq-zero neg-equal-iff-equal*

**lemma** *verit-minus-simplify*:
‹$(a :: 'a :: cancel\text{-}comm\text{-}monoid\text{-}add) - a = 0$›
‹$(a :: 'a :: cancel\text{-}comm\text{-}monoid\text{-}add) - 0 = a$›
‹$0 - (b :: 'b :: \{group\text{-}add\}) = -b$›
‹$- (- (b :: 'b :: group\text{-}add)) = b$›
⟨*proof*⟩

**lemma** *verit-sum-simplify*:
‹$(a :: 'a :: cancel\text{-}comm\text{-}monoid\text{-}add) + 0 = a$›
⟨*proof*⟩

**lemmas** *verit-prod-simplify* =

*mult-1*
*mult-1-right*

**lemma** *verit-comp-simplify1*:
‹$(a :: 'a ::order) < a \longleftrightarrow False$›
‹$a \leq a$›
‹$\neg(b' \leq a') \longleftrightarrow (a' :: 'b :: linorder) < b'$›
⟨*proof*⟩

**lemmas** *verit-comp-simplify* =
*verit-comp-simplify1*
*le-numeral-simps*
*le-num-simps*
*less-numeral-simps*
*less-num-simps*
*zero-less-one*
*zero-le-one*
*less-neg-numeral-simps*

**lemma** *verit-la-disequality*:
‹$(a :: 'a ::linorder) = b \vee \neg a \leq b \vee \neg b \leq a$›
⟨*proof*⟩

**context**
**begin**

For the reconstruction, we need to keep the order of the arguments.

**named-theorems** *smt-arith-multiplication* ‹*Theorems to reconstruct arithmetic theorems.*›

**named-theorems** *smt-arith-combine* ‹*Theorems to reconstruct arithmetic theo-*

*rems.*›

**named-theorems** *smt-arith-simplify* ‹*Theorems to combine theorems in the LA procedure*›

**lemmas** [*smt-arith-simplify*] =
*div-add dvd-numeral-simp divmod-steps less-num-simps le-num-simps if-True if-False divmod-cancel*
*dvd-mult dvd-mult2 less-irrefl prod.case numeral-plus-one divmod-step-def order.refl le-zero-eq*
*le-numeral-simps less-numeral-simps mult.right-neutral simp-thms divides-aux-eq*
*mult-nonneg-nonneg dvd-imp-mod-0 dvd-add zero-less-one mod-mult-self4 numeral-mod-numeral*
*divmod-trivial prod.sel mult.left-neutral div-pos-pos-trivial arith-simps div-add div-mult-self1*
*add-le-cancel-left add-le-same-cancel2 not-one-le-zero le-numeral-simps add-le-same-cancel1*
*zero-neq-one zero-le-one le-num-simps add-Suc mod-div-trivial nat.distinct mult-minus-right*
*add.inverse-inverse distrib-left-numeral mult-num-simps numeral-times-numeral add-num-simps*
*divmod-steps rel-simps if-True if-False numeral-div-numeral divmod-cancel prod.case*
*add-num-simps one-plus-numeral fst-conv arith-simps sub-num-simps dbl-inc-simps*
*dbl-simps mult-1 add-le-cancel-right left-diff-distrib-numeral add-uminus-conv-diff zero-neq-one*
*zero-le-one One-nat-def add-Suc mod-div-trivial nat.distinct of-int-1 numerals numeral-One*
*of-int-numeral add-uminus-conv-diff zle-diff1-eq add-less-same-cancel2 minus-add-distrib*
*add-uminus-conv-diff mult.left-neutral semiring-class.distrib-right*
*add-diff-cancel-left′ add-diff-eq ring-distribs mult-minus-left minus-diff-eq*

**lemma** [*smt-arith-simplify*]:
‹$\neg$ $(a' :: 'a :: linorder) < b' \longleftrightarrow b' \leq a'$›
‹$\neg$ $(a' :: 'a :: linorder) \leq b' \longleftrightarrow b' < a'$›
‹$(c::int)$ *mod Numeral1* $= 0$›
‹$(a::nat)$ *mod Numeral1* $= 0$›
‹$(c::int)$ *div Numeral1* $= c$›
‹$a$ *div Numeral1* $= a$›
‹$(c::int)$ *mod* $1 = 0$›
‹$a$ *mod* $1 = 0$›
‹$(c::int)$ *div* $1 = c$›
‹$a$ *div* $1 = a$›
‹$\neg(a' \neq b') \longleftrightarrow a' = b'$›
⟨*proof*⟩

**lemma** *div-mod-decomp*: $A = (A\ div\ n) * n + (A\ mod\ n)$ **for** $A :: nat$
⟨*proof*⟩

**lemma** *div-less-mono*:
**fixes** $A\ B :: nat$

**assumes** $A < B$ $0 < n$ **and**
*mod*: $A$ *mod* $n = 0B$ *mod* $n = 0$
**shows** $(A$ *div* $n) < (B$ *div* $n)$
⟨*proof*⟩

**lemma** *verit-le-mono-div*:
**fixes** $A$ $B$ :: *nat*
**assumes** $A < B$ $0 < n$
**shows** $(A$ *div* $n) + ($*if* $B$ *mod* $n = 0$ *then* $1$ *else* $0) \leq (B$ *div* $n)$
⟨*proof*⟩

**lemmas** [*smt-arith-multiplication*] =
*verit-le-mono-div*[*THEN mult-le-mono1*, *unfolded add-mult-distrib*]
*div-le-mono*[*THEN mult-le-mono2*, *unfolded add-mult-distrib*]

**lemma** *div-mod-decomp-int*: $A = (A$ *div* $n) * n + (A$ *mod* $n)$ **for** $A$ :: *int*
⟨*proof*⟩

**lemma** *zdiv-mono-strict*:
**fixes** $A$ $B$ :: *int*
**assumes** $A < B$ $0 < n$ **and**
*mod*: $A$ *mod* $n = 0B$ *mod* $n = 0$
**shows** $(A$ *div* $n) < (B$ *div* $n)$
⟨*proof*⟩

**lemma** *verit-le-mono-div-int*:
‹$A$ *div* $n + ($*if* $B$ *mod* $n = 0$ *then* $1$ *else* $0) \leq B$ *div* $n$›
**if** ‹$A < B$› ‹$0 < n$›
**for** $A$ $B$ $n$ :: *int*
⟨*proof*⟩

**lemma** *verit-less-mono-div-int2*:
**fixes** $A$ $B$ :: *int*
**assumes** $A \leq B$ $0 < -n$
**shows** $(A$ *div* $n) \geq (B$ *div* $n)$
⟨*proof*⟩

**lemmas** [*smt-arith-multiplication*] =
*verit-le-mono-div-int*[*THEN mult-left-mono*, *unfolded int-distrib*]
*zdiv-mono1*[*THEN mult-left-mono*, *unfolded int-distrib*]

**lemmas** [*smt-arith-multiplication*] =
*arg-cong*[*of - -* ‹$\lambda a$ :: *nat*. $a$ *div* $n * p$› **for** $n$ $p$ :: *nat*, *THEN sym*]
*arg-cong*[*of - -* ‹$\lambda a$ :: *int*. $a$ *div* $n * p$› **for** $n$ $p$ :: *int*, *THEN sym*]

**lemma** [*smt-arith-combine*]:
$a < b \Longrightarrow c < d \Longrightarrow a + c + 2 \leq b + d$
$a < b \Longrightarrow c \leq d \Longrightarrow a + c + 1 \leq b + d$
$a \leq b \Longrightarrow c < d \Longrightarrow a + c + 1 \leq b + d$ **for** $a$ $b$ $c$ :: *int*

⟨*proof*⟩

**lemma** [*smt-arith-combine*]:
$a < b \Longrightarrow c < d \Longrightarrow a + c + 2 \leq b + d$
$a < b \Longrightarrow c \leq d \Longrightarrow a + c + 1 \leq b + d$
$a \leq b \Longrightarrow c < d \Longrightarrow a + c + 1 \leq b + d$ **for** $a\ b\ c$ :: *nat*
⟨*proof*⟩

**lemmas** [*smt-arith-combine*] =
*add-strict-mono*
*add-less-le-mono*
*add-mono*
*add-le-less-mono*

**lemma** [*smt-arith-combine*]:
‹$m < n \Longrightarrow c = d \Longrightarrow m + c < n + d$›
‹$m \leq n \Longrightarrow c = d \Longrightarrow m + c \leq n + d$›
‹$c = d \Longrightarrow m < n \Longrightarrow m + c < n + d$›
‹$c = d \Longrightarrow m \leq n \Longrightarrow m + c \leq n + d$›
**for** $m$ :: ‹$'a$ :: *ordered-cancel-ab-semigroup-add*›
⟨*proof*⟩

**lemma** *verit-negate-coefficient*:
‹$a \leq (b$ :: $'a$ :: {*ordered-ab-group-add*}) $\Longrightarrow -a \geq -b$›
‹$a < b \Longrightarrow -a > -b$›
‹$a = b \Longrightarrow -a = -b$›
⟨*proof*⟩

**end**

**lemma** *verit-ite-intro*:
‹(*if* $a$ *then* $P$ (*if* $a$ *then* $a'$ *else* $b'$) *else* $Q$) $\longleftrightarrow$ (*if* $a$ *then* $P\ a'$ *else* $Q$)›
‹(*if* $a$ *then* $P'$ *else* $Q'$ (*if* $a$ *then* $a'$ *else* $b'$)) $\longleftrightarrow$ (*if* $a$ *then* $P'$ *else* $Q'\ b'$)›
‹$A = f$ (*if* $a$ *then* $R$ *else* $S$) $\longleftrightarrow$ (*if* $a$ *then* $A = f\ R$ *else* $A = f\ S$)›
⟨*proof*⟩

**lemma** *verit-ite-if-cong*:
**fixes** $x\ y$ :: *bool*
**assumes** $b=c$
**and** $c \equiv$ *True* $\Longrightarrow x = u$
**and** $c \equiv$ *False* $\Longrightarrow y = v$
**shows** (*if* $b$ *then* $x$ *else* $y$) $\equiv$ (*if* $c$ *then* $u$ *else* $v$)
⟨*proof*⟩

## 63.7 Setup

⟨*ML*⟩

## 63.8 Configuration

The current configuration can be printed by the command *smt-status*, which shows the values of most options.

## 63.9 General configuration options

The option *smt-solver* can be used to change the target SMT solver. The possible values can be obtained from the *smt-status* command.

**declare** [[*smt-solver* = *z3*]]

Since SMT solvers are potentially nonterminating, there is a timeout (given in seconds) to restrict their runtime.

**declare** [[*smt-timeout* = *0*]]

SMT solvers apply randomized heuristics. In case a problem is not solvable by an SMT solver, changing the following option might help.

**declare** [[*smt-random-seed* = *1*]]

In general, the binding to SMT solvers runs as an oracle, i.e, the SMT solvers are fully trusted without additional checks. The following option can cause the SMT solver to run in proof-producing mode, giving a checkable certificate. This is currently implemented only for veriT and Z3.

**declare** [[*smt-oracle* = *false*]]

Each SMT solver provides several command-line options to tweak its behaviour. They can be passed to the solver by setting the following options.

**declare** [[*cvc4-options* = ]]
**declare** [[*cvc5-options* = ]]
**declare** [[*cvc5-proof-options* = *−−proof−format−mode=alethe −−proof−granularity=dsl−rewrite*]]
**declare** [[*verit-options* = ]]
**declare** [[*z3-options* = ]]

The SMT method provides an inference mechanism to detect simple triggers in quantified formulas, which might increase the number of problems solvable by SMT solvers (note: triggers guide quantifier instantiations in the SMT solver). To turn it on, set the following option.

**declare** [[*smt-infer-triggers* = *false*]]

Enable the following option to use built-in support for datatypes, codatatypes, and records in CVC4 and cvc5. Currently, this is implemented only in oracle mode.

**declare** [[*cvc-extensions* = *false*]]

Enable the following option to use built-in support for div/mod, datatypes, and records in Z3. Currently, this is implemented only in oracle mode.

**declare** [[*z3-extensions* = *false*]]

## 63.10 Certificates

By setting the option *smt-certificates* to the name of a file, all following applications of an SMT solver are cached in that file. Any further application of the same SMT solver (using the very same configuration) re-uses the cached certificate instead of invoking the solver. An empty string disables caching certificates.

The filename should be given as an explicit path. It is good practice to use the name of the current theory (with ending *.certs* instead of *.thy*) as the certificates file. Certificate files should be used at most once in a certain theory context, to avoid race conditions with other concurrent accesses.

**declare** [[*smt-certificates* = ]]

The option *smt-read-only-certificates* controls whether only stored certificates should be used or invocation of an SMT solver is allowed. When set to *true*, no SMT solver will ever be invoked and only the existing certificates found in the configured cache are used; when set to *false* and there is no cached certificate for some proposition, then the configured SMT solver is invoked.

**declare** [[*smt-read-only-certificates* = *false*]]

## 63.11 Tracing

The SMT method, when applied, traces important information. To make it entirely silent, set the following option to *false*.

**declare** [[*smt-verbose* = *true*]]

For tracing the generated problem file given to the SMT solver as well as the returned result of the solver, the option *smt-trace* should be set to *true*.

**declare** [[*smt-trace* = *false*]]

## 63.12 Schematic rules for Z3 proof reconstruction

Several prof rules of Z3 are not very well documented. There are two lemma groups which can turn failing Z3 proof reconstruction attempts into succeeding ones: the facts in *z3-rule* are tried prior to any implemented reconstruction procedure for all uncertain Z3 proof rules; the facts in *z3-simp* are only fed to invocations of the simplifier when reconstructing theory-specific proof steps.

**lemmas** [*z3-rule*] =
*refl eq-commute conj-commute disj-commute simp-thms nnf-simps*
*ring-distribs field-simps times-divide-eq-right times-divide-eq-left*
*if-True if-False not-not*
*NO-MATCH-def*

**lemma** [*z3-rule*]:
$(P \wedge Q) = (\neg (\neg P \vee \neg Q))$
$(P \wedge Q) = (\neg (\neg Q \vee \neg P))$
$(\neg P \wedge Q) = (\neg (P \vee \neg Q))$
$(\neg P \wedge Q) = (\neg (\neg Q \vee P))$
$(P \wedge \neg Q) = (\neg (\neg P \vee Q))$
$(P \wedge \neg Q) = (\neg (Q \vee \neg P))$
$(\neg P \wedge \neg Q) = (\neg (P \vee Q))$
$(\neg P \wedge \neg Q) = (\neg (Q \vee P))$
$\langle proof \rangle$

**lemma** [*z3-rule*]:
$(P \longrightarrow Q) = (Q \vee \neg P)$
$(\neg P \longrightarrow Q) = (P \vee Q)$
$(\neg P \longrightarrow Q) = (Q \vee P)$
$(\mathit{True} \longrightarrow P) = P$
$(P \longrightarrow \mathit{True}) = \mathit{True}$
$(\mathit{False} \longrightarrow P) = \mathit{True}$
$(P \longrightarrow P) = \mathit{True}$
$(\neg (A \longleftrightarrow \neg B)) \longleftrightarrow (A \longleftrightarrow B)$
$\langle proof \rangle$

**lemma** [*z3-rule*]:
$((P = Q) \longrightarrow R) = (R \vee (Q = (\neg P)))$
$\langle proof \rangle$

**lemma** [*z3-rule*]:
$(\neg \mathit{True}) = \mathit{False}$
$(\neg \mathit{False}) = \mathit{True}$
$(x = x) = \mathit{True}$
$(P = \mathit{True}) = P$
$(\mathit{True} = P) = P$
$(P = \mathit{False}) = (\neg P)$
$(\mathit{False} = P) = (\neg P)$
$((\neg P) = P) = \mathit{False}$
$(P = (\neg P)) = \mathit{False}$
$((\neg P) = (\neg Q)) = (P = Q)$
$\neg (P = (\neg Q)) = (P = Q)$
$\neg ((\neg P) = Q) = (P = Q)$
$(P \neq Q) = (Q = (\neg P))$
$(P = Q) = ((\neg P \vee Q) \wedge (P \vee \neg Q))$
$(P \neq Q) = ((\neg P \vee \neg Q) \wedge (P \vee Q))$
$\langle proof \rangle$

**lemma** [*z3-rule*]:
$(\mathit{if}\ P\ \mathit{then}\ P\ \mathit{else}\ \neg P) = \mathit{True}$
$(\mathit{if}\ \neg P\ \mathit{then}\ \neg P\ \mathit{else}\ P) = \mathit{True}$
$(\mathit{if}\ P\ \mathit{then}\ \mathit{True}\ \mathit{else}\ \mathit{False}) = P$

$(\textit{if } P \textit{ then False else True}) = (\neg\ P)$
$(\textit{if } P \textit{ then } Q \textit{ else True}) = ((\neg\ P) \vee Q)$
$(\textit{if } P \textit{ then } Q \textit{ else True}) = (Q \vee (\neg\ P))$
$(\textit{if } P \textit{ then } Q \textit{ else } \neg\ Q) = (P = Q)$
$(\textit{if } P \textit{ then } Q \textit{ else } \neg\ Q) = (Q = P)$
$(\textit{if } P \textit{ then } \neg\ Q \textit{ else } Q) = (P = (\neg\ Q))$
$(\textit{if } P \textit{ then } \neg\ Q \textit{ else } Q) = ((\neg\ Q) = P)$
$(\textit{if } \neg\ P \textit{ then } x \textit{ else } y) = (\textit{if } P \textit{ then } y \textit{ else } x)$
$(\textit{if } P \textit{ then } (\textit{if } Q \textit{ then } x \textit{ else } y) \textit{ else } x) = (\textit{if } P \wedge (\neg\ Q) \textit{ then } y \textit{ else } x)$
$(\textit{if } P \textit{ then } (\textit{if } Q \textit{ then } x \textit{ else } y) \textit{ else } x) = (\textit{if } (\neg\ Q) \wedge P \textit{ then } y \textit{ else } x)$
$(\textit{if } P \textit{ then } (\textit{if } Q \textit{ then } x \textit{ else } y) \textit{ else } y) = (\textit{if } P \wedge Q \textit{ then } x \textit{ else } y)$
$(\textit{if } P \textit{ then } (\textit{if } Q \textit{ then } x \textit{ else } y) \textit{ else } y) = (\textit{if } Q \wedge P \textit{ then } x \textit{ else } y)$
$(\textit{if } P \textit{ then } x \textit{ else if } P \textit{ then } y \textit{ else } z) = (\textit{if } P \textit{ then } x \textit{ else } z)$
$(\textit{if } P \textit{ then } x \textit{ else if } Q \textit{ then } x \textit{ else } y) = (\textit{if } P \vee Q \textit{ then } x \textit{ else } y)$
$(\textit{if } P \textit{ then } x \textit{ else if } Q \textit{ then } x \textit{ else } y) = (\textit{if } Q \vee P \textit{ then } x \textit{ else } y)$
$(\textit{if } P \textit{ then } x = y \textit{ else } x = z) = (x = (\textit{if } P \textit{ then } y \textit{ else } z))$
$(\textit{if } P \textit{ then } x = y \textit{ else } y = z) = (y = (\textit{if } P \textit{ then } x \textit{ else } z))$
$(\textit{if } P \textit{ then } x = y \textit{ else } z = y) = (y = (\textit{if } P \textit{ then } x \textit{ else } z))$
$\langle \textit{proof} \rangle$

**lemma** [*z3-rule*]:
$0 + (x::int) = x$
$x + 0 = x$
$x + x = 2 * x$
$0 * x = 0$
$1 * x = x$
$x + y = y + x$
$\langle \textit{proof} \rangle$

**lemma** [*z3-rule*]:
$P = Q \vee P \vee Q$
$P = Q \vee \neg\ P \vee \neg\ Q$
$(\neg\ P) = Q \vee \neg\ P \vee Q$
$(\neg\ P) = Q \vee P \vee \neg\ Q$
$P = (\neg\ Q) \vee \neg\ P \vee Q$
$P = (\neg\ Q) \vee P \vee \neg\ Q$
$P \neq Q \vee P \vee \neg\ Q$
$P \neq Q \vee \neg\ P \vee Q$
$P \neq (\neg\ Q) \vee P \vee Q$
$(\neg\ P) \neq Q \vee P \vee Q$
$P \vee Q \vee P \neq (\neg\ Q)$
$P \vee Q \vee (\neg\ P) \neq Q$
$P \vee \neg\ Q \vee P \neq Q$
$\neg\ P \vee Q \vee P \neq Q$
$P \vee y = (\textit{if } P \textit{ then } x \textit{ else } y)$
$P \vee (\textit{if } P \textit{ then } x \textit{ else } y) = y$
$\neg\ P \vee x = (\textit{if } P \textit{ then } x \textit{ else } y)$
$\neg\ P \vee (\textit{if } P \textit{ then } x \textit{ else } y) = x$
$P \vee R \vee \neg\ (\textit{if } P \textit{ then } Q \textit{ else } R)$

$\neg P \vee Q \vee \neg$ (*if* $P$ *then* $Q$ *else* $R$)
$\neg$ (*if* $P$ *then* $Q$ *else* $R$) $\vee \neg P \vee Q$
$\neg$ (*if* $P$ *then* $Q$ *else* $R$) $\vee P \vee R$
(*if* $P$ *then* $Q$ *else* $R$) $\vee \neg P \vee \neg Q$
(*if* $P$ *then* $Q$ *else* $R$) $\vee P \vee \neg R$
(*if* $P$ *then* $\neg Q$ *else* $R$) $\vee \neg P \vee Q$
(*if* $P$ *then* $Q$ *else* $\neg R$) $\vee P \vee R$
⟨*proof*⟩

**hide-type** (**open**) *symb-list pattern*
**hide-const** (**open**) *Symb-Nil Symb-Cons trigger pat nopat fun-app z3div z3mod*

**end**

# 64 Sledgehammer: Isabelle–ATP Linkup

**theory** *Sledgehammer*
**imports**
— FIXME: *HOL.Try0-HOL* has to be imported first so that *try0-schedule* gets the value assigned value there. Otherwise, the value is the one assigned in *HOL.Try0*, which is imported transitively by both *HOL.Presburger* and *HOL.SMT*. It seems that, when merging the attributes from two theories, the value assigned int the leftmost theory has precedence.
*Try0-HOL*
*Presburger*
*SMT*
**keywords**
*sledgehammer* :: *diag* **and**
*sledgehammer-params* :: *thy-decl*
**begin**

⟨*ML*⟩

**end**

# 65 Setup for Lifting/Transfer for the set type

**theory** *Lifting-Set*
**imports** *Lifting Groups-Big*
**begin**

## 65.1 Relator and predicator properties

**lemma** *rel-setD1*: $[\![$ *rel-set* $R\ A\ B$; $x \in A$ $]\!] \Longrightarrow \exists y \in B.\ R\ x\ y$
**and** *rel-setD2*: $[\![$ *rel-set* $R\ A\ B$; $y \in B$ $]\!] \Longrightarrow \exists x \in A.\ R\ x\ y$
⟨*proof*⟩

**lemma** *rel-set-conversep* [*simp*]: *rel-set* $A^{-1-1}$ = (*rel-set* $A)^{-1-1}$

⟨*proof*⟩

**lemma** *rel-set-eq* [*relator-eq*]: *rel-set* $(=) = (=)$
⟨*proof*⟩

**lemma** *rel-set-mono*[*relator-mono*]:
**assumes** $A \leq B$
**shows** *rel-set* $A \leq$ *rel-set* $B$
⟨*proof*⟩

**lemma** *rel-set-OO*[*relator-distr*]: *rel-set* $R$ *OO* *rel-set* $S$ = *rel-set* ($R$ *OO* $S$)
⟨*proof*⟩

**lemma** *Domainp-set*[*relator-domain*]:
*Domainp* (*rel-set* $T$) = ($\lambda A$. *Ball* $A$ (*Domainp* $T$))
⟨*proof*⟩

**lemma** *left-total-rel-set*[*transfer-rule*]:
*left-total* $A \Longrightarrow$ *left-total* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *left-unique-rel-set*[*transfer-rule*]:
*left-unique* $A \Longrightarrow$ *left-unique* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *right-total-rel-set* [*transfer-rule*]:
*right-total* $A \Longrightarrow$ *right-total* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *right-unique-rel-set* [*transfer-rule*]:
*right-unique* $A \Longrightarrow$ *right-unique* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *bi-total-rel-set* [*transfer-rule*]:
*bi-total* $A \Longrightarrow$ *bi-total* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *bi-unique-rel-set* [*transfer-rule*]:
*bi-unique* $A \Longrightarrow$ *bi-unique* (*rel-set* $A$)
⟨*proof*⟩

**lemma** *set-relator-eq-onp* [*relator-eq-onp*]:
*rel-set* (*eq-onp* $P$) = *eq-onp* ($\lambda A$. *Ball* $A$ $P$)
⟨*proof*⟩

**lemma** *bi-unique-rel-set-lemma*:
**assumes** *bi-unique* $R$ **and** *rel-set* $R$ $X$ $Y$
**obtains** $f$ **where** $Y$ = *image* $f$ $X$ **and** *inj-on* $f$ $X$ **and** $\forall x \in X.\ R\ x\ (f\ x)$
⟨*proof*⟩

## 65.2 Quotient theorem for the Lifting package

**lemma** *Quotient-set*[*quot-map*]:
**assumes** *Quotient R Abs Rep T*
**shows** *Quotient* (*rel-set R*) (*image Abs*) (*image Rep*) (*rel-set T*)
⟨*proof*⟩

## 65.3 Transfer rules for the Transfer package

### 65.3.1 Unconditional transfer rules

**context includes** *lifting-syntax*
**begin**

**lemma** *empty-transfer* [*transfer-rule*]: (*rel-set A*) {} {}
⟨*proof*⟩

**lemma** *insert-transfer* [*transfer-rule*]:
(*A* ===> *rel-set A* ===> *rel-set A*) *insert insert*
⟨*proof*⟩

**lemma** *union-transfer* [*transfer-rule*]:
(*rel-set A* ===> *rel-set A* ===> *rel-set A*) *union union*
⟨*proof*⟩

**lemma** *Union-transfer* [*transfer-rule*]:
(*rel-set* (*rel-set A*) ===> *rel-set A*) *Union Union*
⟨*proof*⟩

**lemma** *image-transfer* [*transfer-rule*]:
((*A* ===> *B*) ===> *rel-set A* ===> *rel-set B*) *image image*
⟨*proof*⟩

**lemma** *UNION-transfer* [*transfer-rule*]: — TODO deletion candidate
(*rel-set A* ===> (*A* ===> *rel-set B*) ===> *rel-set B*) $(\lambda A\ f.\ \bigcup(f\ `\ A))$ $(\lambda A\ f.\ \bigcup(f\ `\ A))$
⟨*proof*⟩

**lemma** *Ball-transfer* [*transfer-rule*]:
(*rel-set A* ===> (*A* ===> (=)) ===> (=)) *Ball Ball*
⟨*proof*⟩

**lemma** *Bex-transfer* [*transfer-rule*]:
(*rel-set A* ===> (*A* ===> (=)) ===> (=)) *Bex Bex*
⟨*proof*⟩

**lemma** *Pow-transfer* [*transfer-rule*]:
(*rel-set A* ===> *rel-set* (*rel-set A*)) *Pow Pow*
⟨*proof*⟩

**lemma** *rel-set-transfer* [*transfer-rule*]:
((*A* ===> *B* ===> (=)) ===> *rel-set A* ===> *rel-set B* ===> (=)) *rel-set rel-set*
⟨*proof*⟩

**lemma** *bind-transfer* [*transfer-rule*]:
(*rel-set A* ===> (*A* ===> *rel-set B*) ===> *rel-set B*) *Set.bind Set.bind*
⟨*proof*⟩

**lemma** *INF-parametric* [*transfer-rule*]: — TODO deletion candidate
(*rel-set A* ===> (*A* ===> *HOL.eq*) ===> *HOL.eq*) ($\lambda A$ *f*. *Inf* (*f* ' *A*)) ($\lambda A$ *f*. *Inf* (*f* ' *A*))
⟨*proof*⟩

**lemma** *SUP-parametric* [*transfer-rule*]: — TODO deletion candidate
(*rel-set R* ===> (*R* ===> *HOL.eq*) ===> *HOL.eq*) ($\lambda A$ *f*. *Sup* (*f* ' *A*)) ($\lambda A$ *f*. *Sup* (*f* ' *A*))
⟨*proof*⟩

### 65.3.2 Rules requiring bi-unique, bi-total or right-total relations

**lemma** *member-transfer* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*A* ===> *rel-set A* ===> (=)) (∈) (∈)
⟨*proof*⟩

**lemma** *right-total-Collect-transfer*[*transfer-rule*]:
**assumes** *right-total A*
**shows** ((*A* ===> (=)) ===> *rel-set A*) ($\lambda P$. *Collect* ($\lambda x$. *P x* ∧ *Domainp A x*)) *Collect*
⟨*proof*⟩

**lemma** *Collect-transfer* [*transfer-rule*]:
**assumes** *bi-total A*
**shows** ((*A* ===> (=)) ===> *rel-set A*) *Collect Collect*
⟨*proof*⟩

**lemma** *inter-transfer* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*rel-set A* ===> *rel-set A* ===> *rel-set A*) *inter inter*
⟨*proof*⟩

**lemma** *Diff-transfer* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*rel-set A* ===> *rel-set A* ===> *rel-set A*) (−) (−)
⟨*proof*⟩

**lemma** *subset-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*

**shows** (*rel-set A* ===> *rel-set A* ===> (=)) (⊆) (⊆)
⟨*proof*⟩

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *strict-subset-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*rel-set A* ===> *rel-set A* ===> (=)) (⊂) (⊂)
⟨*proof*⟩

**end**

**declare** *right-total-UNIV-transfer*[*transfer-rule*]

**lemma** *UNIV-transfer* [*transfer-rule*]:
**assumes** *bi-total A*
**shows** (*rel-set A*) *UNIV UNIV*
⟨*proof*⟩

**lemma** *right-total-Compl-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A* **and** [*transfer-rule*]: *right-total A*
**shows** (*rel-set A* ===> *rel-set A*) ($\lambda S$. *uminus S* ∩ *Collect* (*Domainp A*)) *uminus*
⟨*proof*⟩

**lemma** *Compl-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A* **and** [*transfer-rule*]: *bi-total A*
**shows** (*rel-set A* ===> *rel-set A*) *uminus uminus*
⟨*proof*⟩

**lemma** *right-total-Inter-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A* **and** [*transfer-rule*]: *right-total A*
**shows** (*rel-set* (*rel-set A*) ===> *rel-set A*) ($\lambda S$. $\bigcap$ *S* ∩ *Collect* (*Domainp A*)) *Inter*
⟨*proof*⟩

**lemma** *Inter-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A* **and** [*transfer-rule*]: *bi-total A*
**shows** (*rel-set* (*rel-set A*) ===> *rel-set A*) *Inter Inter*
⟨*proof*⟩

**lemma** *filter-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** ((*A* ===> (=)) ===> *rel-set A* ===> *rel-set A*) *Set.filter Set.filter*
⟨*proof*⟩

**lemma** *finite-transfer* [*transfer-rule*]:

*bi-unique A* $\Longrightarrow$ (*rel-set A* ===> (=)) *finite finite*
⟨*proof*⟩

**lemma** *card-transfer* [*transfer-rule*]:
*bi-unique A* $\Longrightarrow$ (*rel-set A* ===> (=)) *card card*
⟨*proof*⟩

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *vimage-right-total-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique B right-total A*
**shows** ((*A* ===> *B*) ===> *rel-set B* ===> *rel-set A*) ($\lambda f\, X.\ f$ −' $X \cap$ *Collect* (*Domainp A*)) *vimage*
⟨*proof*⟩

**end**

**lemma** *vimage-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total A bi-unique B*
**shows** ((*A* ===> *B*) ===> *rel-set B* ===> *rel-set A*) *vimage vimage*
⟨*proof*⟩

**lemma** *Image-parametric* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*rel-set* (*rel-prod A B*) ===> *rel-set A* ===> *rel-set B*) ('') ('')
⟨*proof*⟩

**lemma** *inj-on-transfer*[*transfer-rule*]:
((*A* ===> *B*) ===> *rel-set A* ===> (=)) *inj-on inj-on*
**if** [*transfer-rule*]: *bi-unique A bi-unique B*
⟨*proof*⟩

**end**

**lemma** (**in** *comm-monoid-set*) *F-parametric* [*transfer-rule*]:
**fixes** $A :: {'b} \Rightarrow {'c} \Rightarrow bool$
**assumes** *bi-unique A*
**shows** *rel-fun* (*rel-fun A* (=)) (*rel-fun* (*rel-set A*) (=)) *F F*
⟨*proof*⟩

**lemmas** *sum-parametric* = *sum.F-parametric*
**lemmas** *prod-parametric* = *prod.F-parametric*

**lemma** *rel-set-UNION*:
**assumes** [*transfer-rule*]: *rel-set Q A B rel-fun Q* (*rel-set R*) *f g*
**shows** *rel-set R* ($\bigcup$(*f* ' *A*)) ($\bigcup$(*g* ' *B*))
⟨*proof*⟩

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *fold-graph-transfer*[*transfer-rule*]:
**assumes** *bi-unique R right-total R*
**shows** ((*R* ===> (=) ===> (=)) ===> (=) ===> *rel-set R* ===> (=) ===> (=)) *fold-graph fold-graph*
⟨*proof*⟩

**lemma** *fold-transfer*[*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique R right-total R*
**shows** ((*R* ===> (=) ===> (=)) ===> (=) ===> *rel-set R* ===> (=)) *Finite-Set.fold Finite-Set.fold*
⟨*proof*⟩

**end**

**end**

# 66 The datatype of finite lists

**theory** *List*
**imports** *Sledgehammer Lifting-Set*
**begin**

**datatype** (*set*: *'a*) *list* =
*Nil* (‹[]›)
| *Cons* (*hd*: *'a*) (*tl*: *'a list*) (**infixr** ‹#› *65*)
**for**
*map*: *map*
*rel*: *list-all2*
*pred*: *list-all*
**where**
*tl* [] = []

**bundle** *list-syntax*
**begin**
**notation** *Nil* (‹[]›)
**and** *Cons* (**infixr** ‹#› *65*)
**end**

**datatype-compat** *list*

**lemma** [*case-names Nil Cons*, *cases type*: *list*]:
— for backward compatibility – names of variables differ
$(y = [] \Longrightarrow P) \Longrightarrow (\bigwedge a\ list.\ y = a\ \#\ list \Longrightarrow P) \Longrightarrow P$

⟨*proof*⟩

**lemma** [*case-names Nil Cons*, *induct type*: *list*]:
— for backward compatibility – names of variables differ
$P\ [] \Longrightarrow (\bigwedge a\ list.\ P\ list \Longrightarrow P\ (a\ \#\ list)) \Longrightarrow P\ list$
⟨*proof*⟩

Compatibility:

⟨*ML*⟩

**lemmas** *inducts* = *list.induct*
**lemmas** *recs* = *list.rec*
**lemmas** *cases* = *list.case*

⟨*ML*⟩

**lemmas** *set-simps* = *list.set*

List enumeration

**open-bundle** *list-enumeration-syntax*
**begin**

**syntax**
*-list* :: *args* ⇒ *'a list* (‹(‹*indent=1 notation=*‹*mixfix list enumeration*››[-])›)
**syntax-consts**
*-list* ⇌ *Cons*
**translations**
[*x*, *xs*] ⇌ *x*#[*xs*]
[*x*] ⇌ *x*#[]

**end**

## 66.1 Basic list processing functions

**primrec** (*nonexhaustive*) *last* :: *'a list* ⇒ *'a* **where**
*last* (*x* # *xs*) = (*if xs* = [] *then x else last xs*)

**primrec** *butlast* :: *'a list* ⇒ *'a list* **where**
*butlast* [] = [] |
*butlast* (*x* # *xs*) = (*if xs* = [] *then* [] *else x* # *butlast xs*)

**lemma** *set-rec*: *set xs* = *rec-list* {} (λ*x* -. *insert x*) *xs*
⟨*proof*⟩

**definition** *coset* :: *'a list* ⇒ *'a set* **where**
[*simp*]: *coset xs* = − *set xs*

**primrec** *append* :: *'a list* ⇒ *'a list* ⇒ *'a list* (**infixr** ‹@› *65*) **where**
*append-Nil*: [] @ *ys* = *ys* |

*append-Cons*: $(x\#xs)$ @ $ys = x \# xs$ @ $ys$

**primrec** *rev* :: $'a$ *list* $\Rightarrow$ $'a$ *list* **where**
*rev* [] = [] |
*rev* $(x \# xs)$ = *rev xs* @ $[x]$

**primrec** *filter*:: $('a \Rightarrow bool) \Rightarrow 'a$ *list* $\Rightarrow$ $'a$ *list* **where**
*filter P* [] = [] |
*filter P* $(x \# xs)$ = (*if P x then x* # *filter P xs else filter P xs*)

**open-bundle** *filter-syntax* — Special input syntax for filter
**begin**

**syntax** (*ASCII*)
  *-filter* :: [*pttrn*, $'a$ *list*, *bool*] => $'a$ *list* (‹(‹*indent=1 notation=*‹*mixfix filter*››[-<−-./
-])›)
**syntax**
  *-filter* :: [*pttrn*, $'a$ *list*, *bool*] => $'a$ *list* (‹(‹*indent=1 notation=*‹*mixfix filter*››[-←-
./ -])›)
**syntax-consts**
  *-filter* ⇌ *filter*
**translations**
  [*x*<−*xs* . *P*] ⇀ *CONST filter* (λ*x*. *P*) *xs*

**end**

**primrec** *fold* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'a$ *list* $\Rightarrow 'b \Rightarrow 'b$ **where**
*fold-Nil*: *fold f* [] = *id* |
*fold-Cons*: *fold f* $(x \# xs)$ = *fold f xs* ∘ *f x*

**primrec** *foldr* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'a$ *list* $\Rightarrow 'b \Rightarrow 'b$ **where**
*foldr-Nil*: *foldr f* [] = *id* |
*foldr-Cons*: *foldr f* $(x \# xs)$ = *f x* ∘ *foldr f xs*

**primrec** *foldl* :: $('b \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$ *list* $\Rightarrow 'b$ **where**
*foldl-Nil*: *foldl f a* [] = *a* |
*foldl-Cons*: *foldl f a* $(x \# xs)$ = *foldl f* $(f\ a\ x)$ *xs*

**primrec** *concat*:: $'a$ *list list* $\Rightarrow 'a$ *list* **where**
*concat* [] = [] |
*concat* $(x \# xs)$ = *x* @ *concat xs*

**primrec** *drop*:: *nat* $\Rightarrow 'a$ *list* $\Rightarrow 'a$ *list* **where**
*drop-Nil*: *drop n* [] = [] |
*drop-Cons*: *drop n* $(x \# xs)$ = (*case n of 0* ⇒ *x* # *xs* | *Suc m* ⇒ *drop m xs*)
  — Warning: simpset does not contain this definition, but separate theorems for
$n = 0$ and $n = Suc\ k$

**primrec** *take*:: *nat* $\Rightarrow 'a$ *list* $\Rightarrow 'a$ *list* **where**

*take-Nil*:*take n* [] = [] |
*take-Cons*: *take n* (*x* # *xs*) = (*case n of 0* ⇒ [] | *Suc m* ⇒ *x* # *take m xs*)
— Warning: simpset does not contain this definition, but separate theorems for $n = 0$ and $n = Suc\ k$

**primrec** (*nonexhaustive*) *nth* :: *'a list* => *nat* => *'a* (**infixl** ‹!› *100*) **where**
*nth-Cons*: (*x* # *xs*) ! *n* = (*case n of 0* ⇒ *x* | *Suc k* ⇒ *xs* ! *k*)
— Warning: simpset does not contain this definition, but separate theorems for $n = 0$ and $n = Suc\ k$

**primrec** *list-update* :: *'a list* ⇒ *nat* ⇒ *'a* ⇒ *'a list* **where**
*list-update* [] *i v* = [] |
*list-update* (*x* # *xs*) *i v* =
(*case i of 0* ⇒ *v* # *xs* | *Suc j* ⇒ *x* # *list-update xs j v*)

**nonterminal** *lupdbinds* **and** *lupdbind*

**open-bundle** *list-update-syntax*
**begin**

**syntax**
*-lupdbind*:: [*'a*, *'a*] => *lupdbind* (‹(‹*indent=2 notation=*‹*mixfix update*››- :=/ -)›)
:: *lupdbind* => *lupdbinds* (‹-›)
*-lupdbinds* :: [*lupdbind*, *lupdbinds*] => *lupdbinds* (‹-,/ -›)
*-LUpdate* :: [*'a*, *lupdbinds*] => *'a*
(‹(‹*open-block notation=*‹*mixfix list update*››-/[(-)])› [*1000*,*0*] *900*)
**syntax-consts**
*-LUpdate* ⇌ *list-update*
**translations**
*-LUpdate xs* (*-lupdbinds b bs*) == *-LUpdate* (*-LUpdate xs b*) *bs*
*xs*[*i*:=*x*] == *CONST list-update xs i x*

**end**

**primrec** *takeWhile* :: (*'a* ⇒ *bool*) ⇒ *'a list* ⇒ *'a list* **where**
*takeWhile P* [] = [] |
*takeWhile P* (*x* # *xs*) = (*if P x then x* # *takeWhile P xs else* [])

**primrec** *dropWhile* :: (*'a* ⇒ *bool*) ⇒ *'a list* ⇒ *'a list* **where**
*dropWhile P* [] = [] |
*dropWhile P* (*x* # *xs*) = (*if P x then dropWhile P xs else x* # *xs*)

**primrec** *zip* :: *'a list* ⇒ *'b list* ⇒ (*'a* × *'b*) *list* **where**
*zip xs* [] = [] |
*zip-Cons*: *zip xs* (*y* # *ys*) =
(*case xs of* [] ⇒ [] | *z* # *zs* ⇒ (*z*, *y*) # *zip zs ys*)
— Warning: simpset does not contain this definition, but separate theorems for *xs* = [] and *xs* = *z* # *zs*

**abbreviation** *map2* :: (′*a* ⇒ ′*b* ⇒ ′*c*) ⇒ ′*a list* ⇒ ′*b list* ⇒ ′*c list* **where**
*map2 f xs ys* ≡ *map* (λ(*x*,*y*). *f x y*) (*zip xs ys*)

**primrec** *product* :: ′*a list* ⇒ ′*b list* ⇒ (′*a* × ′*b*) *list* **where**
*product* [] - = [] |
*product* (*x*#*xs*) *ys* = *map* (*Pair x*) *ys* @ *product xs ys*

**hide-const** (**open**) *product*

**primrec** *product-lists* :: ′*a list list* ⇒ ′*a list list* **where**
*product-lists* [] = [[]] |
*product-lists* (*xs* # *xss*) = *concat* (*map* (λ*x*. *map* (*Cons x*) (*product-lists xss*)) *xs*)

**primrec** *upt* :: *nat* ⇒ *nat* ⇒ *nat list* (‹(‹*indent=1 notation=*‹*mixfix list interval*››[-..</-'])›) **where**
*upt-0*: [*i*..<*0*] = [] |
*upt-Suc*: [*i*..<(*Suc j*)] = (*if i* ≤ *j then* [*i*..<*j*] @ [*j*] *else* [])

**definition** *insert* :: ′*a* ⇒ ′*a list* ⇒ ′*a list* **where**
*insert x xs* = (*if x* ∈ *set xs then xs else x* # *xs*)

**definition** *union* :: ′*a list* ⇒ ′*a list* ⇒ ′*a list* **where**
*union* = *fold insert*

**hide-const** (**open**) *insert union*
**hide-fact** (**open**) *insert-def union-def*

**primrec** *find* :: (′*a* ⇒ *bool*) ⇒ ′*a list* ⇒ ′*a option* **where**
*find* - [] = *None* |
*find P* (*x*#*xs*) = (*if P x then Some x else find P xs*)

In the context of multisets, *count-list* is equivalent to *count* ∘ *mset* and it is advisable to use the latter.

**primrec** *count-list* :: ′*a list* ⇒ ′*a* ⇒ *nat* **where**
*count-list* [] *y* = *0* |
*count-list* (*x*#*xs*) *y* = (*if x*=*y then count-list xs y* + *1 else count-list xs y*)

**definition**
  *extract* :: (′*a* ⇒ *bool*) ⇒ ′*a list* ⇒ (′*a list* ∗ ′*a* ∗ ′*a list*) *option*
**where** *extract P xs* =
 (*case dropWhile* (*Not* ∘ *P*) *xs of*
   [] ⇒ *None* |
   *y*#*ys* ⇒ *Some*(*takeWhile* (*Not* ∘ *P*) *xs*, *y*, *ys*))

**hide-const** (**open**) *extract*

**primrec** *those* :: ′*a option list* ⇒ ′*a list option*
**where**

$those\ [] = Some\ []\ |$
$those\ (x\ \#\ xs) = (case\ x\ of$
$\ \ None \Rightarrow None$
$|\ Some\ y \Rightarrow map\text{-}option\ (Cons\ y)\ (those\ xs))$

**primrec** $remove1 :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$remove1\ x\ [] = []\ |$
$remove1\ x\ (y\ \#\ xs) = (if\ x = y\ then\ xs\ else\ y\ \#\ remove1\ x\ xs)$

**primrec** $removeAll :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$removeAll\ x\ [] = []\ |$
$removeAll\ x\ (y\ \#\ xs) = (if\ x = y\ then\ removeAll\ x\ xs\ else\ y\ \#\ removeAll\ x\ xs)$

**definition** $minus\text{-}list\text{-}mset :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$minus\text{-}list\text{-}mset\ xs\ ys = foldr\ remove1\ ys\ xs$

**definition** $minus\text{-}list\text{-}set :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$minus\text{-}list\text{-}set\ xs\ ys = foldr\ removeAll\ ys\ xs$

**definition** $inter\text{-}list\text{-}set :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$inter\text{-}list\text{-}set\ xs\ ys = filter\ (\lambda x.\ x \in set\ ys)\ xs$

**primrec** $distinct :: 'a\ list \Rightarrow bool$ **where**
$distinct\ [] \longleftrightarrow True\ |$
$distinct\ (x\ \#\ xs) \longleftrightarrow x \notin set\ xs \wedge distinct\ xs$

**fun** $successively :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$ **where**
$successively\ P\ []\ = True\ |$
$successively\ P\ [x] = True\ |$
$successively\ P\ (x\ \#\ y\ \#\ xs) = (P\ x\ y \wedge successively\ P\ (y\#xs))$

**definition** $distinct\text{-}adj$ **where**
$distinct\text{-}adj = successively\ (\neq)$

**primrec** $remdups :: 'a\ list \Rightarrow 'a\ list$ **where**
$remdups\ [] = []\ |$
$remdups\ (x\ \#\ xs) = (if\ x \in set\ xs\ then\ remdups\ xs\ else\ x\ \#\ remdups\ xs)$

**fun** $remdups\text{-}adj :: 'a\ list \Rightarrow 'a\ list$ **where**
$remdups\text{-}adj\ [] = []\ |$
$remdups\text{-}adj\ [x] = [x]\ |$
$remdups\text{-}adj\ (x\ \#\ y\ \#\ xs) = (if\ x = y\ then\ remdups\text{-}adj\ (x\ \#\ xs)\ else\ x\ \#\ remdups\text{-}adj\ (y\ \#\ xs))$

**primrec** $replicate :: nat \Rightarrow 'a \Rightarrow 'a\ list$ **where**
$replicate\text{-}0\text{: } replicate\ 0\ x = []\ |$
$replicate\text{-}Suc\text{: } replicate\ (Suc\ n)\ x = x\ \#\ replicate\ n\ x$

Function *size* is overloaded for all datatypes. Users may refer to the list version as *length*.

**abbreviation** *length* :: $'a$ *list* ⇒ *nat* **where**
*length* ≡ *size*

**definition** *enumerate* :: *nat* ⇒ $'a$ *list* ⇒ (*nat* × $'a$) *list* **where**
*enumerate-eq-zip*: *enumerate n xs* = *zip* [*n*..<*n* + *length xs*] *xs*

**primrec** *rotate1* :: $'a$ *list* ⇒ $'a$ *list* **where**
*rotate1* [] = [] |
*rotate1* (*x* # *xs*) = *xs* @ [*x*]

**definition** *rotate* :: *nat* ⇒ $'a$ *list* ⇒ $'a$ *list* **where**
*rotate n* = *rotate1* ^^ *n*

**definition** *nths* :: $'a$ *list* => *nat set* => $'a$ *list* **where**
*nths xs A* = *map fst* (*filter* (λ*p*. *snd p* ∈ *A*) (*zip xs* [*0*..<*size xs*]))

**primrec** *subseqs* :: $'a$ *list* ⇒ $'a$ *list list* **where**
*subseqs* [] = [[]] |
*subseqs* (*x*#*xs*) = (*let xss* = *subseqs xs in map* (*Cons x*) *xss* @ *xss*)

**primrec** *n-lists* :: *nat* ⇒ $'a$ *list* ⇒ $'a$ *list list* **where**
*n-lists 0 xs* = [[]] |
*n-lists* (*Suc n*) *xs* = *concat* (*map* (λ*ys*. *map* (λ*y*. *y* # *ys*) *xs*) (*n-lists n xs*))

**hide-const** (**open**) *n-lists*

**function** *splice* :: $'a$ *list* ⇒ $'a$ *list* ⇒ $'a$ *list* **where**
*splice* [] *ys* = *ys* |
*splice* (*x*#*xs*) *ys* = *x* # *splice ys xs*
⟨*proof*⟩

**termination**
⟨*proof*⟩

**function** *shuffles* **where**
*shuffles* [] *ys* = {*ys*}
| *shuffles xs* [] = {*xs*}
| *shuffles* (*x* # *xs*) (*y* # *ys*) = (#) *x* ‘ *shuffles xs* (*y* # *ys*) ∪ (#) *y* ‘ *shuffles* (*x* # *xs*) *ys*
⟨*proof*⟩
**termination** ⟨*proof*⟩

Use only if you cannot use *Min* instead:

**fun** *min-list* :: $'a$::*ord list* ⇒ $'a$ **where**
*min-list* (*x* # *xs*) = (*case xs of* [] ⇒ *x* | - ⇒ *min x* (*min-list xs*))

Returns first minimum:

**fun** *arg-min-list* :: ($'a$ ⇒ ($'b$::*linorder*)) ⇒ $'a$ *list* ⇒ $'a$ **where**

*arg-min-list f* $[x] = x$ |
*arg-min-list f* $(x\#y\#zs) = ($*let* $m =$ *arg-min-list f* $(y\#zs)$ *in if* $f\ x \leq f\ m$ *then* $x$ *else* $m)$

Figure 1 shows characteristic examples that should give an intuitive understanding of the above functions.

The following simple sort(ed) functions are intended for proofs, not for efficient implementations.

A sorted predicate w.r.t. a relation:

**fun** *sorted-wrt* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool$ **where**
*sorted-wrt* $P\ [] = True$ |
*sorted-wrt* $P\ (x\ \#\ ys) = ((\forall\, y \in set\ ys.\ P\ x\ y) \wedge$ *sorted-wrt* $P\ ys)$

A class-based sorted predicate:

**context** *linorder*
**begin**

**abbreviation** *sorted* :: $'a\ list \Rightarrow bool$ **where**
*sorted* $\equiv$ *sorted-wrt* $(\leq)$

**lemma** *sorted-simps*: *sorted* $[] = True$ *sorted* $(x\ \#\ ys) = ((\forall\, y \in set\ ys.\ x \leq y) \wedge$ *sorted* $ys)$
$\langle proof \rangle$

**lemma** *strict-sorted-simps*: *sorted-wrt* $(<)\ [] = True$ *sorted-wrt* $(<)\ (x\ \#\ ys) = ((\forall\, y \in set\ ys.\ x<y) \wedge$ *sorted-wrt* $(<)\ ys)$
$\langle proof \rangle$

**primrec** *insort-key* :: $('b \Rightarrow 'a) \Rightarrow 'b \Rightarrow 'b\ list \Rightarrow 'b\ list$ **where**
*insort-key* $f\ x\ [] = [x]$ |
*insort-key* $f\ x\ (y\#ys) =$
$($*if* $f\ x \leq f\ y$ *then* $(x\#y\#ys)$ *else* $y\#($*insort-key* $f\ x\ ys))$

**definition** *sort-key* :: $('b \Rightarrow 'a) \Rightarrow 'b\ list \Rightarrow 'b\ list$ **where**
*sort-key* $f\ xs =$ *foldr* $($*insort-key* $f)\ xs\ []$

**definition** *insort-insert-key* :: $('b \Rightarrow 'a) \Rightarrow 'b \Rightarrow 'b\ list \Rightarrow 'b\ list$ **where**
*insort-insert-key* $f\ x\ xs =$
$($*if* $f\ x \in f$ ` *set* $xs$ *then* $xs$ *else* *insort-key* $f\ x\ xs)$

**abbreviation** *sort* $\equiv$ *sort-key* $(\lambda x.\ x)$
**abbreviation** *insort* $\equiv$ *insort-key* $(\lambda x.\ x)$
**abbreviation** *insort-insert* $\equiv$ *insort-insert-key* $(\lambda x.\ x)$

**definition** *stable-sort-key* :: $(('b \Rightarrow 'a) \Rightarrow 'b\ list \Rightarrow 'b\ list) \Rightarrow bool$ **where**
*stable-sort-key* $sk =$
$(\forall f\ xs\ k.\ filter\ (\lambda y.\ f\ y = k)\ (sk\ f\ xs) = filter\ (\lambda y.\ f\ y = k)\ xs)$

$[a, b] @ [c, d] = [a, b, c, d]$
$\mathit{length}\ [a, b, c] = 3$
$\mathit{set}\ [a, b, c] = \{a, b, c\}$
$\mathit{map}\ f\ [a, b, c] = [f\ a, f\ b, f\ c]$
$\mathit{rev}\ [a, b, c] = [c, b, a]$
$\mathit{hd}\ [a, b, c, d] = a$
$\mathit{tl}\ [a, b, c, d] = [b, c, d]$
$\mathit{last}\ [a, b, c, d] = d$
$\mathit{butlast}\ [a, b, c, d] = [a, b, c]$
$\mathit{filter}\ (\lambda n{::}\mathit{nat}.\ n<2)\ [0,2,1] = [0,1]$
$\mathit{concat}\ [[a, b], [c, d, e], [], [f]] = [a, b, c, d, e, f]$
$\mathit{fold}\ f\ [a, b, c]\ x = f\ c\ (f\ b\ (f\ a\ x))$
$\mathit{foldr}\ f\ [a, b, c]\ x = f\ a\ (f\ b\ (f\ c\ x))$
$\mathit{foldl}\ f\ x\ [a, b, c] = f\ (f\ (f\ x\ a)\ b)\ c$
$\mathit{successively}\ (\neq)\ [\mathit{True}, \mathit{False}, \mathit{True}, \mathit{False}]$
$\mathit{zip}\ [a, b, c]\ [x, y, z] = [(a, x), (b, y), (c, z)]$
$\mathit{zip}\ [a, b]\ [x, y, z] = [(a, x), (b, y)]$
$\mathit{enumerate}\ 3\ [a, b, c] = [(3, a), (4, b), (5, c)]$
$\mathit{List.product}\ [a, b]\ [c, d] = [(a, c), (a, d), (b, c), (b, d)]$
$\mathit{product\text{-}lists}\ [[a, b], [c], [d, e]] = [[a, c, d], [a, c, e], [b, c, d], [b, c, e]]$
$\mathit{splice}\ [a, b, c]\ [x, y, z] = [a, x, b, y, c, z]$
$\mathit{splice}\ [a, b, c, d]\ [x, y] = [a, x, b, y, c, d]$
$\mathit{shuffles}\ [a, b]\ [c, d] = \{[a, b, c, d], [a, c, b, d], [a, c, d, b], [c, a, b, d], [c, a, d, b], [c, d, a, b]\}$
$\mathit{take}\ 2\ [a, b, c, d] = [a, b]$
$\mathit{take}\ 6\ [a, b, c, d] = [a, b, c, d]$
$\mathit{drop}\ 2\ [a, b, c, d] = [c, d]$
$\mathit{drop}\ 6\ [a, b, c, d] = []$
$\mathit{takeWhile}\ (\lambda n.\ n < 3)\ [1, 2, 3, 0] = [1, 2]$
$\mathit{dropWhile}\ (\lambda n.\ n < 3)\ [1, 2, 3, 0] = [3, 0]$
$\mathit{distinct}\ [2, 0, 1]$
$\mathit{remdups}\ [2, 0, 2, 1, 2] = [0, 1, 2]$
$\mathit{remdups\text{-}adj}\ [2, 2, 3, 1, 1, 2, 1] = [2, 3, 1, 2, 1]$
$\mathit{List.insert}\ 2\ [0, 1, 2] = [0, 1, 2]$
$\mathit{List.insert}\ 3\ [0, 1, 2] = [3, 0, 1, 2]$
$\mathit{List.union}\ [2, 3, 4]\ [0, 1, 2] = [4, 3, 0, 1, 2]$
$\mathit{find}\ ((<)\ 0)\ [0, 0] = \mathit{None}$
$\mathit{find}\ ((<)\ 0)\ [0, 1, 0, 2] = \mathit{Some}\ 1$
$\mathit{count\text{-}list}\ [0, 1, 0, 2]\ 0 = 2$
$\mathit{List.extract}\ ((<)\ 0)\ [0, 0] = \mathit{None}$
$\mathit{List.extract}\ ((<)\ 0)\ [0, 1, 0, 2] = \mathit{Some}\ ([0], 1, [0, 2])$
$\mathit{remove1}\ 2\ [2, 0, 2, 1, 2] = [0, 2, 1, 2]$
$\mathit{removeAll}\ 2\ [2, 0, 2, 1, 2] = [0, 1]$
$[a, b, c, d]\ !\ 2 = c$
$[a, b, c, d][2 := x] = [a, b, x, d]$
$\mathit{nths}\ [a, b, c, d, e]\ \{0, 2, 3\} = [a, c, d]$
$\mathit{subseqs}\ [a, b] = [[a, b], [a], [b], []]$
$\mathit{List.n\text{-}lists}\ 2\ [a, b, c] = [[a, a], [b, a], [c, a], [a, b], [b, b], [c, b], [a, c], [b, c], [c, c]]$
$\mathit{rotate1}\ [a, b, c, d] = [b, c, d, a]$
$\mathit{rotate}\ 3\ [a, b, c, d] = [d, a, b, c]$
$\mathit{replicate}\ 4\ a = [a, a, a, a]$
$[2..<5] = [2, 3, 4]$
$\mathit{min\text{-}list}\ [3, 1, -2] = -2$
$\mathit{arg\text{-}min\text{-}list}\ (\lambda i.\ i * i)\ [3, -1, 1, -2] = -1$

**lemma** *strict-sorted-iff*: *sorted-wrt* $(<)$ $l \longleftrightarrow$ *sorted* $l \land$ *distinct* $l$
⟨*proof*⟩

**lemma** *strict-sorted-imp-sorted*: *sorted-wrt* $(<)$ *xs* $\Longrightarrow$ *sorted xs*
⟨*proof*⟩

**end**

### 66.1.1 List comprehension

Input syntax for Haskell-like list comprehension notation. Typical example: $[(x,y).\ x \leftarrow xs,\ y \leftarrow ys,\ x \neq y]$, the list of all pairs of distinct elements from *xs* and *ys*. The syntax is as in Haskell, except that | becomes a dot (like in Isabelle's set comprehension): $[e.\ x \leftarrow xs,\ \ldots]$ rather than `[e| x <- xs, ...]`.

The qualifiers after the dot are

**generators** $p \leftarrow xs$, where $p$ is a pattern and *xs* an expression of list type, or

**guards** $b$, where $b$ is a boolean expression.

Just like in Haskell, list comprehension is just a shorthand. To avoid misunderstandings, the translation into desugared form is not reversed upon output. Note that the translation of $[e.\ x \leftarrow xs]$ is optmized to *map* $(\lambda x.\ e)$ *xs*.

It is easy to write short list comprehensions which stand for complex expressions. During proofs, they may become unreadable (and mangled). In such cases it can be advisable to introduce separate definitions for the list comprehensions in question.

**nonterminal** *lc-qual* **and** *lc-quals*

**open-bundle** *list-comprehension-syntax*
**begin**
**syntax**
  *-listcompr* :: *'a* ⇒ *lc-qual* ⇒ *lc-quals* ⇒ *'a list*  (‹[- . --›)
  *-lc-gen* :: *'a* ⇒ *'a list* ⇒ *lc-qual*  (‹- ← -›)
  *-lc-test* :: *bool* ⇒ *lc-qual* (‹-›)

  *-lc-end* :: *lc-quals* (‹]›)
  *-lc-quals* :: *lc-qual* ⇒ *lc-quals* ⇒ *lc-quals*  (‹, --›)

**syntax** (*ASCII*)
  *-lc-gen* :: *'a* ⇒ *'a list* ⇒ *lc-qual*  (‹- <– -›)
**end**

⟨*ML*⟩

**code-datatype** *set coset*
**hide-const** (**open**) *coset*

### 66.1.2 [] and (#)

**lemma** *not-Cons-self* [*simp*]:
$xs \neq x \# xs$
⟨*proof*⟩

**lemma** *not-Cons-self2* [*simp*]: $x \# xs \neq xs$
⟨*proof*⟩

**lemma** *neq-Nil-conv*: $(xs \neq []) = (\exists\, y\ ys.\ xs = y \# ys)$
⟨*proof*⟩

**lemma** *tl-Nil*: $tl\ xs = [] \longleftrightarrow xs = [] \lor (\exists\, x.\ xs = [x])$
⟨*proof*⟩

**lemmas** *Nil-tl* = *tl-Nil*[*THEN eq-iff-swap*]

**lemma** *length-induct*:
$(\bigwedge xs.\ \forall\, ys.\ length\ ys < length\ xs \longrightarrow P\ ys \Longrightarrow P\ xs) \Longrightarrow P\ xs$
⟨*proof*⟩

**lemma** *induct-list012*:
$[\![P\ [];\ \bigwedge x.\ P\ [x];\ \bigwedge x\ y\ zs.\ [\![\ P\ zs;\ P\ (y \# zs)\ ]\!] \Longrightarrow P\ (x \# y \# zs)]\!] \Longrightarrow P\ xs$
⟨*proof*⟩

**lemma** *list-nonempty-induct* [*consumes 1*, *case-names single cons*]:
$[\![\ xs \neq [];\ \bigwedge x.\ P\ [x];\ \bigwedge x\ xs.\ xs \neq [] \Longrightarrow P\ xs \Longrightarrow P\ (x \# xs)]\!] \Longrightarrow P\ xs$
⟨*proof*⟩

**lemma** *inj-split-Cons*: $inj\text{-}on\ (\lambda(xs,\ n).\ n\#xs)\ X$
⟨*proof*⟩

**lemma** *inj-on-Cons1* [*simp*]: $inj\text{-}on\ ((\#)\ x)\ A$
⟨*proof*⟩

### 66.1.3 *length*

Needs to come before @ because of theorem *append-eq-append-conv*.

**lemma** *length-append* [*simp*]: $length\ (xs\ @\ ys) = length\ xs + length\ ys$
⟨*proof*⟩

**lemma** *length-map* [*simp*]: $length\ (map\ f\ xs) = length\ xs$
⟨*proof*⟩

**lemma** *length-rev* [*simp*]: *length* (*rev xs*) = *length xs*
⟨*proof*⟩

**lemma** *length-tl* [*simp*]: *length* (*tl xs*) = *length xs* − *1*
⟨*proof*⟩

**lemma** *length-0-conv* [*iff*]: (*length xs* = *0*) = (*xs* = [])
⟨*proof*⟩

**lemma** *length-greater-0-conv* [*iff*]: (*0* < *length xs*) = (*xs* ≠ [])
⟨*proof*⟩

**lemma** *length-pos-if-in-set*: *x* ∈ *set xs* ⟹ *length xs* > *0*
⟨*proof*⟩

**lemma** *length-Suc-conv*: (*length xs* = *Suc n*) = (∃ *y ys*. *xs* = *y* # *ys* ∧ *length ys* = *n*)
⟨*proof*⟩

**lemmas** *Suc-length-conv* = *length-Suc-conv*[*THEN eq-iff-swap*]

**lemma** *Suc-le-length-iff*:
(*Suc n* ≤ *length xs*) = (∃ *x ys*. *xs* = *x* # *ys* ∧ *n* ≤ *length ys*)
⟨*proof*⟩

**lemma** *impossible-Cons*: *length xs* ≤ *length ys* ⟹ *xs* = *x* # *ys* = *False*
⟨*proof*⟩

**lemma** *list-induct2* [*consumes 1*, *case-names Nil Cons*]:
*length xs* = *length ys* ⟹ *P* [] [] ⟹
(⋀*x xs y ys*. *length xs* = *length ys* ⟹ *P xs ys* ⟹ *P* (*x*#*xs*) (*y*#*ys*))
⟹ *P xs ys*
⟨*proof*⟩

**lemma** *list-induct3* [*consumes 2*, *case-names Nil Cons*]:
*length xs* = *length ys* ⟹ *length ys* = *length zs* ⟹ *P* [] [] [] ⟹
(⋀*x xs y ys z zs*. *length xs* = *length ys* ⟹ *length ys* = *length zs* ⟹ *P xs ys zs* ⟹ *P* (*x*#*xs*) (*y*#*ys*) (*z*#*zs*))
⟹ *P xs ys zs*
⟨*proof*⟩

**lemma** *list-induct4* [*consumes 3*, *case-names Nil Cons*]:
*length xs* = *length ys* ⟹ *length ys* = *length zs* ⟹ *length zs* = *length ws* ⟹
*P* [] [] [] [] ⟹ (⋀*x xs y ys z zs w ws*. *length xs* = *length ys* ⟹
*length ys* = *length zs* ⟹ *length zs* = *length ws* ⟹ *P xs ys zs ws* ⟹
*P* (*x*#*xs*) (*y*#*ys*) (*z*#*zs*) (*w*#*ws*)) ⟹ *P xs ys zs ws*
⟨*proof*⟩

**lemma** *list-induct2'*:

$\llbracket P\ []\ [];$
$\bigwedge x\ xs.\ P\ (x\#xs)\ [];$
$\bigwedge y\ ys.\ P\ []\ (y\#ys);$
$\bigwedge x\ xs\ y\ ys.\ P\ xs\ ys \implies P\ (x\#xs)\ (y\#ys)\ \rrbracket$
$\implies P\ xs\ ys$
⟨*proof*⟩

**lemma** *list-all2-iff*:
$list\text{-}all2\ P\ xs\ ys \longleftrightarrow length\ xs = length\ ys \wedge (\forall (x,\ y) \in set\ (zip\ xs\ ys).\ P\ x\ y)$
⟨*proof*⟩

**lemma** *neq-if-length-neq*: $length\ xs \neq length\ ys \implies (xs = ys) == False$
⟨*proof*⟩

### 66.1.4 @ – append

**global-interpretation** *append*: *monoid append Nil*
⟨*proof*⟩

**lemma** *append-assoc* [*simp*]: $(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
⟨*proof*⟩

**lemma** *append-Nil2*: $xs\ @\ [] = xs$
⟨*proof*⟩

**lemma** *append-is-Nil-conv* [*iff*]: $(xs\ @\ ys = []) = (xs = [] \wedge ys = [])$
⟨*proof*⟩

**lemmas** *Nil-is-append-conv* [*iff*] = *append-is-Nil-conv*[*THEN eq-iff-swap*]

**lemma** *append-self-conv* [*iff*]: $(xs\ @\ ys = xs) = (ys = [])$
⟨*proof*⟩

**lemmas** *self-append-conv* [*iff*] = *append-self-conv*[*THEN eq-iff-swap*]

**lemma** *append-eq-append-conv* [*simp*]:
$length\ xs = length\ ys \vee length\ us = length\ vs$
$\implies (xs@us = ys@vs) = (xs{=}ys \wedge us{=}vs)$
⟨*proof*⟩

**lemma** *append-eq-append-conv2*: $(xs\ @\ ys = zs\ @\ ts) =$
$(\exists us.\ xs = zs\ @\ us \wedge us\ @\ ys = ts \vee xs\ @\ us = zs \wedge ys = us\ @\ ts)$
⟨*proof*⟩

**lemma** *same-append-eq* [*iff*, *induct-simp*]: $(xs\ @\ ys = xs\ @\ zs) = (ys = zs)$
⟨*proof*⟩

**lemma** *append1-eq-conv* [*iff*]: $(xs\ @\ [x] = ys\ @\ [y]) = (xs = ys \wedge x = y)$
⟨*proof*⟩

**lemma** *append-same-eq* [*iff*, *induct-simp*]: $(ys @ xs = zs @ xs) = (ys = zs)$
⟨*proof*⟩

**lemma** *append-self-conv2* [*iff*]: $(xs @ ys = ys) = (xs = [])$
⟨*proof*⟩

**lemmas** *self-append-conv2* [*iff*] = *append-self-conv2*[*THEN eq-iff-swap*]

**lemma** *hd-Cons-tl*: $xs \neq [] \Longrightarrow hd\ xs\ \#\ tl\ xs = xs$
⟨*proof*⟩

**lemma** *hd-append*: $hd\ (xs @ ys) = (if\ xs = []\ then\ hd\ ys\ else\ hd\ xs)$
⟨*proof*⟩

**lemma** *hd-append2* [*simp*]: $xs \neq [] \Longrightarrow hd\ (xs @ ys) = hd\ xs$
⟨*proof*⟩

**lemma** *tl-append*: $tl\ (xs @ ys) = (case\ xs\ of\ [] \Rightarrow tl\ ys \mid z\#zs \Rightarrow zs @ ys)$
⟨*proof*⟩

**lemma** *tl-append2* [*simp*]: $xs \neq [] \Longrightarrow tl\ (xs @ ys) = tl\ xs @ ys$
⟨*proof*⟩

**lemma** *tl-append-if*: $tl\ (xs @ ys) = (if\ xs = []\ then\ tl\ ys\ else\ tl\ xs @ ys)$
⟨*proof*⟩

**lemma** *Cons-eq-append-conv*: $x\#xs = ys@zs =$
$(ys = [] \wedge x\#xs = zs \vee (\exists\, ys'.\ x\#ys' = ys \wedge xs = ys'@zs))$
⟨*proof*⟩

**lemma** *append-eq-Cons-conv*: $(ys@zs = x\#xs) =$
$(ys = [] \wedge zs = x\#xs \vee (\exists\, ys'.\ ys = x\#ys' \wedge ys'@zs = xs))$
⟨*proof*⟩

**lemma** *longest-common-prefix*:
$\exists\, ps\ xs'\ ys'.\ xs = ps @ xs' \wedge ys = ps @ ys'$
$\wedge (xs' = [] \vee ys' = [] \vee hd\ xs' \neq hd\ ys')$
⟨*proof*⟩

Trivial rules for solving @-equations automatically.

**lemma** *eq-Nil-appendI*: $xs = ys \Longrightarrow xs = [] @ ys$
⟨*proof*⟩

**lemma** *Cons-eq-appendI*: $[\![x\ \#\ xs1 = ys;\ xs = xs1 @ zs]\!] \Longrightarrow x\ \#\ xs = ys @ zs$
⟨*proof*⟩

**lemma** *append-eq-appendI*: $[\![xs @ xs1 = zs;\ ys = xs1 @ us]\!] \Longrightarrow xs @ ys = zs @ us$

⟨proof⟩

Simplification procedure for all list equalities. Currently only tries to rearrange @ to see if - both lists end in a singleton list, - or both lists end in the same list.

⟨ML⟩

### 66.1.5 *map*

**lemma** *hd-map*: $xs \neq [] \Longrightarrow hd\ (map\ f\ xs) = f\ (hd\ xs)$
⟨proof⟩

**lemma** *map-tl*: $map\ f\ (tl\ xs) = tl\ (map\ f\ xs)$
⟨proof⟩

**lemma** *map-ext*: $(\bigwedge x.\ x \in set\ xs \longrightarrow f\ x = g\ x) \Longrightarrow map\ f\ xs = map\ g\ xs$
⟨proof⟩

**lemma** *map-ident* [*simp*]: $map\ (\lambda x.\ x) = (\lambda xs.\ xs)$
⟨proof⟩

**lemma** *map-append* [*simp*]: $map\ f\ (xs\ @\ ys) = map\ f\ xs\ @\ map\ f\ ys$
⟨proof⟩

**lemma** *map-map* [*simp*]: $map\ f\ (map\ g\ xs) = map\ (f \circ g)\ xs$
⟨proof⟩

**lemma** *map-comp-map*[*simp*]: $((map\ f) \circ (map\ g)) = map(f \circ g)$
⟨proof⟩

**lemma** *rev-map*: $rev\ (map\ f\ xs) = map\ f\ (rev\ xs)$
⟨proof⟩

**lemma** *map-eq-conv*[*simp*]: $(map\ f\ xs = map\ g\ xs) = (\forall\, x \in set\ xs.\ f\ x = g\ x)$
⟨proof⟩

**lemma** *map-cong* [*fundef-cong*]:
$xs = ys \Longrightarrow (\bigwedge x.\ x \in set\ ys \Longrightarrow f\ x = g\ x) \Longrightarrow map\ f\ xs = map\ g\ ys$
⟨proof⟩

**lemma** *map-is-Nil-conv* [*iff*]: $(map\ f\ xs = []) = (xs = [])$
⟨proof⟩

**lemmas** *Nil-is-map-conv* [*iff*] = *map-is-Nil-conv*[*THEN eq-iff-swap*]

**lemma** *map-eq-Cons-conv*:
$(map\ f\ xs = y\#ys) = (\exists\, z\ zs.\ xs = z\#zs \land f\ z = y \land map\ f\ zs = ys)$
⟨proof⟩

**lemma** *Cons-eq-map-conv*:
$(x\#xs = map\ f\ ys) = (\exists\, z\ zs.\ ys = z\#zs \wedge x = f\ z \wedge xs = map\ f\ zs)$
⟨*proof*⟩

**lemmas** *map-eq-Cons-D* = *map-eq-Cons-conv* [*THEN iffD1*]
**lemmas** *Cons-eq-map-D* = *Cons-eq-map-conv* [*THEN iffD1*]
**declare** *map-eq-Cons-D* [*dest*!] *Cons-eq-map-D* [*dest*!]

**lemma** *ex-map-conv*:
$(\exists\, xs.\ ys = map\ f\ xs) = (\forall\, y \in set\ ys.\ \exists\, x.\ y = f\ x)$
⟨*proof*⟩

**lemma** *map-eq-imp-length-eq*:
**assumes** $map\ f\ xs = map\ g\ ys$
**shows** $length\ xs = length\ ys$
⟨*proof*⟩

**lemma** *map-inj-on*:
**assumes** *map*: $map\ f\ xs = map\ f\ ys$ **and** *inj*: $inj\text{-}on\ f\ (set\ xs\ Un\ set\ ys)$
**shows** $xs = ys$
⟨*proof*⟩

**lemma** *inj-on-map-eq-map*:
$inj\text{-}on\ f\ (set\ xs\ Un\ set\ ys) \Longrightarrow (map\ f\ xs = map\ f\ ys) = (xs = ys)$
⟨*proof*⟩

**lemma** *map-injective*:
$map\ f\ xs = map\ f\ ys \Longrightarrow inj\ f \Longrightarrow xs = ys$
⟨*proof*⟩

**lemma** *inj-map-eq-map*[*simp*]: $inj\ f \Longrightarrow (map\ f\ xs = map\ f\ ys) = (xs = ys)$
⟨*proof*⟩

**lemma** *inj-mapI*: $inj\ f \Longrightarrow inj\ (map\ f)$
⟨*proof*⟩

**lemma** *inj-mapD*: $inj\ (map\ f) \Longrightarrow inj\ f$
⟨*proof*⟩

**lemma** *inj-map*[*iff*]: $inj\ (map\ f) = inj\ f$
⟨*proof*⟩

**lemma** *inj-on-mapI*: $inj\text{-}on\ f\ (\bigcup(set\ `\ A)) \Longrightarrow inj\text{-}on\ (map\ f)\ A$
⟨*proof*⟩

**lemma** *map-idI*: $(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x = x) \Longrightarrow map\ f\ xs = xs$
⟨*proof*⟩

**lemma** *map-fun-upd* [*simp*]: $y \notin set\ xs \Longrightarrow map\ (f(y{:=}v))\ xs = map\ f\ xs$

⟨*proof*⟩

**lemma** *map-fst-zip*[*simp*]:
$length\ xs = length\ ys \Longrightarrow map\ fst\ (zip\ xs\ ys) = xs$
⟨*proof*⟩

**lemma** *map-snd-zip*[*simp*]:
$length\ xs = length\ ys \Longrightarrow map\ snd\ (zip\ xs\ ys) = ys$
⟨*proof*⟩

**lemma** *map-fst-zip-take*:
$map\ fst\ (zip\ xs\ ys) = take\ (min\ (length\ xs)\ (length\ ys))\ xs$
⟨*proof*⟩

**lemma** *map-snd-zip-take*:
$map\ snd\ (zip\ xs\ ys) = take\ (min\ (length\ xs)\ (length\ ys))\ ys$
⟨*proof*⟩

**lemma** *map2-map-map*: $map2\ h\ (map\ f\ xs)\ (map\ g\ xs) = map\ (\lambda x.\ h\ (f\ x)\ (g\ x))\ xs$
⟨*proof*⟩

**functor** *map*: *map*
⟨*proof*⟩

**declare** *map.id* [*simp*]

### 66.1.6 *rev*

**lemma** *rev-append* [*simp*]: $rev\ (xs\ @\ ys) = rev\ ys\ @\ rev\ xs$
⟨*proof*⟩

**lemma** *rev-rev-ident* [*simp*]: $rev\ (rev\ xs) = xs$
⟨*proof*⟩

**lemma** *rev-involution*[*simp*]: $rev \circ rev = id$
⟨*proof*⟩

**lemma** *rev-swap*: $(rev\ xs = ys) = (xs = rev\ ys)$
⟨*proof*⟩

**lemma** *rev-is-Nil-conv* [*iff*]: $(rev\ xs = []) = (xs = [])$
⟨*proof*⟩

**lemmas** *Nil-is-rev-conv* [*iff*] = *rev-is-Nil-conv*[*THEN eq-iff-swap*]

**lemma** *rev-singleton-conv* [*simp*]: $(rev\ xs = [x]) = (xs = [x])$
⟨*proof*⟩

**lemma** *singleton-rev-conv* [*simp*]: $([x] = rev\ xs) = ([x] = xs)$
⟨*proof*⟩

**lemma** *rev-is-rev-conv* [*iff*]: $(rev\ xs = rev\ ys) = (xs = ys)$
⟨*proof*⟩

**lemma** *rev-eq-append-conv*: $rev\ xs = ys\ @\ zs \longleftrightarrow xs = rev\ zs\ @\ rev\ ys$
⟨*proof*⟩

**lemma** *append-eq-rev-conv*: $ys\ @\ zs = rev\ xs \longleftrightarrow rev\ zs\ @\ rev\ ys = xs$
⟨*proof*⟩

**lemma** *rev-eq-Cons-iff*[*iff*]: $(rev\ xs = y\#ys) = (xs = rev\ ys\ @\ [y])$
⟨*proof*⟩

**lemmas** *Cons-eq-rev-iff* = *rev-eq-Cons-iff*[*THEN eq-iff-swap*]

**lemma** *inj-on-rev*[*iff*]: *inj-on rev A*
⟨*proof*⟩

**lemma** *rev-induct* [*case-names Nil snoc*]:
  **assumes** $P\ []$ **and** $\bigwedge x\ xs.\ P\ xs \Longrightarrow P\ (xs\ @\ [x])$
  **shows** $P\ xs$
⟨*proof*⟩

**lemma** *rev-exhaust* [*case-names Nil snoc*]:
  $(xs = [] \Longrightarrow P) \Longrightarrow (\bigwedge ys\ y.\ xs = ys\ @\ [y] \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemmas** *rev-cases* = *rev-exhaust*

**lemma** *rev-nonempty-induct* [*consumes 1*, *case-names single snoc*]:
  **assumes** $xs \neq []$
  **and** *single*: $\bigwedge x.\ P\ [x]$
  **and** *snoc'*: $\bigwedge x\ xs.\ xs \neq [] \Longrightarrow P\ xs \Longrightarrow P\ (xs@[x])$
  **shows** $P\ xs$
⟨*proof*⟩

**lemma** *rev-induct2*:
  $[\![\ P\ []\ [];$
  $\bigwedge x\ xs.\ P\ (xs\ @\ [x])\ [];$
  $\bigwedge y\ ys.\ P\ []\ (ys\ @\ [y]);$
  $\bigwedge x\ xs\ y\ ys.\ P\ xs\ ys \Longrightarrow P\ (xs\ @\ [x])\ (ys\ @\ [y])\ ]\!]$
 $\Longrightarrow P\ xs\ ys$
⟨*proof*⟩

**lemma** *length-Suc-conv-rev*: $(length\ xs = Suc\ n) = (\exists\ y\ ys.\ xs = ys\ @\ [y] \wedge length\ ys = n)$
⟨*proof*⟩

### 66.1.7 *set*

**declare** *list.set*[*code-post*] — pretty output

**lemma** *finite-set* [*iff*]: *finite* (*set xs*)
⟨*proof*⟩

**lemma** *set-append* [*simp*]: $set\ (xs\ @\ ys) = (set\ xs \cup set\ ys)$
⟨*proof*⟩

**lemma** *hd-in-set*[*simp*]: $xs \neq [] \Longrightarrow hd\ xs \in set\ xs$
⟨*proof*⟩

**lemma** *set-subset-Cons*: $set\ xs \subseteq set\ (x\ \#\ xs)$
⟨*proof*⟩

**lemma** *set-ConsD*: $y \in set\ (x\ \#\ xs) \Longrightarrow y{=}x \vee y \in set\ xs$
⟨*proof*⟩

**lemma** *set-empty* [*iff*]: $(set\ xs = \{\}) = (xs = [])$
⟨*proof*⟩

**lemmas** *set-empty2*[*iff*] = *set-empty*[*THEN eq-iff-swap*]

**lemma** *append-eq-append-conv-if-disj*:
$(set\ xs \cup set\ xs') \cap (set\ ys \cup set\ ys') = \{\}$
$\Longrightarrow\ xs\ @\ ys = xs'\ @\ ys' \longleftrightarrow xs = xs' \wedge ys = ys'$
⟨*proof*⟩

**lemma** *set-rev* [*simp*]: $set\ (rev\ xs) = set\ xs$
⟨*proof*⟩

**lemma** *set-map* [*simp*]: $set\ (map\ f\ xs) = f\,`(set\ xs)$
⟨*proof*⟩

**lemma** *set-filter* [*simp*]: $set\ (filter\ P\ xs) = \{x.\ x \in set\ xs \wedge P\ x\}$
⟨*proof*⟩

**lemma** *set-upt* [*simp*]: $set[i..<j] = \{i..<j\}$
⟨*proof*⟩

**lemma** *atMost-upto*:
‹$\{..n\} = set\ [0..<Suc\ n]$›
⟨*proof*⟩

**lemma** *atLeast-upt*:
‹$\{..<n\} = set\ [0..<n]$›
⟨*proof*⟩

**lemma** *greaterThanLessThan-upt*:

$\langle\{n{<}..{<}m\} = set\ [Suc\ n..{<}m]\rangle$
$\langle proof \rangle$

**lemma** *atLeastLessThan-upt*:
$\langle\{i..{<}j\} = set\ [i..{<}j]\rangle$
$\langle proof \rangle$

**lemma** *greaterThanAtMost-upt*:
$\{n{<}..m\} = set\ [Suc\ n..{<}Suc\ m]$
$\langle proof \rangle$

**lemma** *atLeastAtMost-upt*:
$\{n..m\} = set\ [n..{<}Suc\ m]$
$\langle proof \rangle$

**lemma** *split-list*: $x \in set\ xs \Longrightarrow \exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs$
$\langle proof \rangle$

**lemma** *in-set-conv-decomp*: $x \in set\ xs \longleftrightarrow (\exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs)$
$\langle proof \rangle$

**lemma** *split-list-first*: $x \in set\ xs \Longrightarrow \exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge x \notin set\ ys$
$\langle proof \rangle$

**lemma** *in-set-conv-decomp-first*:
$(x \in set\ xs) = (\exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge x \notin set\ ys)$
$\langle proof \rangle$

**lemma** *split-list-last*: $x \in set\ xs \Longrightarrow \exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge x \notin set\ zs$
$\langle proof \rangle$

**lemma** *in-set-conv-decomp-last*:
$(x \in set\ xs) = (\exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge x \notin set\ zs)$
$\langle proof \rangle$

**lemma** *split-list-prop*: $\exists\, x \in set\ xs.\ P\ x \Longrightarrow \exists\, ys\ x\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge P\ x$
$\langle proof \rangle$

**lemma** *split-list-propE*:
**assumes** $\exists\, x \in set\ xs.\ P\ x$
**obtains** $ys\ x\ zs$ **where** $xs = ys\ @\ x\ \#\ zs$ **and** $P\ x$
$\langle proof \rangle$

**lemma** *split-list-first-prop*:
$\exists\, x \in set\ xs.\ P\ x \Longrightarrow$
$\exists\, ys\ x\ zs.\ xs = ys@x\#zs \wedge P\ x \wedge (\forall\, y \in set\ ys.\ \neg\ P\ y)$
$\langle proof \rangle$

**lemma** *split-list-first-propE*:

**assumes** $\exists\, x \in set\ xs.\ P\ x$
**obtains** $ys\ x\ zs$ **where** $xs = ys\ @\ x\ \#\ zs$ **and** $P\ x$ **and** $\forall\, y \in set\ ys.\ \neg\ P\ y$
⟨*proof*⟩

**lemma** *split-list-first-prop-iff*:
$(\exists\, x \in set\ xs.\ P\ x) \longleftrightarrow$
$(\exists\, ys\ x\ zs.\ xs = ys@x\#zs \wedge P\ x \wedge (\forall\, y \in set\ ys.\ \neg\ P\ y))$
⟨*proof*⟩

**lemma** *split-list-last-prop*:
$\exists\, x \in set\ xs.\ P\ x \Longrightarrow$
$\exists\, ys\ x\ zs.\ xs = ys@x\#zs \wedge P\ x \wedge (\forall\, z \in set\ zs.\ \neg\ P\ z)$
⟨*proof*⟩

**lemma** *split-list-last-propE*:
**assumes** $\exists\, x \in set\ xs.\ P\ x$
**obtains** $ys\ x\ zs$ **where** $xs = ys\ @\ x\ \#\ zs$ **and** $P\ x$ **and** $\forall\, z \in set\ zs.\ \neg\ P\ z$
⟨*proof*⟩

**lemma** *split-list-last-prop-iff*:
$(\exists\, x \in set\ xs.\ P\ x) \longleftrightarrow$
$(\exists\, ys\ x\ zs.\ xs = ys@x\#zs \wedge P\ x \wedge (\forall\, z \in set\ zs.\ \neg\ P\ z))$
⟨*proof*⟩

**lemma** *finite-list*: $finite\ A \Longrightarrow \exists\, xs.\ set\ xs = A$
⟨*proof*⟩

**lemma** *card-length*: $card\ (set\ xs) \le length\ xs$
⟨*proof*⟩

**lemma** *set-minus-filter-out*:
$set\ xs - \{y\} = set\ (filter\ (\lambda x.\ \neg\ (x = y))\ xs)$
⟨*proof*⟩

**lemma** *append-Cons-eq-iff*:
$[\![\ x \notin set\ xs;\ x \notin set\ ys\ ]\!] \Longrightarrow$
$xs\ @\ x\ \#\ ys = xs'\ @\ x\ \#\ ys' \longleftrightarrow (xs = xs' \wedge ys = ys')$
⟨*proof*⟩

### 66.1.8 *concat*

**lemma** *concat-append* [*simp*]: $concat\ (xs\ @\ ys) = concat\ xs\ @\ concat\ ys$
⟨*proof*⟩

**lemma** *concat-eq-Nil-conv* [*simp*]: $(concat\ xss = []) = (\forall\, xs \in set\ xss.\ xs = [])$
⟨*proof*⟩

**lemmas** *Nil-eq-concat-conv* [*simp*] = *concat-eq-Nil-conv*[*THEN eq-iff-swap*]

**lemma** *set-concat* [*simp*]: $set\ (concat\ xs) = (\bigcup x \in set\ xs.\ set\ x)$
⟨*proof*⟩

**lemma** *concat-map-singleton*[*simp*]: $concat(map\ (\%x.\ [f\ x])\ xs) = map\ f\ xs$
⟨*proof*⟩

**lemma** *map-concat*: $map\ f\ (concat\ xs) = concat\ (map\ (map\ f)\ xs)$
⟨*proof*⟩

**lemma** *rev-concat*: $rev\ (concat\ xs) = concat\ (map\ rev\ (rev\ xs))$
⟨*proof*⟩

**lemma** *length-concat-rev*[*simp*]: $length\ (concat\ (rev\ xs)) = length\ (concat\ xs)$
⟨*proof*⟩

**lemma** *concat-eq-concat-iff*: $\forall (x,\ y) \in set\ (zip\ xs\ ys).\ length\ x = length\ y \Longrightarrow length\ xs = length\ ys \Longrightarrow (concat\ xs = concat\ ys) = (xs = ys)$
⟨*proof*⟩

**lemma** *concat-injective*: $concat\ xs = concat\ ys \Longrightarrow length\ xs = length\ ys \Longrightarrow \forall (x,\ y) \in set\ (zip\ xs\ ys).\ length\ x = length\ y \Longrightarrow xs = ys$
⟨*proof*⟩

**lemma** *concat-eq-appendD*:
**assumes** $concat\ xss = ys\ @\ zs\ xss \neq []$
**shows** $\exists xss1\ xs\ xs'\ xss2.\ xss = xss1\ @\ (xs\ @\ xs')\ \#\ xss2 \wedge ys = concat\ xss1\ @\ xs \wedge zs = xs'\ @\ concat\ xss2$
⟨*proof*⟩

**lemma** *concat-eq-append-conv*:
$concat\ xss = ys\ @\ zs \longleftrightarrow$
$(if\ xss = []\ then\ ys = [] \wedge zs = []$
$else\ \exists xss1\ xs\ xs'\ xss2.\ xss = xss1\ @\ (xs\ @\ xs')\ \#\ xss2 \wedge ys = concat\ xss1\ @\ xs \wedge zs = xs'\ @\ concat\ xss2)$
⟨*proof*⟩

**lemma** *hd-concat*: $[\![xs \neq [];\ hd\ xs \neq []]\!] \Longrightarrow hd\ (concat\ xs) = hd\ (hd\ xs)$
⟨*proof*⟩

⟨*ML*⟩

### 66.1.9 *filter*

**lemma** *filter-append* [*simp*]: $filter\ P\ (xs\ @\ ys) = filter\ P\ xs\ @\ filter\ P\ ys$
⟨*proof*⟩

**lemma** *rev-filter*: $rev\ (filter\ P\ xs) = filter\ P\ (rev\ xs)$

⟨proof⟩

**lemma** *filter-filter* [*simp*]: $filter\ P\ (filter\ Q\ xs) = filter\ (\lambda x.\ Q\ x \wedge P\ x)\ xs$
⟨proof⟩

**lemma** *filter-concat*: $filter\ p\ (concat\ xs) = concat\ (map\ (filter\ p)\ xs)$
⟨proof⟩

**lemma** *length-filter-le* [*simp*]: $length\ (filter\ P\ xs) \leq length\ xs$
⟨proof⟩

**lemma** *sum-length-filter-compl*:
$length(filter\ P\ xs) + length(filter\ (\lambda x.\ \neg P\ x)\ xs) = length\ xs$
⟨proof⟩

**lemma** *filter-True* [*simp*]: $\forall x \in set\ xs.\ P\ x \Longrightarrow filter\ P\ xs = xs$
⟨proof⟩

**lemma** *filter-False* [*simp*]: $\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow filter\ P\ xs = []$
⟨proof⟩

**lemma** *filter-empty-conv*: $(filter\ P\ xs = []) = (\forall x{\in}set\ xs.\ \neg\ P\ x)$
⟨proof⟩

**lemmas** *empty-filter-conv* = *filter-empty-conv*[*THEN eq-iff-swap*]

**lemma** *filter-id-conv*: $(filter\ P\ xs = xs) = (\forall x{\in}set\ xs.\ P\ x)$
⟨proof⟩

**lemma** *filter-map*: $filter\ P\ (map\ f\ xs) = map\ f\ (filter\ (P \circ f)\ xs)$
⟨proof⟩

**lemma** *length-filter-map*[*simp*]:
$length\ (filter\ P\ (map\ f\ xs)) = length(filter\ (P \circ f)\ xs)$
⟨proof⟩

**lemma** *filter-is-subset* [*simp*]: $set\ (filter\ P\ xs) \leq set\ xs$
⟨proof⟩

**lemma** *length-filter-less*:
$[\![\ x \in set\ xs;\ \neg\ P\ x\ ]\!] \Longrightarrow length(filter\ P\ xs) < length\ xs$
⟨proof⟩

**lemma** *length-filter-conv-card*:
$length(filter\ p\ xs) = card\{i.\ i < length\ xs \wedge p(xs!i)\}$
⟨proof⟩

**lemma** *Cons-eq-filterD*:
$x\#xs = filter\ P\ ys \Longrightarrow$

$\exists$ *us vs. ys* = *us* @ *x* # *vs* $\wedge$ ($\forall$ *u*$\in$*set us.* $\neg$ *P u*) $\wedge$ *P x* $\wedge$ *xs* = *filter P vs*
(**is** - $\Longrightarrow$ $\exists$ *us vs. ?P ys us vs*)
⟨*proof*⟩

**lemma** *filter-eq-ConsD*:
*filter P ys* = *x#xs* $\Longrightarrow$
$\exists$ *us vs. ys* = *us* @ *x* # *vs* $\wedge$ ($\forall$ *u*$\in$*set us.* $\neg$ *P u*) $\wedge$ *P x* $\wedge$ *xs* = *filter P vs*
⟨*proof*⟩

**lemma** *filter-eq-Cons-iff*:
(*filter P ys* = *x#xs*) =
($\exists$ *us vs. ys* = *us* @ *x* # *vs* $\wedge$ ($\forall$ *u*$\in$*set us.* $\neg$ *P u*) $\wedge$ *P x* $\wedge$ *xs* = *filter P vs*)
⟨*proof*⟩

**lemmas** *Cons-eq-filter-iff* = *filter-eq-Cons-iff*[*THEN eq-iff-swap*]

**lemma** *inj-on-filter-key-eq*:
**assumes** *inj-on f* (*insert y* (*set xs*))
**shows** *filter* ($\lambda$*x. f y* = *f x*) *xs* = *filter* (*HOL.eq y*) *xs*
⟨*proof*⟩

**lemma** *filter-cong*[*fundef-cong*]:
*xs* = *ys* $\Longrightarrow$ ($\bigwedge$*x. x* $\in$ *set ys* $\Longrightarrow$ *P x* = *Q x*) $\Longrightarrow$ *filter P xs* = *filter Q ys*
⟨*proof*⟩

### 66.1.10 List partitioning

**primrec** *partition* :: (*'a* $\Rightarrow$ *bool*) $\Rightarrow$*'a list* $\Rightarrow$ *'a list* $\times$ *'a list* **where**
*partition P* [] = ([], []) |
*partition P* (*x* # *xs*) =
(*let* (*yes*, *no*) = *partition P xs*
*in if P x then* (*x* # *yes*, *no*) *else* (*yes*, *x* # *no*))

**lemma** *partition-filter1*: *fst* (*partition P xs*) = *filter P xs*
⟨*proof*⟩

**lemma** *partition-filter2*: *snd* (*partition P xs*) = *filter* (*Not* $\circ$ *P*) *xs*
⟨*proof*⟩

**lemma** *partition-P*:
**assumes** *partition P xs* = (*yes*, *no*)
**shows** ($\forall$ *p* $\in$ *set yes.* *P p*) $\wedge$ ($\forall$ *p* $\in$ *set no.* $\neg$ *P p*)
⟨*proof*⟩

**lemma** *partition-set*:
**assumes** *partition P xs* = (*yes*, *no*)
**shows** *set yes* $\cup$ *set no* = *set xs*
⟨*proof*⟩

**lemma** *partition-filter-conv*[*simp*]:
$partition\ f\ xs = (filter\ f\ xs, filter\ (Not \circ f)\ xs)$
⟨*proof*⟩

**declare** *partition.simps*[*simp del*]

### 66.1.11 (!)

**lemma** *nth-Cons-0* [*simp*, *code*]: $(x \# xs)!0 = x$
⟨*proof*⟩

**lemma** *nth-Cons-Suc* [*simp*, *code*]: $(x \# xs)!(Suc\ n) = xs!n$
⟨*proof*⟩

**declare** *nth.simps* [*simp del*]

**lemma** *nth-Cons-pos*[*simp*]: $0 < n \Longrightarrow (x\#xs) ! n = xs ! (n - 1)$
⟨*proof*⟩

**lemma** *nth-append*:
$(xs @ ys)!n = (if\ n < length\ xs\ then\ xs!n\ else\ ys!(n - length\ xs))$
⟨*proof*⟩

**lemma** *nth-append-left*: $i < length\ xs \Longrightarrow (xs @ ys) ! i = xs ! i$
⟨*proof*⟩

**lemma** *nth-append-right*: $i \geq length\ xs \Longrightarrow (xs @ ys) ! i = ys ! (i - length\ xs)$
⟨*proof*⟩

**lemma** *nth-append-length* [*simp*]: $(xs @ x \# ys) ! length\ xs = x$
⟨*proof*⟩

**lemma** *nth-append-length-plus*[*simp*]: $(xs @ ys) ! (length\ xs + n) = ys ! n$
⟨*proof*⟩

**lemma** *nth-map* [*simp*]: $n < length\ xs \Longrightarrow (map\ f\ xs)!n = f(xs!n)$
⟨*proof*⟩

**lemma** *nth-tl*: $n < length\ (tl\ xs) \Longrightarrow tl\ xs ! n = xs ! Suc\ n$
⟨*proof*⟩

**lemma** *hd-conv-nth*: $xs \neq [] \Longrightarrow hd\ xs = xs!0$
⟨*proof*⟩

**lemma** *list-eq-iff-nth-eq*:
$(xs = ys) = (length\ xs = length\ ys \wedge (\forall i<length\ xs.\ xs!i = ys!i))$
⟨*proof*⟩

**lemma** *map-equality-iff*:

$map\ f\ xs = map\ g\ ys \longleftrightarrow length\ xs = length\ ys \wedge (\forall\, i{<}length\ ys.\ f\ (xs!i) = g\ (ys!i))$
⟨proof⟩

**lemma** *set-conv-nth*: $set\ xs = \{xs!i \mid i.\ i < length\ xs\}$
⟨proof⟩

**lemma** *in-set-conv-nth*: $(x \in set\ xs) = (\exists\, i < length\ xs.\ xs!i = x)$
⟨proof⟩

**lemma** *nth-equal-first-eq*:
**assumes** $x \notin set\ xs$
**assumes** $n \leq length\ xs$
**shows** $(x \ \#\ xs)\ !\ n = x \longleftrightarrow n = 0$ (**is** $?lhs \longleftrightarrow ?rhs$)
⟨proof⟩

**lemma** *nth-non-equal-first-eq*:
**assumes** $x \neq y$
**shows** $(x \ \#\ xs)\ !\ n = y \longleftrightarrow xs\ !\ (n - 1) = y \wedge n > 0$ (**is** $?lhs \longleftrightarrow ?rhs$)
⟨proof⟩

**lemma** *list-ball-nth*: $[\![n < length\ xs; \forall\, x \in set\ xs.\ P\ x]\!] \Longrightarrow P(xs!n)$
⟨proof⟩

**lemma** *nth-mem* [*simp*]: $n < length\ xs \Longrightarrow xs!n \in set\ xs$
⟨proof⟩

**lemma** *all-nth-imp-all-set*:
$[\![\forall\, i < length\ xs.\ P(xs!i); x \in set\ xs]\!] \Longrightarrow P\ x$
⟨proof⟩

**lemma** *all-set-conv-all-nth*:
$(\forall\, x \in set\ xs.\ P\ x) = (\forall\, i.\ i < length\ xs \longrightarrow P\ (xs\ !\ i))$
⟨proof⟩

**lemma** *rev-nth*:
$n < size\ xs \Longrightarrow rev\ xs\ !\ n = xs\ !\ (length\ xs - Suc\ n)$
⟨proof⟩

**lemma** *Skolem-list-nth*:
$(\forall\, i{<}k.\ \exists\, x.\ P\ i\ x) = (\exists\, xs.\ size\ xs = k \wedge (\forall\, i{<}k.\ P\ i\ (xs!i)))$
(**is** $\text{-} = (\exists\, xs.\ ?P\ k\ xs)$)
⟨proof⟩

### 66.1.12 *list-update*

**lemma** *length-list-update* [*simp*]: $length(xs[i{:=}x]) = length\ xs$
⟨proof⟩

**lemma** *nth-list-update*:
$i < length\ xs \Longrightarrow (xs[i:=x])!j = (if\ i = j\ then\ x\ else\ xs!j)$
⟨*proof*⟩

**lemma** *nth-list-update-eq* [*simp*]: $i < length\ xs \Longrightarrow (xs[i:=x])!i = x$
⟨*proof*⟩

**lemma** *nth-list-update-neq* [*simp*]: $i \neq j \Longrightarrow xs[i:=x]!j = xs!j$
⟨*proof*⟩

**lemma** *list-update-id*[*simp*]: $xs[i := xs!i] = xs$
⟨*proof*⟩

**lemma** *list-update-beyond*: $length\ xs \leq i \Longrightarrow xs[i:=x] = xs$
⟨*proof*⟩

**lemma** *list-update-nonempty*[*simp*]: $xs[k:=x] = [] \longleftrightarrow xs=[]$
⟨*proof*⟩

**lemma** *list-update-same-conv*:
$i < length\ xs \Longrightarrow (xs[i := x] = xs) = (xs!i = x)$
⟨*proof*⟩

**lemma** *list-update-append1*:
$i < size\ xs \Longrightarrow (xs\ @\ ys)[i:=x] = xs[i:=x]\ @\ ys$
⟨*proof*⟩

**lemma** *list-update-append*:
$(xs\ @\ ys)\ [n:= x] =$
$(if\ n < length\ xs\ then\ xs[n:= x]\ @\ ys\ else\ xs\ @\ (ys\ [n-length\ xs:= x]))$
⟨*proof*⟩

**lemma** *list-update-length* [*simp*]:
$(xs\ @\ x\ \#\ ys)[length\ xs := y] = (xs\ @\ y\ \#\ ys)$
⟨*proof*⟩

**lemma** *map-update*: $map\ f\ (xs[k:= y]) = (map\ f\ xs)[k := f\ y]$
⟨*proof*⟩

**lemma** *rev-update*:
$k < length\ xs \Longrightarrow rev\ (xs[k:= y]) = (rev\ xs)[length\ xs - k - 1 := y]$
⟨*proof*⟩

**lemma** *update-zip*:
$(zip\ xs\ ys)[i:=xy] = zip\ (xs[i:=fst\ xy])\ (ys[i:=snd\ xy])$
⟨*proof*⟩

**lemma** *set-update-subset-insert*: $set(xs[i:=x]) \leq insert\ x\ (set\ xs)$
⟨*proof*⟩

**lemma** *set-update-subsetI*: $[\![set\ xs \subseteq A;\ x \in A]\!] \Longrightarrow set(xs[i := x]) \subseteq A$
⟨*proof*⟩

**lemma** *set-update-memI*: $n < length\ xs \Longrightarrow x \in set\ (xs[n := x])$
⟨*proof*⟩

**lemma** *list-update-overwrite*[*simp*]:
$xs\ [i := x,\ i := y] = xs\ [i := y]$
⟨*proof*⟩

**lemma** *list-update-swap*:
$i \neq i' \Longrightarrow xs\ [i := x,\ i' := x'] = xs\ [i' := x',\ i := x]$
⟨*proof*⟩

**lemma** *list-update-code* [*code*]:
$[][i := y] = []$
$(x\ \#\ xs)[0 := y] = y\ \#\ xs$
$(x\ \#\ xs)[Suc\ i := y] = x\ \#\ xs[i := y]$
⟨*proof*⟩

### 66.1.13 *last* and *butlast*

**lemma** *hd-Nil-eq-last*: $hd\ Nil = last\ Nil$
⟨*proof*⟩

**lemma** *last-snoc* [*simp*]: $last\ (xs\ @\ [x]) = x$
⟨*proof*⟩

**lemma** *butlast-snoc* [*simp*]: $butlast\ (xs\ @\ [x]) = xs$
⟨*proof*⟩

**lemma** *last-ConsL*: $xs = [] \Longrightarrow last(x\#xs) = x$
⟨*proof*⟩

**lemma** *last-ConsR*: $xs \neq [] \Longrightarrow last(x\#xs) = last\ xs$
⟨*proof*⟩

**lemma** *last-append*: $last(xs\ @\ ys) = (if\ ys = []\ then\ last\ xs\ else\ last\ ys)$
⟨*proof*⟩

**lemma** *last-appendL*[*simp*]: $ys = [] \Longrightarrow last(xs\ @\ ys) = last\ xs$
⟨*proof*⟩

**lemma** *last-appendR*[*simp*]: $ys \neq [] \Longrightarrow last(xs\ @\ ys) = last\ ys$
⟨*proof*⟩

**lemma** *last-tl*: $xs = [] \lor tl\ xs \neq [] \Longrightarrow last\ (tl\ xs) = last\ xs$
⟨*proof*⟩

**lemma** *butlast-tl*: $butlast\ (tl\ xs) = tl\ (butlast\ xs)$
⟨*proof*⟩

**lemma** *hd-rev*: $hd(rev\ xs) = last\ xs$
⟨*proof*⟩

**lemma** *last-rev*: $last(rev\ xs) = hd\ xs$
⟨*proof*⟩

**lemma** *last-in-set*[*simp*]: $as \neq [] \Longrightarrow last\ as \in set\ as$
⟨*proof*⟩

**lemma** *length-butlast* [*simp*]: $length\ (butlast\ xs) = length\ xs - 1$
⟨*proof*⟩

**lemma** *butlast-append*:
$butlast\ (xs\ @\ ys) = (if\ ys = []\ then\ butlast\ xs\ else\ xs\ @\ butlast\ ys)$
⟨*proof*⟩

**lemma** *append-butlast-last-id* [*simp*]:
$xs \neq [] \Longrightarrow butlast\ xs\ @\ [last\ xs] = xs$
⟨*proof*⟩

**lemma** *in-set-butlastD*: $x \in set\ (butlast\ xs) \Longrightarrow x \in set\ xs$
⟨*proof*⟩

**lemma** *in-set-butlast-appendI*:
$x \in set\ (butlast\ xs) \lor x \in set\ (butlast\ ys) \Longrightarrow x \in set\ (butlast\ (xs\ @\ ys))$
⟨*proof*⟩

**lemma** *last-drop*[*simp*]: $n < length\ xs \Longrightarrow last\ (drop\ n\ xs) = last\ xs$
⟨*proof*⟩

**lemma** *nth-butlast*:
**assumes** $n < length\ (butlast\ xs)$ **shows** $butlast\ xs\ !\ n = xs\ !\ n$
⟨*proof*⟩

**lemma** *last-conv-nth*: $xs{\neq}[] \Longrightarrow last\ xs = xs!(length\ xs - 1)$
⟨*proof*⟩

**lemma** *butlast-conv-take*: $butlast\ xs = take\ (length\ xs - 1)\ xs$
⟨*proof*⟩

**lemma** *last-list-update*:
$xs \neq [] \Longrightarrow last(xs[k{:=}x]) = (if\ k = size\ xs - 1\ then\ x\ else\ last\ xs)$
⟨*proof*⟩

**lemma** *butlast-list-update*:

$butlast(xs[k:=x]) =$
$(if\ k = size\ xs - 1\ then\ butlast\ xs\ else\ (butlast\ xs)[k:=x])$
⟨*proof*⟩

**lemma** *last-map*: $xs \neq [] \Longrightarrow last\ (map\ f\ xs) = f\ (last\ xs)$
⟨*proof*⟩

**lemma** *map-butlast*: $map\ f\ (butlast\ xs) = butlast\ (map\ f\ xs)$
⟨*proof*⟩

**lemma** *snoc-eq-iff-butlast*:
$xs\ @\ [x] = ys \longleftrightarrow (ys \neq [] \wedge butlast\ ys = xs \wedge last\ ys = x)$
⟨*proof*⟩

**corollary** *longest-common-suffix*:
$\exists\, ss\ xs'\ ys'.\ xs = xs'\ @\ ss \wedge ys = ys'\ @\ ss$
$\wedge\ (xs' = [] \vee ys' = [] \vee last\ xs' \neq last\ ys')$
⟨*proof*⟩

**lemma** *butlast-rev* [*simp*]: $butlast\ (rev\ xs) = rev\ (tl\ xs)$
⟨*proof*⟩

### 66.1.14 *take* and *drop*

**lemma** *take-0*: $take\ 0\ xs = []$
⟨*proof*⟩

**lemma** *drop-0*: $drop\ 0\ xs = xs$
⟨*proof*⟩

**lemma** *take0*[*simp*]: $take\ 0 = (\lambda xs.\ [])$
⟨*proof*⟩

**lemma** *drop0*[*simp*]: $drop\ 0 = (\lambda x.\ x)$
⟨*proof*⟩

**lemma** *take-Suc-Cons* [*simp*]: $take\ (Suc\ n)\ (x\ \#\ xs) = x\ \#\ take\ n\ xs$
⟨*proof*⟩

**lemma** *drop-Suc-Cons* [*simp*]: $drop\ (Suc\ n)\ (x\ \#\ xs) = drop\ n\ xs$
⟨*proof*⟩

**declare** *take-Cons* [*simp del*] **and** *drop-Cons* [*simp del*]

**lemma** *take-Suc*: $xs \neq [] \Longrightarrow take\ (Suc\ n)\ xs = hd\ xs\ \#\ take\ n\ (tl\ xs)$
⟨*proof*⟩

**lemma** *drop-Suc*: $drop\ (Suc\ n)\ xs = drop\ n\ (tl\ xs)$
⟨*proof*⟩

**lemma** *hd-take*[*simp*]: $j > 0 \Longrightarrow hd\ (take\ j\ xs) = hd\ xs$
⟨*proof*⟩

**lemma** *take-tl*: $take\ n\ (tl\ xs) = tl\ (take\ (Suc\ n)\ xs)$
⟨*proof*⟩

**lemma** *drop-tl*: $drop\ n\ (tl\ xs) = tl(drop\ n\ xs)$
⟨*proof*⟩

**lemma** *tl-take*: $tl\ (take\ n\ xs) = take\ (n - 1)\ (tl\ xs)$
⟨*proof*⟩

**lemma** *tl-drop*: $tl\ (drop\ n\ xs) = drop\ n\ (tl\ xs)$
⟨*proof*⟩

**lemma** *nth-via-drop*: $drop\ n\ xs = y\#ys \Longrightarrow xs!n = y$
⟨*proof*⟩

**lemma** *take-Suc-conv-app-nth*:
$i < length\ xs \Longrightarrow take\ (Suc\ i)\ xs = take\ i\ xs\ @\ [xs!i]$
⟨*proof*⟩

**lemma** *Cons-nth-drop-Suc*:
$i < length\ xs \Longrightarrow (xs!i)\ \#\ (drop\ (Suc\ i)\ xs) = drop\ i\ xs$
⟨*proof*⟩

**lemma** *length-take* [*simp*]: $length\ (take\ n\ xs) = min\ (length\ xs)\ n$
⟨*proof*⟩

**lemma** *length-drop* [*simp*]: $length\ (drop\ n\ xs) = (length\ xs - n)$
⟨*proof*⟩

**lemma** *take-all* [*simp*]: $length\ xs \leq n \Longrightarrow take\ n\ xs = xs$
⟨*proof*⟩

**lemma** *drop-all* [*simp*]: $length\ xs \leq n \Longrightarrow drop\ n\ xs = []$
⟨*proof*⟩

**lemma** *take-all-iff* [*simp*]: $take\ n\ xs = xs \longleftrightarrow length\ xs \leq n$
⟨*proof*⟩

**lemma** *take-eq-Nil*[*simp*]: $(take\ n\ xs = []) = (n = 0 \lor xs = [])$
⟨*proof*⟩

**lemmas** *take-eq-Nil2*[*simp*] = *take-eq-Nil*[*THEN eq-iff-swap*]

**lemma** *drop-eq-Nil* [*simp*]: $drop\ n\ xs = [] \longleftrightarrow length\ xs \leq n$
⟨*proof*⟩

**lemmas** *drop-eq-Nil2* [*simp*] = *drop-eq-Nil*[*THEN eq-iff-swap*]

**lemma** *take-append* [*simp*]:
$take\ n\ (xs\ @\ ys) = (take\ n\ xs\ @\ take\ (n - length\ xs)\ ys)$
⟨*proof*⟩

**lemma** *drop-append* [*simp*]:
$drop\ n\ (xs\ @\ ys) = drop\ n\ xs\ @\ drop\ (n - length\ xs)\ ys$
⟨*proof*⟩

**lemma** *take-take* [*simp*]: $take\ n\ (take\ m\ xs) = take\ (min\ n\ m)\ xs$
⟨*proof*⟩

**lemma** *drop-drop* [*simp*]: $drop\ n\ (drop\ m\ xs) = drop\ (n + m)\ xs$
⟨*proof*⟩

**lemma** *take-drop*: $take\ n\ (drop\ m\ xs) = drop\ m\ (take\ (n + m)\ xs)$
⟨*proof*⟩

**lemma** *drop-take*: $drop\ n\ (take\ m\ xs) = take\ (m-n)\ (drop\ n\ xs)$
⟨*proof*⟩

**lemma** *append-take-drop-id* [*simp*]: $take\ n\ xs\ @\ drop\ n\ xs = xs$
⟨*proof*⟩

**lemma** *take-map*: $take\ n\ (map\ f\ xs) = map\ f\ (take\ n\ xs)$
⟨*proof*⟩

**lemma** *drop-map*: $drop\ n\ (map\ f\ xs) = map\ f\ (drop\ n\ xs)$
⟨*proof*⟩

**lemma** *rev-take*: $rev\ (take\ i\ xs) = drop\ (length\ xs - i)\ (rev\ xs)$
⟨*proof*⟩

**lemma** *rev-drop*: $rev\ (drop\ i\ xs) = take\ (length\ xs - i)\ (rev\ xs)$
⟨*proof*⟩

**lemma** *drop-rev*: $drop\ n\ (rev\ xs) = rev\ (take\ (length\ xs - n)\ xs)$
⟨*proof*⟩

**lemma** *take-rev*: $take\ n\ (rev\ xs) = rev\ (drop\ (length\ xs - n)\ xs)$
⟨*proof*⟩

**lemma** *nth-take* [*simp*]: $i < n \Longrightarrow (take\ n\ xs)!i = xs!i$
⟨*proof*⟩

**lemma** *nth-drop* [*simp*]:
$n \le length\ xs \Longrightarrow (drop\ n\ xs)!i = xs!(n + i)$
⟨*proof*⟩

**lemma** *butlast-take*:
$n \le length\ xs \Longrightarrow butlast\ (take\ n\ xs) = take\ (n - 1)\ xs$
⟨*proof*⟩

**lemma** *butlast-drop*: $butlast\ (drop\ n\ xs) = drop\ n\ (butlast\ xs)$
⟨*proof*⟩

**lemma** *take-butlast*: $n < length\ xs \Longrightarrow take\ n\ (butlast\ xs) = take\ n\ xs$
⟨*proof*⟩

**lemma** *drop-butlast*: $drop\ n\ (butlast\ xs) = butlast\ (drop\ n\ xs)$
⟨*proof*⟩

**lemma** *butlast-power*: $(butlast\ \widehat{}\widehat{}\ n)\ xs = take\ (length\ xs - n)\ xs$
⟨*proof*⟩

**lemma** *hd-drop-conv-nth*: $n < length\ xs \Longrightarrow hd(drop\ n\ xs) = xs!n$
⟨*proof*⟩

**lemma** *set-take-subset-set-take*:
$m \le n \Longrightarrow set(take\ m\ xs) \le set(take\ n\ xs)$
⟨*proof*⟩

**lemma** *set-take-subset*: $set(take\ n\ xs) \subseteq set\ xs$
⟨*proof*⟩

**lemma** *set-drop-subset*: $set(drop\ n\ xs) \subseteq set\ xs$
⟨*proof*⟩

**lemma** *set-drop-subset-set-drop*:
$m \ge n \Longrightarrow set(drop\ m\ xs) \le set(drop\ n\ xs)$
⟨*proof*⟩

**lemma** *in-set-takeD*: $x \in set(take\ n\ xs) \Longrightarrow x \in set\ xs$
⟨*proof*⟩

**lemma** *in-set-dropD*: $x \in set(drop\ n\ xs) \Longrightarrow x \in set\ xs$
⟨*proof*⟩

**lemma** *append-eq-conv-conj*:
$(xs\ @\ ys = zs) = (xs = take\ (length\ xs)\ zs \wedge ys = drop\ (length\ xs)\ zs)$
⟨*proof*⟩

**lemma** *map-eq-append-conv*:
$map\ f\ xs = ys\ @\ zs \longleftrightarrow (\exists\ us\ vs.\ xs = us\ @\ vs \wedge ys = map\ f\ us \wedge zs = map\ f$

*vs*)
⟨*proof*⟩

**lemmas** *append-eq-map-conv* = *map-eq-append-conv*[*THEN eq-iff-swap*]

**lemma** *take-add*: *take* $(i+j)$ *xs* = *take i xs* @ *take j* (*drop i xs*)
⟨*proof*⟩

**lemma** *append-eq-append-conv-if*:
($xs_1$ @ $xs_2$ = $ys_1$ @ $ys_2$) =
(*if size* $xs_1 \leq$ *size* $ys_1$
*then* $xs_1$ = *take* (*size* $xs_1$) $ys_1$ $\wedge$ $xs_2$ = *drop* (*size* $xs_1$) $ys_1$ @ $ys_2$
*else take* (*size* $ys_1$) $xs_1$ = $ys_1$ $\wedge$ *drop* (*size* $ys_1$) $xs_1$ @ $xs_2$ = $ys_2$)
⟨*proof*⟩

**lemma** *take-hd-drop*:
$n <$ *length xs* $\Longrightarrow$ *take n xs* @ [*hd* (*drop n xs*)] = *take* (*Suc n*) *xs*
⟨*proof*⟩

**lemma** *id-take-nth-drop*:
$i <$ *length xs* $\Longrightarrow$ *xs* = *take i xs* @ *xs*!*i* # *drop* (*Suc i*) *xs*
⟨*proof*⟩

**lemma** *take-update-cancel*[*simp*]: $n \leq m \Longrightarrow$ *take n* (*xs*[$m$ := $y$]) = *take n xs*
⟨*proof*⟩

**lemma** *drop-update-cancel*[*simp*]: $n < m \Longrightarrow$ *drop m* (*xs*[$n$ := $x$]) = *drop m xs*
⟨*proof*⟩

**lemma** *upd-conv-take-nth-drop*:
$i <$ *length xs* $\Longrightarrow$ *xs*[$i$:=$a$] = *take i xs* @ *a* # *drop* (*Suc i*) *xs*
⟨*proof*⟩

**lemma** *take-update-swap*: *take m* (*xs*[$n$ := $x$]) = (*take m xs*)[$n$ := $x$]
⟨*proof*⟩

**lemma** *drop-update-swap*:
**assumes** $m \leq n$ **shows** *drop m* (*xs*[$n$ := $x$]) = (*drop m xs*)[$n-m$ := $x$]
⟨*proof*⟩

**lemma** *nth-image*: $l \leq$ *size xs* $\Longrightarrow$ *nth xs* ` {*0*..<*l*} = *set*(*take l xs*)
⟨*proof*⟩

**lemma** *set-list-update*:
*set* (*xs* [$i$ := $k$]) = *insert k* (*set* (*take i xs*) $\cup$ *set* (*drop* (*Suc i*) *xs*))
**if** ‹$i <$ *length xs*›
⟨*proof*⟩

### 66.1.15 *takeWhile* and *dropWhile*

**lemma** *length-takeWhile-le*: $length\ (takeWhile\ P\ xs) \leq length\ xs$
⟨*proof*⟩

**lemma** *takeWhile-dropWhile-id* [*simp*]: $takeWhile\ P\ xs\ @\ dropWhile\ P\ xs = xs$
⟨*proof*⟩

**lemma** *takeWhile-append1* [*simp*]:
$[\![x \in set\ xs;\ \neg P(x)]\!] \Longrightarrow takeWhile\ P\ (xs\ @\ ys) = takeWhile\ P\ xs$
⟨*proof*⟩

**lemma** *takeWhile-append2* [*simp*]:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow P\ x) \Longrightarrow takeWhile\ P\ (xs\ @\ ys) = xs\ @\ takeWhile\ P\ ys$
⟨*proof*⟩

**lemma** *takeWhile-append*:
$takeWhile\ P\ (xs\ @\ ys) = (if\ \forall x{\in}set\ xs.\ P\ x\ then\ xs\ @\ takeWhile\ P\ ys\ else\ takeWhile\ P\ xs)$
⟨*proof*⟩

**lemma** *takeWhile-tail*: $\neg\ P\ x \Longrightarrow takeWhile\ P\ (xs\ @\ (x\#l)) = takeWhile\ P\ xs$
⟨*proof*⟩

**lemma** *takeWhile-eq-Nil-iff*: $takeWhile\ P\ xs = [] \longleftrightarrow xs = [] \vee \neg P\ (hd\ xs)$
⟨*proof*⟩

**lemma** *takeWhile-nth*: $j < length\ (takeWhile\ P\ xs) \Longrightarrow takeWhile\ P\ xs\ !\ j = xs\ !\ j$
⟨*proof*⟩

**lemma** *takeWhile-takeWhile*: $takeWhile\ Q\ (takeWhile\ P\ xs) = takeWhile\ (\lambda x.\ P\ x \wedge Q\ x)\ xs$
⟨*proof*⟩

**lemma** *dropWhile-nth*: $j < length\ (dropWhile\ P\ xs) \Longrightarrow$
$dropWhile\ P\ xs\ !\ j = xs\ !\ (j + length\ (takeWhile\ P\ xs))$
⟨*proof*⟩

**lemma** *length-dropWhile-le*: $length\ (dropWhile\ P\ xs) \leq length\ xs$
⟨*proof*⟩

**lemma** *dropWhile-append1* [*simp*]:
$[\![x \in set\ xs;\ \neg P(x)]\!] \Longrightarrow dropWhile\ P\ (xs\ @\ ys) = (dropWhile\ P\ xs)@ys$
⟨*proof*⟩

**lemma** *dropWhile-append2* [*simp*]:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow P(x)) \Longrightarrow dropWhile\ P\ (xs\ @\ ys) = dropWhile\ P\ ys$
⟨*proof*⟩

**lemma** *dropWhile-id*[*simp*]:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow \neg\ P\ x) \Longrightarrow dropWhile\ P\ xs = xs$
⟨*proof*⟩

**lemma** *dropWhile-append3*:
$\neg\ P\ y \Longrightarrow dropWhile\ P\ (xs\ @\ y\ \#\ ys) = dropWhile\ P\ xs\ @\ y\ \#\ ys$
⟨*proof*⟩

**lemma** *dropWhile-append*:
$dropWhile\ P\ (xs\ @\ ys) = (if\ \forall x \in set\ xs.\ P\ x\ then\ dropWhile\ P\ ys\ else\ dropWhile\ P\ xs\ @\ ys)$
⟨*proof*⟩

**lemma** *dropWhile-last*:
$x \in set\ xs \Longrightarrow \neg\ P\ x \Longrightarrow last\ (dropWhile\ P\ xs) = last\ xs$
⟨*proof*⟩

**lemma** *set-dropWhileD*: $x \in set\ (dropWhile\ P\ xs) \Longrightarrow x \in set\ xs$
⟨*proof*⟩

**lemma** *set-takeWhileD*: $x \in set\ (takeWhile\ P\ xs) \Longrightarrow x \in set\ xs \land P\ x$
⟨*proof*⟩

**lemma** *takeWhile-eq-all-conv*[*simp*]:
$(takeWhile\ P\ xs = xs) = (\forall x \in set\ xs.\ P\ x)$
⟨*proof*⟩

**lemma** *dropWhile-eq-Nil-conv*[*simp*]:
$(dropWhile\ P\ xs = []) = (\forall x \in set\ xs.\ P\ x)$
⟨*proof*⟩

**lemma** *dropWhile-eq-Cons-conv*:
$(dropWhile\ P\ xs = y\#ys) = (xs = takeWhile\ P\ xs\ @\ y\ \#\ ys \land \neg\ P\ y)$
⟨*proof*⟩

**lemma** *dropWhile-eq-self-iff*: $dropWhile\ P\ xs = xs \longleftrightarrow xs = [] \lor \neg P\ (hd\ xs)$
⟨*proof*⟩

**lemma** *dropWhile-dropWhile1*: $(\bigwedge x.\ Q\ x \Longrightarrow P\ x) \Longrightarrow dropWhile\ Q\ (dropWhile\ P\ xs) = dropWhile\ P\ xs$
⟨*proof*⟩

**lemma** *dropWhile-dropWhile2*: $(\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow takeWhile\ P\ (takeWhile\ Q\ xs) = takeWhile\ P\ xs$
⟨*proof*⟩

**lemma** *dropWhile-takeWhile*:
$(\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow dropWhile\ P\ (takeWhile\ Q\ xs) = takeWhile\ Q\ (dropWhile\ P\ xs)$

$\langle proof \rangle$

**lemma** *distinct-takeWhile*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (takeWhile\ P\ xs)$
$\langle proof \rangle$

**lemma** *distinct-dropWhile*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (dropWhile\ P\ xs)$
$\langle proof \rangle$

**lemma** *takeWhile-map*: $takeWhile\ P\ (map\ f\ xs) = map\ f\ (takeWhile\ (P \circ f)\ xs)$
$\langle proof \rangle$

**lemma** *dropWhile-map*: $dropWhile\ P\ (map\ f\ xs) = map\ f\ (dropWhile\ (P \circ f)\ xs)$
$\langle proof \rangle$

**lemma** *takeWhile-eq-take*: $takeWhile\ P\ xs = take\ (length\ (takeWhile\ P\ xs))\ xs$
$\langle proof \rangle$

**lemma** *dropWhile-eq-drop*: $dropWhile\ P\ xs = drop\ (length\ (takeWhile\ P\ xs))\ xs$
$\langle proof \rangle$

**lemma** *hd-dropWhile*: $dropWhile\ P\ xs \neq [] \Longrightarrow \neg\ P\ (hd\ (dropWhile\ P\ xs))$
$\langle proof \rangle$

**lemma** *takeWhile-eq-filter*:
**assumes** $\bigwedge x.\ x \in set\ (dropWhile\ P\ xs) \Longrightarrow \neg\ P\ x$
**shows** $takeWhile\ P\ xs = filter\ P\ xs$
$\langle proof \rangle$

**lemma** *takeWhile-eq-take-P-nth*:
$[\![ \bigwedge i.\ [\![ i < n\ ;\ i < length\ xs ]\!] \Longrightarrow P\ (xs\ !\ i)\ ;\ n < length\ xs \Longrightarrow \neg\ P\ (xs\ !\ n) ]\!]$
$\Longrightarrow$
$takeWhile\ P\ xs = take\ n\ xs$
$\langle proof \rangle$

**lemma** *nth-length-takeWhile*:
$length\ (takeWhile\ P\ xs) < length\ xs \Longrightarrow \neg\ P\ (xs\ !\ length\ (takeWhile\ P\ xs))$
$\langle proof \rangle$

**lemma** *length-takeWhile-less-P-nth*:
**assumes** *all*: $\bigwedge i.\ i < j \Longrightarrow P\ (xs\ !\ i)$ **and** $j \leq length\ xs$
**shows** $j \leq length\ (takeWhile\ P\ xs)$
$\langle proof \rangle$

**lemma** *takeWhile-neq-rev*: $[\![distinct\ xs;\ x \in set\ xs]\!] \Longrightarrow$
$takeWhile\ (\lambda y.\ y \neq x)\ (rev\ xs) = rev\ (tl\ (dropWhile\ (\lambda y.\ y \neq x)\ xs))$
$\langle proof \rangle$

**lemma** *dropWhile-neq-rev*: $[\![distinct\ xs;\ x \in set\ xs]\!] \Longrightarrow$
$dropWhile\ (\lambda y.\ y \neq x)\ (rev\ xs) = x\ \#\ rev\ (takeWhile\ (\lambda y.\ y \neq x)\ xs)$

⟨*proof*⟩

**lemma** *takeWhile-not-last*:
  $distinct\ xs \Longrightarrow takeWhile\ (\lambda y.\ y \neq last\ xs)\ xs = butlast\ xs$
  ⟨*proof*⟩

**lemma** *takeWhile-cong* [*fundef-cong*]:
  $[\![l = k;\ \bigwedge x.\ x \in set\ l \Longrightarrow P\ x = Q\ x]\!]$
  $\Longrightarrow takeWhile\ P\ l = takeWhile\ Q\ k$
  ⟨*proof*⟩

**lemma** *dropWhile-cong* [*fundef-cong*]:
  $[\![l = k;\ \bigwedge x.\ x \in set\ l \Longrightarrow P\ x = Q\ x]\!]$
  $\Longrightarrow dropWhile\ P\ l = dropWhile\ Q\ k$
  ⟨*proof*⟩

**lemma** *takeWhile-idem* [*simp*]:
  $takeWhile\ P\ (takeWhile\ P\ xs) = takeWhile\ P\ xs$
  ⟨*proof*⟩

**lemma** *dropWhile-idem* [*simp*]:
  $dropWhile\ P\ (dropWhile\ P\ xs) = dropWhile\ P\ xs$
  ⟨*proof*⟩

### 66.1.16 *zip*

**lemma** *zip-Nil* [*simp*]: $zip\ []\ ys = []$
  ⟨*proof*⟩

**lemma** *zip-Cons-Cons* [*simp*]: $zip\ (x\ \#\ xs)\ (y\ \#\ ys) = (x,\ y)\ \#\ zip\ xs\ ys$
  ⟨*proof*⟩

**declare** *zip-Cons* [*simp del*]

**lemma** [*code*]:
  $zip\ []\ ys = []$
  $zip\ xs\ [] = []$
  $zip\ (x\ \#\ xs)\ (y\ \#\ ys) = (x,\ y)\ \#\ zip\ xs\ ys$
  ⟨*proof*⟩

**lemma** *zip-Cons1*:
  $zip\ (x\#xs)\ ys = (case\ ys\ of\ [] \Rightarrow []\ |\ y\#ys \Rightarrow (x,y)\#zip\ xs\ ys)$
  ⟨*proof*⟩

**lemma** *length-zip* [*simp*]:
  $length\ (zip\ xs\ ys) = min\ (length\ xs)\ (length\ ys)$
  ⟨*proof*⟩

**lemma** *zip-obtain-same-length*:

**assumes** $\bigwedge zs\ ws\ n.\ length\ zs = length\ ws \Longrightarrow n = min\ (length\ xs)\ (length\ ys)$
$\Longrightarrow zs = take\ n\ xs \Longrightarrow ws = take\ n\ ys \Longrightarrow P\ (zip\ zs\ ws)$
**shows** $P\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *zip-append1*:
$zip\ (xs\ @\ ys)\ zs =$
$zip\ xs\ (take\ (length\ xs)\ zs)\ @\ zip\ ys\ (drop\ (length\ xs)\ zs)$
⟨*proof*⟩

**lemma** *zip-append2*:
$zip\ xs\ (ys\ @\ zs) =$
$zip\ (take\ (length\ ys)\ xs)\ ys\ @\ zip\ (drop\ (length\ ys)\ xs)\ zs$
⟨*proof*⟩

**lemma** *zip-append* [*simp*]:
$[\![length\ xs = length\ us]\!] \Longrightarrow$
$zip\ (xs@ys)\ (us@vs) = zip\ xs\ us\ @\ zip\ ys\ vs$
⟨*proof*⟩

**lemma** *zip-rev*:
$length\ xs = length\ ys \Longrightarrow zip\ (rev\ xs)\ (rev\ ys) = rev\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *zip-map-map*:
$zip\ (map\ f\ xs)\ (map\ g\ ys) = map\ (\lambda\ (x,\ y).\ (f\ x,\ g\ y))\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *zip-map1*:
$zip\ (map\ f\ xs)\ ys = map\ (\lambda(x,\ y).\ (f\ x,\ y))\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *zip-map2*:
$zip\ xs\ (map\ f\ ys) = map\ (\lambda(x,\ y).\ (x,\ f\ y))\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *map-zip-map*:
$map\ f\ (zip\ (map\ g\ xs)\ ys) = map\ (\%(x,y).\ f(g\ x,\ y))\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *map-zip-map2*:
$map\ f\ (zip\ xs\ (map\ g\ ys)) = map\ (\%(x,y).\ f(x,\ g\ y))\ (zip\ xs\ ys)$
⟨*proof*⟩

Courtesy of Andreas Lochbihler:

**lemma** *zip-same-conv-map*: $zip\ xs\ xs = map\ (\lambda x.\ (x,\ x))\ xs$
⟨*proof*⟩

**lemma** *nth-zip* [*simp*]:

$[\![i < \mathit{length}\ xs;\ i < \mathit{length}\ ys]\!] \Longrightarrow (\mathit{zip}\ xs\ ys)!i = (xs!i,\ ys!i)$
⟨proof⟩

**lemma** *set-zip*:
$\mathit{set}\ (\mathit{zip}\ xs\ ys) = \{(xs!i,\ ys!i) \mid i.\ i < \mathit{min}\ (\mathit{length}\ xs)\ (\mathit{length}\ ys)\}$
⟨proof⟩

**lemma** *zip-same*: $((a,b) \in \mathit{set}\ (\mathit{zip}\ xs\ xs)) = (a \in \mathit{set}\ xs \wedge a = b)$
⟨proof⟩

**lemma** *zip-update*: $\mathit{zip}\ (xs[i:=x])\ (ys[i:=y]) = (\mathit{zip}\ xs\ ys)[i:=(x,y)]$
⟨proof⟩

**lemma** *zip-replicate* [*simp*]:
$\mathit{zip}\ (\mathit{replicate}\ i\ x)\ (\mathit{replicate}\ j\ y) = \mathit{replicate}\ (\mathit{min}\ i\ j)\ (x,y)$
⟨proof⟩

**lemma** *zip-replicate1*: $\mathit{zip}\ (\mathit{replicate}\ n\ x)\ ys = \mathit{map}\ (\mathit{Pair}\ x)\ (\mathit{take}\ n\ ys)$
⟨proof⟩

**lemma** *take-zip*: $\mathit{take}\ n\ (\mathit{zip}\ xs\ ys) = \mathit{zip}\ (\mathit{take}\ n\ xs)\ (\mathit{take}\ n\ ys)$
⟨proof⟩

**lemma** *drop-zip*: $\mathit{drop}\ n\ (\mathit{zip}\ xs\ ys) = \mathit{zip}\ (\mathit{drop}\ n\ xs)\ (\mathit{drop}\ n\ ys)$
⟨proof⟩

**lemma** *zip-takeWhile-fst*: $\mathit{zip}\ (\mathit{takeWhile}\ P\ xs)\ ys = \mathit{takeWhile}\ (P \circ \mathit{fst})\ (\mathit{zip}\ xs\ ys)$
⟨proof⟩

**lemma** *zip-takeWhile-snd*: $\mathit{zip}\ xs\ (\mathit{takeWhile}\ P\ ys) = \mathit{takeWhile}\ (P \circ \mathit{snd})\ (\mathit{zip}\ xs\ ys)$
⟨proof⟩

**lemma** *set-zip-leftD*: $(x,y) \in \mathit{set}\ (\mathit{zip}\ xs\ ys) \Longrightarrow x \in \mathit{set}\ xs$
⟨proof⟩

**lemma** *set-zip-rightD*: $(x,y) \in \mathit{set}\ (\mathit{zip}\ xs\ ys) \Longrightarrow y \in \mathit{set}\ ys$
⟨proof⟩

**lemma** *in-set-zipE*:
$(x,y) \in \mathit{set}(\mathit{zip}\ xs\ ys) \Longrightarrow ([\![\ x \in \mathit{set}\ xs;\ y \in \mathit{set}\ ys\ ]\!] \Longrightarrow R) \Longrightarrow R$
⟨proof⟩

**lemma** *zip-map-fst-snd*: $\mathit{zip}\ (\mathit{map}\ \mathit{fst}\ zs)\ (\mathit{map}\ \mathit{snd}\ zs) = zs$
⟨proof⟩

**lemma** *zip-eq-conv*:
$\mathit{length}\ xs = \mathit{length}\ ys \Longrightarrow \mathit{zip}\ xs\ ys = zs \longleftrightarrow \mathit{map}\ \mathit{fst}\ zs = xs \wedge \mathit{map}\ \mathit{snd}\ zs = ys$

⟨*proof*⟩

**lemma** *in-set-zip*:
$p \in set\ (zip\ xs\ ys) \longleftrightarrow (\exists n.\ xs\ !\ n = fst\ p \wedge ys\ !\ n = snd\ p$
$\wedge n < length\ xs \wedge n < length\ ys)$
⟨*proof*⟩

**lemma** *in-set-impl-in-set-zip1*:
**assumes** $length\ xs = length\ ys$
**assumes** $x \in set\ xs$
**obtains** $y$ **where** $(x, y) \in set\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *in-set-impl-in-set-zip2*:
**assumes** $length\ xs = length\ ys$
**assumes** $y \in set\ ys$
**obtains** $x$ **where** $(x, y) \in set\ (zip\ xs\ ys)$
⟨*proof*⟩

**lemma** *zip-eq-Nil-iff* [*simp*]:
$zip\ xs\ ys = [] \longleftrightarrow xs = [] \vee ys = []$
⟨*proof*⟩

**lemmas** *Nil-eq-zip-iff* [*simp*] = *zip-eq-Nil-iff* [*THEN eq-iff-swap*]

**lemma** *zip-eq-ConsE*:
**assumes** $zip\ xs\ ys = xy\ \#\ xys$
**obtains** $x\ xs'\ y\ ys'$ **where** $xs = x\ \#\ xs'$
**and** $ys = y\ \#\ ys'$ **and** $xy = (x, y)$
**and** $xys = zip\ xs'\ ys'$
⟨*proof*⟩

**lemma** *semilattice-map2*:
*semilattice* (*map2* (∗)) **if** *semilattice* (∗)
**for** *f* (**infixl** ‹∗› *70*)
⟨*proof*⟩

**lemma** *pair-list-eqI*:
**assumes** $map\ fst\ xs = map\ fst\ ys$ **and** $map\ snd\ xs = map\ snd\ ys$
**shows** $xs = ys$
⟨*proof*⟩

**lemma** *hd-zip*:
‹$hd\ (zip\ xs\ ys) = (hd\ xs, hd\ ys)$› **if** ‹$xs \neq []$› **and** ‹$ys \neq []$›
⟨*proof*⟩

**lemma** *last-zip*:
‹$last\ (zip\ xs\ ys) = (last\ xs, last\ ys)$› **if** ‹$xs \neq []$› **and** ‹$ys \neq []$›
**and** ‹$length\ xs = length\ ys$›

⟨*proof*⟩

### 66.1.17 *list-all2*

**lemma** *list-all2-lengthD* [*intro?*]:
$list\text{-}all2\ P\ xs\ ys \Longrightarrow length\ xs = length\ ys$
⟨*proof*⟩

**lemma** *list-all2-Nil* [*iff*, *code*]: $list\text{-}all2\ P\ []\ ys = (ys = [])$
⟨*proof*⟩

**lemma** *list-all2-Nil2* [*iff*, *code*]: $list\text{-}all2\ P\ xs\ [] = (xs = [])$
⟨*proof*⟩

**lemma** *list-all2-Cons* [*iff*, *code*]:
$list\text{-}all2\ P\ (x\ \#\ xs)\ (y\ \#\ ys) = (P\ x\ y \wedge list\text{-}all2\ P\ xs\ ys)$
⟨*proof*⟩

**lemma** *list-all2-Cons1*:
$list\text{-}all2\ P\ (x\ \#\ xs)\ ys = (\exists\, z\ zs.\ ys = z\ \#\ zs \wedge P\ x\ z \wedge list\text{-}all2\ P\ xs\ zs)$
⟨*proof*⟩

**lemma** *list-all2-Cons2*:
$list\text{-}all2\ P\ xs\ (y\ \#\ ys) = (\exists\, z\ zs.\ xs = z\ \#\ zs \wedge P\ z\ y \wedge list\text{-}all2\ P\ zs\ ys)$
⟨*proof*⟩

**lemma** *list-all2-induct*
[*consumes 1*, *case-names Nil Cons*, *induct set*: *list-all2*]:
**assumes** *P*: $list\text{-}all2\ P\ xs\ ys$
**assumes** *Nil*: $R\ []\ []$
**assumes** *Cons*: $\bigwedge x\ xs\ y\ ys.$
$[\![P\ x\ y;\ list\text{-}all2\ P\ xs\ ys;\ R\ xs\ ys]\!] \Longrightarrow R\ (x\ \#\ xs)\ (y\ \#\ ys)$
**shows** $R\ xs\ ys$
⟨*proof*⟩

**lemma** *list-all2-rev* [*iff*]:
$list\text{-}all2\ P\ (rev\ xs)\ (rev\ ys) = list\text{-}all2\ P\ xs\ ys$
⟨*proof*⟩

**lemma** *list-all2-rev1*:
$list\text{-}all2\ P\ (rev\ xs)\ ys = list\text{-}all2\ P\ xs\ (rev\ ys)$
⟨*proof*⟩

**lemma** *list-all2-append1*:
$list\text{-}all2\ P\ (xs\ @\ ys)\ zs =$
$(\exists\, us\ vs.\ zs = us\ @\ vs \wedge length\ us = length\ xs \wedge length\ vs = length\ ys \wedge$
$list\text{-}all2\ P\ xs\ us \wedge list\text{-}all2\ P\ ys\ vs)$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *list-all2-append2*:
$list\text{-}all2\ P\ xs\ (ys\ @\ zs) =$
$(\exists\, us\ vs.\ xs = us\ @\ vs \wedge length\ us = length\ ys \wedge length\ vs = length\ zs \wedge$
$list\text{-}all2\ P\ us\ ys \wedge list\text{-}all2\ P\ vs\ zs)$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *list-all2-append*:
$length\ xs = length\ ys \Longrightarrow$
$list\text{-}all2\ P\ (xs@us)\ (ys@vs) = (list\text{-}all2\ P\ xs\ ys \wedge list\text{-}all2\ P\ us\ vs)$
⟨*proof*⟩

**lemma** *list-all2-appendI* [*intro?*, *trans*]:
$[\![\ list\text{-}all2\ P\ a\ b;\ list\text{-}all2\ P\ c\ d\ ]\!] \Longrightarrow list\text{-}all2\ P\ (a@c)\ (b@d)$
⟨*proof*⟩

**lemma** *list-all2-conv-all-nth*:
$list\text{-}all2\ P\ xs\ ys =$
$(length\ xs = length\ ys \wedge (\forall\, i < length\ xs.\ P\ (xs!i)\ (ys!i)))$
⟨*proof*⟩

**lemma** *list-all2-trans*:
**assumes** *tr*: $!!a\ b\ c.\ P1\ a\ b \Longrightarrow P2\ b\ c \Longrightarrow P3\ a\ c$
**shows** $!!bs\ cs.\ list\text{-}all2\ P1\ as\ bs \Longrightarrow list\text{-}all2\ P2\ bs\ cs \Longrightarrow list\text{-}all2\ P3\ as\ cs$
(**is** $!!bs\ cs.\ PROP\ ?Q\ as\ bs\ cs$)
⟨*proof*⟩

**lemma** *list-all2-all-nthI* [*intro?*]:
$length\ a = length\ b \Longrightarrow (\bigwedge n.\ n < length\ a \Longrightarrow P\ (a!n)\ (b!n)) \Longrightarrow list\text{-}all2\ P\ a\ b$
⟨*proof*⟩

**lemma** *list-all2I*:
$\forall\, x \in set\ (zip\ a\ b).\ case\text{-}prod\ P\ x \Longrightarrow length\ a = length\ b \Longrightarrow list\text{-}all2\ P\ a\ b$
⟨*proof*⟩

**lemma** *list-all2-nthD*:
$[\![\ list\text{-}all2\ P\ xs\ ys;\ p < size\ xs\ ]\!] \Longrightarrow P\ (xs!p)\ (ys!p)$
⟨*proof*⟩

**lemma** *list-all2-nthD2*:
$[\![list\text{-}all2\ P\ xs\ ys;\ p < size\ ys]\!] \Longrightarrow P\ (xs!p)\ (ys!p)$
⟨*proof*⟩

**lemma** *list-all2-map1*:
$list\text{-}all2\ P\ (map\ f\ as)\ bs = list\text{-}all2\ (\lambda x\ y.\ P\ (f\ x)\ y)\ as\ bs$
⟨*proof*⟩

**lemma** *list-all2-map2*:
$list\text{-}all2\ P\ as\ (map\ f\ bs) = list\text{-}all2\ (\lambda x\ y.\ P\ x\ (f\ y))\ as\ bs$
⟨*proof*⟩

**lemma** *list-all2-refl* [*intro?*]:
$(\bigwedge x.\ P\ x\ x) \Longrightarrow list\text{-}all2\ P\ xs\ xs$
⟨*proof*⟩

**lemma** *list-all2-update-cong*:
$[\![\ list\text{-}all2\ P\ xs\ ys;\ P\ x\ y\ ]\!] \Longrightarrow list\text{-}all2\ P\ (xs[i:=x])\ (ys[i:=y])$
⟨*proof*⟩

**lemma** *list-all2-takeI* [*simp*,*intro?*]:
$list\text{-}all2\ P\ xs\ ys \Longrightarrow list\text{-}all2\ P\ (take\ n\ xs)\ (take\ n\ ys)$
⟨*proof*⟩

**lemma** *list-all2-dropI* [*simp*,*intro?*]:
$list\text{-}all2\ P\ xs\ ys \Longrightarrow list\text{-}all2\ P\ (drop\ n\ xs)\ (drop\ n\ ys)$
⟨*proof*⟩

**lemma** *list-all2-mono* [*intro?*]:
$list\text{-}all2\ P\ xs\ ys \Longrightarrow (\bigwedge xs\ ys.\ P\ xs\ ys \Longrightarrow Q\ xs\ ys) \Longrightarrow list\text{-}all2\ Q\ xs\ ys$
⟨*proof*⟩

**lemma** *list-all2-eq*:
$xs = ys \longleftrightarrow list\text{-}all2\ (=)\ xs\ ys$
⟨*proof*⟩

**lemma** *list-eq-iff-zip-eq*:
$xs = ys \longleftrightarrow length\ xs = length\ ys \wedge (\forall (x,y) \in set\ (zip\ xs\ ys).\ x = y)$
⟨*proof*⟩

**lemma** *list-all2-same*: $list\text{-}all2\ P\ xs\ xs \longleftrightarrow (\forall x{\in}set\ xs.\ P\ x\ x)$
⟨*proof*⟩

**lemma** *zip-assoc*:
$zip\ xs\ (zip\ ys\ zs) = map\ (\lambda((x,\ y),\ z).\ (x,\ y,\ z))\ (zip\ (zip\ xs\ ys)\ zs)$
⟨*proof*⟩

**lemma** *zip-commute*: $zip\ xs\ ys = map\ (\lambda(x,\ y).\ (y,\ x))\ (zip\ ys\ xs)$
⟨*proof*⟩

**lemma** *zip-left-commute*:
$zip\ xs\ (zip\ ys\ zs) = map\ (\lambda(y,\ (x,\ z)).\ (x,\ y,\ z))\ (zip\ ys\ (zip\ xs\ zs))$
⟨*proof*⟩

**lemma** *zip-replicate2*: $zip\ xs\ (replicate\ n\ y) = map\ (\lambda x.\ (x,\ y))\ (take\ n\ xs)$
⟨*proof*⟩

### 66.1.18 *List.product* and *product-lists*

**lemma** *product-concat-map*:

$List.product\ xs\ ys = concat\ (map\ (\lambda x.\ map\ (\lambda y.\ (x,y))\ ys)\ xs)$
$\langle proof \rangle$

**lemma** *set-product*[*simp*]: $set\ (List.product\ xs\ ys) = set\ xs \times set\ ys$
$\langle proof \rangle$

**lemma** *length-product* [*simp*]:
$length\ (List.product\ xs\ ys) = length\ xs * length\ ys$
$\langle proof \rangle$

**lemma** *product-nth*:
**assumes** $n < length\ xs * length\ ys$
**shows** $List.product\ xs\ ys\ !\ n = (xs\ !\ (n\ div\ length\ ys),\ ys\ !\ (n\ mod\ length\ ys))$
$\langle proof \rangle$

**lemma** *in-set-product-lists-length*:
$xs \in set\ (product\text{-}lists\ xss) \Longrightarrow length\ xs = length\ xss$
$\langle proof \rangle$

**lemma** *product-lists-set*:
$set\ (product\text{-}lists\ xss) = \{xs.\ list\text{-}all2\ (\lambda x\ ys.\ x \in set\ ys)\ xs\ xss\}$ (**is** $?L = Collect\ ?R$)
$\langle proof \rangle$

### 66.1.19 *fold* with natural argument order

**lemma** *fold-simps* [*code*]: — eta-expanded variant for generated code – enables tail-recursion optimisation in Scala
$fold\ f\ []\ s = s$
$fold\ f\ (x\ \#\ xs)\ s = fold\ f\ xs\ (f\ x\ s)$
$\langle proof \rangle$

**lemma** *fold-remove1-split*:
$[\![\ \bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow f\ x \circ f\ y = f\ y \circ f\ x;$
$x \in set\ xs\ ]\!]$
$\Longrightarrow fold\ f\ xs = fold\ f\ (remove1\ x\ xs) \circ f\ x$
$\langle proof \rangle$

**lemma** *fold-cong* [*fundef-cong*]:
$a = b \Longrightarrow xs = ys \Longrightarrow (\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x = g\ x)$
$\Longrightarrow fold\ f\ xs\ a = fold\ g\ ys\ b$
$\langle proof \rangle$

**lemma** *fold-id*: $(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x = id) \Longrightarrow fold\ f\ xs = id$
$\langle proof \rangle$

**lemma** *fold-commute*:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow h \circ g\ x = f\ x \circ h) \Longrightarrow h \circ fold\ g\ xs = fold\ f\ xs \circ h$
$\langle proof \rangle$

**lemma** *fold-commute-apply*:
**assumes** $\bigwedge x.\ x \in set\ xs \Longrightarrow h \circ g\ x = f\ x \circ h$
**shows** $h\ (fold\ g\ xs\ s) = fold\ f\ xs\ (h\ s)$
$\langle proof \rangle$

**lemma** *fold-invariant*:
$[\![\ \bigwedge x.\ x \in set\ xs \Longrightarrow Q\ x;\ \ P\ s;\ \ \bigwedge x\ s.\ Q\ x \Longrightarrow P\ s \Longrightarrow P\ (f\ x\ s)\ ]\!]$
$\Longrightarrow P\ (fold\ f\ xs\ s)$
$\langle proof \rangle$

**lemma** *fold-append* [*simp*]: $fold\ f\ (xs\ @\ ys) = fold\ f\ ys \circ fold\ f\ xs$
$\langle proof \rangle$

**lemma** *fold-map* [*code-unfold*]: $fold\ g\ (map\ f\ xs) = fold\ (g \circ f)\ xs$
$\langle proof \rangle$

**lemma** *fold-filter*:
$fold\ f\ (filter\ P\ xs) = fold\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ id)\ xs$
$\langle proof \rangle$

**lemma** *fold-rev*:
$(\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow f\ y \circ f\ x = f\ x \circ f\ y)$
$\Longrightarrow fold\ f\ (rev\ xs) = fold\ f\ xs$
$\langle proof \rangle$

**lemma** *fold-Cons-rev*: $fold\ Cons\ xs = append\ (rev\ xs)$
$\langle proof \rangle$

**lemma** *rev-conv-fold* [*code*]: $rev\ xs = fold\ Cons\ xs\ []$
$\langle proof \rangle$

**lemma** *fold-append-concat-rev*: $fold\ append\ xss = append\ (concat\ (rev\ xss))$
$\langle proof \rangle$

**lemma** *fold-inject*:
**assumes**
$\bigwedge w\ x\ y\ z.\ f\ w\ x = f\ y\ z \longleftrightarrow w = y \wedge x = z$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq a$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq b$
**shows** $fold\ f\ xs\ a = fold\ f\ ys\ b \longleftrightarrow xs = ys \wedge a = b$
$\langle proof \rangle$

### *Finite-Set.fold* and *fold*

**lemma** (**in** *comp-fun-commute-on*) *fold-set-fold-remdups*:
**assumes** $set\ xs \subseteq S$
**shows** $Finite\text{-}Set.fold\ f\ y\ (set\ xs) = fold\ f\ (remdups\ xs)\ y$
$\langle proof \rangle$

**lemma** (**in** *comp-fun-idem-on*) *fold-set-fold*:
**assumes** *set xs* $\subseteq$ *S*
**shows** *Finite-Set.fold f y* (*set xs*) = *fold f xs y*
⟨*proof*⟩

**lemma** *union-set-fold* [*code*]: *set xs* $\cup$ *A* = *fold Set.insert xs A*
⟨*proof*⟩

**lemma** *union-coset-filter* [*code*]:
*List.coset xs* $\cup$ *A* = *List.coset* (*List.filter* ($\lambda x.$ $x \notin A$) *xs*)
⟨*proof*⟩

**lemma** *minus-set-fold* [*code*]: *A* − *set xs* = *fold Set.remove xs A*
⟨*proof*⟩

**lemma** *minus-coset-filter* [*code*]:
*A* − *List.coset xs* = *set* (*List.filter* ($\lambda x.$ $x \in A$) *xs*)
⟨*proof*⟩

**lemma** *inter-set-filter* [*code*]:
*A* $\cap$ *set xs* = *set* (*List.filter* ($\lambda x.$ $x \in A$) *xs*)
⟨*proof*⟩

**lemma** *inter-coset-fold* [*code*]:
*A* $\cap$ *List.coset xs* = *fold Set.remove xs A*
⟨*proof*⟩

**definition** *abort-empty-set* :: ‹('*a set* ⇒ '*a*) ⇒ '*a*›
**where** [*simp*]: ‹*abort-empty-set F* = *F* {}›

**declare** [[*code abort*: *abort-empty-set*]]

**lemma** (**in** *semilattice-set*) *set-empty-abort* [*code*]:
‹*F* (*set* []) = *abort-empty-set F*›
⟨*proof*⟩

**lemma** (**in** *semilattice-set*) *set-eq-fold* [*code*]:
‹*F* (*set* (*x* # *xs*)) = *fold f xs x*›
⟨*proof*⟩

**lemma** (**in** *complete-lattice*) *Inf-set-fold*:
*Inf* (*set xs*) = *fold inf xs top*
⟨*proof*⟩

**declare** *Inf-set-fold* [**where** '*a* = '*a set*, *code*]

**lemma** (**in** *complete-lattice*) *Sup-set-fold*:
*Sup* (*set xs*) = *fold sup xs bot*
⟨*proof*⟩

**declare** *Sup-set-fold* [**where** $'a = 'a$ *set*, *code*]

**lemma** (**in** *complete-lattice*) *INF-set-fold*:
$\bigsqcap (f \text{ ` } set\ xs) = fold\ (inf \circ f)\ xs\ top$
⟨*proof*⟩

**lemma** (**in** *complete-lattice*) *SUP-set-fold*:
$\bigsqcup (f \text{ ` } set\ xs) = fold\ (sup \circ f)\ xs\ bot$
⟨*proof*⟩

### 66.1.20 Fold variants: *foldr* and *foldl*

Correspondence

**lemma** *foldr-conv-fold* [*code-abbrev*]: $foldr\ f\ xs = fold\ f\ (rev\ xs)$
⟨*proof*⟩

**lemma** *foldl-conv-fold*: $foldl\ f\ s\ xs = fold\ (\lambda x\ s.\ f\ s\ x)\ xs\ s$
⟨*proof*⟩

**lemma** *foldr-conv-foldl*: — The "Third Duality Theorem" in Bird & Wadler:
$foldr\ f\ xs\ a = foldl\ (\lambda x\ y.\ f\ y\ x)\ a\ (rev\ xs)$
⟨*proof*⟩

**lemma** *foldl-conv-foldr*:
$foldl\ f\ a\ xs = foldr\ (\lambda x\ y.\ f\ y\ x)\ (rev\ xs)\ a$
⟨*proof*⟩

**lemma** *foldr-fold*:
$(\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow f\ y \circ f\ x = f\ x \circ f\ y)$
$\Longrightarrow foldr\ f\ xs = fold\ f\ xs$
⟨*proof*⟩

**lemma** *foldr-cong* [*fundef-cong*]:
$a = b \Longrightarrow l = k \Longrightarrow (\bigwedge a\ x.\ x \in set\ l \Longrightarrow f\ x\ a = g\ x\ a) \Longrightarrow foldr\ f\ l\ a = foldr\ g\ k\ b$
⟨*proof*⟩

**lemma** *foldl-cong* [*fundef-cong*]:
$a = b \Longrightarrow l = k \Longrightarrow (\bigwedge a\ x.\ x \in set\ l \Longrightarrow f\ a\ x = g\ a\ x) \Longrightarrow foldl\ f\ a\ l = foldl\ g\ b\ k$
⟨*proof*⟩

**lemma** *foldr-append* [*simp*]: $foldr\ f\ (xs\ @\ ys)\ a = foldr\ f\ xs\ (foldr\ f\ ys\ a)$
⟨*proof*⟩

**lemma** *foldl-append* [*simp*]: $foldl\ f\ a\ (xs\ @\ ys) = foldl\ f\ (foldl\ f\ a\ xs)\ ys$
⟨*proof*⟩

**lemma** *foldr-map* [*code-unfold*]: $foldr\ g\ (map\ f\ xs)\ a = foldr\ (g \circ f)\ xs\ a$
⟨*proof*⟩

**lemma** *foldr-filter*:
$foldr\ f\ (filter\ P\ xs) = foldr\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ id)\ xs$
⟨*proof*⟩

**lemma** *foldl-map* [*code-unfold*]:
$foldl\ g\ a\ (map\ f\ xs) = foldl\ (\lambda a\ x.\ g\ a\ (f\ x))\ a\ xs$
⟨*proof*⟩

**lemma** *concat-conv-foldr* [*code*]:
$concat\ xss = foldr\ append\ xss\ []$
⟨*proof*⟩

**lemma** *foldl-inject*:
**assumes**
$\bigwedge w\ x\ y\ z.\ f\ w\ x = f\ y\ z \longleftrightarrow w = y \wedge x = z$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq a$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq b$
**shows** $foldl\ f\ a\ xs = foldl\ f\ b\ ys \longleftrightarrow a = b \wedge xs = ys$
⟨*proof*⟩

**lemma** *foldr-inject*:
**assumes**
$\bigwedge w\ x\ y\ z.\ f\ w\ x = f\ y\ z \longleftrightarrow w = y \wedge x = z$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq a$ **and**
$\bigwedge x\ y.\ f\ x\ y \neq b$
**shows** $foldr\ f\ xs\ a = foldr\ f\ ys\ b \longleftrightarrow xs = ys \wedge a = b$
⟨*proof*⟩

### 66.1.21 *upt*

**lemma** *upt-rec*[*code*]: $[i..<j] = (if\ i<j\ then\ i\#[Suc\ i..<j]\ else\ [])$
— simp does not terminate!
⟨*proof*⟩

**lemmas** *upt-rec-numeral*[*simp*] = *upt-rec*[*of numeral m numeral n*] **for** *m n*

**lemma** *upt-conv-Nil* [*simp*]: $j \leq i \Longrightarrow [i..<j] = []$
⟨*proof*⟩

**lemma** *upt-eq-Nil-conv*[*simp*]: $([i..<j] = []) = (j = 0 \vee j \leq i)$
⟨*proof*⟩

**lemma** *upt-eq-Cons-conv*:
$([i..<j] = x\#xs) = (i < j \wedge i = x \wedge [i+1..<j] = xs)$
⟨*proof*⟩

**lemma** *upt-Suc-append*: $i \le j \Longrightarrow [i..<(Suc\ j)] = [i..<j]@[j]$
— Only needed if *upt-Suc* is deleted from the simpset.
⟨*proof*⟩

**lemma** *upt-conv-Cons*: $i < j \Longrightarrow [i..<j] = i \# [Suc\ i..<j]$
⟨*proof*⟩

**lemma** *upt-conv-Cons-Cons*: — no precondition
$m \# n \# ns = [m..<q] \longleftrightarrow n \# ns = [Suc\ m..<q]$
⟨*proof*⟩

**lemma** *upt-add-eq-append*: $i<=j \Longrightarrow [i..<j+k] = [i..<j]@[j..<j+k]$
— LOOPS as a simprule, since $j \le j$.
⟨*proof*⟩

**lemma** *length-upt* [*simp*]: $length\ [i..<j] = j - i$
⟨*proof*⟩

**lemma** *nth-upt* [*simp*]: $i + k < j \Longrightarrow [i..<j] ! k = i + k$
⟨*proof*⟩

**lemma** *hd-upt*[*simp*]: $i < j \Longrightarrow hd[i..<j] = i$
⟨*proof*⟩

**lemma** *tl-upt* [*simp*]: $tl\ [m..<n] = [Suc\ m..<n]$
⟨*proof*⟩

**lemma** *last-upt*[*simp*]: $i < j \Longrightarrow last[i..<j] = j - 1$
⟨*proof*⟩

**lemma** *take-upt* [*simp*]: $i+m \le n \Longrightarrow take\ m\ [i..<n] = [i..<i+m]$
⟨*proof*⟩

**lemma** *drop-upt*[*simp*]: $drop\ m\ [i..<j] = [i+m..<j]$
⟨*proof*⟩

**lemma** *map-Suc-upt*: $map\ Suc\ [m..<n] = [Suc\ m..<Suc\ n]$
⟨*proof*⟩

**lemma** *map-add-upt*: $map\ (\lambda i.\ i + n)\ [0..<m] = [n..<m + n]$
⟨*proof*⟩

**lemma** *nth-map-upt*: $i < n-m \Longrightarrow (map\ f\ [m..<n])\ !\ i = f(m+i)$
⟨*proof*⟩

**lemma** *map-decr-upt*: $map\ (\lambda n.\ n - Suc\ 0)\ [Suc\ m..<Suc\ n] = [m..<n]$
⟨*proof*⟩

**lemma** *map-upt-Suc*: $map\ f\ [0\ ..<\ Suc\ n] = f\ 0\ \#\ map\ (\lambda i.\ f\ (Suc\ i))\ [0\ ..<\ n]$

$\langle proof \rangle$

**lemma** *nth-take-lemma*:
$k \le length\ xs \Longrightarrow k \le length\ ys \Longrightarrow$
$(\bigwedge i.\ i < k \longrightarrow xs!i = ys!i) \Longrightarrow take\ k\ xs = take\ k\ ys$
$\langle proof \rangle$

**lemma** *nth-equalityI*:
$[\![length\ xs = length\ ys;\ \bigwedge i.\ i < length\ xs \Longrightarrow xs!i = ys!i]\!] \Longrightarrow xs = ys$
$\langle proof \rangle$

**lemma** *map-nth*:
$map\ (\lambda i.\ xs\ !\ i)\ [0..<length\ xs] = xs$
$\langle proof \rangle$

**lemma** *list-all2-antisym*:
$[\![\ (\bigwedge x\ y.\ [\![P\ x\ y;\ Q\ y\ x]\!] \Longrightarrow x = y);\ list\text{-}all2\ P\ xs\ ys;\ list\text{-}all2\ Q\ ys\ xs\ ]\!]$
$\Longrightarrow xs = ys$
$\langle proof \rangle$

**lemma** *take-equalityI*: $(\forall i.\ take\ i\ xs = take\ i\ ys) \Longrightarrow xs = ys$
— The famous take-lemma.
$\langle proof \rangle$

**lemma** *take-Cons'*:
$take\ n\ (x\ \#\ xs) = (if\ n = 0\ then\ []\ else\ x\ \#\ take\ (n - 1)\ xs)$
$\langle proof \rangle$

**lemma** *drop-Cons'*:
$drop\ n\ (x\ \#\ xs) = (if\ n = 0\ then\ x\ \#\ xs\ else\ drop\ (n - 1)\ xs)$
$\langle proof \rangle$

**lemma** *nth-Cons'*: $(x\ \#\ xs)!n = (if\ n = 0\ then\ x\ else\ xs!(n - 1))$
$\langle proof \rangle$

**lemma** *take-Cons-numeral* [*simp*]:
$take\ (numeral\ v)\ (x\ \#\ xs) = x\ \#\ take\ (numeral\ v - 1)\ xs$
$\langle proof \rangle$

**lemma** *drop-Cons-numeral* [*simp*]:
$drop\ (numeral\ v)\ (x\ \#\ xs) = drop\ (numeral\ v - 1)\ xs$
$\langle proof \rangle$

**lemma** *nth-Cons-numeral* [*simp*]:
$(x\ \#\ xs)\ !\ numeral\ v = xs\ !\ (numeral\ v - 1)$
$\langle proof \rangle$

**lemma** *map-upt-eqI*:
‹$map\ f\ [m..<n] = xs$› **if** ‹$length\ xs = n - m$›

‹$\bigwedge i.\ i < length\ xs \Longrightarrow xs\ !\ i = f\ (m + i)$›
⟨*proof*⟩

### 66.1.22 *upto*: interval-list on *int*

**function** *upto* :: *int* ⇒ *int* ⇒ *int list* (‹(‹*indent=1 notation=*‹*mixfix list interval*››[-../-])›) **where**
*upto i j* = (*if i* ≤ *j then i* # [*i*+*1*..*j*] *else* [])
⟨*proof*⟩
**termination**
⟨*proof*⟩

**declare** *upto.simps*[*simp del*]

**lemmas** *upto-rec-numeral* [*simp*] =
*upto.simps*[*of numeral m numeral n*]
*upto.simps*[*of numeral m* − *numeral n*]
*upto.simps*[*of* − *numeral m numeral n*]
*upto.simps*[*of* − *numeral m* − *numeral n*] **for** *m n*

**lemma** *upto-empty*[*simp*]: $j < i \Longrightarrow [i..j] = []$
⟨*proof*⟩

**lemma** *upto-single*[*simp*]: $[i..i] = [i]$
⟨*proof*⟩

**lemma** *upto-Nil*[*simp*]: $[i..j] = [] \longleftrightarrow j < i$
⟨*proof*⟩

**lemmas** *upto-Nil2*[*simp*] = *upto-Nil*[*THEN eq-iff-swap*]

**lemma** *upto-rec1*: $i \le j \Longrightarrow [i..j] = i\#[i+1..j]$
⟨*proof*⟩

**lemma** *upto-rec2*: $i \le j \Longrightarrow [i..j] = [i..j - 1]@[j]$
⟨*proof*⟩

**lemma** *length-upto*[*simp*]: *length* $[i..j] = nat(j - i + 1)$
⟨*proof*⟩

**lemma** *set-upto*[*simp*]: $set[i..j] = \{i..j\}$
⟨*proof*⟩

**lemma** *nth-upto*[*simp*]: $i + int\ k \le j \Longrightarrow [i..j]\ !\ k = i + int\ k$
⟨*proof*⟩

**lemma** *upto-split1*:
$i \le j \Longrightarrow j \le k \Longrightarrow [i..k] = [i..j-1]\ @\ [j..k]$
⟨*proof*⟩

**lemma** *upto-split2*:
$i \le j \Longrightarrow j \le k \Longrightarrow [i..k] = [i..j] @ [j+1..k]$
⟨*proof*⟩

**lemma** *upto-split3*: $[\![ i \le j; j \le k ]\!] \Longrightarrow [i..k] = [i..j-1] @ j \# [j+1..k]$
⟨*proof*⟩

Tail recursive version for code generation:

**definition** *upto-aux* :: *int* ⇒ *int* ⇒ *int list* ⇒ *int list* **where**
*upto-aux i j js* = $[i..j]$ @ *js*

**lemma** *upto-aux-rec* [*code*]:
*upto-aux i j js* = (*if j<i then js else upto-aux i* $(j - 1)$ $(j\#js)$)
⟨*proof*⟩

**lemma** *upto-code*[*code*]: $[i..j]$ = *upto-aux i j* []
⟨*proof*⟩

### 66.1.23 *successively*

**lemma** *successively-Cons*:
*successively P* (*x* # *xs*) ⟷ *xs* = [] ∨ *P x* (*hd xs*) ∧ *successively P xs*
⟨*proof*⟩

**lemma** *successively-cong* [*cong*]:
**assumes** ⋀*x y*. *x* ∈ *set xs* ⟹ *y* ∈ *set xs* ⟹ *P x y* ⟷ *Q x y xs* = *ys*
**shows** *successively P xs* ⟷ *successively Q ys*
⟨*proof*⟩

**lemma** *successively-append-iff*:
*successively P* (*xs* @ *ys*) ⟷
*successively P xs* ∧ *successively P ys* ∧
(*xs* = [] ∨ *ys* = [] ∨ *P* (*last xs*) (*hd ys*))
⟨*proof*⟩

**lemma** *successively-if-sorted-wrt*: *sorted-wrt P xs* ⟹ *successively P xs*
⟨*proof*⟩

**lemma** *successively-iff-sorted-wrt-strong*:
**assumes** ⋀*x y z*. *x* ∈ *set xs* ⟹ *y* ∈ *set xs* ⟹ *z* ∈ *set xs* ⟹
*P x y* ⟹ *P y z* ⟹ *P x z*
**shows** *successively P xs* ⟷ *sorted-wrt P xs*
⟨*proof*⟩

**lemma** *successively-conv-sorted-wrt*:
**assumes** *transp P*

**shows** *successively P xs* $\longleftrightarrow$ *sorted-wrt P xs*
⟨*proof*⟩

**lemma** *successively-rev* [*simp*]: *successively P* (*rev xs*) $\longleftrightarrow$ *successively* ($\lambda x\ y.\ P\ y\ x$) *xs*
⟨*proof*⟩

**lemma** *successively-map*: *successively P* (*map f xs*) $\longleftrightarrow$ *successively* ($\lambda x\ y.\ P\ (f\ x)\ (f\ y)$) *xs*
⟨*proof*⟩

**lemma** *successively-mono*:
**assumes** *successively P xs*
**assumes** $\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow P\ x\ y \Longrightarrow Q\ x\ y$
**shows** *successively Q xs*
⟨*proof*⟩

**lemma** *successively-altdef*:
*successively* = ($\lambda P.$ *rec-list True* ($\lambda x\ xs\ b.$ *case xs of* [] $\Rightarrow$ *True* | *y* # - $\Rightarrow$ $P\ x\ y \wedge b$))
⟨*proof*⟩

### 66.1.24 *distinct* and *remdups* and *remdups-adj*

**lemma** *distinct-tl*: *distinct xs* $\Longrightarrow$ *distinct* (*tl xs*)
⟨*proof*⟩

**lemma** *distinct-append* [*simp*]:
*distinct* (*xs* @ *ys*) = (*distinct xs* $\wedge$ *distinct ys* $\wedge$ *set xs* $\cap$ *set ys* = {})
⟨*proof*⟩

**lemma** *distinct-rev*[*simp*]: *distinct*(*rev xs*) = *distinct xs*
⟨*proof*⟩

**lemma** *set-remdups* [*simp*]: *set* (*remdups xs*) = *set xs*
⟨*proof*⟩

**lemma** *distinct-remdups* [*iff*]: *distinct* (*remdups xs*)
⟨*proof*⟩

**lemma** *distinct-remdups-id*: *distinct xs* $\Longrightarrow$ *remdups xs* = *xs*
⟨*proof*⟩

**lemma** *remdups-id-iff-distinct* [*simp*]: *remdups xs* = *xs* $\longleftrightarrow$ *distinct xs*
⟨*proof*⟩

**lemma** *finite-distinct-list*: *finite A* $\Longrightarrow$ $\exists xs.$ *set xs* = *A* $\wedge$ *distinct xs*
⟨*proof*⟩

**lemma** *remdups-eq-nil-iff* [*simp*]: $(remdups\ x = []) = (x = [])$
⟨*proof*⟩

**lemmas** *remdups-eq-nil-right-iff* [*simp*] = *remdups-eq-nil-iff*[*THEN eq-iff-swap*]

**lemma** *length-remdups-leq*[*iff*]: $length(remdups\ xs) \leq length\ xs$
⟨*proof*⟩

**lemma** *length-remdups-eq*[*iff*]:
  $(length\ (remdups\ xs) = length\ xs) = (remdups\ xs = xs)$
⟨*proof*⟩

**lemma** *remdups-filter*: $remdups(filter\ P\ xs) = filter\ P\ (remdups\ xs)$
⟨*proof*⟩

**lemma** *distinct-map*:
  $distinct(map\ f\ xs) = (distinct\ xs \land inj\text{-}on\ f\ (set\ xs))$
⟨*proof*⟩

**lemma** *distinct-map-filter*:
  $distinct\ (map\ f\ xs) \Longrightarrow distinct\ (map\ f\ (filter\ P\ xs))$
⟨*proof*⟩

**lemma** *distinct-filter* [*simp*]: $distinct\ xs \Longrightarrow distinct\ (filter\ P\ xs)$
⟨*proof*⟩

**lemma** *distinct-upt*[*simp*]: $distinct[i..{<}j]$
⟨*proof*⟩

**lemma** *distinct-upto*[*simp*]: $distinct[i..j]$
⟨*proof*⟩

**lemma** *distinct-take*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (take\ i\ xs)$
⟨*proof*⟩

**lemma** *distinct-drop*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (drop\ i\ xs)$
⟨*proof*⟩

**lemma** *distinct-list-update*:
  **assumes** $d$: $distinct\ xs$ **and** $a$: $a \notin set\ xs - \{xs!i\}$
  **shows** $distinct\ (xs[i:=a])$
⟨*proof*⟩

**lemma** *distinct-concat-rev*[*simp*]: $distinct\ (concat\ (rev\ xs)) = distinct\ (concat\ xs)$
⟨*proof*⟩

**lemma** *distinct-concat*:
  $[\![\ distinct\ xs;$
    $\bigwedge ys.\ ys \in set\ xs \Longrightarrow distinct\ ys;$

$\bigwedge ys\ zs.\ [\![\ ys \in set\ xs\ ;\ zs \in set\ xs\ ;\ ys \neq zs\ ]\!] \Longrightarrow set\ ys \cap set\ zs = \{\}$
$]\!] \Longrightarrow distinct\ (concat\ xs)$
⟨*proof*⟩

An iff-version of $[\![distinct\ ?xs;\ \bigwedge ys.\ ys \in set\ ?xs \Longrightarrow distinct\ ys;\ \bigwedge ys\ zs.\ [\![ys \in set\ ?xs;\ zs \in set\ ?xs;\ ys \neq zs]\!] \Longrightarrow set\ ys \cap set\ zs = \{\}]\!] \Longrightarrow distinct\ (concat\ ?xs)$ is available further down as *distinct-concat-iff*.

It is best to avoid the following indexed version of distinct, but sometimes it is useful.

**lemma** *distinct-conv-nth*: $distinct\ xs = (\forall\, i < size\ xs.\ \forall\, j < size\ xs.\ i \neq j \longrightarrow xs!i \neq xs!j)$
⟨*proof*⟩

**lemma** *nth-eq-iff-index-eq*:
$[\![\ distinct\ xs;\ i < length\ xs;\ j < length\ xs\ ]\!] \Longrightarrow (xs!i = xs!j) = (i = j)$
⟨*proof*⟩

**lemma** *distinct-Ex1*:
$distinct\ xs \Longrightarrow x \in set\ xs \Longrightarrow (\exists!i.\ i < length\ xs \land xs\ !\ i = x)$
⟨*proof*⟩

**lemma** *inj-on-nth*: $distinct\ xs \Longrightarrow \forall\, i \in I.\ i < length\ xs \Longrightarrow inj\text{-}on\ (nth\ xs)\ I$
⟨*proof*⟩

**lemma** *bij-betw-nth*:
**assumes** $distinct\ xs\ A = \{..{<}length\ xs\}\ B = set\ xs$
**shows** $bij\text{-}betw\ ((!)\ xs)\ A\ B$
⟨*proof*⟩

**lemma** *set-update-distinct*: $[\![\ distinct\ xs;\ \ n < length\ xs\ ]\!] \Longrightarrow$
$set(xs[n := x]) = insert\ x\ (set\ xs - \{xs!n\})$
⟨*proof*⟩

**lemma** *distinct-swap*[*simp*]: $[\![\ i < size\ xs;\ j < size\ xs]\!] \Longrightarrow$
$distinct(xs[i := xs!j,\ j := xs!i]) = distinct\ xs$
⟨*proof*⟩

**lemma** *set-swap*[*simp*]:
$[\![\ i < size\ xs;\ j < size\ xs\ ]\!] \Longrightarrow set(xs[i := xs!j,\ j := xs!i]) = set\ xs$
⟨*proof*⟩

**lemma** *distinct-card*: $distinct\ xs \Longrightarrow card\ (set\ xs) = size\ xs$
⟨*proof*⟩

**lemma** *card-distinct*: $card\ (set\ xs) = size\ xs \Longrightarrow distinct\ xs$
⟨*proof*⟩

**lemma** *distinct-length-filter*: $distinct\ xs \Longrightarrow length\ (filter\ P\ xs) = card\ (\{x.\ P\ x\}$

*Int set xs*)
⟨*proof*⟩

**lemma** *not-distinct-decomp*: $\neg$ *distinct ws* $\Longrightarrow \exists$ *xs ys zs y*. *ws* = *xs*@[*y*]@*ys*@[*y*]@*zs*
⟨*proof*⟩

**lemma** *not-distinct-conv-prefix*:
**defines** *dec as xs y ys* $\equiv$ *y* $\in$ *set xs* $\wedge$ *distinct xs* $\wedge$ *as* = *xs* @ *y* # *ys*
**shows** $\neg$*distinct as* $\longleftrightarrow$ ($\exists$ *xs y ys*. *dec as xs y ys*) (**is** *?L* = *?R*)
⟨*proof*⟩

**lemma** *distinct-product*:
*distinct xs* $\Longrightarrow$ *distinct ys* $\Longrightarrow$ *distinct* (*List.product xs ys*)
⟨*proof*⟩

**lemma** *distinct-product-lists*:
**assumes** $\forall$ *xs* $\in$ *set xss*. *distinct xs*
**shows** *distinct* (*product-lists xss*)
⟨*proof*⟩

**lemma** *length-remdups-concat*:
*length* (*remdups* (*concat xss*)) = *card* ($\bigcup$*xs*$\in$*set xss*. *set xs*)
⟨*proof*⟩

**lemma** *remdups-append2*:
*remdups* (*xs* @ *remdups ys*) = *remdups* (*xs* @ *ys*)
⟨*proof*⟩

**lemma** *length-remdups-card-conv*: *length*(*remdups xs*) = *card*(*set xs*)
⟨*proof*⟩

**lemma** *remdups-remdups*: *remdups* (*remdups xs*) = *remdups xs*
⟨*proof*⟩

**lemma** *distinct-butlast*:
**assumes** *distinct xs*
**shows** *distinct* (*butlast xs*)
⟨*proof*⟩

**lemma** *remdups-map-remdups*:
*remdups* (*map f* (*remdups xs*)) = *remdups* (*map f xs*)
⟨*proof*⟩

**lemma** *distinct-zipI1*:
**assumes** *distinct xs*
**shows** *distinct* (*zip xs ys*)
⟨*proof*⟩

**lemma** *distinct-zipI2*:

**assumes** *distinct ys*
**shows** *distinct* (*zip xs ys*)
⟨*proof*⟩

**lemma** *set-take-disj-set-drop-if-distinct*:
$distinct\ vs \Longrightarrow i \leq j \Longrightarrow set\ (take\ i\ vs) \cap set\ (drop\ j\ vs) = \{\}$
⟨*proof*⟩

**lemma** *distinct-singleton*: *distinct* [*x*] ⟨*proof*⟩

**lemma** *distinct-length-2-or-more*:
$distinct\ (a\ \#\ b\ \#\ xs) \longleftrightarrow (a \neq b \wedge distinct\ (a\ \#\ xs) \wedge distinct\ (b\ \#\ xs))$
⟨*proof*⟩

**lemma** *remdups-adj-altdef*: $(remdups\text{-}adj\ xs = ys) \longleftrightarrow$
$(\exists f{::}nat => nat.\ mono\ f \wedge f\ `\ \{0\ ..{<}\ size\ xs\} = \{0\ ..{<}\ size\ ys\}$
$\wedge (\forall i < size\ xs.\ xs!i = ys!(f\ i))$
$\wedge (\forall i.\ i + 1 < size\ xs \longrightarrow (xs!i = xs!(i+1) \longleftrightarrow f\ i = f(i+1))))$ (**is** $?L \longleftrightarrow$
$(\exists f.\ ?p\ f\ xs\ ys))$
⟨*proof*⟩

**lemma** *hd-remdups-adj*[*simp*]: *hd* (*remdups-adj xs*) = *hd xs*
⟨*proof*⟩

**lemma** *remdups-adj-Cons*: *remdups-adj* (*x* # *xs*) =
(*case remdups-adj xs of* [] ⇒ [*x*] | *y* # *xs* ⇒ *if x* = *y then y* # *xs else x* # *y* #
*xs*)
⟨*proof*⟩

**lemma** *remdups-adj-append-two*:
*remdups-adj* (*xs* @ [*x*,*y*]) = *remdups-adj* (*xs* @ [*x*]) @ (*if x* = *y then* [] *else* [*y*])
⟨*proof*⟩

**lemma** *remdups-adj-adjacent*:
$Suc\ i < length\ (remdups\text{-}adj\ xs) \Longrightarrow remdups\text{-}adj\ xs\ !\ i \neq remdups\text{-}adj\ xs\ !\ Suc\ i$
⟨*proof*⟩

**lemma** *remdups-adj-rev*[*simp*]: *remdups-adj* (*rev xs*) = *rev* (*remdups-adj xs*)
⟨*proof*⟩

**lemma** *remdups-adj-length*[*simp*]: $length\ (remdups\text{-}adj\ xs) \leq length\ xs$
⟨*proof*⟩

**lemma** *remdups-adj-length-ge1*[*simp*]: $xs \neq [] \Longrightarrow length\ (remdups\text{-}adj\ xs) \geq Suc$
$0$
⟨*proof*⟩

**lemma** *remdups-adj-Nil-iff* [*simp*]: *remdups-adj xs* = [] $\longleftrightarrow$ *xs* = []
⟨*proof*⟩

**lemma** *remdups-adj-set*[*simp*]: *set* (*remdups-adj xs*) = *set xs*
⟨*proof*⟩

**lemma** *last-remdups-adj* [*simp*]: *last* (*remdups-adj xs*) = *last xs*
⟨*proof*⟩

**lemma** *remdups-adj-Cons-alt*[*simp*]: *x* # *tl* (*remdups-adj* (*x* # *xs*)) = *remdups-adj* (*x* # *xs*)
⟨*proof*⟩

**lemma** *remdups-adj-distinct*: *distinct xs* $\Longrightarrow$ *remdups-adj xs* = *xs*
⟨*proof*⟩

**lemma** *remdups-adj-append*:
*remdups-adj* ($xs_1$ @ *x* # $xs_2$) = *remdups-adj* ($xs_1$ @ [*x*]) @ *tl* (*remdups-adj* (*x* # $xs_2$))
⟨*proof*⟩

**lemma** *remdups-adj-singleton*:
*remdups-adj xs* = [*x*] $\Longrightarrow$ *xs* = *replicate* (*length xs*) *x*
⟨*proof*⟩

**lemma** *remdups-adj-map-injective*:
**assumes** *inj f*
**shows** *remdups-adj* (*map f xs*) = *map f* (*remdups-adj xs*)
⟨*proof*⟩

**lemma** *remdups-adj-replicate*:
*remdups-adj* (*replicate n x*) = (*if n* = *0 then* [] *else* [*x*])
⟨*proof*⟩

**lemma** *remdups-upt* [*simp*]: *remdups* [*m*..<*n*] = [*m*..<*n*]
⟨*proof*⟩

**lemma** *successively-remdups-adjI*:
*successively P xs* $\Longrightarrow$ *successively P* (*remdups-adj xs*)
⟨*proof*⟩

**lemma** *successively-remdups-adj-iff*:
($\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *P x x*) $\Longrightarrow$
*successively P* (*remdups-adj xs*) $\longleftrightarrow$ *successively P xs*
⟨*proof*⟩

**lemma** *successively-conv-nth*:
*successively P xs* $\longleftrightarrow$ ($\forall$ *i*. *Suc i* < *length xs* $\longrightarrow$ *P* (*xs* ! *i*) (*xs* ! *Suc i*))
⟨*proof*⟩

**lemma** *successively-nth*: $successively\ P\ xs \Longrightarrow Suc\ i < length\ xs \Longrightarrow P\ (xs\ !\ i)\ (xs\ !\ Suc\ i)$
⟨*proof*⟩

**lemma** *distinct-adj-conv-nth*:
$distinct\text{-}adj\ xs \longleftrightarrow (\forall i.\ Suc\ i < length\ xs \longrightarrow xs\ !\ i \neq xs\ !\ Suc\ i)$
⟨*proof*⟩

**lemma** *distinct-adj-nth*: $distinct\text{-}adj\ xs \Longrightarrow Suc\ i < length\ xs \Longrightarrow xs\ !\ i \neq xs\ !\ Suc\ i$
⟨*proof*⟩

**lemma** *remdups-adj-Cons′*:
$remdups\text{-}adj\ (x\ \#\ xs) = x\ \#\ remdups\text{-}adj\ (dropWhile\ (\lambda y.\ y = x)\ xs)$
⟨*proof*⟩

**lemma** *remdups-adj-singleton-iff*:
$length\ (remdups\text{-}adj\ xs) = Suc\ 0 \longleftrightarrow xs \neq [] \wedge xs = replicate\ (length\ xs)\ (hd\ xs)$
⟨*proof*⟩

**lemma** *tl-remdups-adj*:
$ys \neq [] \Longrightarrow tl\ (remdups\text{-}adj\ ys) = remdups\text{-}adj\ (dropWhile\ (\lambda x.\ x = hd\ ys)\ (tl\ ys))$
⟨*proof*⟩

**lemma** *remdups-adj-append-dropWhile*:
$remdups\text{-}adj\ (xs\ @\ y\ \#\ ys) = remdups\text{-}adj\ (xs\ @\ [y])\ @\ remdups\text{-}adj\ (dropWhile\ (\lambda x.\ x = y)\ ys)$
⟨*proof*⟩

**lemma** *remdups-adj-append′*:
**assumes** $xs = [] \vee ys = [] \vee last\ xs \neq hd\ ys$
**shows** $remdups\text{-}adj\ (xs\ @\ ys) = remdups\text{-}adj\ xs\ @\ remdups\text{-}adj\ ys$
⟨*proof*⟩

**lemma** *remdups-adj-append′′*: $xs \neq []$
$\Longrightarrow remdups\text{-}adj\ (xs\ @\ ys) = remdups\text{-}adj\ xs\ @\ remdups\text{-}adj\ (dropWhile\ (\lambda y.\ y = last\ xs)\ ys)$
⟨*proof*⟩

**lemma** *remdups-filter-last*:
$last\ [x \leftarrow remdups\ xs.\ P\ x] = last\ [x \leftarrow xs.\ P\ x]$
⟨*proof*⟩

**lemma** *remdups-append*:
$set\ xs \subseteq set\ ys \Longrightarrow remdups\ (xs\ @\ ys) = remdups\ ys$
⟨*proof*⟩

**lemma** *remdups-concat*:
*remdups* (*concat* (*remdups xs*)) = *remdups* (*concat xs*)
⟨*proof*⟩

## 66.2 *distinct-adj*

**lemma** *distinct-adj-Nil* [*simp*]: *distinct-adj* []
**and** *distinct-adj-singleton* [*simp*]: *distinct-adj* [*x*]
**and** *distinct-adj-Cons-Cons* [*simp*]: *distinct-adj* (*x* # *y* # *xs*) ⟷ *x* ≠ *y* ∧ *distinct-adj* (*y* # *xs*)
⟨*proof*⟩

**lemma** *distinct-adj-Cons*: *distinct-adj* (*x* # *xs*) ⟷ *xs* = [] ∨ *x* ≠ *hd xs* ∧ *distinct-adj xs*
⟨*proof*⟩

**lemma** *distinct-adj-ConsD*: *distinct-adj* (*x* # *xs*) ⟹ *distinct-adj xs*
⟨*proof*⟩

**lemma** *distinct-adj-remdups-adj*[*simp*]: *distinct-adj* (*remdups-adj xs*)
⟨*proof*⟩

**lemma** *distinct-adj-altdef*: *distinct-adj xs* ⟷ *remdups-adj xs* = *xs*
⟨*proof*⟩

**lemma** *distinct-adj-rev* [*simp*]: *distinct-adj* (*rev xs*) ⟷ *distinct-adj xs*
⟨*proof*⟩

**lemma** *distinct-adj-append-iff*:
*distinct-adj* (*xs* @ *ys*) ⟷
*distinct-adj xs* ∧ *distinct-adj ys* ∧ (*xs* = [] ∨ *ys* = [] ∨ *last xs* ≠ *hd ys*)
⟨*proof*⟩

**lemma** *distinct-adj-appendD1* [*dest*]: *distinct-adj* (*xs* @ *ys*) ⟹ *distinct-adj xs*
**and** *distinct-adj-appendD2* [*dest*]: *distinct-adj* (*xs* @ *ys*) ⟹ *distinct-adj ys*
⟨*proof*⟩

**lemma** *distinct-adj-mapI*: *distinct-adj xs* ⟹ *inj-on f* (*set xs*) ⟹ *distinct-adj* (*map f xs*)
⟨*proof*⟩

**lemma** *distinct-adj-mapD*: *distinct-adj* (*map f xs*) ⟹ *distinct-adj xs*
⟨*proof*⟩

**lemma** *distinct-adj-map-iff*: *inj-on f* (*set xs*) ⟹ *distinct-adj* (*map f xs*) ⟷ *distinct-adj xs*
⟨*proof*⟩

**lemma** *distinct-adj-conv-length-remdups-adj*:

$distinct\text{-}adj\ xs \longleftrightarrow length\ (remdups\text{-}adj\ xs) = length\ xs$
⟨proof⟩

### 66.2.1 *insert*

**lemma** *in-set-insert* [*simp*]:
$x \in set\ xs \Longrightarrow List.insert\ x\ xs = xs$
⟨proof⟩

**lemma** *not-in-set-insert* [*simp*]:
$x \notin set\ xs \Longrightarrow List.insert\ x\ xs = x \#\ xs$
⟨proof⟩

**lemma** *insert-Nil* [*simp*]: $List.insert\ x\ [] = [x]$
⟨proof⟩

**lemma** *set-insert* [*simp*]: $set\ (List.insert\ x\ xs) = insert\ x\ (set\ xs)$
⟨proof⟩

**lemma** *distinct-insert* [*simp*]: $distinct\ (List.insert\ x\ xs) = distinct\ xs$
⟨proof⟩

**lemma** *insert-remdups*:
$List.insert\ x\ (remdups\ xs) = remdups\ (List.insert\ x\ xs)$
⟨proof⟩

### 66.2.2 *List.union*

This is all one should need to know about union:

**lemma** *set-union*[*simp*]: $set\ (List.union\ xs\ ys) = set\ xs \cup set\ ys$
⟨proof⟩

**lemma** *distinct-union*[*simp*]: $distinct(List.union\ xs\ ys) = distinct\ ys$
⟨proof⟩

### 66.2.3 *find*

**lemma** *find-None-iff*: $List.find\ P\ xs = None \longleftrightarrow \neg\ (\exists\, x.\ x \in set\ xs \wedge P\ x)$
⟨proof⟩

**lemmas** *find-None-iff2* = *find-None-iff*[*THEN eq-iff-swap*]

**lemma** *find-Some-iff*:
$List.find\ P\ xs = Some\ x \longleftrightarrow$
$(\exists\, i<length\ xs.\ P\ (xs!i) \wedge x = xs!i \wedge (\forall\, j<i.\ \neg\ P\ (xs!j)))$
⟨proof⟩

**lemmas** *find-Some-iff2* = *find-Some-iff*[*THEN eq-iff-swap*]

**lemma** *find-cong*[*fundef-cong*]:
**assumes** $xs = ys$ **and** $\bigwedge x.\ x \in set\ ys \Longrightarrow P\ x = Q\ x$
**shows** $List.find\ P\ xs = List.find\ Q\ ys$
⟨*proof*⟩

**lemma** *find-dropWhile*:
$List.find\ P\ xs = (case\ dropWhile\ (Not \circ P)\ xs$
$of\ [] \Rightarrow None$
$\mid x\ \#\ \text{-} \Rightarrow Some\ x)$
⟨*proof*⟩

### 66.2.4 *count-list*

This library is intentionally minimal. See the remark about multisets at the point above where *count-list* is defined.

**lemma** *count-list-append*[*simp*]: $count\text{-}list\ (xs\ @\ ys)\ x = count\text{-}list\ xs\ x + count\text{-}list\ ys\ x$
⟨*proof*⟩

**lemma** *count-list-0-iff*: $count\text{-}list\ xs\ x = 0 \longleftrightarrow x \notin set\ xs$
⟨*proof*⟩

**lemma** *count-notin*[*simp*]: $x \notin set\ xs \Longrightarrow count\text{-}list\ xs\ x = 0$
⟨*proof*⟩

**lemma** *count-le-length*: $count\text{-}list\ xs\ x \leq length\ xs$
⟨*proof*⟩

**lemma** *count-list-map-ge*: $count\text{-}list\ xs\ x \leq count\text{-}list\ (map\ f\ xs)\ (f\ x)$
⟨*proof*⟩

**lemma** *count-list-inj-map*:
$[\![\ inj\text{-}on\ f\ (set\ xs);\ x \in set\ xs\ ]\!] \Longrightarrow count\text{-}list\ (map\ f\ xs)\ (f\ x) = count\text{-}list\ xs\ x$
⟨*proof*⟩

**lemma** *count-list-map-conv*:
**assumes** $inj\ f$ **shows** $count\text{-}list\ (map\ f\ xs)\ (f\ x) = count\text{-}list\ xs\ x$
⟨*proof*⟩

**lemma** *count-list-rev*[*simp*]: $count\text{-}list\ (rev\ xs)\ x = count\text{-}list\ xs\ x$
⟨*proof*⟩

**lemma** *sum-count-set*:
$set\ xs \subseteq X \Longrightarrow finite\ X \Longrightarrow sum\ (count\text{-}list\ xs)\ X = length\ xs$
⟨*proof*⟩

**lemma** *count-list-Suc-split-first*:
**assumes** $count\text{-}list\ xs\ x = Suc\ n$
**shows** $\exists\ pref\ rest.\ xs = pref\ @\ x\ \#\ rest \land x \notin set\ pref \land count\text{-}list\ rest\ x = n$

⟨*proof*⟩

**lemma** *count-list-eq-length-filter*: $count\text{-}list\ xs\ y = length(filter\ ((=)\ y)\ xs)$
⟨*proof*⟩

**lemma** *split-list-cycles*:
$\exists\, pref\ xss.\ xs = pref\ @\ concat\ xss \wedge x \notin set\ pref \wedge (\forall\, ys \in set\ xss.\ \exists\, zs.\ ys = x\ \#\ zs)$
⟨*proof*⟩

### 66.2.5 *List.extract*

**lemma** *extract-None-iff*: $List.extract\ P\ xs = None \longleftrightarrow \neg\ (\exists\ x{\in}set\ xs.\ P\ x)$
⟨*proof*⟩

**lemma** *extract-SomeE*:
$List.extract\ P\ xs = Some\ (ys,\ y,\ zs) \Longrightarrow$
$xs = ys\ @\ y\ \#\ zs \wedge P\ y \wedge \neg\ (\exists\ y \in set\ ys.\ P\ y)$
⟨*proof*⟩

**lemma** *extract-Some-iff*:
$List.extract\ P\ xs = Some\ (ys,\ y,\ zs) \longleftrightarrow$
$xs = ys\ @\ y\ \#\ zs \wedge P\ y \wedge \neg\ (\exists\ y \in set\ ys.\ P\ y)$
⟨*proof*⟩

**lemma** *extract-Nil-code* [*code*]: $List.extract\ P\ [] = None$
⟨*proof*⟩

**lemma** *extract-Cons-code* [*code*]:
$List.extract\ P\ (x\ \#\ xs) = (if\ P\ x\ then\ Some\ ([],\ x,\ xs)\ else$
$(case\ List.extract\ P\ xs\ of$
$None \Rightarrow None\ |$
$Some\ (ys,\ y,\ zs) \Rightarrow Some\ (x\#ys,\ y,\ zs)))$
⟨*proof*⟩

### 66.2.6 *remove1*

**lemma** *count-list-remove1* [*simp*]:
$count\text{-}list\ (remove1\ a\ xs)\ b = count\text{-}list\ xs\ b - (if\ a{=}b\ then\ 1\ else\ 0)$
⟨*proof*⟩

**lemma** *remove1-append*:
$remove1\ x\ (xs\ @\ ys) =$
$(if\ x \in set\ xs\ then\ remove1\ x\ xs\ @\ ys\ else\ xs\ @\ remove1\ x\ ys)$
⟨*proof*⟩

**lemma** *remove1-commute*: $remove1\ x\ (remove1\ y\ zs) = remove1\ y\ (remove1\ x\ zs)$
⟨*proof*⟩

**lemma** *in-set-remove1* [*simp*]:

$a \neq b \Longrightarrow a \in set(remove1\ b\ xs) = (a \in set\ xs)$
⟨proof⟩

**lemma** *set-remove1-subset*: $set(remove1\ x\ xs) \subseteq set\ xs$
⟨proof⟩

**lemma** *set-remove1-eq* [*simp*]: $distinct\ xs \Longrightarrow set(remove1\ x\ xs) = set\ xs - \{x\}$
⟨proof⟩

**lemma** *length-remove1*:
$length(remove1\ x\ xs) = (if\ x \in set\ xs\ then\ length\ xs - 1\ else\ length\ xs)$
⟨proof⟩

**lemma** *remove1-filter-not*[*simp*]:
$\neg P\ x \Longrightarrow remove1\ x\ (filter\ P\ xs) = filter\ P\ xs$
⟨proof⟩

**lemma** *filter-remove1*:
$filter\ Q\ (remove1\ x\ xs) = remove1\ x\ (filter\ Q\ xs)$
⟨proof⟩

**lemma** *notin-set-remove1*[*simp*]: $x \notin set\ xs \Longrightarrow x \notin set(remove1\ y\ xs)$
⟨proof⟩

**lemma** *distinct-remove1*[*simp*]: $distinct\ xs \Longrightarrow distinct(remove1\ x\ xs)$
⟨proof⟩

**lemma** *remove1-remdups*:
$distinct\ xs \Longrightarrow remove1\ x\ (remdups\ xs) = remdups\ (remove1\ x\ xs)$
⟨proof⟩

**lemma** *remove1-idem*: $x \notin set\ xs \Longrightarrow remove1\ x\ xs = xs$
⟨proof⟩

**lemma** *remove1-split*:
$a \in set\ xs \Longrightarrow remove1\ a\ xs = ys \longleftrightarrow (\exists ls\ rs.\ xs = ls\ @\ a\ \#\ rs \wedge a \notin set\ ls \wedge ys = ls\ @\ rs)$
⟨proof⟩

**lemma** *foldr-fold-remove1*[*code-unfold*]: $foldr\ remove1 = fold\ remove1$
⟨proof⟩

### 66.2.7 *removeAll*

**lemma** *removeAll-filter-not-eq*:
$removeAll\ x = filter\ (\lambda y.\ x \neq y)$
⟨proof⟩

**lemma** *removeAll-append*[*simp*]:

$removeAll\ x\ (xs\ @\ ys) = removeAll\ x\ xs\ @\ removeAll\ x\ ys$
⟨proof⟩

**lemma** *removeAll-commute*: $removeAll\ x\ (removeAll\ y\ zs) = removeAll\ y\ (removeAll\ x\ zs)$
⟨proof⟩

**lemma** *set-removeAll*[*simp*]: $set(removeAll\ x\ xs) = set\ xs - \{x\}$
⟨proof⟩

**lemma** *removeAll-id*[*simp*]: $x \notin set\ xs \Longrightarrow removeAll\ x\ xs = xs$
⟨proof⟩

**lemma** *length-removeAll*:
$length(removeAll\ x\ xs) = length\ xs - count\text{-}list\ xs\ x$
⟨proof⟩

**lemma** *removeAll-filter-not*[*simp*]:
$\neg\ P\ x \Longrightarrow removeAll\ x\ (filter\ P\ xs) = filter\ P\ xs$
⟨proof⟩

**lemma** *distinct-removeAll*:
$distinct\ xs \Longrightarrow distinct\ (removeAll\ x\ xs)$
⟨proof⟩

**lemma** *distinct-remove1-removeAll*:
$distinct\ xs \Longrightarrow remove1\ x\ xs = removeAll\ x\ xs$
⟨proof⟩

**lemma** *map-removeAll-inj-on*: $inj\text{-}on\ f\ (insert\ x\ (set\ xs)) \Longrightarrow$
$map\ f\ (removeAll\ x\ xs) = removeAll\ (f\ x)\ (map\ f\ xs)$
⟨proof⟩

**lemma** *map-removeAll-inj*: $inj\ f \Longrightarrow$
$map\ f\ (removeAll\ x\ xs) = removeAll\ (f\ x)\ (map\ f\ xs)$
⟨proof⟩

**lemma** *length-removeAll-less-eq* [*simp*]:
$length\ (removeAll\ x\ xs) \leq length\ xs$
⟨proof⟩

**lemma** *length-removeAll-less* [*termination-simp*]:
$x \in set\ xs \Longrightarrow length\ (removeAll\ x\ xs) < length\ xs$
⟨proof⟩

**lemma** *distinct-concat-iff*: $distinct\ (concat\ xs) \longleftrightarrow$
$distinct\ (removeAll\ []\ xs)\ \wedge$
$(\forall\ ys.\ ys \in set\ xs \longrightarrow distinct\ ys)\ \wedge$
$(\forall\ ys\ zs.\ ys \in set\ xs \wedge zs \in set\ xs \wedge ys \neq zs \longrightarrow set\ ys \cap set\ zs = \{\})$

⟨*proof*⟩

**lemma** *foldr-fold-removeAll*[*code-unfold*]: *foldr removeAll* = *fold removeAll*
⟨*proof*⟩

### 66.2.8 *minus-list-mset*

The difference of two lists viewed as multisets. Conceptually, the result of *minus-list-mset* is only determined up to permutation, i.e. up to the multiset of elements. Thus this function comes into its own in connection with multisets where *mset*(*minus-list-mset xs ys*) = *mset xs* − *mset ys* is proved. Lemma *count-list-minus-list-mset* is the equivalent on the list level.

**lemma** *minus-list-mset-Nil2* [*simp*]: *minus-list-mset xs* [] = *xs*
⟨*proof*⟩

**lemma** *minus-list-mset-Cons2* [*simp*]: *minus-list-mset xs* (*y*#*ys*) = *remove1 y* (*minus-list-mset xs ys*)
⟨*proof*⟩

**lemma** *count-list-minus-list-mset*[*simp*]:
*count-list* (*minus-list-mset xs ys*) *a* = *count-list xs a* − *count-list ys a*
⟨*proof*⟩

**lemma** *minus-list-set-subset-minus-list-mset*: *set xs* − *set ys* ⊆ *set*(*minus-list-mset xs ys*)
⟨*proof*⟩

**lemma** *minus-list-mset-remove1-commute*:
*minus-list-mset* (*remove1 x xs*) *ys* = *remove1 x* (*minus-list-mset xs ys*)
⟨*proof*⟩

**lemma** *minus-list-mset-append* [*simp*]:
*minus-list-mset xs* (*ys*@*zs*) = *minus-list-mset* (*minus-list-mset xs ys*) *zs*
⟨*proof*⟩

**lemma** *minus-list-mset-Nil1* [*simp*]: *minus-list-mset* [] *xs* = []
⟨*proof*⟩

**lemma** *minus-list-mset-Cons1*: *minus-list-mset* (*x*#*xs*) *ys* =
(*if x* ∈ *set ys then minus-list-mset xs* (*remove1 x ys*) *else x* # (*minus-list-mset xs ys*))
⟨*proof*⟩

**lemma** *length-minus-list-mset*: *length*(*minus-list-mset xs ys*) ≤ *length xs*
⟨*proof*⟩

**lemma** *minus-list-mset-subset*:
*set* (*minus-list-mset xs ys*) ⊆ *set xs*

$\langle proof \rangle$

**lemma** *distinct-minus-list-mset*:
**assumes** *distinct xs*
**shows** *distinct* (*minus-list-mset xs ys*)
$\langle proof \rangle$

**lemma** *set-minus-list-mset-distinct*:
**assumes** *distinct xs* **shows** *set* (*minus-list-mset xs ys*) = *set xs* − *set ys*
$\langle proof \rangle$

### 66.2.9 *minus-list-set*

The difference of two lists viewed as sets. Conceptually, the result of *minus-list-set* is only determined up to the set of elements:

**lemma** *set-minus-list-set*[*simp*]: *set*(*minus-list-set xs ys*) = *set xs* − *set ys*
$\langle proof \rangle$

**lemma** *minus-list-set-Nil2* [*simp*]: *minus-list-set xs* [] = *xs*
$\langle proof \rangle$

**lemma** *minus-list-set-Cons2* [*simp*]: *minus-list-set xs* (*y*#*ys*) = *removeAll y* (*minus-list-set xs ys*)
$\langle proof \rangle$

**lemma** *minus-list-set-eq-filter*: *minus-list-set xs ys* = *filter* ($\lambda x.\ x \notin$ *set ys*) *xs*
$\langle proof \rangle$

**lemma** *minus-list-set-removeAll-commute*:
*minus-list-set* (*removeAll x xs*) *ys* = *removeAll x* (*minus-list-set xs ys*)
$\langle proof \rangle$

**lemma** *minus-list-set-Nil1* [*simp*]: *minus-list-set* [] *xs* = []
$\langle proof \rangle$

**lemma** *minus-list-set-Cons1*: *minus-list-set* (*x*#*xs*) *ys* =
(*if x* ∈ *set ys then minus-list-set xs ys else x* # (*minus-list-set xs ys*))
$\langle proof \rangle$

**lemma** *minus-list-set-append2* [*simp*]:
*minus-list-set xs* (*ys* @ *zs*) = *minus-list-set* (*minus-list-set xs ys*) *zs*
$\langle proof \rangle$

**lemma** *length-minus-list-set*: *length*(*minus-list-set xs ys*) ≤ *length xs*
$\langle proof \rangle$

**lemma** *distinct-minus-list-set*: *distinct xs* ⟹ *distinct* (*minus-list-set xs ys*)
$\langle proof \rangle$

### 66.2.10 *inter-list-set*

The intersection of two lists viewed as sets. Conceptually, the result of *inter-list-set* is only determined up to the set of elements:

**lemma** *set-inter-list-set*[*simp*]: $set(inter\text{-}list\text{-}set\ xs\ ys) = set\ xs \cap set\ ys$
⟨*proof*⟩

**lemma** *inter-list-set-Nil*[*simp*]: $inter\text{-}list\text{-}set\ []\ xs = []$
⟨*proof*⟩

**lemma** *inter-list-set-Cons*[*simp*]: $inter\text{-}list\text{-}set\ (x\#xs)\ ys =$
$(if\ x \in set\ ys\ then\ x\ \#\ inter\text{-}list\text{-}set\ xs\ ys\ else\ inter\text{-}list\text{-}set\ xs\ ys)$
⟨*proof*⟩

**lemma** *inter-list-set-Nil2*[*simp*]: $inter\text{-}list\text{-}set\ xs\ [] = []$
⟨*proof*⟩

**lemma** *distinct-inter-list-set*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (inter\text{-}list\text{-}set\ xs\ ys)$
⟨*proof*⟩

**lemma** *inter-list-set-append*[*simp*]:
$inter\text{-}list\text{-}set\ (xs\ @\ ys)\ zs = inter\text{-}list\text{-}set\ xs\ zs\ @\ inter\text{-}list\text{-}set\ ys\ zs$
⟨*proof*⟩

**lemma** *length-inter-list-set*: $length(inter\text{-}list\text{-}set\ xs\ ys) \leq length\ xs$
⟨*proof*⟩

### 66.2.11 *replicate*

**lemma** *length-replicate* [*simp*]: $length\ (replicate\ n\ x) = n$
⟨*proof*⟩

**lemma** *replicate-eqI*:
**assumes** $length\ xs = n$ **and** $\bigwedge y.\ y \in set\ xs \Longrightarrow y = x$
**shows** $xs = replicate\ n\ x$
⟨*proof*⟩

**lemma** *Ex-list-of-length*: $\exists\, xs.\ length\ xs = n$
⟨*proof*⟩

**lemma** *map-replicate* [*simp*]: $map\ f\ (replicate\ n\ x) = replicate\ n\ (f\ x)$
⟨*proof*⟩

**lemma** *map-replicate-const*:
$map\ (\lambda\ x.\ k)\ lst = replicate\ (length\ lst)\ k$
⟨*proof*⟩

**lemma** *replicate-app-Cons-same*:
$(replicate\ n\ x)\ @\ (x\ \#\ xs) = x\ \#\ replicate\ n\ x\ @\ xs$

⟨*proof*⟩

**lemma** *rev-replicate* [*simp*]: $rev\ (replicate\ n\ x) = replicate\ n\ x$
⟨*proof*⟩

**lemma** *replicate-add*: $replicate\ (n + m)\ x = replicate\ n\ x\ @\ replicate\ m\ x$
⟨*proof*⟩

Courtesy of Matthias Daum:

**lemma** *append-replicate-commute*:
$replicate\ n\ x\ @\ replicate\ k\ x = replicate\ k\ x\ @\ replicate\ n\ x$
⟨*proof*⟩

Courtesy of Andreas Lochbihler:

**lemma** *filter-replicate*:
$filter\ P\ (replicate\ n\ x) = (if\ P\ x\ then\ replicate\ n\ x\ else\ [])$
⟨*proof*⟩

**lemma** *hd-replicate* [*simp*]: $n \neq 0 \Longrightarrow hd\ (replicate\ n\ x) = x$
⟨*proof*⟩

**lemma** *tl-replicate* [*simp*]: $tl\ (replicate\ n\ x) = replicate\ (n - 1)\ x$
⟨*proof*⟩

**lemma** *last-replicate* [*simp*]: $n \neq 0 \Longrightarrow last\ (replicate\ n\ x) = x$
⟨*proof*⟩

**lemma** *nth-replicate*[*simp*]: $i < n \Longrightarrow (replicate\ n\ x)!i = x$
⟨*proof*⟩

Courtesy of Matthias Daum (2 lemmas):

**lemma** *take-replicate*[*simp*]: $take\ i\ (replicate\ k\ x) = replicate\ (min\ i\ k)\ x$
⟨*proof*⟩

**lemma** *drop-replicate*[*simp*]: $drop\ i\ (replicate\ k\ x) = replicate\ (k-i)\ x$
⟨*proof*⟩

**lemma** *set-replicate-Suc*: $set\ (replicate\ (Suc\ n)\ x) = \{x\}$
⟨*proof*⟩

**lemma** *set-replicate* [*simp*]: $n \neq 0 \Longrightarrow set\ (replicate\ n\ x) = \{x\}$
⟨*proof*⟩

**lemma** *set-replicate-conv-if*: $set\ (replicate\ n\ x) = (if\ n = 0\ then\ \{\}\ else\ \{x\})$
⟨*proof*⟩

**lemma** *in-set-replicate*[*simp*]: $(x \in set\ (replicate\ n\ y)) = (x = y \wedge n \neq 0)$
⟨*proof*⟩

**lemma** *card-set-1-iff-replicate*:
$card(set\ xs) = Suc\ 0 \longleftrightarrow xs \neq [] \wedge (\exists x.\ xs = replicate\ (length\ xs)\ x)$
⟨*proof*⟩

**lemma** *Ball-set-replicate*[*simp*]:
$(\forall x \in set(replicate\ n\ a).\ P\ x) = (P\ a \vee n{=}0)$
⟨*proof*⟩

**lemma** *Bex-set-replicate*[*simp*]:
$(\exists x \in set(replicate\ n\ a).\ P\ x) = (P\ a \wedge n{\neq}0)$
⟨*proof*⟩

**lemma** *replicate-append-same*:
$replicate\ i\ x\ @\ [x] = x\ \#\ replicate\ i\ x$
⟨*proof*⟩

**lemma** *map-replicate-trivial*:
$map\ (\lambda i.\ x)\ [0..{<}i] = replicate\ i\ x$
⟨*proof*⟩

**lemma** *concat-replicate-trivial*[*simp*]:
$concat\ (replicate\ i\ []) = []$
⟨*proof*⟩

**lemma** *concat-replicate-single*[*simp*]: $concat\ (replicate\ m\ [a]) = replicate\ m\ a$
⟨*proof*⟩

**lemma** *replicate-empty*[*simp*]: $(replicate\ n\ x = []) \longleftrightarrow n{=}0$
⟨*proof*⟩

**lemmas** *empty-replicate*[*simp*] = *replicate-empty*[*THEN eq-iff-swap*]

**lemma** *replicate-eq-replicate*[*simp*]:
$(replicate\ m\ x = replicate\ n\ y) \longleftrightarrow (m{=}n \wedge (m{\neq}0 \longrightarrow x{=}y))$
⟨*proof*⟩

**lemma** *takeWhile-replicate*[*simp*]:
$takeWhile\ P\ (replicate\ n\ x) = (if\ P\ x\ then\ replicate\ n\ x\ else\ [])$
⟨*proof*⟩

**lemma** *dropWhile-replicate*[*simp*]:
$dropWhile\ P\ (replicate\ n\ x) = (if\ P\ x\ then\ []\ else\ replicate\ n\ x)$
⟨*proof*⟩

**lemma** *replicate-length-filter*:
$replicate\ (length\ (filter\ (\lambda y.\ x = y)\ xs))\ x = filter\ (\lambda y.\ x = y)\ xs$
⟨*proof*⟩

**lemma** *comm-append-are-replicate*:

$xs\ @\ ys = ys\ @\ xs \Longrightarrow \exists\, m\ n\ zs.\ concat\ (replicate\ m\ zs) = xs \wedge concat\ (replicate\ n\ zs) = ys$
$\langle proof \rangle$

**lemma** *comm-append-is-replicate*:
**fixes** $xs\ ys :: 'a\ list$
**assumes** $xs \neq []\ ys \neq []$
**assumes** $xs\ @\ ys = ys\ @\ xs$
**shows** $\exists\, n\ zs.\ n > 1 \wedge concat\ (replicate\ n\ zs) = xs\ @\ ys$
$\langle proof \rangle$

**lemma** *Cons-replicate-eq*:
$x\ \#\ xs = replicate\ n\ y \longleftrightarrow x = y \wedge n > 0 \wedge xs = replicate\ (n - 1)\ x$
$\langle proof \rangle$

**lemma** *replicate-length-same*:
$(\forall\, y \in set\ xs.\ y = x) \Longrightarrow replicate\ (length\ xs)\ x = xs$
$\langle proof \rangle$

**lemma** *foldr-replicate* [*simp*]:
$foldr\ f\ (replicate\ n\ x) = f\ x\ \hat{}\hat{}\ n$
$\langle proof \rangle$

**lemma** *fold-replicate* [*simp*]:
$fold\ f\ (replicate\ n\ x) = f\ x\ \hat{}\hat{}\ n$
$\langle proof \rangle$

### 66.2.12 *enumerate*

**lemma** *enumerate-simps* [*simp*, *code*]:
$enumerate\ n\ [] = []$
$enumerate\ n\ (x\ \#\ xs) = (n,\ x)\ \#\ enumerate\ (Suc\ n)\ xs$
$\langle proof \rangle$

**lemma** *length-enumerate* [*simp*]:
$length\ (enumerate\ n\ xs) = length\ xs$
$\langle proof \rangle$

**lemma** *map-fst-enumerate* [*simp*]:
$map\ fst\ (enumerate\ n\ xs) = [n..<n + length\ xs]$
$\langle proof \rangle$

**lemma** *map-snd-enumerate* [*simp*]:
$map\ snd\ (enumerate\ n\ xs) = xs$
$\langle proof \rangle$

**lemma** *in-set-enumerate-eq*:
$p \in set\ (enumerate\ n\ xs) \longleftrightarrow n \leq fst\ p \wedge fst\ p < length\ xs + n \wedge nth\ xs\ (fst\ p - n) = snd\ p$

⟨*proof*⟩

**lemma** *nth-enumerate-eq*: $m < length\ xs \Longrightarrow enumerate\ n\ xs\ !\ m = (n + m,\ xs\ !\ m)$
⟨*proof*⟩

**lemma** *enumerate-replicate-eq*:
$enumerate\ n\ (replicate\ m\ a) = map\ (\lambda q.\ (q,\ a))\ [n..<n + m]$
⟨*proof*⟩

**lemma** *enumerate-Suc-eq*:
$enumerate\ (Suc\ n)\ xs = map\ (apfst\ Suc)\ (enumerate\ n\ xs)$
⟨*proof*⟩

**lemma** *distinct-enumerate* [*simp*]:
$distinct\ (enumerate\ n\ xs)$
⟨*proof*⟩

**lemma** *enumerate-append-eq*:
$enumerate\ n\ (xs\ @\ ys) = enumerate\ n\ xs\ @\ enumerate\ (n + length\ xs)\ ys$
⟨*proof*⟩

**lemma** *enumerate-map-upt*:
$enumerate\ n\ (map\ f\ [n..<m]) = map\ (\lambda k.\ (k,\ f\ k))\ [n..<m]$
⟨*proof*⟩

### 66.2.13 *rotate1* and *rotate*

**lemma** *rotate0*[*simp*]: $rotate\ 0 = id$
⟨*proof*⟩

**lemma** *rotate-Suc*[*simp*]: $rotate\ (Suc\ n)\ xs = rotate1(rotate\ n\ xs)$
⟨*proof*⟩

**lemma** *rotate-add*:
$rotate\ (m+n) = rotate\ m \circ rotate\ n$
⟨*proof*⟩

**lemma** *rotate-rotate*: $rotate\ m\ (rotate\ n\ xs) = rotate\ (m+n)\ xs$
⟨*proof*⟩

**lemma** *rotate1-map*: $rotate1\ (map\ f\ xs) = map\ f\ (rotate1\ xs)$
⟨*proof*⟩

**lemma** *rotate1-rotate-swap*: $rotate1\ (rotate\ n\ xs) = rotate\ n\ (rotate1\ xs)$
⟨*proof*⟩

**lemma** *rotate1-length01*[*simp*]: $length\ xs \leq 1 \Longrightarrow rotate1\ xs = xs$
⟨*proof*⟩

**lemma** *rotate-length01* [*simp*]: $length\ xs \leq 1 \Longrightarrow rotate\ n\ xs = xs$
⟨*proof*⟩

**lemma** *rotate1-hd-tl*: $xs \neq [] \Longrightarrow rotate1\ xs = tl\ xs\ @\ [hd\ xs]$
⟨*proof*⟩

**lemma** *rotate-drop-take*:
$rotate\ n\ xs = drop\ (n\ mod\ length\ xs)\ xs\ @\ take\ (n\ mod\ length\ xs)\ xs$
⟨*proof*⟩

**lemma** *rotate-conv-mod*: $rotate\ n\ xs = rotate\ (n\ mod\ length\ xs)\ xs$
⟨*proof*⟩

**lemma** *rotate-id*[*simp*]: $n\ mod\ length\ xs = 0 \Longrightarrow rotate\ n\ xs = xs$
⟨*proof*⟩

**lemma** *length-rotate1* [*simp*]: $length(rotate1\ xs) = length\ xs$
⟨*proof*⟩

**lemma** *length-rotate*[*simp*]: $length(rotate\ n\ xs) = length\ xs$
⟨*proof*⟩

**lemma** *distinct1-rotate*[*simp*]: $distinct(rotate1\ xs) = distinct\ xs$
⟨*proof*⟩

**lemma** *distinct-rotate*[*simp*]: $distinct(rotate\ n\ xs) = distinct\ xs$
⟨*proof*⟩

**lemma** *rotate-map*: $rotate\ n\ (map\ f\ xs) = map\ f\ (rotate\ n\ xs)$
⟨*proof*⟩

**lemma** *set-rotate1* [*simp*]: $set(rotate1\ xs) = set\ xs$
⟨*proof*⟩

**lemma** *set-rotate*[*simp*]: $set(rotate\ n\ xs) = set\ xs$
⟨*proof*⟩

**lemma** *rotate1-replicate*[*simp*]: $rotate1\ (replicate\ n\ a) = replicate\ n\ a$
⟨*proof*⟩

**lemma** *rotate1-is-Nil-conv*[*simp*]: $(rotate1\ xs = []) = (xs = [])$
⟨*proof*⟩

**lemma** *rotate-is-Nil-conv*[*simp*]: $(rotate\ n\ xs = []) = (xs = [])$
⟨*proof*⟩

**lemma** *rotate-rev*:
$rotate\ n\ (rev\ xs) = rev(rotate\ (length\ xs - (n\ mod\ length\ xs))\ xs)$

⟨*proof*⟩

**lemma** *hd-rotate-conv-nth*:
**assumes** $xs \neq []$ **shows** $hd(rotate\ n\ xs) = xs!(n\ mod\ length\ xs)$
⟨*proof*⟩

**lemma** *rotate-append*: $rotate\ (length\ l)\ (l\ @\ q) = q\ @\ l$
⟨*proof*⟩

**lemma** *nth-rotate*:
‹$rotate\ m\ xs\ !\ n = xs\ !\ ((m + n)\ mod\ length\ xs)$› **if** ‹$n < length\ xs$›
⟨*proof*⟩

**lemma** *nth-rotate1*:
‹$rotate1\ xs\ !\ n = xs\ !\ (Suc\ n\ mod\ length\ xs)$› **if** ‹$n < length\ xs$›
⟨*proof*⟩

**lemma** *inj-rotate1*: $inj\ rotate1$
⟨*proof*⟩

**lemma** *surj-rotate1*: $surj\ rotate1$
⟨*proof*⟩

**lemma** *bij-rotate1*: $bij\ (rotate1 :: 'a\ list \Rightarrow 'a\ list)$
⟨*proof*⟩

**lemma** *rotate1-fixpoint-card*: $rotate1\ xs = xs \Longrightarrow xs = [] \lor card(set\ xs) = 1$
⟨*proof*⟩

### 66.2.14 *nths* — a generalization of (!) to sets

**lemma** *nths-empty* [*simp*]: $nths\ xs\ \{\} = []$
⟨*proof*⟩

**lemma** *nths-nil* [*simp*]: $nths\ []\ A = []$
⟨*proof*⟩

**lemma** *nths-all*: $\forall\, i < length\ xs.\ i \in I \Longrightarrow nths\ xs\ I = xs$
⟨*proof*⟩

**lemma** *length-nths*:
$length\ (nths\ xs\ I) = card\{i.\ i < length\ xs \land i \in I\}$
⟨*proof*⟩

**lemma** *nths-shift-lemma-Suc*:
$map\ fst\ (filter\ (\lambda p.\ P(Suc(snd\ p)))\ (zip\ xs\ is)) =$
$map\ fst\ (filter\ (\lambda p.\ P(snd\ p))\ (zip\ xs\ (map\ Suc\ is)))$
⟨*proof*⟩

**lemma** *nths-shift-lemma*:
$map\ fst\ (filter\ (\lambda p.\ snd\ p \in A)\ (zip\ xs\ [i..<i + length\ xs])) =$
$map\ fst\ (filter\ (\lambda p.\ snd\ p + i \in A)\ (zip\ xs\ [0..<length\ xs]))$
$\langle proof \rangle$

**lemma** *nths-append*:
$nths\ (l\ @\ l')\ A = nths\ l\ A\ @\ nths\ l'\ \{j.\ j + length\ l \in A\}$
$\langle proof \rangle$

**lemma** *nths-Cons*:
$nths\ (x\ \#\ l)\ A = (if\ 0 \in A\ then\ [x]\ else\ [])\ @\ nths\ l\ \{j.\ Suc\ j \in A\}$
$\langle proof \rangle$

**lemma** *nths-map*: $nths\ (map\ f\ xs)\ I = map\ f\ (nths\ xs\ I)$
$\langle proof \rangle$

**lemma** *set-nths*: $set(nths\ xs\ I) = \{xs!i|i.\ i<size\ xs \wedge i \in I\}$
$\langle proof \rangle$

**lemma** *set-nths-subset*: $set(nths\ xs\ I) \subseteq set\ xs$
$\langle proof \rangle$

**lemma** *notin-set-nthsI*[*simp*]: $x \notin set\ xs \Longrightarrow x \notin set(nths\ xs\ I)$
$\langle proof \rangle$

**lemma** *in-set-nthsD*: $x \in set(nths\ xs\ I) \Longrightarrow x \in set\ xs$
$\langle proof \rangle$

**lemma** *nths-singleton* [*simp*]: $nths\ [x]\ A = (if\ 0 \in A\ then\ [x]\ else\ [])$
$\langle proof \rangle$

**lemma** *distinct-nthsI*[*simp*]: $distinct\ xs \Longrightarrow distinct\ (nths\ xs\ I)$
$\langle proof \rangle$

**lemma** *nths-upt-eq-take* [*simp*]: $nths\ l\ \{..<n\} = take\ n\ l$
$\langle proof \rangle$

**lemma** *nths-nths*: $nths\ (nths\ xs\ A)\ B = nths\ xs\ \{i \in A.\ \exists j \in B.\ card\ \{i' \in A.\ i' < i\} = j\}$
$\langle proof \rangle$

**lemma** *drop-eq-nths*: $drop\ n\ xs = nths\ xs\ \{i.\ i \geq n\}$
$\langle proof \rangle$

**lemma** *nths-drop*: $nths\ (drop\ n\ xs)\ I = nths\ xs\ ((+)\ n\ `\ I)$
$\langle proof \rangle$

**lemma** *filter-eq-nths*: $filter\ P\ xs = nths\ xs\ \{i.\ i<length\ xs \wedge P(xs!i)\}$
$\langle proof \rangle$

**lemma** *filter-in-nths*:
$distinct\ xs \Longrightarrow filter\ (\%x.\ x \in set\ (nths\ xs\ s))\ xs = nths\ xs\ s$
⟨*proof*⟩

### 66.2.15 *subseqs* and *List.n-lists*

**lemma** *length-subseqs*: $length\ (subseqs\ xs) = 2\ \hat{}\ length\ xs$
⟨*proof*⟩

**lemma** *subseqs-powset*: $set\ `\ set\ (subseqs\ xs) = Pow\ (set\ xs)$
⟨*proof*⟩

**lemma** *distinct-set-subseqs*:
**assumes** *distinct xs*
**shows** *distinct* (*map set* (*subseqs xs*))
⟨*proof*⟩

**lemma** *n-lists-Nil* [*simp*]: $List.n\text{-}lists\ n\ [] = (if\ n = 0\ then\ [[]]\ else\ [])$
⟨*proof*⟩

**lemma** *length-n-lists-elem*: $ys \in set\ (List.n\text{-}lists\ n\ xs) \Longrightarrow length\ ys = n$
⟨*proof*⟩

**lemma** *set-n-lists*: $set\ (List.n\text{-}lists\ n\ xs) = \{ys.\ length\ ys = n \wedge set\ ys \subseteq set\ xs\}$
⟨*proof*⟩

**lemma** *subseqs-refl*: $xs \in set\ (subseqs\ xs)$
⟨*proof*⟩

**lemma** *subset-subseqs*: $X \subseteq set\ xs \Longrightarrow X \in set\ `\ set\ (subseqs\ xs)$
⟨*proof*⟩

**lemma** *Cons-in-subseqsD*: $y\ \#\ ys \in set\ (subseqs\ xs) \Longrightarrow ys \in set\ (subseqs\ xs)$
⟨*proof*⟩

**lemma** *subseqs-distinctD*: $[\![\ ys \in set\ (subseqs\ xs);\ distinct\ xs\ ]\!] \Longrightarrow distinct\ ys$
⟨*proof*⟩

### 66.2.16 *splice*

**lemma** *splice-Nil2* [*simp*]: $splice\ xs\ [] = xs$
⟨*proof*⟩

**lemma** *length-splice*[*simp*]: $length(splice\ xs\ ys) = length\ xs + length\ ys$
⟨*proof*⟩

**lemma** *split-Nil-iff*[*simp*]: $splice\ xs\ ys = [] \longleftrightarrow xs = [] \wedge ys = []$
⟨*proof*⟩

**lemma** *splice-replicate*[*simp*]: $splice\ (replicate\ m\ x)\ (replicate\ n\ x) = replicate\ (m+n)\ x$
⟨*proof*⟩

### 66.2.17 *shuffles*

**lemma** *shuffles-commutes*: $shuffles\ xs\ ys = shuffles\ ys\ xs$
⟨*proof*⟩

**lemma** *Nil-in-shuffles*[*simp*]: $[] \in shuffles\ xs\ ys \longleftrightarrow xs = [] \wedge ys = []$
⟨*proof*⟩

**lemma** *shufflesE*:
$zs \in shuffles\ xs\ ys \Longrightarrow$
$(zs = xs \Longrightarrow ys = [] \Longrightarrow P) \Longrightarrow$
$(zs = ys \Longrightarrow xs = [] \Longrightarrow P) \Longrightarrow$
$(\bigwedge x\ xs'\ z\ zs'.\ xs = x\ \#\ xs' \Longrightarrow zs = z\ \#\ zs' \Longrightarrow x = z \Longrightarrow zs' \in shuffles\ xs'\ ys \Longrightarrow P) \Longrightarrow$
$(\bigwedge y\ ys'\ z\ zs'.\ ys = y\ \#\ ys' \Longrightarrow zs = z\ \#\ zs' \Longrightarrow y = z \Longrightarrow zs' \in shuffles\ xs\ ys' \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *Cons-in-shuffles-iff*:
$z\ \#\ zs \in shuffles\ xs\ ys \longleftrightarrow$
$(xs \neq [] \wedge hd\ xs = z \wedge zs \in shuffles\ (tl\ xs)\ ys\ \vee$
$ys \neq [] \wedge hd\ ys = z \wedge zs \in shuffles\ xs\ (tl\ ys))$
⟨*proof*⟩

**lemma** *splice-in-shuffles* [*simp*, *intro*]: $splice\ xs\ ys \in shuffles\ xs\ ys$
⟨*proof*⟩

**lemma** *Nil-in-shufflesI*: $xs = [] \Longrightarrow ys = [] \Longrightarrow [] \in shuffles\ xs\ ys$
⟨*proof*⟩

**lemma** *Cons-in-shuffles-leftI*: $zs \in shuffles\ xs\ ys \Longrightarrow z\ \#\ zs \in shuffles\ (z\ \#\ xs)\ ys$
⟨*proof*⟩

**lemma** *Cons-in-shuffles-rightI*: $zs \in shuffles\ xs\ ys \Longrightarrow z\ \#\ zs \in shuffles\ xs\ (z\ \#\ ys)$
⟨*proof*⟩

**lemma** *finite-shuffles* [*simp*, *intro*]: $finite\ (shuffles\ xs\ ys)$
⟨*proof*⟩

**lemma** *length-shuffles*: $zs \in shuffles\ xs\ ys \Longrightarrow length\ zs = length\ xs + length\ ys$
⟨*proof*⟩

**lemma** *set-shuffles*: $zs \in shuffles\ xs\ ys \Longrightarrow set\ zs = set\ xs \cup set\ ys$

$\langle proof \rangle$

**lemma** *distinct-disjoint-shuffles*:
**assumes** $distinct\ xs\ distinct\ ys\ set\ xs \cap set\ ys = \{\}\ zs \in shuffles\ xs\ ys$
**shows** $distinct\ zs$
$\langle proof \rangle$

**lemma** *Cons-shuffles-subset1*: $(\#)\ x\ `\ shuffles\ xs\ ys \subseteq shuffles\ (x\ \#\ xs)\ ys$
$\langle proof \rangle$

**lemma** *Cons-shuffles-subset2*: $(\#)\ y\ `\ shuffles\ xs\ ys \subseteq shuffles\ xs\ (y\ \#\ ys)$
$\langle proof \rangle$

**lemma** *filter-shuffles*:
$filter\ P\ `\ shuffles\ xs\ ys = shuffles\ (filter\ P\ xs)\ (filter\ P\ ys)$
$\langle proof \rangle$

**lemma** *filter-shuffles-disjoint1*:
**assumes** $set\ xs \cap set\ ys = \{\}\ zs \in shuffles\ xs\ ys$
**shows** $filter\ (\lambda x.\ x \in set\ xs)\ zs = xs$ (**is** $filter\ ?P\ \text{-} = \text{-}$)
**and** $filter\ (\lambda x.\ x \notin set\ xs)\ zs = ys$ (**is** $filter\ ?Q\ \text{-} = \text{-}$)
$\langle proof \rangle$

**lemma** *filter-shuffles-disjoint2*:
**assumes** $set\ xs \cap set\ ys = \{\}\ zs \in shuffles\ xs\ ys$
**shows** $filter\ (\lambda x.\ x \in set\ ys)\ zs = ys\ filter\ (\lambda x.\ x \notin set\ ys)\ zs = xs$
$\langle proof \rangle$

**lemma** *partition-in-shuffles*:
$xs \in shuffles\ (filter\ P\ xs)\ (filter\ (\lambda x.\ \neg P\ x)\ xs)$
$\langle proof \rangle$

**lemma** *inv-image-partition*:
**assumes** $\bigwedge x.\ x \in set\ xs \Longrightarrow P\ x\ \bigwedge y.\ y \in set\ ys \Longrightarrow \neg P\ y$
**shows** $partition\ P\ -`\ \{(xs,\ ys)\} = shuffles\ xs\ ys$
$\langle proof \rangle$

### 66.2.18 Transpose

**function** *transpose* **where**
$transpose\ []\ = []\ |$
$transpose\ ([]\ \#\ xss) = transpose\ xss\ |$
$transpose\ ((x\#xs)\ \#\ xss) =$
$(x\ \#\ [h.\ (h\#t) \leftarrow xss])\ \#\ transpose\ (xs\ \#\ [t.\ (h\#t) \leftarrow xss])$
$\langle proof \rangle$

**lemma** *transpose-aux-filter-head*:
$concat\ (map\ (case\text{-}list\ []\ (\lambda h\ t.\ [h]))\ xss) =$
$map\ (\lambda xs.\ hd\ xs)\ (filter\ (\lambda ys.\ ys \neq [])\ xss)$

⟨*proof*⟩

**lemma** *transpose-aux-filter-tail*:
$concat\ (map\ (case\text{-}list\ []\ (\lambda h\ t.\ [t]))\ xss) =$
$map\ (\lambda xs.\ tl\ xs)\ (filter\ (\lambda ys.\ ys \neq [])\ xss)$
⟨*proof*⟩

**lemma** *transpose-aux-max*:
$max\ (Suc\ (length\ xs))\ (foldr\ (\lambda xs.\ max\ (length\ xs))\ xss\ 0) =$
$Suc\ (max\ (length\ xs)\ (foldr\ (\lambda x.\ max\ (length\ x - Suc\ 0))\ (filter\ (\lambda ys.\ ys \neq [])$
$xss)\ 0))$
(**is** $max$ - $?foldB = Suc\ (max$ - $?foldA))$
⟨*proof*⟩

**termination** *transpose*
⟨*proof*⟩

**lemma** *transpose-empty*: $(transpose\ xs = []) \longleftrightarrow (\forall x \in set\ xs.\ x = [])$
⟨*proof*⟩

**lemma** *length-transpose*:
**fixes** $xs :: 'a\ list\ list$
**shows** $length\ (transpose\ xs) = foldr\ (\lambda xs.\ max\ (length\ xs))\ xs\ 0$
⟨*proof*⟩

**lemma** *nth-transpose*:
**fixes** $xs :: 'a\ list\ list$
**assumes** $i < length\ (transpose\ xs)$
**shows** $transpose\ xs\ !\ i = map\ (\lambda xs.\ xs\ !\ i)\ (filter\ (\lambda ys.\ i < length\ ys)\ xs)$
⟨*proof*⟩

**lemma** *transpose-map-map*:
$transpose\ (map\ (map\ f)\ xs) = map\ (map\ f)\ (transpose\ xs)$
⟨*proof*⟩

### 66.2.19 *min* and *arg-min*

**lemma** *min-list-Min*: $xs \neq [] \Longrightarrow min\text{-}list\ xs = Min\ (set\ xs)$
⟨*proof*⟩

**lemma** *f-arg-min-list-f*: $xs \neq [] \Longrightarrow f\ (arg\text{-}min\text{-}list\ f\ xs) = Min\ (f\ `\ (set\ xs))$
⟨*proof*⟩

**lemma** *arg-min-list-in*: $xs \neq [] \Longrightarrow arg\text{-}min\text{-}list\ f\ xs \in set\ xs$
⟨*proof*⟩

### 66.2.20 (In)finiteness

**lemma** *finite-list-length*: $finite\ \{xs::('a::finite)\ list.\ length\ xs = n\}$
⟨*proof*⟩

**lemma** *finite-maxlen*:
$finite\ (M::'a\ list\ set) \Longrightarrow \exists\, n.\ \forall\, s{\in}M.\ size\ s < n$
⟨*proof*⟩

**lemma** *lists-length-Suc-eq*:
$\{xs.\ set\ xs \subseteq A \land length\ xs = Suc\ n\} =$
$(\lambda(xs,\ n).\ n\#xs)$ ‘ $(\{xs.\ set\ xs \subseteq A \land length\ xs = n\} \times A)$
⟨*proof*⟩

**lemma**
**assumes** *finite A*
**shows** *finite-lists-length-eq*: $finite\ \{xs.\ set\ xs \subseteq A \land length\ xs = n\}$
**and** *card-lists-length-eq*: $card\ \{xs.\ set\ xs \subseteq A \land length\ xs = n\} = (card\ A)\widehat{\ }n$
⟨*proof*⟩

**lemma** *finite-lists-length-le*:
**assumes** *finite A* **shows** $finite\ \{xs.\ set\ xs \subseteq A \land length\ xs \leq n\}$
(**is** *finite ?S*)
⟨*proof*⟩

**lemma** *card-lists-length-le*:
**assumes** *finite A* **shows** $card\ \{xs.\ set\ xs \subseteq A \land length\ xs \leq n\} = (\sum i{\leq}n.\ card\ A\widehat{\ }i)$
⟨*proof*⟩

**lemma** *finite-subset-distinct*:
**assumes** *finite A*
**shows** $finite\ \{xs.\ set\ xs \subseteq A \land distinct\ xs\}$ (**is** *finite ?S*)
⟨*proof*⟩

**lemma** *card-lists-distinct-length-eq*:
**assumes** *finite A* $k \leq card\ A$
**shows** $card\ \{xs.\ length\ xs = k \land distinct\ xs \land set\ xs \subseteq A\} = \prod\{card\ A - k + 1\ ..\ card\ A\}$
⟨*proof*⟩

**lemma** *card-lists-distinct-length-eq'*:
**assumes** $k < card\ A$
**shows** $card\ \{xs.\ length\ xs = k \land distinct\ xs \land set\ xs \subseteq A\} = \prod\{card\ A - k + 1\ ..\ card\ A\}$
⟨*proof*⟩

**lemma** *infinite-UNIV-listI*: $\neg\ finite(UNIV::'a\ list\ set)$
⟨*proof*⟩

**lemma** *same-length-different*:
**assumes** $xs \neq ys$ **and** $length\ xs = length\ ys$
**shows** $\exists\, pre\ x\ xs'\ y\ ys'.\ x{\neq}y \land xs = pre\ @\ [x]\ @\ xs' \land ys = pre\ @\ [y]\ @\ ys'$

⟨*proof*⟩

## 66.3 Sorting

### 66.3.1 *sorted-wrt*

Sometimes the second equation in the definition of *sorted-wrt* is too aggressive because it relates each list element to *all* its successors. Then this equation should be removed and *sorted-wrt2-simps* should be added instead.

**lemma** *sorted-wrt1*: $sorted\text{-}wrt\ P\ [x] = True$
⟨*proof*⟩

**lemma** *sorted-wrt2*: $transp\ P \Longrightarrow sorted\text{-}wrt\ P\ (x\ \#\ y\ \#\ zs) = (P\ x\ y \wedge sorted\text{-}wrt\ P\ (y\ \#\ zs))$
⟨*proof*⟩

**lemmas** *sorted-wrt2-simps* = *sorted-wrt1 sorted-wrt2*

**lemma** *sorted-wrt-true* [*simp*]:
$sorted\text{-}wrt\ (\lambda\text{-}\ \text{-}.\ True)\ xs$
⟨*proof*⟩

**lemma** *sorted-wrt-append*:
$sorted\text{-}wrt\ P\ (xs\ @\ ys) \longleftrightarrow$
$sorted\text{-}wrt\ P\ xs \wedge sorted\text{-}wrt\ P\ ys \wedge (\forall x \in set\ xs.\ \forall y \in set\ ys.\ P\ x\ y)$
⟨*proof*⟩

**lemma** *sorted-wrt-map*:
$sorted\text{-}wrt\ R\ (map\ f\ xs) = sorted\text{-}wrt\ (\lambda x\ y.\ R\ (f\ x)\ (f\ y))\ xs$
⟨*proof*⟩

**lemma**
**assumes** *sorted-wrt f xs*
**shows** *sorted-wrt-take*[*simp*]: $sorted\text{-}wrt\ f\ (take\ n\ xs)$
**and** *sorted-wrt-drop*[*simp*]: $sorted\text{-}wrt\ f\ (drop\ n\ xs)$
⟨*proof*⟩

**lemma** *sorted-wrt-dropWhile*[*simp*]: $sorted\text{-}wrt\ R\ xs \Longrightarrow sorted\text{-}wrt\ R\ (dropWhile\ P\ xs)$
⟨*proof*⟩

**lemma** *sorted-wrt-takeWhile*[*simp*]: $sorted\text{-}wrt\ R\ xs \Longrightarrow sorted\text{-}wrt\ R\ (takeWhile\ P\ xs)$
⟨*proof*⟩

**lemma** *sorted-wrt-filter*:
$sorted\text{-}wrt\ f\ xs \Longrightarrow sorted\text{-}wrt\ f\ (filter\ P\ xs)$
⟨*proof*⟩

**lemma** *sorted-wrt-rev*:
$sorted\text{-}wrt\ P\ (rev\ xs) = sorted\text{-}wrt\ (\lambda x\ y.\ P\ y\ x)\ xs$
⟨*proof*⟩

**lemma** *sorted-wrt-mono-rel*:
$(\bigwedge x\ y.\ [\![\ x \in set\ xs;\ y \in set\ xs;\ P\ x\ y\ ]\!] \Longrightarrow Q\ x\ y) \Longrightarrow sorted\text{-}wrt\ P\ xs \Longrightarrow sorted\text{-}wrt\ Q\ xs$
⟨*proof*⟩

**lemma** *sorted-wrt01*: $length\ xs \leq 1 \Longrightarrow sorted\text{-}wrt\ P\ xs$
⟨*proof*⟩

**lemma** *sorted-wrt-iff-nth-less*:
$sorted\text{-}wrt\ P\ xs = (\forall\ i\ j.\ i < j \longrightarrow j < length\ xs \longrightarrow P\ (xs\ !\ i)\ (xs\ !\ j))$
⟨*proof*⟩

**lemma** *sorted-wrt-nth-less*:
$[\![\ sorted\text{-}wrt\ P\ xs;\ i < j;\ j < length\ xs\ ]\!] \Longrightarrow P\ (xs\ !\ i)\ (xs\ !\ j)$
⟨*proof*⟩

**lemma** *sorted-wrt-iff-nth-Suc-transp*: **assumes** *transp P*
**shows** $sorted\text{-}wrt\ P\ xs \longleftrightarrow (\forall\ i.\ \ Suc\ i < length\ xs \longrightarrow P\ (xs!i)\ (xs!(Suc\ i)))$ (**is** $?L = ?R$)
⟨*proof*⟩

**lemma** *sorted-wrt-upt*[*simp*]: $sorted\text{-}wrt\ (<)\ [m..{<}n]$
⟨*proof*⟩

**lemma** *sorted-wrt-upto*[*simp*]: $sorted\text{-}wrt\ (<)\ [i..j]$
⟨*proof*⟩

Each element is greater or equal to its index:

**lemma** *sorted-wrt-less-idx*:
$sorted\text{-}wrt\ (<)\ ns \Longrightarrow i < length\ ns \Longrightarrow i \leq ns!i$
⟨*proof*⟩

### 66.3.2 *sorted*

**context** *linorder*
**begin**

Sometimes the second equation in the definition of *sorted* is too aggressive because it relates each list element to *all* its successors. Then this equation should be removed and *sorted2-simps* should be added instead. Executable code is one such use case.

**lemma** *sorted0*: $sorted\ [] = True$
⟨*proof*⟩

**lemma** *sorted1*: $sorted\ [x] = True$

$\langle proof \rangle$

**lemma** *sorted2*: $sorted\ (x\ \#\ y\ \#\ zs) = (x \leq y \wedge sorted\ (y\ \#\ zs))$
$\langle proof \rangle$

**lemmas** *sorted2-simps* = *sorted1 sorted2*

**lemma** *sorted-append*:
$sorted\ (xs@ys) = (sorted\ xs \wedge sorted\ ys \wedge (\forall x \in set\ xs.\ \forall y \in set\ ys.\ x \leq y))$
$\langle proof \rangle$

**lemma** *sorted-map*:
$sorted\ (map\ f\ xs) = sorted\text{-}wrt\ (\lambda x\ y.\ f\ x \leq f\ y)\ xs$
$\langle proof \rangle$

**lemma** *sorted01*: $length\ xs \leq 1 \Longrightarrow sorted\ xs$
$\langle proof \rangle$

**lemma** *sorted-tl*:
$sorted\ xs \Longrightarrow sorted\ (tl\ xs)$
$\langle proof \rangle$

**lemma** *sorted-iff-nth-mono-less*:
$sorted\ xs = (\forall i\ j.\ i < j \longrightarrow j < length\ xs \longrightarrow xs\ !\ i \leq xs\ !\ j)$
$\langle proof \rangle$

**lemma** *sorted-iff-nth-mono*:
$sorted\ xs = (\forall i\ j.\ i \leq j \longrightarrow j < length\ xs \longrightarrow xs\ !\ i \leq xs\ !\ j)$
$\langle proof \rangle$

**lemma** *sorted-nth-mono*:
$sorted\ xs \Longrightarrow i \leq j \Longrightarrow j < length\ xs \Longrightarrow xs!i \leq xs!j$
$\langle proof \rangle$

**lemma** *sorted-iff-nth-Suc*:
$sorted\ xs \longleftrightarrow (\forall i.\ \ Suc\ i < length\ xs \longrightarrow xs!i \leq xs!(Suc\ i))$
$\langle proof \rangle$

**lemma** *sorted-rev-nth-mono*:
$sorted\ (rev\ xs) \Longrightarrow i \leq j \Longrightarrow j < length\ xs \Longrightarrow xs!j \leq xs!i$
$\langle proof \rangle$

**lemma** *sorted-rev-iff-nth-mono*:
$sorted\ (rev\ xs) \longleftrightarrow (\forall\ \ i\ j.\ i \leq j \longrightarrow j < length\ xs \longrightarrow xs!j \leq xs!i)$ (**is** $?L = ?R$)
$\langle proof \rangle$

**lemma** *sorted-rev-iff-nth-Suc*:
$sorted\ (rev\ xs) \longleftrightarrow (\forall i.\ \ Suc\ i < length\ xs \longrightarrow xs!(Suc\ i) \leq xs!i)$
$\langle proof \rangle$

**lemma** *sorted-map-remove1*:
$sorted\ (map\ f\ xs) \Longrightarrow sorted\ (map\ f\ (remove1\ x\ xs))$
⟨*proof*⟩

**lemma** *sorted-remove1*: $sorted\ xs \Longrightarrow sorted\ (remove1\ a\ xs)$
⟨*proof*⟩

**lemma** *sorted-butlast*:
**assumes** *sorted xs*
**shows** *sorted* (*butlast xs*)
⟨*proof*⟩

**lemma** *sorted-replicate* [*simp*]: *sorted*(*replicate n x*)
⟨*proof*⟩

**lemma** *sorted-remdups*[*simp*]:
$sorted\ xs \Longrightarrow sorted\ (remdups\ xs)$
⟨*proof*⟩

**lemma** *sorted-remdups-adj*[*simp*]:
$sorted\ xs \Longrightarrow sorted\ (remdups\text{-}adj\ xs)$
⟨*proof*⟩

**lemma** *sorted-nths*: $sorted\ xs \Longrightarrow sorted\ (nths\ xs\ I)$
⟨*proof*⟩

**lemma** *sorted-distinct-set-unique*:
**assumes** *sorted xs distinct xs sorted ys distinct ys* $set\ xs = set\ ys$
**shows** $xs = ys$
⟨*proof*⟩

**lemma** *map-sorted-distinct-set-unique*:
**assumes** *inj-on f* $(set\ xs \cup set\ ys)$
**assumes** *sorted* (*map f xs*) *distinct* (*map f xs*)
*sorted* (*map f ys*) *distinct* (*map f ys*)
**assumes** $set\ xs = set\ ys$
**shows** $xs = ys$
⟨*proof*⟩

**lemma** *sorted-dropWhile*: $sorted\ xs \Longrightarrow sorted\ (dropWhile\ P\ xs)$
⟨*proof*⟩

**lemma** *sorted-takeWhile*: $sorted\ xs \Longrightarrow sorted\ (takeWhile\ P\ xs)$
⟨*proof*⟩

**lemma** *sorted-filter*:
$sorted\ (map\ f\ xs) \Longrightarrow sorted\ (map\ f\ (filter\ P\ xs))$
⟨*proof*⟩

**lemma** *foldr-max-sorted*:
**assumes** *sorted* (*rev xs*)
**shows** *foldr max xs y* = (*if xs* = [] *then y else max* (*xs* ! *0*) *y*)
⟨*proof*⟩

**lemma** *filter-equals-takeWhile-sorted-rev*:
**assumes** *sorted*: *sorted* (*rev* (*map f xs*))
**shows** *filter* ($\lambda x.\ t < f\ x$) *xs* = *takeWhile* ($\lambda\ x.\ t < f\ x$) *xs*
(**is** *filter ?P xs* = *?tW*)
⟨*proof*⟩

**lemma** *sorted-map-same*:
*sorted* (*map f* (*filter* ($\lambda x.\ f\ x = g\ xs$) *xs*))
⟨*proof*⟩

**lemma** *sorted-same*:
*sorted* (*filter* ($\lambda x.\ x = g\ xs$) *xs*)
⟨*proof*⟩

**end**

**lemma** *sorted-upt*[*simp*]: *sorted* [*m*..<*n*]
⟨*proof*⟩

**lemma** *sorted-upto*[*simp*]: *sorted* [*m*..*n*]
⟨*proof*⟩

### 66.3.3 Sorting functions

Currently it is not shown that *sort* returns a permutation of its input because the nicest proof is via multisets, which are not part of Main. Alternatively one could define a function that counts the number of occurrences of an element in a list and use that instead of multisets to state the correctness property.

**context** *linorder*
**begin**

**lemma** *set-insort-key*:
*set* (*insort-key f x xs*) = *insert x* (*set xs*)
⟨*proof*⟩

**lemma** *length-insort* [*simp*]:
*length* (*insort-key f x xs*) = *Suc* (*length xs*)
⟨*proof*⟩

**lemma** *insort-key-left-comm*:
**assumes** $f\ x \neq f\ y$

**shows** *insort-key f y* (*insort-key f x xs*) = *insort-key f x* (*insort-key f y xs*)
⟨*proof*⟩

**lemma** *insort-left-comm*:
*insort x* (*insort y xs*) = *insort y* (*insort x xs*)
⟨*proof*⟩

**lemma** *comp-fun-commute-insort*: *comp-fun-commute insort*
⟨*proof*⟩

**lemma** *sort-key-simps* [*simp*]:
*sort-key f* [] = []
*sort-key f* (*x*#*xs*) = *insort-key f x* (*sort-key f xs*)
⟨*proof*⟩

**lemma** *sort-key-conv-fold*:
**assumes** *inj-on f* (*set xs*)
**shows** *sort-key f xs* = *fold* (*insort-key f*) *xs* []
⟨*proof*⟩

**lemma** *sort-conv-fold*:
*sort xs* = *fold insort xs* []
⟨*proof*⟩

**lemma** *length-sort*[*simp*]: *length* (*sort-key f xs*) = *length xs*
⟨*proof*⟩

**lemma** *set-sort*[*simp*]: *set*(*sort-key f xs*) = *set xs*
⟨*proof*⟩

**lemma** *distinct-insort*: *distinct* (*insort-key f x xs*) = ($x \notin$ *set xs* $\wedge$ *distinct xs*)
⟨*proof*⟩

**lemma** *distinct-insort-key*:
*distinct* (*map f* (*insort-key f x xs*)) = (*f x* $\notin$ *f* ‘ *set xs* $\wedge$ (*distinct* (*map f xs*)))
⟨*proof*⟩

**lemma** *distinct-sort*[*simp*]: *distinct* (*sort-key f xs*) = *distinct xs*
⟨*proof*⟩

**lemma** *sorted-insort-key*: *sorted* (*map f* (*insort-key f x xs*)) = *sorted* (*map f xs*)
⟨*proof*⟩

**lemma** *sorted-insort*: *sorted* (*insort x xs*) = *sorted xs*
⟨*proof*⟩

**theorem** *sorted-sort-key* [*simp*]: *sorted* (*map f* (*sort-key f xs*))
⟨*proof*⟩

**theorem** *sorted-sort* [*simp*]: *sorted* (*sort xs*)
⟨*proof*⟩

**lemma** *insort-not-Nil* [*simp*]:
*insort-key f a xs* $\neq$ []
⟨*proof*⟩

**lemma** *insort-is-Cons*: $\forall x \in$ *set xs*. *f a* $\leq$ *f x* $\Longrightarrow$ *insort-key f a xs* = *a* # *xs*
⟨*proof*⟩

**lemma** *sort-key-id-if-sorted*: *sorted* (*map f xs*) $\Longrightarrow$ *sort-key f xs* = *xs*
⟨*proof*⟩

Subsumed by *sorted* (*map ?f ?xs*) $\Longrightarrow$ *sort-key ?f ?xs* = *?xs* but easier to find:

**lemma** *sorted-sort-id*: *sorted xs* $\Longrightarrow$ *sort xs* = *xs*
⟨*proof*⟩

**lemma** *sort-replicate* [*simp*]: *sort* (*replicate n x*) = *replicate n x*
⟨*proof*⟩

**lemma** *insort-key-remove1*:
**assumes** *a* $\in$ *set xs* **and** *sorted* (*map f xs*) **and** *hd* (*filter* ($\lambda x$. *f a* = *f x*) *xs*) = *a*
**shows** *insort-key f a* (*remove1 a xs*) = *xs*
⟨*proof*⟩

**lemma** *insort-remove1*:
**assumes** *a* $\in$ *set xs* **and** *sorted xs*
**shows** *insort a* (*remove1 a xs*) = *xs*
⟨*proof*⟩

**lemma** *finite-sorted-distinct-unique*:
**assumes** *finite A* **shows** $\exists!$*xs*. *set xs* = *A* $\wedge$ *sorted xs* $\wedge$ *distinct xs*
⟨*proof*⟩

**lemma** *insort-insert-key-triv*:
*f x* $\in$ *f* ‘ *set xs* $\Longrightarrow$ *insort-insert-key f x xs* = *xs*
⟨*proof*⟩

**lemma** *insort-insert-triv*:
*x* $\in$ *set xs* $\Longrightarrow$ *insort-insert x xs* = *xs*
⟨*proof*⟩

**lemma** *insort-insert-insort-key*:
*f x* $\notin$ *f* ‘ *set xs* $\Longrightarrow$ *insort-insert-key f x xs* = *insort-key f x xs*
⟨*proof*⟩

**lemma** *insort-insert-insort*:
*x* $\notin$ *set xs* $\Longrightarrow$ *insort-insert x xs* = *insort x xs*

⟨*proof*⟩

**lemma** *set-insort-insert*:
*set* (*insort-insert* $x$ *xs*) = *insert* $x$ (*set xs*)
⟨*proof*⟩

**lemma** *distinct-insort-insert*:
**assumes** *distinct xs*
**shows** *distinct* (*insort-insert-key f x xs*)
⟨*proof*⟩

**lemma** *sorted-insort-insert-key*:
**assumes** *sorted* (*map f xs*)
**shows** *sorted* (*map f* (*insort-insert-key f x xs*))
⟨*proof*⟩

**lemma** *sorted-insort-insert*:
**assumes** *sorted xs*
**shows** *sorted* (*insort-insert x xs*)
⟨*proof*⟩

**lemma** *filter-insort-triv*:
$\neg$ *P x* $\Longrightarrow$ *filter P* (*insort-key f x xs*) = *filter P xs*
⟨*proof*⟩

**lemma** *filter-insort*:
*sorted* (*map f xs*) $\Longrightarrow$ *P x* $\Longrightarrow$ *filter P* (*insort-key f x xs*) = *insort-key f x* (*filter P xs*)
⟨*proof*⟩

**lemma** *filter-sort*:
*filter P* (*sort-key f xs*) = *sort-key f* (*filter P xs*)
⟨*proof*⟩

**lemma** *remove1-insort-key* [*simp*]:
*remove1 x* (*insort-key f x xs*) = *xs*
⟨*proof*⟩

**end**

**lemma** *sort-upt* [*simp*]: *sort* [*m*..<*n*] = [*m*..<*n*]
⟨*proof*⟩

**lemma** *sort-upto* [*simp*]: *sort* [*i*..*j*] = [*i*..*j*]
⟨*proof*⟩

**lemma** *sorted-find-Min*:
*sorted xs* $\Longrightarrow$ $\exists\, x \in$ *set xs*. *P x* $\Longrightarrow$ *List.find P xs* = *Some* (*Min* {*x*∈*set xs*. *P x*})
⟨*proof*⟩

**lemma** *sorted-enumerate* [*simp*]: *sorted* (*map fst* (*enumerate n xs*))
⟨*proof*⟩

**lemma** *sorted-insort-is-snoc*: *sorted xs* $\Longrightarrow \forall x \in$ *set xs*. $a \geq x \Longrightarrow$ *insort a xs* = *xs* @ [*a*]
⟨*proof*⟩

Stability of *sort-key*:

**lemma** *sort-key-stable*: *filter* ($\lambda y.\ f\ y = k$) (*sort-key f xs*) = *filter* ($\lambda y.\ f\ y = k$) *xs*
⟨*proof*⟩

**corollary** *stable-sort-key-sort-key*: *stable-sort-key sort-key*
⟨*proof*⟩

**lemma** *sort-key-const*: *sort-key* ($\lambda x.\ c$) *xs* = *xs*
⟨*proof*⟩

### 66.3.4 *transpose* on sorted lists

**lemma** *sorted-transpose*[*simp*]: *sorted* (*rev* (*map length* (*transpose xs*)))
⟨*proof*⟩

**lemma** *transpose-max-length*:
*foldr* ($\lambda xs.$ *max* (*length xs*)) (*transpose xs*) *0* = *length* (*filter* ($\lambda x.\ x \neq []$) *xs*)
(**is** *?L* = *?R*)
⟨*proof*⟩

**lemma** *length-transpose-sorted*:
**fixes** *xs* :: *'a list list*
**assumes** *sorted*: *sorted* (*rev* (*map length xs*))
**shows** *length* (*transpose xs*) = (*if xs* = [] *then 0 else length* (*xs* ! *0*))
⟨*proof*⟩

**lemma** *nth-nth-transpose-sorted*[*simp*]:
**fixes** *xs* :: *'a list list*
**assumes** *sorted*: *sorted* (*rev* (*map length xs*))
**and** *i*: $i <$ *length* (*transpose xs*)
**and** *j*: $j <$ *length* (*filter* ($\lambda ys.\ i <$ *length ys*) *xs*)
**shows** *transpose xs* ! *i* ! *j* = *xs* ! *j* ! *i*
⟨*proof*⟩

**lemma** *transpose-column-length*:
**fixes** *xs* :: *'a list list*
**assumes** *sorted*: *sorted* (*rev* (*map length xs*)) **and** $i <$ *length xs*
**shows** *length* (*filter* ($\lambda ys.\ i <$ *length ys*) (*transpose xs*)) = *length* (*xs* ! *i*)
⟨*proof*⟩

**lemma** *transpose-column*:

**fixes** *xs* :: *'a list list*
**assumes** *sorted*: *sorted* (*rev* (*map length xs*)) **and** $i <$ *length xs*
**shows** *map* ($\lambda$*ys*. *ys* ! *i*) (*filter* ($\lambda$*ys*. $i <$ *length ys*) (*transpose xs*))
= *xs* ! *i* (**is** *?R* = -)
⟨*proof*⟩

**lemma** *transpose-transpose*:
**fixes** *xs* :: *'a list list*
**assumes** *sorted*: *sorted* (*rev* (*map length xs*))
**shows** *transpose* (*transpose xs*) = *takeWhile* ($\lambda x$. $x \neq$ []) *xs* (**is** *?L* = *?R*)
⟨*proof*⟩

**theorem** *transpose-rectangle*:
**assumes** *xs* = [] $\Longrightarrow$ *n* = *0*
**assumes** *rect*: $\bigwedge$ *i*. $i <$ *length xs* $\Longrightarrow$ *length* (*xs* ! *i*) = *n*
**shows** *transpose xs* = *map* ($\lambda$ *i*. *map* ($\lambda$ *j*. *xs* ! *j* ! *i*) [*0*..<*length xs*]) [*0*..<*n*]
(**is** *?trans* = *?map*)
⟨*proof*⟩

### 66.3.5 *sorted-key-list-of-set*

This function maps (finite) linearly ordered sets to sorted lists. The linear order is obtained by a key function that maps the elements of the set to a type that is linearly ordered. Warning: in most cases it is not a good idea to convert from sets to lists but one should convert in the other direction (via *set*).

Note: this is a generalisation of the older *sorted-list-of-set* that is obtained by setting the key function to the identity. Consequently, new theorems should be added to the locale below. They should also be aliased to more convenient names for use with *sorted-list-of-set* as seen further below.

**definition** (**in** *linorder*) *sorted-key-list-of-set* :: (*'b* ⇒ *'a*) ⇒ *'b set* ⇒ *'b list*
**where** *sorted-key-list-of-set f* ≡ *folding-on.F* (*insort-key f*) []

**locale** *folding-insort-key* = *lo?*: *linorder less-eq* :: *'a* ⇒ *'a* ⇒ *bool less*
**for** *less-eq* (**infix** ‹⪯› *50*) **and** *less* (**infix** ‹≺› *50*) +
**fixes** *S*
**fixes** *f* :: *'b* ⇒ *'a*
**assumes** *inj-on*: *inj-on f S*
**begin**

**lemma** *insort-key-commute*:
$x \in S \Longrightarrow y \in S \Longrightarrow$ *insort-key f y o insort-key f x* = *insort-key f x o insort-key f y*
⟨*proof*⟩

**sublocale** *fold-insort-key*: *folding-on S insort-key f* []
**rewrites** *folding-on.F* (*insort-key f*) [] = *sorted-key-list-of-set f*
⟨*proof*⟩

**lemma** *idem-if-sorted-distinct*:
**assumes** *set xs* $\subseteq$ *S* **and** *sorted* (*map f xs*) *distinct xs*
**shows** *sorted-key-list-of-set f* (*set xs*) = *xs*
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-empty*:
*sorted-key-list-of-set f* {} = []
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-insert*:
**assumes** *insert x A* $\subseteq$ *S* **and** *finite A x* $\notin$ *A*
**shows** *sorted-key-list-of-set f* (*insert x A*)
= *insort-key f x* (*sorted-key-list-of-set f A*)
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-insert-remove* [*simp*]:
**assumes** *insert x A* $\subseteq$ *S* **and** *finite A*
**shows** *sorted-key-list-of-set f* (*insert x A*)
= *insort-key f x* (*sorted-key-list-of-set f* (*A* − {*x*}))
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-eq-Nil-iff* [*simp*]:
**assumes** *A* $\subseteq$ *S* **and** *finite A*
**shows** *sorted-key-list-of-set f A* = [] $\longleftrightarrow$ *A* = {}
⟨*proof*⟩

**lemma** *set-sorted-key-list-of-set* [*simp*]:
**assumes** *A* $\subseteq$ *S* **and** *finite A*
**shows** *set* (*sorted-key-list-of-set f A*) = *A*
⟨*proof*⟩

**lemma** *sorted-sorted-key-list-of-set* [*simp*]:
**assumes** *A* $\subseteq$ *S*
**shows** *sorted* (*map f* (*sorted-key-list-of-set f A*))
⟨*proof*⟩

**lemma** *distinct-if-distinct-map*: *distinct* (*map f xs*) $\Longrightarrow$ *distinct xs*
⟨*proof*⟩

**lemma** *distinct-sorted-key-list-of-set* [*simp*]:
**assumes** *A* $\subseteq$ *S*
**shows** *distinct* (*map f* (*sorted-key-list-of-set f A*))
⟨*proof*⟩

**lemma** *length-sorted-key-list-of-set* [*simp*]:
**assumes** *A* $\subseteq$ *S*
**shows** *length* (*sorted-key-list-of-set f A*) = *card A*
⟨*proof*⟩

**lemmas** *sorted-key-list-of-set* =
*set-sorted-key-list-of-set sorted-sorted-key-list-of-set distinct-sorted-key-list-of-set*

**lemma** *sorted-key-list-of-set-remove*:
**assumes** *insert* $x$ $A \subseteq S$ **and** *finite* $A$
**shows** *sorted-key-list-of-set* $f$ $(A - \{x\})$ = *remove1* $x$ (*sorted-key-list-of-set* $f$ $A$)
⟨*proof*⟩

**lemma** *strict-sorted-key-list-of-set* [*simp*]:
$A \subseteq S \Longrightarrow$ *sorted-wrt* $(\prec)$ (*map* $f$ (*sorted-key-list-of-set* $f$ $A$))
⟨*proof*⟩

**lemma** *finite-set-strict-sorted*:
**assumes** $A \subseteq S$ **and** *finite* $A$
**obtains** $l$ **where** *sorted-wrt* $(\prec)$ (*map* $f$ $l$) *set* $l = A$ *length* $l$ = *card* $A$
⟨*proof*⟩

**lemma** (**in** *linorder*) *strict-sorted-equal*:
**assumes** *sorted-wrt* $(<)$ *xs*
**and** *sorted-wrt* $(<)$ *ys*
**and** *set ys* = *set xs*
**shows** *ys* = *xs*
⟨*proof*⟩

**lemma** (**in** *linorder*) *strict-sorted-equal-Uniq*: $\exists_{\leq 1}$*xs*. *sorted-wrt* $(<)$ *xs* $\wedge$ *set xs* = $A$
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-inject*:
**assumes** $A \subseteq S$ $B \subseteq S$
**assumes** *sorted-key-list-of-set* $f$ $A$ = *sorted-key-list-of-set* $f$ $B$ *finite* $A$ *finite* $B$
**shows** $A = B$
⟨*proof*⟩

**lemma** *sorted-key-list-of-set-unique*:
**assumes** $A \subseteq S$ **and** *finite* $A$
**shows** *sorted-wrt* $(\prec)$ (*map* $f$ $l$) $\wedge$ *set* $l = A$ $\wedge$ *length* $l$ = *card* $A$
$\longleftrightarrow$ *sorted-key-list-of-set* $f$ $A = l$
⟨*proof*⟩

**end**

**context** *linorder*
**begin**

**definition** *sorted-list-of-set* $\equiv$ *sorted-key-list-of-set* $(\lambda x::'a.\ x)$

We abuse the *rewrites* functionality of locales to remove trivial assumptions that result from instantiating the key function to the identity.

**sublocale** *sorted-list-of-set*: *folding-insort-key* $(\leq)$ $(<)$ *UNIV* $(\lambda x.\ x)$
**rewrites** *sorted-key-list-of-set* $(\lambda x.\ x)$ = *sorted-list-of-set*
**and** $\bigwedge xs.\ map\ (\lambda x.\ x)\ xs \equiv xs$
**and** $\bigwedge X.\ (X \subseteq UNIV) \equiv True$
**and** $\bigwedge x.\ x \in UNIV \equiv True$
**and** $\bigwedge P.\ (True \Longrightarrow P) \equiv Trueprop\ P$
**and** $\bigwedge P\ Q.\ (True \Longrightarrow PROP\ P \Longrightarrow PROP\ Q) \equiv (PROP\ P \Longrightarrow True \Longrightarrow PROP\ Q)$
⟨*proof*⟩

**lemma** *ex1-sorted-list-for-set-if-finite*:
*finite* $X \Longrightarrow \exists!xs.\ sorted\text{-}wrt\ (<)\ xs \wedge set\ xs = X$
⟨*proof*⟩

Alias theorems for backwards compatibility and ease of use.

**lemmas** *sorted-list-of-set* = *sorted-list-of-set.sorted-key-list-of-set* **and**
*sorted-list-of-set-empty* = *sorted-list-of-set.sorted-key-list-of-set-empty* **and**
*sorted-list-of-set-insert* = *sorted-list-of-set.sorted-key-list-of-set-insert* **and**
*sorted-list-of-set-insert-remove* = *sorted-list-of-set.sorted-key-list-of-set-insert-remove* **and**
*sorted-list-of-set-eq-Nil-iff* = *sorted-list-of-set.sorted-key-list-of-set-eq-Nil-iff* **and**
*set-sorted-list-of-set* = *sorted-list-of-set.set-sorted-key-list-of-set* **and**
*sorted-sorted-list-of-set* = *sorted-list-of-set.sorted-sorted-key-list-of-set* **and**
*distinct-sorted-list-of-set* = *sorted-list-of-set.distinct-sorted-key-list-of-set* **and**
*length-sorted-list-of-set* = *sorted-list-of-set.length-sorted-key-list-of-set* **and**
*sorted-list-of-set-remove* = *sorted-list-of-set.sorted-key-list-of-set-remove* **and**
*strict-sorted-list-of-set* = *sorted-list-of-set.strict-sorted-key-list-of-set* **and**
*sorted-list-of-set-inject* = *sorted-list-of-set.sorted-key-list-of-set-inject* **and**
*sorted-list-of-set-unique* = *sorted-list-of-set.sorted-key-list-of-set-unique* **and**
*finite-set-strict-sorted* = *sorted-list-of-set.finite-set-strict-sorted*

**lemma** *sorted-list-of-set-sort-remdups* [*code*]:
*sorted-list-of-set* (*set xs*) = *sort* (*remdups xs*)
⟨*proof*⟩

**end**

**lemma** *sorted-list-of-set-range* [*simp*]:
*sorted-list-of-set* $\{m..<n\} = [m..<n]$
⟨*proof*⟩

**lemma** *sorted-list-of-set-lessThan-Suc* [*simp*]:
*sorted-list-of-set* $\{..<Suc\ k\}$ = *sorted-list-of-set* $\{..<k\}$ @ $[k]$
⟨*proof*⟩

**lemma** *sorted-list-of-set-atMost-Suc* [*simp*]:

$sorted\text{-}list\text{-}of\text{-}set$ $\{..Suc\ k\}$ = $sorted\text{-}list\text{-}of\text{-}set$ $\{..k\}$ @ $[Suc\ k]$
⟨*proof*⟩

**lemma** *sorted-lift-of-set-eq-upto* [*simp*]:
‹*sorted-list-of-set* $\{k..l\} = [k..l]$›
⟨*proof*⟩

**lemma** *sorted-list-of-set-nonempty*:
**assumes** *finite* $A$ $A \neq \{\}$
**shows** *sorted-list-of-set* $A$ = *Min* $A$ # *sorted-list-of-set* $(A - \{Min\ A\})$
⟨*proof*⟩

**lemma** *sorted-list-of-set-greaterThanLessThan*:
**assumes** $Suc\ i < j$
**shows** *sorted-list-of-set* $\{i<..<j\}$ = $Suc\ i$ # *sorted-list-of-set* $\{Suc\ i<..<j\}$
⟨*proof*⟩

**lemma** *sorted-list-of-set-greaterThanAtMost*:
**assumes** $Suc\ i \leq j$
**shows** *sorted-list-of-set* $\{i<..j\}$ = $Suc\ i$ # *sorted-list-of-set* $\{Suc\ i<..j\}$
⟨*proof*⟩

**lemma** *nth-sorted-list-of-set-greaterThanLessThan*:
$n < j - Suc\ i \Longrightarrow$ *sorted-list-of-set* $\{i<..<j\}$ ! $n = Suc\ (i+n)$
⟨*proof*⟩

**lemma** *nth-sorted-list-of-set-greaterThanAtMost*:
$n < j - i \Longrightarrow$ *sorted-list-of-set* $\{i<..j\}$ ! $n = Suc\ (i+n)$
⟨*proof*⟩

**lemma** *sorted-wrt-induct* [*consumes 1*, *case-names Nil Cons*]:
**assumes** *sorted-wrt R xs*
**assumes** $P$ []
$\bigwedge x\ xs.\ (\bigwedge y.\ y \in set\ xs \Longrightarrow R\ x\ y) \Longrightarrow P\ xs \Longrightarrow P\ (x\ \#\ xs)$
**shows** $P\ xs$
⟨*proof*⟩

**lemma** *sorted-wrt-trans-induct* [*consumes 2*, *case-names Nil single Cons*]:
**assumes** *sorted-wrt R xs transp R*
**assumes** $P$ [] $\bigwedge x.\ P\ [x]$
$\bigwedge x\ y\ xs.\ R\ x\ y \Longrightarrow P\ (y\ \#\ xs) \Longrightarrow P\ (x\ \#\ y\ \#\ xs)$
**shows** $P\ xs$
⟨*proof*⟩

**lemmas** *sorted-induct* [*consumes 1*, *case-names Nil single Cons*] =
*sorted-wrt-trans-induct*[*OF - preorder-class.transp-on-le*]

**lemma** *sorted-wrt-map-mono*:

**assumes** *sorted-wrt R xs*
**assumes** $\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow R\ x\ y \Longrightarrow R'\ (f\ x)\ (f\ y)$
**shows** *sorted-wrt R′ (map f xs)*
⟨*proof*⟩

**lemma** *sorted-map-mono*:
**assumes** *sorted xs* **and** *mono-on (set xs) f*
**shows** *sorted (map f xs)*
⟨*proof*⟩

### 66.3.6 *lists*: the list-forming operator over sets

**inductive-set**
*lists* :: *′a set => ′a list set*
**for** *A* :: *′a set*
**where**
*Nil* [*intro*!, *simp*]: $[] \in lists\ A$
| *Cons* [*intro*!, *simp*]: $[\![a \in A;\ l \in lists\ A]\!] \Longrightarrow a\#l \in lists\ A$

**inductive-cases** *listsE* [*elim*!]: $x\#l \in lists\ A$
**inductive-cases** *listspE* [*elim*!]: *listsp A (x # l)*

**inductive-simps** *listsp-simps*[*code*]:
*listsp A* []
*listsp A (x # xs)*

**lemma** *listsp-mono* [*mono*]: $A \leq B \Longrightarrow listsp\ A \leq listsp\ B$
⟨*proof*⟩

**lemmas** *lists-mono* = *listsp-mono* [*to-set*]

**lemma** *listsp-infI*:
**assumes** *l*: *listsp A l* **shows** $listsp\ B\ l \Longrightarrow listsp\ (inf\ A\ B)\ l$ ⟨*proof*⟩

**lemmas** *lists-IntI* = *listsp-infI* [*to-set*]

**lemma** *listsp-inf-eq* [*simp*]: *listsp (inf A B) = inf (listsp A) (listsp B)*
⟨*proof*⟩

**lemmas** *listsp-conj-eq* [*simp*] = *listsp-inf-eq* [*simplified inf-fun-def inf-bool-def*]

**lemmas** *lists-Int-eq* [*simp*] = *listsp-inf-eq* [*to-set*]

**lemma** *Cons-in-lists-iff*[*simp*]: $x\#xs \in lists\ A \longleftrightarrow x \in A \wedge xs \in lists\ A$
⟨*proof*⟩

**lemma** *append-in-listsp-conv* [*iff*]: $(listsp\ A\ (xs\ @\ ys)) = (listsp\ A\ xs \wedge listsp\ A\ ys)$
⟨*proof*⟩

**lemmas** *append-in-lists-conv* [*iff*] = *append-in-listsp-conv* [*to-set*]

**lemma** *in-listsp-conv-set*: (*listsp A xs*) = ($\forall x \in$ *set xs*. *A x*)
— eliminate *listsp* in favour of *set*
⟨*proof*⟩

**lemmas** *in-lists-conv-set* [*code-unfold*] = *in-listsp-conv-set* [*to-set*]

**lemma** *in-listspD* [*dest*!]: *listsp A xs* $\Longrightarrow$ $\forall x\in$*set xs*. *A x*
⟨*proof*⟩

**lemmas** *in-listsD* [*dest*!] = *in-listspD* [*to-set*]

**lemma** *in-listspI* [*intro*!]: $\forall x\in$*set xs*. *A x* $\Longrightarrow$ *listsp A xs*
⟨*proof*⟩

**lemmas** *in-listsI* [*intro*!] = *in-listspI* [*to-set*]

**lemma** *mono-lists*: *mono lists*
⟨*proof*⟩

**lemma** *lists-eq-set*: *lists A* = {*xs*. *set xs* $\leq$ *A*}
⟨*proof*⟩

**lemma** *lists-empty* [*simp*]: *lists* {} = {[]}
⟨*proof*⟩

**lemma** *lists-UNIV* [*simp*]: *lists UNIV* = *UNIV*
⟨*proof*⟩

**lemma** *lists-image*: *lists* (*f*'*A*) = *map f* ' *lists A*
⟨*proof*⟩

**lemma** *inj-on-map-lists*: **assumes** *inj-on f A*
**shows** *inj-on* (*map f*) (*lists A*)
⟨*proof*⟩

**lemma** *bij-lists*: *bij-betw f X Y* $\Longrightarrow$ *bij-betw* (*map f*) (*lists X*) (*lists Y*)
⟨*proof*⟩

**lemma** *replicate-in-lists*: $a \in A$ $\Longrightarrow$ *replicate k a* $\in$ *lists A*
⟨*proof*⟩

### 66.3.7 Inductive definition for membership

**inductive** *ListMem* :: '*a* $\Rightarrow$ '*a list* $\Rightarrow$ *bool*
**where**
*elem*: *ListMem x* (*x* # *xs*)

| *insert*: $ListMem\ x\ xs \Longrightarrow ListMem\ x\ (y\ \#\ xs)$

**lemma** *ListMem-iff*: $(ListMem\ x\ xs) = (x \in set\ xs)$
⟨*proof*⟩

### 66.3.8 Lists as Cartesian products

*set-Cons A Xs*: the set of lists with head drawn from $A$ and tail drawn from $Xs$.

**definition** *set-Cons* :: $'a\ set \Rightarrow 'a\ list\ set \Rightarrow 'a\ list\ set$ **where**
$set\text{-}Cons\ A\ XS = \{z.\ \exists\, x\ xs.\ z = x\ \#\ xs \wedge x \in A \wedge xs \in XS\}$

**lemma** *set-Cons-sing-Nil* [*simp*]: $set\text{-}Cons\ A\ \{[]\} = (\%x.\ [x])\ `A$
⟨*proof*⟩

Yields the set of lists, all of the same length as the argument and with elements drawn from the corresponding element of the argument.

**primrec** *listset* :: $'a\ set\ list \Rightarrow 'a\ list\ set$ **where**
$listset\ [] = \{[]\}\ |$
$listset\ (A\ \#\ As) = set\text{-}Cons\ A\ (listset\ As)$

### 66.3.9 Transitive Closure on Lists

Use $^+$ on binary relations if possible. Transitive closure on lists is useful for executable definitions on the list level. Is not efficient, naive closure computation.

**definition** $trans\text{-}list\text{-}step\ ps = [(a,c).\ (a,b) \leftarrow ps,\ (b',c) \leftarrow ps,\ b{=}b']$

**lemma** *set-trans-list-step-subset-trancl*: $set\ (trans\text{-}list\text{-}step\ ps) \subseteq (set\ ps)\hat{}+$
⟨*proof*⟩

**function** *trancl-list* :: $('a * 'a)\ list \Rightarrow ('a * 'a)\ list$ **where**
$trancl\text{-}list\ ps =$
  $(let\ ps' = trans\text{-}list\text{-}step\ ps$
   $in\ if\ set\ ps' \subseteq\ set\ ps\ then\ ps\ else\ trancl\text{-}list\ (List.union\ ps'\ ps))$
⟨*proof*⟩

**termination**
⟨*proof*⟩

**declare** *trancl-list.simps*[*code*, *simp del*]

**lemma** *set-trancl-list*: $set(trancl\text{-}list\ ps) = (set\ ps)\hat{}+$
⟨*proof*⟩

## 66.4 Relations on Lists

### 66.4.1 Length Lexicographic Ordering

These orderings preserve well-foundedness: shorter lists precede longer lists. These ordering are not used in dictionaries.

**primrec** — The lexicographic ordering for lists of the specified length
$lexn :: ('a \times 'a)\ set \Rightarrow nat \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$lexn\ r\ 0 = \{\}\ |$
$lexn\ r\ (Suc\ n) =$
$(map\text{-}prod\ (\%(x, xs).\ x\#xs)\ (\%(x, xs).\ x\#xs)\ \text{`}\ (r <*lex*> lexn\ r\ n))\ Int$
$\{(xs, ys).\ length\ xs = Suc\ n \wedge length\ ys = Suc\ n\}$

**definition** $lex :: ('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$lex\ r = (\bigcup n.\ lexn\ r\ n)$ — Holds only between lists of the same length

**definition** $lenlex :: ('a \times 'a)\ set => ('a\ list \times 'a\ list)\ set$ **where**
$lenlex\ r = inv\text{-}image\ (less\text{-}than <*lex*> lex\ r)\ (\lambda xs.\ (length\ xs, xs))$
— Compares lists by their length and then lexicographically

**lemma** *wf-lexn*: **assumes** $wf\ r$ **shows** $wf\ (lexn\ r\ n)$
$\langle proof \rangle$

**lemma** *lexn-length*:
$(xs, ys) \in lexn\ r\ n \Longrightarrow length\ xs = n \wedge length\ ys = n$
$\langle proof \rangle$

**lemma** *wf-lex* [*intro*!]:
**assumes** $wf\ r$ **shows** $wf\ (lex\ r)$
$\langle proof \rangle$

**lemma** *lexn-conv*:
$lexn\ r\ n =$
$\{(xs,ys).\ length\ xs = n \wedge length\ ys = n \wedge$
$(\exists xys\ x\ y\ xs'\ ys'.\ xs = xys\ @\ x\#xs' \wedge ys = xys\ @\ y\ \#\ ys' \wedge (x, y) \in r)\}$
(**is** $?L\ n = ?R\ n$ **is** - $= \{(xs,ys).\ ?len\ n\ xs \wedge ?len\ n\ ys \wedge (\exists xys.\ ?P\ xs\ ys\ xys)\}$)
$\langle proof \rangle$

By Mathias Fleury:

**proposition** *lexn-transI*:
**assumes** $trans\ r$ **shows** $trans\ (lexn\ r\ n)$
$\langle proof \rangle$

**corollary** *lex-transI*:
**assumes** $trans\ r$ **shows** $trans\ (lex\ r)$
$\langle proof \rangle$

**lemma** *lex-conv*:
$lex\ r =$

$\{(xs,ys).\ length\ xs = length\ ys \wedge$
$(\exists\, xys\ x\ y\ xs'\ ys'.\ xs = xys\ @\ x\ \#\ xs' \wedge ys = xys\ @\ y\ \#\ ys' \wedge (x,\ y) \in r)\}$
$\langle proof \rangle$

**lemma** *wf-lenlex* [*intro*!]: $wf\ r \Longrightarrow wf\ (lenlex\ r)$
$\langle proof \rangle$

**lemma** *lenlex-conv*:
$lenlex\ r = \{(xs,ys).\ length\ xs < length\ ys \vee$
$length\ xs = length\ ys \wedge (xs,\ ys) \in lex\ r\}$
$\langle proof \rangle$

**lemma** *total-lenlex*:
**assumes** $total\ r$
**shows** $total\ (lenlex\ r)$
$\langle proof \rangle$

**lemma** *lenlex-transI* [*intro*]: $trans\ r \Longrightarrow trans\ (lenlex\ r)$
$\langle proof \rangle$

**lemma** *Nil-notin-lex* [*iff*]: $([],\ ys) \notin lex\ r$
$\langle proof \rangle$

**lemma** *Nil2-notin-lex* [*iff*]: $(xs,\ []) \notin lex\ r$
$\langle proof \rangle$

**lemma** *Cons-in-lex* [*simp*]:
$(x\ \#\ xs,\ y\ \#\ ys) \in lex\ r \longleftrightarrow (x,\ y) \in r \wedge length\ xs = length\ ys \vee x = y \wedge (xs,$
$ys) \in lex\ r$
(**is** $?lhs = ?rhs$)
$\langle proof \rangle$

**lemma** *Nil-lenlex-iff1* [*simp*]: $([],\ ns) \in lenlex\ r \longleftrightarrow ns \neq []$
**and** *Nil-lenlex-iff2* [*simp*]: $(ns,[]) \notin lenlex\ r$
$\langle proof \rangle$

**lemma** *Cons-lenlex-iff*:
$((m\ \#\ ms,\ n\ \#\ ns) \in lenlex\ r) \longleftrightarrow$
$length\ ms < length\ ns$
$\vee\ length\ ms = length\ ns \wedge (m,n) \in r$
$\vee\ (m = n \wedge (ms,ns) \in lenlex\ r)$
$\langle proof \rangle$

**lemma** *lenlex-irreflexive*: $(\bigwedge x.\ (x,x) \notin r) \Longrightarrow (xs,xs) \notin lenlex\ r$
$\langle proof \rangle$

**lemma** *lenlex-trans*:
$[\![(x,y) \in lenlex\ r;\ (y,z) \in lenlex\ r;\ trans\ r]\!] \Longrightarrow (x,z) \in lenlex\ r$
$\langle proof \rangle$

**lemma** *lenlex-length*: $(ms,\ ns) \in lenlex\ r \Longrightarrow length\ ms \leq length\ ns$
⟨*proof*⟩

**lemma** *lex-append-rightI*:
$(xs,\ ys) \in lex\ r \Longrightarrow length\ vs = length\ us \Longrightarrow (xs\ @\ us,\ ys\ @\ vs) \in lex\ r$
⟨*proof*⟩

**lemma** *lex-append-leftI*:
$(ys,\ zs) \in lex\ r \Longrightarrow (xs\ @\ ys,\ xs\ @\ zs) \in lex\ r$
⟨*proof*⟩

**lemma** *lex-append-leftD*:
$\forall x.\ (x,x) \notin r \Longrightarrow (xs\ @\ ys,\ xs\ @\ zs) \in lex\ r \Longrightarrow (ys,\ zs) \in lex\ r$
⟨*proof*⟩

**lemma** *lex-append-left-iff*:
$\forall x.\ (x,x) \notin r \Longrightarrow (xs\ @\ ys,\ xs\ @\ zs) \in lex\ r \longleftrightarrow (ys,\ zs) \in lex\ r$
⟨*proof*⟩

**lemma** *lex-take-index*:
**assumes** $(xs,\ ys) \in lex\ r$
**obtains** $i$ **where** $i < length\ xs$ **and** $i < length\ ys$ **and** $take\ i\ xs = take\ i\ ys$
**and** $(xs\ !\ i,\ ys\ !\ i) \in r$
⟨*proof*⟩

**lemma** *irrefl-lex*: $irrefl\ r \Longrightarrow irrefl\ (lex\ r)$
⟨*proof*⟩

**lemma** *lexl-not-refl* [*simp*]: $irrefl\ r \Longrightarrow (x,x) \notin lex\ r$
⟨*proof*⟩

### 66.4.2 Lexicographic Ordering

Classical lexicographic ordering on lists, ie. "a" < "ab" < "b". This ordering does *not* preserve well-foundedness. Author: N. Voelker, March 2005.

**definition** *lexord* :: $('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$lexord\ r = \{(x,y).\ \exists\ a\ v.\ y = x\ @\ a\ \#\ v\ \vee$
$(\exists\ u\ a\ b\ v\ w.\ (a,b) \in r \wedge x = u\ @\ (a\ \#\ v) \wedge y = u\ @\ (b\ \#\ w))\}$

**lemma** *lexord-Nil-left*[*simp*]: $([],y) \in lexord\ r = (\exists\ a\ x.\ y = a\ \#\ x)$
⟨*proof*⟩

**lemma** *lexord-Nil-right*[*simp*]: $(x,[]) \notin lexord\ r$
⟨*proof*⟩

**lemma** *lexord-cons-cons*[*simp*]:
$(a\ \#\ x,\ b\ \#\ y) \in lexord\ r \longleftrightarrow (a,b) \in r \vee (a = b \wedge (x,y) \in lexord\ r)$ (**is** *?lhs* = *?rhs*)

$\langle proof \rangle$

**lemmas** *lexord-simps* = *lexord-Nil-left lexord-Nil-right lexord-cons-cons*

**lemma** *lexord-same-pref-iff*:
$(xs\ @\ ys,\ xs\ @\ zs) \in lexord\ r \longleftrightarrow (\exists x \in set\ xs.\ (x,x) \in r) \lor (ys,\ zs) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-same-pref-if-irrefl*[*simp*]:
$irrefl\ r \Longrightarrow (xs\ @\ ys,\ xs\ @\ zs) \in lexord\ r \longleftrightarrow (ys,\ zs) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-append-rightI*: $\exists\ b\ z.\ y = b\ \#\ z \Longrightarrow (x,\ x\ @\ y) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-append-left-rightI*:
$(a,b) \in r \Longrightarrow (u\ @\ a\ \#\ x,\ u\ @\ b\ \#\ y) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-append-leftI*: $(u,v) \in lexord\ r \Longrightarrow (x\ @\ u,\ x\ @\ v) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-append-leftD*:
$[\![(x\ @\ u,\ x\ @\ v) \in lexord\ r;\ (\forall a.\ (a,a) \notin r)\ ]\!] \Longrightarrow (u,v) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-take-index-conv*:
$((x,y) \in lexord\ r) =$
$((length\ x < length\ y \land take\ (length\ x)\ y = x) \lor$
$(\exists i.\ i < min(length\ x)(length\ y) \land take\ i\ x = take\ i\ y \land (x!i,y!i) \in r))$
$\langle proof \rangle$
**lemma** *lexord-lex*: $(x,y) \in lex\ r = ((x,y) \in lexord\ r \land length\ x = length\ y)$
$\langle proof \rangle$

**lemma** *lexord-sufI*:
**assumes** $(u,w) \in lexord\ r$ $length\ w \leq length\ u$
**shows** $(u@v,w@z) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-sufE*:
**assumes** $(xs@zs,ys@qs) \in lexord\ r$ $xs \neq ys$ $length\ xs = length\ ys$ $length\ zs = length\ qs$
**shows** $(xs,ys) \in lexord\ r$
$\langle proof \rangle$

**lemma** *lexord-irreflexive*: $\forall x.\ (x,x) \notin r \Longrightarrow (xs,xs) \notin lexord\ r$
$\langle proof \rangle$

By René Thiemann:

**lemma** *lexord-partial-trans*:
$(\bigwedge x\ y\ z.\ x \in set\ xs \Longrightarrow (x,y) \in r \Longrightarrow (y,z) \in r \Longrightarrow (x,z) \in r)$
$\Longrightarrow\ (xs,ys) \in lexord\ r\ \Longrightarrow\ (ys,zs) \in lexord\ r \Longrightarrow\ (xs,zs) \in lexord\ r$
⟨*proof*⟩

**lemma** *lexord-trans*:
$[\![\ (x,\ y) \in lexord\ r;\ (y,\ z) \in lexord\ r;\ trans\ r\ ]\!] \Longrightarrow (x,\ z) \in lexord\ r$
⟨*proof*⟩

**lemma** *lexord-transI*: $trans\ r \Longrightarrow trans\ (lexord\ r)$
⟨*proof*⟩

**lemma** *total-lexord*: $total\ r \Longrightarrow total\ (lexord\ r)$
⟨*proof*⟩

**corollary** *lexord-linear*: $(\forall\ a\ b.\ (a,b) \in r \lor a = b \lor (b,a) \in r) \Longrightarrow (x,y) \in lexord\ r \lor x = y \lor (y,x) \in lexord\ r$
⟨*proof*⟩

**lemma** *lexord-irrefl*:
$irrefl\ R \Longrightarrow irrefl\ (lexord\ R)$
⟨*proof*⟩

**lemma** *lexord-asym*:
**assumes** $asym\ R$
**shows** $asym\ (lexord\ R)$
⟨*proof*⟩

**lemma** *lexord-asymmetric*:
**assumes** $asym\ R$
**assumes** *hyp*: $(a,\ b) \in lexord\ R$
**shows** $(b,\ a) \notin lexord\ R$
⟨*proof*⟩

**lemma** *asym-lex*: $asym\ R \Longrightarrow asym\ (lex\ R)$
⟨*proof*⟩

**lemma** *asym-lenlex*: $asym\ R \Longrightarrow asym\ (lenlex\ R)$
⟨*proof*⟩

**lemma** *lenlex-append1*:
**assumes** *len*: $(us,xs) \in lenlex\ R$ **and** *eq*: $length\ vs = length\ ys$
**shows** $(us\ @\ vs,\ xs\ @\ ys) \in lenlex\ R$
⟨*proof*⟩

**lemma** *lenlex-append2* [*simp*]:
**assumes** $irrefl\ R$
**shows** $(us\ @\ xs,\ us\ @\ ys) \in lenlex\ R \longleftrightarrow (xs,\ ys) \in lenlex\ R$
⟨*proof*⟩

Predicate version of lexicographic order integrated with Isabelle's order type classes. Author: Andreas Lochbihler

**context** *ord*
**begin**

**context**
  **notes** [[*inductive-internals*]]
**begin**

**inductive** *lexordp* :: $'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool*
**where**
  *Nil*: *lexordp* [] ($y$ # *ys*)
| *Cons*: $x < y \Longrightarrow$ *lexordp* ($x$ # *xs*) ($y$ # *ys*)
| *Cons-eq*:
  $[\![$ $\neg$ $x < y$; $\neg$ $y < x$; *lexordp xs ys* $]\!] \Longrightarrow$ *lexordp* ($x$ # *xs*) ($y$ # *ys*)

**end**

**lemma** *lexordp-simps* [*simp*, *code*]:
  *lexordp* [] *ys* $\longleftrightarrow$ *ys* $\neq$ []
  *lexordp xs* [] $\longleftrightarrow$ *False*
  *lexordp* ($x$ # *xs*) ($y$ # *ys*) $\longleftrightarrow$ $x < y \vee \neg\ y < x \wedge$ *lexordp xs ys*
⟨*proof*⟩

**inductive** *lexordp-eq* :: $'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool* **where**
  *Nil*: *lexordp-eq* [] *ys*
| *Cons*: $x < y \Longrightarrow$ *lexordp-eq* ($x$ # *xs*) ($y$ # *ys*)
| *Cons-eq*: $[\![$ $\neg$ $x < y$; $\neg$ $y < x$; *lexordp-eq xs ys* $]\!] \Longrightarrow$ *lexordp-eq* ($x$ # *xs*) ($y$ # *ys*)

**lemma** *lexordp-eq-simps* [*simp*, *code*]:
  *lexordp-eq* [] *ys* $\longleftrightarrow$ *True*
  *lexordp-eq xs* [] $\longleftrightarrow$ *xs* = []
  *lexordp-eq* ($x$ # *xs*) ($y$ # *ys*) $\longleftrightarrow$ $x < y \vee \neg\ y < x \wedge$ *lexordp-eq xs ys*
⟨*proof*⟩

**lemma** *lexordp-append-rightI*: *ys* $\neq$ *Nil* $\Longrightarrow$ *lexordp xs* (*xs* @ *ys*)
  ⟨*proof*⟩

**lemma** *lexordp-append-left-rightI*: $x < y \Longrightarrow$ *lexordp* (*us* @ $x$ # *xs*) (*us* @ $y$ # *ys*)
  ⟨*proof*⟩

**lemma** *lexordp-eq-refl*: *lexordp-eq xs xs*
  ⟨*proof*⟩

**lemma** *lexordp-append-leftI*: *lexordp us vs* $\Longrightarrow$ *lexordp* (*xs* @ *us*) (*xs* @ *vs*)
  ⟨*proof*⟩

**lemma** *lexordp-append-leftD*: $[\![$ *lexordp* (*xs* @ *us*) (*xs* @ *vs*); $\forall\, a.\ \neg\ a < a$ $]\!] \Longrightarrow$
*lexordp us vs*

⟨*proof*⟩

**lemma** *lexordp-irreflexive*:
**assumes** *irrefl*: $\forall x. \neg x < x$
**shows** $\neg$ *lexordp xs xs*
⟨*proof*⟩

**lemma** *lexordp-into-lexordp-eq*:
*lexordp xs ys* $\Longrightarrow$ *lexordp-eq xs ys*
⟨*proof*⟩

**lemma** *lexordp-eq-pref*: *lexordp-eq u* (*u* @ *v*)
⟨*proof*⟩

**end**

**declare** *ord.lexordp-simps* [*simp*, *code*]
**declare** *ord.lexordp-eq-simps* [*simp*, *code*]

**context** *order*
**begin**

**lemma** *lexordp-antisym*:
**assumes** *lexordp xs ys lexordp ys xs*
**shows** *False*
⟨*proof*⟩

**lemma** *lexordp-irreflexive'*: $\neg$ *lexordp xs xs*
⟨*proof*⟩

**end**

**context** *linorder* **begin**

**lemma** *lexordp-cases* [*consumes 1*, *case-names Nil Cons Cons-eq*, *cases pred*: *lexordp*]:
**assumes** *lexordp xs ys*
**obtains** (*Nil*) *y ys'* **where** $xs = []$ $ys = y \# ys'$
| (*Cons*) *x xs' y ys'* **where** $xs = x \# xs'$ $ys = y \# ys'$ $x < y$
| (*Cons-eq*) *x xs' ys'* **where** $xs = x \# xs'$ $ys = x \# ys'$ *lexordp xs' ys'*
⟨*proof*⟩

**lemma** *lexordp-induct* [*consumes 1*, *case-names Nil Cons Cons-eq*, *induct pred*: *lexordp*]:
**assumes** *major*: *lexordp xs ys*
**and** *Nil*: $\bigwedge y\ ys.\ P\ []\ (y \# ys)$
**and** *Cons*: $\bigwedge x\ xs\ y\ ys.\ x < y \Longrightarrow P\ (x \# xs)\ (y \# ys)$
**and** *Cons-eq*: $\bigwedge x\ xs\ ys.\ [\![\ lexordp\ xs\ ys;\ P\ xs\ ys\ ]\!] \Longrightarrow P\ (x \# xs)\ (x \# ys)$
**shows** *P xs ys*

⟨*proof*⟩

**lemma** *lexordp-iff*:
*lexordp xs ys* $\longleftrightarrow$ ($\exists$ *x vs. ys* = *xs* @ *x* # *vs*) $\vee$ ($\exists$ *us a b vs ws.* $a < b \wedge$ *xs* = *us* @ *a* # *vs* $\wedge$ *ys* = *us* @ *b* # *ws*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *lexordp-conv-lexord*:
*lexordp xs ys* $\longleftrightarrow$ (*xs*, *ys*) $\in$ *lexord* $\{(x, y).\ x < y\}$
⟨*proof*⟩

**lemma** *lexordp-eq-antisym*:
**assumes** *lexordp-eq xs ys lexordp-eq ys xs*
**shows** *xs* = *ys*
⟨*proof*⟩

**lemma** *lexordp-eq-trans*:
**assumes** *lexordp-eq xs ys* **and** *lexordp-eq ys zs*
**shows** *lexordp-eq xs zs*
⟨*proof*⟩

**lemma** *lexordp-trans*:
**assumes** *lexordp xs ys lexordp ys zs*
**shows** *lexordp xs zs*
⟨*proof*⟩

**lemma** *lexordp-linear*: *lexordp xs ys* $\vee$ *xs* = *ys* $\vee$ *lexordp ys xs*
⟨*proof*⟩

**lemma** *lexordp-conv-lexordp-eq*: *lexordp xs ys* $\longleftrightarrow$ *lexordp-eq xs ys* $\wedge$ $\neg$ *lexordp-eq ys xs*
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *lexordp-eq-conv-lexord*: *lexordp-eq xs ys* $\longleftrightarrow$ *xs* = *ys* $\vee$ *lexordp xs ys*
⟨*proof*⟩

**lemma** *lexordp-eq-linear*: *lexordp-eq xs ys* $\vee$ *lexordp-eq ys xs*
⟨*proof*⟩

**lemma** *lexordp-linorder*: *class.linorder lexordp-eq lexordp*
⟨*proof*⟩

**end**

### 66.4.3 Lexicographic combination of measure functions

These are useful for termination proofs

**definition** *measures fs* $=$ *inv-image* (*lex less-than*) ($\%a.$ *map* ($\%f.$ $f\ a$) *fs*)

**lemma** *wf-measures*[*simp*]: *wf* (*measures fs*)
⟨*proof*⟩

**lemma** *in-measures*[*simp*]:
$(x, y) \in$ *measures* $[] =$ *False*
$(x, y) \in$ *measures* $(f\ \#\ fs)$
$= (f\ x < f\ y \vee (f\ x = f\ y \wedge (x, y) \in$ *measures fs*$))$
⟨*proof*⟩

**lemma** *measures-less*: $f\ x < f\ y \Longrightarrow (x, y) \in$ *measures* $(f\#fs)$
⟨*proof*⟩

**lemma** *measures-lesseq*: $f\ x \leq f\ y \Longrightarrow (x, y) \in$ *measures fs* $\Longrightarrow (x, y) \in$ *measures* $(f\#fs)$
⟨*proof*⟩

### 66.4.4 Lifting Relations to Lists: one element

**definition** *listrel1* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a$ *list* $\times$ $'a$ *list*$)$ *set* **where**
*listrel1* $r = \{(xs,ys).$
$\exists us\ z\ z'\ vs.\ xs = us\ @\ z\ \#\ vs \wedge (z,z') \in r \wedge ys = us\ @\ z'\ \#\ vs\}$

**lemma** *listrel1I*:
$[\![\ (x, y) \in r;\ \ xs = us\ @\ x\ \#\ vs;\ \ ys = us\ @\ y\ \#\ vs\ ]\!] \Longrightarrow$
$(xs, ys) \in$ *listrel1* $r$
⟨*proof*⟩

**lemma** *listrel1E*:
$[\![\ (xs, ys) \in$ *listrel1* $r;$
$!!x\ y\ us\ vs.\ [\![\ (x, y) \in r;\ \ xs = us\ @\ x\ \#\ vs;\ \ ys = us\ @\ y\ \#\ vs\ ]\!] \Longrightarrow P$
$]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *not-Nil-listrel1* [*iff*]: $([], xs) \notin$ *listrel1* $r$
⟨*proof*⟩

**lemma** *not-listrel1-Nil* [*iff*]: $(xs, []) \notin$ *listrel1* $r$
⟨*proof*⟩

**lemma** *Cons-listrel1-Cons* [*iff*]:
$(x\ \#\ xs, y\ \#\ ys) \in$ *listrel1* $r \longleftrightarrow$
$(x,y) \in r \wedge xs = ys \vee x = y \wedge (xs, ys) \in$ *listrel1* $r$
⟨*proof*⟩

**lemma** *listrel1I1*: $(x,y) \in r \Longrightarrow (x\ \#\ xs, y\ \#\ xs) \in$ *listrel1* $r$
⟨*proof*⟩

**lemma** *listrel1I2*: $(xs,\ ys) \in listrel1\ r \Longrightarrow (x \# xs,\ x \# ys) \in listrel1\ r$
⟨*proof*⟩

**lemma** *append-listrel1I*:
$(xs,\ ys) \in listrel1\ r \wedge us = vs \vee xs = ys \wedge (us,\ vs) \in listrel1\ r$
$\Longrightarrow (xs\ @\ us,\ ys\ @\ vs) \in listrel1\ r$
⟨*proof*⟩

**lemma** *Cons-listrel1E1* [*elim*!]:
**assumes** $(x \# xs,\ ys) \in listrel1\ r$
**and** $\bigwedge y.\ ys = y \# xs \Longrightarrow (x,\ y) \in r \Longrightarrow R$
**and** $\bigwedge zs.\ ys = x \# zs \Longrightarrow (xs,\ zs) \in listrel1\ r \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

**lemma** *Cons-listrel1E2* [*elim*!]:
**assumes** $(xs,\ y \# ys) \in listrel1\ r$
**and** $\bigwedge x.\ xs = x \# ys \Longrightarrow (x,\ y) \in r \Longrightarrow R$
**and** $\bigwedge zs.\ xs = y \# zs \Longrightarrow (zs,\ ys) \in listrel1\ r \Longrightarrow R$
**shows** $R$
⟨*proof*⟩

**lemma** *snoc-listrel1-snoc-iff*:
$(xs\ @\ [x],\ ys\ @\ [y]) \in listrel1\ r$
$\longleftrightarrow (xs,\ ys) \in listrel1\ r \wedge x = y \vee xs = ys \wedge (x,y) \in r$ (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

**lemma** *listrel1-eq-len*: $(xs,ys) \in listrel1\ r \Longrightarrow length\ xs = length\ ys$
⟨*proof*⟩

**lemma** *listrel1-mono*:
$r \subseteq s \Longrightarrow listrel1\ r \subseteq listrel1\ s$
⟨*proof*⟩

**lemma** *listrel1-converse*: $listrel1\ (r^{-1}) = (listrel1\ r)^{-1}$
⟨*proof*⟩

**lemma** *in-listrel1-converse*:
$(x,y) \in listrel1\ (r^{-1}) \longleftrightarrow (x,y) \in (listrel1\ r)^{-1}$
⟨*proof*⟩

**lemma** *listrel1-iff-update*:
$(xs,ys) \in (listrel1\ r)$
$\longleftrightarrow (\exists\, y\ n.\ (xs\ !\ n,\ y) \in r \wedge n < length\ xs \wedge ys = xs[n{:=}y])$ (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

Accessible part and wellfoundedness:

**lemma** *Cons-acc-listrel1I* [*intro*!]:

$x \in$ *Wellfounded.acc* $r \Longrightarrow xs \in$ *Wellfounded.acc* (*listrel1* $r$) $\Longrightarrow$ ($x$ # $xs$) $\in$ *Wellfounded.acc* (*listrel1* $r$)
⟨*proof*⟩

**lemma** *lists-accD*: $xs \in$ *lists* (*Wellfounded.acc* $r$) $\Longrightarrow xs \in$ *Wellfounded.acc* (*listrel1* $r$)
⟨*proof*⟩

**lemma** *lists-accI*: $xs \in$ *Wellfounded.acc* (*listrel1* $r$) $\Longrightarrow xs \in$ *lists* (*Wellfounded.acc* $r$)
⟨*proof*⟩

**lemma** *wf-listrel1-iff* [*simp*]: *wf*(*listrel1* $r$) = *wf* $r$
⟨*proof*⟩

### 66.4.5 Lifting Relations to Lists: all elements

**inductive-set**
*listrel* :: ($'a \times 'b$) *set* $\Rightarrow$ ($'a$ *list* $\times$ $'b$ *list*) *set*
**for** $r$ :: ($'a \times 'b$) *set*
**where**
*Nil*: ([],[]) $\in$ *listrel* $r$
| *Cons*: $[\![(x,y) \in r; (xs,ys) \in$ *listrel* $r]\!] \Longrightarrow (x\#xs,\ y\#ys) \in$ *listrel* $r$

**inductive-cases** *listrel-Nil1* [*elim*!]: ([],$xs$) $\in$ *listrel* $r$
**inductive-cases** *listrel-Nil2* [*elim*!]: ($xs$,[]) $\in$ *listrel* $r$
**inductive-cases** *listrel-Cons1* [*elim*!]: ($y\#ys$,$xs$) $\in$ *listrel* $r$
**inductive-cases** *listrel-Cons2* [*elim*!]: ($xs$,$y\#ys$) $\in$ *listrel* $r$

**lemma** *listrel-eq-len*: ($xs$, $ys$) $\in$ *listrel* $r \Longrightarrow$ *length* $xs$ = *length* $ys$
⟨*proof*⟩

**lemma** *listrel-iff-zip* [*code-unfold*]: ($xs$,$ys$) $\in$ *listrel* $r \longleftrightarrow$
*length* $xs$ = *length* $ys$ $\wedge$ ($\forall$ ($x$,$y$) $\in$ *set*(*zip* $xs$ $ys$). ($x$,$y$) $\in r$) (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

**lemma** *listrel-iff-nth*: ($xs$,$ys$) $\in$ *listrel* $r \longleftrightarrow$
*length* $xs$ = *length* $ys$ $\wedge$ ($\forall n <$ *length* $xs$. ($xs!n$, $ys!n$) $\in r$) (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

**lemma** *listrel-mono*: $r \subseteq s \Longrightarrow$ *listrel* $r \subseteq$ *listrel* $s$
⟨*proof*⟩

**lemma** *listrel-subset*:
**assumes** $r \subseteq A \times A$ **shows** *listrel* $r \subseteq$ *lists* $A \times$ *lists* $A$
⟨*proof*⟩

**lemma** *listrel-refl-on*:

**assumes** *refl-on* $A$ $r$ **shows** *refl-on* (*lists* $A$) (*listrel* $r$)
⟨*proof*⟩

**lemma** *listrel-sym*: *sym* $r \Longrightarrow$ *sym* (*listrel* $r$)
⟨*proof*⟩

**lemma** *listrel-trans*:
**assumes** *trans* $r$ **shows** *trans* (*listrel* $r$)
⟨*proof*⟩

**theorem** *equiv-listrel*: *equiv* $A$ $r \Longrightarrow$ *equiv* (*lists* $A$) (*listrel* $r$)
⟨*proof*⟩

**lemma** *listrel-rtrancl-refl*[*iff*]: $(xs,xs) \in listrel(r^*)$
⟨*proof*⟩

**lemma** *listrel-rtrancl-trans*:
$[\![(xs,ys) \in listrel(r^*);\ \ (ys,zs) \in listrel(r^*)]\!] \Longrightarrow (xs,zs) \in listrel(r^*)$
⟨*proof*⟩

**lemma** *listrel-Nil* [*simp*]: *listrel* $r$ `` $\{[]\} = \{[]\}$
⟨*proof*⟩

**lemma** *listrel-Cons*:
*listrel* $r$ `` $\{x\#xs\}$ = *set-Cons* ($r$``$\{x\}$) (*listrel* $r$ `` $\{xs\}$)
⟨*proof*⟩

Relating *listrel1*, *listrel* and closures:

**lemma** *listrel1-rtrancl-subset-rtrancl-listrel1*: *listrel1* $(r^*) \subseteq$ (*listrel1* $r)^*$
⟨*proof*⟩

**lemma** *rtrancl-listrel1-eq-len*: $(x,y) \in$ (*listrel1* $r)^* \Longrightarrow$ *length* $x$ = *length* $y$
⟨*proof*⟩

**lemma** *rtrancl-listrel1-ConsI1*:
$(xs,ys) \in$ (*listrel1* $r)^* \Longrightarrow (x\#xs,x\#ys) \in$ (*listrel1* $r)^*$
⟨*proof*⟩

**lemma** *rtrancl-listrel1-ConsI2*:
$(x,y) \in r^* \Longrightarrow (xs,\ ys) \in$ (*listrel1* $r)^* \Longrightarrow (x\ \#\ xs,\ y\ \#\ ys) \in$ (*listrel1* $r)^*$
⟨*proof*⟩

**lemma** *listrel1-subset-listrel*:
$r \subseteq r' \Longrightarrow$ *refl* $r' \Longrightarrow$ *listrel1* $r \subseteq listrel(r')$
⟨*proof*⟩

**lemma** *listrel-reflcl-if-listrel1*:
$(xs,ys) \in$ *listrel1* $r \Longrightarrow (xs,ys) \in listrel(r^*)$
⟨*proof*⟩

**lemma** *listrel-rtrancl-eq-rtrancl-listrel1*: $listrel\ (r^*) = (listrel1\ r)^*$
⟨*proof*⟩

**lemma** *rtrancl-listrel1-if-listrel*:
$(xs,ys) \in listrel\ r \Longrightarrow (xs,ys) \in (listrel1\ r)^*$
⟨*proof*⟩

**lemma** *listrel-subset-rtrancl-listrel1*: $listrel\ r \subseteq (listrel1\ r)^*$
⟨*proof*⟩

## 66.5 Size function

**lemma** [*measure-function*]: $is\text{-}measure\ f \Longrightarrow is\text{-}measure\ (size\text{-}list\ f)$
⟨*proof*⟩

**lemma** [*measure-function*]: $is\text{-}measure\ f \Longrightarrow is\text{-}measure\ (size\text{-}option\ f)$
⟨*proof*⟩

**lemma** *size-list-estimation*[*termination-simp*]:
$x \in set\ xs \Longrightarrow y < f\ x \Longrightarrow y < size\text{-}list\ f\ xs$
⟨*proof*⟩

**lemma** *size-list-estimation'*[*termination-simp*]:
$x \in set\ xs \Longrightarrow y \leq f\ x \Longrightarrow y \leq size\text{-}list\ f\ xs$
⟨*proof*⟩

**lemma** *size-list-map*[*simp*]: $size\text{-}list\ f\ (map\ g\ xs) = size\text{-}list\ (f \circ g)\ xs$
⟨*proof*⟩

**lemma** *size-list-append*[*simp*]: $size\text{-}list\ f\ (xs\ @\ ys) = size\text{-}list\ f\ xs + size\text{-}list\ f\ ys$
⟨*proof*⟩

**lemma** *size-list-pointwise*[*termination-simp*]:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x \leq g\ x) \Longrightarrow size\text{-}list\ f\ xs \leq size\text{-}list\ g\ xs$
⟨*proof*⟩

## 66.6 Monad operation

**definition** $bind :: 'a\ list \Rightarrow ('a \Rightarrow 'b\ list) \Rightarrow 'b\ list$ **where**
$bind\ xs\ f = concat\ (map\ f\ xs)$

**hide-const** (**open**) *bind*

**lemma** *bind-simps* [*simp*]:
$List.bind\ []\ f = []$
$List.bind\ (x\ \#\ xs)\ f = f\ x\ @\ List.bind\ xs\ f$
⟨*proof*⟩

**lemma** *list-bind-cong* [*fundef-cong*]:

**assumes** $xs = ys$ $(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x = g\ x)$
**shows** $List.bind\ xs\ f = List.bind\ ys\ g$
⟨*proof*⟩

**lemma** *set-list-bind*: $set\ (List.bind\ xs\ f) = (\bigcup x{\in}set\ xs.\ set\ (f\ x))$
⟨*proof*⟩

## 66.7 Code generation

### 66.7.1 Counterparts for set-related operations

**context**
**begin**

**qualified definition** *member* :: ‹*'a list* ⇒ *'a* ⇒ *bool*› — only for code generation
**where** *member-iff* [*code-abbrev*, *simp*]: ‹$member\ xs\ x \longleftrightarrow x \in set\ xs$›

Use *member* only for generating executable code. Otherwise use $x \in set\ xs$ instead — it is much easier to reason about.

**qualified lemma** *member-code* [*code*, *no-atp*]:
‹$member\ []\ y \longleftrightarrow False$›
‹$member\ (x\ \#\ xs)\ y \longleftrightarrow x = y \lor member\ xs\ y$›
⟨*proof*⟩ **lemma** *Collect-member* [*code-unfold*, *no-atp*]: — make preprocessor setup confluent
‹$\{x.\ List.member\ xs\ x \land P\ x\} = Set.filter\ P\ (set\ xs)$›
⟨*proof*⟩ **lemma** *Collect-pair-member* [*code-unfold*, *no-atp*]: — make preprocessor setup confluent
‹$\{(x, y).\ List.member\ xs\ (x, y) \land P\ x\ y\} = Set.filter\ (\lambda(x, y).\ P\ x\ y)\ (set\ xs)$›
⟨*proof*⟩ **lemma** *Collect-triple-member* [*code-unfold*, *no-atp*]: — make preprocessor setup confluent
‹$\{(x, y, z).\ List.member\ xs\ (x, y, z) \land P\ x\ y\ z\} = Set.filter\ (\lambda(x, y, z).\ P\ x\ y\ z)$ $(set\ xs)$›
⟨*proof*⟩

**end**

**lemma** *list-all-iff* [*code-abbrev*]:
‹$list\text{-}all\ P\ xs \longleftrightarrow Ball\ (set\ xs)\ P$›
⟨*proof*⟩

**definition** *list-ex* :: ‹(*'a* ⇒ *bool*) ⇒ *'a list* ⇒ *bool*›
**where** *list-ex-iff* [*code-abbrev*]: ‹$list\text{-}ex\ P\ xs \longleftrightarrow Bex\ (set\ xs)\ P$›

**definition** *list-ex1* :: ‹(*'a* ⇒ *bool*) ⇒ *'a list* ⇒ *bool*›
**where** *list-ex1-iff* [*code-abbrev*]: ‹$list\text{-}ex1\ P\ xs \longleftrightarrow Set.can\text{-}select\ P\ (set\ xs)$›

Usually you should prefer $\forall x{\in}set\ xs$, $\exists x{\in}set\ xs$ and $\exists!x.\ x{\in}set\ xs \land$ - over *list-all*, *list-ex* and *list-ex1* in specifications.

**lemma** *list-all-Nil-iff* [*code*, *no-atp*]:

‹*list-all P* [] ⟷ *True*›
⟨*proof*⟩

**lemma** *list-all-Cons-iff* [*code*, *no-atp*]:
‹*list-all P* (*x* # *xs*) ⟷ *P x* ∧ *list-all P xs*›
⟨*proof*⟩

**lemma** *list-ex-Nil-iff* [*simp*, *code*, *no-atp*]:
‹*list-ex P* [] ⟷ *False*›
⟨*proof*⟩

**lemma** *list-ex-Cons-iff* [*simp*, *code*, *no-atp*]:
‹*list-ex P* (*x* # *xs*) ⟷ *P x* ∨ *list-ex P xs*›
⟨*proof*⟩

**lemma** *list-ex1-Nil-iff* [*simp*, *code*, *no-atp*]:
‹*list-ex1 P* [] ⟷ *False*›
⟨*proof*⟩

**lemma** *list-ex1-Cons-iff* [*simp*, *code*, *no-atp*]:
‹*list-ex1 P* (*x* # *xs*) ⟷ (*if P x then list-all* (λ*y*. ¬ *P y* ∨ *x* = *y*) *xs else list-ex1 P xs*)›
⟨*proof*⟩

**lemma** *list-all-append* [*simp*]:
‹*list-all P* (*xs* @ *ys*) ⟷ *list-all P xs* ∧ *list-all P ys*›
⟨*proof*⟩

**lemma** *list-ex-append* [*simp*]:
‹*list-ex P* (*xs* @ *ys*) ⟷ *list-ex P xs* ∨ *list-ex P ys*›
⟨*proof*⟩

**lemma** *list-all-rev* [*simp*]:
‹*list-all P* (*rev xs*) ⟷ *list-all P xs*›
⟨*proof*⟩

**lemma** *list-ex-rev* [*simp*]:
‹*list-ex P* (*rev xs*) ⟷ *list-ex P xs*›
⟨*proof*⟩

**lemma** *list-all-length*:
‹*list-all P xs* ⟷ (∀ *n* < *length xs*. *P* (*xs* ! *n*))›
⟨*proof*⟩

**lemma** *list-ex-length*:
‹*list-ex P xs* ⟷ (∃ *n* < *length xs*. *P* (*xs* ! *n*))›
⟨*proof*⟩

**lemma** *list-all-cong* [*fundef-cong*]:

‹*list-all f xs* = *list-all g ys*›
**if** ‹*xs* = *ys*› ‹($\bigwedge x.\ x \in set\ ys \Longrightarrow f\ x = g\ x$)›
⟨*proof*⟩

**lemma** *list-ex-cong* [*fundef-cong*]:
‹*list-ex f xs* = *list-ex g ys*›
**if** ‹*xs* = *ys*› ‹($\bigwedge x.\ x \in set\ ys \Longrightarrow f\ x = g\ x$)›
⟨*proof*⟩

**context**
**begin**

**qualified definition** *superset* :: ‹$'a\ list \Rightarrow 'a\ list \Rightarrow bool$›
**where** *superset-iff* [*code-abbrev*, *simp*]: ‹$superset\ ys\ xs \longleftrightarrow set\ xs \subseteq set\ ys$›

**lemma** [*code*, *no-atp*]:
‹$superset\ xs = list\text{-}all\ (\lambda x.\ x \in set\ xs)$›
⟨*proof*⟩

**end**

Executable checks for relations on sets

**definition** *listrel1p* :: ‹$('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$› — only for code generation
**where** ‹$listrel1p\ r\ xs\ ys \longleftrightarrow (xs,\ ys) \in listrel1\ \{(x,\ y).\ r\ x\ y\}$›

**lemma** [*code-unfold*]:
‹$(xs,\ ys) \in listrel1\ r \longleftrightarrow listrel1p\ (\lambda x\ y.\ (x,\ y) \in r)\ xs\ ys$›
⟨*proof*⟩

**lemma** [*code*]:
‹$listrel1p\ r\ []\ xs \longleftrightarrow False$›
‹$listrel1p\ r\ xs\ []\ \longleftrightarrow\ False$›
‹$listrel1p\ r\ (x\ \#\ xs)\ (y\ \#\ ys) \longleftrightarrow$
$r\ x\ y \wedge xs = ys \vee x = y \wedge listrel1p\ r\ xs\ ys$›
⟨*proof*⟩

**definition** *lexordp* :: ‹$('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$› — only for code generation
**where** ‹$lexordp\ r\ xs\ ys \longleftrightarrow (xs,\ ys) \in lexord\ \{(x,\ y).\ r\ x\ y\}$›

**lemma** [*code-unfold*]:
‹$(xs,\ ys) \in lexord\ r = lexordp\ (\lambda x\ y.\ (x,\ y) \in r)\ xs\ ys$›
⟨*proof*⟩

**lemma** [*code*]:
‹$lexordp\ r\ xs\ [] \longleftrightarrow False$›
‹$lexordp\ r\ []\ (y\ \#\ ys) \longleftrightarrow True$›
‹$lexordp\ r\ (x\ \#\ xs)\ (y\ \#\ ys) \longleftrightarrow$

$r\ x\ y \vee (x = y \wedge lexordp\ r\ xs\ ys)$›
⟨proof⟩

Executable intervals

**context** *preorder*
**begin**

**lemma** *forall-less-eq-iff* [*code-unfold*]:
‹$(\forall\, n \le b.\ P\ n) \longleftrightarrow (\forall\, n \in \{..b\}.\ P\ n)$›
⟨proof⟩

**lemma** *exists-less-eq-iff* [*code-unfold*]:
‹$(\exists\, n \le b.\ P\ n) \longleftrightarrow (\exists\, n \in \{..b\}.\ P\ n)$›
⟨proof⟩

**lemma** *forall-less-iff* [*code-unfold*]:
‹$(\forall\, n < b.\ P\ n) \longleftrightarrow (\forall\, n \in \{..<b\}.\ P\ n)$›
⟨proof⟩

**lemma** *exists-less-iff* [*code-unfold*]:
‹$(\exists\, n < b.\ P\ n) \longleftrightarrow (\exists\, n \in \{..<b\}.\ P\ n)$›
⟨proof⟩

**lemma** *forall-greater-eq-iff* [*code-unfold*]:
‹$(\forall\, n \ge a.\ P\ n) \longleftrightarrow (\forall\, n \in \{a..\}.\ P\ n)$›
⟨proof⟩

**lemma** *exists-greater-eq-iff* [*code-unfold*]:
‹$(\exists\, n \ge a.\ P\ n) \longleftrightarrow (\exists\, n \in \{a..\}.\ P\ n)$›
⟨proof⟩

**lemma** *forall-greater-iff* [*code-unfold*]:
‹$(\forall\, n > a.\ P\ n) \longleftrightarrow (\forall\, n \in \{a<..\}.\ P\ n)$›
⟨proof⟩

**lemma** *exists-greater-iff* [*code-unfold*]:
‹$(\exists\, n > a.\ P\ n) \longleftrightarrow (\exists\, n \in \{a<..\}.\ P\ n)$›
⟨proof⟩

**end**

**class** *interval* = *linorder* + *comm-semiring-1-cancel* +
**assumes** *finite-atLeastAtMost*: ‹*finite* $\{a..b\}$›
**assumes** *dec-less-imp-less-eq*: ‹$a - 1 < b \Longrightarrow a \le b$›
**assumes** *less-inc-imp-less-eq*: ‹$a < b + 1 \Longrightarrow a \le b$›
**assumes** *dec-greater-eq-self-imp-bot*: ‹$a \le a - 1 \Longrightarrow a \le c$›
**assumes** *inc-less-eq-self-imp-top*: ‹$b + 1 \le b \Longrightarrow d \le b$›
**begin**

**context**
**begin**

**qualified lemma** *less-imp-less-eq-dec*:
‹$c < b \Longrightarrow a < b \Longrightarrow a \leq b - 1$›
⟨*proof*⟩ **lemma** *less-imp-in-less-eq*:
‹$a < c \Longrightarrow a < b \Longrightarrow a + 1 \leq b$›
⟨*proof*⟩ **lemma** *less-eq-dec-imp-less*:
‹$c < b \Longrightarrow a \leq b - 1 \Longrightarrow a < b$›
⟨*proof*⟩ **lemma** *inc-less-eq-imp-less*:
‹$a < c \Longrightarrow a + 1 \leq b \Longrightarrow a < b$›
⟨*proof*⟩ **definition** *interval* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$ *list*› — only for code generation
**where** *interval-eq*: ‹*interval a b* = *sorted-list-of-set* $\{a..b\}$›

**qualified lemma** *set-interval-eq* [*simp*]:
‹*set* (*interval a b*) = $\{a..b\}$›
⟨*proof*⟩ **lemma** *distinct-interval* [*simp*]:
‹*distinct* (*interval a b*)›
⟨*proof*⟩ **lemma** *interval-code* [*code*]:
‹*interval a b* = (*if* $a < b$ *then a* # *interval* $(a + 1)$ *b else if* $a = b$ *then* [*a*] *else* [])›
⟨*proof*⟩ **lemma** *atLeastAtMost-eq-interval* [*code*]:
‹$\{a..b\}$ = *set* (*interval a b*)›
⟨*proof*⟩ **lemma** *atLeastLessThan-eq-interval* [*code*]:
‹$\{a..<b\}$ = (*let* $d = b - 1$ *in if* $d < b$ *then set* (*interval a d*) *else* $\{\}$)›
⟨*proof*⟩ **lemma** *greaterThanAtMost-eq-interval* [*code*]:
‹$\{a<..b\}$ = (*let* $c = a + 1$ *in if* $a < c$ *then set* (*interval c b*) *else* $\{\}$)›
⟨*proof*⟩ **lemma** *greaterThanLessThan-eq-interval* [*code*]:
‹$\{a<..<b\}$ = (*let* $c = a + 1$; $d = b - 1$ *in if* $a < c \wedge d < b$ *then set* (*interval c d*) *else* $\{\}$)›
⟨*proof*⟩ **definition** *all-interval* :: ‹$('a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$› — only for code generation
**where** *all-interval-iff* [*code-post*, *simp*]: ‹*all-interval P a b* $\longleftrightarrow$ $(\forall n \in \{a..b\}.\ P\ n)$›

**qualified lemma** *all-interval-code* [*code*]:
‹*all-interval P a b* $\longleftrightarrow$ (($a < b \longrightarrow P\ a \wedge$ *all-interval P* $(a + 1)$ *b*) $\wedge$ ($a = b \longrightarrow P\ a$))›
⟨*proof*⟩ **lemma** *forall-atLeastAtMost-iff* [*code-unfold*]:
‹$(\forall n \in \{a..b\}.\ P\ n) \longleftrightarrow$ *all-interval P a b*›
⟨*proof*⟩ **lemma** *exists-atLeastAtMost-iff* [*code-unfold*]:
‹$(\exists n \in \{a..b\}.\ P\ n) \longleftrightarrow \neg$ *all-interval* (*Not* $\circ$ *P*) *a b*›
⟨*proof*⟩ **lemma** *forall-atLeastLessThan-iff* [*code-unfold*]:
‹$(\forall n \in \{a..<b\}.\ P\ n) \longleftrightarrow$ (*let* $d = b - 1$ *in* $d < b \longrightarrow$ *all-interval P a d*)›
⟨*proof*⟩ **lemma** *exists-atLeastLessThan-iff* [*code-unfold*]:
‹$(\exists n \in \{a..<b\}.\ P\ n) \longleftrightarrow$ (*let* $d = b - 1$ *in* $d < b \wedge \neg$ *all-interval* (*Not* $\circ$ *P*) *a d*)›
⟨*proof*⟩ **lemma** *forall-greaterThanAtMost-iff* [*code-unfold*]:
‹$(\forall n \in \{a<..b\}.\ P\ n) \longleftrightarrow$ (*let* $c = a + 1$ *in* $a < c \longrightarrow$ *all-interval P c b*)›

⟨*proof*⟩ **lemma** *exists-greaterThanAtMost-iff* [*code-unfold*]:
‹$(\exists\, n\in\{a<..b\}.\ P\ n) \longleftrightarrow (let\ c = a + 1\ in\ a < c \wedge \neg\ all\text{-}interval\ (Not \circ P)\ c\ b)$›
⟨*proof*⟩ **lemma** *forall-greaterThanLessThan-iff* [*code-unfold*]:
‹$(\forall\, n\in\{a<..<b\}.\ P\ n) \longleftrightarrow (let\ c = a + 1;\ d = b - 1\ in\ a < c \longrightarrow d < b \longrightarrow all\text{-}interval\ P\ c\ d)$›
⟨*proof*⟩ **lemma** *exists-greaterThanLessThan-iff* [*code-unfold*]:
‹$(\exists\, n\in\{a<..<b\}.\ P\ n) \longleftrightarrow (let\ c = a + 1;\ d = b - 1\ in\ a < c \wedge d < b \wedge \neg\ all\text{-}interval\ (Not \circ P)\ c\ d)$›
⟨*proof*⟩

**end**

**end**

**class** *interval-top* = *interval* + *order-top*
**begin**

**lemma** *atLeast-eq-atLeastAtMost-top* [*code*, *code-unfold*]:
‹$\{a..\} = \{a..top\}$›
⟨*proof*⟩

**lemma** *greaterThan-eq-greaterThanAtMost-top* [*code*, *code-unfold*]:
‹$\{a<..\} = \{a<..top\}$›
⟨*proof*⟩

**end**

**class** *interval-bot* = *interval* + *order-bot*
**begin**

**lemma** *atMost-eq-atLeastAtMost-bot* [*code*, *code-unfold*]:
‹$\{..b\} = \{bot..b\}$›
⟨*proof*⟩

**lemma** *lessThan-eq-atLeastLessThan-bot* [*code*, *code-unfold*]:
‹$\{..<b\} = \{bot..<b\}$›
⟨*proof*⟩

**end**

**instance** *nat* :: *interval-bot*
⟨*proof*⟩

**instance** *int* :: *interval*
⟨*proof*⟩

**context**
**begin**

**qualified lemma** *interval-eq-upt* [*simp*]:
‹*List.interval m n* = [*m*..<*Suc n*]›
⟨*proof*⟩ **lemma** *interval-eq-upto* [*simp*]:
‹*List.interval i k* = [*i*..*k*]›
⟨*proof*⟩

**end**

### 66.7.2 Special implementations

**context**
**begin**

**qualified definition** *map-tailrec-rev* :: ‹('*a* ⇒ '*b*) ⇒ '*a list* ⇒ '*b list* ⇒ '*b list*› — only for code generation
**where** *map-tailrec-rev* [*simp*]: ‹*map-tailrec-rev f as bs* = *rev* (*map f as*) @ *bs*›

Optional tail recursive version of *map*. Can avoid stack overflow in some target languages. Do not use for proving.

**qualified lemma** *map-tailrec-rev-code* [*code*, *no-atp*]:
‹*map-tailrec-rev f* [] *bs* = *bs*›
‹*map-tailrec-rev f* (*a* # *as*) *bs* = *map-tailrec-rev f as* (*f a* # *bs*)›
⟨*proof*⟩ **definition** *map-tailrec* :: ‹('*a* ⇒ '*b*) ⇒ '*a list* ⇒ '*b list*› — only for code generation
**where** *map-tailrec-eq* [*simp*]: ‹*map-tailrec* = *map*›

**qualified lemma** *map-tailrec-code* [*code*, *no-atp*]:
‹*map-tailrec f as* = *rev* (*map-tailrec-rev f as* [])›
⟨*proof*⟩

Potential code equation:

**qualified lemma** *map-eq-map-tailrec*:
‹*map* = *map-tailrec*›
⟨*proof*⟩

**end**

**definition** *map-filter* :: ‹('*a* ⇒ '*b option*) ⇒ '*a list* ⇒ '*b list*›
**where** [*code-post*]: *map-filter f xs* = *map* (*the* ∘ *f*) (*filter* (λ*x*. *f x* ≠ *None*) *xs*)

Operation *map-filter* avoids intermediate lists on execution – do not use for proving.

**lemma** *map-filter-simps* [*simp*, *code*, *no-atp*]:
‹*map-filter f* [] = []›
‹*map-filter f* (*x* # *xs*) = (*case f x of None* ⇒ *map-filter f xs* | *Some y* ⇒ *y* # *map-filter f xs*)›
⟨*proof*⟩

**lemma** *map-filter-map-filter* [*code-unfold*]:
‹*map f* (*filter P xs*) = *map-filter* (λ*x*. *if P x then Some* (*f x*) *else None*) *xs*›
⟨*proof*⟩

**hide-const** (**open**) *map-filter*

### 66.7.3 Operations for optimization and efficiency

**context**
**begin**

**qualified definition** *null* :: ‹′*a list* ⇒ *bool*› — only for code generation
**where** *null-iff* [*code-abbrev*, *simp*]: ‹*null xs* ⟷ *xs* = []›

**qualified lemma** *null-code* [*code*, *no-atp*]:
‹*null* [] ⟷ *True*›
‹*null* (*x* # *xs*) ⟷ *False*›
⟨*proof*⟩ **lemma** *equal-Nil-null* [*code-unfold*, *no-atp*]:
‹*HOL.equal xs* [] ⟷ *null xs*›
‹*HOL.equal* [] = *null*›
⟨*proof*⟩ **definition** *length-tailrec* :: ‹′*a list* ⇒ *nat* ⇒ *nat*› — only for code generation
**where** *length-tailrec-eq* [*simp*]: ‹*length-tailrec xs* = (+) (*length xs*)›

optimized code (tail-recursive) for *length*

**qualified lemma** *length-tailrec-code* [*code*, *no-atp*]:
‹*length-tailrec* [] *n* = *n*›
‹*length-tailrec* (*x* # *xs*) *n* = *length-tailrec xs* (*Suc n*)›
⟨*proof*⟩ **lemma** *length-code* [*code*, *no-atp*]:
‹*length xs* = *length-tailrec xs 0*›
⟨*proof*⟩ **definition** *maps* :: ‹(′*a* ⇒ ′*b list*) ⇒ ′*a list* ⇒ ′*b list*› — only for code generation
**where** *maps-eq* [*code-abbrev*, *simp*]: ‹*maps f xs* = *concat* (*map f xs*)›

Operation *maps* avoids intermediate lists on execution – do not use for proving.

**qualified lemma** *maps-code* [*code*, *no-atp*]:
‹*maps f* [] = []›
‹*maps f* (*x* # *xs*) = *f x* @ *maps f xs*›
⟨*proof*⟩

**end**

### 66.7.4 Implementation of sets by lists

**lemma** *is-empty-set* [*code*]:
*Set.is-empty* (*set xs*) ⟷ *List.null xs*
⟨*proof*⟩

**lemma** *empty-set* [*code*]:
$\{\} = set\ []$
⟨*proof*⟩

**lemma** *UNIV-coset* [*code*]:
$UNIV = List.coset\ []$
⟨*proof*⟩

**lemma** *compl-set* [*code*]:
$-\ set\ xs = List.coset\ xs$
⟨*proof*⟩

**lemma** *compl-coset* [*code*]:
$-\ List.coset\ xs = set\ xs$
⟨*proof*⟩

**lemma** [*code*]:
$x \in set\ xs \longleftrightarrow List.member\ xs\ x$
$x \in List.coset\ xs \longleftrightarrow \neg\ List.member\ xs\ x$
⟨*proof*⟩

**lemma** *insert-code* [*code*]:
$insert\ x\ (set\ xs) = set\ (List.insert\ x\ xs)$
$insert\ x\ (List.coset\ xs) = List.coset\ (removeAll\ x\ xs)$
⟨*proof*⟩

**lemma** *remove-code* [*code*]:
$Set.remove\ x\ (set\ xs) = set\ (removeAll\ x\ xs)$
$Set.remove\ x\ (List.coset\ xs) = List.coset\ (List.insert\ x\ xs)$
⟨*proof*⟩

**lemma** *filter-set* [*code*]:
$Set.filter\ P\ (set\ xs) = set\ (filter\ P\ xs)$
⟨*proof*⟩

**lemma** *image-set* [*code*]:
$image\ f\ (set\ xs) = set\ (map\ f\ xs)$
⟨*proof*⟩

**lemma** *subset-code* [*code*]:
$set\ xs \subseteq B \longleftrightarrow (\forall x{\in}set\ xs.\ x \in B)$
$A \subseteq List.coset\ ys \longleftrightarrow (\forall y{\in}set\ ys.\ y \notin A)$
$List.coset\ [] \subseteq set\ [] \longleftrightarrow False$
⟨*proof*⟩

**lemma** *Ball-set* [*code*]:
$Ball\ (set\ xs)\ P \longleftrightarrow list\text{-}all\ P\ xs$
⟨*proof*⟩

**lemma** *Bex-set* [*code*]:
$Bex\ (set\ xs)\ P \longleftrightarrow list\text{-}ex\ P\ xs$
⟨*proof*⟩

**lemma** *card-set* [*code*]:
$card\ (set\ xs) = length\ (remdups\ xs)$
⟨*proof*⟩

**lemma** *the-elem-set* [*code*]:
$the\text{-}elem\ (set\ [x]) = x$
⟨*proof*⟩

**lemma** *Pow-set* [*code*]:
$Pow\ (set\ []) = \{\{\}\}$
$Pow\ (set\ (x\ \#\ xs)) = (let\ A = Pow\ (set\ xs)\ in\ A \cup insert\ x\ `\ A)$
⟨*proof*⟩

**lemma** *these-set-code* [*code*]:
‹$Option.these\ (set\ xs) = set\ (List.map\text{-}filter\ (\lambda x.\ x)\ xs)$›
⟨*proof*⟩

**lemma** *image-filter-set-eq* [*code*]:
‹$Option.image\text{-}filter\ f\ (set\ xs) = set\ (List.map\text{-}filter\ f\ xs)$›
⟨*proof*⟩

**lemma** *can-select-set-list-ex1* [*code*]:
$Set.can\text{-}select\ P\ (set\ A) = list\text{-}ex1\ P\ A$
⟨*proof*⟩

**lemma** *product-code* [*code*]:
$Product\text{-}Type.product\ (set\ xs)\ (set\ ys) = set\ [(x,\ y).\ x \leftarrow xs,\ y \leftarrow ys]$
⟨*proof*⟩

**lemma** *Id-on-set* [*code*]:
$Id\text{-}on\ (set\ xs) = set\ [(x,\ x).\ x \leftarrow xs]$
⟨*proof*⟩

**lemma** *Image-code* [*code*]:
$R\ ``\ S = Option.image\text{-}filter\ (\lambda(x,\ y).\ if\ x \in S\ then\ Some\ y\ else\ None)\ R$
⟨*proof*⟩

**lemma** *trancl-set-ntrancl* [*code*]:
$trancl\ (set\ xs) = ntrancl\ (card\ (set\ xs) - 1)\ (set\ xs)$
⟨*proof*⟩

**lemma** *set-relcomp* [*code*]:
$set\ xys\ O\ set\ yzs = set\ ([(fst\ xy,\ snd\ yz).\ xy \leftarrow xys,\ yz \leftarrow yzs,\ snd\ xy = fst\ yz])$
⟨*proof*⟩

**lemma** *wf-set*:
*wf* (*set xs*) = *acyclic* (*set xs*)
⟨*proof*⟩

**lemma** *wf-code-set* [*code*]:
*wf-code* (*set xs*) = *acyclic* (*set xs*)
⟨*proof*⟩

### 66.7.5 Pretty lists

⟨*ML*⟩

**code-printing**
**type-constructor** *list* ⇀
(*SML*) - *list*
**and** (*OCaml*) - *list*
**and** (*Haskell*) ![(-)]
**and** (*Scala*) *List*[(-)]
| **constant** *Nil* ⇀
(*SML*) []
**and** (*OCaml*) []
**and** (*Haskell*) []
**and** (*Scala*) !*Nil*
| **class-instance** *list* :: *equal* ⇀
(*Haskell*) −
| **constant** *HOL.equal* :: *'a list* ⇒ *'a list* ⇒ *bool* ⇀
(*Haskell*) **infix** *4* ==

⟨*ML*⟩

**code-reserved**
(*SML*) *list*
**and** (*OCaml*) *list*

### 66.7.6 Use convenient predefined operations

**code-printing**
**constant** (@) ⇀
(*SML*) **infixr** *7* @
**and** (*OCaml*) **infixr** *6* @
**and** (*Haskell*) **infixr** *5* ++
**and** (*Scala*) **infixl** *7* ++
| **constant** *map* ⇀
(*Haskell*) *map*
| **constant** *filter* ⇀
(*Haskell*) *filter*
| **constant** *concat* ⇀
(*Haskell*) *concat*
| **constant** *List.maps* ⇀
(*Haskell*) *concatMap*

| **constant** *rev* ⇀
  (*Haskell*) *reverse*
| **constant** *zip* ⇀
  (*Haskell*) *zip*
| **constant** *List.null* ⇀
  (*Haskell*) *null*
| **constant** *takeWhile* ⇀
  (*Haskell*) *takeWhile*
| **constant** *dropWhile* ⇀
  (*Haskell*) *dropWhile*
| **constant** *list-all* ⇀
  (*Haskell*) *all*
| **constant** *list-ex* ⇀
  (*Haskell*) *any*

## 66.8 Setup for Lifting/Transfer

### 66.8.1 Transfer rules for the Transfer package

**context includes** *lifting-syntax*
**begin**

**lemma** *tl-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A*) *tl tl*
⟨*proof*⟩

**lemma** *butlast-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A*) *butlast butlast*
⟨*proof*⟩

**lemma** *append-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A* ===> *list-all2 A*) *append append*
⟨*proof*⟩

**lemma** *rev-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A*) *rev rev*
⟨*proof*⟩

**lemma** *filter-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> *list-all2 A*) *filter filter*
⟨*proof*⟩

**lemma** *fold-transfer* [*transfer-rule*]:
((*A* ===> *B* ===> *B*) ===> *list-all2 A* ===> *B* ===> *B*) *fold fold*
⟨*proof*⟩

**lemma** *foldr-transfer* [*transfer-rule*]:
((*A* ===> *B* ===> *B*) ===> *list-all2 A* ===> *B* ===> *B*) *foldr foldr*
⟨*proof*⟩

**lemma** *foldl-transfer* [*transfer-rule*]:
((*B* ===> *A* ===> *B*) ===> *B* ===> *list-all2 A* ===> *B*) *foldl foldl*
⟨*proof*⟩

**lemma** *concat-transfer* [*transfer-rule*]:
(*list-all2* (*list-all2 A*) ===> *list-all2 A*) *concat concat*
⟨*proof*⟩

**lemma** *drop-transfer* [*transfer-rule*]:
((=) ===> *list-all2 A* ===> *list-all2 A*) *drop drop*
⟨*proof*⟩

**lemma** *take-transfer* [*transfer-rule*]:
((=) ===> *list-all2 A* ===> *list-all2 A*) *take take*
⟨*proof*⟩

**lemma** *list-update-transfer* [*transfer-rule*]:
(*list-all2 A* ===> (=) ===> *A* ===> *list-all2 A*) *list-update list-update*
⟨*proof*⟩

**lemma** *takeWhile-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> *list-all2 A*) *takeWhile takeWhile*
⟨*proof*⟩

**lemma** *dropWhile-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> *list-all2 A*) *dropWhile dropWhile*
⟨*proof*⟩

**lemma** *zip-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 B* ===> *list-all2* (*rel-prod A B*)) *zip zip*
⟨*proof*⟩

**lemma** *product-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 B* ===> *list-all2* (*rel-prod A B*)) *List.product List.product*
⟨*proof*⟩

**lemma** *product-lists-transfer* [*transfer-rule*]:
(*list-all2* (*list-all2 A*) ===> *list-all2* (*list-all2 A*)) *product-lists product-lists*
⟨*proof*⟩

**lemma** *insert-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*A* ===> *list-all2 A* ===> *list-all2 A*) *List.insert List.insert*
⟨*proof*⟩

**lemma** *find-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> *rel-option A*) *List.find List.find*
⟨*proof*⟩

**lemma** *those-transfer* [*transfer-rule*]:
(*list-all2* (*rel-option P*) ===> *rel-option* (*list-all2 P*)) *those those*
⟨*proof*⟩

**lemma** *remove1-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*A* ===> *list-all2 A* ===> *list-all2 A*) *remove1 remove1*
⟨*proof*⟩

**lemma** *removeAll-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*A* ===> *list-all2 A* ===> *list-all2 A*) *removeAll removeAll*
⟨*proof*⟩

**lemma** *successively-transfer* [*transfer-rule*]:
((*A* ===> *A* ===> (=)) ===> *list-all2 A* ===> (=)) *successively successively*
⟨*proof*⟩

**lemma** *distinct-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*list-all2 A* ===> (=)) *distinct distinct*
⟨*proof*⟩

**lemma** *distinct-adj-transfer* [*transfer-rule*]:
**assumes** *bi-unique A*
**shows** (*list-all2 A* ===> (=)) *distinct-adj distinct-adj*
⟨*proof*⟩

**lemma** *remdups-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*list-all2 A* ===> *list-all2 A*) *remdups remdups*
⟨*proof*⟩

**lemma** *remdups-adj-transfer* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A*
**shows** (*list-all2 A* ===> *list-all2 A*) *remdups-adj remdups-adj*
⟨*proof*⟩

**lemma** *replicate-transfer* [*transfer-rule*]:
((=) ===> *A* ===> *list-all2 A*) *replicate replicate*
⟨*proof*⟩

**lemma** *length-transfer* [*transfer-rule*]:
(*list-all2 A* ===> (=)) *length length*
⟨*proof*⟩

**lemma** *rotate1-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A*) *rotate1 rotate1*
⟨*proof*⟩

**lemma** *rotate-transfer* [*transfer-rule*]:
((=) ===> *list-all2 A* ===> *list-all2 A*) *rotate rotate*
⟨*proof*⟩

**lemma** *nths-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *rel-set* (=) ===> *list-all2 A*) *nths nths*
⟨*proof*⟩

**lemma** *subseqs-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2* (*list-all2 A*)) *subseqs subseqs*
⟨*proof*⟩

**lemma** *partition-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> *rel-prod* (*list-all2 A*) (*list-all2 A*))
*partition partition*
⟨*proof*⟩

**lemma** *lists-transfer* [*transfer-rule*]:
(*rel-set A* ===> *rel-set* (*list-all2 A*)) *lists lists*
⟨*proof*⟩

**lemma** *set-Cons-transfer* [*transfer-rule*]:
(*rel-set A* ===> *rel-set* (*list-all2 A*) ===> *rel-set* (*list-all2 A*))
*set-Cons set-Cons*
⟨*proof*⟩

**lemma** *listset-transfer* [*transfer-rule*]:
(*list-all2* (*rel-set A*) ===> *rel-set* (*list-all2 A*)) *listset listset*
⟨*proof*⟩

**lemma** *null-transfer* [*transfer-rule*]:
(*list-all2 A* ===> (=)) *List.null List.null*
⟨*proof*⟩

**lemma** *list-all-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> (=)) *list-all list-all*
⟨*proof*⟩

**lemma** *list-ex-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *list-all2 A* ===> (=)) *list-ex list-ex*
⟨*proof*⟩

**lemma** *splice-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A* ===> *list-all2 A*) *splice splice*
⟨*proof*⟩

**lemma** *shuffles-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *list-all2 A* ===> *rel-set* (*list-all2 A*)) *shuffles shuffles*

⟨*proof*⟩

**lemma** *rtrancl-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-unique A bi-total A*
**shows** (*rel-set* (*rel-prod A A*) ===> *rel-set* (*rel-prod A A*)) *rtrancl rtrancl*
⟨*proof*⟩

**lemma** *monotone-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total A*
**shows** ((*A* ===> *A* ===> (=)) ===> (*B* ===> *B* ===> (=)) ===> (*A* ===> *B*) ===> (=)) *monotone monotone*
⟨*proof*⟩

**lemma** *fun-ord-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total C*
**shows** ((*A* ===> *B* ===> (=)) ===> (*C* ===> *A*) ===> (*C* ===> *B*) ===> (=)) *fun-ord fun-ord*
⟨*proof*⟩

**lemma** *fun-lub-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total A* *bi-unique A*
**shows** ((*rel-set A* ===> *B*) ===> *rel-set* (*C* ===> *A*) ===> *C* ===> *B*) *fun-lub fun-lub*
⟨*proof*⟩

**end**

### 66.9 Misc

**lemma** *Ball-set-list-all*:
*Ball* (*set xs*) *P* ⟷ *list-all P xs*
⟨*proof*⟩

**lemma** *Bex-set-list-ex*:
*Bex* (*set xs*) *P* ⟷ *list-ex P xs*
⟨*proof*⟩

**end**

## 67 Sum and product over lists

**theory** *Groups-List*
**imports** *List*
**begin**

**locale** *monoid-list* = *monoid*
**begin**

**definition** *F* :: *'a list* ⇒ *'a*

**where**
*eq-foldr* [*code*]: $F\ xs = foldr\ f\ xs\ \mathbf{1}$

**lemma** *Nil* [*simp*]:
$F\ [] = \mathbf{1}$
⟨*proof*⟩

**lemma** *Cons* [*simp*]:
$F\ (x\ \#\ xs) = x * F\ xs$
⟨*proof*⟩

**lemma** *append* [*simp*]:
$F\ (xs\ @\ ys) = F\ xs * F\ ys$
⟨*proof*⟩

**end**

**locale** *comm-monoid-list* = *comm-monoid* + *monoid-list*
**begin**

**lemma** *rev* [*simp*]:
$F\ (rev\ xs) = F\ xs$
⟨*proof*⟩

**end**

**locale** *comm-monoid-list-set* = *list*: *comm-monoid-list* + *set*: *comm-monoid-set*
**begin**

**lemma** *distinct-set-conv-list*:
$distinct\ xs \Longrightarrow set.F\ g\ (set\ xs) = list.F\ (map\ g\ xs)$
⟨*proof*⟩

**lemma** *set-conv-list* [*code*]:
$set.F\ g\ (set\ xs) = list.F\ (map\ g\ (remdups\ xs))$
⟨*proof*⟩

**lemma** *list-conv-set-nth*:
$list.F\ xs = set.F\ (\lambda i.\ xs\ !\ i)\ \{0..<length\ xs\}$
⟨*proof*⟩

**lemma** *atLeastAtMost-conv-list* [*code-unfold*]:
‹$set.F\ g\ \{a..b\} = list.F\ (map\ g\ (List.interval\ a\ b))$›
⟨*proof*⟩

**lemma** *atLeastLessThan-conv-list* [*code-unfold*]:
‹$set.F\ g\ \{a..<b\} = (let\ d = b - 1\ in\ if\ d < b$
$then\ list.F\ (map\ g\ (List.interval\ a\ d))$
$else\ \mathbf{1})$›

⟨proof⟩

**lemma** *greaterThanAtMost-conv-list* [*code-unfold*]:
‹*set.F g* {*a*<..*b*} = (*let c* = *a* + *1 in if a* < *c*
*then list.F* (*map g* (*List.interval c b*))
*else* **1**)›
⟨proof⟩

**lemma** *greaterThanLessThan-conv-list* [*code-unfold*]:
‹*set.F g* {*a*<..<*b*} = (*let c* = *a* + *1*; *d* = *b* − *1 in if a* < *c* ∧ *d* < *b*
*then list.F* (*map g* (*List.interval* (*a* + *1*) (*b* − *1*)))
*else* **1**)›
⟨proof⟩

**end**

## 67.1 List summation

**context** *monoid-add*
**begin**

**sublocale** *sum-list*: *monoid-list plus 0*
**defines**
*sum-list* = *sum-list.F* ⟨proof⟩

**end**

**context** *comm-monoid-add*
**begin**

**sublocale** *sum-list*: *comm-monoid-list plus 0*
**rewrites**
*monoid-list.F plus 0* = *sum-list*
⟨proof⟩

**sublocale** *sum*: *comm-monoid-list-set plus 0*
**rewrites**
*monoid-list.F plus 0* = *sum-list*
**and** *comm-monoid-set.F plus 0* = *sum*
⟨proof⟩

**end**

Some syntactic sugar for summing a function over a list:

**open-bundle** *sum-list-syntax*
**begin**

**syntax** (*ASCII*)
*-sum-list* :: *pttrn* => *'a list* => *'b* => *'b* (‹(‹*indent=3 notation*=‹*binder*

*SUM››SUM -<−-. -)› [0, 51, 10] 10)*
**syntax**
*-sum-list* :: *pttrn* => *'a list* => *'b* => *'b* (‹(‹*indent=3 notation=‹binder* $\sum$››$\sum$ -←-. -)› *[0, 51, 10] 10)*
**syntax-consts**
*-sum-list* == *sum-list*
**translations** — Beware of argument permutation!
$\sum x{\leftarrow}xs.\ b$ == *CONST sum-list* (*CONST map* ($\lambda x.\ b$) *xs*)

**end**

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *sum-list-transfer* [*transfer-rule*]:
(*list-all2 A* ===> *A*) *sum-list sum-list*
**if** [*transfer-rule*]: *A 0 0* (*A* ===> *A* ===> *A*) (+) (+)
⟨*proof*⟩

**end**

TODO duplicates

**lemmas** *sum-list-simps* = *sum-list.Nil sum-list.Cons*
**lemmas** *sum-list-append* = *sum-list.append*
**lemmas** *sum-list-rev* = *sum-list.rev*

**lemma** (**in** *monoid-add*) *fold-plus-sum-list-rev*:
*fold plus xs* = *plus* (*sum-list* (*rev xs*))
⟨*proof*⟩

**lemma** *sum-list-of-nat*: *sum-list* (*map of-nat xs*) = *of-nat* (*sum-list xs*)
⟨*proof*⟩

**lemma** *sum-list-of-int*: *sum-list* (*map of-int xs*) = *of-int* (*sum-list xs*)
⟨*proof*⟩

**lemma** *count-list-concat*: *count-list* (*concat xss*) *x* = *sum-list* (*map* ($\lambda xs.$ *count-list xs x*) *xss*)
⟨*proof*⟩

**lemma** (**in** *comm-monoid-add*) *sum-list-map-remove1*:
$x \in$ *set xs* $\Longrightarrow$ *sum-list* (*map f xs*) = *f x* + *sum-list* (*map f* (*remove1 x xs*))
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *size-list-conv-sum-list*:
*size-list f xs* = *sum-list* (*map f xs*) + *size xs*
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *length-concat*:
  $length\ (concat\ xss) = sum\text{-}list\ (map\ length\ xss)$
  ⟨*proof*⟩

**lemma** (**in** *monoid-add*) *length-product-lists*:
  $length\ (product\text{-}lists\ xss) = foldr\ (*)\ (map\ length\ xss)\ 1$
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *sum-list-map-filter*:
  **assumes** $\bigwedge x.\ x \in set\ xs \Longrightarrow \neg\ P\ x \Longrightarrow f\ x = 0$
  **shows** $sum\text{-}list\ (map\ f\ (filter\ P\ xs)) = sum\text{-}list\ (map\ f\ xs)$
  ⟨*proof*⟩

**lemma** *sum-list-filter-le-nat*:
  **fixes** $f :: 'a \Rightarrow nat$
  **shows** $sum\text{-}list\ (map\ f\ (filter\ P\ xs)) \leq sum\text{-}list\ (map\ f\ xs)$
⟨*proof*⟩

**lemma** (**in** *comm-monoid-add*) *distinct-sum-list-conv-Sum*:
  $distinct\ xs \Longrightarrow sum\text{-}list\ xs = Sum\ (set\ xs)$
  ⟨*proof*⟩

**lemma** *sum-list-upt*[*simp*]:
  $m \leq n \Longrightarrow sum\text{-}list\ [m..{<}n] = \sum\ \{m..{<}n\}$
⟨*proof*⟩

**context** *ordered-comm-monoid-add*
**begin**

**lemma** *sum-list-nonneg*: $(\bigwedge x.\ x \in set\ xs \Longrightarrow 0 \leq x) \Longrightarrow 0 \leq sum\text{-}list\ xs$
  ⟨*proof*⟩

**lemma** *sum-list-nonpos*: $(\bigwedge x.\ x \in set\ xs \Longrightarrow x \leq 0) \Longrightarrow sum\text{-}list\ xs \leq 0$
  ⟨*proof*⟩

**lemma** *sum-list-nonneg-eq-0-iff*:
  $(\bigwedge x.\ x \in set\ xs \Longrightarrow 0 \leq x) \Longrightarrow sum\text{-}list\ xs = 0 \longleftrightarrow (\forall x \in set\ xs.\ x = 0)$
  ⟨*proof*⟩

**end**

**context** *canonically-ordered-monoid-add*
**begin**

**lemma** *sum-list-eq-0-iff* [*simp*]:
  $sum\text{-}list\ ns = 0 \longleftrightarrow (\forall\, n \in set\ ns.\ n = 0)$
  ⟨*proof*⟩

**lemma** *member-le-sum-list*:

$x \in set\ xs \Longrightarrow x \leq sum\text{-}list\ xs$
⟨*proof*⟩

**lemma** *elem-le-sum-list*:
$k < size\ ns \Longrightarrow ns\ !\ k \leq sum\text{-}list\ (ns)$
⟨*proof*⟩

**end**

**lemma** (**in** *ordered-cancel-comm-monoid-diff*) *sum-list-update*:
$k < size\ xs \Longrightarrow sum\text{-}list\ (xs[k := x]) = sum\text{-}list\ xs + x - xs\ !\ k$
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *sum-list-triv*:
$(\sum x \leftarrow xs.\ r) = of\text{-}nat\ (length\ xs) * r$
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *sum-list-0* [*simp*]:
$(\sum x \leftarrow xs.\ 0) = 0$
⟨*proof*⟩

For non-Abelian groups *xs* needs to be reversed on one side:

**lemma** (**in** *ab-group-add*) *uminus-sum-list-map*:
$-\ sum\text{-}list\ (map\ f\ xs) = sum\text{-}list\ (map\ (uminus \circ f)\ xs)$
⟨*proof*⟩

**lemma** (**in** *comm-monoid-add*) *sum-list-addf*:
$(\sum x \leftarrow xs.\ f\ x + g\ x) = sum\text{-}list\ (map\ f\ xs) + sum\text{-}list\ (map\ g\ xs)$
⟨*proof*⟩

**lemma** (**in** *ab-group-add*) *sum-list-subtractf*:
$(\sum x \leftarrow xs.\ f\ x - g\ x) = sum\text{-}list\ (map\ f\ xs) - sum\text{-}list\ (map\ g\ xs)$
⟨*proof*⟩

**lemma** (**in** *semiring-0*) *sum-list-const-mult*:
$(\sum x \leftarrow xs.\ c * f\ x) = c * (\sum x \leftarrow xs.\ f\ x)$
⟨*proof*⟩

**lemma** (**in** *semiring-0*) *sum-list-mult-const*:
$(\sum x \leftarrow xs.\ f\ x * c) = (\sum x \leftarrow xs.\ f\ x) * c$
⟨*proof*⟩

**lemma** (**in** *ordered-ab-group-add-abs*) *sum-list-abs*:
$|sum\text{-}list\ xs| \leq sum\text{-}list\ (map\ abs\ xs)$
⟨*proof*⟩

**lemma** *sum-list-mono*:
**fixes** $f\ g :: 'a \Rightarrow 'b::\{monoid\text{-}add,\ ordered\text{-}ab\text{-}semigroup\text{-}add\}$
**shows** $(\bigwedge x.\ x \in set\ xs \Longrightarrow f\ x \leq g\ x) \Longrightarrow (\sum x \leftarrow xs.\ f\ x) \leq (\sum x \leftarrow xs.\ g\ x)$

$\langle proof \rangle$

**lemma** *sum-list-strict-mono*:
**fixes** $f\ g$ :: $'a \Rightarrow\ 'b$::{*monoid-add*, *strict-ordered-ab-semigroup-add*}
**shows** $[\![\ xs \neq [\,];\ \bigwedge x.\ x \in set\ xs \Longrightarrow f\ x < g\ x\ ]\!]$
$\Longrightarrow sum\text{-}list\ (map\ f\ xs) < sum\text{-}list\ (map\ g\ xs)$
$\langle proof \rangle$

A much more general version of this monotonicity lemma can be formulated with multisets and the multiset order

**lemma** *sum-list-mono2*: **fixes** $xs$ :: $'a$ ::*ordered-comm-monoid-add list*
**shows** $[\![\ length\ xs = length\ ys;\ \bigwedge i.\ i < length\ xs \longrightarrow xs!i \leq ys!i\ ]\!]$
$\Longrightarrow sum\text{-}list\ xs \leq sum\text{-}list\ ys$
$\langle proof \rangle$

**lemma** (**in** *monoid-add*) *sum-list-distinct-conv-sum-set*:
$distinct\ xs \Longrightarrow sum\text{-}list\ (map\ f\ xs) = sum\ f\ (set\ xs)$
$\langle proof \rangle$

**lemma** (**in** *monoid-add*) *interv-sum-list-conv-sum-set-nat*:
$sum\text{-}list\ (map\ f\ [m..<n]) = sum\ f\ (set\ [m..<n])$
$\langle proof \rangle$

**lemma** (**in** *monoid-add*) *interv-sum-list-conv-sum-set-int*:
$sum\text{-}list\ (map\ f\ [k..l]) = sum\ f\ (set\ [k..l])$
$\langle proof \rangle$

General equivalence between *sum-list* and *sum*

**lemma** (**in** *monoid-add*) *sum-list-sum-nth*:
$sum\text{-}list\ xs = (\sum\ i = 0\ ..< length\ xs.\ xs\ !\ i)$
$\langle proof \rangle$

**lemma** *sum-list-map-eq-sum-count*:
$sum\text{-}list\ (map\ f\ xs) = sum\ (\lambda x.\ count\text{-}list\ xs\ x * f\ x)\ (set\ xs)$
$\langle proof \rangle$

**lemma** *sum-list-map-eq-sum-count2*:
**assumes** $set\ xs \subseteq X$ *finite* $X$
**shows** $sum\text{-}list\ (map\ f\ xs) = sum\ (\lambda x.\ count\text{-}list\ xs\ x * f\ x)\ X$
$\langle proof \rangle$

**lemma** *sum-list-replicate*: $sum\text{-}list\ (replicate\ n\ c) = of\text{-}nat\ n * c$
$\langle proof \rangle$

**lemma** *sum-list-nonneg*:
$(\bigwedge x.\ x \in set\ xs \Longrightarrow (x :: 'a :: ordered\text{-}comm\text{-}monoid\text{-}add) \geq 0) \Longrightarrow sum\text{-}list\ xs \geq 0$
$\langle proof \rangle$

**lemma** *sum-list-Suc*:
$sum\text{-}list\ (map\ (\lambda x.\ Suc(f\ x))\ xs) = sum\text{-}list\ (map\ f\ xs) + length\ xs$
⟨*proof*⟩

**lemma** (**in** *monoid-add*) *sum-list-map-filter'*:
$sum\text{-}list\ (map\ f\ (filter\ P\ xs)) = sum\text{-}list\ (map\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ 0)\ xs)$
⟨*proof*⟩

Summation of a strictly ascending sequence with length $n$ can be upper-bounded by summation over $\{0..<n\}$.

**lemma** *sorted-wrt-less-sum-mono-lowerbound*:
**fixes** $f :: nat \Rightarrow ('b::ordered\text{-}comm\text{-}monoid\text{-}add)$
**assumes** *mono*: $\bigwedge x\ y.\ x \le y \Longrightarrow f\ x \le f\ y$
**shows** $sorted\text{-}wrt\ (<)\ ns \Longrightarrow$
$(\sum i \in \{0..<length\ ns\}.\ f\ i) \le (\sum i \leftarrow ns.\ f\ i)$
⟨*proof*⟩

**lemma** *member-le-sum-list*:
**fixes** $x :: 'a :: ordered\text{-}comm\text{-}monoid\text{-}add$
**assumes** $x \in set\ xs\ \bigwedge x.\ x \in set\ xs \Longrightarrow x \ge 0$
**shows** $x \le sum\text{-}list\ xs$
⟨*proof*⟩

## 67.2 Horner sums

**context** *comm-semiring-0*
**begin**

**definition** *horner-sum* :: ‹$('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b\ list \Rightarrow 'a$›
**where** *horner-sum-foldr*: ‹$horner\text{-}sum\ f\ a\ xs = foldr\ (\lambda x\ b.\ f\ x + a * b)\ xs\ 0$›

**lemma** *horner-sum-simps* [*simp*]:
‹$horner\text{-}sum\ f\ a\ [] = 0$›
‹$horner\text{-}sum\ f\ a\ (x\ \#\ xs) = f\ x + a * horner\text{-}sum\ f\ a\ xs$›
⟨*proof*⟩

**lemma** *horner-sum-eq-sum-funpow*:
‹$horner\text{-}sum\ f\ a\ xs = (\sum n = 0..<length\ xs.\ ((*)\ a\ \widehat{\ }\widehat{\ }\ n)\ (f\ (xs\ !\ n)))$›
⟨*proof*⟩

**end**

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *horner-sum-transfer* [*transfer-rule*]:

‹((B ===> A) ===> A ===> list-all2 B ===> A) horner-sum horner-sum›
**if** [transfer-rule]: ‹A 0 0›
**and** [transfer-rule]: ‹(A ===> A ===> A) (+) (+)›
**and** [transfer-rule]: ‹(A ===> A ===> A) (∗) (∗)›
⟨proof⟩

**end**

**context** comm-semiring-1
**begin**

**lemma** horner-sum-eq-sum:
‹horner-sum f a xs = ($\sum$ n = 0..<length xs. f (xs ! n) ∗ a ^ n)›
⟨proof⟩

**lemma** horner-sum-append:
‹horner-sum f a (xs @ ys) = horner-sum f a xs + a ^ length xs ∗ horner-sum f a ys›
⟨proof⟩

**end**

**context** linordered-semidom
**begin**

**lemma** horner-sum-nonnegative:
‹0 ≤ horner-sum of-bool 2 bs›
⟨proof⟩

**end**

**context** discrete-linordered-semidom
**begin**

**lemma** horner-sum-bound:
‹horner-sum of-bool 2 bs < 2 ^ length bs›
⟨proof⟩

**lemma** horner-sum-of-bool-2-less:
‹(horner-sum of-bool 2 bs) < 2 ^ length bs›
⟨proof⟩

**end**

**lemma** nat-horner-sum [simp]:
‹nat (horner-sum of-bool 2 bs) = horner-sum of-bool 2 bs›
⟨proof⟩

**context** discrete-linordered-semidom

**begin**

**lemma** *horner-sum-less-eq-iff-lexordp-eq*:
‹*horner-sum of-bool 2 bs* ≤ *horner-sum of-bool 2 cs* ⟷ *lexordp-eq* (*rev bs*) (*rev cs*)›
**if** ‹*length bs* = *length cs*›
⟨*proof*⟩

**lemma** *horner-sum-less-iff-lexordp*:
‹*horner-sum of-bool 2 bs* < *horner-sum of-bool 2 cs* ⟷ *ord-class.lexordp* (*rev bs*) (*rev cs*)›
**if** ‹*length bs* = *length cs*›
⟨*proof*⟩

**end**

## 67.3 Further facts about *List.n-lists*

**lemma** *length-n-lists*: *length* (*List.n-lists n xs*) = *length xs* ^ *n*
⟨*proof*⟩

**lemma** *distinct-n-lists*:
**assumes** *distinct xs*
**shows** *distinct* (*List.n-lists n xs*)
⟨*proof*⟩

## 67.4 Tools setup

**lemmas** *sum-code* = *sum.set-conv-list*

**lemma** *sum-set-upto-conv-sum-list-int*:
*sum f* (*set* [*i*..*j*::*int*]) = *sum-list* (*map f* [*i*..*j*])
⟨*proof*⟩

**lemma** *sum-set-upt-conv-sum-list-nat*:
*sum f* (*set* [*m*..<*n*]) = *sum-list* (*map f* [*m*..<*n*])
⟨*proof*⟩

## 67.5 List product

**context** *monoid-mult*
**begin**

**sublocale** *prod-list*: *monoid-list times 1*
**defines**
*prod-list* = *prod-list.F* ⟨*proof*⟩

**end**

**context** *comm-monoid-mult*

**begin**

**sublocale** *prod-list*: *comm-monoid-list times 1*
**rewrites**
*monoid-list.F times 1 = prod-list*
⟨*proof*⟩

**sublocale** *prod*: *comm-monoid-list-set times 1*
**rewrites**
*monoid-list.F times 1 = prod-list*
**and** *comm-monoid-set.F times 1 = prod*
⟨*proof*⟩

**end**

Some syntactic sugar:

**open-bundle** *prod-list-syntax*
**begin**

**syntax** (*ASCII*)
*-prod-list* :: *pttrn => 'a list => 'b => 'b* (‹(‹indent=3 notation=‹binder PROD››PROD -<−-. -)› [*0*, *51*, *10*] *10*)
**syntax**
*-prod-list* :: *pttrn => 'a list => 'b => 'b* (‹(‹indent=3 notation=‹binder ∏››∏ -←-. -)› [*0*, *51*, *10*] *10*)
**syntax-consts**
*-prod-list* ⇌ *prod-list*
**translations** — Beware of argument permutation!
$\prod x{\leftarrow}xs.\ b \rightleftharpoons CONST\ prod\text{-}list\ (CONST\ map\ (\lambda x.\ b)\ xs)$

**end**

**context**
**includes** *lifting-syntax*
**begin**

**lemma** *prod-list-transfer* [*transfer-rule*]:
(*list-all2 A ===> A*) *prod-list prod-list*
**if** [*transfer-rule*]: *A 1 1* (*A ===> A ===> A*) (∗) (∗)
⟨*proof*⟩

**end**

**lemma** *prod-list-zero-iff*:
$prod\text{-}list\ xs = 0 \longleftrightarrow (0 :: 'a :: \{semiring\text{-}no\text{-}zero\text{-}divisors,\ semiring\text{-}1\}) \in set\ xs$
⟨*proof*⟩

**lemma** *prod-list-nonneg*: $(\bigwedge x.\ (x :: 'a :: ordered\text{-}semiring\text{-}1) \in set\ xs \Longrightarrow x \geq 0) \Longrightarrow prod\text{-}list\ xs \geq 0$

⟨*proof*⟩

**lemma** *prod-list-replicate*[*simp*]: *prod-list* (*replicate n a*) = *a* ^ *n*
⟨*proof*⟩

**lemma** *prod-list-power*:
**fixes** *xs* :: *'a* :: *comm-monoid-mult list*
**shows** *prod-list xs* ^ *n* = (∏ *x*←*xs*. *x* ^ *n*)
⟨*proof*⟩

**lemma** *prod-list-dvd*:
**assumes** (*x* :: *'a* :: *comm-monoid-mult*) ∈ *set xs*
**shows** *x dvd prod-list xs*
⟨*proof*⟩

**end**

# 68 Bit operations in suitable algebraic structures

**theory** *Bit-Operations*
**imports** *Presburger Groups-List*
**begin**

## 68.1 Abstract bit structures

**class** *semiring-bits* = *semiring-parity* + *semiring-modulo-trivial* +
**assumes** *bit-induct* [*case-names stable rec*]:
‹(⋀*a*. *a div 2* = *a* ⟹ *P a*)
⟹ (⋀*a b*. *P a* ⟹ (*of-bool b* + *2* ∗ *a*) *div 2* = *a* ⟹ *P* (*of-bool b* + *2* ∗ *a*))
⟹ *P a*›
**assumes** *bits-mod-div-trivial* [*simp*]: ‹*a mod b div b* = *0*›
**and** *half-div-exp-eq*: ‹*a div 2 div 2* ^ *n* = *a div 2* ^ *Suc n*›
**and** *even-double-div-exp-iff*: ‹*2* ^ *Suc n* ≠ *0* ⟹ *even* (*2* ∗ *a div 2* ^ *Suc n*) ⟷ *even* (*a div 2* ^ *n*)›
**fixes** *bit* :: ‹*'a* ⇒ *nat* ⇒ *bool*›
**assumes** *bit-iff-odd*: ‹*bit a n* ⟷ *odd* (*a div 2* ^ *n*)›
**begin**

Having *bit* as definitional class operation takes into account that specific instances can be implemented differently wrt. code generation.

**lemma** *half-1* [*simp*]:
‹*1 div 2* = *0*›
⟨*proof*⟩

**lemma** *div-exp-eq-funpow-half*:
‹*a div 2* ^ *n* = ((λ*a*. *a div 2*) ^^ *n*) *a*›
⟨*proof*⟩

**lemma** *div-exp-eq*:

‹$a$ *div* $2$ ^ $m$ *div* $2$ ^ $n$ = $a$ *div* $2$ ^ ($m$ + $n$)›
⟨*proof*⟩

**lemma** *bit-0*:
‹*bit a 0* ⟷ *odd a*›
⟨*proof*⟩

**lemma** *bit-Suc*:
‹*bit a* (*Suc n*) ⟷ *bit* (*a div 2*) *n*›
⟨*proof*⟩

**lemma** *bit-rec*:
‹*bit a n* ⟷ (*if n* = *0 then odd a else bit* (*a div 2*) (*n* − *1*))›
⟨*proof*⟩

**context**
**fixes** *a*
**assumes** *stable*: ‹*a div 2* = *a*›
**begin**

**lemma** *bits-stable-imp-add-self*:
‹*a* + *a mod 2* = *0*›
⟨*proof*⟩

**lemma** *stable-imp-bit-iff-odd*:
‹*bit a n* ⟷ *odd a*›
⟨*proof*⟩

**end**

**lemma** *bit-iff-odd-imp-stable*:
‹*a div 2* = *a*› **if** ‹⋀*n. bit a n* ⟷ *odd a*›
⟨*proof*⟩

**lemma** *even-succ-div-exp* [*simp*]:
‹(*1* + *a*) *div 2* ^ *n* = *a div 2* ^ *n*› **if** ‹*even a*› **and** ‹*n* > *0*›
⟨*proof*⟩

**lemma** *even-succ-mod-exp* [*simp*]:
‹(*1* + *a*) *mod 2* ^ *n* = *1* + (*a mod 2* ^ *n*)› **if** ‹*even a*› **and** ‹*n* > *0*›
⟨*proof*⟩

**lemma** *half-numeral-Bit1-eq* [*simp*]:
‹*numeral* (*num.Bit1 m*) *div 2* = *numeral* (*num.Bit0 m*) *div 2*›
⟨*proof*⟩

**lemma** *double-half-numeral-Bit-0-eq* [*simp*]:
‹*2* ∗ (*numeral* (*num.Bit0 m*) *div 2*) = *numeral* (*num.Bit0 m*)›
‹(*numeral* (*num.Bit0 m*) *div 2*) ∗ *2* = *numeral* (*num.Bit0 m*)›

⟨proof⟩

**named-theorems** *bit-simps* ‹*Simplification rules for* ***const*** ‹*bit*››

**definition** *possible-bit* :: ‹*'a itself* ⇒ *nat* ⇒ *bool*›
**where** ‹*possible-bit TYPE('a) n* ⟷ *2 ^ n* ≠ *0*›
— This auxiliary avoids non-termination with extensionality.

**lemma** *possible-bit-0* [*simp*]:
‹*possible-bit TYPE('a) 0*›
⟨proof⟩

**lemma** *fold-possible-bit*:
‹*2 ^ n* = *0* ⟷ ¬ *possible-bit TYPE('a) n*›
⟨proof⟩

**lemma** *bit-imp-possible-bit*:
‹*possible-bit TYPE('a) n*› **if** ‹*bit a n*›
⟨proof⟩

**lemma** *impossible-bit*:
‹¬ *bit a n*› **if** ‹¬ *possible-bit TYPE('a) n*›
⟨proof⟩

**lemma** *possible-bit-less-imp*:
‹*possible-bit TYPE('a) j*› **if** ‹*possible-bit TYPE('a) i*› ‹*j* ≤ *i*›
⟨proof⟩

**lemma** *possible-bit-min* [*simp*]:
‹*possible-bit TYPE('a) (min i j)* ⟷ *possible-bit TYPE('a) i* ∨ *possible-bit TYPE('a) j*›
⟨proof⟩

**lemma** *bit-eqI*:
‹*a* = *b*› **if** ‹⋀*n*. *possible-bit TYPE('a) n* ⟹ *bit a n* ⟷ *bit b n*›
⟨proof⟩

**lemma** *bit-eq-rec*:
‹*a* = *b* ⟷ (*even a* ⟷ *even b*) ∧ *a div 2* = *b div 2*› (**is** ‹*?P* = *?Q*›)
⟨proof⟩

**lemma** *bit-eq-iff*:
‹*a* = *b* ⟷ (∀ *n*. *possible-bit TYPE('a) n* ⟶ *bit a n* ⟷ *bit b n*)›
⟨proof⟩

**lemma** *bit-0-eq* [*simp*]:
‹*bit 0* = ⊥›
⟨proof⟩

**lemma** *bit-double-Suc-iff*:
‹$bit\ (2 * a)\ (Suc\ n) \longleftrightarrow possible\text{-}bit\ TYPE('a)\ (Suc\ n) \wedge bit\ a\ n$›
⟨*proof*⟩

**lemma** *bit-double-iff* [*bit-simps*]:
‹$bit\ (2 * a)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n \neq 0 \wedge bit\ a\ (n - 1)$›
⟨*proof*⟩

**lemma** *even-bit-succ-iff*:
‹$bit\ (1 + a)\ n \longleftrightarrow bit\ a\ n \vee n = 0$› **if** ‹*even a*›
⟨*proof*⟩

**lemma** *odd-bit-iff-bit-pred*:
‹$bit\ a\ n \longleftrightarrow bit\ (a - 1)\ n \vee n = 0$› **if** ‹*odd a*›
⟨*proof*⟩

**lemma** *bit-exp-iff* [*bit-simps*]:
‹$bit\ (2\ \widehat{}\ m)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n = m$›
⟨*proof*⟩

**lemma** *bit-1-iff* [*bit-simps*]:
‹$bit\ 1\ n \longleftrightarrow n = 0$›
⟨*proof*⟩

**lemma** *bit-2-iff* [*bit-simps*]:
‹$bit\ 2\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ 1 \wedge n = 1$›
⟨*proof*⟩

**lemma** *bit-of-bool-iff* [*bit-simps*]:
‹$bit\ (of\text{-}bool\ b)\ n \longleftrightarrow n = 0 \wedge b$›
⟨*proof*⟩

**lemma** *bit-mod-2-iff* [*simp*]:
‹$bit\ (a\ mod\ 2)\ n \longleftrightarrow n = 0 \wedge odd\ a$›
⟨*proof*⟩

**lemma** *stable-index*:
**obtains** *m* **where** ‹$possible\text{-}bit\ TYPE('a)\ m$›
‹$\bigwedge n.\ possible\text{-}bit\ TYPE('a)\ n \Longrightarrow n \geq m \Longrightarrow bit\ a\ n \longleftrightarrow bit\ a\ m$›
⟨*proof*⟩

**end**

**lemma** *nat-bit-induct* [*case-names zero even odd*]:
‹$P\ n$› **if** *zero*: ‹$P\ 0$›
**and** *even*: ‹$\bigwedge n.\ P\ n \Longrightarrow n > 0 \Longrightarrow P\ (2 * n)$›
**and** *odd*: ‹$\bigwedge n.\ P\ n \Longrightarrow P\ (Suc\ (2 * n))$›
⟨*proof*⟩

**instantiation** *nat* :: *semiring-bits*
**begin**

**definition** *bit-nat* :: ‹$nat \Rightarrow nat \Rightarrow bool$›
  **where** ‹$bit\text{-}nat\ m\ n \longleftrightarrow odd\ (m\ div\ 2\ \hat{}\ n)$›

**instance**
⟨*proof*⟩

**end**

**lemma** *possible-bit-nat* [*simp*]:
  ‹*possible-bit TYPE*(*nat*) *n*›
  ⟨*proof*⟩

**lemma** *bit-Suc-0-iff* [*bit-simps*]:
  ‹$bit\ (Suc\ 0)\ n \longleftrightarrow n = 0$›
  ⟨*proof*⟩

**lemma** *not-bit-Suc-0-Suc* [*simp*]:
  ‹$\neg\ bit\ (Suc\ 0)\ (Suc\ n)$›
  ⟨*proof*⟩

**lemma** *not-bit-Suc-0-numeral* [*simp*]:
  ‹$\neg\ bit\ (Suc\ 0)\ (numeral\ n)$›
  ⟨*proof*⟩

**context** *semiring-bits*
**begin**

**lemma** *bit-of-nat-iff* [*bit-simps*]:
  ‹$bit\ (of\text{-}nat\ m)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge bit\ m\ n$›
⟨*proof*⟩

**end**

**lemma** *int-bit-induct* [*case-names zero minus even odd*]:
  ‹*P k*› **if** *zero-int*: ‹$P\ 0$›
    **and** *minus-int*: ‹$P\ (-\ 1)$›
    **and** *even-int*: ‹$\bigwedge k.\ P\ k \Longrightarrow k \neq 0 \Longrightarrow P\ (k * 2)$›
    **and** *odd-int*: ‹$\bigwedge k.\ P\ k \Longrightarrow k \neq -\ 1 \Longrightarrow P\ (1 + (k * 2))$› **for** *k* :: *int*
⟨*proof*⟩

**instantiation** *int* :: *semiring-bits*
**begin**

**definition** *bit-int* :: ‹$int \Rightarrow nat \Rightarrow bool$›
  **where** ‹$bit\text{-}int\ k\ n \longleftrightarrow odd\ (k\ div\ 2\ \hat{}\ n)$›

**instance**
⟨*proof*⟩

**end**

**lemma** *possible-bit-int* [*simp*]:
‹*possible-bit TYPE*(*int*) *n*›
⟨*proof*⟩

**lemma** *bit-nat-iff* [*bit-simps*]:
‹$bit\ (nat\ k)\ n \longleftrightarrow k \geq 0 \wedge bit\ k\ n$›
⟨*proof*⟩

## 68.2 Bit operations

**class** *semiring-bit-operations* = *semiring-bits* +
**fixes** *and* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹*AND*› *64*)
**and** *or* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹*OR*› *59*)
**and** *xor* :: ‹$'a \Rightarrow 'a \Rightarrow 'a$› (**infixr** ‹*XOR*› *59*)
**and** *mask* :: ‹$nat \Rightarrow 'a$›
**and** *set-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**and** *unset-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**and** *flip-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**and** *push-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**and** *drop-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**and** *take-bit* :: ‹$nat \Rightarrow 'a \Rightarrow 'a$›
**assumes** *and-rec*: ‹*a AND b* = *of-bool* (*odd a* ∧ *odd b*) + *2* ∗ ((*a div 2*) *AND* (*b div 2*))›
**and** *or-rec*: ‹*a OR b* = *of-bool* (*odd a* ∨ *odd b*) + *2* ∗ ((*a div 2*) *OR* (*b div 2*))›
**and** *xor-rec*: ‹*a XOR b* = *of-bool* (*odd a* ≠ *odd b*) + *2* ∗ ((*a div 2*) *XOR* (*b div 2*))›
**and** *mask-eq-exp-minus-1*: ‹*mask n* = *2* ^ *n* − *1*›
**and** *set-bit-eq-or*: ‹*set-bit n a* = *a OR push-bit n 1*›
**and** *unset-bit-eq-or-xor*: ‹*unset-bit n a* = (*a OR push-bit n 1*) *XOR push-bit n 1*›
**and** *flip-bit-eq-xor*: ‹*flip-bit n a* = *a XOR push-bit n 1*›
**and** *push-bit-eq-mult*: ‹*push-bit n a* = *a* ∗ *2* ^ *n*›
**and** *drop-bit-eq-div*: ‹*drop-bit n a* = *a div 2* ^ *n*›
**and** *take-bit-eq-mod*: ‹*take-bit n a* = *a mod 2* ^ *n*›
**begin**

We want the bitwise operations to bind slightly weaker than + and −.

Logically, *push-bit*, *drop-bit* and *take-bit* are just aliases; having them as separate operations makes proofs easier, otherwise proof automation would fiddle with concrete expressions $(2::'a)^n$ in a way obfuscating the basic algebraic relationships between those operations.

For the sake of code generation operations are specified as definitional class operations, taking into account that specific instances of these can be implemented differently wrt. code generation.

**lemma** *bit-iff-odd-drop-bit*:
‹$bit\ a\ n \longleftrightarrow odd\ (drop\text{-}bit\ n\ a)$›
⟨*proof*⟩

**lemma** *even-drop-bit-iff-not-bit*:
‹$even\ (drop\text{-}bit\ n\ a) \longleftrightarrow \neg\ bit\ a\ n$›
⟨*proof*⟩

**lemma** *bit-and-iff* [*bit-simps*]:
‹$bit\ (a\ AND\ b)\ n \longleftrightarrow bit\ a\ n \wedge bit\ b\ n$›
⟨*proof*⟩

**lemma** *bit-or-iff* [*bit-simps*]:
‹$bit\ (a\ OR\ b)\ n \longleftrightarrow bit\ a\ n \vee bit\ b\ n$›
⟨*proof*⟩

**lemma** *bit-xor-iff* [*bit-simps*]:
‹$bit\ (a\ XOR\ b)\ n \longleftrightarrow bit\ a\ n \neq bit\ b\ n$›
⟨*proof*⟩

**sublocale** *and*: *semilattice* ‹$(AND)$›
⟨*proof*⟩

**sublocale** *or*: *semilattice-neutr* ‹$(OR)$› *0*
⟨*proof*⟩

**sublocale** *xor*: *comm-monoid* ‹$(XOR)$› *0*
⟨*proof*⟩

**lemma** *even-and-iff*:
‹$even\ (a\ AND\ b) \longleftrightarrow even\ a \vee even\ b$›
⟨*proof*⟩

**lemma** *even-or-iff*:
‹$even\ (a\ OR\ b) \longleftrightarrow even\ a \wedge even\ b$›
⟨*proof*⟩

**lemma** *even-xor-iff*:
‹$even\ (a\ XOR\ b) \longleftrightarrow (even\ a \longleftrightarrow even\ b)$›
⟨*proof*⟩

**lemma** *zero-and-eq* [*simp*]:
‹$0\ AND\ a = 0$›
⟨*proof*⟩

**lemma** *and-zero-eq* [*simp*]:
‹$a\ AND\ 0 = 0$›
⟨*proof*⟩

**lemma** *one-and-eq*:
‹$1$ $AND$ $a = a$ $mod$ $2$›
⟨*proof*⟩

**lemma** *and-one-eq*:
‹$a$ $AND$ $1 = a$ $mod$ $2$›
⟨*proof*⟩

**lemma** *one-or-eq*:
‹$1$ $OR$ $a = a + of\text{-}bool\ (even\ a)$›
⟨*proof*⟩

**lemma** *or-one-eq*:
‹$a$ $OR$ $1 = a + of\text{-}bool\ (even\ a)$›
⟨*proof*⟩

**lemma** *one-xor-eq*:
‹$1$ $XOR$ $a = a + of\text{-}bool\ (even\ a) - of\text{-}bool\ (odd\ a)$›
⟨*proof*⟩

**lemma** *xor-one-eq*:
‹$a$ $XOR$ $1 = a + of\text{-}bool\ (even\ a) - of\text{-}bool\ (odd\ a)$›
⟨*proof*⟩

**lemma** *xor-self-eq* [*simp*]:
‹$a$ $XOR$ $a = 0$›
⟨*proof*⟩

**lemma** *mask-0* [*simp*]:
‹$mask\ 0 = 0$›
⟨*proof*⟩

**lemma** *inc-mask-eq-exp*:
‹$mask\ n + 1 = 2\ \hat{}\ n$›
⟨*proof*⟩

**lemma** *mask-eq-iff-eq-exp*:
‹$mask\ n = a \longleftrightarrow a + 1 = 2\ \hat{}\ n$›
⟨*proof*⟩

**lemma** *eq-mask-iff-eq-exp*:
‹$a = mask\ n \longleftrightarrow a + 1 = 2\ \hat{}\ n$›
⟨*proof*⟩

**lemma** *mask-Suc-double*:
‹$mask\ (Suc\ n) = 1\ OR\ 2 * mask\ n$›
⟨*proof*⟩

**lemma** *bit-mask-iff* [*bit-simps*]:
‹$bit\ (mask\ m)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n < m$›
⟨*proof*⟩

**lemma** *even-mask-iff*:
‹$even\ (mask\ n) \longleftrightarrow n = 0$›
⟨*proof*⟩

**lemma** *mask-Suc-0* [*simp*]:
‹$mask\ (Suc\ 0) = 1$›
⟨*proof*⟩

**lemma** *mask-Suc-exp*:
‹$mask\ (Suc\ n) = 2\ \hat{}\ n\ OR\ mask\ n$›
⟨*proof*⟩

**lemma** *mask-numeral*:
‹$mask\ (numeral\ n) = 1 + 2 * mask\ (pred\text{-}numeral\ n)$›
⟨*proof*⟩

**lemma** *push-bit-0-id* [*simp*]:
‹$push\text{-}bit\ 0 = id$›
⟨*proof*⟩

**lemma** *push-bit-Suc* [*simp*]:
‹$push\text{-}bit\ (Suc\ n)\ a = push\text{-}bit\ n\ (a * 2)$›
⟨*proof*⟩

**lemma** *push-bit-double*:
‹$push\text{-}bit\ n\ (a * 2) = push\text{-}bit\ n\ a * 2$›
⟨*proof*⟩

**lemma** *bit-push-bit-iff* [*bit-simps*]:
‹$bit\ (push\text{-}bit\ m\ a)\ n \longleftrightarrow m \leq n \wedge possible\text{-}bit\ TYPE('a)\ n \wedge bit\ a\ (n - m)$›
⟨*proof*⟩

**lemma** *funpow-double-eq-push-bit*:
‹$times\ 2\ \hat{}\hat{}\ n = push\text{-}bit\ n$›
⟨*proof*⟩

**lemma** *div-push-bit-of-1-eq-drop-bit*:
‹$a\ div\ push\text{-}bit\ n\ 1 = drop\text{-}bit\ n\ a$›
⟨*proof*⟩

**lemma** *bits-ident*:
‹$push\text{-}bit\ n\ (drop\text{-}bit\ n\ a) + take\text{-}bit\ n\ a = a$›
⟨*proof*⟩

**lemma** *push-bit-push-bit* [*simp*]:

‹*push-bit m* (*push-bit n a*) = *push-bit* (*m* + *n*) *a*›
⟨*proof*⟩

**lemma** *push-bit-of-0* [*simp*]:
‹*push-bit n 0* = *0*›
⟨*proof*⟩

**lemma** *push-bit-of-1* [*simp*]:
‹*push-bit n 1* = *2* ^ *n*›
⟨*proof*⟩

**lemma** *push-bit-add*:
‹*push-bit n* (*a* + *b*) = *push-bit n a* + *push-bit n b*›
⟨*proof*⟩

**lemma** *push-bit-numeral* [*simp*]:
‹*push-bit* (*numeral l*) (*numeral k*) = *push-bit* (*pred-numeral l*) (*numeral* (*Num.Bit0* *k*))›
⟨*proof*⟩

**lemma** *bit-drop-bit-eq* [*bit-simps*]:
‹*bit* (*drop-bit n a*) = *bit a* ∘ (+) *n*›
⟨*proof*⟩

**lemma** *disjunctive-xor-eq-or*:
‹*a XOR b* = *a OR b*› **if** ‹*a AND b* = *0*›
⟨*proof*⟩

**lemma** *disjunctive-add-eq-or*:
‹*a* + *b* = *a OR b*› **if** ‹*a AND b* = *0*›
⟨*proof*⟩

**lemma** *disjunctive-add-eq-xor*:
‹*a* + *b* = *a XOR b*› **if** ‹*a AND b* = *0*›
⟨*proof*⟩

**lemma** *take-bit-0* [*simp*]:
*take-bit 0 a* = *0*
⟨*proof*⟩

**lemma** *bit-take-bit-iff* [*bit-simps*]:
‹*bit* (*take-bit m a*) *n* ⟷ *n* < *m* ∧ *bit a n*›
⟨*proof*⟩

**lemma** *take-bit-Suc*:
‹*take-bit* (*Suc n*) *a* = *take-bit n* (*a div 2*) ∗ *2* + *a mod 2*› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

**lemma** *take-bit-rec*:

‹*take-bit n a* = (*if n* = *0 then 0 else take-bit* (*n* − *1*) (*a div 2*) ∗ *2* + *a mod 2*)›
⟨*proof*⟩

**lemma** *take-bit-Suc-0* [*simp*]:
‹*take-bit* (*Suc 0*) *a* = *a mod 2*›
⟨*proof*⟩

**lemma** *take-bit-of-0* [*simp*]:
‹*take-bit n 0* = *0*›
⟨*proof*⟩

**lemma** *take-bit-of-1* [*simp*]:
‹*take-bit n 1* = *of-bool* (*n* > *0*)›
⟨*proof*⟩

**lemma** *drop-bit-of-0* [*simp*]:
‹*drop-bit n 0* = *0*›
⟨*proof*⟩

**lemma** *drop-bit-of-1* [*simp*]:
‹*drop-bit n 1* = *of-bool* (*n* = *0*)›
⟨*proof*⟩

**lemma** *drop-bit-0* [*simp*]:
‹*drop-bit 0* = *id*›
⟨*proof*⟩

**lemma** *drop-bit-Suc*:
‹*drop-bit* (*Suc n*) *a* = *drop-bit n* (*a div 2*)›
⟨*proof*⟩

**lemma** *drop-bit-rec*:
‹*drop-bit n a* = (*if n* = *0 then a else drop-bit* (*n* − *1*) (*a div 2*))›
⟨*proof*⟩

**lemma** *drop-bit-half*:
‹*drop-bit n* (*a div 2*) = *drop-bit n a div 2*›
⟨*proof*⟩

**lemma** *drop-bit-of-bool* [*simp*]:
‹*drop-bit n* (*of-bool b*) = *of-bool* (*n* = *0* ∧ *b*)›
⟨*proof*⟩

**lemma** *even-take-bit-eq* [*simp*]:
‹*even* (*take-bit n a*) ⟷ *n* = *0* ∨ *even a*›
⟨*proof*⟩

**lemma** *take-bit-take-bit* [*simp*]:
‹*take-bit m* (*take-bit n a*) = *take-bit* (*min m n*) *a*›

⟨proof⟩

**lemma** *drop-bit-drop-bit* [*simp*]:
‹*drop-bit m* (*drop-bit n a*) = *drop-bit* (*m* + *n*) *a*›
⟨proof⟩

**lemma** *push-bit-take-bit*:
‹*push-bit m* (*take-bit n a*) = *take-bit* (*m* + *n*) (*push-bit m a*)›
⟨proof⟩

**lemma** *take-bit-push-bit*:
‹*take-bit m* (*push-bit n a*) = *push-bit n* (*take-bit* (*m* − *n*) *a*)›
⟨proof⟩

**lemma** *take-bit-drop-bit*:
‹*take-bit m* (*drop-bit n a*) = *drop-bit n* (*take-bit* (*m* + *n*) *a*)›
⟨proof⟩

**lemma** *drop-bit-take-bit*:
‹*drop-bit m* (*take-bit n a*) = *take-bit* (*n* − *m*) (*drop-bit m a*)›
⟨proof⟩

**lemma** *even-push-bit-iff* [*simp*]:
‹*even* (*push-bit n a*) ⟷ *n* ≠ *0* ∨ *even a*›
⟨proof⟩

**lemma** *stable-imp-drop-bit-eq*:
‹*drop-bit n a* = *a*›
**if** ‹*a div 2* = *a*›
⟨proof⟩

**lemma** *stable-imp-take-bit-eq*:
‹*take-bit n a* = (*if even a then 0 else mask n*)›
**if** ‹*a div 2* = *a*›
⟨proof⟩

**lemma** *exp-dvdE*:
**assumes** ‹*2* ^ *n dvd a*›
**obtains** *b* **where** ‹*a* = *push-bit n b*›
⟨proof⟩

**lemma** *take-bit-eq-0-iff*:
‹*take-bit n a* = *0* ⟷ *2* ^ *n dvd a*› (**is** ‹*?P* ⟷ *?Q*›)
⟨proof⟩

**lemma** *take-bit-tightened*:
‹*take-bit m a* = *take-bit m b*› **if** ‹*take-bit n a* = *take-bit n b*› **and** ‹*m* ≤ *n*›
⟨proof⟩

**lemma** *take-bit-eq-self-iff-drop-bit-eq-0*:
‹*take-bit n a = a* ⟷ *drop-bit n a = 0*› (**is** ‹*?P* ⟷ *?Q*›)
⟨*proof*⟩

**lemma** *impossible-bit-imp-take-bit-eq-self*:
‹*take-bit n a = a*› **if** ‹¬ *possible-bit TYPE('a) n*›
⟨*proof*⟩

**lemma** *drop-bit-exp-eq*:
‹*drop-bit m (2 ^ n) = of-bool (m ≤ n ∧ possible-bit TYPE('a) n) * 2 ^ (n − m)*›
⟨*proof*⟩

**lemma** *take-bit-and* [*simp*]:
‹*take-bit n (a AND b) = take-bit n a AND take-bit n b*›
⟨*proof*⟩

**lemma** *take-bit-or* [*simp*]:
‹*take-bit n (a OR b) = take-bit n a OR take-bit n b*›
⟨*proof*⟩

**lemma** *take-bit-xor* [*simp*]:
‹*take-bit n (a XOR b) = take-bit n a XOR take-bit n b*›
⟨*proof*⟩

**lemma** *push-bit-and* [*simp*]:
‹*push-bit n (a AND b) = push-bit n a AND push-bit n b*›
⟨*proof*⟩

**lemma** *push-bit-or* [*simp*]:
‹*push-bit n (a OR b) = push-bit n a OR push-bit n b*›
⟨*proof*⟩

**lemma** *push-bit-xor* [*simp*]:
‹*push-bit n (a XOR b) = push-bit n a XOR push-bit n b*›
⟨*proof*⟩

**lemma** *drop-bit-and* [*simp*]:
‹*drop-bit n (a AND b) = drop-bit n a AND drop-bit n b*›
⟨*proof*⟩

**lemma** *drop-bit-or* [*simp*]:
‹*drop-bit n (a OR b) = drop-bit n a OR drop-bit n b*›
⟨*proof*⟩

**lemma** *drop-bit-xor* [*simp*]:
‹*drop-bit n (a XOR b) = drop-bit n a XOR drop-bit n b*›
⟨*proof*⟩

**lemma** *take-bit-of-mask* [*simp*]:

‹*take-bit m* (*mask n*) = *mask* (*min m n*)›
⟨*proof*⟩

**lemma** *take-bit-eq-mask*:
‹*take-bit n a* = *a AND mask n*›
⟨*proof*⟩

**lemma** *or-eq-0-iff*:
‹*a OR b* = *0* ⟷ *a* = *0* ∧ *b* = *0*›
⟨*proof*⟩

**lemma** *bit-iff-and-drop-bit-eq-1*:
‹*bit a n* ⟷ *drop-bit n a AND 1* = *1*›
⟨*proof*⟩

**lemma** *bit-iff-and-push-bit-not-eq-0*:
‹*bit a n* ⟷ *a AND push-bit n 1* ≠ *0*›
⟨*proof*⟩

**lemma** *bit-set-bit-iff* [*bit-simps*]:
‹*bit* (*set-bit m a*) *n* ⟷ *bit a n* ∨ (*m* = *n* ∧ *possible-bit TYPE*(*'a*) *n*)›
⟨*proof*⟩

**lemma** *even-set-bit-iff*:
‹*even* (*set-bit m a*) ⟷ *even a* ∧ *m* ≠ *0*›
⟨*proof*⟩

**lemma** *bit-unset-bit-iff* [*bit-simps*]:
‹*bit* (*unset-bit m a*) *n* ⟷ *bit a n* ∧ *m* ≠ *n*›
⟨*proof*⟩

**lemma** *even-unset-bit-iff*:
‹*even* (*unset-bit m a*) ⟷ *even a* ∨ *m* = *0*›
⟨*proof*⟩

**lemma** *bit-flip-bit-iff* [*bit-simps*]:
‹*bit* (*flip-bit m a*) *n* ⟷ (*m* = *n* ⟷ ¬ *bit a n*) ∧ *possible-bit TYPE*(*'a*) *n*›
⟨*proof*⟩

**lemma** *even-flip-bit-iff*:
‹*even* (*flip-bit m a*) ⟷ ¬ (*even a* ⟷ *m* = *0*)›
⟨*proof*⟩

**lemma** *and-exp-eq-0-iff-not-bit*:
‹*a AND 2* ^ *n* = *0* ⟷ ¬ *bit a n*› (**is** ‹*?P* ⟷ *?Q*›)
⟨*proof*⟩

**lemma** *bit-sum-mult-2-cases*:
**assumes** *a*: ‹∀ *j*. ¬ *bit a* (*Suc j*)›

**shows** ‹*bit* (*a* + *2* ∗ *b*) *n* = (*if n* = *0 then odd a else bit* (*2* ∗ *b*) *n*)›
⟨*proof*⟩

**lemma** *set-bit-0*:
‹*set-bit 0 a* = *1* + *2* ∗ (*a div 2*)›
⟨*proof*⟩

**lemma** *set-bit-Suc*:
‹*set-bit* (*Suc n*) *a* = *a mod 2* + *2* ∗ *set-bit n* (*a div 2*)›
⟨*proof*⟩

**lemma** *unset-bit-0*:
‹*unset-bit 0 a* = *2* ∗ (*a div 2*)›
⟨*proof*⟩

**lemma** *unset-bit-Suc*:
‹*unset-bit* (*Suc n*) *a* = *a mod 2* + *2* ∗ *unset-bit n* (*a div 2*)›
⟨*proof*⟩

**lemma** *flip-bit-0*:
‹*flip-bit 0 a* = *of-bool* (*even a*) + *2* ∗ (*a div 2*)›
⟨*proof*⟩

**lemma** *flip-bit-Suc*:
‹*flip-bit* (*Suc n*) *a* = *a mod 2* + *2* ∗ *flip-bit n* (*a div 2*)›
⟨*proof*⟩

**lemma** *flip-bit-eq-if*:
‹*flip-bit n a* = (*if bit a n then unset-bit else set-bit*) *n a*›
⟨*proof*⟩

**lemma** *take-bit-set-bit-eq*:
‹*take-bit n* (*set-bit m a*) = (*if n* ≤ *m then take-bit n a else set-bit m* (*take-bit n a*))›
⟨*proof*⟩

**lemma** *take-bit-unset-bit-eq*:
‹*take-bit n* (*unset-bit m a*) = (*if n* ≤ *m then take-bit n a else unset-bit m* (*take-bit n a*))›
⟨*proof*⟩

**lemma** *take-bit-flip-bit-eq*:
‹*take-bit n* (*flip-bit m a*) = (*if n* ≤ *m then take-bit n a else flip-bit m* (*take-bit n a*))›
⟨*proof*⟩

**lemma** *push-bit-Suc-numeral* [*simp*]:
‹*push-bit* (*Suc n*) (*numeral k*) = *push-bit n* (*numeral* (*Num.Bit0 k*))›
⟨*proof*⟩

**lemma** *mask-eq-0-iff* [*simp*]:
‹$mask\ n = 0 \longleftrightarrow n = 0$›
⟨*proof*⟩

**lemma** *bit-horner-sum-bit-iff* [*bit-simps*]:
‹$bit\ (horner\text{-}sum\ of\text{-}bool\ 2\ bs)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n < length\ bs \wedge bs\ !\ n$›
⟨*proof*⟩

**lemma** *horner-sum-bit-eq-take-bit*:
‹$horner\text{-}sum\ of\text{-}bool\ 2\ (map\ (bit\ a)\ [0..<n]) = take\text{-}bit\ n\ a$›
⟨*proof*⟩

**lemma** *take-bit-horner-sum-bit-eq*:
‹$take\text{-}bit\ n\ (horner\text{-}sum\ of\text{-}bool\ 2\ bs) = horner\text{-}sum\ of\text{-}bool\ 2\ (take\ n\ bs)$›
⟨*proof*⟩

**lemma** *take-bit-sum*:
‹$take\text{-}bit\ n\ a = (\sum k = 0..<n.\ push\text{-}bit\ k\ (of\text{-}bool\ (bit\ a\ k)))$›
⟨*proof*⟩

**lemma** *set-bit-eq*:
‹$set\text{-}bit\ n\ a = a + of\text{-}bool\ (\neg\ bit\ a\ n) * 2\ \widehat{}\ n$›
⟨*proof*⟩

**end**

**class** *ring-bit-operations* = *semiring-bit-operations* + *ring-parity* +
**fixes** *not* :: ‹$'a \Rightarrow 'a$› (‹*NOT*›)
**assumes** *not-eq-complement*: ‹$NOT\ a = -\ a - 1$›
**begin**

For the sake of code generation *NOT* is specified as definitional class operation. Note that *NOT* has no sensible definition for unlimited but only positive bit strings (type *nat*).

**lemma** *bits-minus-1-mod-2-eq* [*simp*]:
‹$(-\ 1)\ mod\ 2 = 1$›
⟨*proof*⟩

**lemma** *minus-eq-not-plus-1*:
‹$-\ a = NOT\ a + 1$›
⟨*proof*⟩

**lemma** *minus-eq-not-minus-1*:
‹$-\ a = NOT\ (a - 1)$›
⟨*proof*⟩

**lemma** *not-rec*:

‹$NOT\ a = of\text{-}bool\ (even\ a) + 2 * NOT\ (a\ div\ 2)$›
⟨proof⟩

**lemma** *decr-eq-not-minus*:
‹$a - 1 = NOT\ (- a)$›
⟨proof⟩

**lemma** *even-not-iff* [*simp*]:
‹$even\ (NOT\ a) \longleftrightarrow odd\ a$›
⟨proof⟩

**lemma** *bit-not-iff* [*bit-simps*]:
‹$bit\ (NOT\ a)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge \neg\ bit\ a\ n$›
⟨proof⟩

**lemma** *bit-not-exp-iff* [*bit-simps*]:
‹$bit\ (NOT\ (2\ \widehat{}\ m))\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n \neq m$›
⟨proof⟩

**lemma** *bit-minus-iff* [*bit-simps*]:
‹$bit\ (- a)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge \neg\ bit\ (a - 1)\ n$›
⟨proof⟩

**lemma** *bit-minus-1-iff* [*simp*]:
‹$bit\ (- 1)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n$›
⟨proof⟩

**lemma** *bit-minus-exp-iff* [*bit-simps*]:
‹$bit\ (- (2\ \widehat{}\ m))\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n \geq m$›
⟨proof⟩

**lemma** *bit-minus-2-iff* [*simp*]:
‹$bit\ (- 2)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge n > 0$›
⟨proof⟩

**lemma** *bit-decr-iff*:
‹$bit\ (a - 1)\ n \longleftrightarrow possible\text{-}bit\ TYPE('a)\ n \wedge \neg\ bit\ (- a)\ n$›
⟨proof⟩

**lemma** *bit-not-iff-eq*:
‹$bit\ (NOT\ a)\ n \longleftrightarrow 2\ \widehat{}\ n \neq 0 \wedge \neg\ bit\ a\ n$›
⟨proof⟩

**lemma** *not-one-eq* [*simp*]:
‹$NOT\ 1 = - 2$›
⟨proof⟩

**sublocale** *and*: *semilattice-neutr* ‹$(AND)$› ‹$- 1$›
⟨proof⟩

**sublocale** *bit*: *abstract-boolean-algebra* ‹(*AND*)› ‹(*OR*)› *NOT* *0* ‹− *1*›
⟨*proof*⟩

**sublocale** *bit*: *abstract-boolean-algebra-sym-diff* ‹(*AND*)› ‹(*OR*)› *NOT* *0* ‹− *1*› ‹(*XOR*)›
⟨*proof*⟩

**lemma** *and-eq-not-not-or*:
‹*a AND b* = *NOT* (*NOT a OR NOT b*)›
⟨*proof*⟩

**lemma** *or-eq-not-not-and*:
‹*a OR b* = *NOT* (*NOT a AND NOT b*)›
⟨*proof*⟩

**lemma** *not-add-distrib*:
‹*NOT* (*a* + *b*) = *NOT a* − *b*›
⟨*proof*⟩

**lemma** *not-diff-distrib*:
‹*NOT* (*a* − *b*) = *NOT a* + *b*›
⟨*proof*⟩

**lemma** *and-eq-minus-1-iff*:
‹*a AND b* = − *1* ⟷ *a* = − *1* ∧ *b* = − *1*›
⟨*proof*⟩

**lemma** *disjunctive-and-not-eq-xor*:
‹*a AND NOT b* = *a XOR b*› **if** ‹*NOT a AND b* = *0*›
⟨*proof*⟩

**lemma** *disjunctive-diff-eq-and-not*:
‹*a* − *b* = *a AND NOT b*› **if** ‹*NOT a AND b* = *0*›
⟨*proof*⟩

**lemma** *disjunctive-diff-eq-xor*:
‹*a AND NOT b* = *a XOR b*› **if** ‹*NOT a AND b* = *0*›
⟨*proof*⟩

**lemma** *push-bit-minus*:
‹*push-bit n* (− *a*) = − *push-bit n a*›
⟨*proof*⟩

**lemma** *take-bit-not-take-bit*:
‹*take-bit n* (*NOT* (*take-bit n a*)) = *take-bit n* (*NOT a*)›
⟨*proof*⟩

**lemma** *take-bit-not-iff*:

‹*take-bit n* (*NOT a*) = *take-bit n* (*NOT b*) ⟷ *take-bit n a* = *take-bit n b*›
⟨*proof*⟩

**lemma** *take-bit-not-eq-mask-diff*:
‹*take-bit n* (*NOT a*) = *mask n* − *take-bit n a*›
⟨*proof*⟩

**lemma** *mask-eq-take-bit-minus-one*:
‹*mask n* = *take-bit n* (− *1*)›
⟨*proof*⟩

**lemma** *take-bit-minus-one-eq-mask* [*simp*]:
‹*take-bit n* (− *1*) = *mask n*›
⟨*proof*⟩

**lemma** *minus-exp-eq-not-mask*:
‹− (*2* ^ *n*) = *NOT* (*mask n*)›
⟨*proof*⟩

**lemma** *push-bit-minus-one-eq-not-mask* [*simp*]:
‹*push-bit n* (− *1*) = *NOT* (*mask n*)›
⟨*proof*⟩

**lemma** *take-bit-not-mask-eq-0*:
‹*take-bit m* (*NOT* (*mask n*)) = *0*› **if** ‹*n* ≥ *m*›
⟨*proof*⟩

**lemma** *mask-eq-minus-one-if-not-possible-bit*:
‹*mask n* = − *1*› **if** ‹¬ *possible-bit TYPE*(*'a*) *n*›
⟨*proof*⟩

**lemma** *unset-bit-eq-and-not*:
‹*unset-bit n a* = *a AND NOT* (*push-bit n 1*)›
⟨*proof*⟩

**lemma** *push-bit-Suc-minus-numeral* [*simp*]:
‹*push-bit* (*Suc n*) (− *numeral k*) = *push-bit n* (− *numeral* (*Num.Bit0 k*))›
⟨*proof*⟩

**lemma** *push-bit-minus-numeral* [*simp*]:
‹*push-bit* (*numeral l*) (− *numeral k*) = *push-bit* (*pred-numeral l*) (− *numeral* (*Num.Bit0 k*))›
⟨*proof*⟩

**lemma** *take-bit-Suc-minus-1-eq*:
‹*take-bit* (*Suc n*) (− *1*) = *2* ^ *Suc n* − *1*›
⟨*proof*⟩

**lemma** *take-bit-numeral-minus-1-eq*:

‹$take\text{-}bit\ (numeral\ k)\ (-\ 1) = 2\ \hat{}\ numeral\ k - 1$›
⟨*proof*⟩

**lemma** *push-bit-mask-eq*:
‹$push\text{-}bit\ m\ (mask\ n) = mask\ (n + m)\ AND\ NOT\ (mask\ m)$›
⟨*proof*⟩

**lemma** *slice-eq-mask*:
‹$push\text{-}bit\ n\ (take\text{-}bit\ m\ (drop\text{-}bit\ n\ a)) = a\ AND\ mask\ (m + n)\ AND\ NOT\ (mask\ n)$›
⟨*proof*⟩

**lemma** *push-bit-numeral-minus-1* [*simp*]:
‹$push\text{-}bit\ (numeral\ n)\ (-\ 1) = -\ (2\ \hat{}\ numeral\ n)$›
⟨*proof*⟩

**lemma** *unset-bit-eq*:
‹$unset\text{-}bit\ n\ a = a - of\text{-}bool\ (bit\ a\ n) * 2\ \hat{}\ n$›
⟨*proof*⟩

**end**

## 68.3 Common algebraic structure

**class** *linordered-euclidean-semiring-bit-operations* =
*linordered-euclidean-semiring* + *semiring-bit-operations*
**begin**

**lemma** *possible-bit* [*simp*]:
‹$possible\text{-}bit\ TYPE('a)\ n$›
⟨*proof*⟩

**lemma** *take-bit-of-exp* [*simp*]:
‹$take\text{-}bit\ m\ (2\ \hat{}\ n) = of\text{-}bool\ (n < m) * 2\ \hat{}\ n$›
⟨*proof*⟩

**lemma** *take-bit-of-2* [*simp*]:
‹$take\text{-}bit\ n\ 2 = of\text{-}bool\ (2 \leq n) * 2$›
⟨*proof*⟩

**lemma** *push-bit-eq-0-iff* [*simp*]:
‹$push\text{-}bit\ n\ a = 0 \longleftrightarrow a = 0$›
⟨*proof*⟩

**lemma** *take-bit-add*:
‹$take\text{-}bit\ n\ (take\text{-}bit\ n\ a + take\text{-}bit\ n\ b) = take\text{-}bit\ n\ (a + b)$›
⟨*proof*⟩

**lemma** *take-bit-of-1-eq-0-iff* [*simp*]:

‹*take-bit n 1 = 0* ⟷ *n = 0*›
⟨*proof*⟩

**lemma** *drop-bit-Suc-bit0* [*simp*]:
‹*drop-bit* (*Suc n*) (*numeral* (*Num.Bit0 k*)) = *drop-bit n* (*numeral k*)›
⟨*proof*⟩

**lemma** *drop-bit-Suc-bit1* [*simp*]:
‹*drop-bit* (*Suc n*) (*numeral* (*Num.Bit1 k*)) = *drop-bit n* (*numeral k*)›
⟨*proof*⟩

**lemma** *drop-bit-numeral-bit0* [*simp*]:
‹*drop-bit* (*numeral l*) (*numeral* (*Num.Bit0 k*)) = *drop-bit* (*pred-numeral l*) (*numeral k*)›
⟨*proof*⟩

**lemma** *drop-bit-numeral-bit1* [*simp*]:
‹*drop-bit* (*numeral l*) (*numeral* (*Num.Bit1 k*)) = *drop-bit* (*pred-numeral l*) (*numeral k*)›
⟨*proof*⟩

**lemma** *take-bit-Suc-1* [*simp*]:
‹*take-bit* (*Suc n*) *1 = 1*›
⟨*proof*⟩

**lemma** *take-bit-Suc-bit0*:
‹*take-bit* (*Suc n*) (*numeral* (*Num.Bit0 k*)) = *take-bit n* (*numeral k*) * *2*›
⟨*proof*⟩

**lemma** *take-bit-Suc-bit1*:
‹*take-bit* (*Suc n*) (*numeral* (*Num.Bit1 k*)) = *take-bit n* (*numeral k*) * *2* + *1*›
⟨*proof*⟩

**lemma** *take-bit-numeral-1* [*simp*]:
‹*take-bit* (*numeral l*) *1 = 1*›
⟨*proof*⟩

**lemma** *take-bit-numeral-bit0*:
‹*take-bit* (*numeral l*) (*numeral* (*Num.Bit0 k*)) = *take-bit* (*pred-numeral l*) (*numeral k*) * *2*›
⟨*proof*⟩

**lemma** *take-bit-numeral-bit1*:
‹*take-bit* (*numeral l*) (*numeral* (*Num.Bit1 k*)) = *take-bit* (*pred-numeral l*) (*numeral k*) * *2* + *1*›
⟨*proof*⟩

**lemma** *bit-of-nat-iff-bit* [*bit-simps*]:
‹*bit* (*of-nat m*) *n* ⟷ *bit m n*›

⟨*proof*⟩

**lemma** *drop-bit-mask-eq*:
‹$drop\text{-}bit\ m\ (mask\ n) = mask\ (n - m)$›
⟨*proof*⟩

**lemma** *bit-push-bit-iff′*:
‹$bit\ (push\text{-}bit\ m\ a)\ n \longleftrightarrow m \leq n \wedge bit\ a\ (n - m)$›
⟨*proof*⟩

**lemma** *mask-half*:
‹$mask\ n\ div\ 2 = mask\ (n - 1)$›
⟨*proof*⟩

**lemma** *take-bit-Suc-from-most*:
‹$take\text{-}bit\ (Suc\ n)\ a = 2\ \widehat{}\ n * of\text{-}bool\ (bit\ a\ n) + take\text{-}bit\ n\ a$›
⟨*proof*⟩

**lemma** *take-bit-nonnegative* [*simp*]:
‹$0 \leq take\text{-}bit\ n\ a$›
⟨*proof*⟩

**lemma** *not-take-bit-negative* [*simp*]:
‹$\neg\ take\text{-}bit\ n\ a < 0$›
⟨*proof*⟩

**lemma** *bit-imp-take-bit-positive*:
‹$0 < take\text{-}bit\ m\ a$› **if** ‹$n < m$› **and** ‹$bit\ a\ n$›
⟨*proof*⟩

**lemma** *take-bit-mult*:
‹$take\text{-}bit\ n\ (take\text{-}bit\ n\ a * take\text{-}bit\ n\ b) = take\text{-}bit\ n\ (a * b)$›
⟨*proof*⟩

**lemma** *drop-bit-push-bit*:
‹$drop\text{-}bit\ m\ (push\text{-}bit\ n\ a) = drop\text{-}bit\ (m - n)\ (push\text{-}bit\ (n - m)\ a)$›
⟨*proof*⟩

**end**

## 68.4 Instance *int*

**locale** *fold2-bit-int* =
**fixes** $f$ :: ‹$bool \Rightarrow bool \Rightarrow bool$›
**begin**

**context**
**begin**

**function** $F$ :: ‹$int \Rightarrow int \Rightarrow int$›
  **where** ‹$F\ k\ l = (if\ k \in \{0, -\ 1\} \wedge l \in \{0, -\ 1\}$
    $then\ -\ of\text{-}bool\ (f\ (odd\ k)\ (odd\ l))$
    $else\ of\text{-}bool\ (f\ (odd\ k)\ (odd\ l)) + 2 * (F\ (k\ div\ 2)\ (l\ div\ 2)))$›
  ⟨*proof*⟩ **termination** ⟨*proof*⟩

**declare** *F.simps* [*simp del*]

**lemma** *rec*:
  ‹$F\ k\ l = of\text{-}bool\ (f\ (odd\ k)\ (odd\ l)) + 2 * (F\ (k\ div\ 2)\ (l\ div\ 2))$›
    **for** $k\ l$ :: *int*
⟨*proof*⟩

**lemma** *bit-iff*:
  ‹$bit\ (F\ k\ l)\ n \longleftrightarrow f\ (bit\ k\ n)\ (bit\ l\ n)$› **for** $k\ l$ :: *int*
⟨*proof*⟩

**end**

**end**

**instantiation** *int* :: *ring-bit-operations*
**begin**

**definition** *not-int* :: ‹$int \Rightarrow int$›
  **where** ‹$not\text{-}int\ k = -\ k - 1$›

**global-interpretation** *and-int*: *fold2-bit-int* ‹$(\wedge)$›
  **defines** $and\text{-}int = and\text{-}int.F$ ⟨*proof*⟩

**global-interpretation** *or-int*: *fold2-bit-int* ‹$(\vee)$›
  **defines** $or\text{-}int = or\text{-}int.F$ ⟨*proof*⟩

**global-interpretation** *xor-int*: *fold2-bit-int* ‹$(\neq)$›
  **defines** $xor\text{-}int = xor\text{-}int.F$ ⟨*proof*⟩

**definition** *mask-int* :: ‹$nat \Rightarrow int$›
  **where** ‹$mask\ n = (2 :: int)\ \hat{}\ n - 1$›

**definition** *push-bit-int* :: ‹$nat \Rightarrow int \Rightarrow int$›
  **where** ‹$push\text{-}bit\text{-}int\ n\ k = k * 2\ \hat{}\ n$›

**definition** *drop-bit-int* :: ‹$nat \Rightarrow int \Rightarrow int$›
  **where** ‹$drop\text{-}bit\text{-}int\ n\ k = k\ div\ 2\ \hat{}\ n$›

**definition** *take-bit-int* :: ‹$nat \Rightarrow int \Rightarrow int$›
  **where** ‹$take\text{-}bit\text{-}int\ n\ k = k\ mod\ 2\ \hat{}\ n$›

**definition** *set-bit-int* :: ‹$nat \Rightarrow int \Rightarrow int$›

**where** ‹*set-bit n k = k OR push-bit n 1*› **for** *k* :: *int*

**definition** *unset-bit-int* :: ‹*nat* ⇒ *int* ⇒ *int*›
**where** ‹*unset-bit n k = k AND NOT* (*push-bit n 1*)› **for** *k* :: *int*

**definition** *flip-bit-int* :: ‹*nat* ⇒ *int* ⇒ *int*›
**where** ‹*flip-bit n k = k XOR push-bit n 1*› **for** *k* :: *int*

**lemma** *not-int-div-2*:
‹*NOT k div 2 = NOT* (*k div 2*)› **for** *k* :: *int*
⟨*proof*⟩

**lemma** *bit-not-int-iff*:
‹*bit* (*NOT k*) *n* ⟷ ¬ *bit k n*› **for** *k* :: *int*
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instance** *int* :: *linordered-euclidean-semiring-bit-operations* ⟨*proof*⟩

**context** *ring-bit-operations*
**begin**

**lemma** *even-of-int-iff*:
‹*even* (*of-int k*) ⟷ *even k*›
⟨*proof*⟩

**lemma** *bit-of-int-iff* [*bit-simps*]:
‹*bit* (*of-int k*) *n* ⟷ *possible-bit TYPE*('*a*) *n* ∧ *bit k n*›
⟨*proof*⟩

**lemma** *push-bit-of-int*:
‹*push-bit n* (*of-int k*) = *of-int* (*push-bit n k*)›
⟨*proof*⟩

**lemma** *of-int-push-bit*:
‹*of-int* (*push-bit n k*) = *push-bit n* (*of-int k*)›
⟨*proof*⟩

**lemma** *take-bit-of-int*:
‹*take-bit n* (*of-int k*) = *of-int* (*take-bit n k*)›
⟨*proof*⟩

**lemma** *of-int-take-bit*:
‹*of-int* (*take-bit n k*) = *take-bit n* (*of-int k*)›
⟨*proof*⟩

**lemma** *of-int-not-eq*:
‹*of-int* (*NOT k*) = *NOT* (*of-int k*)›
⟨*proof*⟩

**lemma** *of-int-not-numeral*:
‹*of-int* (*NOT* (*numeral k*)) = *NOT* (*numeral k*)›
⟨*proof*⟩

**lemma** *of-int-and-eq*:
‹*of-int* (*k AND l*) = *of-int k AND of-int l*›
⟨*proof*⟩

**lemma** *of-int-or-eq*:
‹*of-int* (*k OR l*) = *of-int k OR of-int l*›
⟨*proof*⟩

**lemma** *of-int-xor-eq*:
‹*of-int* (*k XOR l*) = *of-int k XOR of-int l*›
⟨*proof*⟩

**lemma** *of-int-mask-eq*:
‹*of-int* (*mask n*) = *mask n*›
⟨*proof*⟩

**end**

**lemma** *take-bit-int-less-exp* [*simp*]:
‹*take-bit n k* < *2* ^ *n*› **for** *k* :: *int*
⟨*proof*⟩

**lemma** *take-bit-int-eq-self-iff*:
‹*take-bit n k* = *k* ⟷ *0* ≤ *k* ∧ *k* < *2* ^ *n*› (**is** ‹*?P* ⟷ *?Q*›)
**for** *k* :: *int*
⟨*proof*⟩

**lemma** *take-bit-int-eq-self*:
‹*take-bit n k* = *k*› **if** ‹*0* ≤ *k*› ‹*k* < *2* ^ *n*› **for** *k* :: *int*
⟨*proof*⟩

**lemma** *mask-nonnegative-int* [*simp*]:
‹*mask n* ≥ (*0*::*int*)›
⟨*proof*⟩

**lemma** *not-mask-negative-int* [*simp*]:
‹¬ *mask n* < (*0*::*int*)›
⟨*proof*⟩

**lemma** *not-nonnegative-int-iff* [*simp*]:
‹*NOT k* ≥ *0* ⟷ *k* < *0*› **for** *k* :: *int*

⟨*proof*⟩

**lemma** *not-negative-int-iff* [*simp*]:
‹$NOT\ k < 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *and-nonnegative-int-iff* [*simp*]:
‹$k\ AND\ l \geq 0 \longleftrightarrow k \geq 0 \lor l \geq 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *and-negative-int-iff* [*simp*]:
‹$k\ AND\ l < 0 \longleftrightarrow k < 0 \land l < 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *and-less-eq*:
‹$k\ AND\ l \leq k$› **if** ‹$l < 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *or-nonnegative-int-iff* [*simp*]:
‹$k\ OR\ l \geq 0 \longleftrightarrow k \geq 0 \land l \geq 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *or-negative-int-iff* [*simp*]:
‹$k\ OR\ l < 0 \longleftrightarrow k < 0 \lor l < 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *or-greater-eq*:
‹$k\ OR\ l \geq k$› **if** ‹$l \geq 0$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *xor-nonnegative-int-iff* [*simp*]:
‹$k\ XOR\ l \geq 0 \longleftrightarrow (k \geq 0 \longleftrightarrow l \geq 0)$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *xor-negative-int-iff* [*simp*]:
‹$k\ XOR\ l < 0 \longleftrightarrow (k < 0) \neq (l < 0)$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *OR-upper*:
‹$x\ OR\ y < 2\ \hat{}\ n$› **if** ‹$0 \leq x$› ‹$x < 2\ \hat{}\ n$› ‹$y < 2\ \hat{}\ n$› **for** $x\ y :: int$
⟨*proof*⟩

**lemma** *XOR-upper*:
‹$x\ XOR\ y < 2\ \hat{}\ n$› **if** ‹$0 \leq x$› ‹$x < 2\ \hat{}\ n$› ‹$y < 2\ \hat{}\ n$› **for** $x\ y :: int$
⟨*proof*⟩

**lemma** *AND-lower* [*simp*]:
‹$0 \leq x\ AND\ y$› **if** ‹$0 \leq x$› **for** $x\ y :: int$
⟨*proof*⟩

**lemma** *OR-lower* [*simp*]:
‹$0 \leq x$ *OR* $y$› **if** ‹$0 \leq x$› ‹$0 \leq y$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *XOR-lower* [*simp*]:
‹$0 \leq x$ *XOR* $y$› **if** ‹$0 \leq x$› ‹$0 \leq y$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper1* [*simp*]:
‹$x$ *AND* $y \leq x$› **if** ‹$0 \leq x$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper1′* [*simp*]:
‹$y$ *AND* $x \leq z$› **if** ‹$0 \leq y$› ‹$y \leq z$› **for** $x$ $y$ $z$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper1″* [*simp*]:
‹$y$ *AND* $x < z$› **if** ‹$0 \leq y$› ‹$y < z$› **for** $x$ $y$ $z$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper2* [*simp*]:
‹$x$ *AND* $y \leq y$› **if** ‹$0 \leq y$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper2′* [*simp*]:
‹$x$ *AND* $y \leq z$› **if** ‹$0 \leq y$› ‹$y \leq z$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *AND-upper2″* [*simp*]:
‹$x$ *AND* $y < z$› **if** ‹$0 \leq y$› ‹$y < z$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *plus-and-or*:
‹($x$ *AND* $y$) + ($x$ *OR* $y$) = $x + y$› **for** $x$ $y$ :: *int*
⟨*proof*⟩

**lemma** *push-bit-minus-one*:
‹*push-bit* $n$ ($-$ *1* :: *int*) = $-$ (*2* ^ $n$)›
⟨*proof*⟩

**lemma** *minus-1-div-exp-eq-int*:
‹$-$ *1* *div* (*2* :: *int*) ^ $n$ = $-$ *1*›
⟨*proof*⟩

**lemma** *drop-bit-minus-one* [*simp*]:
‹*drop-bit* $n$ ($-$ *1* :: *int*) = $-$ *1*›
⟨*proof*⟩

**lemma** *take-bit-minus*:
‹$take\text{-}bit\ n\ (-\ take\text{-}bit\ n\ k) = take\text{-}bit\ n\ (-\ k)$›
**for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-diff*:
‹$take\text{-}bit\ n\ (take\text{-}bit\ n\ k - take\text{-}bit\ n\ l) = take\text{-}bit\ n\ (k - l)$›
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** (**in** *ring-1*) *of-nat-nat-take-bit-eq* [*simp*]:
‹$of\text{-}nat\ (nat\ (take\text{-}bit\ n\ k)) = of\text{-}int\ (take\text{-}bit\ n\ k)$›
⟨*proof*⟩

**lemma** *take-bit-minus-small-eq*:
‹$take\text{-}bit\ n\ (-\ k) = 2\ \hat{}\ n - k$› **if** ‹$0 < k$› ‹$k \leq 2\ \hat{}\ n$› **for** $k :: int$
⟨*proof*⟩

**lemma** *push-bit-nonnegative-int-iff* [*simp*]:
‹$push\text{-}bit\ n\ k \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *push-bit-negative-int-iff* [*simp*]:
‹$push\text{-}bit\ n\ k < 0 \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *drop-bit-nonnegative-int-iff* [*simp*]:
‹$drop\text{-}bit\ n\ k \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *drop-bit-negative-int-iff* [*simp*]:
‹$drop\text{-}bit\ n\ k < 0 \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *set-bit-nonnegative-int-iff* [*simp*]:
‹$set\text{-}bit\ n\ k \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *set-bit-negative-int-iff* [*simp*]:
‹$set\text{-}bit\ n\ k < 0 \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *unset-bit-nonnegative-int-iff* [*simp*]:
‹$unset\text{-}bit\ n\ k \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *unset-bit-negative-int-iff* [*simp*]:
‹$unset\text{-}bit\ n\ k < 0 \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *flip-bit-nonnegative-int-iff* [*simp*]:
‹$flip\text{-}bit\ n\ k \geq 0 \longleftrightarrow k \geq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *flip-bit-negative-int-iff* [*simp*]:
‹$flip\text{-}bit\ n\ k < 0 \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *set-bit-greater-eq*:
‹$set\text{-}bit\ n\ k \geq k$› **for** $k :: int$
⟨*proof*⟩

**lemma** *unset-bit-less-eq*:
‹$unset\text{-}bit\ n\ k \leq k$› **for** $k :: int$
⟨*proof*⟩

**lemma** *and-int-unfold*:
‹$k\ AND\ l = (if\ k = 0 \vee l = 0\ then\ 0\ else\ if\ k = -\ 1\ then\ l\ else\ if\ l = -\ 1\ then\ k$
$else\ (k\ mod\ 2) * (l\ mod\ 2) + 2 * ((k\ div\ 2)\ AND\ (l\ div\ 2)))$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *or-int-unfold*:
‹$k\ OR\ l = (if\ k = -\ 1 \vee l = -\ 1\ then\ -\ 1\ else\ if\ k = 0\ then\ l\ else\ if\ l = 0\ then$
$k$
$else\ max\ (k\ mod\ 2)\ (l\ mod\ 2) + 2 * ((k\ div\ 2)\ OR\ (l\ div\ 2)))$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *xor-int-unfold*:
‹$k\ XOR\ l = (if\ k = -\ 1\ then\ NOT\ l\ else\ if\ l = -\ 1\ then\ NOT\ k\ else\ if\ k = 0$
$then\ l\ else\ if\ l = 0\ then\ k$
$else\ |k\ mod\ 2 - l\ mod\ 2| + 2 * ((k\ div\ 2)\ XOR\ (l\ div\ 2)))$› **for** $k\ l :: int$
⟨*proof*⟩

**lemma** *bit-minus-int-iff*:
‹$bit\ (-\ k)\ n \longleftrightarrow bit\ (NOT\ (k - 1))\ n$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-incr-eq*:
‹$take\text{-}bit\ n\ (k + 1) = 1 + take\text{-}bit\ n\ k$› **if** ‹$take\text{-}bit\ n\ k \neq 2\ \hat{}\ n - 1$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-decr-eq*:
‹$take\text{-}bit\ n\ (k - 1) = take\text{-}bit\ n\ k - 1$› **if** ‹$take\text{-}bit\ n\ k \neq 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-greater-eq*:
‹$k + 2\ \hat{}\ n \leq take\text{-}bit\ n\ k$› **if** ‹$k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-less-eq*:
‹$take\text{-}bit\ n\ k \leq k - 2 \ \hat{}\ n$› **if** ‹$2 \ \hat{}\ n \leq k$› **and** ‹$n > 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-less-eq-self-iff*:
‹$take\text{-}bit\ n\ k \leq k \longleftrightarrow 0 \leq k$› (**is** ‹$?P \longleftrightarrow ?Q$›) **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-less-self-iff*:
‹$take\text{-}bit\ n\ k < k \longleftrightarrow 2 \ \hat{}\ n \leq k$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-less-eq-mask*:
‹$take\text{-}bit\ n\ k \leq mask\ n$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-greater-self-iff*:
‹$k < take\text{-}bit\ n\ k \longleftrightarrow k < 0$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-int-greater-eq-self-iff*:
‹$k \leq take\text{-}bit\ n\ k \longleftrightarrow k < 2 \ \hat{}\ n$› **for** $k :: int$
⟨*proof*⟩

**lemma** *take-bit-tightened-less-eq-int*:
‹$take\text{-}bit\ m\ k \leq take\text{-}bit\ n\ k$› **if** ‹$m \leq n$› **for** $k :: int$
⟨*proof*⟩

**lemma** *not-exp-less-eq-0-int* [*simp*]:
‹$\neg\ 2 \ \hat{}\ n \leq (0::int)$›
⟨*proof*⟩

**lemma** *int-bit-bound*:
**fixes** $k :: int$
**obtains** $n$ **where** ‹$\bigwedge m.\ n \leq m \Longrightarrow bit\ k\ m \longleftrightarrow bit\ k\ n$›
**and** ‹$n > 0 \Longrightarrow bit\ k\ (n - 1) \neq bit\ k\ n$›
⟨*proof*⟩

## 68.5 Instance *nat*

**instantiation** $nat :: semiring\text{-}bit\text{-}operations$
**begin**

**definition** $and\text{-}nat :: \langle nat \Rightarrow nat \Rightarrow nat\rangle$
**where** ‹$m\ AND\ n = nat\ (int\ m\ AND\ int\ n)$› **for** $m\ n :: nat$

**definition** $or\text{-}nat :: \langle nat \Rightarrow nat \Rightarrow nat\rangle$
**where** ‹$m\ OR\ n = nat\ (int\ m\ OR\ int\ n)$› **for** $m\ n :: nat$

**definition** *xor-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹$m$ *XOR* $n$ = *nat* (*int* $m$ *XOR* *int* $n$)› **for** $m$ $n$ :: *nat*

**definition** *mask-nat* :: ‹*nat* ⇒ *nat*›
**where** ‹*mask* $n$ = (*2* :: *nat*) ^ $n$ − *1*›

**definition** *push-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*push-bit-nat* $n$ $m$ = $m$ ∗ *2* ^ $n$›

**definition** *drop-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*drop-bit-nat* $n$ $m$ = $m$ *div* *2* ^ $n$›

**definition** *take-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*take-bit-nat* $n$ $m$ = $m$ *mod* *2* ^ $n$›

**definition** *set-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*set-bit* $m$ $n$ = $n$ *OR* *push-bit* $m$ *1*› **for** $m$ $n$ :: *nat*

**definition** *unset-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*unset-bit* $m$ $n$ = ($n$ *OR* *push-bit* $m$ *1*) *XOR* *push-bit* $m$ *1*› **for** $m$ $n$ :: *nat*

**definition** *flip-bit-nat* :: ‹*nat* ⇒ *nat* ⇒ *nat*›
**where** ‹*flip-bit* $m$ $n$ = $n$ *XOR* *push-bit* $m$ *1*› **for** $m$ $n$ :: *nat*

**instance** ⟨*proof*⟩

**end**

**instance** *nat* :: *linordered-euclidean-semiring-bit-operations* ⟨*proof*⟩

**context** *semiring-bit-operations*
**begin**

**lemma** *push-bit-of-nat*:
‹*push-bit* $n$ (*of-nat* $m$) = *of-nat* (*push-bit* $n$ $m$)›
⟨*proof*⟩

**lemma** *of-nat-push-bit*:
‹*of-nat* (*push-bit* $m$ $n$) = *push-bit* $m$ (*of-nat* $n$)›
⟨*proof*⟩

**lemma** *take-bit-of-nat*:
‹*take-bit* $n$ (*of-nat* $m$) = *of-nat* (*take-bit* $n$ $m$)›
⟨*proof*⟩

**lemma** *of-nat-take-bit*:
‹*of-nat* (*take-bit* $n$ $m$) = *take-bit* $n$ (*of-nat* $m$)›
⟨*proof*⟩

**lemma** *of-nat-and-eq*:
‹*of-nat* (*m AND n*) = *of-nat m AND of-nat n*›
⟨*proof*⟩

**lemma** *of-nat-or-eq*:
‹*of-nat* (*m OR n*) = *of-nat m OR of-nat n*›
⟨*proof*⟩

**lemma** *of-nat-xor-eq*:
‹*of-nat* (*m XOR n*) = *of-nat m XOR of-nat n*›
⟨*proof*⟩

**lemma** *of-nat-mask-eq*:
‹*of-nat* (*mask n*) = *mask n*›
⟨*proof*⟩

**lemma** *of-nat-set-bit-eq*:
‹*of-nat* (*set-bit n m*) = *set-bit n* (*of-nat m*)›
⟨*proof*⟩

**lemma** *of-nat-unset-bit-eq*:
‹*of-nat* (*unset-bit n m*) = *unset-bit n* (*of-nat m*)›
⟨*proof*⟩

**lemma** *of-nat-flip-bit-eq*:
‹*of-nat* (*flip-bit n m*) = *flip-bit n* (*of-nat m*)›
⟨*proof*⟩

**end**

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**lemma** *drop-bit-of-nat*:
*drop-bit n* (*of-nat m*) = *of-nat* (*drop-bit n m*)
⟨*proof*⟩

**lemma** *of-nat-drop-bit*:
‹*of-nat* (*drop-bit m n*) = *drop-bit m* (*of-nat n*)›
⟨*proof*⟩

**end**

**lemma** *take-bit-nat-less-exp* [*simp*]:
‹*take-bit n m* < *2* ^ *n*› **for** *n m* :: *nat*
⟨*proof*⟩

**lemma** *take-bit-nat-eq-self-iff*:

‹*take-bit n m* = *m* ⟷ *m* < *2* ^ *n*› (**is** ‹*?P* ⟷ *?Q*›) **for** *n m* :: *nat*
⟨*proof*⟩

**lemma** *take-bit-nat-eq-self*:
‹*take-bit n m* = *m*› **if** ‹*m* < *2* ^ *n*› **for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *take-bit-nat-less-eq-self* [*simp*]:
‹*take-bit n m* ≤ *m*› **for** *n m* :: *nat*
⟨*proof*⟩

**lemma** *take-bit-nat-less-self-iff*:
‹*take-bit n m* < *m* ⟷ *2* ^ *n* ≤ *m*› (**is** ‹*?P* ⟷ *?Q*›) **for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *Suc-0-and-eq* [*simp*]:
‹*Suc 0 AND n* = *n mod 2*›
⟨*proof*⟩

**lemma** *and-Suc-0-eq* [*simp*]:
‹*n AND Suc 0* = *n mod 2*›
⟨*proof*⟩

**lemma** *Suc-0-or-eq*:
‹*Suc 0 OR n* = *n* + *of-bool* (*even n*)›
⟨*proof*⟩

**lemma** *or-Suc-0-eq*:
‹*n OR Suc 0* = *n* + *of-bool* (*even n*)›
⟨*proof*⟩

**lemma** *Suc-0-xor-eq*:
‹*Suc 0 XOR n* = *n* + *of-bool* (*even n*) − *of-bool* (*odd n*)›
⟨*proof*⟩

**lemma** *xor-Suc-0-eq*:
‹*n XOR Suc 0* = *n* + *of-bool* (*even n*) − *of-bool* (*odd n*)›
⟨*proof*⟩

**lemma** *and-nat-unfold* [*code*]:
‹*m AND n* = (*if m* = *0* ∨ *n* = *0 then 0 else* (*m mod 2*) ∗ (*n mod 2*) + *2* ∗ ((*m div 2*) *AND* (*n div 2*)))›
**for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *or-nat-unfold* [*code*]:
‹*m OR n* = (*if m* = *0 then n else if n* = *0 then m*
*else max* (*m mod 2*) (*n mod 2*) + *2* ∗ ((*m div 2*) *OR* (*n div 2*)))› **for** *m n* :: *nat*

⟨*proof*⟩

**lemma** *xor-nat-unfold* [*code*]:
‹*m XOR n* = (*if m* = *0 then n else if n* = *0 then m*
*else* (*m mod 2* + *n mod 2*) *mod 2* + *2* ∗ ((*m div 2*) *XOR* (*n div 2*)))› **for** *m n*
:: *nat*
⟨*proof*⟩

**lemma** [*code*]:
‹*unset-bit 0 m* = *2* ∗ (*m div 2*)›
‹*unset-bit* (*Suc n*) *m* = *m mod 2* + *2* ∗ *unset-bit n* (*m div 2*)› **for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *push-bit-of-Suc-0* [*simp*]:
‹*push-bit n* (*Suc 0*) = *2* ^ *n*›
⟨*proof*⟩

**lemma** *take-bit-of-Suc-0* [*simp*]:
‹*take-bit n* (*Suc 0*) = *of-bool* (*0* < *n*)›
⟨*proof*⟩

**lemma** *drop-bit-of-Suc-0* [*simp*]:
‹*drop-bit n* (*Suc 0*) = *of-bool* (*n* = *0*)›
⟨*proof*⟩

**lemma** *Suc-mask-eq-exp*:
‹*Suc* (*mask n*) = *2* ^ *n*›
⟨*proof*⟩

**lemma** *less-eq-mask*:
‹*n* ≤ *mask n*›
⟨*proof*⟩

**lemma** *less-mask*:
‹*n* < *mask n*› **if** ‹*Suc 0* < *n*›
⟨*proof*⟩

**lemma** *mask-nat-less-exp* [*simp*]:
‹(*mask n* :: *nat*) < *2* ^ *n*›
⟨*proof*⟩

**lemma** *mask-nat-positive-iff* [*simp*]:
‹(*0*::*nat*) < *mask n* ⟷ *0* < *n*›
⟨*proof*⟩

**lemma** *take-bit-nat-less-eq-mask*:
‹*take-bit n m* ≤ *mask n*› **for** *m* :: *nat*
⟨*proof*⟩

**lemma** *take-bit-tightened-less-eq-nat*:
‹*take-bit m q* ≤ *take-bit n q*› **if** ‹*m* ≤ *n*› **for** *q* :: *nat*
⟨*proof*⟩

**lemma** *push-bit-nat-eq*:
‹*push-bit n* (*nat k*) = *nat* (*push-bit n k*)›
⟨*proof*⟩

**lemma** *drop-bit-nat-eq*:
‹*drop-bit n* (*nat k*) = *nat* (*drop-bit n k*)›
⟨*proof*⟩

**lemma** *take-bit-nat-eq*:
‹*take-bit n* (*nat k*) = *nat* (*take-bit n k*)› **if** ‹*k* ≥ *0*›
⟨*proof*⟩

**lemma** *nat-take-bit-eq*:
‹*nat* (*take-bit n k*) = *take-bit n* (*nat k*)›
**if** ‹*k* ≥ *0*›
⟨*proof*⟩

**lemma** *nat-mask-eq*:
‹*nat* (*mask n*) = *mask n*›
⟨*proof*⟩

## 68.6 Symbolic computations on numeral expressions

**context** *semiring-bits*
**begin**

**lemma** *bit-1-0* [*simp*]:
‹*bit 1 0*›
⟨*proof*⟩

**lemma** *not-bit-1-Suc* [*simp*]:
‹¬ *bit 1* (*Suc n*)›
⟨*proof*⟩

**lemma** *not-bit-1-numeral* [*simp*]:
‹¬ *bit 1* (*numeral m*)›
⟨*proof*⟩

**lemma** *not-bit-numeral-Bit0-0* [*simp*]:
‹¬ *bit* (*numeral* (*Num.Bit0 m*)) *0*›
⟨*proof*⟩

**lemma** *bit-numeral-Bit1-0* [*simp*]:
‹*bit* (*numeral* (*Num.Bit1 m*)) *0*›
⟨*proof*⟩

**lemma** *bit-numeral-Bit0-iff*:
‹*bit* (*numeral* (*num.Bit0 m*)) *n*
⟷ *possible-bit TYPE*('*a*) *n* ∧ *n* > *0* ∧ *bit* (*numeral m*) (*n* − *1*)›
⟨*proof*⟩

**lemma** *bit-numeral-Bit1-Suc-iff*:
‹*bit* (*numeral* (*num.Bit1 m*)) (*Suc n*)
⟷ *possible-bit TYPE*('*a*) (*Suc n*) ∧ *bit* (*numeral m*) *n*›
⟨*proof*⟩

**end**

**context** *ring-bit-operations*
**begin**

**lemma** *not-bit-minus-numeral-Bit0-0* [*simp*]:
‹¬ *bit* (− *numeral* (*Num.Bit0 m*)) *0*›
⟨*proof*⟩

**lemma** *bit-minus-numeral-Bit1-0* [*simp*]:
‹*bit* (− *numeral* (*Num.Bit1 m*)) *0*›
⟨*proof*⟩

**lemma** *bit-minus-numeral-Bit0-Suc-iff*:
‹*bit* (− *numeral* (*num.Bit0 m*)) (*Suc n*)
⟷ *possible-bit TYPE*('*a*) (*Suc n*) ∧ *bit* (− *numeral m*) *n*›
⟨*proof*⟩

**lemma** *bit-minus-numeral-Bit1-Suc-iff*:
‹*bit* (− *numeral* (*num.Bit1 m*)) (*Suc n*)
⟷ *possible-bit TYPE*('*a*) (*Suc n*) ∧ ¬ *bit* (*numeral m*) *n*›
⟨*proof*⟩

**lemma** *bit-numeral-BitM-0* [*simp*]:
‹*bit* (*numeral* (*Num.BitM m*)) *0*›
⟨*proof*⟩

**lemma** *bit-numeral-BitM-Suc-iff*:
‹*bit* (*numeral* (*Num.BitM m*)) (*Suc n*) ⟷ *possible-bit TYPE*('*a*) (*Suc n*) ∧ ¬ *bit* (− *numeral m*) *n*›
⟨*proof*⟩

**end**

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**lemma** *bit-numeral-iff*:

‹*bit* (*numeral m*) *n* ⟷ *bit* (*numeral m* :: *nat*) *n*›
⟨*proof*⟩

**lemma** *bit-numeral-Bit0-Suc-iff* [*simp*]:
‹*bit* (*numeral* (*Num.Bit0 m*)) (*Suc n*) ⟷ *bit* (*numeral m*) *n*›
⟨*proof*⟩

**lemma** *bit-numeral-Bit1-Suc-iff* [*simp*]:
‹*bit* (*numeral* (*Num.Bit1 m*)) (*Suc n*) ⟷ *bit* (*numeral m*) *n*›
⟨*proof*⟩

**lemma** *bit-numeral-rec*:
‹*bit* (*numeral* (*Num.Bit0 w*)) *n* ⟷ (*case n of 0* ⇒ *False* | *Suc m* ⇒ *bit* (*numeral w*) *m*)›
‹*bit* (*numeral* (*Num.Bit1 w*)) *n* ⟷ (*case n of 0* ⇒ *True* | *Suc m* ⇒ *bit* (*numeral w*) *m*)›
⟨*proof*⟩

**lemma** *bit-numeral-simps* [*simp*]:
‹*bit* (*numeral* (*Num.Bit0 w*)) (*numeral n*) ⟷ *bit* (*numeral w*) (*pred-numeral n*)›
‹*bit* (*numeral* (*Num.Bit1 w*)) (*numeral n*) ⟷ *bit* (*numeral w*) (*pred-numeral n*)›
⟨*proof*⟩

**lemma** *and-numerals* [*simp*]:
‹*1 AND numeral* (*Num.Bit0 y*) = *0*›
‹*1 AND numeral* (*Num.Bit1 y*) = *1*›
‹*numeral* (*Num.Bit0 x*) *AND numeral* (*Num.Bit0 y*) = *2* ∗ (*numeral x AND numeral y*)›
‹*numeral* (*Num.Bit0 x*) *AND numeral* (*Num.Bit1 y*) = *2* ∗ (*numeral x AND numeral y*)›
‹*numeral* (*Num.Bit0 x*) *AND 1* = *0*›
‹*numeral* (*Num.Bit1 x*) *AND numeral* (*Num.Bit0 y*) = *2* ∗ (*numeral x AND numeral y*)›
‹*numeral* (*Num.Bit1 x*) *AND numeral* (*Num.Bit1 y*) = *1* + *2* ∗ (*numeral x AND numeral y*)›
‹*numeral* (*Num.Bit1 x*) *AND 1* = *1*›
⟨*proof*⟩

**lemma** *or-numerals* [*simp*]:
‹*1 OR numeral* (*Num.Bit0 y*) = *numeral* (*Num.Bit1 y*)›
‹*1 OR numeral* (*Num.Bit1 y*) = *numeral* (*Num.Bit1 y*)›
‹*numeral* (*Num.Bit0 x*) *OR numeral* (*Num.Bit0 y*) = *2* ∗ (*numeral x OR numeral y*)›
‹*numeral* (*Num.Bit0 x*) *OR numeral* (*Num.Bit1 y*) = *1* + *2* ∗ (*numeral x OR numeral y*)›
‹*numeral* (*Num.Bit0 x*) *OR 1* = *numeral* (*Num.Bit1 x*)›
‹*numeral* (*Num.Bit1 x*) *OR numeral* (*Num.Bit0 y*) = *1* + *2* ∗ (*numeral x OR numeral y*)›
‹*numeral* (*Num.Bit1 x*) *OR numeral* (*Num.Bit1 y*) = *1* + *2* ∗ (*numeral x OR*

*numeral y*)›
‹*numeral* (*Num.Bit1 x*) *OR 1* = *numeral* (*Num.Bit1 x*)›
⟨*proof*⟩

**lemma** *xor-numerals* [*simp*]:
‹*1 XOR numeral* (*Num.Bit0 y*) = *numeral* (*Num.Bit1 y*)›
‹*1 XOR numeral* (*Num.Bit1 y*) = *numeral* (*Num.Bit0 y*)›
‹*numeral* (*Num.Bit0 x*) *XOR numeral* (*Num.Bit0 y*) = *2* ∗ (*numeral x XOR numeral y*)›
‹*numeral* (*Num.Bit0 x*) *XOR numeral* (*Num.Bit1 y*) = *1* + *2* ∗ (*numeral x XOR numeral y*)›
‹*numeral* (*Num.Bit0 x*) *XOR 1* = *numeral* (*Num.Bit1 x*)›
‹*numeral* (*Num.Bit1 x*) *XOR numeral* (*Num.Bit0 y*) = *1* + *2* ∗ (*numeral x XOR numeral y*)›
‹*numeral* (*Num.Bit1 x*) *XOR numeral* (*Num.Bit1 y*) = *2* ∗ (*numeral x XOR numeral y*)›
‹*numeral* (*Num.Bit1 x*) *XOR 1* = *numeral* (*Num.Bit0 x*)›
⟨*proof*⟩

**end**

**lemma** *drop-bit-Suc-minus-bit0* [*simp*]:
‹*drop-bit* (*Suc n*) (− *numeral* (*Num.Bit0 k*)) = *drop-bit n* (− *numeral k* :: *int*)›
⟨*proof*⟩

**lemma** *drop-bit-Suc-minus-bit1* [*simp*]:
‹*drop-bit* (*Suc n*) (− *numeral* (*Num.Bit1 k*)) = *drop-bit n* (− *numeral* (*Num.inc k*) :: *int*)›
⟨*proof*⟩

**lemma** *drop-bit-numeral-minus-bit0* [*simp*]:
‹*drop-bit* (*numeral l*) (− *numeral* (*Num.Bit0 k*)) = *drop-bit* (*pred-numeral l*) (− *numeral k* :: *int*)›
⟨*proof*⟩

**lemma** *drop-bit-numeral-minus-bit1* [*simp*]:
‹*drop-bit* (*numeral l*) (− *numeral* (*Num.Bit1 k*)) = *drop-bit* (*pred-numeral l*) (− *numeral* (*Num.inc k*) :: *int*)›
⟨*proof*⟩

**lemma** *take-bit-Suc-minus-bit0*:
‹*take-bit* (*Suc n*) (− *numeral* (*Num.Bit0 k*)) = *take-bit n* (− *numeral k*) ∗ (*2* :: *int*)›
⟨*proof*⟩

**lemma** *take-bit-Suc-minus-bit1*:
‹*take-bit* (*Suc n*) (− *numeral* (*Num.Bit1 k*)) = *take-bit n* (− *numeral* (*Num.inc k*)) ∗ *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *take-bit-numeral-minus-bit0*:
‹*take-bit* (*numeral l*) (− *numeral* (*Num.Bit0 k*)) = *take-bit* (*pred-numeral l*) (− *numeral k*) ∗ (*2* :: *int*)›
⟨*proof*⟩

**lemma** *take-bit-numeral-minus-bit1*:
‹*take-bit* (*numeral l*) (− *numeral* (*Num.Bit1 k*)) = *take-bit* (*pred-numeral l*) (− *numeral* (*Num.inc k*)) ∗ *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *and-nat-numerals* [*simp*]:
‹*Suc 0 AND numeral* (*Num.Bit0 y*) = *0*›
‹*Suc 0 AND numeral* (*Num.Bit1 y*) = *1*›
‹*numeral* (*Num.Bit0 x*) *AND Suc 0* = *0*›
‹*numeral* (*Num.Bit1 x*) *AND Suc 0* = *1*›
⟨*proof*⟩

**lemma** *or-nat-numerals* [*simp*]:
‹*Suc 0 OR numeral* (*Num.Bit0 y*) = *numeral* (*Num.Bit1 y*)›
‹*Suc 0 OR numeral* (*Num.Bit1 y*) = *numeral* (*Num.Bit1 y*)›
‹*numeral* (*Num.Bit0 x*) *OR Suc 0* = *numeral* (*Num.Bit1 x*)›
‹*numeral* (*Num.Bit1 x*) *OR Suc 0* = *numeral* (*Num.Bit1 x*)›
⟨*proof*⟩

**lemma** *xor-nat-numerals* [*simp*]:
‹*Suc 0 XOR numeral* (*Num.Bit0 y*) = *numeral* (*Num.Bit1 y*)›
‹*Suc 0 XOR numeral* (*Num.Bit1 y*) = *numeral* (*Num.Bit0 y*)›
‹*numeral* (*Num.Bit0 x*) *XOR Suc 0* = *numeral* (*Num.Bit1 x*)›
‹*numeral* (*Num.Bit1 x*) *XOR Suc 0* = *numeral* (*Num.Bit0 x*)›
⟨*proof*⟩

**context** *ring-bit-operations*
**begin**

**lemma** *minus-numeral-inc-eq*:
‹− *numeral* (*Num.inc n*) = *NOT* (*numeral n*)›
⟨*proof*⟩

**lemma** *sub-one-eq-not-neg*:
‹*Num.sub n num.One* = *NOT* (− *numeral n*)›
⟨*proof*⟩

**lemma** *minus-numeral-eq-not-sub-one*:
‹− *numeral n* = *NOT* (*Num.sub n num.One*)›
⟨*proof*⟩

**lemma** *not-numeral-eq* [*simp*]:
‹*NOT* (*numeral n*) = − *numeral* (*Num.inc n*)›

⟨*proof*⟩

**lemma** *not-minus-numeral-eq* [*simp*]:
‹*NOT* (− *numeral n*) = *Num.sub n num.One*›
⟨*proof*⟩

**lemma** *minus-not-numeral-eq* [*simp*]:
‹− (*NOT* (*numeral n*)) = *numeral* (*Num.inc n*)›
⟨*proof*⟩

**lemma** *not-numeral-BitM-eq*:
‹*NOT* (*numeral* (*Num.BitM n*)) = − *numeral* (*num.Bit0 n*)›
⟨*proof*⟩

**lemma** *not-numeral-Bit0-eq*:
‹*NOT* (*numeral* (*Num.Bit0 n*)) = − *numeral* (*num.Bit1 n*)›
⟨*proof*⟩

**end**

**lemma** *bit-minus-numeral-int* [*simp*]:
‹*bit* (− *numeral* (*num.Bit0 w*) :: *int*) (*numeral n*) ⟷ *bit* (− *numeral w* :: *int*) (*pred-numeral n*)›
‹*bit* (− *numeral* (*num.Bit1 w*) :: *int*) (*numeral n*) ⟷ ¬ *bit* (*numeral w* :: *int*) (*pred-numeral n*)›
⟨*proof*⟩

**lemma** *bit-minus-numeral-Bit0-Suc-iff* [*simp*]:
‹*bit* (− *numeral* (*num.Bit0 w*) :: *int*) (*Suc n*) ⟷ *bit* (− *numeral w* :: *int*) *n*›
⟨*proof*⟩

**lemma** *bit-minus-numeral-Bit1-Suc-iff* [*simp*]:
‹*bit* (− *numeral* (*num.Bit1 w*) :: *int*) (*Suc n*) ⟷ ¬ *bit* (*numeral w* :: *int*) *n*›
⟨*proof*⟩

**lemma** *and-not-numerals*:
‹*1 AND NOT 1* = (*0* :: *int*)›
‹*1 AND NOT* (*numeral* (*Num.Bit0 n*)) = (*1* :: *int*)›
‹*1 AND NOT* (*numeral* (*Num.Bit1 n*)) = (*0* :: *int*)›
‹*numeral* (*Num.Bit0 m*) *AND NOT* (*1* :: *int*) = *numeral* (*Num.Bit0 m*)›
‹*numeral* (*Num.Bit0 m*) *AND NOT* (*numeral* (*Num.Bit0 n*)) = (*2* :: *int*) ∗ (*numeral m AND NOT* (*numeral n*))›
‹*numeral* (*Num.Bit0 m*) *AND NOT* (*numeral* (*Num.Bit1 n*)) = (*2* :: *int*) ∗ (*numeral m AND NOT* (*numeral n*))›
‹*numeral* (*Num.Bit1 m*) *AND NOT* (*1* :: *int*) = *numeral* (*Num.Bit0 m*)›
‹*numeral* (*Num.Bit1 m*) *AND NOT* (*numeral* (*Num.Bit0 n*)) = *1* + (*2* :: *int*) ∗ (*numeral m AND NOT* (*numeral n*))›
‹*numeral* (*Num.Bit1 m*) *AND NOT* (*numeral* (*Num.Bit1 n*)) = (*2* :: *int*) ∗ (*numeral m AND NOT* (*numeral n*))›

⟨*proof*⟩

**fun** *and-not-num* :: ‹*num* ⇒ *num* ⇒ *num option*›
**where**
‹*and-not-num num.One num.One* = *None*›
| ‹*and-not-num num.One* (*num.Bit0 n*) = *Some num.One*›
| ‹*and-not-num num.One* (*num.Bit1 n*) = *None*›
| ‹*and-not-num* (*num.Bit0 m*) *num.One* = *Some* (*num.Bit0 m*)›
| ‹*and-not-num* (*num.Bit0 m*) (*num.Bit0 n*) = *map-option num.Bit0* (*and-not-num m n*)›
| ‹*and-not-num* (*num.Bit0 m*) (*num.Bit1 n*) = *map-option num.Bit0* (*and-not-num m n*)›
| ‹*and-not-num* (*num.Bit1 m*) *num.One* = *Some* (*num.Bit0 m*)›
| ‹*and-not-num* (*num.Bit1 m*) (*num.Bit0 n*) = (*case and-not-num m n of None* ⇒ *Some num.One* | *Some n'* ⇒ *Some* (*num.Bit1 n'*))›
| ‹*and-not-num* (*num.Bit1 m*) (*num.Bit1 n*) = *map-option num.Bit0* (*and-not-num m n*)›

**lemma** *int-numeral-and-not-num*:
‹*numeral m AND NOT* (*numeral n*) = (*case and-not-num m n of None* ⇒ *0* :: *int* | *Some n'* ⇒ *numeral n'*)›
⟨*proof*⟩

**lemma** *int-numeral-not-and-num*:
‹*NOT* (*numeral m*) *AND numeral n* = (*case and-not-num n m of None* ⇒ *0* :: *int* | *Some n'* ⇒ *numeral n'*)›
⟨*proof*⟩

**lemma** *and-not-num-eq-None-iff*:
‹*and-not-num m n* = *None* ⟷ *numeral m AND NOT* (*numeral n*) = (*0* :: *int*)›
⟨*proof*⟩

**lemma** *and-not-num-eq-Some-iff*:
‹*and-not-num m n* = *Some q* ⟷ *numeral m AND NOT* (*numeral n*) = (*numeral q* :: *int*)›
⟨*proof*⟩

**lemma** *and-minus-numerals* [*simp*]:
‹*1 AND* − (*numeral* (*num.Bit0 n*)) = (*0*::*int*)›
‹*1 AND* − (*numeral* (*num.Bit1 n*)) = (*1*::*int*)›
‹*numeral m AND* − (*numeral* (*num.Bit0 n*)) = (*case and-not-num m* (*Num.BitM n*) *of None* ⇒ *0* :: *int* | *Some n'* ⇒ *numeral n'*)›
‹*numeral m AND* − (*numeral* (*num.Bit1 n*)) = (*case and-not-num m* (*Num.Bit0 n*) *of None* ⇒ *0* :: *int* | *Some n'* ⇒ *numeral n'*)›
‹− (*numeral* (*num.Bit0 n*)) *AND 1* = (*0*::*int*)›
‹− (*numeral* (*num.Bit1 n*)) *AND 1* = (*1*::*int*)›
‹− (*numeral* (*num.Bit0 n*)) *AND numeral m* = (*case and-not-num m* (*Num.BitM n*) *of None* ⇒ *0* :: *int* | *Some n'* ⇒ *numeral n'*)›
‹− (*numeral* (*num.Bit1 n*)) *AND numeral m* = (*case and-not-num m* (*Num.Bit0*

*n*) *of None* ⇒ *0* :: *int* | *Some n′* ⇒ *numeral n′*)›
⟨*proof*⟩

**lemma** *and-minus-minus-numerals* [*simp*]:
‹− (*numeral m* :: *int*) *AND* − (*numeral n* :: *int*) = *NOT* ((*numeral m* − *1*) *OR* (*numeral n* − *1*))›
⟨*proof*⟩

**lemma** *or-not-numerals*:
‹*1 OR NOT 1* = *NOT* (*0* :: *int*)›
‹*1 OR NOT* (*numeral* (*Num.Bit0 n*)) = *NOT* (*numeral* (*Num.Bit0 n*) :: *int*)›
‹*1 OR NOT* (*numeral* (*Num.Bit1 n*)) = *NOT* (*numeral* (*Num.Bit0 n*) :: *int*)›
‹*numeral* (*Num.Bit0 m*) *OR NOT* (*1* :: *int*) = *NOT* (*1* :: *int*)›
‹*numeral* (*Num.Bit0 m*) *OR NOT* (*numeral* (*Num.Bit0 n*)) = *1* + (*2* :: *int*) ∗ (*numeral m OR NOT* (*numeral n*))›
‹*numeral* (*Num.Bit0 m*) *OR NOT* (*numeral* (*Num.Bit1 n*)) = (*2* :: *int*) ∗ (*numeral m OR NOT* (*numeral n*))›
‹*numeral* (*Num.Bit1 m*) *OR NOT* (*1* :: *int*) = *NOT* (*0* :: *int*)›
‹*numeral* (*Num.Bit1 m*) *OR NOT* (*numeral* (*Num.Bit0 n*)) = *1* + (*2* :: *int*) ∗ (*numeral m OR NOT* (*numeral n*))›
‹*numeral* (*Num.Bit1 m*) *OR NOT* (*numeral* (*Num.Bit1 n*)) = *1* + (*2* :: *int*) ∗ (*numeral m OR NOT* (*numeral n*))›
⟨*proof*⟩

**fun** *or-not-num-neg* :: ‹*num* ⇒ *num* ⇒ *num*›
**where**
‹*or-not-num-neg num.One num.One* = *num.One*›
| ‹*or-not-num-neg num.One* (*num.Bit0 m*) = *num.Bit1 m*›
| ‹*or-not-num-neg num.One* (*num.Bit1 m*) = *num.Bit1 m*›
| ‹*or-not-num-neg* (*num.Bit0 n*) *num.One* = *num.Bit0 num.One*›
| ‹*or-not-num-neg* (*num.Bit0 n*) (*num.Bit0 m*) = *Num.BitM* (*or-not-num-neg n m*)›
| ‹*or-not-num-neg* (*num.Bit0 n*) (*num.Bit1 m*) = *num.Bit0* (*or-not-num-neg n m*)›
| ‹*or-not-num-neg* (*num.Bit1 n*) *num.One* = *num.One*›
| ‹*or-not-num-neg* (*num.Bit1 n*) (*num.Bit0 m*) = *Num.BitM* (*or-not-num-neg n m*)›
| ‹*or-not-num-neg* (*num.Bit1 n*) (*num.Bit1 m*) = *Num.BitM* (*or-not-num-neg n m*)›

**lemma** *int-numeral-or-not-num-neg*:
‹*numeral m OR NOT* (*numeral n* :: *int*) = − *numeral* (*or-not-num-neg m n*)›
⟨*proof*⟩

**lemma** *int-numeral-not-or-num-neg*:
‹*NOT* (*numeral m*) *OR* (*numeral n* :: *int*) = − *numeral* (*or-not-num-neg n m*)›
⟨*proof*⟩

**lemma** *numeral-or-not-num-eq*:
‹*numeral* (*or-not-num-neg m n*) = − (*numeral m OR NOT* (*numeral n* :: *int*))›

⟨*proof*⟩

**lemma** *or-minus-numerals* [*simp*]:
‹*1 OR* − (*numeral* (*num.Bit0 n*)) = − (*numeral* (*or-not-num-neg num.One* (*Num.BitM n*)) :: *int*)›
‹*1 OR* − (*numeral* (*num.Bit1 n*)) = − (*numeral* (*num.Bit1 n*) :: *int*)›
‹*numeral m OR* − (*numeral* (*num.Bit0 n*)) = − (*numeral* (*or-not-num-neg m* (*Num.BitM n*)) :: *int*)›
‹*numeral m OR* − (*numeral* (*num.Bit1 n*)) = − (*numeral* (*or-not-num-neg m* (*Num.Bit0 n*)) :: *int*)›
‹− (*numeral* (*num.Bit0 n*)) *OR 1* = − (*numeral* (*or-not-num-neg num.One* (*Num.BitM n*)) :: *int*)›
‹− (*numeral* (*num.Bit1 n*)) *OR 1* = − (*numeral* (*num.Bit1 n*) :: *int*)›
‹− (*numeral* (*num.Bit0 n*)) *OR numeral m* = − (*numeral* (*or-not-num-neg m* (*Num.BitM n*)) :: *int*)›
‹− (*numeral* (*num.Bit1 n*)) *OR numeral m* = − (*numeral* (*or-not-num-neg m* (*Num.Bit0 n*)) :: *int*)›
⟨*proof*⟩

**lemma** *or-minus-minus-numerals* [*simp*]:
‹− (*numeral m* :: *int*) *OR* − (*numeral n* :: *int*) = *NOT* ((*numeral m* − *1*) *AND* (*numeral n* − *1*))›
⟨*proof*⟩

**lemma** *xor-minus-numerals* [*simp*]:
‹− *numeral n XOR k* = *NOT* (*neg-numeral-class.sub n num.One XOR k*)›
‹*k XOR* − *numeral n* = *NOT* (*k XOR* (*neg-numeral-class.sub n num.One*))› **for** *k* :: *int*
⟨*proof*⟩

**definition** *take-bit-num* :: ‹*nat* ⇒ *num* ⇒ *num option*›
**where** ‹*take-bit-num n m* =
(*if take-bit n* (*numeral m* :: *nat*) = *0 then None else Some* (*num-of-nat* (*take-bit n* (*numeral m* :: *nat*))))›

**lemma** *take-bit-num-simps*:
‹*take-bit-num 0 m* = *None*›
‹*take-bit-num* (*Suc n*) *Num.One* =
*Some Num.One*›
‹*take-bit-num* (*Suc n*) (*Num.Bit0 m*) =
(*case take-bit-num n m of None* ⇒ *None* | *Some q* ⇒ *Some* (*Num.Bit0 q*))›
‹*take-bit-num* (*Suc n*) (*Num.Bit1 m*) =
*Some* (*case take-bit-num n m of None* ⇒ *Num.One* | *Some q* ⇒ *Num.Bit1 q*)›
‹*take-bit-num* (*numeral r*) *Num.One* =
*Some Num.One*›
‹*take-bit-num* (*numeral r*) (*Num.Bit0 m*) =
(*case take-bit-num* (*pred-numeral r*) *m of None* ⇒ *None* | *Some q* ⇒ *Some* (*Num.Bit0 q*))›
‹*take-bit-num* (*numeral r*) (*Num.Bit1 m*) =

*Some* (*case take-bit-num* (*pred-numeral r*) *m of None* ⇒ *Num.One* | *Some q* ⇒ *Num.Bit1 q*))›
⟨*proof*⟩

**lemma** *take-bit-num-code* [*code*]:
— Ocaml-style pattern matching is more robust wrt. different representations of *nat*
‹*take-bit-num n m* = (*case* (*n*, *m*)
*of* (*0*, -) ⇒ *None*
| (*Suc n*, *Num.One*) ⇒ *Some Num.One*
| (*Suc n*, *Num.Bit0 m*) ⇒ (*case take-bit-num n m of None* ⇒ *None* | *Some q* ⇒ *Some* (*Num.Bit0 q*))
| (*Suc n*, *Num.Bit1 m*) ⇒ *Some* (*case take-bit-num n m of None* ⇒ *Num.One* | *Some q* ⇒ *Num.Bit1 q*))›
⟨*proof*⟩

**context** *semiring-bit-operations*
**begin**

**lemma** *take-bit-num-eq-None-imp*:
‹*take-bit m* (*numeral n*) = *0*› **if** ‹*take-bit-num m n* = *None*›
⟨*proof*⟩

**lemma** *take-bit-num-eq-Some-imp*:
‹*take-bit m* (*numeral n*) = *numeral q*› **if** ‹*take-bit-num m n* = *Some q*›
⟨*proof*⟩

**lemma** *take-bit-numeral-numeral*:
‹*take-bit* (*numeral m*) (*numeral n*) =
(*case take-bit-num* (*numeral m*) *n of None* ⇒ *0* | *Some q* ⇒ *numeral q*)›
⟨*proof*⟩

**end**

**lemma** *take-bit-numeral-minus-numeral-int*:
‹*take-bit* (*numeral m*) (− *numeral n* :: *int*) =
(*case take-bit-num* (*numeral m*) *n of None* ⇒ *0* | *Some q* ⇒ *take-bit* (*numeral m*) (*2* ^ *numeral m* − *numeral q*))› (**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

**declare** *take-bit-num-simps* [*simp*]
*take-bit-numeral-numeral* [*simp*]
*take-bit-numeral-minus-numeral-int* [*simp*]

## 68.7 Symbolic computations for code generation

**lemma** *bit-int-code* [*code*]:
‹*bit* (*0*::*int*) *n* ⟷ *False*›
‹*bit* (*Int.Neg num.One*) *n* ⟷ *True*›

‹bit (Int.Pos num.One) 0 ⟷ True›
‹bit (Int.Pos (num.Bit0 m)) 0 ⟷ False›
‹bit (Int.Pos (num.Bit1 m)) 0 ⟷ True›
‹bit (Int.Neg (num.Bit0 m)) 0 ⟷ False›
‹bit (Int.Neg (num.Bit1 m)) 0 ⟷ True›
‹bit (Int.Pos num.One) (Suc n) ⟷ False›
‹bit (Int.Pos (num.Bit0 m)) (Suc n) ⟷ bit (Int.Pos m) n›
‹bit (Int.Pos (num.Bit1 m)) (Suc n) ⟷ bit (Int.Pos m) n›
‹bit (Int.Neg (num.Bit0 m)) (Suc n) ⟷ bit (Int.Neg m) n›
‹bit (Int.Neg (num.Bit1 m)) (Suc n) ⟷ bit (Int.Neg (Num.inc m)) n›
⟨proof⟩

**lemma** *not-int-code* [*code*]:
‹NOT (0 :: int) = − 1›
‹NOT (Int.Pos n) = Int.Neg (Num.inc n)›
‹NOT (Int.Neg n) = Num.sub n num.One›
⟨proof⟩

**fun** *and-num* :: ‹num ⇒ num ⇒ num option›
**where**
‹and-num num.One num.One = Some num.One›
| ‹and-num num.One (num.Bit0 n) = None›
| ‹and-num num.One (num.Bit1 n) = Some num.One›
| ‹and-num (num.Bit0 m) num.One = None›
| ‹and-num (num.Bit0 m) (num.Bit0 n) = map-option num.Bit0 (and-num m n)›
| ‹and-num (num.Bit0 m) (num.Bit1 n) = map-option num.Bit0 (and-num m n)›
| ‹and-num (num.Bit1 m) num.One = Some num.One›
| ‹and-num (num.Bit1 m) (num.Bit0 n) = map-option num.Bit0 (and-num m n)›
| ‹and-num (num.Bit1 m) (num.Bit1 n) = (case and-num m n of None ⇒ Some num.One | Some n′ ⇒ Some (num.Bit1 n′))›

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**lemma** *numeral-and-num*:
‹numeral m AND numeral n = (case and-num m n of None ⇒ 0 | Some n′ ⇒ numeral n′)›
⟨proof⟩

**lemma** *and-num-eq-None-iff*:
‹and-num m n = None ⟷ numeral m AND numeral n = 0›
⟨proof⟩

**lemma** *and-num-eq-Some-iff*:
‹and-num m n = Some q ⟷ numeral m AND numeral n = numeral q›
⟨proof⟩

**end**

**lemma** *and-int-code* [*code*]:
**fixes** *i j* :: *int* **shows**
‹*0 AND j = 0*›
‹*i AND 0 = 0*›
‹*Int.Pos n AND Int.Pos m = (case and-num n m of None ⇒ 0 | Some n′ ⇒ Int.Pos n′)*›
‹*Int.Neg n AND Int.Neg m = NOT (Num.sub n num.One OR Num.sub m num.One)*›
‹*Int.Pos n AND Int.Neg num.One = Int.Pos n*›
‹*Int.Pos n AND Int.Neg (num.Bit0 m) = Num.sub (or-not-num-neg (Num.BitM m) n) num.One*›
‹*Int.Pos n AND Int.Neg (num.Bit1 m) = Num.sub (or-not-num-neg (num.Bit0 m) n) num.One*›
‹*Int.Neg num.One AND Int.Pos m = Int.Pos m*›
‹*Int.Neg (num.Bit0 n) AND Int.Pos m = Num.sub (or-not-num-neg (Num.BitM n) m) num.One*›
‹*Int.Neg (num.Bit1 n) AND Int.Pos m = Num.sub (or-not-num-neg (num.Bit0 n) m) num.One*›
⟨*proof*⟩

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**fun** *or-num* :: ‹*num ⇒ num ⇒ num*›
**where**
‹*or-num num.One num.One = num.One*›
| ‹*or-num num.One (num.Bit0 n) = num.Bit1 n*›
| ‹*or-num num.One (num.Bit1 n) = num.Bit1 n*›
| ‹*or-num (num.Bit0 m) num.One = num.Bit1 m*›
| ‹*or-num (num.Bit0 m) (num.Bit0 n) = num.Bit0 (or-num m n)*›
| ‹*or-num (num.Bit0 m) (num.Bit1 n) = num.Bit1 (or-num m n)*›
| ‹*or-num (num.Bit1 m) num.One = num.Bit1 m*›
| ‹*or-num (num.Bit1 m) (num.Bit0 n) = num.Bit1 (or-num m n)*›
| ‹*or-num (num.Bit1 m) (num.Bit1 n) = num.Bit1 (or-num m n)*›

**lemma** *numeral-or-num*:
‹*numeral m OR numeral n = numeral (or-num m n)*›
⟨*proof*⟩

**lemma** *numeral-or-num-eq*:
‹*numeral (or-num m n) = numeral m OR numeral n*›
⟨*proof*⟩

**end**

**lemma** *or-int-code* [*code*]:
**fixes** *i j* :: *int* **shows**
‹*0 OR j = j*›
‹*i OR 0 = i*›

‹*Int.Pos n OR Int.Pos m = Int.Pos (or-num n m)*›
‹*Int.Neg n OR Int.Neg m = NOT (Num.sub n num.One AND Num.sub m num.One)*›
‹*Int.Pos n OR Int.Neg num.One = Int.Neg num.One*›
‹*Int.Pos n OR Int.Neg (num.Bit0 m) = (case and-not-num (Num.BitM m) n of None ⇒ −1 | Some n′ ⇒ Int.Neg (Num.inc n′))*›
‹*Int.Pos n OR Int.Neg (num.Bit1 m) = (case and-not-num (num.Bit0 m) n of None ⇒ −1 | Some n′ ⇒ Int.Neg (Num.inc n′))*›
‹*Int.Neg num.One OR Int.Pos m = Int.Neg num.One*›
‹*Int.Neg (num.Bit0 n) OR Int.Pos m = (case and-not-num (Num.BitM n) m of None ⇒ −1 | Some n′ ⇒ Int.Neg (Num.inc n′))*›
‹*Int.Neg (num.Bit1 n) OR Int.Pos m = (case and-not-num (num.Bit0 n) m of None ⇒ −1 | Some n′ ⇒ Int.Neg (Num.inc n′))*›
⟨*proof*⟩

**fun** *xor-num* :: ‹*num ⇒ num ⇒ num option*›
**where**
‹*xor-num num.One num.One = None*›
| ‹*xor-num num.One (num.Bit0 n) = Some (num.Bit1 n)*›
| ‹*xor-num num.One (num.Bit1 n) = Some (num.Bit0 n)*›
| ‹*xor-num (num.Bit0 m) num.One = Some (num.Bit1 m)*›
| ‹*xor-num (num.Bit0 m) (num.Bit0 n) = map-option num.Bit0 (xor-num m n)*›
| ‹*xor-num (num.Bit0 m) (num.Bit1 n) = Some (case xor-num m n of None ⇒ num.One | Some n′ ⇒ num.Bit1 n′)*›
| ‹*xor-num (num.Bit1 m) num.One = Some (num.Bit0 m)*›
| ‹*xor-num (num.Bit1 m) (num.Bit0 n) = Some (case xor-num m n of None ⇒ num.One | Some n′ ⇒ num.Bit1 n′)*›
| ‹*xor-num (num.Bit1 m) (num.Bit1 n) = map-option num.Bit0 (xor-num m n)*›

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**lemma** *numeral-xor-num*:
‹*numeral m XOR numeral n = (case xor-num m n of None ⇒ 0 | Some n′ ⇒ numeral n′)*›
⟨*proof*⟩

**lemma** *xor-num-eq-None-iff*:
‹*xor-num m n = None ⟷ numeral m XOR numeral n = 0*›
⟨*proof*⟩

**lemma** *xor-num-eq-Some-iff*:
‹*xor-num m n = Some q ⟷ numeral m XOR numeral n = numeral q*›
⟨*proof*⟩

**end**

**context** *semiring-bit-operations*
**begin**

**lemma** *push-bit-eq-pow*:
‹*push-bit* (*numeral n*) *1* = *numeral* (*Num.pow* (*Num.Bit0 Num.One*) *n*)›
⟨*proof*⟩

**lemma** *set-bit-of-0* [*simp*]:
‹*set-bit n 0* = *2* ^ *n*›
⟨*proof*⟩

**lemma** *unset-bit-of-0* [*simp*]:
‹*unset-bit n 0* = *0*›
⟨*proof*⟩

**lemma** *flip-bit-of-0* [*simp*]:
‹*flip-bit n 0* = *2* ^ *n*›
⟨*proof*⟩

**lemma** *set-bit-0-numeral-eq* [*simp*]:
‹*set-bit 0* (*numeral Num.One*) = *1*›
‹*set-bit 0* (*numeral* (*Num.Bit0 m*)) = *numeral* (*Num.Bit1 m*)›
‹*set-bit 0* (*numeral* (*Num.Bit1 m*)) = *numeral* (*Num.Bit1 m*)›
⟨*proof*⟩

**lemma** *set-bit-numeral-eq-or* [*simp*]:
‹*set-bit* (*numeral n*) (*numeral m*) = *numeral m OR push-bit* (*numeral n*) *1*›
⟨*proof*⟩

**lemma** *unset-bit-0-numeral-eq-and-not'* [*simp*]:
‹*unset-bit 0* (*numeral Num.One*) = *0*›
‹*unset-bit 0* (*numeral* (*Num.Bit0 m*)) = *numeral* (*Num.Bit0 m*)›
‹*unset-bit 0* (*numeral* (*Num.Bit1 m*)) = *numeral* (*Num.Bit0 m*)›
⟨*proof*⟩

**lemma** *unset-bit-numeral-eq-or* [*simp*]:
‹*unset-bit* (*numeral n*) (*numeral m*) =
(*case and-not-num m* (*Num.pow* (*Num.Bit0 Num.One*) *n*)
*of None* ⇒ *0*
| *Some q* ⇒ *numeral q*)› (**is** ‹*?lhs* = -›)
⟨*proof*⟩

**lemma** *flip-bit-0-numeral-eq-or* [*simp*]:
‹*flip-bit 0* (*numeral Num.One*) = *0*›
‹*flip-bit 0* (*numeral* (*Num.Bit0 m*)) = *numeral* (*Num.Bit1 m*)›
‹*flip-bit 0* (*numeral* (*Num.Bit1 m*)) = *numeral* (*Num.Bit0 m*)›
⟨*proof*⟩

**lemma** *flip-bit-numeral-eq-xor* [*simp*]:
‹*flip-bit* (*numeral n*) (*numeral m*) = *numeral m XOR push-bit* (*numeral n*) *1*›
⟨*proof*⟩

**end**

**context** *ring-bit-operations*
**begin**

**lemma** *set-bit-minus-numeral-eq-or* [*simp*]:
‹*set-bit* (*numeral n*) (− *numeral m*) = − *numeral m OR push-bit* (*numeral n*) *1*›
⟨*proof*⟩

**lemma** *unset-bit-minus-numeral-eq-and-not* [*simp*]:
‹*unset-bit* (*numeral n*) (− *numeral m*) = − *numeral m AND NOT* (*push-bit* (*numeral n*) *1*)›
⟨*proof*⟩

**lemma** *flip-bit-minus-numeral-eq-xor* [*simp*]:
‹*flip-bit* (*numeral n*) (− *numeral m*) = − *numeral m XOR push-bit* (*numeral n*) *1*›
⟨*proof*⟩

**end**

**lemma** *xor-int-code* [*code*]:
**fixes** *i j* :: *int* **shows**
‹*0 XOR j* = *j*›
‹*i XOR 0* = *i*›
‹*Int.Pos n XOR Int.Pos m* = (*case xor-num n m of None* ⇒ *0* | *Some n′* ⇒ *Int.Pos n′*)›
‹*Int.Neg n XOR Int.Neg m* = *Num.sub n num.One XOR Num.sub m num.One*›
‹*Int.Neg n XOR Int.Pos m* = *NOT* (*Num.sub n num.One XOR Int.Pos m*)›
‹*Int.Pos n XOR Int.Neg m* = *NOT* (*Int.Pos n XOR Num.sub m num.One*)›
⟨*proof*⟩

**lemma** *push-bit-int-code* [*code*]:
‹*push-bit 0 i* = *i*›
‹*push-bit* (*Suc n*) *i* = *push-bit n* (*Int.dup i*)›
⟨*proof*⟩

**lemma** *drop-bit-int-code* [*code*]:
**fixes** *i* :: *int* **shows**
‹*drop-bit 0 i* = *i*›
‹*drop-bit* (*Suc n*) *0* = (*0* :: *int*)›
‹*drop-bit* (*Suc n*) (*Int.Pos num.One*) = *0*›
‹*drop-bit* (*Suc n*) (*Int.Pos* (*num.Bit0 m*)) = *drop-bit n* (*Int.Pos m*)›
‹*drop-bit* (*Suc n*) (*Int.Pos* (*num.Bit1 m*)) = *drop-bit n* (*Int.Pos m*)›
‹*drop-bit* (*Suc n*) (*Int.Neg num.One*) = − *1*›
‹*drop-bit* (*Suc n*) (*Int.Neg* (*num.Bit0 m*)) = *drop-bit n* (*Int.Neg m*)›
‹*drop-bit* (*Suc n*) (*Int.Neg* (*num.Bit1 m*)) = *drop-bit n* (*Int.Neg* (*Num.inc m*))›
⟨*proof*⟩

## 68.8 More properties

**lemma** *take-bit-eq-mask-iff*:
‹*take-bit n k = mask n* ⟷ *take-bit n* (*k* + *1*) = *0*› (**is** ‹*?P* ⟷ *?Q*›)
**for** *k* :: *int*
⟨*proof*⟩

**lemma** *take-bit-eq-mask-iff-exp-dvd*:
‹*take-bit n k = mask n* ⟷ *2* ^ *n dvd k* + *1*›
**for** *k* :: *int*
⟨*proof*⟩

## 68.9 Bit concatenation

**definition** *concat-bit* :: ‹*nat* ⇒ *int* ⇒ *int* ⇒ *int*›
**where** ‹*concat-bit n k l = take-bit n k OR push-bit n l*›

**lemma** *bit-concat-bit-iff* [*bit-simps*]:
‹*bit* (*concat-bit m k l*) *n* ⟷ *n* < *m* ∧ *bit k n* ∨ *m* ≤ *n* ∧ *bit l* (*n* − *m*)›
⟨*proof*⟩

**lemma** *concat-bit-eq*:
‹*concat-bit n k l = take-bit n k* + *push-bit n l*›
⟨*proof*⟩

**lemma** *concat-bit-0* [*simp*]:
‹*concat-bit 0 k l = l*›
⟨*proof*⟩

**lemma** *concat-bit-Suc*:
‹*concat-bit* (*Suc n*) *k l = k mod 2* + *2* ∗ *concat-bit n* (*k div 2*) *l*›
⟨*proof*⟩

**lemma** *concat-bit-of-zero-1* [*simp*]:
‹*concat-bit n 0 l = push-bit n l*›
⟨*proof*⟩

**lemma** *concat-bit-of-zero-2* [*simp*]:
‹*concat-bit n k 0 = take-bit n k*›
⟨*proof*⟩

**lemma** *concat-bit-nonnegative-iff* [*simp*]:
‹*concat-bit n k l* ≥ *0* ⟷ *l* ≥ *0*›
⟨*proof*⟩

**lemma** *concat-bit-negative-iff* [*simp*]:
‹*concat-bit n k l* < *0* ⟷ *l* < *0*›
⟨*proof*⟩

**lemma** *concat-bit-assoc*:

‹*concat-bit n k* (*concat-bit m l r*) = *concat-bit* (*m* + *n*) (*concat-bit n k l*) *r*›
⟨*proof*⟩

**lemma** *concat-bit-assoc-sym*:
‹*concat-bit m* (*concat-bit n k l*) *r* = *concat-bit* (*min m n*) *k* (*concat-bit* (*m* − *n*) *l r*)›
⟨*proof*⟩

**lemma** *concat-bit-eq-iff*:
‹*concat-bit n k l* = *concat-bit n r s*
⟷ *take-bit n k* = *take-bit n r* ∧ *l* = *s*› (**is** ‹*?P* ⟷ *?Q*›)
⟨*proof*⟩

**lemma** *take-bit-concat-bit-eq*:
‹*take-bit m* (*concat-bit n k l*) = *concat-bit* (*min m n*) *k* (*take-bit* (*m* − *n*) *l*)›
⟨*proof*⟩

**lemma** *concat-bit-take-bit-eq*:
‹*concat-bit n* (*take-bit n b*) = *concat-bit n b*›
⟨*proof*⟩

## 68.10 Taking bits with sign propagation

**context** *ring-bit-operations*
**begin**

**definition** *signed-take-bit* :: ‹*nat* ⇒ *'a* ⇒ *'a*›
**where** ‹*signed-take-bit n a* = *take-bit n a OR* (*of-bool* (*bit a n*) ∗ *NOT* (*mask n*))›

**lemma** *signed-take-bit-eq-if-positive*:
‹*signed-take-bit n a* = *take-bit n a*› **if** ‹¬ *bit a n*›
⟨*proof*⟩

**lemma** *signed-take-bit-eq-if-negative*:
‹*signed-take-bit n a* = *take-bit n a OR NOT* (*mask n*)› **if** ‹*bit a n*›
⟨*proof*⟩

**lemma** *even-signed-take-bit-iff*:
‹*even* (*signed-take-bit m a*) ⟷ *even a*›
⟨*proof*⟩

**lemma** *bit-signed-take-bit-iff* [*bit-simps*]:
‹*bit* (*signed-take-bit m a*) *n* ⟷ *possible-bit TYPE*(*'a*) *n* ∧ *bit a* (*min m n*)›
⟨*proof*⟩

**lemma** *signed-take-bit-0* [*simp*]:
‹*signed-take-bit 0 a* = − (*a mod 2*)›
⟨*proof*⟩

**lemma** *signed-take-bit-Suc*:
‹*signed-take-bit* (*Suc n*) *a* = *a mod 2* + *2* ∗ *signed-take-bit n* (*a div 2*)›
⟨*proof*⟩

**lemma** *signed-take-bit-of-0* [*simp*]:
‹*signed-take-bit n 0* = *0*›
⟨*proof*⟩

**lemma** *signed-take-bit-of-minus-1* [*simp*]:
‹*signed-take-bit n* (− *1*) = − *1*›
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-1* [*simp*]:
‹*signed-take-bit* (*Suc n*) *1* = *1*›
⟨*proof*⟩

**lemma** *signed-take-bit-numeral-of-1* [*simp*]:
‹*signed-take-bit* (*numeral k*) *1* = *1*›
⟨*proof*⟩

**lemma** *signed-take-bit-rec*:
‹*signed-take-bit n a* = (*if n* = *0 then* − (*a mod 2*) *else a mod 2* + *2* ∗ *signed-take-bit* (*n* − *1*) (*a div 2*))›
⟨*proof*⟩

**lemma** *signed-take-bit-eq-iff-take-bit-eq*:
‹*signed-take-bit n a* = *signed-take-bit n b* ⟷ *take-bit* (*Suc n*) *a* = *take-bit* (*Suc n*) *b*›
⟨*proof*⟩

**lemma** *signed-take-bit-signed-take-bit* [*simp*]:
‹*signed-take-bit m* (*signed-take-bit n a*) = *signed-take-bit* (*min m n*) *a*›
⟨*proof*⟩

**lemma** *signed-take-bit-take-bit*:
‹*signed-take-bit m* (*take-bit n a*) = (*if n* ≤ *m then take-bit n else signed-take-bit m*) *a*›
⟨*proof*⟩

**lemma** *take-bit-signed-take-bit*:
‹*take-bit m* (*signed-take-bit n a*) = *take-bit m a*› **if** ‹*m* ≤ *Suc n*›
⟨*proof*⟩

**lemma** *signed-take-bit-eq-take-bit-add*:
‹*signed-take-bit n k* = *take-bit* (*Suc n*) *k* + *NOT* (*mask* (*Suc n*)) ∗ *of-bool* (*bit k n*)›
⟨*proof*⟩

**lemma** *signed-take-bit-eq-take-bit-minus*:
‹$signed\text{-}take\text{-}bit\ n\ k = take\text{-}bit\ (Suc\ n)\ k - 2\ \widehat{}\ Suc\ n * of\text{-}bool\ (bit\ k\ n)$›
⟨*proof*⟩

**end**

Modulus centered around 0

**lemma** *signed-take-bit-eq-concat-bit*:
‹$signed\text{-}take\text{-}bit\ n\ k = concat\text{-}bit\ n\ k\ (-\ of\text{-}bool\ (bit\ k\ n))$›
⟨*proof*⟩

**lemma** *signed-take-bit-add*:
‹$signed\text{-}take\text{-}bit\ n\ (signed\text{-}take\text{-}bit\ n\ k + signed\text{-}take\text{-}bit\ n\ l) = signed\text{-}take\text{-}bit\ n\ (k + l)$›
**for** $k\ l$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-diff*:
‹$signed\text{-}take\text{-}bit\ n\ (signed\text{-}take\text{-}bit\ n\ k - signed\text{-}take\text{-}bit\ n\ l) = signed\text{-}take\text{-}bit\ n\ (k - l)$›
**for** $k\ l$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-minus*:
‹$signed\text{-}take\text{-}bit\ n\ (-\ signed\text{-}take\text{-}bit\ n\ k) = signed\text{-}take\text{-}bit\ n\ (-\ k)$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-mult*:
‹$signed\text{-}take\text{-}bit\ n\ (signed\text{-}take\text{-}bit\ n\ k * signed\text{-}take\text{-}bit\ n\ l) = signed\text{-}take\text{-}bit\ n\ (k * l)$›
**for** $k\ l$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-eq-take-bit-shift*:
‹$signed\text{-}take\text{-}bit\ n\ k = take\text{-}bit\ (Suc\ n)\ (k + 2\ \widehat{}\ n) - 2\ \widehat{}\ n$› (**is** ‹$?lhs = ?rhs$›)
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-nonnegative-iff* [*simp*]:
‹$0 \leq signed\text{-}take\text{-}bit\ n\ k \longleftrightarrow \neg\ bit\ k\ n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-negative-iff* [*simp*]:
‹$signed\text{-}take\text{-}bit\ n\ k < 0 \longleftrightarrow bit\ k\ n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-greater-eq-minus-exp* [*simp*]:
‹$-$ ($2$ ^ $n$) $\leq$ *signed-take-bit n k*›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-less-exp* [*simp*]:
‹*signed-take-bit n k* $<$ $2$ ^ $n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-eq-self-iff*:
‹*signed-take-bit n k* $= k \longleftrightarrow -$ ($2$ ^ $n$) $\leq k \wedge k <$ $2$ ^ $n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-eq-self*:
‹*signed-take-bit n k* $= k$› **if** ‹$-$ ($2$ ^ $n$) $\leq k$› ‹$k <$ $2$ ^ $n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-less-eq-self-iff*:
‹*signed-take-bit n k* $\leq k \longleftrightarrow -$ ($2$ ^ $n$) $\leq k$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-less-self-iff*:
‹*signed-take-bit n k* $< k \longleftrightarrow$ $2$ ^ $n \leq k$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-greater-self-iff*:
‹$k <$ *signed-take-bit n k* $\longleftrightarrow k < -$ ($2$ ^ $n$)›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-greater-eq-self-iff*:
‹$k \leq$ *signed-take-bit n k* $\longleftrightarrow k <$ $2$ ^ $n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-greater-eq*:
‹$k +$ $2$ ^ *Suc n* $\leq$ *signed-take-bit n k*› **if** ‹$k < -$ ($2$ ^ $n$)›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-int-less-eq*:
‹*signed-take-bit n k* $\leq k -$ $2$ ^ *Suc n*› **if** ‹$k \geq$ $2$ ^ $n$›
**for** $k$ :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-sgn-eq* [*simp*]:
‹*signed-take-bit* (*Suc n*) (*sgn k*) = *sgn k*› **for** *k* :: *int*
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-bit0* [*simp*]:
‹*signed-take-bit* (*Suc n*) (*numeral* (*Num.Bit0 k*)) = *signed-take-bit n* (*numeral k*) * (*2* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-bit1* [*simp*]:
‹*signed-take-bit* (*Suc n*) (*numeral* (*Num.Bit1 k*)) = *signed-take-bit n* (*numeral k*) * *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-minus-bit0* [*simp*]:
‹*signed-take-bit* (*Suc n*) (− *numeral* (*Num.Bit0 k*)) = *signed-take-bit n* (− *numeral k*) * (*2* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-Suc-minus-bit1* [*simp*]:
‹*signed-take-bit* (*Suc n*) (− *numeral* (*Num.Bit1 k*)) = *signed-take-bit n* (− *numeral k* − *1*) * *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-numeral-bit0* [*simp*]:
‹*signed-take-bit* (*numeral l*) (*numeral* (*Num.Bit0 k*)) = *signed-take-bit* (*pred-numeral l*) (*numeral k*) * (*2* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-numeral-bit1* [*simp*]:
‹*signed-take-bit* (*numeral l*) (*numeral* (*Num.Bit1 k*)) = *signed-take-bit* (*pred-numeral l*) (*numeral k*) * *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-numeral-minus-bit0* [*simp*]:
‹*signed-take-bit* (*numeral l*) (− *numeral* (*Num.Bit0 k*)) = *signed-take-bit* (*pred-numeral l*) (− *numeral k*) * (*2* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-numeral-minus-bit1* [*simp*]:
‹*signed-take-bit* (*numeral l*) (− *numeral* (*Num.Bit1 k*)) = *signed-take-bit* (*pred-numeral l*) (− *numeral k* − *1*) * *2* + (*1* :: *int*)›
⟨*proof*⟩

**lemma** *signed-take-bit-code* [*code*]:
‹*signed-take-bit n a* =
(*let l* = *take-bit* (*Suc n*) *a*
*in if bit l n then l* + *push-bit* (*Suc n*) (− *1*) *else l*)›

⟨*proof*⟩

## 68.11 Key ideas of bit operations

When formalizing bit operations, it is tempting to represent bit values as explicit lists over a binary type. This however is a bad idea, mainly due to the inherent ambiguities in representation concerning repeating leading bits.

Hence this approach avoids such explicit lists altogether following an algebraic path:

- Bit values are represented by numeric types: idealized unbounded bit values can be represented by type *int*, bounded bit values by quotient types over *int*.
- (A special case are idealized unbounded bit values ending in *0* which can be represented by type *nat* but only support a restricted set of operations).
- From this idea follows that
  - multiplication by *2* is a bit shift to the left and
  - division by *2* is a bit shift to the right.
- Concerning bounded bit values, iterated shifts to the left may result in eliminating all bits by shifting them all beyond the boundary. The property $2^n \neq 0$ represents that $n$ is *not* beyond that boundary.
- The projection on a single bit is then $bit\ a\ n = odd\ (a\ div\ 2^n)$.
- This leads to the most fundamental properties of bit values:
  - Equality rule: $(\bigwedge n.\ possible\text{-}bit\ TYPE(int)\ n \Longrightarrow bit\ a\ n = bit\ b\ n) \Longrightarrow a = b$
  - Induction rule: $[\![\bigwedge a.\ a\ div\ 2 = a \Longrightarrow P\ a; \bigwedge a\ b.\ [\![P\ a; (of\text{-}bool\ b + 2 * a)\ div\ 2 = a]\!] \Longrightarrow P\ (of\text{-}bool\ b + 2 * a)]\!] \Longrightarrow P\ a$
- Typical operations are characterized as follows:
  - Singleton $n$th bit: $2^n$
  - Bit mask upto bit $n$: $mask\ n = 2^n - 1$
  - Left shift: $push\text{-}bit\ n\ a = a * 2^n$
  - Right shift: $drop\text{-}bit\ n\ a = a\ div\ 2^n$
  - Truncation: $take\text{-}bit\ n\ a = a\ mod\ 2^n$

– Negation: *bit* (*NOT a*) *n* = (*possible-bit TYPE*(*int*) *n* $\wedge$ $\neg$ *bit a n*)
– And: *bit* (*a AND b*) *n* = (*bit a n* $\wedge$ *bit b n*)
– Or: *bit* (*a OR b*) *n* = (*bit a n* $\vee$ *bit b n*)
– Xor: *bit* (*a XOR b*) *n* = (*bit a n* $\neq$ *bit b n*)
– Set a single bit: *set-bit n a* = *a OR push-bit n 1*
– Unset a single bit: *unset-bit n a* = *a AND NOT* (*push-bit n 1*)
– Flip a single bit: *flip-bit n a* = *a XOR push-bit n 1*
– Signed truncation, or modulus centered around *0*: *signed-take-bit n a* = *take-bit n a OR of-bool* (*bit a n*) $*$ *NOT* (*mask n*)
– Bit concatenation: *concat-bit n k l* = *take-bit n k OR push-bit n l*
– (Bounded) conversion from and to a list of bits: *horner-sum of-bool 2* (*map* (*bit a*) [*0..<n*]) = *take-bit n a*

## 68.12 Lemma duplicates and other

**context** *semiring-bits*
**begin**

**lemma** *exp-div-exp-eq*:
‹*2* ^ *m div 2* ^ *n* = *of-bool* (*2* ^ *m* $\neq$ *0* $\wedge$ *m* $\geq$ *n*) $*$ *2* ^ (*m* − *n*)›
⟨*proof*⟩

**lemma** *bits-1-div-2*:
‹*1 div 2* = *0*›
⟨*proof*⟩

**lemma** *bits-1-div-exp*:
‹*1 div 2* ^ *n* = *of-bool* (*n* = *0*)›
⟨*proof*⟩

**lemma** *exp-add-not-zero-imp*:
‹*2* ^ *m* $\neq$ *0*› **and** ‹*2* ^ *n* $\neq$ *0*› **if** ‹*2* ^ (*m* + *n*) $\neq$ *0*›
⟨*proof*⟩

**lemma**
*exp-add-not-zero-imp-left*: ‹*2* ^ *m* $\neq$ *0*›
**and** *exp-add-not-zero-imp-right*: ‹*2* ^ *n* $\neq$ *0*›
**if** ‹*2* ^ (*m* + *n*) $\neq$ *0*›
⟨*proof*⟩

**lemma** *exp-not-zero-imp-exp-diff-not-zero*:
‹*2* ^ (*n* − *m*) $\neq$ *0*› **if** ‹*2* ^ *n* $\neq$ *0*›

⟨*proof*⟩

**lemma** *exp-eq-0-imp-not-bit*:
$\langle \neg$ *bit a n*$\rangle$ **if** $\langle 2 \text{ ^ } n = 0 \rangle$
⟨*proof*⟩

**lemma** *bit-disjunctive-add-iff*:
$\langle$*bit* $(a + b)$ *n* $\longleftrightarrow$ *bit a n* $\vee$ *bit b n*$\rangle$
**if** $\langle \bigwedge n. \neg$ *bit a n* $\vee \neg$ *bit b n*$\rangle$
⟨*proof*⟩

**end**

**context** *semiring-bit-operations*
**begin**

**lemma** *even-mask-div-iff*:
$\langle$*even* $((2 \text{ ^ } m - 1)$ *div* $2 \text{ ^ } n) \longleftrightarrow 2 \text{ ^ } n = 0 \vee m \leq n\rangle$
⟨*proof*⟩

**lemma** *mod-exp-eq*:
$\langle$*a mod* $2 \text{ ^ } m$ *mod* $2 \text{ ^ } n = a$ *mod* $2 \text{ ^ }$ *min m n*$\rangle$
⟨*proof*⟩

**lemma** *mult-exp-mod-exp-eq*:
$\langle m \leq n \Longrightarrow (a * 2 \text{ ^ } m)$ *mod* $(2 \text{ ^ } n) = (a$ *mod* $2 \text{ ^ } (n - m)) * 2 \text{ ^ } m\rangle$
⟨*proof*⟩

**lemma** *div-exp-mod-exp-eq*:
$\langle$*a div* $2 \text{ ^ } n$ *mod* $2 \text{ ^ } m = a$ *mod* $(2 \text{ ^ } (n + m))$ *div* $2 \text{ ^ } n\rangle$
⟨*proof*⟩

**lemma** *even-mult-exp-div-exp-iff*:
$\langle$*even* $(a * 2 \text{ ^ } m$ *div* $2 \text{ ^ } n) \longleftrightarrow m > n \vee 2 \text{ ^ } n = 0 \vee (m \leq n \wedge$ *even* $(a$ *div* $2$
$\text{ ^ } (n - m)))\rangle$
⟨*proof*⟩

**lemma** *mod-exp-div-exp-eq-0*:
$\langle$*a mod* $2 \text{ ^ } n$ *div* $2 \text{ ^ } n = 0\rangle$
⟨*proof*⟩

**lemmas** *bits-one-mod-two-eq-one* = *one-mod-two-eq-one*

**lemmas** *set-bit-def* = *set-bit-eq-or*

**lemmas** *unset-bit-def* = *unset-bit-eq-and-not*

**lemmas** *flip-bit-def* = *flip-bit-eq-xor*

**lemma** *disjunctive-add*:
‹$a + b = a\ OR\ b$› **if** ‹$\bigwedge n.\ \neg\ bit\ a\ n \lor \neg\ bit\ b\ n$›
⟨*proof*⟩

**lemma** *even-mod-exp-div-exp-iff*:
‹$even\ (a\ mod\ 2\ \hat{}\ m\ div\ 2\ \hat{}\ n) \longleftrightarrow m \leq n \lor even\ (a\ div\ 2\ \hat{}\ n)$›
⟨*proof*⟩

**end**

**context** *ring-bit-operations*
**begin**

**lemma** *disjunctive-diff*:
‹$a - b = a\ AND\ NOT\ b$› **if** ‹$\bigwedge n.\ bit\ b\ n \Longrightarrow bit\ a\ n$›
⟨*proof*⟩

**end**

**lemma** *and-nat-rec*:
‹$m\ AND\ n = of\text{-}bool\ (odd\ m \land odd\ n) + 2 * ((m\ div\ 2)\ AND\ (n\ div\ 2))$› **for** $m$ $n :: nat$
⟨*proof*⟩

**lemma** *or-nat-rec*:
‹$m\ OR\ n = of\text{-}bool\ (odd\ m \lor odd\ n) + 2 * ((m\ div\ 2)\ OR\ (n\ div\ 2))$› **for** $m$ $n :: nat$
⟨*proof*⟩

**lemma** *xor-nat-rec*:
‹$m\ XOR\ n = of\text{-}bool\ (odd\ m \neq odd\ n) + 2 * ((m\ div\ 2)\ XOR\ (n\ div\ 2))$› **for** $m$ $n :: nat$
⟨*proof*⟩

**lemma** *bit-push-bit-iff-nat*:
‹$bit\ (push\text{-}bit\ m\ q)\ n \longleftrightarrow m \leq n \land bit\ q\ (n - m)$› **for** $q :: nat$
⟨*proof*⟩

**lemma** *mask-half-int*:
‹$mask\ n\ div\ 2 = (mask\ (n - 1) :: int)$›
⟨*proof*⟩

**lemma** *not-int-rec*:
‹$NOT\ k = of\text{-}bool\ (even\ k) + 2 * NOT\ (k\ div\ 2)$› **for** $k :: int$
⟨*proof*⟩

**lemma** *even-not-iff-int*:
‹$even\ (NOT\ k) \longleftrightarrow odd\ k$› **for** $k :: int$
⟨*proof*⟩

**lemma** *bit-not-int-iff'*:
‹*bit* ($-$ *k* $-$ *1*) *n* $\longleftrightarrow$ $\neg$ *bit k n*› **for** *k* :: *int*
⟨*proof*⟩

**lemmas** *and-int-rec* = *and-int.rec*

**lemma** *even-and-iff-int*:
‹*even* (*k AND l*) $\longleftrightarrow$ *even k* $\vee$ *even l*› **for** *k l* :: *int*
⟨*proof*⟩

**lemmas** *bit-and-int-iff* = *and-int.bit-iff*

**lemmas** *or-int-rec* = *or-int.rec*

**lemmas** *bit-or-int-iff* = *or-int.bit-iff*

**lemmas** *xor-int-rec* = *xor-int.rec*

**lemmas** *bit-xor-int-iff* = *xor-int.bit-iff*

**lemma** *drop-bit-push-bit-int*:
‹*drop-bit m* (*push-bit n k*) = *drop-bit* (*m* $-$ *n*) (*push-bit* (*n* $-$ *m*) *k*)› **for** *k* :: *int*
⟨*proof*⟩

**lemma** *bit-push-bit-iff-int*:
‹*bit* (*push-bit m k*) *n* $\longleftrightarrow$ *m* $\leq$ *n* $\wedge$ *bit k* (*n* $-$ *m*)› **for** *k* :: *int*
⟨*proof*⟩

**bundle** *bit-operations-syntax*
**begin**
**notation**
*not* (‹*NOT*›)
**and** *and* (**infixr** ‹*AND*› *64*)
**and** *or* (**infixr** ‹*OR*› *59*)
**and** *xor* (**infixr** ‹*XOR*› *59*)
**end**

**unbundle** *no bit-operations-syntax*

**end**

# 69 Numeric types for code generation onto target language numerals only

**theory** *Code-Numeral*
**imports** *Lifting Bit-Operations*
**begin**

## 69.1 Type of target language integers

**typedef** *integer* = *UNIV* :: *int set*
**morphisms** *int-of-integer integer-of-int* ⟨*proof*⟩

**setup-lifting** *type-definition-integer*

**lemma** *integer-eq-iff*:
$k = l \longleftrightarrow$ *int-of-integer* $k =$ *int-of-integer* $l$
⟨*proof*⟩

**lemma** *integer-eqI*:
*int-of-integer* $k =$ *int-of-integer* $l \Longrightarrow k = l$
⟨*proof*⟩

**lemma** *int-of-integer-integer-of-int* [*simp*]:
*int-of-integer* (*integer-of-int* $k$) = $k$
⟨*proof*⟩

**lemma** *integer-of-int-int-of-integer* [*simp*]:
*integer-of-int* (*int-of-integer* $k$) = $k$
⟨*proof*⟩

**instantiation** *integer* :: *ring-1*
**begin**

**lift-definition** *zero-integer* :: *integer*
**is** *0* :: *int*
⟨*proof*⟩

**declare** *zero-integer.rep-eq* [*simp*]

**lift-definition** *one-integer* :: *integer*
**is** *1* :: *int*
⟨*proof*⟩

**declare** *one-integer.rep-eq* [*simp*]

**lift-definition** *plus-integer* :: *integer* ⇒ *integer* ⇒ *integer*
**is** *plus* :: *int* ⇒ *int* ⇒ *int*
⟨*proof*⟩

**declare** *plus-integer.rep-eq* [*simp*]

**lift-definition** *uminus-integer* :: *integer* ⇒ *integer*
**is** *uminus* :: *int* ⇒ *int*
⟨*proof*⟩

**declare** *uminus-integer.rep-eq* [*simp*]

**lift-definition** *minus-integer* :: *integer* ⇒ *integer* ⇒ *integer*
**is** *minus* :: *int* ⇒ *int* ⇒ *int*
⟨*proof*⟩

**declare** *minus-integer.rep-eq* [*simp*]

**lift-definition** *times-integer* :: *integer* ⇒ *integer* ⇒ *integer*
**is** *times* :: *int* ⇒ *int* ⇒ *int*
⟨*proof*⟩

**declare** *times-integer.rep-eq* [*simp*]

**instance** ⟨*proof*⟩

**end**

**instance** *integer* :: *Rings.dvd* ⟨*proof*⟩

**context**
**includes** *lifting-syntax*
**notes** *transfer-rule-numeral* [*transfer-rule*]
**begin**

**lemma** [*transfer-rule*]:
(*pcr-integer* ===> *pcr-integer* ===> (⟷)) (*dvd*) (*dvd*)
⟨*proof*⟩

**lemma** [*transfer-rule*]:
((⟷) ===> *pcr-integer*) *of-bool of-bool*
⟨*proof*⟩

**lemma** [*transfer-rule*]:
((=) ===> *pcr-integer*) *int of-nat*
⟨*proof*⟩

**lemma** [*transfer-rule*]:
((=) ===> *pcr-integer*) (λ*k*. *k*) *of-int*
⟨*proof*⟩

**lemma** [*transfer-rule*]:
((=) ===> *pcr-integer*) *numeral numeral*
⟨*proof*⟩

**lemma** [*transfer-rule*]:
((=) ===> (=) ===> *pcr-integer*) *Num.sub Num.sub*
⟨*proof*⟩

**lemma** [*transfer-rule*]:
(*pcr-integer* ===> (=) ===> *pcr-integer*) (^) (^)

⟨*proof*⟩

**end**

**lemma** *int-of-integer-of-nat* [*simp*]:
*int-of-integer* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

**lift-definition** *integer-of-nat* :: *nat* ⇒ *integer*
**is** *of-nat* :: *nat* ⇒ *int*
⟨*proof*⟩

**lemma** *integer-of-nat-eq-of-nat* [*code*]:
*integer-of-nat* = *of-nat*
⟨*proof*⟩

**lemma** *int-of-integer-integer-of-nat* [*simp*]:
*int-of-integer* (*integer-of-nat n*) = *of-nat n*
⟨*proof*⟩

**lift-definition** *nat-of-integer* :: *integer* ⇒ *nat*
**is** *Int.nat*
⟨*proof*⟩

**lemma** *nat-of-integer-0* [*simp*]:
‹*nat-of-integer 0* = *0*›
⟨*proof*⟩

**lemma** *nat-of-integer-1* [*simp*]:
‹*nat-of-integer 1* = *1*›
⟨*proof*⟩

**lemma** *nat-of-integer-numeral* [*simp*]:
‹*nat-of-integer* (*numeral n*) = *numeral n*›
⟨*proof*⟩

**lemma** *nat-of-integer-of-nat* [*simp*]:
*nat-of-integer* (*of-nat n*) = *n*
⟨*proof*⟩

**lemma** *int-of-integer-of-int* [*simp*]:
*int-of-integer* (*of-int k*) = *k*
⟨*proof*⟩

**lemma** *nat-of-integer-integer-of-nat* [*simp*]:
*nat-of-integer* (*integer-of-nat n*) = *n*
⟨*proof*⟩

**lemma** *integer-of-int-eq-of-int* [*simp*, *code-abbrev*]:

*integer-of-int* = *of-int*
⟨*proof*⟩

**lemma** *of-int-integer-of* [*simp*]:
*of-int* (*int-of-integer k*) = (*k* :: *integer*)
⟨*proof*⟩

**lemma** *int-of-integer-numeral* [*simp*]:
*int-of-integer* (*numeral k*) = *numeral k*
⟨*proof*⟩

**lemma** *int-of-integer-sub* [*simp*]:
*int-of-integer* (*Num.sub k l*) = *Num.sub k l*
⟨*proof*⟩

**definition** *integer-of-num* :: *num* ⇒ *integer*
**where** [*simp*]: *integer-of-num* = *numeral*

**lemma** *integer-of-num* [*code*]:
*integer-of-num Num.One* = *1*
*integer-of-num* (*Num.Bit0 n*) = (*let k* = *integer-of-num n in k* + *k*)
*integer-of-num* (*Num.Bit1 n*) = (*let k* = *integer-of-num n in k* + *k* + *1*)
⟨*proof*⟩

**lemma** *integer-of-num-triv*:
*integer-of-num Num.One* = *1*
*integer-of-num* (*Num.Bit0 Num.One*) = *2*
⟨*proof*⟩

**instantiation** *integer* :: *equal*
**begin**

**lift-definition** *equal-integer* :: ‹*integer* ⇒ *integer* ⇒ *bool*›
**is** ‹*HOL.equal* :: *int* ⇒ *int* ⇒ *bool*›
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**instantiation** *integer* :: *linordered-idom*
**begin**

**lift-definition** *abs-integer* :: ‹*integer* ⇒ *integer*›
**is** ‹*abs* :: *int* ⇒ *int*›
⟨*proof*⟩

**declare** *abs-integer.rep-eq* [*simp*]

**lift-definition** *sgn-integer* :: ‹*integer* ⇒ *integer*›
  **is** ‹*sgn* :: *int* ⇒ *int*›
  ⟨*proof*⟩

**declare** *sgn-integer.rep-eq* [*simp*]

**lift-definition** *less-eq-integer* :: ‹*integer* ⇒ *integer* ⇒ *bool*›
  **is** ‹*less-eq* :: *int* ⇒ *int* ⇒ *bool*›
  ⟨*proof*⟩

**lemma** *integer-less-eq-iff*:
  ‹$k \leq l \longleftrightarrow$ *int-of-integer* $k \leq$ *int-of-integer* $l$›
  ⟨*proof*⟩

**lift-definition** *less-integer* :: ‹*integer* ⇒ *integer* ⇒ *bool*›
  **is** ‹*less* :: *int* ⇒ *int* ⇒ *bool*›
  ⟨*proof*⟩

**lemma** *integer-less-iff*:
  ‹$k < l \longleftrightarrow$ *int-of-integer* $k <$ *int-of-integer* $l$›
  ⟨*proof*⟩

**instance**
  ⟨*proof*⟩

**end**

**instance** *integer* :: *discrete-linordered-semidom*
  ⟨*proof*⟩

**context**
  **includes** *lifting-syntax*
**begin**

**lemma** [*transfer-rule*]:
  ‹(*pcr-integer* ===> *pcr-integer* ===> *pcr-integer*) *min min*›
  ⟨*proof*⟩

**lemma** [*transfer-rule*]:
  ‹(*pcr-integer* ===> *pcr-integer* ===> *pcr-integer*) *max max*›
  ⟨*proof*⟩

**end**

**lemma** *int-of-integer-min* [*simp*]:
  *int-of-integer* (*min* $k$ $l$) = *min* (*int-of-integer* $k$) (*int-of-integer* $l$)
  ⟨*proof*⟩

**lemma** *int-of-integer-max* [*simp*]:
*int-of-integer* (*max k l*) = *max* (*int-of-integer k*) (*int-of-integer l*)
⟨*proof*⟩

**lemma** *nat-of-integer-non-positive* [*simp*]:
$k \le 0 \Longrightarrow$ *nat-of-integer k* = *0*
⟨*proof*⟩

**lemma** *of-nat-of-integer* [*simp*]:
*of-nat* (*nat-of-integer k*) = *max 0 k*
⟨*proof*⟩

**instantiation** *integer* :: *unique-euclidean-ring*
**begin**

**lift-definition** *divide-integer* :: *integer* ⇒ *integer* ⇒ *integer*
**is** *divide* :: *int* ⇒ *int* ⇒ *int*
⟨*proof*⟩

**declare** *divide-integer.rep-eq* [*simp*]

**lift-definition** *modulo-integer* :: *integer* ⇒ *integer* ⇒ *integer*
**is** *modulo* :: *int* ⇒ *int* ⇒ *int*
⟨*proof*⟩

**declare** *modulo-integer.rep-eq* [*simp*]

**lift-definition** *euclidean-size-integer* :: *integer* ⇒ *nat*
**is** *euclidean-size* :: *int* ⇒ *nat*
⟨*proof*⟩

**declare** *euclidean-size-integer.rep-eq* [*simp*]

**lift-definition** *division-segment-integer* :: *integer* ⇒ *integer*
**is** *division-segment* :: *int* ⇒ *int*
⟨*proof*⟩

**declare** *division-segment-integer.rep-eq* [*simp*]

**instance**
⟨*proof*⟩

**end**

**lemma** [*code*]:
*euclidean-size* = *nat-of-integer* ∘ *abs*
⟨*proof*⟩

**lemma** [*code*]:

*division-segment* $(k :: integer) = (if\ k \geq 0\ then\ 1\ else\ -\ 1)$
⟨*proof*⟩

**instance** *integer* :: *linordered-euclidean-semiring*
⟨*proof*⟩

**instantiation** *integer* :: *ring-bit-operations*
**begin**

**lift-definition** *bit-integer* :: ‹*integer* ⇒ *nat* ⇒ *bool*›
**is** *bit* ⟨*proof*⟩

**lift-definition** *not-integer* :: ‹*integer* ⇒ *integer*›
**is** *not* ⟨*proof*⟩

**lift-definition** *and-integer* :: ‹*integer* ⇒ *integer* ⇒ *integer*›
**is** ‹*and*› ⟨*proof*⟩

**lift-definition** *or-integer* :: ‹*integer* ⇒ *integer* ⇒ *integer*›
**is** *or* ⟨*proof*⟩

**lift-definition** *xor-integer* :: ‹*integer* ⇒ *integer* ⇒ *integer*›
**is** *xor* ⟨*proof*⟩

**lift-definition** *mask-integer* :: ‹*nat* ⇒ *integer*›
**is** *mask* ⟨*proof*⟩

**lift-definition** *set-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *set-bit* ⟨*proof*⟩

**lift-definition** *unset-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *unset-bit* ⟨*proof*⟩

**lift-definition** *flip-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *flip-bit* ⟨*proof*⟩

**lift-definition** *push-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *push-bit* ⟨*proof*⟩

**lift-definition** *drop-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *drop-bit* ⟨*proof*⟩

**lift-definition** *take-bit-integer* :: ‹*nat* ⇒ *integer* ⇒ *integer*›
**is** *take-bit* ⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instance** *integer* :: *linordered-euclidean-semiring-bit-operations* ⟨*proof*⟩

**instantiation** *integer* :: *linordered-euclidean-semiring-division*
**begin**

**definition** *divmod-integer* :: *num* ⇒ *num* ⇒ *integer* × *integer*
**where**
*divmod-integer′-def*: *divmod-integer m n* = (*numeral m div numeral n*, *numeral m mod numeral n*)

**definition** *divmod-step-integer* :: *integer* ⇒ *integer* × *integer* ⇒ *integer* × *integer*
**where**
*divmod-step-integer l qr* = (*let* (*q*, *r*) = *qr*
*in if* |*l*| ≤ |*r*| *then* (*2* ∗ *q* + *1*, *r* − *l*)
*else* (*2* ∗ *q*, *r*))

**instance** ⟨*proof*⟩

**end**

**lemma** *integer-of-nat-0*: *integer-of-nat 0* = *0*
⟨*proof*⟩

**lemma** *integer-of-nat-1*: *integer-of-nat 1* = *1*
⟨*proof*⟩

**lemma** *integer-of-nat-numeral*:
*integer-of-nat* (*numeral n*) = *numeral n*
⟨*proof*⟩

## 69.2 Code theorems for target language integers

Constructors

**definition** *Pos* :: *num* ⇒ *integer*
**where**
[*simp*, *code-post*]: *Pos* = *numeral*

**context**
**includes** *lifting-syntax*
**begin**

**lemma** [*transfer-rule*]:
‹((=) ===> *pcr-integer*) *numeral Pos*›
⟨*proof*⟩

**end**

**lemma** *Pos-fold* [*code-unfold*]:
*numeral Num.One* = *Pos Num.One*
*numeral* (*Num.Bit0 k*) = *Pos* (*Num.Bit0 k*)
*numeral* (*Num.Bit1 k*) = *Pos* (*Num.Bit1 k*)
⟨*proof*⟩

**definition** *Neg* :: *num* ⇒ *integer*
**where**
[*simp*, *code-abbrev*]: *Neg n* = − *Pos n*

**context**
**includes** *lifting-syntax*
**begin**

**lemma** [*transfer-rule*]:
‹((=) ===> *pcr-integer*) (λ*n*. − *numeral n*) *Neg*›
⟨*proof*⟩

**end**

**code-datatype** *0*::*integer Pos Neg*

A further pair of constructors for generated computations

**context**
**begin**

**qualified definition** *positive* :: *num* ⇒ *integer*
**where** [*simp*]: *positive* = *numeral*

**qualified definition** *negative* :: *num* ⇒ *integer*
**where** [*simp*]: *negative* = *uminus* ∘ *numeral*

**lemma** [*code-computation-unfold*]:
*numeral* = *positive*
*Pos* = *positive*
*Neg* = *negative*
⟨*proof*⟩

**end**

Auxiliary operations

**lift-definition** *dup* :: *integer* ⇒ *integer*
**is** λ*k*::*int*. *k* + *k*
⟨*proof*⟩

**lemma** *dup-code* [*code*]:
*dup 0* = *0*
*dup* (*Pos n*) = *Pos* (*Num.Bit0 n*)
*dup* (*Neg n*) = *Neg* (*Num.Bit0 n*)

⟨*proof*⟩

**lift-definition** *sub* :: *num* ⇒ *num* ⇒ *integer*
**is** $\lambda m\ n.\ numeral\ m - numeral\ n :: int$
⟨*proof*⟩

**lemma** *sub-code* [*code*]:
*sub Num.One Num.One* = *0*
*sub* (*Num.Bit0 m*) *Num.One* = *Pos* (*Num.BitM m*)
*sub* (*Num.Bit1 m*) *Num.One* = *Pos* (*Num.Bit0 m*)
*sub Num.One* (*Num.Bit0 n*) = *Neg* (*Num.BitM n*)
*sub Num.One* (*Num.Bit1 n*) = *Neg* (*Num.Bit0 n*)
*sub* (*Num.Bit0 m*) (*Num.Bit0 n*) = *dup* (*sub m n*)
*sub* (*Num.Bit1 m*) (*Num.Bit1 n*) = *dup* (*sub m n*)
*sub* (*Num.Bit1 m*) (*Num.Bit0 n*) = *dup* (*sub m n*) + *1*
*sub* (*Num.Bit0 m*) (*Num.Bit1 n*) = *dup* (*sub m n*) − *1*
⟨*proof*⟩

Implementations

**lemma** *one-integer-code* [*code*, *code-unfold*]:
*1* = *Pos Num.One*
⟨*proof*⟩

**lemma** *plus-integer-code* [*code*]:
*k* + *0* = (*k*::*integer*)
*0* + *l* = (*l*::*integer*)
*Pos m* + *Pos n* = *Pos* (*m* + *n*)
*Pos m* + *Neg n* = *sub m n*
*Neg m* + *Pos n* = *sub n m*
*Neg m* + *Neg n* = *Neg* (*m* + *n*)
⟨*proof*⟩

**lemma** *uminus-integer-code* [*code*]:
*uminus 0* = (*0*::*integer*)
*uminus* (*Pos m*) = *Neg m*
*uminus* (*Neg m*) = *Pos m*
⟨*proof*⟩

**lemma** *minus-integer-code* [*code*]:
*k* − *0* = (*k*::*integer*)
*0* − *l* = *uminus* (*l*::*integer*)
*Pos m* − *Pos n* = *sub m n*
*Pos m* − *Neg n* = *Pos* (*m* + *n*)
*Neg m* − *Pos n* = *Neg* (*m* + *n*)
*Neg m* − *Neg n* = *sub n m*
⟨*proof*⟩

**lemma** *abs-integer-code* [*code*]:
|*k*| = (*if* (*k*::*integer*) < *0 then* − *k else k*)

⟨*proof*⟩

**lemma** *sgn-integer-code* [*code*]:
$sgn\ k = (if\ k = 0\ then\ 0\ else\ if\ (k::integer) < 0\ then\ -\ 1\ else\ 1)$
⟨*proof*⟩

**lemma** *times-integer-code* [*code*]:
$k * 0 = (0::integer)$
$0 * l = (0::integer)$
$Pos\ m * Pos\ n = Pos\ (m * n)$
$Pos\ m * Neg\ n = Neg\ (m * n)$
$Neg\ m * Pos\ n = Neg\ (m * n)$
$Neg\ m * Neg\ n = Pos\ (m * n)$
⟨*proof*⟩

**definition** *divmod-integer* :: *integer* ⇒ *integer* ⇒ *integer* × *integer*
**where**
$divmod\text{-}integer\ k\ l = (k\ div\ l,\ k\ mod\ l)$

**lemma** *fst-divmod-integer* [*simp*]:
$fst\ (divmod\text{-}integer\ k\ l) = k\ div\ l$
⟨*proof*⟩

**lemma** *snd-divmod-integer* [*simp*]:
$snd\ (divmod\text{-}integer\ k\ l) = k\ mod\ l$
⟨*proof*⟩

**definition** *divmod-abs* :: *integer* ⇒ *integer* ⇒ *integer* × *integer*
**where**
$divmod\text{-}abs\ k\ l = (|k|\ div\ |l|,\ |k|\ mod\ |l|)$

**lemma** *fst-divmod-abs* [*simp*]:
$fst\ (divmod\text{-}abs\ k\ l) = |k|\ div\ |l|$
⟨*proof*⟩

**lemma** *snd-divmod-abs* [*simp*]:
$snd\ (divmod\text{-}abs\ k\ l) = |k|\ mod\ |l|$
⟨*proof*⟩

**declare** *divmod-algorithm-code* [**where** $?'a = integer$,
*folded integer-of-num-def*, *unfolded integer-of-num-triv*,
*code*]

**lemma** *divmod-abs-code* [*code*]:
$divmod\text{-}abs\ 0\ j = (0,\ 0)$
$divmod\text{-}abs\ j\ 0 = (0,\ |j|)$
$divmod\text{-}abs\ (Pos\ k)\ (Pos\ l) = divmod\ k\ l$
$divmod\text{-}abs\ (Pos\ k)\ (Neg\ l) = divmod\ k\ l$
$divmod\text{-}abs\ (Neg\ k)\ (Pos\ l) = divmod\ k\ l$

*divmod-abs* (*Neg k*) (*Neg l*) = *divmod k l*
⟨*proof*⟩

**lemma** *divmod-integer-eq-cases*:
*divmod-integer k l* =
(*if k* = *0 then* (*0*, *0*) *else if l* = *0 then* (*0*, *k*) *else*
(*apsnd* ∘ *times* ∘ *sgn*) *l* (*if sgn k* = *sgn l*
*then divmod-abs k l*
*else* (*let* (*r*, *s*) = *divmod-abs k l in*
*if s* = *0 then* (− *r*, *0*) *else* (− *r* − *1*, |*l*| − *s*))))
⟨*proof*⟩

**lemma** *divmod-integer-code* [*code*]:
*divmod-integer k l* =
(*if k* = *0 then* (*0*, *0*)
*else if l* > *0 then*
(*if k* > *0 then divmod-abs k l*
*else case divmod-abs k l of* (*r*, *s*) ⇒
*if s* = *0 then* (− *r*, *0*) *else* (− *r* − *1*, *l* − *s*))
*else if l* = *0 then* (*0*, *k*)
*else apsnd uminus*
(*if k* < *0 then divmod-abs k l*
*else case divmod-abs k l of* (*r*, *s*) ⇒
*if s* = *0 then* (− *r*, *0*) *else* (− *r* − *1*, − *l* − *s*)))
⟨*proof*⟩

**lemma** *div-integer-code* [*code*]:
*k div l* = *fst* (*divmod-integer k l*)
⟨*proof*⟩

**lemma** *mod-integer-code* [*code*]:
*k mod l* = *snd* (*divmod-integer k l*)
⟨*proof*⟩

**context**
**includes** *bit-operations-syntax*
**begin**

**lemma** *and-integer-code* [*code*]:
‹*0 AND k* = *0*›
‹*k AND 0* = *0*›
‹*Neg Num.One AND k* = *k*›
‹*k AND Neg Num.One* = *k*›
‹*Pos Num.One AND Pos Num.One* = *Pos Num.One*›
‹*Pos Num.One AND Pos* (*Num.Bit0 n*) = *0*›
‹*Pos* (*Num.Bit0 m*) *AND Pos Num.One* = *0*›
‹*Pos Num.One AND Pos* (*Num.Bit1 n*) = *Pos Num.One*›
‹*Pos* (*Num.Bit1 m*) *AND Pos Num.One* = *Pos Num.One*›
‹*Pos* (*Num.Bit0 m*) *AND Pos* (*Num.Bit0 n*) = *dup* (*Pos m AND Pos n*)›

‹Pos (Num.Bit0 m) AND Pos (Num.Bit1 n) = dup (Pos m AND Pos n)›
‹Pos (Num.Bit1 m) AND Pos (Num.Bit0 n) = dup (Pos m AND Pos n)›
‹Pos (Num.Bit1 m) AND Pos (Num.Bit1 n) = Pos Num.One + dup (Pos m AND Pos n)›
‹Pos m AND Neg (num.Bit0 n) = (case and-not-num m (Num.BitM n) of None ⇒ 0 | Some n′ ⇒ Pos n′)›
‹Neg (num.Bit0 m) AND Pos n = (case and-not-num n (Num.BitM m) of None ⇒ 0 | Some n′ ⇒ Pos n′)›
‹Pos m AND Neg (num.Bit1 n) = (case and-not-num m (Num.Bit0 n) of None ⇒ 0 | Some n′ ⇒ Pos n′)›
‹Neg (num.Bit1 m) AND Pos n = (case and-not-num n (Num.Bit0 m) of None ⇒ 0 | Some n′ ⇒ Pos n′)›
‹Neg m AND Neg n = NOT (sub m Num.One OR sub n Num.One)›
**for** *k* :: *integer*
⟨*proof*⟩

**lemma** *or-integer-code* [*code*]:
‹0 OR k = k›
‹k OR 0 = k›
‹Neg Num.One OR k = Neg Num.One›
‹k OR Neg Num.One = Neg Num.One›
‹Pos Num.One OR Pos Num.One = Pos Num.One›
‹Pos Num.One OR Pos (Num.Bit0 n) = Pos (Num.Bit1 n)›
‹Pos (Num.Bit0 m) OR Pos Num.One = Pos (Num.Bit1 m)›
‹Pos Num.One OR Pos (Num.Bit1 n) = Pos (Num.Bit1 n)›
‹Pos (Num.Bit1 m) OR Pos Num.One = Pos (Num.Bit1 m)›
‹Pos (Num.Bit0 m) OR Pos (Num.Bit0 n) = dup (Pos m OR Pos n)›
‹Pos (Num.Bit0 m) OR Pos (Num.Bit1 n) = Pos Num.One + dup (Pos m OR Pos n)›
‹Pos (Num.Bit1 m) OR Pos (Num.Bit0 n) = Pos Num.One + dup (Pos m OR Pos n)›
‹Pos (Num.Bit1 m) OR Pos (Num.Bit1 n) = Pos Num.One + dup (Pos m OR Pos n)›
‹Pos m OR Neg (num.Bit0 n) = Neg (or-not-num-neg m (Num.BitM n))›
‹Neg (num.Bit0 m) OR Pos n = Neg (or-not-num-neg n (Num.BitM m))›
‹Pos m OR Neg (num.Bit1 n) = Neg (or-not-num-neg m (Num.Bit0 n))›
‹Neg (num.Bit1 m) OR Pos n = Neg (or-not-num-neg n (Num.Bit0 m))›
‹Neg m OR Neg n = NOT (sub m Num.One AND sub n Num.One)›
**for** *k* :: *integer*
⟨*proof*⟩

**lemma** *xor-integer-code* [*code*]:
‹0 XOR k = k›
‹k XOR 0 = k›
‹Neg Num.One XOR k = NOT k›
‹k XOR Neg Num.One = NOT k›
‹Neg m XOR k = NOT (sub m num.One XOR k)›
‹k XOR Neg n = NOT (k XOR (sub n num.One))›
‹Pos Num.One XOR Pos Num.One = 0›

‹*Pos Num.One XOR Pos* (*Num.Bit0 n*) = *Pos* (*Num.Bit1 n*)›
‹*Pos* (*Num.Bit0 m*) *XOR Pos Num.One* = *Pos* (*Num.Bit1 m*)›
‹*Pos Num.One XOR Pos* (*Num.Bit1 n*) = *Pos* (*Num.Bit0 n*)›
‹*Pos* (*Num.Bit1 m*) *XOR Pos Num.One* = *Pos* (*Num.Bit0 m*)›
‹*Pos* (*Num.Bit0 m*) *XOR Pos* (*Num.Bit0 n*) = *dup* (*Pos m XOR Pos n*)›
‹*Pos* (*Num.Bit0 m*) *XOR Pos* (*Num.Bit1 n*) = *Pos Num.One* + *dup* (*Pos m XOR Pos n*)›
‹*Pos* (*Num.Bit1 m*) *XOR Pos* (*Num.Bit0 n*) = *Pos Num.One* + *dup* (*Pos m XOR Pos n*)›
‹*Pos* (*Num.Bit1 m*) *XOR Pos* (*Num.Bit1 n*) = *dup* (*Pos m XOR Pos n*)›
**for** *k* :: *integer*
⟨*proof*⟩

**lemma** [*code*]:
‹*NOT k* = − *k* − *1*› **for** *k* :: *integer*
⟨*proof*⟩

**lemma** [*code*]:
‹*bit k n* ⟷ *k AND push-bit n 1* ≠ (*0* :: *integer*)›
⟨*proof*⟩

**lemma** [*code*]:
‹*mask n* = *push-bit n 1* − (*1* :: *integer*)›
⟨*proof*⟩

**lemma** [*code*]:
‹*set-bit n k* = *k OR push-bit n 1*› **for** *k* :: *integer*
⟨*proof*⟩

**lemma** [*code*]:
‹*unset-bit n k* = *k AND NOT* (*push-bit n 1*)› **for** *k* :: *integer*
⟨*proof*⟩

**lemma** [*code*]:
‹*flip-bit n k* = *k XOR push-bit n 1*› **for** *k* :: *integer*
⟨*proof*⟩

**lemma** [*code*]:
‹*take-bit n k* = *k AND mask n*› **for** *k* :: *integer*
⟨*proof*⟩

**end**

**definition** *bit-cut-integer* :: *integer* ⇒ *integer* × *bool*
**where** *bit-cut-integer k* = (*k div 2*, *odd k*)

**lemma** *bit-cut-integer-code* [*code*]:
*bit-cut-integer k* = (*if k* = *0 then* (*0*, *False*)
*else let* (*r*, *s*) = *Code-Numeral.divmod-abs k 2*

*in* (*if* $k > 0$ *then* $r$ *else* $- r - s$, $s = 1$))
⟨*proof*⟩

**lemma** *equal-integer-code* [*code*]:
*HOL.equal* $0$ ($0$::*integer*) ⟷ *True*
*HOL.equal* $0$ (*Pos* $l$) ⟷ *False*
*HOL.equal* $0$ (*Neg* $l$) ⟷ *False*
*HOL.equal* (*Pos* $k$) $0$ ⟷ *False*
*HOL.equal* (*Pos* $k$) (*Pos* $l$) ⟷ *HOL.equal* $k$ $l$
*HOL.equal* (*Pos* $k$) (*Neg* $l$) ⟷ *False*
*HOL.equal* (*Neg* $k$) $0$ ⟷ *False*
*HOL.equal* (*Neg* $k$) (*Pos* $l$) ⟷ *False*
*HOL.equal* (*Neg* $k$) (*Neg* $l$) ⟷ *HOL.equal* $k$ $l$
⟨*proof*⟩

**lemma** *equal-integer-refl* [*code nbe*]:
*HOL.equal* ($k$::*integer*) $k$ ⟷ *True*
⟨*proof*⟩

**lemma** *less-eq-integer-code* [*code*]:
$0 \leq$ ($0$::*integer*) ⟷ *True*
$0 \leq$ *Pos* $l$ ⟷ *True*
$0 \leq$ *Neg* $l$ ⟷ *False*
*Pos* $k \leq 0$ ⟷ *False*
*Pos* $k \leq$ *Pos* $l$ ⟷ $k \leq l$
*Pos* $k \leq$ *Neg* $l$ ⟷ *False*
*Neg* $k \leq 0$ ⟷ *True*
*Neg* $k \leq$ *Pos* $l$ ⟷ *True*
*Neg* $k \leq$ *Neg* $l$ ⟷ $l \leq k$
⟨*proof*⟩

**lemma** *less-integer-code* [*code*]:
$0 <$ ($0$::*integer*) ⟷ *False*
$0 <$ *Pos* $l$ ⟷ *True*
$0 <$ *Neg* $l$ ⟷ *False*
*Pos* $k < 0$ ⟷ *False*
*Pos* $k <$ *Pos* $l$ ⟷ $k < l$
*Pos* $k <$ *Neg* $l$ ⟷ *False*
*Neg* $k < 0$ ⟷ *True*
*Neg* $k <$ *Pos* $l$ ⟷ *True*
*Neg* $k <$ *Neg* $l$ ⟷ $l < k$
⟨*proof*⟩

**lift-definition** *num-of-integer* :: *integer* ⇒ *num*
**is** *num-of-nat* ∘ *nat*
⟨*proof*⟩

**lemma** *num-of-integer-code* [*code*]:
*num-of-integer* $k$ = (*if* $k \leq 1$ *then* *Num.One*

*else let*
*(l, j) = divmod-integer k 2;*
*l′ = num-of-integer l;*
*l′′ = l′ + l′*
*in if j = 0 then l′′ else l′′ + Num.One)*
⟨*proof*⟩

**lemma** *nat-of-integer-code* [*code*]:
*nat-of-integer k = (if k ≤ 0 then 0*
*else let*
*(l, j) = divmod-integer k 2;*
*l′ = nat-of-integer l;*
*l′′ = l′ + l′*
*in if j = 0 then l′′ else l′′ + 1)*
⟨*proof*⟩

**lemma** *int-of-integer-code-nbe* [*code nbe*]:
*int-of-integer 0 = 0*
*int-of-integer (Pos n) = Int.Pos n*
*int-of-integer (Neg n) = Int.Neg n*
⟨*proof*⟩

**lemma** *int-of-integer-code* [*code*]:
‹*int-of-integer k = (*
*if k = 0 then 0*
*else if k = − 1 then − 1*
*else*
*let*
*(l, j) = divmod-integer k 2;*
*l′ = 2 ∗ int-of-integer l*
*in if j = 0 then l′ else l′ + 1)*›
⟨*proof*⟩

**lemma** *integer-of-int-code-nbe* [*code nbe*]:
*integer-of-int 0 = 0*
*integer-of-int (Int.Pos n) = Pos n*
*integer-of-int (Int.Neg n) = Neg n*
⟨*proof*⟩

**lemma** *integer-of-int-code* [*code*]:
‹*integer-of-int k = (*
*if k = 0 then 0*
*else if k = − 1 then − 1*
*else*
*let*
*l = 2 ∗ integer-of-int (k div 2);*
*j = k mod 2*
*in if j = 0 then l else l + 1)*›
⟨*proof*⟩

**hide-const** (**open**) *Pos Neg sub dup divmod-abs*

**context**
**begin**

**qualified definition** *push-bit* :: ‹*integer* ⇒ *integer* ⇒ *integer*›
  **where** ‹*push-bit i k* = *Bit-Operations.push-bit* (*nat-of-integer* |*i*|) *k*›

**qualified lemma** *push-bit-code* [*code*]:
  ‹*push-bit i k* = *k* ∗ *2* ^ *nat-of-integer* |*i*|›
  ⟨*proof*⟩

**lemma** *push-bit-integer-code* [*code*]:
  ‹*Bit-Operations.push-bit n k* = *push-bit* (*of-nat n*) *k*›
  ⟨*proof*⟩ **definition** *drop-bit* :: ‹*integer* ⇒ *integer* ⇒ *integer*›
  **where** ‹*drop-bit i k* = *Bit-Operations.drop-bit* (*nat-of-integer* |*i*|) *k*›

**qualified lemma** *drop-bit-code* [*code*]:
  ‹*drop-bit i k* = *k div 2* ^ *nat-of-integer* |*i*|›
  ⟨*proof*⟩

**lemma** *drop-bit-integer-code* [*code*]:
  ‹*Bit-Operations.drop-bit n k* = *drop-bit* (*of-nat n*) *k*›
  ⟨*proof*⟩

**end**

## 69.3 Serializer setup for target language integers

**code-printing**
  **type-constructor** *integer* ⇀
    (*SML*) *IntInf.int*
    **and** (*OCaml*) *Z.t*
    **and** (*Haskell*) *Integer*
    **and** (*Scala*) *BigInt*
    **and** (*Eval*) *int*
| **class-instance** *integer* :: *equal* ⇀
    (*Haskell*) −

**code-reserved**
  (*Eval*) *int Integer*

**code-printing**
  **constant** *0*::*integer* ⇀
    (*SML*) !(*0*/ :/ *IntInf.int*)
    **and** (*OCaml*) *Z.zero*
    **and** (*Haskell*) !(*0*/ ::/ *Integer*)
    **and** (*Scala*) *BigInt*(*0*)

⟨*ML*⟩

**code-printing**
 **constant** *plus* :: *integer* ⇒ - ⇒ - ⇀
  (*SML*) *IntInf*.+ ((-), (-))
  **and** (*OCaml*) *Z.add*
  **and** (*Haskell*) **infixl** *6* +
  **and** (*Scala*) **infixl** *7* +
  **and** (*Eval*) **infixl** *8* +
| **constant** *uminus* :: *integer* ⇒ - ⇀
  (*SML*) *IntInf*.~
  **and** (*OCaml*) *Z.neg*
  **and** (*Haskell*) *negate*
  **and** (*Scala*) !(− -)
  **and** (*Eval*) ~/ -
| **constant** *minus* :: *integer* ⇒ - ⇀
  (*SML*) *IntInf*.− ((-), (-))
  **and** (*OCaml*) *Z.sub*
  **and** (*Haskell*) **infixl** *6* −
  **and** (*Scala*) **infixl** *7* −
  **and** (*Eval*) **infixl** *8* −
| **constant** *Code-Numeral.dup* ⇀
  (*SML*) *IntInf*.∗/ (*2*,/ (-))
  **and** (*OCaml*) *Z.shift′-left*/ -/ *1*
  **and** (*Haskell*) !(*2* ∗ -)
  **and** (*Scala*) !(*2* ∗ -)
  **and** (*Eval*) !(*2* ∗ -)
| **constant** *Code-Numeral.sub* ⇀
  (*SML*) !(*raise*/ *Fail*/ *sub*)
  **and** (*OCaml*) *failwith*/ *sub*
  **and** (*Haskell*) *error*/ *sub*
  **and** (*Scala*) !*sys.error*(*sub*)
| **constant** *times* :: *integer* ⇒ - ⇒ - ⇀
  (*SML*) *IntInf*.∗ ((-), (-))
  **and** (*OCaml*) *Z.mul*
  **and** (*Haskell*) **infixl** *7* ∗
  **and** (*Scala*) **infixl** *8* ∗
  **and** (*Eval*) **infixl** *9* ∗
| **constant** *Code-Numeral.divmod-abs* ⇀
  (*SML*) *IntInf.divMod*/ (*IntInf.abs* -,/ *IntInf.abs* -)
  **and** (*OCaml*) !(*fun k l* −>/ *if Z.equal Z.zero l then*/ (*Z.zero*, *l*) *else*/ *Z.div′-rem*/ (*Z.abs k*)/ (*Z.abs l*))
  **and** (*Haskell*) *divMod*/ (*abs* -)/ (*abs* -)
  **and** (*Scala*) !((*k*: *BigInt*) => (*l*: *BigInt*) =>/ *l* == *0 match* { *case true* => (*BigInt*(*0*), *k*) *case false* => (*k.abs* ′/% *l.abs*) })
  **and** (*Eval*) *Integer.div′-mod*/ (*abs* -)/ (*abs* -)
| **constant** *HOL.equal* :: *integer* ⇒ - ⇒ *bool* ⇀
  (*SML*) !((- : *IntInf.int*) = -)

**and** (*OCaml*) *Z.equal*
**and** (*Haskell*) **infix** *4* ==
**and** (*Scala*) **infixl** *5* ==
**and** (*Eval*) **infixl** *6* =
| **constant** *less-eq* :: *integer* ⇒ - ⇒ *bool* ⇀
(*SML*) *IntInf*.<= ((-), (-))
**and** (*OCaml*) *Z.leq*
**and** (*Haskell*) **infix** *4* <=
**and** (*Scala*) **infixl** *4* <=
**and** (*Eval*) **infixl** *6* <=
| **constant** *less* :: *integer* ⇒ - ⇒ *bool* ⇀
(*SML*) *IntInf*.< ((-), (-))
**and** (*OCaml*) *Z.lt*
**and** (*Haskell*) **infix** *4* <
**and** (*Scala*) **infixl** *4* <
**and** (*Eval*) **infixl** *6* <
| **constant** *abs* :: *integer* ⇒ - ⇀
(*SML*) *IntInf.abs*
**and** (*OCaml*) *Z.abs*
**and** (*Haskell*) *Prelude.abs*
**and** (*Scala*) *-.abs*
**and** (*Eval*) *abs*
| **constant** *Bit-Operations.and* :: *integer* ⇒ *integer* ⇒ *integer* ⇀
(*SML*) *IntInf.andb* ((-),/ (-))
**and** (*OCaml*) *Z.logand*
**and** (*Haskell*) **infixl** *7* .&.
**and** (*Scala*) **infixl** *3* &
| **constant** *Bit-Operations.or* :: *integer* ⇒ *integer* ⇒ *integer* ⇀
(*SML*) *IntInf.orb* ((-),/ (-))
**and** (*OCaml*) *Z.logor*
**and** (*Haskell*) **infixl** *5* .|.
**and** (*Scala*) **infixl** *1* |
| **constant** *Bit-Operations.xor* :: *integer* ⇒ *integer* ⇒ *integer* ⇀
(*SML*) *IntInf.xorb* ((-),/ (-))
**and** (*OCaml*) *Z.logxor*
**and** (*Haskell*) **infixl** *6* .^.
**and** (*Scala*) **infixl** *2* ^
| **constant** *Bit-Operations.not* :: *integer* ⇒ *integer* ⇀
(*SML*) *IntInf.notb*
**and** (*OCaml*) *Z.lognot*
**and** (*Haskell*) *Data.Bits.complement*
**and** (*Scala*) *-.unary′-~*

**code-reserved**
(*Eval*) *abs*

**code-printing code-module** *Bit-Shifts* ⇀
(*SML*) ‹
*structure Bit-Shifts* : *sig*

*type int = IntInf.int*
*val push : int −> int −> int*
*val drop : int −> int −> int*
*end = struct*

*open IntInf;*

*fun fold - [] y = y*
*| fold f (x :: xs) y = fold f xs (f x y);*

*fun replicate n x = (if n <= 0 then [] else x :: replicate (n − 1) x);*

*val max-index = pow (fromInt 2, Word.wordSize) − fromInt 1; (∗largest possible word∗)*

*val word-of-int = Word.fromLargeInt o toLarge;*

*val word-max-index = word-of-int max-index;*

*fun words-of-int k = case divMod (k, max-index)*
*of (b, s) => word-of-int s :: replicate b word-max-index;*

*fun push′ i k = << (k, i);*

*fun drop′ i k = ~>> (k, i);*

*(∗ The implementations are formally total, though indices >~ max-index will produce heavy computation load ∗)*

*fun push i = fold push′ (words-of-int (abs i));*

*fun drop i = fold drop′ (words-of-int (abs i));*

*end;*› **for constant** *Code-Numeral.push-bit Code-Numeral.drop-bit*
**and** (*OCaml*) ‹
*module Bit-Shifts : sig*
*val push : Z.t −> Z.t −> Z.t*
*val drop : Z.t −> Z.t −> Z.t*
*end = struct*

*let rec fold f xs y = match xs with*
*[] −> y*
*| (x :: xs) −> fold f xs (f x y);;*

*let rec replicate n x = (if Z.leq n Z.zero then [] else x :: replicate (Z.pred n) x);;*

*let max-index = Z.of-int max-int;;*

*let splitIndex i = let (b, s) = Z.div-rem i max-index*

```
  in Z.to-int s :: replicate b max-int;;

let push' i k = Z.shift-left k i;;

let drop' i k = Z.shift-right k i;;

(* The implementations are formally total, though indices >~ max-index will pro-
duce heavy computation load *)

let push i = fold push' (splitIndex (Z.abs i));;

let drop i = fold drop' (splitIndex (Z.abs i));;

end;;
```
› **for constant** *Code-Numeral.push-bit Code-Numeral.drop-bit*
**and** (*Haskell*) ‹

```
module Bit-Shifts (push, drop, push', drop') where

import Prelude (Int, Integer, toInteger, fromInteger, maxBound, divMod, (-), (<=),
abs, flip)
import GHC.Bits (Bits)
import Data.Bits (shiftL, shiftR)

fold :: (a -> b -> b) -> [a] -> b -> b
fold - [] y = y
fold f (x : xs) y = fold f xs (f x y)

replicate :: Integer -> a -> [a]
replicate k x = if k <= 0 then [] else x : replicate (k - 1) x

maxIndex :: Integer
maxIndex = toInteger (maxBound :: Int)

splitIndex :: Integer -> [Int]
splitIndex i = fromInteger s : replicate (fromInteger b) maxBound
  where (b, s) = i `divMod` maxIndex

{- The implementations are formally total, though indices >~ maxIndex will pro-
duce heavy computation load -}

push :: Integer -> Integer -> Integer
push i = fold (flip shiftL) (splitIndex (abs i))

drop :: Integer -> Integer -> Integer
drop i = fold (flip shiftR) (splitIndex (abs i))

push' :: Int -> Int -> Int
push' i = flip shiftL (abs i)
```

*drop′* :: *Int* −> *Int* −> *Int*
*drop′ i* = *flip shiftR* (*abs i*)
› **for constant** *Code-Numeral.push-bit Code-Numeral.drop-bit*
  **and** (*Scala*) ‹
*object Bit-Shifts* {

*private val maxIndex* : *BigInt* = *BigInt*(*Int.MaxValue*);

*private def replicate*[*A*](*i* : *BigInt*, *x* : *A*) : *List*[*A*] =
 *i* <= *0 match* {
  *case true* => *Nil*
  *case false* => *x* :: *replicate*[*A*](*i* − *1*, *x*)
 }

*private def splitIndex*(*i* : *BigInt*) : *List*[*Int*] = {
 *val* (*b*, *s*) = *i* /% *maxIndex*
 *return s.intValue* :: *replicate*(*b*, *Int.MaxValue*)
}

/∗ *The implementations are formally total*, *though indices* >~ *maxIndex will produce heavy computation load* ∗/

*def push*(*i*: *BigInt*, *k*: *BigInt*) : *BigInt* =
 *splitIndex*(*i*).*foldLeft*(*k*) { (*l*, *j*) => *l* << *j* }

*def drop*(*i*: *BigInt*, *k*: *BigInt*) : *BigInt* =
 *splitIndex*(*i*).*foldLeft*(*k*) { (*l*, *j*) => *l* >> *j* }

}
› **for constant** *Code-Numeral.push-bit Code-Numeral.drop-bit*

**code-reserved**
 (*SML*) *Bit-Shifts*
 **and** (*Haskell*) *Bit-Shifts*
 **and** (*Scala*) *Bit-Shifts*

**code-printing**
 **constant** *Code-Numeral.push-bit* ⇀
  (*SML*) *Bit′-Shifts.push*
  **and** (*OCaml*) *Bit′-Shifts.push*
  **and** (*Haskell*) *Bit′-Shifts.push*
  **and** (*Scala*) *Bit′-Shifts.push*
| **constant** *Code-Numeral.drop-bit* ⇀
  (*SML*) *Bit′-Shifts.drop*
  **and** (*OCaml*) *Bit′-Shifts.drop*
  **and** (*Haskell*) *Bit′-Shifts.drop*
  **and** (*Scala*) *Bit′-Shifts.drop*

**code-identifier**

**code-module** *Code-Numeral* $\rightharpoonup$ (*SML*) *Arith* **and** (*OCaml*) *Arith* **and** (*Haskell*) *Arith*

## 69.4 Type of target language naturals

**typedef** *natural* = *UNIV* :: *nat set*
**morphisms** *nat-of-natural natural-of-nat* ⟨*proof*⟩

**setup-lifting** *type-definition-natural*

**lemma** *natural-eq-iff* [*termination-simp*]:
*m* = *n* ⟷ *nat-of-natural m* = *nat-of-natural n*
⟨*proof*⟩

**lemma** *natural-eqI*:
*nat-of-natural m* = *nat-of-natural n* ⟹ *m* = *n*
⟨*proof*⟩

**lemma** *nat-of-natural-of-nat-inverse* [*simp*]:
*nat-of-natural* (*natural-of-nat n*) = *n*
⟨*proof*⟩

**lemma** *natural-of-nat-of-natural-inverse* [*simp*]:
*natural-of-nat* (*nat-of-natural n*) = *n*
⟨*proof*⟩

**instantiation** *natural* :: {*comm-monoid-diff*, *semiring-1*}
**begin**

**lift-definition** *zero-natural* :: *natural*
**is** *0* :: *nat*
⟨*proof*⟩

**declare** *zero-natural.rep-eq* [*simp*]

**lift-definition** *one-natural* :: *natural*
**is** *1* :: *nat*
⟨*proof*⟩

**declare** *one-natural.rep-eq* [*simp*]

**lift-definition** *plus-natural* :: *natural* ⇒ *natural* ⇒ *natural*
**is** *plus* :: *nat* ⇒ *nat* ⇒ *nat*
⟨*proof*⟩

**declare** *plus-natural.rep-eq* [*simp*]

**lift-definition** *minus-natural* :: *natural* ⇒ *natural* ⇒ *natural*
**is** *minus* :: *nat* ⇒ *nat* ⇒ *nat*

⟨proof⟩

**declare** *minus-natural.rep-eq* [*simp*]

**lift-definition** *times-natural* :: *natural* ⇒ *natural* ⇒ *natural*
**is** *times* :: *nat* ⇒ *nat* ⇒ *nat*
⟨proof⟩

**declare** *times-natural.rep-eq* [*simp*]

**instance** ⟨proof⟩

**end**

**instance** *natural* :: *Rings.dvd* ⟨proof⟩

**context**
**includes** *lifting-syntax*
**begin**

**lemma** [*transfer-rule*]:
‹(*pcr-natural* ===> *pcr-natural* ===> (⟷)) (*dvd*) (*dvd*)›
⟨proof⟩

**lemma** [*transfer-rule*]:
‹((⟷) ===> *pcr-natural*) *of-bool of-bool*›
⟨proof⟩

**lemma** [*transfer-rule*]:
‹((=) ===> *pcr-natural*) (λ*n*. *n*) *of-nat*›
⟨proof⟩

**lemma** [*transfer-rule*]:
‹((=) ===> *pcr-natural*) *numeral numeral*›
⟨proof⟩

**lemma** [*transfer-rule*]:
‹(*pcr-natural* ===> (=) ===> *pcr-natural*) (^) (^)›
⟨proof⟩

**end**

**lemma** *nat-of-natural-of-nat* [*simp*]:
*nat-of-natural* (*of-nat n*) = *n*
⟨proof⟩

**lemma** *natural-of-nat-of-nat* [*simp*, *code-abbrev*]:
*natural-of-nat* = *of-nat*
⟨proof⟩

**lemma** *of-nat-of-natural* [*simp*]:
*of-nat* (*nat-of-natural n*) = *n*
⟨*proof*⟩

**lemma** *nat-of-natural-numeral* [*simp*]:
*nat-of-natural* (*numeral k*) = *numeral k*
⟨*proof*⟩

**instantiation** *natural* :: {*linordered-semiring*, *equal*}
**begin**

**lift-definition** *less-eq-natural* :: *natural* ⇒ *natural* ⇒ *bool*
**is** *less-eq* :: *nat* ⇒ *nat* ⇒ *bool*
⟨*proof*⟩

**declare** *less-eq-natural.rep-eq* [*termination-simp*]

**lift-definition** *less-natural* :: *natural* ⇒ *natural* ⇒ *bool*
**is** *less* :: *nat* ⇒ *nat* ⇒ *bool*
⟨*proof*⟩

**declare** *less-natural.rep-eq* [*termination-simp*]

**lift-definition** *equal-natural* :: *natural* ⇒ *natural* ⇒ *bool*
**is** *HOL.equal* :: *nat* ⇒ *nat* ⇒ *bool*
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**context**
**includes** *lifting-syntax*
**begin**

**lemma** [*transfer-rule*]:
‹(*pcr-natural* ===> *pcr-natural* ===> *pcr-natural*) *min min*›
⟨*proof*⟩

**lemma** [*transfer-rule*]:
‹(*pcr-natural* ===> *pcr-natural* ===> *pcr-natural*) *max max*›
⟨*proof*⟩

**end**

**lemma** *nat-of-natural-min* [*simp*]:
*nat-of-natural* (*min k l*) = *min* (*nat-of-natural k*) (*nat-of-natural l*)
⟨*proof*⟩

**lemma** *nat-of-natural-max* [*simp*]:
*nat-of-natural* (*max k l*) = *max* (*nat-of-natural k*) (*nat-of-natural l*)
⟨*proof*⟩

**instantiation** *natural* :: *unique-euclidean-semiring*
**begin**

**lift-definition** *divide-natural* :: *natural* ⇒ *natural* ⇒ *natural*
**is** *divide* :: *nat* ⇒ *nat* ⇒ *nat*
⟨*proof*⟩

**declare** *divide-natural.rep-eq* [*simp*]

**lift-definition** *modulo-natural* :: *natural* ⇒ *natural* ⇒ *natural*
**is** *modulo* :: *nat* ⇒ *nat* ⇒ *nat*
⟨*proof*⟩

**declare** *modulo-natural.rep-eq* [*simp*]

**lift-definition** *euclidean-size-natural* :: *natural* ⇒ *nat*
**is** *euclidean-size* :: *nat* ⇒ *nat*
⟨*proof*⟩

**declare** *euclidean-size-natural.rep-eq* [*simp*]

**lift-definition** *division-segment-natural* :: *natural* ⇒ *natural*
**is** *division-segment* :: *nat* ⇒ *nat*
⟨*proof*⟩

**declare** *division-segment-natural.rep-eq* [*simp*]

**instance**
⟨*proof*⟩

**end**

**lemma** [*code*]:
*euclidean-size* = *nat-of-natural*
⟨*proof*⟩

**lemma** [*code*]:
*division-segment* (*n*::*natural*) = *1*
⟨*proof*⟩

**instance** *natural* :: *discrete-linordered-semidom*
⟨*proof*⟩

**instance** *natural* :: *linordered-euclidean-semiring*

⟨*proof*⟩

**instantiation** *natural* :: *semiring-bit-operations*
**begin**

**lift-definition** *bit-natural* :: ‹*natural* ⇒ *nat* ⇒ *bool*›
 **is** *bit* ⟨*proof*⟩

**lift-definition** *and-natural* :: ‹*natural* ⇒ *natural* ⇒ *natural*›
 **is** ‹*and*› ⟨*proof*⟩

**lift-definition** *or-natural* :: ‹*natural* ⇒ *natural* ⇒ *natural*›
 **is** *or* ⟨*proof*⟩

**lift-definition** *xor-natural* :: ‹*natural* ⇒ *natural* ⇒ *natural*›
 **is** *xor* ⟨*proof*⟩

**lift-definition** *mask-natural* :: ‹*nat* ⇒ *natural*›
 **is** *mask* ⟨*proof*⟩

**lift-definition** *set-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *set-bit* ⟨*proof*⟩

**lift-definition** *unset-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *unset-bit* ⟨*proof*⟩

**lift-definition** *flip-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *flip-bit* ⟨*proof*⟩

**lift-definition** *push-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *push-bit* ⟨*proof*⟩

**lift-definition** *drop-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *drop-bit* ⟨*proof*⟩

**lift-definition** *take-bit-natural* :: ‹*nat* ⇒ *natural* ⇒ *natural*›
 **is** *take-bit* ⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instance** *natural* :: *linordered-euclidean-semiring-bit-operations* ⟨*proof*⟩

**lift-definition** *natural-of-integer* :: *integer* ⇒ *natural*
 **is** *nat* :: *int* ⇒ *nat*
 ⟨*proof*⟩

**lift-definition** *integer-of-natural* :: *natural* ⇒ *integer*

**is** *of-nat* :: *nat* ⇒ *int*
⟨*proof*⟩

**lemma** *natural-of-integer-of-natural* [*simp*]:
*natural-of-integer* (*integer-of-natural n*) = *n*
⟨*proof*⟩

**lemma** *integer-of-natural-of-integer* [*simp*]:
*integer-of-natural* (*natural-of-integer k*) = *max 0 k*
⟨*proof*⟩

**lemma** *int-of-integer-of-natural* [*simp*]:
*int-of-integer* (*integer-of-natural n*) = *of-nat* (*nat-of-natural n*)
⟨*proof*⟩

**lemma** *integer-of-natural-of-nat* [*simp*]:
*integer-of-natural* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

**lemma** [*measure-function*]:
*is-measure nat-of-natural*
⟨*proof*⟩

## 69.5 Inductive representation of target language naturals

**lift-definition** *Suc* :: *natural* ⇒ *natural*
**is** *Nat.Suc*
⟨*proof*⟩

**declare** *Suc.rep-eq* [*simp*]

**old-rep-datatype** *0*::*natural Suc*
⟨*proof*⟩

**lemma** *natural-cases* [*case-names nat*, *cases type*: *natural*]:
**fixes** *m* :: *natural*
**assumes** $\bigwedge n.\ m = \textit{of-nat}\ n \Longrightarrow P$
**shows** *P*
⟨*proof*⟩

**instantiation** *natural* :: *size*
**begin**

**definition** *size-nat* **where** [*simp*, *code*]: *size-nat* = *nat-of-natural*

**instance** ⟨*proof*⟩

**end**

**lemma** *natural-decr* [*termination-simp*]:
$n \neq 0 \Longrightarrow$ *nat-of-natural* $n -$ *Nat.Suc* $0 <$ *nat-of-natural* $n$
⟨*proof*⟩

**lemma** *natural-zero-minus-one*: $(0::natural) - 1 = 0$
⟨*proof*⟩

**lemma** *Suc-natural-minus-one*: *Suc* $n - 1 = n$
⟨*proof*⟩

**hide-const** (**open**) *Suc*

## 69.6 Code refinement for target language naturals

**lift-definition** *Nat* :: *integer* ⇒ *natural*
**is** *nat*
⟨*proof*⟩

**lemma** [*code-post*]:
*Nat* $0 = 0$
*Nat* $1 = 1$
*Nat* (*numeral* $k$) = *numeral* $k$
⟨*proof*⟩

**lemma** [*code abstype*]:
*Nat* (*integer-of-natural* $n$) = $n$
⟨*proof*⟩

**lemma** [*code*]:
*natural-of-nat* $n$ = *natural-of-integer* (*integer-of-nat* $n$)
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*natural-of-integer* $k$) = *max* $0$ $k$
⟨*proof*⟩

**lemma** [*code*]:
‹*integer-of-natural* (*mask* $n$) = *mask* $n$›
⟨*proof*⟩

**lemma** [*code-abbrev*]:
*natural-of-integer* (*Code-Numeral.Pos* $k$) = *numeral* $k$
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* $0 = 0$
⟨*proof*⟩

**lemma** [*code abstract*]:

*integer-of-natural 1 = 1*
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*Code-Numeral.Suc n*) = *integer-of-natural n* + *1*
⟨*proof*⟩

**lemma** [*code*]:
*nat-of-natural* = *nat-of-integer* ∘ *integer-of-natural*
⟨*proof*⟩

**lemma** [*code*, *code-unfold*]:
*case-natural f g n* = (*if n* = *0 then f else g* (*n* − *1*))
⟨*proof*⟩

**declare** *natural.rec* [*code del*]

**lemma** [*code abstract*]:
*integer-of-natural* (*m* + *n*) = *integer-of-natural m* + *integer-of-natural n*
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*m* − *n*) = *max 0* (*integer-of-natural m* − *integer-of-natural n*)
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*m* ∗ *n*) = *integer-of-natural m* ∗ *integer-of-natural n*
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*m div n*) = *integer-of-natural m div integer-of-natural n*
⟨*proof*⟩

**lemma** [*code abstract*]:
*integer-of-natural* (*m mod n*) = *integer-of-natural m mod integer-of-natural n*
⟨*proof*⟩

**lemma** [*code nbe*]: *HOL.equal n* (*n*::*natural*) ⟷ *True*
⟨*proof*⟩

**lemma** [*code*]:
*HOL.equal m n* ⟷ *HOL.equal* (*integer-of-natural m*) (*integer-of-natural n*)
⟨*proof*⟩

**lemma** [*code*]: *m* ≤ *n* ⟷ *integer-of-natural m* ≤ *integer-of-natural n*
⟨*proof*⟩

**lemma** [*code*]: *m* < *n* ⟷ *integer-of-natural m* < *integer-of-natural n*

⟨proof⟩

**context**
**includes** *bit-operations-syntax*
**begin**

**lemma** [*code*]:
‹*bit m n* ⟷ *bit* (*integer-of-natural m*) *n*›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*m AND n*) = *integer-of-natural m AND integer-of-natural n*›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*m OR n*) = *integer-of-natural m OR integer-of-natural n*›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*m XOR n*) = *integer-of-natural m XOR integer-of-natural n*›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*mask n*) = *mask n*›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*set-bit n m*) = *set-bit n* (*integer-of-natural m*)›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*unset-bit n m*) = *unset-bit n* (*integer-of-natural m*)›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*flip-bit n m*) = *flip-bit n* (*integer-of-natural m*)›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*push-bit n m*) = *push-bit n* (*integer-of-natural m*)›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*drop-bit n m*) = *drop-bit n* (*integer-of-natural m*)›
⟨proof⟩

**lemma** [*code abstract*]:
‹*integer-of-natural* (*take-bit n m*) = *take-bit n* (*integer-of-natural m*)›
⟨proof⟩

**end**

**hide-const** (**open**) *Nat*

**code-reflect** *Code-Numeral*
  **datatypes** *natural*
  **functions** *Code-Numeral.Suc 0* :: *natural 1* :: *natural*
    *plus* :: *natural* ⇒ - *minus* :: *natural* ⇒ -
    *times* :: *natural* ⇒ - *divide* :: *natural* ⇒ -
    *modulo* :: *natural* ⇒ -
    *integer-of-natural natural-of-integer*

**lifting-update** *integer.lifting*
**lifting-forget** *integer.lifting*

**lifting-update** *natural.lifting*
**lifting-forget** *natural.lifting*

**end**

# 70 A HOL random engine

**theory** *Random*
**imports** *List Groups-List Code-Numeral*
**begin**

## 70.1 Auxiliary functions

**fun** *log* :: *natural* ⇒ *natural* ⇒ *natural* **where**
  $log\ b\ i = (if\ b \leq 1 \vee i < b\ then\ 1\ else\ 1 + log\ b\ (i\ div\ b))$

**definition** *inc-shift* :: *natural* ⇒ *natural* ⇒ *natural* **where**
  $inc\text{-}shift\ v\ k = (if\ v = k\ then\ 1\ else\ k + 1)$

**definition** *minus-shift* :: *natural* ⇒ *natural* ⇒ *natural* ⇒ *natural* **where**
  $minus\text{-}shift\ r\ k\ l = (if\ k < l\ then\ r + k - l\ else\ k - l)$

## 70.2 Random seeds

**type-synonym** $seed = natural \times natural$

**primrec** *next* :: *seed* ⇒ *natural* × *seed* **where**
  $next\ (v, w) = (let$
    $k = \ v\ div\ 53668;$
    $v' = minus\text{-}shift\ 2147483563\ ((v\ mod\ 53668) * 40014)\ (k * 12211);$
    $l = \ w\ div\ 52774;$
    $w' = minus\text{-}shift\ 2147483399\ ((w\ mod\ 52774) * 40692)\ (l * 3791);$
    $z = \ minus\text{-}shift\ 2147483562\ v'\ (w' + 1) + 1$

*in* $(z, (v', w'))$)

**definition** *split-seed* :: *seed* ⇒ *seed* × *seed* **where**
*split-seed s* = (*let*
$(v, w) = s$;
$(v', w') = snd\ (next\ s)$;
$v'' = inc\text{-}shift\ 2147483562\ v$;
$w'' = inc\text{-}shift\ 2147483398\ w$
*in* $((v'', w'), (v', w''))$)

## 70.3 Base selectors

**context**
**includes** *state-combinator-syntax*
**begin**

**fun** *iterate* :: *natural* ⇒ ('b ⇒ 'a ⇒ 'b × 'a) ⇒ 'b ⇒ 'a ⇒ 'b × 'a **where**
*iterate k f x* = (*if k* = *0 then Pair x else f x* ∘→ *iterate* (*k* − *1*) *f*)

**definition** *range* :: *natural* ⇒ *seed* ⇒ *natural* × *seed* **where**
*range k* = *iterate* (*log 2147483561 k*)
(λ*l*. *next* ∘→ (λ*v*. *Pair* (*v* + *l* ∗ *2147483561*))) *1*
∘→ (λ*v*. *Pair* (*v mod k*))

**lemma** *range*:
*k* > *0* ⟹ *fst* (*range k s*) < *k*
⟨*proof*⟩

**definition** *select* :: 'a *list* ⇒ *seed* ⇒ 'a × *seed* **where**
*select xs* = *range* (*natural-of-nat* (*length xs*))
∘→ (λ*k*. *Pair* (*nth xs* (*nat-of-natural k*)))

**lemma** *select*:
**assumes** *xs* ≠ []
**shows** *fst* (*select xs s*) ∈ *set xs*
⟨*proof*⟩

**primrec** *pick* :: (*natural* × 'a) *list* ⇒ *natural* ⇒ 'a **where**
*pick* (*x* # *xs*) *i* = (*if i* < *fst x then snd x else pick xs* (*i* − *fst x*))

**lemma** *pick-member*:
*i* < *sum-list* (*map fst xs*) ⟹ *pick xs i* ∈ *set* (*map snd xs*)
⟨*proof*⟩

**lemma** *pick-drop-zero*:
*pick* (*filter* (λ(*k*, -). *k* > *0*) *xs*) = *pick xs*
⟨*proof*⟩

**lemma** *pick-same*:

$l < length\ xs \Longrightarrow Random.pick\ (map\ (Pair\ 1)\ xs)\ (natural\text{-}of\text{-}nat\ l) = nth\ xs\ l$
⟨*proof*⟩

**definition** *select-weight* :: (*natural* × '*a*) *list* ⇒ *seed* ⇒ '*a* × *seed* **where**
*select-weight xs* = *range* (*sum-list* (*map fst xs*))
∘→ (λ*k*. *Pair* (*pick xs k*))

**lemma** *select-weight-member*:
**assumes** $0 <$ *sum-list* (*map fst xs*)
**shows** *fst* (*select-weight xs s*) ∈ *set* (*map snd xs*)
⟨*proof*⟩

**lemma** *select-weight-cons-zero*:
*select-weight* ((*0*, *x*) # *xs*) = *select-weight xs*
⟨*proof*⟩

**lemma** *select-weight-drop-zero*:
*select-weight* (*filter* (λ(*k*, -). *k* > *0*) *xs*) = *select-weight xs*
⟨*proof*⟩

**lemma** *select-weight-select*:
**assumes** *xs* ≠ []
**shows** *select-weight* (*map* (*Pair 1*) *xs*) = *select xs*
⟨*proof*⟩

**end**

## 70.4 *ML* interface

**code-reflect** *Random-Engine*
**functions** *range select select-weight*

⟨*ML*⟩

**hide-type** (**open**) *seed*
**hide-const** (**open**) *inc-shift minus-shift log next split-seed*
*iterate range select pick select-weight*
**hide-fact** (**open**) *range-def*

**end**

# 71 Maps

**theory** *Map*
**imports** *List*
**abbrevs** (= = $\subseteq_m$
**begin**

**type-synonym** ('*a*, '*b*) *map* = '*a* ⇒ '*b* *option* (**infixr** ‹⇀› *0*)

**abbreviation** (*input*)
$empty :: 'a \rightharpoonup 'b$ **where**
$empty \equiv \lambda x.\ None$

**definition**
$map\text{-}comp :: ('b \rightharpoonup 'c) \Rightarrow ('a \rightharpoonup 'b) \Rightarrow ('a \rightharpoonup 'c)$ (**infixl** ‹$\circ_m$› *55*) **where**
$f \circ_m g = (\lambda k.\ case\ g\ k\ of\ None \Rightarrow None \mid Some\ v \Rightarrow f\ v)$

**definition**
$map\text{-}add :: ('a \rightharpoonup 'b) \Rightarrow ('a \rightharpoonup 'b) \Rightarrow ('a \rightharpoonup 'b)$ (**infixl** ‹++› *100*) **where**
$m1 ++ m2 = (\lambda x.\ case\ m2\ x\ of\ None \Rightarrow m1\ x \mid Some\ y \Rightarrow Some\ y)$

**definition**
$restrict\text{-}map :: ('a \rightharpoonup 'b) \Rightarrow 'a\ set \Rightarrow ('a \rightharpoonup 'b)$ (**infixl** ‹|‘› *110*) **where**
$m|`A = (\lambda x.\ if\ x \in A\ then\ m\ x\ else\ None)$

**notation** (*latex* **output**)
*restrict-map* (‹-⌈-› [*111*,*110*] *110*)

**definition**
$dom :: ('a \rightharpoonup 'b) \Rightarrow 'a\ set$ **where**
$dom\ m = \{a.\ m\ a \neq None\}$

**definition**
$ran :: ('a \rightharpoonup 'b) \Rightarrow 'b\ set$ **where**
$ran\ m = \{b.\ \exists\, a.\ m\ a = Some\ b\}$

**definition**
$graph :: ('a \rightharpoonup 'b) \Rightarrow ('a \times 'b)\ set$ **where**
$graph\ m = \{(a,\ b) \mid a\ b.\ m\ a = Some\ b\}$

**definition**
$map\text{-}le :: ('a \rightharpoonup 'b) \Rightarrow ('a \rightharpoonup 'b) \Rightarrow bool$ (**infix** ‹$\subseteq_m$› *50*) **where**
$(m_1 \subseteq_m m_2) \longleftrightarrow (\forall\, a \in dom\ m_1.\ m_1\ a = m_2\ a)$

Function update syntax $f(x := y, \ldots)$ is extended with $x \mapsto y$, which is short for $x := Some\ y$. $:=$ and $\mapsto$ can be mixed freely. The syntax $[x \mapsto y, \ldots]$ is short for $Map.empty(x \mapsto y, \ldots)$ but must only contain $\mapsto$, not $:=$, because $[x{:=}y]$ clashes with the list update syntax $xs[i{:=}x]$.

**nonterminal** *maplet* **and** *maplets*

**open-bundle** *maplet-syntax*
**begin**

**syntax**
*-maplet* :: $['a, 'a] \Rightarrow maplet$ (‹(‹*open-block notation*=‹*mixfix maplet*››- /↦/ -)›)
:: $maplet \Rightarrow updbind$ (‹-›)
:: $maplet \Rightarrow maplets$ (‹-›)

*-Maplets* :: [*maplet*, *maplets*] ⇒ *maplets* (‹-,/ -›)
*-Map* :: *maplets* ⇒ *'a* ⇀ *'b* (‹(‹*indent=1 notation=*‹*mixfix map*››[-])›)

**syntax** (*ASCII*)
*-maplet* :: [*'a*, *'a*] ⇒ *maplet* (‹(‹*open-block notation=*‹*mixfix maplet*››- /|−>/ -)›)

**syntax-consts**
*-maplet -Maplets -Map* ⇌ *fun-upd*

**translations**
*-Update f* (*-maplet x y*) ⇌ *f*(*x* := *CONST Some y*)
*-Maplets m ms* ⇀ *-updbinds m ms*
*-Map ms* ⇀ *-Update* (*CONST empty*) *ms*

*-Map* (*-maplet x y*) ↽ *-Update* (λ*u*. *CONST None*) (*-maplet x y*)
*-Map* (*-updbinds m* (*-maplet x y*)) ↽ *-Update* (*-Map m*) (*-maplet x y*)

**end**

Updating with lists:

**primrec** *map-of* :: (*'a* × *'b*) *list* ⇒ *'a* ⇀ *'b* **where**
*map-of* [] = *empty*
| *map-of* (*p* # *ps*) = (*map-of ps*)(*fst p* ↦ *snd p*)

**lemma** *map-of-Cons-code* [*code*]:
*map-of* [] *k* = *None*
*map-of* ((*l*, *v*) # *ps*) *k* = (*if l* = *k then Some v else map-of ps k*)
⟨*proof*⟩

**definition** *map-upds* :: (*'a* ⇀ *'b*) ⇒ *'a list* ⇒ *'b list* ⇒ *'a* ⇀ *'b* **where**
*map-upds m xs ys* = *m* ++ *map-of* (*rev* (*zip xs ys*))

There is also the more specialized update syntax *xs* [↦] *ys* for lists *xs* and *ys*.

**open-bundle** *list-maplet-syntax*
**begin**

**syntax**
*-maplets* :: [*'a*, *'a*] ⇒ *maplet* (‹(‹*open-block notation=*‹*mixfix maplet*››- /[↦]/ -)›)

**syntax** (*ASCII*)
*-maplets* :: [*'a*, *'a*] ⇒ *maplet* (‹(‹*open-block notation=*‹*mixfix maplet*››- /[|−>]/ -)›)

**syntax-consts**

*-maplets* $\rightleftharpoons$ *map-upds*

**translations**
*-Update m (-maplets xs ys)* $\rightleftharpoons$ *CONST map-upds m xs ys*

*-Map (-maplets xs ys)* $\leftharpoondown$ *-Update (λu. CONST None) (-maplets xs ys)*
*-Map (-updbinds m (-maplets xs ys))* $\leftharpoondown$ *-Update (-Map m) (-maplets xs ys)*

**end**

## 71.1 *empty*

**lemma** *empty-upd-none* [*simp*]: *empty*($x$ := *None*) = *empty*
⟨*proof*⟩

## 71.2 *map-upd*

**lemma** *map-upd-triv*: $t\ k$ = *Some* $x \Longrightarrow t(k \mapsto x) = t$
⟨*proof*⟩

**lemma** *map-upd-nonempty* [*simp*]: $t(k \mapsto x) \neq$ *empty*
⟨*proof*⟩

**lemma** *map-upd-eqD1*:
**assumes** $m(a \mapsto x) = n(a \mapsto y)$
**shows** $x = y$
⟨*proof*⟩

**lemma** *map-upd-Some-unfold*:
$((m(a \mapsto b))\ x =$ *Some* $y) = (x = a \land b = y \lor x \neq a \land m\ x =$ *Some* $y)$
⟨*proof*⟩

**lemma** *image-map-upd* [*simp*]: $x \notin A \Longrightarrow m(x \mapsto y)$ ` $A = m$ ` $A$
⟨*proof*⟩

**lemma** *finite-range-updI*:
**assumes** *finite* (*range* $f$) **shows** *finite* (*range* $(f(a \mapsto b))$)
⟨*proof*⟩

## 71.3 *map-of*

**lemma** *map-of-eq-empty-iff* [*simp*]:
*map-of xys* = *empty* $\longleftrightarrow$ *xys* = []
⟨*proof*⟩

**lemma** *empty-eq-map-of-iff* [*simp*]:
*empty* = *map-of xys* $\longleftrightarrow$ *xys* = []
⟨*proof*⟩

**lemma** *map-of-eq-None-iff*:

$(map\text{-}of\ xys\ x = None) = (x \notin fst\ `\ (set\ xys))$
$\langle proof \rangle$

**lemma** *map-of-eq-Some-iff* [*simp*]:
$distinct(map\ fst\ xys) \Longrightarrow (map\text{-}of\ xys\ x = Some\ y) = ((x,y) \in set\ xys)$
$\langle proof \rangle$

**lemma** *Some-eq-map-of-iff* [*simp*]:
$distinct(map\ fst\ xys) \Longrightarrow (Some\ y = map\text{-}of\ xys\ x) = ((x,y) \in set\ xys)$
$\langle proof \rangle$

**lemma** *map-of-is-SomeI* [*simp*]:
$[\![distinct(map\ fst\ xys);\ (x,y) \in set\ xys]\!] \Longrightarrow map\text{-}of\ xys\ x = Some\ y$
$\langle proof \rangle$

**lemma** *map-of-zip-is-None* [*simp*]:
$length\ xs = length\ ys \Longrightarrow (map\text{-}of\ (zip\ xs\ ys)\ x = None) = (x \notin set\ xs)$
$\langle proof \rangle$

**lemma** *map-of-zip-is-Some*:
**assumes** $length\ xs = length\ ys$
**shows** $x \in set\ xs \longleftrightarrow (\exists\,y.\ map\text{-}of\ (zip\ xs\ ys)\ x = Some\ y)$
$\langle proof \rangle$

**lemma** *map-of-zip-upd*:
**fixes** $x :: 'a$ **and** $xs :: 'a\ list$ **and** $ys\ zs :: 'b\ list$
**assumes** $length\ ys = length\ xs$
**and** $length\ zs = length\ xs$
**and** $x \notin set\ xs$
**and** $(map\text{-}of\ (zip\ xs\ ys))(x \mapsto y) = (map\text{-}of\ (zip\ xs\ zs))(x \mapsto z)$
**shows** $map\text{-}of\ (zip\ xs\ ys) = map\text{-}of\ (zip\ xs\ zs)$
$\langle proof \rangle$

**lemma** *map-of-zip-inject*:
**assumes** $length\ ys = length\ xs$
**and** $length\ zs = length\ xs$
**and** *dist*: $distinct\ xs$
**and** *map-of*: $map\text{-}of\ (zip\ xs\ ys) = map\text{-}of\ (zip\ xs\ zs)$
**shows** $ys = zs$
$\langle proof \rangle$

**lemma** *map-of-zip-nth*:
**assumes** $length\ xs = length\ ys$
**assumes** $distinct\ xs$
**assumes** $i < length\ ys$
**shows** $map\text{-}of\ (zip\ xs\ ys)\ (xs\ !\ i) = Some\ (ys\ !\ i)$
$\langle proof \rangle$

**lemma** *map-of-zip-map*:

$map\text{-}of\ (zip\ xs\ (map\ f\ xs)) = (\lambda x.\ if\ x \in set\ xs\ then\ Some\ (f\ x)\ else\ None)$
⟨proof⟩

**lemma** *finite-range-map-of*: $finite\ (range\ (map\text{-}of\ xys))$
⟨proof⟩

**lemma** *map-of-SomeD*: $map\text{-}of\ xs\ k = Some\ y \Longrightarrow (k,\ y) \in set\ xs$
⟨proof⟩

**lemma** *map-of-mapk-SomeI*:
$inj\ f \Longrightarrow map\text{-}of\ t\ k = Some\ x \Longrightarrow$
$map\text{-}of\ (map\ (case\text{-}prod\ (\lambda k.\ Pair\ (f\ k)))\ t)\ (f\ k) = Some\ x$
⟨proof⟩

**lemma** *weak-map-of-SomeI*: $(k,\ x) \in set\ l \Longrightarrow \exists x.\ map\text{-}of\ l\ k = Some\ x$
⟨proof⟩

**lemma** *map-of-filter-in*:
$map\text{-}of\ xs\ k = Some\ z \Longrightarrow P\ k\ z \Longrightarrow map\text{-}of\ (filter\ (case\text{-}prod\ P)\ xs)\ k = Some\ z$
⟨proof⟩

**lemma** *map-of-map*:
$map\text{-}of\ (map\ (\lambda(k,\ v).\ (k,\ f\ v))\ xs) = map\text{-}option\ f \circ map\text{-}of\ xs$
⟨proof⟩

**lemma** *map-of-filter*:
$map\text{-}of\ (filter\ (\lambda x.\ P\ (fst\ x))\ xs) = map\text{-}of\ xs\ |`\ Collect\ P$
⟨proof⟩

**lemma** *dom-map-option*:
$dom\ (\lambda k.\ map\text{-}option\ (f\ k)\ (m\ k)) = dom\ m$
⟨proof⟩

**lemma** *dom-map-option-comp* [*simp*]:
$dom\ (map\text{-}option\ g \circ m) = dom\ m$
⟨proof⟩

## 71.4 *map-option* related

**lemma** *map-option-o-empty* [*simp*]: $map\text{-}option\ f \circ empty = empty$
⟨proof⟩

**lemma** *map-option-o-map-upd* [*simp*]:
$map\text{-}option\ f \circ m(a \mapsto b) = (map\text{-}option\ f \circ m)(a \mapsto f\ b)$
⟨proof⟩

## 71.5 *map-comp* related

**lemma** *map-comp-empty* [*simp*]:

$m \circ_m empty = empty$
$empty \circ_m m = empty$
⟨proof⟩

**lemma** *map-comp-simps* [*simp*]:
$m2\ k = None \Longrightarrow (m1 \circ_m m2)\ k = None$
$m2\ k = Some\ k' \Longrightarrow (m1 \circ_m m2)\ k = m1\ k'$
⟨proof⟩

**lemma** *map-comp-Some-iff*:
$((m1 \circ_m m2)\ k = Some\ v) = (\exists k'.\ m2\ k = Some\ k' \wedge m1\ k' = Some\ v)$
⟨proof⟩

**lemma** *map-comp-None-iff*:
$((m1 \circ_m m2)\ k = None) = (m2\ k = None \vee (\exists k'.\ m2\ k = Some\ k' \wedge m1\ k' = None))$
⟨proof⟩

## 71.6 ++

**lemma** *map-add-empty*[*simp*]: $m ++ empty = m$
⟨proof⟩

**lemma** *empty-map-add*[*simp*]: $empty ++ m = m$
⟨proof⟩

**lemma** *map-add-assoc*[*simp*]: $m1 ++ (m2 ++ m3) = (m1 ++ m2) ++ m3$
⟨proof⟩

**lemma** *map-add-Some-iff*:
$((m ++ n)\ k = Some\ x) = (n\ k = Some\ x \vee n\ k = None \wedge m\ k = Some\ x)$
⟨proof⟩

**lemma** *map-add-SomeD* [*dest!*]:
$(m ++ n)\ k = Some\ x \Longrightarrow n\ k = Some\ x \vee n\ k = None \wedge m\ k = Some\ x$
⟨proof⟩

**lemma** *map-add-find-right* [*simp*]: $n\ k = Some\ xx \Longrightarrow (m ++ n)\ k = Some\ xx$
⟨proof⟩

**lemma** *map-add-None* [*iff*]: $((m ++ n)\ k = None) = (n\ k = None \wedge m\ k = None)$
⟨proof⟩

**lemma** *map-add-upd*[*simp*]: $f ++ g(x \mapsto y) = (f ++ g)(x \mapsto y)$
⟨proof⟩

**lemma** *map-add-upds*[*simp*]: $m1 ++ (m2(xs[\mapsto]ys)) = (m1{+}{+}m2)(xs[\mapsto]ys)$
⟨proof⟩

**lemma** *map-add-upd-left*: $m \notin dom\ e2 \Longrightarrow e1(m \mapsto u1) ++ e2 = (e1 ++ e2)(m \mapsto u1)$
⟨*proof*⟩

**lemma** *map-of-append*[*simp*]: $map\text{-}of\ (xs\ @\ ys) = map\text{-}of\ ys ++ map\text{-}of\ xs$
⟨*proof*⟩

**lemma** *finite-range-map-of-map-add*:
$finite\ (range\ f) \Longrightarrow finite\ (range\ (f ++ map\text{-}of\ l))$
⟨*proof*⟩

**lemma** *inj-on-map-add-dom* [*iff*]:
$inj\text{-}on\ (m ++ m')\ (dom\ m') = inj\text{-}on\ m'\ (dom\ m')$
⟨*proof*⟩

**lemma** *map-upds-fold-map-upd*:
$m(ks[\mapsto]vs) = foldl\ (\lambda m\ (k,\ v).\ m(k \mapsto v))\ m\ (zip\ ks\ vs)$
⟨*proof*⟩

**lemma** *map-add-map-of-foldr*:
$m ++ map\text{-}of\ ps = foldr\ (\lambda(k,\ v)\ m.\ m(k \mapsto v))\ ps\ m$
⟨*proof*⟩

## 71.7 *restrict-map*

**lemma** *restrict-map-to-empty* [*simp*]: $m|`\{\} = empty$
⟨*proof*⟩

**lemma** *restrict-map-insert*: $f\ |`\ (insert\ a\ A) = (f\ |`\ A)(a := f\ a)$
⟨*proof*⟩

**lemma** *restrict-map-empty* [*simp*]: $empty|`D = empty$
⟨*proof*⟩

**lemma** *restrict-in* [*simp*]: $x \in A \Longrightarrow (m|`A)\ x = m\ x$
⟨*proof*⟩

**lemma** *restrict-out* [*simp*]: $x \notin A \Longrightarrow (m|`A)\ x = None$
⟨*proof*⟩

**lemma** *ran-restrictD*: $y \in ran\ (m|`A) \Longrightarrow \exists x \in A.\ m\ x = Some\ y$
⟨*proof*⟩

**lemma** *dom-restrict* [*simp*]: $dom\ (m|`A) = dom\ m \cap A$
⟨*proof*⟩

**lemma** *restrict-upd-same* [*simp*]: $m(x \mapsto y)|`(-\{x\}) = m|`(-\{x\})$
⟨*proof*⟩

**lemma** *restrict-restrict* [*simp*]: $m|`A|`B = m|`(A \cap B)$
⟨*proof*⟩

**lemma** *restrict-fun-upd* [*simp*]:
$m(x := y)|`D = (\mathit{if}\ x \in D\ \mathit{then}\ (m|`(D-\{x\}))(x := y)\ \mathit{else}\ m|`D)$
⟨*proof*⟩

**lemma** *fun-upd-None-restrict* [*simp*]:
$(m|`D)(x := None) = (\mathit{if}\ x \in D\ \mathit{then}\ m|`(D - \{x\})\ \mathit{else}\ m|`D)$
⟨*proof*⟩

**lemma** *fun-upd-restrict*: $(m|`D)(x := y) = (m|`(D-\{x\}))(x := y)$
⟨*proof*⟩

**lemma** *fun-upd-restrict-conv* [*simp*]:
$x \in D \Longrightarrow (m|`D)(x := y) = (m|`(D-\{x\}))(x := y)$
⟨*proof*⟩

**lemma** *map-of-map-restrict*:
$\mathit{map\text{-}of}\ (\mathit{map}\ (\lambda k.\ (k, f\ k))\ ks) = (\mathit{Some} \circ f)\ |`\ \mathit{set}\ ks$
⟨*proof*⟩

**lemma** *restrict-complement-singleton-eq*:
$f\ |`\ (-\ \{x\}) = f(x := None)$
⟨*proof*⟩

## 71.8 *map-upds*

**lemma** *map-upds-Nil1* [*simp*]: $m([]\ [\mapsto]\ bs) = m$
⟨*proof*⟩

**lemma** *map-upds-Nil2* [*simp*]: $m(as\ [\mapsto]\ []) = m$
⟨*proof*⟩

**lemma** *map-upds-Cons* [*simp*]: $m(a\#as\ [\mapsto]\ b\#bs) = (m(a \mapsto b))(as[\mapsto]bs)$
⟨*proof*⟩

**lemma** *map-upds-append1* [*simp*]:
$\mathit{size}\ xs < \mathit{size}\ ys \Longrightarrow m(xs@[x]\ [\mapsto]\ ys) = m(xs\ [\mapsto]\ ys,\ x \mapsto ys!\mathit{size}\ xs)$
⟨*proof*⟩

**lemma** *map-upds-list-update2-drop* [*simp*]:
$\mathit{size}\ xs \leq i \Longrightarrow m(xs[\mapsto]ys[i{:=}y]) = m(xs[\mapsto]ys)$
⟨*proof*⟩

Something weirdly sensitive about this proof, which needs only four lines in apply style

**lemma** *map-upd-upds-conv-if*:
$(f(x \mapsto y))(xs\ [\mapsto]\ ys) =$

$(\textit{if } x \in \textit{set}(\textit{take } (\textit{length } ys) \ xs) \ \textit{then } f(xs \ [\mapsto] \ ys)$
$\quad\quad\quad\quad\quad\quad \textit{else } (f(xs \ [\mapsto] \ ys))(x \mapsto y))$
⟨proof⟩

**lemma** *map-upds-twist* [*simp*]:
$a \notin \textit{set } as \Longrightarrow m(a \mapsto b,\ as[\mapsto]bs) = m(as[\mapsto]bs,\ a \mapsto b)$
⟨proof⟩

**lemma** *map-upds-apply-nontin* [*simp*]:
$x \notin \textit{set } xs \Longrightarrow (f(xs[\mapsto]ys)) \ x = f \ x$
⟨proof⟩

**lemma** *fun-upds-append-drop* [*simp*]:
$\textit{size } xs = \textit{size } ys \Longrightarrow m(xs@zs[\mapsto]ys) = m(xs[\mapsto]ys)$
⟨proof⟩

**lemma** *fun-upds-append2-drop* [*simp*]:
$\textit{size } xs = \textit{size } ys \Longrightarrow m(xs[\mapsto]ys@zs) = m(xs[\mapsto]ys)$
⟨proof⟩

**lemma** *restrict-map-upds*[*simp*]:
$[\![ \ \textit{length } xs = \textit{length } ys;\ \textit{set } xs \subseteq D \ ]\!]$
$\Longrightarrow m(xs \ [\mapsto] \ ys)|`D = (m|`(D - \textit{set } xs))(xs \ [\mapsto] \ ys)$
⟨proof⟩

## 71.9 *dom*

**lemma** *dom-eq-empty-conv* [*simp*]: $\textit{dom } f = \{\} \longleftrightarrow f = \textit{empty}$
⟨proof⟩

**lemma** *domI*: $m \ a = \textit{Some } b \Longrightarrow a \in \textit{dom } m$
⟨proof⟩

**lemma** *domD*: $a \in \textit{dom } m \Longrightarrow \exists \, b. \ m \ a = \textit{Some } b$
⟨proof⟩

**lemma** *domIff* [*iff*, *simp del*, *code-unfold*]: $a \in \textit{dom } m \longleftrightarrow m \ a \neq \textit{None}$
⟨proof⟩

**lemma** *dom-empty* [*simp*]: $\textit{dom empty} = \{\}$
⟨proof⟩

**lemma** *dom-fun-upd* [*simp*]:
$\textit{dom}(f(x := y)) = (\textit{if } y = \textit{None then dom } f - \{x\} \textit{ else insert } x \ (\textit{dom } f))$
⟨proof⟩

**lemma** *dom-if*:

$dom\ (\lambda x.\ if\ P\ x\ then\ f\ x\ else\ g\ x) = dom\ f \cap \{x.\ P\ x\} \cup dom\ g \cap \{x.\ \neg\ P\ x\}$
⟨proof⟩

**lemma** *dom-map-of-conv-image-fst*:
$dom\ (map\text{-}of\ xys) = fst\ `\ set\ xys$
⟨proof⟩

**lemma** *dom-map-of-zip* [*simp*]: $length\ xs = length\ ys \Longrightarrow dom\ (map\text{-}of\ (zip\ xs\ ys)) = set\ xs$
⟨proof⟩

**lemma** *finite-dom-map-of*: $finite\ (dom\ (map\text{-}of\ l))$
⟨proof⟩

**lemma** *dom-map-upds* [*simp*]:
$dom(m(xs[\mapsto]ys)) = set(take\ (length\ ys)\ xs) \cup dom\ m$
⟨proof⟩

**lemma** *dom-map-add* [*simp*]: $dom\ (m\ {+}{+}\ n) = dom\ n \cup dom\ m$
⟨proof⟩

**lemma** *dom-override-on* [*simp*]:
$dom\ (override\text{-}on\ f\ g\ A) =$
$(dom\ f\ - \{a.\ a \in A - dom\ g\}) \cup \{a.\ a \in A \cap dom\ g\}$
⟨proof⟩

**lemma** *map-add-comm*: $dom\ m1 \cap dom\ m2 = \{\} \Longrightarrow m1\ {+}{+}\ m2 = m2\ {+}{+}\ m1$
⟨proof⟩

**lemma** *map-add-dom-app-simps*:
$m \in dom\ l2 \Longrightarrow (l1\ {+}{+}\ l2)\ m = l2\ m$
$m \notin dom\ l1 \Longrightarrow (l1\ {+}{+}\ l2)\ m = l2\ m$
$m \notin dom\ l2 \Longrightarrow (l1\ {+}{+}\ l2)\ m = l1\ m$
⟨proof⟩

**lemma** *dom-const* [*simp*]:
$dom\ (\lambda x.\ Some\ (f\ x)) = UNIV$
⟨proof⟩

**lemma** *finite-map-freshness*:
$finite\ (dom\ (f :: 'a \rightharpoonup 'b)) \Longrightarrow \neg\ finite\ (UNIV :: 'a\ set) \Longrightarrow$
$\exists x.\ f\ x = None$
⟨proof⟩

**lemma** *dom-minus*:
$f\ x = None \Longrightarrow dom\ f - insert\ x\ A = dom\ f - A$
⟨proof⟩

**lemma** *insert-dom*:
$f\ x = Some\ y \Longrightarrow insert\ x\ (dom\ f) = dom\ f$
⟨*proof*⟩

**lemma** *map-of-map-keys*:
$set\ xs = dom\ m \Longrightarrow map\text{-}of\ (map\ (\lambda k.\ (k,\ the\ (m\ k)))\ xs) = m$
⟨*proof*⟩

**lemma** *map-of-eqI*:
**assumes** *set-eq*: $set\ (map\ fst\ xs) = set\ (map\ fst\ ys)$
**assumes** *map-eq*: $\forall k \in set\ (map\ fst\ xs).\ map\text{-}of\ xs\ k = map\text{-}of\ ys\ k$
**shows** $map\text{-}of\ xs = map\text{-}of\ ys$
⟨*proof*⟩

**lemma** *map-of-eq-dom*:
**assumes** $map\text{-}of\ xs = map\text{-}of\ ys$
**shows** $fst\ `\ set\ xs = fst\ `\ set\ ys$
⟨*proof*⟩

**lemma** *finite-set-of-finite-maps*:
**assumes** *finite A finite B*
**shows** $finite\ \{m.\ dom\ m = A \wedge ran\ m \subseteq B\}$ (**is** *finite ?S*)
⟨*proof*⟩

## 71.10 *ran*

**lemma** *ranI*: $m\ a = Some\ b \Longrightarrow b \in ran\ m$
⟨*proof*⟩

**lemma** *ran-empty* [*simp*]: $ran\ empty = \{\}$
⟨*proof*⟩

**lemma** *ran-map-upd* [*simp*]: $m\ a = None \Longrightarrow ran(m(a \mapsto b)) = insert\ b\ (ran\ m)$
⟨*proof*⟩

**lemma** *fun-upd-None-if-notin-dom*[*simp*]: $k \notin dom\ m \Longrightarrow m(k := None) = m$
⟨*proof*⟩

**lemma** *ran-map-upd-Some*:
$[\![\ m\ x = Some\ y;\ inj\text{-}on\ m\ (dom\ m);\ z \notin ran\ m\ ]\!] \Longrightarrow ran(m(x := Some\ z)) = ran\ m - \{y\} \cup \{z\}$
⟨*proof*⟩

**lemma** *ran-map-add*:
**assumes** $dom\ m1 \cap dom\ m2 = \{\}$
**shows** $ran\ (m1\ {+}{+}\ m2) = ran\ m1 \cup ran\ m2$
⟨*proof*⟩

**lemma** *finite-ran*:
**assumes** *finite* (*dom p*)
**shows** *finite* (*ran p*)
⟨*proof*⟩

**lemma** *ran-distinct*:
**assumes** *dist*: *distinct* (*map fst al*)
**shows** *ran* (*map-of al*) = *snd* ' *set al*
⟨*proof*⟩

**lemma** *ran-map-of-zip*:
**assumes** *length xs* = *length ys distinct xs*
**shows** *ran* (*map-of* (*zip xs ys*)) = *set ys*
⟨*proof*⟩

**lemma** *ran-map-option*: *ran* ($\lambda x$. *map-option f* (*m x*)) = *f* ' *ran m*
⟨*proof*⟩

## 71.11 *graph*

**lemma** *graph-empty*[*simp*]: *graph empty* = {}
⟨*proof*⟩

**lemma** *in-graphI*: *m k* = *Some v* $\Longrightarrow$ (*k*, *v*) $\in$ *graph m*
⟨*proof*⟩

**lemma** *in-graphD*: (*k*, *v*) $\in$ *graph m* $\Longrightarrow$ *m k* = *Some v*
⟨*proof*⟩

**lemma** *graph-map-upd*[*simp*]: *graph* (*m*(*k* $\mapsto$ *v*)) = *insert* (*k*, *v*) (*graph* (*m*(*k* := *None*)))
⟨*proof*⟩

**lemma** *graph-fun-upd-None*: *graph* (*m*(*k* := *None*)) = {*e* $\in$ *graph m*. *fst e* $\neq$ *k*}
⟨*proof*⟩

**lemma** *graph-restrictD*:
**assumes** (*k*, *v*) $\in$ *graph* (*m* |' *A*)
**shows** *k* $\in$ *A* **and** *m k* = *Some v*
⟨*proof*⟩

**lemma** *graph-map-comp*[*simp*]: *graph* (*m1* $\circ_m$ *m2*) = *graph m2 O graph m1*
⟨*proof*⟩

**lemma** *graph-map-add*: *dom m1* $\cap$ *dom m2* = {} $\Longrightarrow$ *graph* (*m1* ++ *m2*) = *graph m1* $\cup$ *graph m2*
⟨*proof*⟩

**lemma** *graph-eq-to-snd-dom*: $graph\ m = (\lambda x.\ (x,\ the\ (m\ x)))$ ‘ $dom\ m$
⟨*proof*⟩

**lemma** *fst-graph-eq-dom*: $fst$ ‘ $graph\ m = dom\ m$
⟨*proof*⟩

**lemma** *graph-domD*: $x \in graph\ m \Longrightarrow fst\ x \in dom\ m$
⟨*proof*⟩

**lemma** *snd-graph-ran*: $snd$ ‘ $graph\ m = ran\ m$
⟨*proof*⟩

**lemma** *graph-ranD*: $x \in graph\ m \Longrightarrow snd\ x \in ran\ m$
⟨*proof*⟩

**lemma** *finite-graph-map-of*: $finite\ (graph\ (map\text{-}of\ al))$
⟨*proof*⟩

**lemma** *graph-map-of-if-distinct-dom*: $distinct\ (map\ fst\ al) \Longrightarrow graph\ (map\text{-}of\ al) = set\ al$
⟨*proof*⟩

**lemma** *finite-graph-iff-finite-dom*[*simp*]: $finite\ (graph\ m) = finite\ (dom\ m)$
⟨*proof*⟩

**lemma** *inj-on-fst-graph*: $inj\text{-}on\ fst\ (graph\ m)$
⟨*proof*⟩

## 71.12 *map-le*

**lemma** *map-le-empty* [*simp*]: $empty \subseteq_m g$
⟨*proof*⟩

**lemma** *upd-None-map-le* [*simp*]: $f(x := None) \subseteq_m f$
⟨*proof*⟩

**lemma** *map-le-upd*[*simp*]: $f \subseteq_m g$ ==> $f(a := b) \subseteq_m g(a := b)$
⟨*proof*⟩

**lemma** *map-le-imp-upd-le* [*simp*]: $m1 \subseteq_m m2 \Longrightarrow m1(x := None) \subseteq_m m2(x \mapsto y)$
⟨*proof*⟩

**lemma** *map-le-upds* [*simp*]:
$f \subseteq_m g \Longrightarrow f(as\ [\mapsto]\ bs) \subseteq_m g(as\ [\mapsto]\ bs)$
⟨*proof*⟩

**lemma** *map-le-implies-dom-le*: $(f \subseteq_m g) \Longrightarrow (dom\ f \subseteq dom\ g)$
⟨*proof*⟩

**lemma** *map-le-refl* [*simp*]: $f \subseteq_m f$
⟨*proof*⟩

**lemma** *map-le-trans*[*trans*]: $[\![ m1 \subseteq_m m2; m2 \subseteq_m m3 ]\!] \Longrightarrow m1 \subseteq_m m3$
⟨*proof*⟩

**lemma** *map-le-antisym*: $[\![ f \subseteq_m g; g \subseteq_m f ]\!] \Longrightarrow f = g$
⟨*proof*⟩

**lemma** *map-le-map-add* [*simp*]: $f \subseteq_m g ++ f$
⟨*proof*⟩

**lemma** *map-le-iff-map-add-commute*: $f \subseteq_m f ++ g \longleftrightarrow f ++ g = g ++ f$
⟨*proof*⟩

**lemma** *map-add-le-mapE*: $f ++ g \subseteq_m h \Longrightarrow g \subseteq_m h$
⟨*proof*⟩

**lemma** *map-add-le-mapI*: $[\![ f \subseteq_m h; g \subseteq_m h ]\!] \Longrightarrow f ++ g \subseteq_m h$
⟨*proof*⟩

**lemma** *map-add-subsumed1*: $f \subseteq_m g \Longrightarrow f{+}{+}g = g$
⟨*proof*⟩

**lemma** *map-add-subsumed2*: $f \subseteq_m g \Longrightarrow g{+}{+}f = g$
⟨*proof*⟩

**lemma** *dom-eq-singleton-conv*: $dom\ f = \{x\} \longleftrightarrow (\exists\, v.\ f = [x \mapsto v])$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** *map-add-eq-empty-iff*[*simp*]:
$(f{+}{+}g = empty) \longleftrightarrow f = empty \wedge g = empty$
⟨*proof*⟩

**lemma** *empty-eq-map-add-iff*[*simp*]:
$(empty = f{+}{+}g) \longleftrightarrow f = empty \wedge g = empty$
⟨*proof*⟩

## 71.13 Various

**lemma** *set-map-of-compr*:
**assumes** *distinct*: *distinct* (*map fst xs*)
**shows** *set xs* = $\{(k, v).\ map\text{-}of\ xs\ k = Some\ v\}$
⟨*proof*⟩

**lemma** *eq-key-imp-eq-value*:
$v1 = v2$

**if** *distinct* (*map fst xs*) (*k*, *v1*) ∈ *set xs* (*k*, *v2*) ∈ *set xs*
⟨*proof*⟩

**lemma** *map-of-inject-set*:
**assumes** *distinct*: *distinct* (*map fst xs*) *distinct* (*map fst ys*)
**shows** *map-of xs* = *map-of ys* ⟷ *set xs* = *set ys* (**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

**lemma** *finite-Map-induct*[*consumes 1*, *case-names empty update*]:
**assumes** *finite* (*dom m*)
**assumes** *P Map.empty*
**assumes** $\bigwedge k\ v\ m.\ finite\ (dom\ m) \Longrightarrow k \notin dom\ m \Longrightarrow P\ m \Longrightarrow P\ (m(k \mapsto v))$
**shows** *P m*
⟨*proof*⟩

**hide-const** (**open**) *Map.empty Map.graph*

**end**

# 72 Finite types as explicit enumerations

**theory** *Enum*
**imports** *Map Groups-List*
**begin**

## 72.1 Class *enum*

**class** *enum* =
**fixes** *enum* :: *'a list*
**fixes** *enum-all* :: (*'a* ⇒ *bool*) ⇒ *bool*
**fixes** *enum-ex* :: (*'a* ⇒ *bool*) ⇒ *bool*
**assumes** *UNIV-enum*: *UNIV* = *set enum*
**and** *enum-distinct*: *distinct enum*
**assumes** *enum-all-UNIV*: *enum-all P* ⟷ *Ball UNIV P*
**assumes** *enum-ex-UNIV*: *enum-ex P* ⟷ *Bex UNIV P*
— tailored towards simple instantiation
**begin**

**subclass** *finite* ⟨*proof*⟩

**lemma** *enum-UNIV*:
*set enum* = *UNIV*
⟨*proof*⟩

**lemma** *in-enum*: *x* ∈ *set enum*
⟨*proof*⟩

**lemma** *enum-eq-I*:
**assumes** $\bigwedge x.\ x \in set\ xs$

**shows** *set enum* = *set xs*
⟨*proof*⟩

**lemma** *card-UNIV-length-enum*:
*card* (*UNIV* :: *'a set*) = *length enum*
⟨*proof*⟩

**lemma** *enum-all* [*simp*]:
*enum-all* = *HOL.All*
⟨*proof*⟩

**lemma** *enum-ex* [*simp*]:
*enum-ex* = *HOL.Ex*
⟨*proof*⟩

**end**

## 72.2 Implementations using *enum*

### 72.2.1 Unbounded operations and quantifiers

**lemma** *Collect-code* [*code*]:
*Collect P* = *set* (*filter P enum*)
⟨*proof*⟩

**lemma** *vimage-code* [*code*]:
$f$ −‘ $B$ = *set* (*filter* ($\lambda x.\ f\ x \in B$) *enum-class.enum*)
⟨*proof*⟩

**definition** *card-UNIV* :: *'a itself* ⇒ *nat*
**where**
*card-UNIV TYPE('a)* = *card* (*UNIV* :: *'a set*)

**lemma** [*code*]:
*card-UNIV TYPE('a* :: *enum*) = *card* (*set* (*Enum.enum* :: *'a list*))
⟨*proof*⟩

**lemma** *all-code* [*code*]: ($\forall x.\ P\ x$) ⟷ *enum-all P*
⟨*proof*⟩

**lemma** *exists-code* [*code*]: ($\exists x.\ P\ x$) ⟷ *enum-ex P*
⟨*proof*⟩

**lemma** *exists1-code* [*code*]: ($\exists!x.\ P\ x$) ⟷ *list-ex1 P enum*
⟨*proof*⟩

### 72.2.2 An executable choice operator

**definition**
*enum-the* = *The*

**lemma** [*code*]:
  *The P* = (*case filter P enum of* [*x*] ⇒ *x* | - ⇒ *enum-the P*)
⟨*proof*⟩

**declare** [[*code abort*: *enum-the*]]

**code-printing**
  **constant** *enum-the* ⇀ (*Eval*) (*fn* ′- => *raise Match*)

### 72.2.3 Equality and order on functions

**instantiation** *fun* :: (*enum*, *equal*) *equal*
**begin**

**definition**
  *HOL.equal f g* ⟷ (∀ *x* ∈ *set enum*. *f x* = *g x*)

**instance** ⟨*proof*⟩

**end**

**lemma** [*code*]:
  *HOL.equal f g* ⟷ *enum-all* (%*x*. *f x* = *g x*)
  ⟨*proof*⟩

**lemma** [*code nbe*]:
  *HOL.equal* (*f* :: - ⇒ -) *f* ⟷ *True*
  ⟨*proof*⟩

**lemma** *order-fun* [*code*]:
  **fixes** *f g* :: ′*a*::*enum* ⇒ ′*b*::*order*
  **shows** *f* ≤ *g* ⟷ *enum-all* (λ*x*. *f x* ≤ *g x*)
    **and** *f* < *g* ⟷ *f* ≤ *g* ∧ *enum-ex* (λ*x*. *f x* ≠ *g x*)
  ⟨*proof*⟩

### 72.2.4 Operations on relations

**lemma** [*code*]:
  *Id* = *image* (λ*x*. (*x*, *x*)) (*set Enum.enum*)
  ⟨*proof*⟩

**lemma** *tranclp-unfold* [*code*]:
  *tranclp r a b* ⟷ (*a*, *b*) ∈ *trancl* {(*x*, *y*). *r x y*}
  ⟨*proof*⟩

**lemma** *rtranclp-rtrancl-eq* [*code*]:
  *rtranclp r x y* ⟷ (*x*, *y*) ∈ *rtrancl* {(*x*, *y*). *r x y*}
  ⟨*proof*⟩

**lemma** *max-ext-eq* [*code*]:
$max\text{-}ext\ R = \{(X,\ Y).\ finite\ X \wedge finite\ Y \wedge Y \neq \{\} \wedge (\forall x.\ x \in X \longrightarrow (\exists xa \in Y.\ (x,\ xa) \in R))\}$
⟨*proof*⟩

**lemma** *max-extp-eq* [*code*]:
$max\text{-}extp\ r\ x\ y \longleftrightarrow (x,\ y) \in max\text{-}ext\ \{(x,\ y).\ r\ x\ y\}$
⟨*proof*⟩

**lemma** *mlex-eq* [*code*]:
$f <\!*mlex*\!> R = \{(x,\ y).\ f\ x < f\ y \vee (f\ x \leq f\ y \wedge (x,\ y) \in R)\}$
⟨*proof*⟩

### 72.2.5 Bounded accessible part

**primrec** $bacc :: ('a \times 'a)\ set \Rightarrow nat \Rightarrow 'a\ set$
**where**
$bacc\ r\ 0 = \{x.\ \forall\ y.\ (y,\ x) \notin r\}$
$|\ bacc\ r\ (Suc\ n) = (bacc\ r\ n \cup \{x.\ \forall y.\ (y,\ x) \in r \longrightarrow y \in bacc\ r\ n\})$

**lemma** *bacc-subseteq-acc*:
$bacc\ r\ n \subseteq Wellfounded.acc\ r$
⟨*proof*⟩

**lemma** *bacc-mono*:
$n \leq m \Longrightarrow bacc\ r\ n \subseteq bacc\ r\ m$
⟨*proof*⟩

**lemma** *bacc-upper-bound*:
$bacc\ (r :: ('a \times 'a)\ set)\ \ (card\ (UNIV :: 'a{::}finite\ set)) = (\bigcup n.\ bacc\ r\ n)$
⟨*proof*⟩

**lemma** *acc-subseteq-bacc*:
**assumes** *finite r*
**shows** $Wellfounded.acc\ r \subseteq (\bigcup n.\ bacc\ r\ n)$
⟨*proof*⟩

**lemma** *acc-bacc-eq*:
**fixes** $A :: ('a :: finite \times 'a)\ set$
**assumes** *finite A*
**shows** $Wellfounded.acc\ A = bacc\ A\ (card\ (UNIV :: 'a\ set))$
⟨*proof*⟩

**lemma** [*code*]:
**fixes** $xs :: ('a{::}finite \times 'a)\ list$
**shows** $Wellfounded.acc\ (set\ xs) = bacc\ (set\ xs)\ (card\text{-}UNIV\ TYPE('a))$
⟨*proof*⟩

## 72.3 Default instances for *enum*

**lemma** *map-of-zip-enum-is-Some*:
  **assumes** *length ys* = *length* (*enum* :: *'a*::*enum list*)
  **shows** ∃ *y*. *map-of* (*zip* (*enum* :: *'a*::*enum list*) *ys*) *x* = *Some y*
⟨*proof*⟩

**lemma** *map-of-zip-enum-inject*:
  **fixes** *xs ys* :: *'b*::*enum list*
  **assumes** *length*: *length xs* = *length* (*enum* :: *'a*::*enum list*)
    *length ys* = *length* (*enum* :: *'a*::*enum list*)
   **and** *map-of*: *the* ∘ *map-of* (*zip* (*enum* :: *'a*::*enum list*) *xs*) = *the* ∘ *map-of* (*zip* (*enum* :: *'a*::*enum list*) *ys*)
  **shows** *xs* = *ys*
⟨*proof*⟩

**definition** *all-n-lists* :: ((*'a* :: *enum*) *list* ⇒ *bool*) ⇒ *nat* ⇒ *bool*
**where**
  *all-n-lists P n* ⟷ (∀ *xs* ∈ *set* (*List.n-lists n enum*). *P xs*)

**lemma** [*code*]:
  *all-n-lists P n* ⟷ (*if n* = *0 then P* [] *else enum-all* (%*x*. *all-n-lists* (%*xs*. *P* (*x* # *xs*)) (*n* − *1*)))
  ⟨*proof*⟩

**definition** *ex-n-lists* :: ((*'a* :: *enum*) *list* ⇒ *bool*) ⇒ *nat* ⇒ *bool*
**where**
  *ex-n-lists P n* ⟷ (∃ *xs* ∈ *set* (*List.n-lists n enum*). *P xs*)

**lemma** [*code*]:
  *ex-n-lists P n* ⟷ (*if n* = *0 then P* [] *else enum-ex* (%*x*. *ex-n-lists* (%*xs*. *P* (*x* # *xs*)) (*n* − *1*)))
  ⟨*proof*⟩

**instantiation** *fun* :: (*enum*, *enum*) *enum*
**begin**

**definition**
  *enum* = *map* (λ*ys*. *the* ∘ *map-of* (*zip* (*enum*::*'a list*) *ys*)) (*List.n-lists* (*length* (*enum*::*'a*::*enum list*)) *enum*)

**definition**
  *enum-all P* = *all-n-lists* (λ*bs*. *P* (*the* ∘ *map-of* (*zip enum bs*))) (*length* (*enum* :: *'a list*))

**definition**
  *enum-ex P* = *ex-n-lists* (λ*bs*. *P* (*the* ∘ *map-of* (*zip enum bs*))) (*length* (*enum* :: *'a list*))

**instance** ⟨*proof*⟩

**end**

**lemma** *enum-fun-code* [*code*]: *enum* = (*let enum-a* = (*enum* :: $'a$::{*enum*, *equal*} *list*)
*in map* ($\lambda$*ys. the* $\circ$ *map-of* (*zip enum-a ys*)) (*List.n-lists* (*length enum-a*) *enum*))
⟨*proof*⟩

**lemma** *enum-all-fun-code* [*code*]:
*enum-all P* = (*let enum-a* = (*enum* :: $'a$::{*enum*, *equal*} *list*)
*in all-n-lists* ($\lambda$*bs. P* (*the* $\circ$ *map-of* (*zip enum-a bs*))) (*length enum-a*))
⟨*proof*⟩

**lemma** *enum-ex-fun-code* [*code*]:
*enum-ex P* = (*let enum-a* = (*enum* :: $'a$::{*enum*, *equal*} *list*)
*in ex-n-lists* ($\lambda$*bs. P* (*the* $\circ$ *map-of* (*zip enum-a bs*))) (*length enum-a*))
⟨*proof*⟩

**instantiation** *set* :: (*enum*) *enum*
**begin**

**definition**
*enum* = *map set* (*subseqs enum*)

**definition**
*enum-all P* $\longleftrightarrow$ ($\forall$ *A*$\in$*set enum. P* (*A*::$'a$ *set*))

**definition**
*enum-ex P* $\longleftrightarrow$ ($\exists$ *A*$\in$*set enum. P* (*A*::$'a$ *set*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *unit* :: *enum*
**begin**

**definition**
*enum* = [()]

**definition**
*enum-all P* = *P* ()

**definition**
*enum-ex P* = *P* ()

**instance** ⟨*proof*⟩

**end**

**instantiation** *bool* :: *enum*
**begin**

**definition**
  $enum = [False,\ True]$

**definition**
  $enum\text{-}all\ P \longleftrightarrow P\ False \wedge P\ True$

**definition**
  $enum\text{-}ex\ P \longleftrightarrow P\ False \vee P\ True$

**instance** ⟨*proof*⟩

**end**

**instantiation** *prod* :: (*enum*, *enum*) *enum*
**begin**

**definition**
  $enum = List.product\ enum\ enum$

**definition**
  $enum\text{-}all\ P = enum\text{-}all\ (\%x.\ enum\text{-}all\ (\%y.\ P\ (x,\ y)))$

**definition**
  $enum\text{-}ex\ P = enum\text{-}ex\ (\%x.\ enum\text{-}ex\ (\%y.\ P\ (x,\ y)))$

**instance**
  ⟨*proof*⟩

**end**

**instantiation** *sum* :: (*enum*, *enum*) *enum*
**begin**

**definition**
  $enum = map\ Inl\ enum\ @\ map\ Inr\ enum$

**definition**
  $enum\text{-}all\ P \longleftrightarrow enum\text{-}all\ (\lambda x.\ P\ (Inl\ x)) \wedge enum\text{-}all\ (\lambda x.\ P\ (Inr\ x))$

**definition**
  $enum\text{-}ex\ P \longleftrightarrow enum\text{-}ex\ (\lambda x.\ P\ (Inl\ x)) \vee enum\text{-}ex\ (\lambda x.\ P\ (Inr\ x))$

**instance** ⟨*proof*⟩

**end**

**instantiation** *option* :: (*enum*) *enum*
**begin**

**definition**
$enum = None \# map\ Some\ enum$

**definition**
$enum\text{-}all\ P \longleftrightarrow P\ None \wedge enum\text{-}all\ (\lambda x.\ P\ (Some\ x))$

**definition**
$enum\text{-}ex\ P \longleftrightarrow P\ None \vee enum\text{-}ex\ (\lambda x.\ P\ (Some\ x))$

**instance** ⟨*proof*⟩

**end**

## 72.4 Small finite types

We define small finite types for use in Quickcheck

**datatype** (*plugins only*: *code quickcheck extraction*) *finite-1* =
$a_1$

**notation** (**output**) $a_1$ (‹$a_1$›)

**lemma** *UNIV-finite-1*:
$UNIV = \{a_1\}$
⟨*proof*⟩

**instantiation** *finite-1* :: *enum*
**begin**

**definition**
$enum = [a_1]$

**definition**
$enum\text{-}all\ P = P\ a_1$

**definition**
$enum\text{-}ex\ P = P\ a_1$

**instance** ⟨*proof*⟩

**end**

**instantiation** *finite-1* :: *linorder*
**begin**

**definition** *less-finite-1* :: *finite-1* ⇒ *finite-1* ⇒ *bool*
**where**
  $x < (y :: \textit{finite-1}) \longleftrightarrow \textit{False}$

**definition** *less-eq-finite-1* :: *finite-1* ⇒ *finite-1* ⇒ *bool*
**where**
  $x \leq (y :: \textit{finite-1}) \longleftrightarrow \textit{True}$

**instance**
⟨*proof*⟩

**end**

**instance** *finite-1* :: {*dense-linorder*, *wellorder*}
⟨*proof*⟩

**instantiation** *finite-1* :: *complete-lattice*
**begin**

**definition** [*simp*]: $\textit{Inf} = (\lambda\text{-}.\ a_1)$
**definition** [*simp*]: $\textit{Sup} = (\lambda\text{-}.\ a_1)$
**definition** [*simp*]: $\textit{bot} = a_1$
**definition** [*simp*]: $\textit{top} = a_1$
**definition** [*simp*]: $\textit{inf} = (\lambda\text{- -}.\ a_1)$
**definition** [*simp*]: $\textit{sup} = (\lambda\text{- -}.\ a_1)$

**instance** ⟨*proof*⟩
**end**

**instance** *finite-1* :: *complete-distrib-lattice*
  ⟨*proof*⟩

**instance** *finite-1* :: *complete-linorder* ⟨*proof*⟩

**lemma** *finite-1-eq*: $x = a_1$
⟨*proof*⟩

⟨*ML*⟩

**instantiation** *finite-1* :: *complete-boolean-algebra*
**begin**
**definition** [*simp*]: $(-) = (\lambda\text{- -}.\ a_1)$
**definition** [*simp*]: $\textit{uminus} = (\lambda\text{-}.\ a_1)$
**instance** ⟨*proof*⟩
**end**

**instantiation** *finite-1* ::
  {*linordered-ring-strict*, *linordered-comm-semiring-strict*, *ordered-comm-ring*,
   *ordered-cancel-comm-monoid-diff*, *comm-monoid-mult*, *ordered-ring-abs*,

*one*, *modulo*, *sgn*, *inverse*}
**begin**
**definition** [*simp*]: *Groups.zero* = $a_1$
**definition** [*simp*]: *Groups.one* = $a_1$
**definition** [*simp*]: $(+) = (\lambda\text{- -. } a_1)$
**definition** [*simp*]: $(*) = (\lambda\text{- -. } a_1)$
**definition** [*simp*]: $(mod) = (\lambda\text{- -. } a_1)$
**definition** [*simp*]: $abs = (\lambda\text{-. } a_1)$
**definition** [*simp*]: $sgn = (\lambda\text{-. } a_1)$
**definition** [*simp*]: $inverse = (\lambda\text{-. } a_1)$
**definition** [*simp*]: $divide = (\lambda\text{- -. } a_1)$

**instance** ⟨*proof*⟩
**end**

**declare** [[*simproc del*: *finite-1-eq*]]
**hide-const** (**open**) $a_1$

**datatype** (*plugins only*: *code quickcheck extraction*) *finite-2* =
$a_1$ | $a_2$

**notation** (**output**) $a_1$ (‹$a_1$›)
**notation** (**output**) $a_2$ (‹$a_2$›)

**lemma** *UNIV-finite-2*:
*UNIV* = {$a_1$, $a_2$}
⟨*proof*⟩

**instantiation** *finite-2* :: *enum*
**begin**

**definition**
*enum* = [$a_1$, $a_2$]

**definition**
*enum-all* $P \longleftrightarrow P\ a_1 \wedge P\ a_2$

**definition**
*enum-ex* $P \longleftrightarrow P\ a_1 \vee P\ a_2$

**instance** ⟨*proof*⟩

**end**

**instantiation** *finite-2* :: *linorder*
**begin**

**definition** *less-finite-2* :: *finite-2* ⇒ *finite-2* ⇒ *bool*
**where**

$x < y \longleftrightarrow x = a_1 \wedge y = a_2$

**definition** *less-eq-finite-2* :: *finite-2* ⇒ *finite-2* ⇒ *bool*
**where**
$x \leq y \longleftrightarrow x = y \vee x < (y :: \textit{finite-2})$

**instance**
⟨*proof*⟩

**end**

**instance** *finite-2* :: *wellorder*
⟨*proof*⟩

**instantiation** *finite-2* :: *complete-lattice*
**begin**

**definition** $\bigsqcap A = (\textit{if } a_1 \in A \textit{ then } a_1 \textit{ else } a_2)$
**definition** $\bigsqcup A = (\textit{if } a_2 \in A \textit{ then } a_2 \textit{ else } a_1)$
**definition** [*simp*]: $\textit{bot} = a_1$
**definition** [*simp*]: $\textit{top} = a_2$
**definition** $x \sqcap y = (\textit{if } x = a_1 \vee y = a_1 \textit{ then } a_1 \textit{ else } a_2)$
**definition** $x \sqcup y = (\textit{if } x = a_2 \vee y = a_2 \textit{ then } a_2 \textit{ else } a_1)$

**lemma** *neq-finite-2-*$a_1$*-iff* [*simp*]: $x \neq a_1 \longleftrightarrow x = a_2$
⟨*proof*⟩

**lemma** *neq-finite-2-*$a_1$*-iff'* [*simp*]: $a_1 \neq x \longleftrightarrow x = a_2$
⟨*proof*⟩

**lemma** *neq-finite-2-*$a_2$*-iff* [*simp*]: $x \neq a_2 \longleftrightarrow x = a_1$
⟨*proof*⟩

**lemma** *neq-finite-2-*$a_2$*-iff'* [*simp*]: $a_2 \neq x \longleftrightarrow x = a_1$
⟨*proof*⟩

**instance**
⟨*proof*⟩
**end**

**instance** *finite-2* :: *complete-linorder* ⟨*proof*⟩

**instance** *finite-2* :: *complete-distrib-lattice* ⟨*proof*⟩

**instantiation** *finite-2* :: {*field*, *idom-abs-sgn*, *idom-modulo*} **begin**
**definition** [*simp*]: $0 = a_1$
**definition** [*simp*]: $1 = a_2$
**definition** $x + y = (\textit{case } (x, y) \textit{ of } (a_1, a_1) \Rightarrow a_1 \mid (a_2, a_2) \Rightarrow a_1 \mid \text{-} \Rightarrow a_2)$
**definition** $\textit{uminus} = (\lambda x :: \textit{finite-2}.\ x)$

**definition** $(-) = ((+) :: \textit{finite-2} \Rightarrow \text{-})$
**definition** $x * y = (\textit{case}\ (x,\ y)\ \textit{of}\ (a_2,\ a_2) \Rightarrow a_2 \mid \text{-} \Rightarrow a_1)$
**definition** $\textit{inverse} = (\lambda x :: \textit{finite-2}.\ x)$
**definition** $\textit{divide} = ((*) :: \textit{finite-2} \Rightarrow \text{-})$
**definition** $x\ \textit{mod}\ y = (\textit{case}\ (x,\ y)\ \textit{of}\ (a_2,\ a_1) \Rightarrow a_2 \mid \text{-} \Rightarrow a_1)$
**definition** $\textit{abs} = (\lambda x :: \textit{finite-2}.\ x)$
**definition** $\textit{sgn} = (\lambda x :: \textit{finite-2}.\ x)$
**instance**
⟨*proof*⟩
**end**

**lemma** *two-finite-2* [*simp*]:
$2 = a_1$
⟨*proof*⟩

**lemma** *dvd-finite-2-unfold*:
$x\ \textit{dvd}\ y \longleftrightarrow x = a_2 \lor y = a_1$
⟨*proof*⟩

**instantiation** *finite-2* :: {*normalization-semidom*, *unique-euclidean-semiring*} **begin**
**definition** [*simp*]: $\textit{normalize} = (\textit{id} :: \textit{finite-2} \Rightarrow \text{-})$
**definition** [*simp*]: $\textit{unit-factor} = (\textit{id} :: \textit{finite-2} \Rightarrow \text{-})$
**definition** [*simp*]: $\textit{euclidean-size}\ x = (\textit{case}\ x\ \textit{of}\ a_1 \Rightarrow 0 \mid a_2 \Rightarrow 1)$
**definition** [*simp*]: $\textit{division-segment}\ (x :: \textit{finite-2}) = 1$
**instance**
⟨*proof*⟩
**end**

**hide-const** (**open**) $a_1$ $a_2$

**datatype** (*plugins only*: *code quickcheck extraction*) *finite-3* =
$a_1 \mid a_2 \mid a_3$

**notation** (**output**) $a_1$ (‹$a_1$›)
**notation** (**output**) $a_2$ (‹$a_2$›)
**notation** (**output**) $a_3$ (‹$a_3$›)

**lemma** *UNIV-finite-3*:
$\textit{UNIV} = \{a_1,\ a_2,\ a_3\}$
⟨*proof*⟩

**instantiation** *finite-3* :: *enum*
**begin**

**definition**
$\textit{enum} = [a_1,\ a_2,\ a_3]$

**definition**
$enum\text{-}all\ P \longleftrightarrow P\ a_1 \wedge P\ a_2 \wedge P\ a_3$

**definition**
$enum\text{-}ex\ P \longleftrightarrow P\ a_1 \vee P\ a_2 \vee P\ a_3$

**instance** ⟨*proof*⟩

**end**

**lemma** *finite-3-not-eq-unfold*:
$x \neq a_1 \longleftrightarrow x \in \{a_2, a_3\}$
$x \neq a_2 \longleftrightarrow x \in \{a_1, a_3\}$
$x \neq a_3 \longleftrightarrow x \in \{a_1, a_2\}$
⟨*proof*⟩

**instantiation** *finite-3* :: *linorder*
**begin**

**definition** *less-finite-3* :: *finite-3* ⇒ *finite-3* ⇒ *bool*
**where**
$x < y = (case\ x\ of\ a_1 \Rightarrow y \neq a_1 \mid a_2 \Rightarrow y = a_3 \mid a_3 \Rightarrow False)$

**definition** *less-eq-finite-3* :: *finite-3* ⇒ *finite-3* ⇒ *bool*
**where**
$x \leq y \longleftrightarrow x = y \vee x < (y :: finite\text{-}3)$

**instance** ⟨*proof*⟩

**end**

**instance** *finite-3* :: *wellorder*
⟨*proof*⟩

**class** *finite-lattice* = *finite* + *lattice* + *Inf* + *Sup* + *bot* + *top* +
**assumes** *Inf-finite-empty*: *Inf* {} = *Sup UNIV*
**assumes** *Inf-finite-insert*: *Inf* (*insert a A*) = *a* ⊓ *Inf A*
**assumes** *Sup-finite-empty*: *Sup* {} = *Inf UNIV*
**assumes** *Sup-finite-insert*: *Sup* (*insert a A*) = *a* ⊔ *Sup A*
**assumes** *bot-finite-def*: *bot* = *Inf UNIV*
**assumes** *top-finite-def*: *top* = *Sup UNIV*
**begin**

**subclass** *complete-lattice*
⟨*proof*⟩
**end**

**class** *finite-distrib-lattice* = *finite-lattice* + *distrib-lattice*
**begin**

**lemma** *finite-inf-Sup*: $a \sqcap (Sup\ A) = Sup\ \{a \sqcap b \mid b .\ b \in A\}$
⟨*proof*⟩

**lemma** *finite-Inf-Sup*: $\bigsqcap(Sup\ `\ A) \leq \bigsqcup(Inf\ `\ \{f\ `\ A\ |f.\ \forall\ Y{\in}A.\ f\ Y \in Y\})$
⟨*proof*⟩

**subclass** *complete-distrib-lattice*
⟨*proof*⟩
**end**

**instantiation** *finite-3* :: *finite-lattice*
**begin**

**definition** $\bigsqcap A = (if\ a_1 \in A\ then\ a_1\ else\ if\ a_2 \in A\ then\ a_2\ else\ a_3)$
**definition** $\bigsqcup A = (if\ a_3 \in A\ then\ a_3\ else\ if\ a_2 \in A\ then\ a_2\ else\ a_1)$
**definition** [*simp*]: $bot = a_1$
**definition** [*simp*]: $top = a_3$
**definition** [*simp*]: $inf = (min :: finite\text{-}3 \Rightarrow \text{-})$
**definition** [*simp*]: $sup = (max :: finite\text{-}3 \Rightarrow \text{-})$

**instance**
⟨*proof*⟩
**end**

**instance** *finite-3* :: *complete-lattice* ⟨*proof*⟩

**instance** *finite-3* :: *finite-distrib-lattice*
⟨*proof*⟩

**instance** *finite-3* :: *complete-distrib-lattice* ⟨*proof*⟩

**instance** *finite-3* :: *complete-linorder* ⟨*proof*⟩

**instantiation** *finite-3* :: {*field*, *idom-abs-sgn*, *idom-modulo*} **begin**
**definition** [*simp*]: $0 = a_1$
**definition** [*simp*]: $1 = a_2$
**definition**
$x + y = (case\ (x,\ y)\ of$
$(a_1,\ a_1) \Rightarrow a_1 \mid (a_2,\ a_3) \Rightarrow a_1 \mid (a_3,\ a_2) \Rightarrow a_1$
$\mid (a_1,\ a_2) \Rightarrow a_2 \mid (a_2,\ a_1) \Rightarrow a_2 \mid (a_3,\ a_3) \Rightarrow a_2$
$\mid \text{-} \Rightarrow a_3)$
**definition** $-\ x = (case\ x\ of\ a_1 \Rightarrow a_1 \mid a_2 \Rightarrow a_3 \mid a_3 \Rightarrow a_2)$
**definition** $x - y = x + (-\ y :: finite\text{-}3)$
**definition** $x * y = (case\ (x,\ y)\ of\ (a_2,\ a_2) \Rightarrow a_2 \mid (a_3,\ a_3) \Rightarrow a_2 \mid (a_2,\ a_3) \Rightarrow a_3 \mid (a_3,\ a_2) \Rightarrow a_3 \mid \text{-} \Rightarrow a_1)$
**definition** $inverse = (\lambda x :: finite\text{-}3.\ x)$
**definition** $x\ div\ y = x * inverse\ (y :: finite\text{-}3)$
**definition** $x\ mod\ y = (case\ y\ of\ a_1 \Rightarrow x \mid \text{-} \Rightarrow a_1)$
**definition** $abs = (\lambda x.\ case\ x\ of\ a_3 \Rightarrow a_2 \mid \text{-} \Rightarrow x)$

**definition** *sgn* = ($\lambda x$ :: *finite-3*. $x$)
**instance**
⟨*proof*⟩
**end**

**lemma** *two-finite-3* [*simp*]:
$2 = a_3$
⟨*proof*⟩

**lemma** *dvd-finite-3-unfold*:
$x$ *dvd* $y \longleftrightarrow x = a_2 \lor x = a_3 \lor y = a_1$
⟨*proof*⟩

**instantiation** *finite-3* :: {*normalization-semidom*, *unique-euclidean-semiring*} **begin**
**definition** [*simp*]: *normalize* $x$ = (*case* $x$ *of* $a_3 \Rightarrow a_2$ | - $\Rightarrow x$)
**definition** [*simp*]: *unit-factor* = (*id* :: *finite-3* $\Rightarrow$ -)
**definition** [*simp*]: *euclidean-size* $x$ = (*case* $x$ *of* $a_1 \Rightarrow 0$ | - $\Rightarrow 1$)
**definition** [*simp*]: *division-segment* ($x$ :: *finite-3*) = *1*
**instance**
⟨*proof*⟩
**end**

**hide-const** (**open**) $a_1$ $a_2$ $a_3$

**datatype** (*plugins only*: *code quickcheck extraction*) *finite-4* =
$a_1$ | $a_2$ | $a_3$ | $a_4$

**notation** (**output**) $a_1$ (‹$a_1$›)
**notation** (**output**) $a_2$ (‹$a_2$›)
**notation** (**output**) $a_3$ (‹$a_3$›)
**notation** (**output**) $a_4$ (‹$a_4$›)

**lemma** *UNIV-finite-4*:
*UNIV* = $\{a_1, a_2, a_3, a_4\}$
⟨*proof*⟩

**instantiation** *finite-4* :: *enum*
**begin**

**definition**
*enum* = $[a_1, a_2, a_3, a_4]$

**definition**
*enum-all* $P \longleftrightarrow P\ a_1 \land P\ a_2 \land P\ a_3 \land P\ a_4$

**definition**
*enum-ex* $P \longleftrightarrow P\ a_1 \lor P\ a_2 \lor P\ a_3 \lor P\ a_4$

**instance** ⟨*proof*⟩

**end**

**instantiation** *finite-4* :: *finite-distrib-lattice* **begin**

$a_1 < a_2$,$a_3 < a_4$, but $a_2$ and $a_3$ are incomparable.

**definition**
$x < y \longleftrightarrow (case\ (x,\ y)\ of$
$(a_1,\ a_1) \Rightarrow False \mid (a_1,\ \text{-}) \Rightarrow True$
$\mid\ (a_2,\ a_4) \Rightarrow True$
$\mid\ (a_3,\ a_4) \Rightarrow True\ \mid \text{-} \Rightarrow False)$

**definition**
$x \le y \longleftrightarrow (case\ (x,\ y)\ of$
$(a_1,\ \text{-}) \Rightarrow True$
$\mid (a_2,\ a_2) \Rightarrow True \mid (a_2,\ a_4) \Rightarrow True$
$\mid (a_3,\ a_3) \Rightarrow True \mid (a_3,\ a_4) \Rightarrow True$
$\mid (a_4,\ a_4) \Rightarrow True \mid \text{-} \Rightarrow False)$

**definition**
$\bigsqcap A = (if\ a_1 \in A \lor a_2 \in A \land a_3 \in A\ then\ a_1\ else\ if\ a_2 \in A\ then\ a_2\ else\ if\ a_3 \in A\ then\ a_3\ else\ a_4)$
**definition**
$\bigsqcup A = (if\ a_4 \in A \lor a_2 \in A \land a_3 \in A\ then\ a_4\ else\ if\ a_2 \in A\ then\ a_2\ else\ if\ a_3 \in A\ then\ a_3\ else\ a_1)$
**definition** [*simp*]: $bot = a_1$
**definition** [*simp*]: $top = a_4$
**definition**
$x \sqcap y = (case\ (x,\ y)\ of$
$(a_1,\ \text{-}) \Rightarrow a_1 \mid (\text{-},\ a_1) \Rightarrow a_1 \mid (a_2,\ a_3) \Rightarrow a_1 \mid (a_3,\ a_2) \Rightarrow a_1$
$\mid (a_2,\ \text{-}) \Rightarrow a_2 \mid (\text{-},\ a_2) \Rightarrow a_2$
$\mid (a_3,\ \text{-}) \Rightarrow a_3 \mid (\text{-},\ a_3) \Rightarrow a_3$
$\mid \text{-} \Rightarrow a_4)$
**definition**
$x \sqcup y = (case\ (x,\ y)\ of$
$(a_4,\ \text{-}) \Rightarrow a_4 \mid (\text{-},\ a_4) \Rightarrow a_4 \mid (a_2,\ a_3) \Rightarrow a_4 \mid (a_3,\ a_2) \Rightarrow a_4$
$\mid (a_2,\ \text{-}) \Rightarrow a_2 \mid (\text{-},\ a_2) \Rightarrow a_2$
$\mid (a_3,\ \text{-}) \Rightarrow a_3 \mid (\text{-},\ a_3) \Rightarrow a_3$
$\mid \text{-} \Rightarrow a_1)$

**instance**
⟨*proof*⟩
**end**

**instance** *finite-4* :: *complete-lattice* ⟨*proof*⟩

**instance** *finite-4* :: *complete-distrib-lattice* ⟨*proof*⟩

**instantiation** *finite-4* :: *complete-boolean-algebra* **begin**
**definition** $- x = (case\ x\ of\ a_1 \Rightarrow a_4 \mid a_2 \Rightarrow a_3 \mid a_3 \Rightarrow a_2 \mid a_4 \Rightarrow a_1)$
**definition** $x - y = x \sqcap - (y :: \textit{finite-4})$
**instance**
⟨*proof*⟩
**end**

**hide-const** (**open**) $a_1$ $a_2$ $a_3$ $a_4$

**datatype** (*plugins only*: *code quickcheck extraction*) *finite-5* =
$a_1 \mid a_2 \mid a_3 \mid a_4 \mid a_5$

**notation** (**output**) $a_1$ (‹$a_1$›)
**notation** (**output**) $a_2$ (‹$a_2$›)
**notation** (**output**) $a_3$ (‹$a_3$›)
**notation** (**output**) $a_4$ (‹$a_4$›)
**notation** (**output**) $a_5$ (‹$a_5$›)

**lemma** *UNIV-finite-5*:
$UNIV = \{a_1, a_2, a_3, a_4, a_5\}$
⟨*proof*⟩

**instantiation** *finite-5* :: *enum*
**begin**

**definition**
$enum = [a_1, a_2, a_3, a_4, a_5]$

**definition**
$\textit{enum-all}\ P \longleftrightarrow P\ a_1 \wedge P\ a_2 \wedge P\ a_3 \wedge P\ a_4 \wedge P\ a_5$

**definition**
$\textit{enum-ex}\ P \longleftrightarrow P\ a_1 \vee P\ a_2 \vee P\ a_3 \vee P\ a_4 \vee P\ a_5$

**instance** ⟨*proof*⟩

**end**

**instantiation** *finite-5* :: *finite-lattice*
**begin**

The non-distributive pentagon lattice $N_5$

**definition**
$x < y \longleftrightarrow (case\ (x, y)\ of$
$(a_1, a_1) \Rightarrow False \mid (a_1, \text{-}) \Rightarrow True$
$\mid (a_2, a_3) \Rightarrow True \mid (a_2, a_5) \Rightarrow True$
$\mid (a_3, a_5) \Rightarrow True$
$\mid (a_4, a_5) \Rightarrow True \mid \text{-} \Rightarrow False)$

**definition**
$x \leq y \longleftrightarrow (case\ (x,\ y)\ of$
$(a_1,\ \text{-}) \Rightarrow True$
$|\ (a_2,\ a_2) \Rightarrow True\ |\ (a_2,\ a_3) \Rightarrow True\ |\ (a_2,\ a_5) \Rightarrow True$
$|\ (a_3,\ a_3) \Rightarrow True\ |\ (a_3,\ a_5) \Rightarrow True$
$|\ (a_4,\ a_4) \Rightarrow True\ |\ (a_4,\ a_5) \Rightarrow True$
$|\ (a_5,\ a_5) \Rightarrow True\ |\ \text{-} \Rightarrow False)$

**definition**
$\bigsqcap A =$
$(if\ a_1 \in A \lor a_4 \in A \land (a_2 \in A \lor a_3 \in A)\ then\ a_1$
$else\ if\ a_2 \in A\ then\ a_2$
$else\ if\ a_3 \in A\ then\ a_3$
$else\ if\ a_4 \in A\ then\ a_4$
$else\ a_5)$
**definition**
$\bigsqcup A =$
$(if\ a_5 \in A \lor a_4 \in A \land (a_2 \in A \lor a_3 \in A)\ then\ a_5$
$else\ if\ a_3 \in A\ then\ a_3$
$else\ if\ a_2 \in A\ then\ a_2$
$else\ if\ a_4 \in A\ then\ a_4$
$else\ a_1)$
**definition** [*simp*]: $bot = a_1$
**definition** [*simp*]: $top = a_5$
**definition**
$x \sqcap y = (case\ (x,\ y)\ of$
$(a_1,\ \text{-}) \Rightarrow a_1\ |\ (\text{-},\ a_1) \Rightarrow a_1\ |\ (a_2,\ a_4) \Rightarrow a_1\ |\ (a_4,\ a_2) \Rightarrow a_1\ |\ (a_3,\ a_4) \Rightarrow a_1\ |$
$(a_4,\ a_3) \Rightarrow a_1$
$|\ (a_2,\ \text{-}) \Rightarrow a_2\ |\ (\text{-},\ a_2) \Rightarrow a_2$
$|\ (a_3,\ \text{-}) \Rightarrow a_3\ |\ (\text{-},\ a_3) \Rightarrow a_3$
$|\ (a_4,\ \text{-}) \Rightarrow a_4\ |\ (\text{-},\ a_4) \Rightarrow a_4$
$|\ \text{-} \Rightarrow a_5)$
**definition**
$x \sqcup y = (case\ (x,\ y)\ of$
$(a_5,\ \text{-}) \Rightarrow a_5\ |\ (\text{-},\ a_5) \Rightarrow a_5\ |\ (a_2,\ a_4) \Rightarrow a_5\ |\ (a_4,\ a_2) \Rightarrow a_5\ |\ (a_3,\ a_4) \Rightarrow a_5\ |$
$(a_4,\ a_3) \Rightarrow a_5$
$|\ (a_3,\ \text{-}) \Rightarrow a_3\ |\ (\text{-},\ a_3) \Rightarrow a_3$
$|\ (a_2,\ \text{-}) \Rightarrow a_2\ |\ (\text{-},\ a_2) \Rightarrow a_2$
$|\ (a_4,\ \text{-}) \Rightarrow a_4\ |\ (\text{-},\ a_4) \Rightarrow a_4$
$|\ \text{-} \Rightarrow a_1)$

**instance**
⟨*proof*⟩
**end**

**instance** *finite-5* :: *complete-lattice* ⟨*proof*⟩

**hide-const** (**open**) $a_1$ $a_2$ $a_3$ $a_4$ $a_5$

### 72.5 Closing up

**hide-type** (**open**) *finite-1 finite-2 finite-3 finite-4 finite-5*
**hide-const** (**open**) *enum enum-all enum-ex all-n-lists ex-n-lists ntrancl*

**end**

# 73 Character and string types

**theory** *String*
**imports** *Enum Bit-Operations Code-Numeral*
**begin**

## 73.1 Strings as list of bytes

When modelling strings, we follow the approach given in https://utf8everywhere.org/:

- Strings are a list of bytes (8 bit).
- Byte values from 0 to 127 are US-ASCII.
- Byte values from 128 to 255 are uninterpreted blobs.

### 73.1.1 Bytes as datatype

**datatype** *char* =
  *Char* (*digit0*: *bool*) (*digit1*: *bool*) (*digit2*: *bool*) (*digit3*: *bool*)
    (*digit4*: *bool*) (*digit5*: *bool*) (*digit6*: *bool*) (*digit7*: *bool*)

**context** *comm-semiring-1*
**begin**

**definition** *of-char* :: ‹*char* ⇒ *'a*›
  **where** ‹*of-char c* = *horner-sum of-bool 2* [*digit0 c*, *digit1 c*, *digit2 c*, *digit3 c*, *digit4 c*, *digit5 c*, *digit6 c*, *digit7 c*]›

**lemma** *of-char-Char* [*simp*]:
  ‹*of-char* (*Char b0 b1 b2 b3 b4 b5 b6 b7*) =
    *horner-sum of-bool 2* [*b0*, *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7*]›
  ⟨*proof*⟩

**end**

**lemma** (**in** *comm-semiring-1*) *of-nat-of-char*:

‹*of-nat* (*of-char c*) = *of-char c*›
⟨*proof*⟩

**lemma** (**in** *comm-ring-1*) *of-int-of-char*:
‹*of-int* (*of-char c*) = *of-char c*›
⟨*proof*⟩

**lemma** *nat-of-char* [*simp*]:
‹*nat* (*of-char c*) = *of-char c*›
⟨*proof*⟩

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**definition** *char-of* :: ‹*'a* ⇒ *char*›
**where** ‹*char-of n* = *Char* (*bit n 0*) (*bit n 1*) (*bit n 2*) (*bit n 3*) (*bit n 4*) (*bit n 5*) (*bit n 6*) (*bit n 7*)›

**lemma** *char-of-take-bit-eq*:
‹*char-of* (*take-bit n m*) = *char-of m*› **if** ‹$n \geq 8$›
⟨*proof*⟩

**lemma** *char-of-char* [*simp*]:
‹*char-of* (*of-char c*) = *c*›
⟨*proof*⟩

**lemma** *char-of-comp-of-char* [*simp*]:
*char-of* ∘ *of-char* = *id*
⟨*proof*⟩

**lemma** *inj-of-char*:
‹*inj of-char*›
⟨*proof*⟩

**lemma** *of-char-eqI*:
‹*c* = *d*› **if** ‹*of-char c* = *of-char d*›
⟨*proof*⟩

**lemma** *of-char-eq-iff* [*simp*]:
‹*of-char c* = *of-char d* ⟷ *c* = *d*›
⟨*proof*⟩

**lemma** *of-char-of* [*simp*]:
‹*of-char* (*char-of a*) = *a mod 256*›
⟨*proof*⟩

**lemma** *char-of-mod-256* [*simp*]:
‹*char-of* (*n mod 256*) = *char-of n*›

⟨proof⟩

**lemma** *of-char-mod-256* [*simp*]:
‹*of-char c mod 256 = of-char c*›
⟨proof⟩

**lemma** *char-of-quasi-inj* [*simp*]:
‹*char-of m = char-of n* ⟷ *m mod 256 = n mod 256*› (**is** ‹*?P* ⟷ *?Q*›)
⟨proof⟩

**lemma** *char-of-eq-iff*:
‹*char-of n = c* ⟷ *take-bit 8 n = of-char c*›
⟨proof⟩

**lemma** *char-of-nat* [*simp*]:
‹*char-of (of-nat n) = char-of n*›
⟨proof⟩

**end**

**lemma** *inj-on-char-of-nat* [*simp*]:
*inj-on char-of* {*0::nat..<256*}
⟨proof⟩

**lemma** *nat-of-char-less-256* [*simp*]:
*of-char c* < (*256* :: *nat*)
⟨proof⟩

**lemma** *range-nat-of-char*:
*range of-char* = {*0::nat..<256*}
⟨proof⟩

**lemma** *UNIV-char-of-nat*:
*UNIV* = *char-of* ‘ {*0::nat..<256*}
⟨proof⟩

**lemma** *card-UNIV-char*:
*card* (*UNIV* :: *char set*) = *256*
⟨proof⟩

**context**
**includes** *lifting-syntax* **and** *integer.lifting* **and** *natural.lifting*
**begin**

**lemma** [*transfer-rule*]:
‹(*pcr-integer* ===> (=)) *char-of char-of*›
⟨proof⟩

**lemma** [*transfer-rule*]:

‹((=) ===> *pcr-integer*) *of-char of-char*›
⟨*proof*⟩

**lemma** [*transfer-rule*]:
‹(*pcr-natural* ===> (=)) *char-of char-of*›
⟨*proof*⟩

**lemma** [*transfer-rule*]:
‹((=) ===> *pcr-natural*) *of-char of-char*›
⟨*proof*⟩

**end**

**lifting-update** *integer.lifting*
**lifting-forget** *integer.lifting*

**lifting-update** *natural.lifting*
**lifting-forget** *natural.lifting*

**lemma** *size-char-eq-0* [*simp*, *code*]:
‹*size c* = *0*› **for** *c* :: *char*
⟨*proof*⟩

**lemma** *size′-char-eq-0* [*simp*, *code*]:
‹*size-char c* = *0*›
⟨*proof*⟩

**syntax**
*-Char* :: *str-position* ⇒ *char* (‹(‹*open-block notation*=‹*literal char*››*CHR* -)›)
*-Char-ord* :: *num-const* ⇒ *char* (‹(‹*open-block notation*=‹*literal char code*››*CHR* -)›)
**syntax-consts**
*-Char -Char-ord* ⇌ *Char*

**type-synonym** *string* = *char list*

**syntax**
*-String* :: *str-position* ⇒ *string* (‹(‹*open-block notation*=‹*literal string*››-)›)

⟨*ML*⟩

**instantiation** *char* :: *enum*
**begin**

**definition**
*Enum.enum* = [
*CHR 0x00*, *CHR 0x01*, *CHR 0x02*, *CHR 0x03*,
*CHR 0x04*, *CHR 0x05*, *CHR 0x06*, *CHR 0x07*,
*CHR 0x08*, *CHR 0x09*, *CHR* ''$\boxed{\hookleftarrow}$'', *CHR 0x0B*,

CHR 0x0C, CHR 0x0D, CHR 0x0E, CHR 0x0F,
CHR 0x10, CHR 0x11, CHR 0x12, CHR 0x13,
CHR 0x14, CHR 0x15, CHR 0x16, CHR 0x17,
CHR 0x18, CHR 0x19, CHR 0x1A, CHR 0x1B,
CHR 0x1C, CHR 0x1D, CHR 0x1E, CHR 0x1F,
CHR " ", CHR "!", CHR 0x22, CHR "#",
CHR "$", CHR "%", CHR "&", CHR 0x27,
CHR "(", CHR ")", CHR "*", CHR "+",
CHR ",", CHR "−", CHR ".", CHR "/",
CHR "0", CHR "1", CHR "2", CHR "3",
CHR "4", CHR "5", CHR "6", CHR "7",
CHR "8", CHR "9", CHR ":", CHR ";",
CHR "<", CHR "=", CHR ">", CHR "?",
CHR "@", CHR "A", CHR "B", CHR "C",
CHR "D", CHR "E", CHR "F", CHR "G",
CHR "H", CHR "I", CHR "J", CHR "K",
CHR "L", CHR "M", CHR "N", CHR "O",
CHR "P", CHR "Q", CHR "R", CHR "S",
CHR "T", CHR "U", CHR "V", CHR "W",
CHR "X", CHR "Y", CHR "Z", CHR "[",
CHR 0x5C, CHR "]", CHR "^", CHR "-",
CHR 0x60, CHR "a", CHR "b", CHR "c",
CHR "d", CHR "e", CHR "f", CHR "g",
CHR "h", CHR "i", CHR "j", CHR "k",
CHR "l", CHR "m", CHR "n", CHR "o",
CHR "p", CHR "q", CHR "r", CHR "s",
CHR "t", CHR "u", CHR "v", CHR "w",
CHR "x", CHR "y", CHR "z", CHR "{",
CHR "|", CHR "}", CHR "~", CHR 0x7F,
CHR 0x80, CHR 0x81, CHR 0x82, CHR 0x83,
CHR 0x84, CHR 0x85, CHR 0x86, CHR 0x87,
CHR 0x88, CHR 0x89, CHR 0x8A, CHR 0x8B,
CHR 0x8C, CHR 0x8D, CHR 0x8E, CHR 0x8F,
CHR 0x90, CHR 0x91, CHR 0x92, CHR 0x93,
CHR 0x94, CHR 0x95, CHR 0x96, CHR 0x97,
CHR 0x98, CHR 0x99, CHR 0x9A, CHR 0x9B,
CHR 0x9C, CHR 0x9D, CHR 0x9E, CHR 0x9F,
CHR 0xA0, CHR 0xA1, CHR 0xA2, CHR 0xA3,
CHR 0xA4, CHR 0xA5, CHR 0xA6, CHR 0xA7,
CHR 0xA8, CHR 0xA9, CHR 0xAA, CHR 0xAB,
CHR 0xAC, CHR 0xAD, CHR 0xAE, CHR 0xAF,
CHR 0xB0, CHR 0xB1, CHR 0xB2, CHR 0xB3,
CHR 0xB4, CHR 0xB5, CHR 0xB6, CHR 0xB7,
CHR 0xB8, CHR 0xB9, CHR 0xBA, CHR 0xBB,
CHR 0xBC, CHR 0xBD, CHR 0xBE, CHR 0xBF,
CHR 0xC0, CHR 0xC1, CHR 0xC2, CHR 0xC3,
CHR 0xC4, CHR 0xC5, CHR 0xC6, CHR 0xC7,
CHR 0xC8, CHR 0xC9, CHR 0xCA, CHR 0xCB,
CHR 0xCC, CHR 0xCD, CHR 0xCE, CHR 0xCF,

*CHR 0xD0*, *CHR 0xD1*, *CHR 0xD2*, *CHR 0xD3*,
*CHR 0xD4*, *CHR 0xD5*, *CHR 0xD6*, *CHR 0xD7*,
*CHR 0xD8*, *CHR 0xD9*, *CHR 0xDA*, *CHR 0xDB*,
*CHR 0xDC*, *CHR 0xDD*, *CHR 0xDE*, *CHR 0xDF*,
*CHR 0xE0*, *CHR 0xE1*, *CHR 0xE2*, *CHR 0xE3*,
*CHR 0xE4*, *CHR 0xE5*, *CHR 0xE6*, *CHR 0xE7*,
*CHR 0xE8*, *CHR 0xE9*, *CHR 0xEA*, *CHR 0xEB*,
*CHR 0xEC*, *CHR 0xED*, *CHR 0xEE*, *CHR 0xEF*,
*CHR 0xF0*, *CHR 0xF1*, *CHR 0xF2*, *CHR 0xF3*,
*CHR 0xF4*, *CHR 0xF5*, *CHR 0xF6*, *CHR 0xF7*,
*CHR 0xF8*, *CHR 0xF9*, *CHR 0xFA*, *CHR 0xFB*,
*CHR 0xFC*, *CHR 0xFD*, *CHR 0xFE*, *CHR 0xFF*]

**definition**
*Enum.enum-all P* ⟷ *list-all P* (*Enum.enum* :: *char list*)

**definition**
*Enum.enum-ex P* ⟷ *list-ex P* (*Enum.enum* :: *char list*)

**lemma** *enum-char-unfold*:
*Enum.enum* = *map char-of* [*0*..<*256*]
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**lemma** *linorder-char*:
*class.linorder* (λ*c d*. *of-char c* ≤ (*of-char d* :: *nat*)) (λ*c d*. *of-char c* < (*of-char d* :: *nat*))
⟨*proof*⟩

Optimized version for execution

**definition** *char-of-integer* :: *integer* ⇒ *char*
**where** [*code-abbrev*]: *char-of-integer* = *char-of*

**definition** *integer-of-char* :: *char* ⇒ *integer*
**where** [*code-abbrev*]: *integer-of-char* = *of-char*

**lemma** *char-of-integer-code* [*code*]:
*char-of-integer k* = (*let*
(*q0*, *b0*) = *bit-cut-integer k*;
(*q1*, *b1*) = *bit-cut-integer q0*;
(*q2*, *b2*) = *bit-cut-integer q1*;
(*q3*, *b3*) = *bit-cut-integer q2*;
(*q4*, *b4*) = *bit-cut-integer q3*;
(*q5*, *b5*) = *bit-cut-integer q4*;
(*q6*, *b6*) = *bit-cut-integer q5*;
(-, *b7*) = *bit-cut-integer q6*

  *in Char b0 b1 b2 b3 b4 b5 b6 b7*)
⟨*proof*⟩

**lemma** *integer-of-char-code* [*code*]:
*integer-of-char* (*Char b0 b1 b2 b3 b4 b5 b6 b7*) =
((((((*of-bool b7* ∗ *2* + *of-bool b6*) ∗ *2* +
*of-bool b5*) ∗ *2* + *of-bool b4*) ∗ *2* +
*of-bool b3*) ∗ *2* + *of-bool b2*) ∗ *2* +
*of-bool b1*) ∗ *2* + *of-bool b0*
⟨*proof*⟩

## 73.2 Strings as dedicated type for target language code generation

### 73.2.1 Logical specification

**context**
**begin**

**qualified definition** *ascii-of* :: *char* ⇒ *char*
**where** *ascii-of c* = *Char* (*digit0 c*) (*digit1 c*) (*digit2 c*) (*digit3 c*) (*digit4 c*) (*digit5 c*) (*digit6 c*) *False*

**qualified lemma** *ascii-of-Char* [*simp*]:
*ascii-of* (*Char b0 b1 b2 b3 b4 b5 b6 b7*) = *Char b0 b1 b2 b3 b4 b5 b6 False*
⟨*proof*⟩ **lemma** *digit0-ascii-of-iff* [*simp*]:
*digit0* (*String.ascii-of c*) ⟷ *digit0 c*
⟨*proof*⟩ **lemma** *digit1-ascii-of-iff* [*simp*]:
*digit1* (*String.ascii-of c*) ⟷ *digit1 c*
⟨*proof*⟩ **lemma** *digit2-ascii-of-iff* [*simp*]:
*digit2* (*String.ascii-of c*) ⟷ *digit2 c*
⟨*proof*⟩ **lemma** *digit3-ascii-of-iff* [*simp*]:
*digit3* (*String.ascii-of c*) ⟷ *digit3 c*
⟨*proof*⟩ **lemma** *digit4-ascii-of-iff* [*simp*]:
*digit4* (*String.ascii-of c*) ⟷ *digit4 c*
⟨*proof*⟩ **lemma** *digit5-ascii-of-iff* [*simp*]:
*digit5* (*String.ascii-of c*) ⟷ *digit5 c*
⟨*proof*⟩ **lemma** *digit6-ascii-of-iff* [*simp*]:
*digit6* (*String.ascii-of c*) ⟷ *digit6 c*
⟨*proof*⟩ **lemma** *not-digit7-ascii-of* [*simp*]:
¬ *digit7* (*ascii-of c*)
⟨*proof*⟩ **lemma** *ascii-of-idem*:
*ascii-of c* = *c* **if** ¬ *digit7 c*
⟨*proof*⟩ **typedef** *literal* = {*cs*. ∀ *c*∈*set cs*. ¬ *digit7 c*}
**morphisms** *explode Abs-literal*
⟨*proof*⟩ **setup-lifting** *type-definition-literal*

**qualified lift-definition** *implode* :: *string* ⇒ *literal*
**is** *map ascii-of*
⟨*proof*⟩ **lemma** *implode-explode-eq* [*simp*]:

*String.implode* (*String.explode s*) = *s*
⟨*proof*⟩ **lemma** *explode-implode-eq* [*simp*]:
*String.explode* (*String.implode cs*) = *map ascii-of cs*
⟨*proof*⟩

**end**

**context** *linordered-euclidean-semiring-bit-operations*
**begin**

**context**
**begin**

**qualified lemma** *char-of-ascii-of* [*simp*]:
‹*of-char* (*String.ascii-of c*) = *take-bit 7* (*of-char c*)›
⟨*proof*⟩ **lemma** *ascii-of-char-of*:
‹*String.ascii-of* (*char-of a*) = *char-of* (*take-bit 7 a*)›
⟨*proof*⟩

**end**

**end**

### 73.2.2 Syntactic representation

Logical ground representations for literals are:

1. *0* for the empty literal;
2. *Literal b0 ... b6 s* for a literal starting with one character and continued by another literal.

   Syntactic representations for literals are:

3. Printable text as string prefixed with *STR*;
4. A single ascii value as numerical hexadecimal value prefixed with *STR*.

**instantiation** *String.literal* :: *zero*
**begin**

**context**
**begin**

**qualified lift-definition** *zero-literal* :: *String.literal*
**is** *Nil*
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**end**

**context**
**begin**

**qualified abbreviation** (**output**) *empty-literal* :: *String.literal*
**where** *empty-literal* ≡ *0*

**qualified lift-definition** *Literal* :: *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *String.literal* ⇒ *String.literal*
**is** *λb0 b1 b2 b3 b4 b5 b6 cs. Char b0 b1 b2 b3 b4 b5 b6 False # cs*
⟨*proof*⟩ **lemma** *Literal-eq-iff* [*simp*]:
*Literal b0 b1 b2 b3 b4 b5 b6 s* = *Literal c0 c1 c2 c3 c4 c5 c6 t*
⟷ (*b0* ⟷ *c0*) ∧ (*b1* ⟷ *c1*) ∧ (*b2* ⟷ *c2*) ∧ (*b3* ⟷ *c3*)
∧ (*b4* ⟷ *c4*) ∧ (*b5* ⟷ *c5*) ∧ (*b6* ⟷ *c6*) ∧ *s* = *t*
⟨*proof*⟩ **lemma** *empty-neq-Literal* [*simp*]:
*empty-literal* ≠ *Literal b0 b1 b2 b3 b4 b5 b6 s*
⟨*proof*⟩ **lemma** *Literal-neq-empty* [*simp*]:
*Literal b0 b1 b2 b3 b4 b5 b6 s* ≠ *empty-literal*
⟨*proof*⟩

**end**

**code-datatype** *0* :: *String.literal String.Literal*

**syntax**
*-Literal* :: *str-position* ⇒ *String.literal*
(‹(‹*open-block notation*=‹*literal string*››*STR* -)›)
*-Ascii* :: *num-const* ⇒ *String.literal*
(‹(‹*open-block notation*=‹*literal char code*››*STR* -)›)
**syntax-consts**
*-Literal -Ascii* ⇌ *String.Literal*

⟨*ML*⟩

### 73.2.3 Operations

**instantiation** *String.literal* :: *plus*
**begin**

**context**
**begin**

**qualified lift-definition** *plus-literal* :: *String.literal* ⇒ *String.literal* ⇒ *String.literal*
**is** (@)
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**end**

**instance** *String.literal* :: *monoid-add*
⟨*proof*⟩

**lemma** *add-Literal-assoc*:
‹*String.Literal b0 b1 b2 b3 b4 b5 b6 t* + *s* = *String.Literal b0 b1 b2 b3 b4 b5 b6* (*t* + *s*)›
⟨*proof*⟩

**instantiation** *String.literal* :: *size*
**begin**

**context**
**includes** *literal.lifting*
**begin**

**lift-definition** *size-literal* :: *String.literal* ⇒ *nat*
**is** *length* ⟨*proof*⟩

**end**

**instance** ⟨*proof*⟩

**end**

**instantiation** *String.literal* :: *equal*
**begin**

**context**
**begin**

**qualified lift-definition** *equal-literal* :: *String.literal* ⇒ *String.literal* ⇒ *bool*
**is** *HOL.equal* ⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**end**

**instantiation** *String.literal* :: *linorder*
**begin**

**context**
**begin**

**qualified lift-definition** *less-eq-literal* :: *String.literal* ⇒ *String.literal* ⇒ *bool*
  **is** *ord.lexordp-eq* ($\lambda c\ d.$ *of-char* $c <$ (*of-char* $d$ :: *nat*))
  ⟨*proof*⟩ **lift-definition** *less-literal* :: *String.literal* ⇒ *String.literal* ⇒ *bool*
  **is** *ord.lexordp* ($\lambda c\ d.$ *of-char* $c <$ (*of-char* $d$ :: *nat*))
  ⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**end**

**lemma** *infinite-literal*:
  *infinite* (*UNIV* :: *String.literal set*)
⟨*proof*⟩

**lemma** *add-literal-code* [*code*]:
  ‹*STR* '''' + *s* = *s*›
  ‹*s* + *STR* '''' = *s*›
  ‹*String.Literal b0 b1 b2 b3 b4 b5 b6 t* + *s* = *String.Literal b0 b1 b2 b3 b4 b5 b6* (*t* + *s*)›
  ⟨*proof*⟩

### 73.2.4 Executable conversions

**context**
**begin**

**qualified lift-definition** *asciis-of-literal* :: *String.literal* ⇒ *integer list*
  **is** *map of-char*
  ⟨*proof*⟩ **lemma** *asciis-of-zero* [*simp*, *code*]:
  *asciis-of-literal 0* = []
  ⟨*proof*⟩ **lemma** *asciis-of-Literal* [*simp*, *code*]:
  *asciis-of-literal* (*String.Literal b0 b1 b2 b3 b4 b5 b6 s*) =
    *of-char* (*Char b0 b1 b2 b3 b4 b5 b6 False*) # *asciis-of-literal s*
  ⟨*proof*⟩ **lift-definition** *literal-of-asciis* :: *integer list* ⇒ *String.literal*
  **is** *map* (*String.ascii-of* ∘ *char-of*)
  ⟨*proof*⟩ **lemma** *literal-of-asciis-Nil* [*simp*, *code*]:
  *literal-of-asciis* [] = *0*
  ⟨*proof*⟩ **lemma** *literal-of-asciis-Cons* [*simp*, *code*]:
  *literal-of-asciis* (*k* # *ks*) = (*case char-of k*
    *of Char b0 b1 b2 b3 b4 b5 b6 b7* ⇒ *String.Literal b0 b1 b2 b3 b4 b5 b6* (*literal-of-asciis ks*))
  ⟨*proof*⟩ **lemma** *literal-of-asciis-of-literal* [*simp*]:
  *literal-of-asciis* (*asciis-of-literal s*) = *s*

⟨*proof*⟩ **lemma** *explode-code* [*code*]:
*String.explode s = map char-of (asciis-of-literal s)*
⟨*proof*⟩ **lemma** *implode-code* [*code*]:
*String.implode cs = literal-of-asciis (map of-char cs)*
⟨*proof*⟩ **lemma** *equal-literal* [*code*]:
*HOL.equal (String.Literal b0 b1 b2 b3 b4 b5 b6 s)*
*(String.Literal a0 a1 a2 a3 a4 a5 a6 r)*
$\longleftrightarrow$ $(b0 \longleftrightarrow a0) \wedge (b1 \longleftrightarrow a1) \wedge (b2 \longleftrightarrow a2) \wedge (b3 \longleftrightarrow a3)$
$\wedge\ (b4 \longleftrightarrow a4) \wedge (b5 \longleftrightarrow a5) \wedge (b6 \longleftrightarrow a6) \wedge (s = r)$
⟨*proof*⟩

**end**

### 73.2.5 Technical code generation setup

Alternative constructor for generated computations

**context**
**begin**

**qualified definition** *Literal′* :: *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *String.literal* ⇒ *String.literal*
**where** [*simp*]: *Literal′ = String.Literal*

**lemma** [*code*]:
‹*Literal′ b0 b1 b2 b3 b4 b5 b6 s = String.literal-of-asciis*
[*foldr* ($\lambda b\ k.$ *of-bool* $b + k * 2$) [*b0*, *b1*, *b2*, *b3*, *b4*, *b5*, *b6*] *0*] + *s*›
⟨*proof*⟩

**lemma** [*code-computation-unfold*]:
*String.Literal = Literal′*
⟨*proof*⟩

**end**

**code-reserved**
(*SML*) *string String Char Str-Literal*
**and** (*OCaml*) *string String Char Str-Literal*
**and** (*Haskell*) *Str-Literal*
**and** (*Scala*) *String Str-Literal*

**code-identifier**
**code-module** *String* ⇀
(*SML*) *Str* **and** (*OCaml*) *Str* **and** (*Haskell*) *Str* **and** (*Scala*) *Str*

**code-printing**
**type-constructor** *String.literal* ⇀
(*SML*) *string*
**and** (*OCaml*) *string*
**and** (*Haskell*) *String*

**and** (*Scala*) *String*
| **constant** *STR* *''''* ⇀
(*SML*)
**and** (*OCaml*)
**and** (*Haskell*)
**and** (*Scala*)

⟨*ML*⟩

**code-printing**
**code-module** *Str-Literal* ⇀
(*SML*) ‹*structure Str-Literal* : *sig*
*type int* = *IntInf.int*
*val literal-of-asciis* : *int list* −> *string*
*val asciis-of-literal* : *string* −> *int list*
*end* = *struct*

*open IntInf*;

*fun map f* [] = []
| *map f* (*x* :: *xs*) = *f x* :: *map f xs*; (∗ *deliberate clone not relying on List.- structure* ∗)

*fun check-ascii k* =
*if 0* <= *k andalso k* < *128*
*then k*
*else raise Fail Non*−*ASCII character in literal*;

*val char-of-ascii* = *Char.chr o toInt o* (*fn k* => *k mod 128*);

*val ascii-of-char* = *check-ascii o fromInt o Char.ord*;

*val literal-of-asciis* = *String.implode o map char-of-ascii*;

*val asciis-of-literal* = *map ascii-of-char o String.explode*;

*end*;› **for constant** *String.literal-of-asciis String.asciis-of-literal*
**and** (*OCaml*) ‹*module Str-Literal* : *sig*
*val literal-of-asciis* : *Z.t list* −> *string*
*val asciis-of-literal*: *string* −> *Z.t list*
*end* = *struct*

(∗ *deliberate clones not relying on List.- module* ∗)

*let rec length xs* = *match xs with*
[] −> *0*
| *x* :: *xs* −> *1* + *length xs*;;

*let rec nth xs n* = *match xs with*

```
  (x :: xs) -> if n <= 0 then x else nth xs (n - 1);;

let rec map-range f n =
  if n <= 0
    then []
    else
      let m = n - 1
    in map-range f m @ [f m];;

let implode f xs =
  String.init (length xs) (fun n -> f (nth xs n));;

let explode f s =
  map-range (fun n -> f (String.get s n)) (String.length s);;

let z-128 = Z.of-int 128;;

let check-ascii k =
  if 0 <= k && k < 128
  then k
  else failwith Non-ASCII character in literal;;

let char-of-ascii k = Char.chr (Z.to-int (Z.rem k z-128));;

let ascii-of-char c = Z.of-int (check-ascii (Char.code c));;

let literal-of-asciis ks = implode char-of-ascii ks;;

let asciis-of-literal s = explode ascii-of-char s;;
```

*end;;›* **for constant** *String.literal-of-asciis String.asciis-of-literal*
**and** (*Haskell*) ‹*module Str-Literal*(*literalOfAsciis*, *asciisOfLiteral*) *where*

```
check-ascii :: Int -> Int
check-ascii k
  | (0 <= k && k < 128) = k
  | otherwise = error Non-ASCII character in literal

charOfAscii :: Integer -> Char
charOfAscii = toEnum . Prelude.fromInteger . (\k -> k `mod ` 128)

asciiOfChar :: Char -> Integer
asciiOfChar = toInteger . check-ascii . fromEnum

literalOfAsciis :: [Integer] -> [Char]
literalOfAsciis = map charOfAscii

asciisOfLiteral :: [Char] -> [Integer]
asciisOfLiteral = map asciiOfChar
```

› **for constant** *String.literal-of-asciis String.asciis-of-literal*
**and** (*Scala*) ‹*object Str-Literal* {

```
private def checkAscii(k : Int) : Int =
  0 <= k && k < 128 match {
    case true => k
    case false => sys.error(Non−ASCII character in literal)
  }

private def charOfAscii(k : BigInt) : Char =
  (k % 128).charValue

private def asciiOfChar(c : Char) : BigInt =
  BigInt(checkAscii(c.toInt))

def literalOfAsciis(ks : List[BigInt]) : String =
  ks.map(charOfAscii).mkString

def asciisOfLiteral(s : String) : List[BigInt] =
  s.toList.map(asciiOfChar)

}
```

› **for constant** *String.literal-of-asciis String.asciis-of-literal*
| **constant** ‹(+) :: *String.literal* ⇒ *String.literal* ⇒ *String.literal*› ⇀
(*SML*) **infixl** *18* ^
**and** (*OCaml*) **infixr** *6* ^
**and** (*Haskell*) **infixr** *5* ++
**and** (*Scala*) **infixl** *7* +
| **constant** *String.literal-of-asciis* ⇀
(*SML*) *Str′-Literal.literal′-of′-asciis*
**and** (*OCaml*) *Str′-Literal.literal′-of′-asciis*
**and** (*Haskell*) *Str′-Literal.literalOfAsciis*
**and** (*Scala*) *Str′-Literal.literalOfAsciis*
| **constant** *String.asciis-of-literal* ⇀
(*SML*) *Str′-Literal.asciis′-of′-literal*
**and** (*OCaml*) *Str′-Literal.asciis′-of′-literal*
**and** (*Haskell*) *Str′-Literal.asciisOfLiteral*
**and** (*Scala*) *Str′-Literal.asciisOfLiteral*
| **class-instance** *String.literal* :: *equal* ⇀
(*Haskell*) −
| **constant** ‹*HOL.equal* :: *String.literal* ⇒ *String.literal* ⇒ *bool*› ⇀
(*SML*) !((- : *string*) = -)
**and** (*OCaml*) !((- : *string*) = -)
**and** (*Haskell*) **infix** *4* ==
**and** (*Scala*) **infixl** *5* ==
| **constant** ‹(≤) :: *String.literal* ⇒ *String.literal* ⇒ *bool*› ⇀
(*SML*) !((- : *string*) <= -)
**and** (*OCaml*) !((- : *string*) <= -)
**and** (*Haskell*) **infix** *4* <=

— Order operations for *String.literal* work in Haskell only if no type class instance needs to be generated, because String = [Char] in Haskell and *char list* need not have the same order as *String.literal*.

**and** (*Scala*) **infixl** *4* <=
**and** (*Eval*) **infixl** *6* <=
| **constant** (<) :: *String.literal* ⇒ *String.literal* ⇒ *bool* ⇀
(*SML*) !((- : *string*) < -)
**and** (*OCaml*) !((- : *string*) < -)
**and** (*Haskell*) **infix** *4* <
**and** (*Scala*) **infixl** *4* <
**and** (*Eval*) **infixl** *6* <

### 73.2.6 Code generation utility

⟨*ML*⟩

**definition** *abort* :: *String.literal* ⇒ (*unit* ⇒ *'a*) ⇒ *'a*
**where** [*simp*]: *abort - f* = *f* ()

**declare** [[*code drop*: *Code.abort*]]

**lemma** *abort-cong*:
*msg* = *msg'* ⟹ *Code.abort msg f* = *Code.abort msg' f*
⟨*proof*⟩

⟨*ML*⟩

**code-printing**
**constant** *Code.abort* ⇀
(*SML*) !(*raise*/ *Fail*/ -)
**and** (*OCaml*) *failwith*
**and** (*Haskell*) !(*error*/ ::/ *forall a*./ *String* −> (() −> *a*) −> *a*)
**and** (*Scala*) !{/ *sys.error*((-));/ ((-)).*apply*(())/ }

### 73.2.7 Finally

**lifting-update** *literal.lifting*
**lifting-forget** *literal.lifting*

**end**

# 74 Reflecting Pure types into HOL

**theory** *Typerep*
**imports** *String*
**begin**

**datatype** *typerep* = *Typerep String.literal typerep list*

**class** *typerep* =
  **fixes** *typerep* :: *'a itself* ⇒ *typerep*
**begin**

**definition** *typerep-of* :: *'a* ⇒ *typerep* **where**
  [*simp*]: *typerep-of x* = *typerep TYPE('a)*

**end**

**syntax**
  *-TYPEREP* :: *type* => *logic* (‹(‹*indent*=*1 notation*=‹*mixfix TYPEREP*››*TYPEREP*/(*1*'(-')))›)
**syntax-consts**
  *-TYPEREP* ⇌ *typerep*

⟨*ML*⟩

**lemma** [*code*]:
  *HOL.equal* (*Typerep tyco1 tys1*) (*Typerep tyco2 tys2*) ⟷ *HOL.equal tyco1 tyco2*
    ∧ *list-all2 HOL.equal tys1 tys2*
  ⟨*proof*⟩

**lemma** [*code nbe*]:
  *HOL.equal* (*x* :: *typerep*) *x* ⟷ *True*
  ⟨*proof*⟩

**code-printing**
  **type-constructor** *typerep* ⇀ (*Eval*) *Term.typ*
| **constant** *Typerep* ⇀ (*Eval*) *Term.Type*/ (-, -)

**code-reserved**
  (*Eval*) *Term*

**hide-const** (**open**) *typerep Typerep*

**end**

# 75 Predicates as enumerations

**theory** *Predicate*
**imports** *String*
**begin**

## 75.1 The type of predicate enumerations (a monad)

**datatype** (*plugins only*: *extraction*) (*dead 'a*) *pred* = *Pred* (*eval*: *'a* ⇒ *bool*)

**lemma** *pred-eqI*:
  (⋀*w*. *eval P w* ⟷ *eval Q w*) ⟹ *P* = *Q*
  ⟨*proof*⟩

**lemma** *pred-eq-iff*:
$P = Q \Longrightarrow (\bigwedge w.\ eval\ P\ w \longleftrightarrow eval\ Q\ w)$
⟨*proof*⟩

**instantiation** *pred* :: (*type*) *complete-lattice*
**begin**

**definition**
$P \le Q \longleftrightarrow eval\ P \le eval\ Q$

**definition**
$P < Q \longleftrightarrow eval\ P < eval\ Q$

**definition**
$\bot = Pred\ \bot$

**lemma** *eval-bot* [*simp*]:
$eval\ \bot\ = \bot$
⟨*proof*⟩

**definition**
$\top = Pred\ \top$

**lemma** *eval-top* [*simp*]:
$eval\ \top\ = \top$
⟨*proof*⟩

**definition**
$P \sqcap Q = Pred\ (eval\ P \sqcap eval\ Q)$

**lemma** *eval-inf* [*simp*]:
$eval\ (P \sqcap Q) = eval\ P \sqcap eval\ Q$
⟨*proof*⟩

**definition**
$P \sqcup Q = Pred\ (eval\ P \sqcup eval\ Q)$

**lemma** *eval-sup* [*simp*]:
$eval\ (P \sqcup Q) = eval\ P \sqcup eval\ Q$
⟨*proof*⟩

**definition**
$\bigsqcap A = Pred\ (\bigsqcap (eval\ `\ A))$

**lemma** *eval-Inf* [*simp*]:
$eval\ (\bigsqcap A) = \bigsqcap (eval\ `\ A)$
⟨*proof*⟩

**definition**
$\bigsqcup A = Pred\ (\bigsqcup (eval\ `\ A))$

**lemma** *eval-Sup* [*simp*]:
$eval\ (\bigsqcup A) = \bigsqcup (eval\ `\ A)$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**lemma** *eval-INF* [*simp*]:
$eval\ (\bigsqcap (f\ `\ A)) = \bigsqcap ((eval \circ f)\ `\ A)$
⟨*proof*⟩

**lemma** *eval-SUP* [*simp*]:
$eval\ (\bigsqcup (f\ `\ A)) = \bigsqcup ((eval \circ f)\ `\ A)$
⟨*proof*⟩

**instantiation** *pred* :: (*type*) *complete-boolean-algebra*
**begin**

**definition**
$-\ P = Pred\ (-\ eval\ P)$

**lemma** *eval-compl* [*simp*]:
$eval\ (-\ P) = -\ eval\ P$
⟨*proof*⟩

**definition**
$P - Q = Pred\ (eval\ P - eval\ Q)$

**lemma** *eval-minus* [*simp*]:
$eval\ (P - Q) = eval\ P - eval\ Q$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**definition** *single* :: $'a \Rightarrow 'a\ pred$ **where**
$single\ x = Pred\ ((=)\ x)$

**lemma** *eval-single* [*simp*]:
$eval\ (single\ x) = (=)\ x$
⟨*proof*⟩

**definition** *bind* :: $'a\ pred \Rightarrow ('a \Rightarrow 'b\ pred) \Rightarrow 'b\ pred$ (**infixl** ‹≫=› *70*) **where**
$P \gg= f = (\bigsqcup (f\ `\ \{x.\ eval\ P\ x\}))$

**lemma** *eval-bind* [*simp*]:
$eval\ (P \gg= f) = eval\ (\bigsqcup (f \text{ ` } \{x.\ eval\ P\ x\}))$
⟨*proof*⟩

**lemma** *bind-bind*:
$(P \gg= Q) \gg= R = P \gg= (\lambda x.\ Q\ x \gg= R)$
⟨*proof*⟩

**lemma** *bind-single*:
$P \gg= single = P$
⟨*proof*⟩

**lemma** *single-bind*:
$single\ x \gg= P = P\ x$
⟨*proof*⟩

**lemma** *bottom-bind*:
$\bot \gg= P = \bot$
⟨*proof*⟩

**lemma** *sup-bind*:
$(P \sqcup Q) \gg= R = P \gg= R \sqcup Q \gg= R$
⟨*proof*⟩

**lemma** *Sup-bind*:
$(\bigsqcup A \gg= f) = \bigsqcup ((\lambda x.\ x \gg= f) \text{ ` } A)$
⟨*proof*⟩

**lemma** *pred-iffI*:
**assumes** $\bigwedge x.\ eval\ A\ x \Longrightarrow eval\ B\ x$
**and** $\bigwedge x.\ eval\ B\ x \Longrightarrow eval\ A\ x$
**shows** $A = B$
⟨*proof*⟩

**lemma** *singleI*: $eval\ (single\ x)\ x$
⟨*proof*⟩

**lemma** *singleI-unit*: $eval\ (single\ ())\ x$
⟨*proof*⟩

**lemma** *singleE*: $eval\ (single\ x)\ y \Longrightarrow (y = x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *singleE'*: $eval\ (single\ x)\ y \Longrightarrow (x = y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *bindI*: $eval\ P\ x \Longrightarrow eval\ (Q\ x)\ y \Longrightarrow eval\ (P \gg= Q)\ y$
⟨*proof*⟩

**lemma** *bindE*: $eval\ (R \gg= Q)\ y \Longrightarrow (\bigwedge x.\ eval\ R\ x \Longrightarrow eval\ (Q\ x)\ y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *botE*: $eval\ \bot\ x \Longrightarrow P$
⟨*proof*⟩

**lemma** *supI1*: $eval\ A\ x \Longrightarrow eval\ (A \sqcup B)\ x$
⟨*proof*⟩

**lemma** *supI2*: $eval\ B\ x \Longrightarrow eval\ (A \sqcup B)\ x$
⟨*proof*⟩

**lemma** *supE*: $eval\ (A \sqcup B)\ x \Longrightarrow (eval\ A\ x \Longrightarrow P) \Longrightarrow (eval\ B\ x \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *single-not-bot* [*simp*]:
$single\ x \neq \bot$
⟨*proof*⟩

**lemma** *not-bot*:
**assumes** $A \neq \bot$
**obtains** $x$ **where** $eval\ A\ x$
⟨*proof*⟩

## 75.2 Emptiness check and definite choice

**definition** *is-empty* :: $'a\ pred \Rightarrow bool$ **where**
$is\text{-}empty\ A \longleftrightarrow A = \bot$

**lemma** *is-empty-bot*:
$is\text{-}empty\ \bot$
⟨*proof*⟩

**lemma** *not-is-empty-single*:
$\neg\ is\text{-}empty\ (single\ x)$
⟨*proof*⟩

**lemma** *is-empty-sup*:
$is\text{-}empty\ (A \sqcup B) \longleftrightarrow is\text{-}empty\ A \wedge is\text{-}empty\ B$
⟨*proof*⟩

**definition** *singleton* :: $(unit \Rightarrow 'a) \Rightarrow 'a\ pred \Rightarrow 'a$ **where**
$singleton\ default\ A = (if\ \exists!x.\ eval\ A\ x\ then\ THE\ x.\ eval\ A\ x\ else\ default\ ())$ **for** *default*

**lemma** *singleton-eqI*:
$\exists!x.\ eval\ A\ x \Longrightarrow eval\ A\ x \Longrightarrow singleton\ default\ A = x$ **for** *default*

⟨*proof*⟩

**lemma** *eval-singletonI*:
$\exists!x.\ eval\ A\ x \Longrightarrow eval\ A\ (singleton\ default\ A)$ **for** *default*
⟨*proof*⟩

**lemma** *single-singleton*:
$\exists!x.\ eval\ A\ x \Longrightarrow single\ (singleton\ default\ A) = A$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-undefinedI*:
$\neg\ (\exists!x.\ eval\ A\ x) \Longrightarrow singleton\ default\ A = default\ ()$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-bot*:
$singleton\ default\ \bot = default\ ()$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-single*:
$singleton\ default\ (single\ x) = x$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-sup-single-single*:
$singleton\ default\ (single\ x \sqcup single\ y) = (if\ x = y\ then\ x\ else\ default\ ())$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-sup-aux*:
$singleton\ default\ (A \sqcup B) = (if\ A = \bot\ then\ singleton\ default\ B$
$else\ if\ B = \bot\ then\ singleton\ default\ A$
$else\ singleton\ default$
$(single\ (singleton\ default\ A) \sqcup single\ (singleton\ default\ B)))$ **for** *default*
⟨*proof*⟩

**lemma** *singleton-sup*:
$singleton\ default\ (A \sqcup B) = (if\ A = \bot\ then\ singleton\ default\ B$
$else\ if\ B = \bot\ then\ singleton\ default\ A$
$else\ if\ singleton\ default\ A = singleton\ default\ B\ then\ singleton\ default\ A\ else\ default\ ())$ **for** *default*
⟨*proof*⟩

## 75.3 Derived operations

**definition** *if-pred* :: $bool \Rightarrow unit\ pred$ **where**
*if-pred-eq*: $if\text{-}pred\ b = (if\ b\ then\ single\ ()\ else\ \bot)$

**definition** *holds* :: $unit\ pred \Rightarrow bool$ **where**
*holds-eq*: $holds\ P = eval\ P\ ()$

**definition** *not-pred* :: *unit pred* $\Rightarrow$ *unit pred* **where**
*not-pred-eq*: *not-pred* $P$ = (*if eval* $P$ () *then* $\perp$ *else single* ())

**lemma** *if-predI*: $P \Longrightarrow$ *eval* (*if-pred* $P$) ()
⟨*proof*⟩

**lemma** *if-predE*: *eval* (*if-pred* $b$) $x \Longrightarrow (b \Longrightarrow x = () \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

**lemma** *not-predI*: $\neg$ $P \Longrightarrow$ *eval* (*not-pred* (*Pred* ($\lambda u.$ $P$))) ()
⟨*proof*⟩

**lemma** *not-predI'*: $\neg$ *eval* $P$ () $\Longrightarrow$ *eval* (*not-pred* $P$) ()
⟨*proof*⟩

**lemma** *not-predE*: *eval* (*not-pred* (*Pred* ($\lambda u.$ $P$))) $x \Longrightarrow (\neg$ $P \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
⟨*proof*⟩

**lemma** *not-predE'*: *eval* (*not-pred* $P$) $x \Longrightarrow (\neg$ *eval* $P$ $x \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*
⟨*proof*⟩
**lemma** $f$ () = *False* $\lor$ $f$ () = *True*
⟨*proof*⟩

**lemma** *closure-of-bool-cases* [*no-atp*]:
**fixes** $f$ :: *unit* $\Rightarrow$ *bool*
**assumes** $f = (\lambda u.$ *False*) $\Longrightarrow P$ $f$
**assumes** $f = (\lambda u.$ *True*) $\Longrightarrow P$ $f$
**shows** $P$ $f$
⟨*proof*⟩

**lemma** *unit-pred-cases*:
**assumes** $P$ $\perp$
**assumes** $P$ (*single* ())
**shows** $P$ $Q$
⟨*proof*⟩

**lemma** *holds-if-pred*:
*holds* (*if-pred* $b$) = $b$
⟨*proof*⟩

**lemma** *if-pred-holds*:
*if-pred* (*holds* $P$) = $P$
⟨*proof*⟩

**lemma** *is-empty-holds*:
*is-empty* $P \longleftrightarrow \neg$ *holds* $P$
⟨*proof*⟩

**definition** *map* :: $('a \Rightarrow 'b) \Rightarrow 'a\ pred \Rightarrow 'b\ pred$ **where**
$map\ f\ P = P \ggg= (single \circ f)$

**lemma** *eval-map* [*simp*]:
$eval\ (map\ f\ P) = (\bigsqcup x{\in}\{x.\ eval\ P\ x\}.\ (\lambda y.\ f\ x = y))$
⟨*proof*⟩

**functor** *map*: *map*
⟨*proof*⟩

## 75.4 Implementation

**datatype** (*plugins only*: *code extraction*) (*dead* $'a$) *seq* =
*Empty*
| *Insert* $'a$ $'a$ *pred*
| *Join* $'a$ *pred* $'a$ *seq*

**primrec** *pred-of-seq* :: $'a\ seq \Rightarrow 'a\ pred$ **where**
$pred\text{-}of\text{-}seq\ Empty = \bot$
| $pred\text{-}of\text{-}seq\ (Insert\ x\ P) = single\ x \sqcup P$
| $pred\text{-}of\text{-}seq\ (Join\ P\ xq) = P \sqcup pred\text{-}of\text{-}seq\ xq$

**definition** *Seq* :: $(unit \Rightarrow 'a\ seq) \Rightarrow 'a\ pred$ **where**
$Seq\ f = pred\text{-}of\text{-}seq\ (f\ ())$

**code-datatype** *Seq*

**primrec** *member* :: $'a\ seq \Rightarrow 'a \Rightarrow bool$ **where**
$member\ Empty\ x \longleftrightarrow False$
| $member\ (Insert\ y\ P)\ x \longleftrightarrow x = y \lor eval\ P\ x$
| $member\ (Join\ P\ xq)\ x \longleftrightarrow eval\ P\ x \lor member\ xq\ x$

**lemma** *eval-member*:
$member\ xq = eval\ (pred\text{-}of\text{-}seq\ xq)$
⟨*proof*⟩

**lemma** *eval-code* [*code*]: $eval\ (Seq\ f) = member\ (f\ ())$
⟨*proof*⟩

**lemma** *single-code* [*code*]:
$single\ x = Seq\ (\lambda u.\ Insert\ x\ \bot)$
⟨*proof*⟩

**primrec** *apply* :: $('a \Rightarrow 'b\ pred) \Rightarrow 'a\ seq \Rightarrow 'b\ seq$ **where**
$apply\ f\ Empty = Empty$
| $apply\ f\ (Insert\ x\ P) = Join\ (f\ x)\ (Join\ (P \ggg= f)\ Empty)$
| $apply\ f\ (Join\ P\ xq) = Join\ (P \ggg= f)\ (apply\ f\ xq)$

**lemma** *apply-bind*:

$pred\text{-}of\text{-}seq\ (apply\ f\ xq) = pred\text{-}of\text{-}seq\ xq \gg= f$
⟨proof⟩

**lemma** *bind-code* [*code*]:
$Seq\ g \gg= f = Seq\ (\lambda u.\ apply\ f\ (g\ ()))$
⟨proof⟩

**lemma** *bot-set-code* [*code*]:
$\bot = Seq\ (\lambda u.\ Empty)$
⟨proof⟩

**primrec** *adjunct* :: $'a\ pred \Rightarrow 'a\ seq \Rightarrow 'a\ seq$ **where**
$adjunct\ P\ Empty = Join\ P\ Empty$
$|\ adjunct\ P\ (Insert\ x\ Q) = Insert\ x\ (Q \sqcup P)$
$|\ adjunct\ P\ (Join\ Q\ xq) = Join\ Q\ (adjunct\ P\ xq)$

**lemma** *adjunct-sup*:
$pred\text{-}of\text{-}seq\ (adjunct\ P\ xq) = P \sqcup pred\text{-}of\text{-}seq\ xq$
⟨proof⟩

**lemma** *sup-code* [*code*]:
$Seq\ f \sqcup Seq\ g = Seq\ (\lambda u.\ case\ f\ ()$
$of\ Empty \Rightarrow g\ ()$
$|\ Insert\ x\ P \Rightarrow Insert\ x\ (P \sqcup Seq\ g)$
$|\ Join\ P\ xq \Rightarrow adjunct\ (Seq\ g)\ (Join\ P\ xq))$
⟨proof⟩

**primrec** *contained* :: $'a\ seq \Rightarrow 'a\ pred \Rightarrow bool$ **where**
$contained\ Empty\ Q \longleftrightarrow True$
$|\ contained\ (Insert\ x\ P)\ Q \longleftrightarrow eval\ Q\ x \wedge P \leq Q$
$|\ contained\ (Join\ P\ xq)\ Q \longleftrightarrow P \leq Q \wedge contained\ xq\ Q$

**lemma** *single-less-eq-eval*:
$single\ x \leq P \longleftrightarrow eval\ P\ x$
⟨proof⟩

**lemma** *contained-less-eq*:
$contained\ xq\ Q \longleftrightarrow pred\text{-}of\text{-}seq\ xq \leq Q$
⟨proof⟩

**lemma** *less-eq-pred-code* [*code*]:
$Seq\ f \leq Q = (case\ f\ ()$
$of\ Empty \Rightarrow True$
$|\ Insert\ x\ P \Rightarrow eval\ Q\ x \wedge P \leq Q$
$|\ Join\ P\ xq \Rightarrow P \leq Q \wedge contained\ xq\ Q)$
⟨proof⟩

**instantiation** *pred* :: (*type*) *equal*
**begin**

**definition** *equal-pred*
**where** [*simp*]: $HOL.equal\ P\ Q \longleftrightarrow P = (Q :: 'a\ pred)$

**instance** ⟨*proof*⟩

**end**

**lemma** [*code nbe*]:
$HOL.equal\ P\ P \longleftrightarrow True$ **for** $P :: 'a\ pred$
⟨*proof*⟩

**lemma** [*code*]:
$HOL.equal\ P\ Q \longleftrightarrow P \le Q \wedge Q \le P$ **for** $P\ Q :: 'a\ pred$
⟨*proof*⟩

**lemma** [*code*]:
$case\text{-}pred\ f\ P = f\ (eval\ P)$
⟨*proof*⟩

**lemma** [*code*]:
$rec\text{-}pred\ f\ P = f\ (eval\ P)$
⟨*proof*⟩

**inductive** $eq :: 'a \Rightarrow 'a \Rightarrow bool$ **where** $eq\ x\ x$

**lemma** *eq-is-eq*: $eq\ x\ y \equiv (x = y)$
⟨*proof*⟩

**primrec** $null :: 'a\ seq \Rightarrow bool$ **where**
$null\ Empty \longleftrightarrow True$
$|\ null\ (Insert\ x\ P) \longleftrightarrow False$
$|\ null\ (Join\ P\ xq) \longleftrightarrow is\text{-}empty\ P \wedge null\ xq$

**lemma** *null-is-empty*:
$null\ xq \longleftrightarrow is\text{-}empty\ (pred\text{-}of\text{-}seq\ xq)$
⟨*proof*⟩

**lemma** *is-empty-code* [*code*]:
$is\text{-}empty\ (Seq\ f) \longleftrightarrow null\ (f\ ())$
⟨*proof*⟩

**primrec** $the\text{-}only :: (unit \Rightarrow 'a) \Rightarrow 'a\ seq \Rightarrow 'a$ **where**
$the\text{-}only\ default\ Empty = default\ ()$ **for** *default*
$|\ the\text{-}only\ default\ (Insert\ x\ P) =$
$(if\ is\text{-}empty\ P\ then\ x\ else\ let\ y = singleton\ default\ P\ in\ if\ x = y\ then\ x\ else\ default\ ())$ **for** *default*
$|\ the\text{-}only\ default\ (Join\ P\ xq) =$
$(if\ is\text{-}empty\ P\ then\ the\text{-}only\ default\ xq\ else\ if\ null\ xq\ then\ singleton\ default\ P$

*else let* $x$ = *singleton default* $P$; $y$ = *the-only default xq in*
*if* $x = y$ *then* $x$ *else default* ()) **for** *default*

**lemma** *the-only-singleton*:
*the-only default xq* = *singleton default* (*pred-of-seq xq*) **for** *default*
⟨*proof*⟩

**lemma** *singleton-code* [*code*]:
*singleton default* (*Seq f*) =
(*case f* () *of*
*Empty* ⇒ *default* ()
| *Insert x P* ⇒ *if is-empty P then x*
*else let* $y$ = *singleton default P in*
*if* $x = y$ *then* $x$ *else default* ()
| *Join P xq* ⇒ *if is-empty P then the-only default xq*
*else if null xq then singleton default P*
*else let* $x$ = *singleton default* $P$; $y$ = *the-only default xq in*
*if* $x = y$ *then* $x$ *else default* ()) **for** *default*
⟨*proof*⟩

**definition** *the* :: $'a$ *pred* ⇒ $'a$ **where**
*the* $A$ = (*THE* $x$. *eval* $A$ $x$)

**lemma** *the-eqI*:
(*THE* $x$. *eval* $P$ $x$) = $x \Longrightarrow$ *the* $P = x$
⟨*proof*⟩

**lemma** *the-eq* [*code*]: *the* $A$ = *singleton* ($\lambda x$. *Code.abort* (*STR* ''*not-unique*'') ($\lambda$-. *the* $A$)) $A$
⟨*proof*⟩

**code-reflect** *Predicate*
**datatypes** *pred* = *Seq* **and** *seq* = *Empty* | *Insert* | *Join*

⟨*ML*⟩

Conversion from and to sets

**definition** *pred-of-set* :: $'a$ *set* ⇒ $'a$ *pred* **where**
*pred-of-set* = *Pred* ∘ ($\lambda A$ $x$. $x \in A$)

**lemma** *eval-pred-of-set* [*simp*]:
*eval* (*pred-of-set* $A$) $x \longleftrightarrow x \in A$
⟨*proof*⟩

**definition** *set-of-pred* :: $'a$ *pred* ⇒ $'a$ *set* **where**
*set-of-pred* = *Collect* ∘ *eval*

**lemma** *member-set-of-pred* [*simp*]:
$x \in$ *set-of-pred* $P \longleftrightarrow$ *Predicate.eval* $P$ $x$

⟨proof⟩

**definition** *set-of-seq* :: $'a$ *seq* ⇒ $'a$ *set* **where**
*set-of-seq* = *set-of-pred* ∘ *pred-of-seq*

**lemma** *member-set-of-seq* [*simp*]:
$x \in$ *set-of-seq xq* = *Predicate.member xq x*
⟨proof⟩

**lemma** *of-pred-code* [*code*]:
*set-of-pred* (*Predicate.Seq f*) = (*case f* () *of*
  *Predicate.Empty* ⇒ {}
| *Predicate.Insert x P* ⇒ *insert x* (*set-of-pred P*)
| *Predicate.Join P xq* ⇒ *set-of-pred P* ∪ *set-of-seq xq*)
⟨proof⟩

**lemma** *of-seq-code* [*code*]:
*set-of-seq Predicate.Empty* = {}
*set-of-seq* (*Predicate.Insert x P*) = *insert x* (*set-of-pred P*)
*set-of-seq* (*Predicate.Join P xq*) = *set-of-pred P* ∪ *set-of-seq xq*
⟨proof⟩

Lazy Evaluation of an indexed function

**function** *iterate-upto* :: (*natural* ⇒ $'a$) ⇒ *natural* ⇒ *natural* ⇒ $'a$ *Predicate.pred*
**where**
*iterate-upto f n m* =
  *Predicate.Seq* (%*u*. *if n* > *m then Predicate.Empty*
   *else Predicate.Insert* (*f n*) (*iterate-upto f* (*n* + *1*) *m*))
⟨proof⟩

**termination** ⟨proof⟩

Misc

**declare** *Inf-set-fold* [**where** $'a$ = $'a$ *Predicate.pred*, *code*]
**declare** *Sup-set-fold* [**where** $'a$ = $'a$ *Predicate.pred*, *code*]

**lemma** *pred-of-set-fold-sup*:
**assumes** *finite A*
**shows** *pred-of-set A* = *Finite-Set.fold sup bot* (*Predicate.single* ‘ *A*) (**is** *?lhs* = *?rhs*)
⟨proof⟩

**lemma** *pred-of-set-set-fold-sup*:
*pred-of-set* (*set xs*) = *fold sup* (*List.map Predicate.single xs*) *bot*
⟨proof⟩

**lemma** *pred-of-set-set-foldr-sup* [*code*]:

*pred-of-set* (*set xs*) = *foldr sup* (*List.map Predicate.single xs*) *bot*
⟨*proof*⟩

**no-notation** *bind* (**infixl** ‹≫=› *70*)

**hide-type** (**open**) *pred seq*
**hide-const** (**open**) *Pred eval single bind is-empty singleton if-pred not-pred holds Empty Insert Join Seq member pred-of-seq apply adjunct null the-only eq map the iterate-upto*
**hide-fact** (**open**) *null-def member-def*

**end**

# 76 Lazy sequences

**theory** *Lazy-Sequence*
**imports** *Predicate*
**begin**

## 76.1 Type of lazy sequences

**datatype** (*plugins only*: *code extraction*) (*dead 'a*) *lazy-sequence* =
*lazy-sequence-of-list 'a list*

**primrec** *list-of-lazy-sequence* :: *'a lazy-sequence* ⇒ *'a list*
**where**
*list-of-lazy-sequence* (*lazy-sequence-of-list xs*) = *xs*

**lemma** *lazy-sequence-of-list-of-lazy-sequence* [*simp*]:
*lazy-sequence-of-list* (*list-of-lazy-sequence xq*) = *xq*
⟨*proof*⟩

**lemma** *lazy-sequence-eqI*:
*list-of-lazy-sequence xq* = *list-of-lazy-sequence yq* ⟹ *xq* = *yq*
⟨*proof*⟩

**lemma** *lazy-sequence-eq-iff*:
*xq* = *yq* ⟷ *list-of-lazy-sequence xq* = *list-of-lazy-sequence yq*
⟨*proof*⟩

**lemma** *case-lazy-sequence* [*simp*]:
*case-lazy-sequence f xq* = *f* (*list-of-lazy-sequence xq*)
⟨*proof*⟩

**lemma** *rec-lazy-sequence* [*simp*]:
*rec-lazy-sequence f xq* = *f* (*list-of-lazy-sequence xq*)
⟨*proof*⟩

**definition** *Lazy-Sequence* :: (*unit* ⇒ (*'a* × *'a lazy-sequence*) *option*) ⇒ *'a lazy-sequence*

**where**
*Lazy-Sequence f* = *lazy-sequence-of-list* (*case f* () *of*
*None* ⇒ []
| *Some* (*x*, *xq*) ⇒ *x* # *list-of-lazy-sequence xq*)

**code-datatype** *Lazy-Sequence*

**declare** *list-of-lazy-sequence.simps* [*code del*]
**declare** *lazy-sequence.case* [*code del*]
**declare** *lazy-sequence.rec* [*code del*]

**lemma** *list-of-Lazy-Sequence* [*simp*]:
*list-of-lazy-sequence* (*Lazy-Sequence f*) = (*case f* () *of*
*None* ⇒ []
| *Some* (*x*, *xq*) ⇒ *x* # *list-of-lazy-sequence xq*)
⟨*proof*⟩

**definition** *yield* :: *'a lazy-sequence* ⇒ (*'a* × *'a lazy-sequence*) *option*
**where**
*yield xq* = (*case list-of-lazy-sequence xq of*
[] ⇒ *None*
| *x* # *xs* ⇒ *Some* (*x*, *lazy-sequence-of-list xs*))

**lemma** *yield-Seq* [*simp*, *code*]:
*yield* (*Lazy-Sequence f*) = *f* ()
⟨*proof*⟩

**lemma** *case-yield-eq* [*simp*]: *case-option g h* (*yield xq*) =
*case-list g* (λ*x*. *curry h x* ∘ *lazy-sequence-of-list*) (*list-of-lazy-sequence xq*)
⟨*proof*⟩

**lemma** *equal-lazy-sequence-code* [*code*]:
*HOL.equal xq yq* = (*case* (*yield xq*, *yield yq*) *of*
(*None*, *None*) ⇒ *True*
| (*Some* (*x*, *xq'*), *Some* (*y*, *yq'*)) ⇒ *HOL.equal x y* ∧ *HOL.equal xq yq*
| - ⇒ *False*)
⟨*proof*⟩

**lemma** [*code nbe*]:
*HOL.equal* (*x* :: *'a lazy-sequence*) *x* ⟷ *True*
⟨*proof*⟩

**definition** *empty* :: *'a lazy-sequence*
**where**
*empty* = *lazy-sequence-of-list* []

**lemma** *list-of-lazy-sequence-empty* [*simp*]:
*list-of-lazy-sequence empty* = []
⟨*proof*⟩

**lemma** *empty-code* [*code*]:
*empty* = *Lazy-Sequence* (λ-. *None*)
⟨*proof*⟩

**definition** *single* :: *'a* ⇒ *'a lazy-sequence*
**where**
*single x* = *lazy-sequence-of-list* [*x*]

**lemma** *list-of-lazy-sequence-single* [*simp*]:
*list-of-lazy-sequence* (*single x*) = [*x*]
⟨*proof*⟩

**lemma** *single-code* [*code*]:
*single x* = *Lazy-Sequence* (λ-. *Some* (*x*, *empty*))
⟨*proof*⟩

**definition** *append* :: *'a lazy-sequence* ⇒ *'a lazy-sequence* ⇒ *'a lazy-sequence*
**where**
*append xq yq* = *lazy-sequence-of-list* (*list-of-lazy-sequence xq* @ *list-of-lazy-sequence yq*)

**lemma** *list-of-lazy-sequence-append* [*simp*]:
*list-of-lazy-sequence* (*append xq yq*) = *list-of-lazy-sequence xq* @ *list-of-lazy-sequence yq*
⟨*proof*⟩

**lemma** *append-code* [*code*]:
*append xq yq* = *Lazy-Sequence* (λ-. *case yield xq of*
*None* ⇒ *yield yq*
| *Some* (*x*, *xq'*) ⇒ *Some* (*x*, *append xq' yq*))
⟨*proof*⟩

**definition** *map* :: (*'a* ⇒ *'b*) ⇒ *'a lazy-sequence* ⇒ *'b lazy-sequence*
**where**
*map f xq* = *lazy-sequence-of-list* (*List.map f* (*list-of-lazy-sequence xq*))

**lemma** *list-of-lazy-sequence-map* [*simp*]:
*list-of-lazy-sequence* (*map f xq*) = *List.map f* (*list-of-lazy-sequence xq*)
⟨*proof*⟩

**lemma** *map-code* [*code*]:
*map f xq* =
*Lazy-Sequence* (λ-. *map-option* (λ(*x*, *xq'*). (*f x*, *map f xq'*)) (*yield xq*))
⟨*proof*⟩

**definition** *flat* :: *'a lazy-sequence lazy-sequence* ⇒ *'a lazy-sequence*
**where**
*flat xqq* = *lazy-sequence-of-list* (*concat* (*List.map list-of-lazy-sequence* (*list-of-lazy-sequence*

*xqq*)))

**lemma** *list-of-lazy-sequence-flat* [*simp*]:
*list-of-lazy-sequence* (*flat xqq*) = *concat* (*List.map list-of-lazy-sequence* (*list-of-lazy-sequence* *xqq*))
⟨*proof*⟩

**lemma** *flat-code* [*code*]:
*flat xqq* = *Lazy-Sequence* (λ-. *case yield xqq of*
*None* ⇒ *None*
| *Some* (*xq*, *xqq′*) ⇒ *yield* (*append xq* (*flat xqq′*)))
⟨*proof*⟩

**definition** *bind* :: ′*a lazy-sequence* ⇒ (′*a* ⇒ ′*b lazy-sequence*) ⇒ ′*b lazy-sequence*
**where**
*bind xq f* = *flat* (*map f xq*)

**definition** *if-seq* :: *bool* ⇒ *unit lazy-sequence*
**where**
*if-seq b* = (*if b then single* () *else empty*)

**definition** *those* :: ′*a option lazy-sequence* ⇒ ′*a lazy-sequence option*
**where**
*those xq* = *map-option lazy-sequence-of-list* (*List.those* (*list-of-lazy-sequence xq*))

**function** *iterate-upto* :: (*natural* ⇒ ′*a*) ⇒ *natural* ⇒ *natural* ⇒ ′*a lazy-sequence*
**where**
*iterate-upto f n m* =
*Lazy-Sequence* (λ-. *if n* > *m then None else Some* (*f n*, *iterate-upto f* (*n* + *1*) *m*))
⟨*proof*⟩

**termination** ⟨*proof*⟩

**definition** *not-seq* :: *unit lazy-sequence* ⇒ *unit lazy-sequence*
**where**
*not-seq xq* = (*case yield xq of*
*None* ⇒ *single* ()
| *Some* ((), *xq*) ⇒ *empty*)

## 76.2 Code setup

**code-reflect** *Lazy-Sequence*
**datatypes** *lazy-sequence* = *Lazy-Sequence*

⟨*ML*⟩

## 76.3 Generator Sequences

### 76.3.1 General lazy sequence operation

**definition** *product* :: $'a$ *lazy-sequence* $\Rightarrow$ $'b$ *lazy-sequence* $\Rightarrow$ $('a \times 'b)$ *lazy-sequence*
**where**
*product s1 s2* = *bind s1* ($\lambda a$. *bind s2* ($\lambda b$. *single* $(a, b)$))

### 76.3.2 Small lazy typeclasses

**class** *small-lazy* =
**fixes** *small-lazy* :: *natural* $\Rightarrow$ $'a$ *lazy-sequence*

**instantiation** *unit* :: *small-lazy*
**begin**

**definition** *small-lazy d* = *single* ()

**instance** ⟨*proof*⟩

**end**

**instantiation** *int* :: *small-lazy*
**begin**

maybe optimise this expression -> append (single x) xs == cons x xs Performance difference?

**function** *small-lazy'* :: *int* $\Rightarrow$ *int* $\Rightarrow$ *int lazy-sequence*
**where**
*small-lazy' d i* = (*if d* < *i then empty*
*else append* (*single i*) (*small-lazy' d* (*i* + *1*)))
⟨*proof*⟩

**termination**
⟨*proof*⟩

**definition**
*small-lazy d* = *small-lazy'* (*int* (*nat-of-natural d*)) (− (*int* (*nat-of-natural d*)))

**instance** ⟨*proof*⟩

**end**

**instantiation** *prod* :: (*small-lazy*, *small-lazy*) *small-lazy*
**begin**

**definition**
*small-lazy d* = *product* (*small-lazy d*) (*small-lazy d*)

**instance** ⟨*proof*⟩

**end**

**instantiation** *list* :: (*small-lazy*) *small-lazy*
**begin**

**fun** *small-lazy-list* :: *natural* ⇒ *'a list lazy-sequence*
**where**
*small-lazy-list d* = *append* (*single* [])
(*if d* > *0 then bind* (*product* (*small-lazy* (*d* − *1*))
(*small-lazy* (*d* − *1*))) (λ(*x*, *xs*). *single* (*x* # *xs*)) *else empty*)

**instance** ⟨*proof*⟩

**end**

## 76.4 With Hit Bound Value

assuming in negative context

**type-synonym** *'a hit-bound-lazy-sequence* = *'a option lazy-sequence*

**definition** *hit-bound* :: *'a hit-bound-lazy-sequence*
**where**
*hit-bound* = *Lazy-Sequence* (λ-. *Some* (*None*, *empty*))

**lemma** *list-of-lazy-sequence-hit-bound* [*simp*]:
*list-of-lazy-sequence hit-bound* = [*None*]
⟨*proof*⟩

**definition** *hb-single* :: *'a* ⇒ *'a hit-bound-lazy-sequence*
**where**
*hb-single x* = *Lazy-Sequence* (λ-. *Some* (*Some x*, *empty*))

**definition** *hb-map* :: (*'a* ⇒ *'b*) ⇒ *'a hit-bound-lazy-sequence* ⇒ *'b hit-bound-lazy-sequence*
**where**
*hb-map f xq* = *map* (*map-option f*) *xq*

**lemma** *hb-map-code* [*code*]:
*hb-map f xq* =
*Lazy-Sequence* (λ-. *map-option* (λ(*x*, *xq'*). (*map-option f x*, *hb-map f xq'*)) (*yield xq*))
⟨*proof*⟩

**definition** *hb-flat* :: *'a hit-bound-lazy-sequence hit-bound-lazy-sequence* ⇒ *'a hit-bound-lazy-sequence*
**where**
*hb-flat xqq* = *lazy-sequence-of-list* (*concat*
(*List.map* ((λ*x*. *case x of None* ⇒ [*None*] | *Some xs* ⇒ *xs*) ∘ *map-option list-of-lazy-sequence*) (*list-of-lazy-sequence xqq*)))

**lemma** *list-of-lazy-sequence-hb-flat* [*simp*]:
  *list-of-lazy-sequence* (*hb-flat xqq*) =
    *concat* (*List.map* (($\lambda x$. *case x of None* $\Rightarrow$ [*None*] | *Some xs* $\Rightarrow$ *xs*) $\circ$ *map-option* *list-of-lazy-sequence*) (*list-of-lazy-sequence xqq*))
  ⟨*proof*⟩

**lemma** *hb-flat-code* [*code*]:
  *hb-flat xqq* = *Lazy-Sequence* ($\lambda$-. *case yield xqq of*
    *None* $\Rightarrow$ *None*
  | *Some* (*xq*, *xqq'*) $\Rightarrow$ *yield*
     (*append* (*case xq of None* $\Rightarrow$ *hit-bound* | *Some xq* $\Rightarrow$ *xq*) (*hb-flat xqq'*)))
  ⟨*proof*⟩

**definition** *hb-bind* :: *'a hit-bound-lazy-sequence* $\Rightarrow$ (*'a* $\Rightarrow$ *'b hit-bound-lazy-sequence*) $\Rightarrow$ *'b hit-bound-lazy-sequence*
**where**
  *hb-bind xq f* = *hb-flat* (*hb-map f xq*)

**definition** *hb-if-seq* :: *bool* $\Rightarrow$ *unit hit-bound-lazy-sequence*
**where**
  *hb-if-seq b* = (*if b then hb-single* () *else empty*)

**definition** *hb-not-seq* :: *unit hit-bound-lazy-sequence* $\Rightarrow$ *unit lazy-sequence*
**where**
  *hb-not-seq xq* = (*case yield xq of*
    *None* $\Rightarrow$ *single* ()
  | *Some* (*x*, *xq*) $\Rightarrow$ *empty*)

**hide-const** (**open**) *yield empty single append flat map bind*
  *if-seq those iterate-upto not-seq product*

**hide-fact** (**open**) *yield-def empty-def single-def append-def flat-def map-def bind-def*
  *if-seq-def those-def not-seq-def product-def*

**end**

# 77 Depth-Limited Sequences with failure element

**theory** *Limited-Sequence*
**imports** *Lazy-Sequence*
**begin**

## 77.1 Depth-Limited Sequence

**type-synonym** *'a dseq* = *natural* $\Rightarrow$ *bool* $\Rightarrow$ *'a lazy-sequence option*

**definition** *empty* :: *'a dseq*
**where**
  *empty* = ($\lambda$- -. *Some Lazy-Sequence.empty*)

**definition** *single* :: $'a \Rightarrow 'a$ *dseq*
**where**
*single* $x = (\lambda$- -. *Some* (*Lazy-Sequence.single* $x$))

**definition** *eval* :: $'a$ *dseq* $\Rightarrow$ *natural* $\Rightarrow$ *bool* $\Rightarrow$ $'a$ *lazy-sequence option*
**where**
[*simp*]: *eval f i pol* = *f i pol*

**definition** *yield* :: $'a$ *dseq* $\Rightarrow$ *natural* $\Rightarrow$ *bool* $\Rightarrow$ ($'a \times 'a$ *dseq*) *option*
**where**
*yield f i pol* = (*case eval f i pol of*
  *None* $\Rightarrow$ *None*
| *Some s* $\Rightarrow$ (*map-option* $\circ$ *apsnd*) ($\lambda r$ - -. *Some r*) (*Lazy-Sequence.yield s*))

**definition** *map-seq* :: ($'a \Rightarrow 'b$ *dseq*) $\Rightarrow$ $'a$ *lazy-sequence* $\Rightarrow$ $'b$ *dseq*
**where**
*map-seq f xq i pol* = *map-option Lazy-Sequence.flat*
  (*Lazy-Sequence.those* (*Lazy-Sequence.map* ($\lambda x$. *f x i pol*) *xq*))

**lemma** *map-seq-code* [*code*]:
*map-seq f xq i pol* = (*case Lazy-Sequence.yield xq of*
  *None* $\Rightarrow$ *Some Lazy-Sequence.empty*
| *Some* ($x$, $xq'$) $\Rightarrow$ (*case eval* (*f x*) *i pol of*
    *None* $\Rightarrow$ *None*
  | *Some yq* $\Rightarrow$ (*case map-seq f* $xq'$ *i pol of*
      *None* $\Rightarrow$ *None*
    | *Some zq* $\Rightarrow$ *Some* (*Lazy-Sequence.append yq zq*))))
⟨*proof*⟩

**definition** *bind* :: $'a$ *dseq* $\Rightarrow$ ($'a \Rightarrow 'b$ *dseq*) $\Rightarrow$ $'b$ *dseq*
**where**
*bind x f* = ($\lambda i$ *pol*.
  *if* $i = 0$ *then*
    (*if pol then Some Lazy-Sequence.empty else None*)
  *else*
    (*case x* ($i - 1$) *pol of*
      *None* $\Rightarrow$ *None*
    | *Some xq* $\Rightarrow$ *map-seq f xq i pol*))

**definition** *union* :: $'a$ *dseq* $\Rightarrow$ $'a$ *dseq* $\Rightarrow$ $'a$ *dseq*
**where**
*union x y* = ($\lambda i$ *pol*. *case* (*x i pol*, *y i pol*) *of*
    (*Some xq*, *Some yq*) $\Rightarrow$ *Some* (*Lazy-Sequence.append xq yq*)
  | - $\Rightarrow$ *None*)

**definition** *if-seq* :: *bool* $\Rightarrow$ *unit dseq*
**where**
*if-seq b* = (*if b then single* () *else empty*)

**definition** *not-seq* :: *unit dseq* ⇒ *unit dseq*
**where**
  *not-seq x* = (λ*i pol. case x i* (¬ *pol*) *of*
    *None* ⇒ *Some Lazy-Sequence.empty*
  | *Some xq* ⇒ (*case Lazy-Sequence.yield xq of*
      *None* ⇒ *Some* (*Lazy-Sequence.single* ())
    | *Some* - ⇒ *Some* (*Lazy-Sequence.empty*)))

**definition** *map* :: (′*a* ⇒ ′*b*) ⇒ ′*a dseq* ⇒ ′*b dseq*
**where**
  *map f g* = (λ*i pol. case g i pol of*
    *None* ⇒ *None*
  | *Some xq* ⇒ *Some* (*Lazy-Sequence.map f xq*))

## 77.2 Positive Depth-Limited Sequence

**type-synonym** ′*a pos-dseq* = *natural* ⇒ ′*a Lazy-Sequence.lazy-sequence*

**definition** *pos-empty* :: ′*a pos-dseq*
**where**
  *pos-empty* = (λ*i. Lazy-Sequence.empty*)

**definition** *pos-single* :: ′*a* ⇒ ′*a pos-dseq*
**where**
  *pos-single x* = (λ*i. Lazy-Sequence.single x*)

**definition** *pos-bind* :: ′*a pos-dseq* ⇒ (′*a* ⇒ ′*b pos-dseq*) ⇒ ′*b pos-dseq*
**where**
  *pos-bind x f* = (λ*i. Lazy-Sequence.bind* (*x i*) (λ*a. f a i*))

**definition** *pos-decr-bind* :: ′*a pos-dseq* ⇒ (′*a* ⇒ ′*b pos-dseq*) ⇒ ′*b pos-dseq*
**where**
  *pos-decr-bind x f* = (λ*i.*
    *if i* = *0 then*
      *Lazy-Sequence.empty*
    *else*
      *Lazy-Sequence.bind* (*x* (*i* − *1*)) (λ*a. f a i*))

**definition** *pos-union* :: ′*a pos-dseq* ⇒ ′*a pos-dseq* ⇒ ′*a pos-dseq*
**where**
  *pos-union xq yq* = (λ*i. Lazy-Sequence.append* (*xq i*) (*yq i*))

**definition** *pos-if-seq* :: *bool* ⇒ *unit pos-dseq*
**where**
  *pos-if-seq b* = (*if b then pos-single* () *else pos-empty*)

**definition** *pos-iterate-upto* :: (*natural* ⇒ ′*a*) ⇒ *natural* ⇒ *natural* ⇒ ′*a pos-dseq*
**where**

*pos-iterate-upto f n m* = ($\lambda i$. *Lazy-Sequence.iterate-upto f n m*)

**definition** *pos-map* :: ('a ⇒ 'b) ⇒ 'a *pos-dseq* ⇒ 'b *pos-dseq*
**where**
*pos-map f xq* = ($\lambda i$. *Lazy-Sequence.map f* (*xq i*))

## 77.3 Negative Depth-Limited Sequence

**type-synonym** 'a *neg-dseq* = *natural* ⇒ 'a *Lazy-Sequence.hit-bound-lazy-sequence*

**definition** *neg-empty* :: 'a *neg-dseq*
**where**
*neg-empty* = ($\lambda i$. *Lazy-Sequence.empty*)

**definition** *neg-single* :: 'a ⇒ 'a *neg-dseq*
**where**
*neg-single x* = ($\lambda i$. *Lazy-Sequence.hb-single x*)

**definition** *neg-bind* :: 'a *neg-dseq* ⇒ ('a ⇒ 'b *neg-dseq*) ⇒ 'b *neg-dseq*
**where**
*neg-bind x f* = ($\lambda i$. *hb-bind* (*x i*) ($\lambda a$. *f a i*))

**definition** *neg-decr-bind* :: 'a *neg-dseq* ⇒ ('a ⇒ 'b *neg-dseq*) ⇒ 'b *neg-dseq*
**where**
*neg-decr-bind x f* = ($\lambda i$.
  *if i* = *0 then*
    *Lazy-Sequence.hit-bound*
  *else*
    *hb-bind* (*x* (*i* − *1*)) ($\lambda a$. *f a i*))

**definition** *neg-union* :: 'a *neg-dseq* ⇒ 'a *neg-dseq* ⇒ 'a *neg-dseq*
**where**
*neg-union x y* = ($\lambda i$. *Lazy-Sequence.append* (*x i*) (*y i*))

**definition** *neg-if-seq* :: *bool* ⇒ *unit neg-dseq*
**where**
*neg-if-seq b* = (*if b then neg-single* () *else neg-empty*)

**definition** *neg-iterate-upto*
**where**
*neg-iterate-upto f n m* = ($\lambda i$. *Lazy-Sequence.iterate-upto* ($\lambda i$. *Some* (*f i*)) *n m*)

**definition** *neg-map* :: ('a ⇒ 'b) ⇒ 'a *neg-dseq* ⇒ 'b *neg-dseq*
**where**
*neg-map f xq* = ($\lambda i$. *Lazy-Sequence.hb-map f* (*xq i*))

## 77.4 Negation

**definition** *pos-not-seq* :: *unit neg-dseq* ⇒ *unit pos-dseq*
**where**

*pos-not-seq xq* = (*λi. Lazy-Sequence.hb-not-seq* (*xq* (*3* ∗ *i*)))

**definition** *neg-not-seq* :: *unit pos-dseq* ⇒ *unit neg-dseq*
**where**
*neg-not-seq x* = (*λi. case Lazy-Sequence.yield* (*x i*) *of*
*None* ⇒ *Lazy-Sequence.hb-single* ()
| *Some* ((), *xq*) ⇒ *Lazy-Sequence.empty*)

⟨*ML*⟩

**code-reserved**
(*Eval*) *Limited-Sequence*

**hide-const** (**open**) *yield empty single eval map-seq bind union if-seq not-seq map*
*pos-empty pos-single pos-bind pos-decr-bind pos-union pos-if-seq pos-iterate-upto*
*pos-not-seq pos-map*
*neg-empty neg-single neg-bind neg-decr-bind neg-union neg-if-seq neg-iterate-upto*
*neg-not-seq neg-map*

**hide-fact** (**open**) *yield-def empty-def single-def eval-def map-seq-def bind-def union-def*
*if-seq-def not-seq-def map-def*
*pos-empty-def pos-single-def pos-bind-def pos-union-def pos-if-seq-def pos-iterate-upto-def*
*pos-not-seq-def pos-map-def*
*neg-empty-def neg-single-def neg-bind-def neg-union-def neg-if-seq-def neg-iterate-upto-def*
*neg-not-seq-def neg-map-def*

**end**

# 78 Term evaluation using the generic code generator

**theory** *Code-Evaluation*
**imports** *Typerep Limited-Sequence*
**keywords** *value* :: *diag*
**begin**

## 78.1 Term representation

### 78.1.1 Terms and class *term-of*

**datatype** (*plugins only*: *extraction*) *term* = *dummy-term*

**definition** *Const* :: *String.literal* ⇒ *typerep* ⇒ *term* **where**
*Const* - - = *dummy-term*

**definition** *App* :: *term* ⇒ *term* ⇒ *term* **where**
*App* - - = *dummy-term*

**definition** *Abs* :: *String.literal* ⇒ *typerep* ⇒ *term* ⇒ *term* **where**
*Abs - - - = dummy-term*

**definition** *Free* :: *String.literal* ⇒ *typerep* ⇒ *term* **where**
*Free - - = dummy-term*

**code-datatype** *Const App Abs Free*

**class** *term-of* = *typerep* +
**fixes** *term-of* :: ′*a* ⇒ *term*

**lemma** *term-of-anything*: *term-of x* ≡ *t*
⟨*proof*⟩

**definition** *valapp* :: (′*a* ⇒ ′*b*) × (*unit* ⇒ *term*)
⇒ ′*a* × (*unit* ⇒ *term*) ⇒ ′*b* × (*unit* ⇒ *term*) **where**
*valapp f x* = (*fst f* (*fst x*), λ*u*. *App* (*snd f* ()) (*snd x* ()))

**lemma** *valapp-code* [*code*, *code-unfold*]:
*valapp* (*f*, *tf*) (*x*, *tx*) = (*f x*, λ*u*. *App* (*tf* ()) (*tx* ()))
⟨*proof*⟩

### 78.1.2 Syntax

**definition** *termify* :: ′*a* ⇒ *term* **where**
[*code del*]: *termify x* = *dummy-term*

**abbreviation** *valtermify* :: ′*a* ⇒ ′*a* × (*unit* ⇒ *term*) **where**
*valtermify x* ≡ (*x*, λ*u*. *termify x*)

**bundle** *term-syntax*
**begin**
**notation** *App* (**infixl** ‹<·>› *70*) **and** *valapp* (**infixl** ‹{·}› *70*)
**end**

## 78.2 Tools setup and evaluation

**context**
**begin**

**qualified definition** *TERM-OF* :: ′*a*::*term-of itself*
**where**
*TERM-OF* = *snd* (*Code-Evaluation.term-of* :: ′*a* ⇒ -, *TYPE*(′*a*))

**qualified definition** *TERM-OF-EQUAL* :: ′*a*::*term-of itself*
**where**
*TERM-OF-EQUAL* = *snd* (λ(*a*::′*a*). (*Code-Evaluation.term-of a*, *HOL.eq a*), *TYPE*(′*a*))

**end**

**lemma** *eq-eq-TrueD*:
  **fixes** $x\ y :: 'a::\{\}$
  **assumes** $(x \equiv y) \equiv$ *Trueprop True*
  **shows** $x \equiv y$
  ⟨*proof*⟩

**code-printing**
  **type-constructor** *term* ⇀ (*Eval*) *Term.term*
| **constant** *Const* ⇀ (*Eval*) *Term.Const*/ ((-), (-))
| **constant** *App* ⇀ (*Eval*) *Term.$*/ ((-), (-))
| **constant** *Abs* ⇀ (*Eval*) *Term.Abs*/ ((-), (-), (-))
| **constant** *Free* ⇀ (*Eval*) *Term.Free*/ ((-), (-))

⟨*ML*⟩

**code-reserved**
  (*Eval*) *Code-Evaluation*

⟨*ML*⟩

## 78.3 Dedicated *term-of* instances

**instantiation** *fun* :: (*typerep*, *typerep*) *term-of*
**begin**

**definition**
  *term-of* $(f :: 'a \Rightarrow 'b) =$
    *Const* (*STR ''Pure.dummy-pattern''*)
      (*Typerep.Typerep* (*STR ''fun''*) [*Typerep.typerep TYPE('a)*, *Typerep.typerep TYPE('b)*])

**instance** ⟨*proof*⟩

**end**

**declare** [[*code drop*:
  *term-of* :: *typerep* ⇒ -
  *term-of* :: *term* ⇒ -
  *term-of* :: *integer* ⇒ -
  *term-of* :: *String.literal* ⇒ -
  *term-of* :: - *Predicate.pred* ⇒ -
  *term-of* :: - *Predicate.seq* ⇒ -]]

**code-printing**
  **constant** *term-of* :: *integer* ⇒ *term* ⇀ (*Eval*) *HOLogic.mk'-number*/ *HOLogic.code'-integerT*
| **constant** *term-of* :: *String.literal* ⇒ *term* ⇀ (*Eval*) *HOLogic.mk'-literal*

**lemma** *term-of-integer* [*unfolded typerep-fun-def typerep-num-def typerep-integer-def*,

*code*]:
*term-of* (*i* :: *integer*) =
(*if i > 0 then*
*App* (*Const* (*STR* ''*Num.numeral-class.numeral*'') (*TYPEREP*(*num* ⇒ *integer*)))
(*term-of* (*num-of-integer i*))
*else if i = 0 then Const* (*STR* ''*Groups.zero-class.zero*'') *TYPEREP*(*integer*)
*else*
*App* (*Const* (*STR* ''*Groups.uminus-class.uminus*'') *TYPEREP*(*integer* ⇒ *integer*))
(*term-of* (− *i*)))
⟨*proof*⟩

**code-reserved**
(*Eval*) *HOLogic*

### 78.4 Generic reification

⟨*ML*⟩

### 78.5 Diagnostic

**definition** *tracing* :: *String.literal* ⇒ '*a* ⇒ '*a* **where**
[*code del*]: *tracing s x* = *x*

**code-printing**
**constant** *tracing* :: *String.literal* => '*a* => '*a* ⇀ (*Eval*) *Code'-Evaluation.tracing*

**hide-const** *dummy-term valapp*
**hide-const** (**open**) *Const App Abs Free termify valtermify term-of tracing*

**end**

# 79 A simple counterexample generator performing random testing

**theory** *Quickcheck-Random*
**imports** *Random Code-Evaluation Enum*
**begin**

⟨*ML*⟩

### 79.1 Catching Match exceptions

**axiomatization** *catch-match* :: '*a* => '*a* => '*a*

**code-printing**
**constant** *catch-match* ⇀ (*Quickcheck*) ((-) *handle Match* => -)

**code-reserved**
($Quickcheck$) $Match$

## 79.2 The *random* class

**class** $random = typerep +$
**fixes** $random :: natural \Rightarrow Random.seed \Rightarrow ('a \times (unit \Rightarrow term)) \times Random.seed$

## 79.3 Fundamental and numeric types

**instantiation** $bool :: random$
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition**
$random\ i = Random.range\ 2 \circ\rightarrow$
$(\lambda k.\ Pair\ (if\ k = 0\ then\ Code\text{-}Evaluation.valtermify\ False\ else\ Code\text{-}Evaluation.valtermify\ True))$

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** $itself :: (typerep)\ random$
**begin**

**definition**
$random\text{-}itself :: natural \Rightarrow Random.seed \Rightarrow ('a\ itself \times (unit \Rightarrow term)) \times Random.seed$
**where** $random\text{-}itself\ \text{-} = Pair\ (Code\text{-}Evaluation.valtermify\ TYPE('a))$

**instance** ⟨*proof*⟩

**end**

**instantiation** $char :: random$
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition**

*random* - = *Random.select* (*Enum.enum* :: *char list*) ∘→ (λ*c*. *Pair* (*c*, λ*u*. *Code-Evaluation.term-of c*))

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** *String.literal* :: *random*
**begin**

**definition**
*random* - = *Pair* (*STR ''''*, λ*u*. *Code-Evaluation.term-of* (*STR ''''*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *nat* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition** *random-nat* :: *natural* ⇒ *Random.seed*
⇒ (*nat* × (*unit* ⇒ *Code-Evaluation.term*)) × *Random.seed*
**where**
*random-nat i* = *Random.range* (*i* + *1*) ∘→ (λ*k*. *Pair* (
*let n* = *nat-of-natural k*
*in* (*n*, λ-. *Code-Evaluation.term-of n*)))

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** *int* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition**
*random i* = *Random.range* (*2* ∗ *i* + *1*) ∘→ (λ*k*. *Pair* (
*let j* = (*if k* ≥ *i then int* (*nat-of-natural* (*k* − *i*)) *else* − (*int* (*nat-of-natural* (*i*

$-$ *k*))))
*in* (*j*, $\lambda$-. *Code-Evaluation.term-of j*)))

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** *natural* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition** *random-natural* :: *natural* $\Rightarrow$ *Random.seed*
$\Rightarrow$ (*natural* $\times$ (*unit* $\Rightarrow$ *Code-Evaluation.term*)) $\times$ *Random.seed*
**where**
*random-natural i* = *Random.range* (*i* + *1*) $\circ\!\!\rightarrow$ ($\lambda$*n*. *Pair* (*n*, $\lambda$-. *Code-Evaluation.term-of n*))

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** *integer* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition** *random-integer* :: *natural* $\Rightarrow$ *Random.seed*
$\Rightarrow$ (*integer* $\times$ (*unit* $\Rightarrow$ *Code-Evaluation.term*)) $\times$ *Random.seed*
**where**
*random-integer i* = *Random.range* (*2* $*$ *i* + *1*) $\circ\!\!\rightarrow$ ($\lambda$*k*. *Pair* (
*let j* = (*if k* $\geq$ *i then integer-of-natural* (*k* $-$ *i*) *else* $-$ (*integer-of-natural* (*i* $-$ *k*)))
*in* (*j*, $\lambda$-. *Code-Evaluation.term-of j*)))

**instance** ⟨*proof*⟩

**end**

**end**

## 79.4 Complex generators

Towards $'a \Rightarrow 'b$

**axiomatization** *random-fun-aux* :: *typerep* ⇒ *typerep* ⇒ (*'a* ⇒ *'a* ⇒ *bool*) ⇒ (*'a* ⇒ *term*)
⇒ (*Random.seed* ⇒ (*'b* × (*unit* ⇒ *term*)) × *Random.seed*)
⇒ (*Random.seed* ⇒ *Random.seed* × *Random.seed*)
⇒ *Random.seed* ⇒ ((*'a* ⇒ *'b*) × (*unit* ⇒ *term*)) × *Random.seed*

**definition** *random-fun-lift* :: (*Random.seed* ⇒ (*'b* × (*unit* ⇒ *term*)) × *Random.seed*)
⇒ *Random.seed* ⇒ ((*'a*::*term-of* ⇒ *'b*::*typerep*) × (*unit* ⇒ *term*)) × *Random.seed*
**where**
*random-fun-lift f* =
*random-fun-aux TYPEREP('a) TYPEREP('b)* (=) *Code-Evaluation.term-of f Random.split-seed*

**instantiation** *fun* :: ({*equal*, *term-of*}, *random*) *random*
**begin**

**definition**
*random-fun* :: *natural* ⇒ *Random.seed* ⇒ ((*'a* ⇒ *'b*) × (*unit* ⇒ *term*)) × *Random.seed*
**where** *random i* = *random-fun-lift* (*random i*)

**instance** ⟨*proof*⟩

**end**

Towards type copies and datatypes

**context**
**includes** *state-combinator-syntax*
**begin**

**definition** *collapse* :: (*'a* ⇒ (*'a* ⇒ *'b* × *'a*) × *'a*) ⇒ *'a* ⇒ *'b* × *'a*
**where** *collapse f* = (*f* ∘→ *id*)

**end**

**definition** *beyond* :: *natural* ⇒ *natural* ⇒ *natural*
**where** *beyond k l* = (*if l* > *k then l else 0*)

**lemma** *beyond-zero*: *beyond k 0* = *0*
⟨*proof*⟩

**context**
**includes** *term-syntax*
**begin**

**definition** [*code-unfold*]:
*valterm-emptyset* = *Code-Evaluation.valtermify* ({} :: ('*a* :: *typerep*) *set*)

**definition** [*code-unfold*]:
*valtermify-insert x s* = *Code-Evaluation.valtermify insert* {·} (*x* :: ('*a* :: *typerep* ∗ -)) {·} *s*

**end**

**instantiation** *set* :: (*random*) *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**fun** *random-aux-set*
**where**
*random-aux-set 0 j* = *collapse* (*Random.select-weight* [(*1*, *Pair valterm-emptyset*)])
| *random-aux-set* (*Code-Numeral.Suc i*) *j* =
*collapse* (*Random.select-weight*
[(*1*, *Pair valterm-emptyset*),
(*Code-Numeral.Suc i*,
*random j* ∘→ (%*x*. *random-aux-set i j* ∘→ (%*s*. *Pair* (*valtermify-insert x s*))))])

**lemma** [*code*]:
*random-aux-set i j* =
*collapse* (*Random.select-weight* [(*1*, *Pair valterm-emptyset*),
(*i*, *random j* ∘→ (%*x*. *random-aux-set* (*i* − *1*) *j* ∘→ (%*s*. *Pair* (*valtermify-insert x s*))))])
⟨*proof*⟩

**definition** *random-set i* = *random-aux-set i i*

**instance** ⟨*proof*⟩

**end**

**end**

**lemma** *random-aux-rec*:
**fixes** *random-aux* :: *natural* ⇒ '*a*
**assumes** *random-aux 0* = *rhs 0*
**and** ⋀*k*. *random-aux* (*Code-Numeral.Suc k*) = *rhs* (*Code-Numeral.Suc k*)
**shows** *random-aux k* = *rhs k*
⟨*proof*⟩

### 79.5 Deriving random generators for datatypes

⟨*ML*⟩

### 79.6 Code setup

**code-printing**
**constant** *random-fun-aux* ⇀ (*Quickcheck*) *Random′-Generators.random′-fun*
— With enough criminal energy this can be abused to derive *False*; for this reason we use a distinguished target *Quickcheck* not spoiling the regular trusted code generation

**code-reserved**
(*Quickcheck*) *Random-Generators*

**hide-const** (**open**) *catch-match random collapse beyond random-fun-aux random-fun-lift*

**hide-fact** (**open**) *collapse-def beyond-def random-fun-lift-def*

**end**

# 80 The Random-Predicate Monad

**theory** *Random-Pred*
**imports** *Quickcheck-Random*
**begin**

**fun** *iter′* :: *′a itself* ⇒ *natural* ⇒ *natural* ⇒ *Random.seed* ⇒ (*′a*::*random*) *Predicate.pred*
**where**
*iter′ T nrandom sz seed* = (*if nrandom* = *0 then bot-class.bot else*
*let* ((*x*, -), *seed′*) = *Quickcheck-Random.random sz seed*
*in Predicate.Seq* (%*u*. *Predicate.Insert x* (*iter′ T* (*nrandom* − *1*) *sz seed′*)))

**definition** *iter* :: *natural* ⇒ *natural* ⇒ *Random.seed* ⇒ (*′a*::*random*) *Predicate.pred*
**where**
*iter nrandom sz seed* = *iter′* (*TYPE*(*′a*)) *nrandom sz seed*

**lemma** [*code*]:
*iter nrandom sz seed* = (*if nrandom* = *0 then bot-class.bot else*
*let* ((*x*, -), *seed′*) = *Quickcheck-Random.random sz seed*
*in Predicate.Seq* (%*u*. *Predicate.Insert x* (*iter* (*nrandom* − *1*) *sz seed′*)))
⟨*proof*⟩

**type-synonym** *′a random-pred* = *Random.seed* ⇒ (*′a Predicate.pred* × *Random.seed*)

**definition** *empty* :: *′a random-pred*
**where** *empty* = *Pair bot*

**definition** *single* :: $'a$ => $'a$ *random-pred*
  **where** *single* $x$ = *Pair* (*Predicate.single* $x$)

**definition** *bind* :: $'a$ *random-pred* $\Rightarrow$ ($'a \Rightarrow$ $'b$ *random-pred*) $\Rightarrow$ $'b$ *random-pred*
  **where**
    *bind* $R$ $f$ = ($\lambda s$. *let*
      ($P$, $s'$) = $R$ $s$;
      ($s1$, $s2$) = *Random.split-seed* $s'$
    *in* (*Predicate.bind* $P$ (%$a$. *fst* ($f$ $a$ $s1$)), $s2$))

**definition** *union* :: $'a$ *random-pred* $\Rightarrow$ $'a$ *random-pred* $\Rightarrow$ $'a$ *random-pred*
**where**
  *union* $R1$ $R2$ = ($\lambda s$. *let*
    ($P1$, $s'$) = $R1$ $s$; ($P2$, $s''$) = $R2$ $s'$
  *in* (*sup-class.sup* $P1$ $P2$, $s''$))

**definition** *if-randompred* :: *bool* $\Rightarrow$ *unit random-pred*
**where**
  *if-randompred* $b$ = (*if* $b$ *then single* () *else empty*)

**definition** *iterate-upto* :: (*natural* $\Rightarrow$ $'a$) => *natural* $\Rightarrow$ *natural* $\Rightarrow$ $'a$ *random-pred*
**where**
  *iterate-upto* $f$ $n$ $m$ = *Pair* (*Predicate.iterate-upto* $f$ $n$ $m$)

**definition** *not-randompred* :: *unit random-pred* $\Rightarrow$ *unit random-pred*
**where**
  *not-randompred* $P$ = ($\lambda s$. *let*
    ($P'$, $s'$) = $P$ $s$
  *in if Predicate.eval* $P'$ () *then* (*Orderings.bot*, $s'$) *else* (*Predicate.single* (), $s'$))

**definition** *Random* :: (*Random.seed* $\Rightarrow$ ($'a \times$ (*unit* $\Rightarrow$ *term*)) $\times$ *Random.seed*) $\Rightarrow$ $'a$ *random-pred*
  **where** *Random* $g$ = *scomp* $g$ (*Pair* $\circ$ (*Predicate.single* $\circ$ *fst*))

**definition** *map* :: ($'a \Rightarrow$ $'b$) $\Rightarrow$ $'a$ *random-pred* $\Rightarrow$ $'b$ *random-pred*
  **where** *map* $f$ $P$ = *bind* $P$ (*single* $\circ$ $f$)

**hide-const** (**open**) *iter'* *iter empty single bind union if-randompred*
  *iterate-upto not-randompred Random map*

**hide-fact** *iter'.simps*

**hide-fact** (**open**) *iter-def empty-def single-def bind-def union-def*
  *if-randompred-def iterate-upto-def not-randompred-def Random-def map-def*

**end**

# 81 Various kind of sequences inside the random monad

**theory** *Random-Sequence*
**imports** *Random-Pred*
**begin**

**type-synonym** $'a$ *random-dseq* = *natural* ⇒ *natural* ⇒ *Random.seed* ⇒ ($'a$ *Limited-Sequence.dseq* × *Random.seed*)

**definition** *empty* :: $'a$ *random-dseq*
**where**
*empty* = (%*nrandom size. Pair* (*Limited-Sequence.empty*))

**definition** *single* :: $'a$ => $'a$ *random-dseq*
**where**
*single x* = (%*nrandom size. Pair* (*Limited-Sequence.single x*))

**definition** *bind* :: $'a$ *random-dseq* => ($'a$ ⇒ $'b$ *random-dseq*) ⇒ $'b$ *random-dseq*
**where**
*bind R f* = (λ*nrandom size s. let*
(*P*, *s'*) = *R nrandom size s*;
(*s1*, *s2*) = *Random.split-seed s'*
*in* (*Limited-Sequence.bind P* (%*a. fst* (*f a nrandom size s1*)), *s2*))

**definition** *union* :: $'a$ *random-dseq* => $'a$ *random-dseq* => $'a$ *random-dseq*
**where**
*union R1 R2* = (λ*nrandom size s. let*
(*S1*, *s'*) = *R1 nrandom size s*; (*S2*, *s''*) = *R2 nrandom size s'*
*in* (*Limited-Sequence.union S1 S2*, *s''*))

**definition** *if-random-dseq* :: *bool* => *unit random-dseq*
**where**
*if-random-dseq b* = (*if b then single* () *else empty*)

**definition** *not-random-dseq* :: *unit random-dseq* => *unit random-dseq*
**where**
*not-random-dseq R* = (λ*nrandom size s. let*
(*S*, *s'*) = *R nrandom size s*
*in* (*Limited-Sequence.not-seq S*, *s'*))

**definition** *map* :: ($'a$ => $'b$) => $'a$ *random-dseq* => $'b$ *random-dseq*
**where**
*map f P* = *bind P* (*single* ∘ *f*)

**fun** *Random* :: (*natural* ⇒ *Random.seed* ⇒ (($'a$ × (*unit* ⇒ *term*)) × *Random.seed*)) ⇒ $'a$ *random-dseq*
**where**
*Random g nrandom* = (%*size. if nrandom* <= *0 then* (*Pair Limited-Sequence.empty*)

*else*
  (*scomp* (*g size*) (%*r*. *scomp* (*Random g* (*nrandom* − *1*) *size*) (%*rs*. *Pair* (*Limited-Sequence.union* (*Limited-Sequence.single* (*fst r*)) *rs*)))))

**type-synonym** ′*a pos-random-dseq* = *natural* ⇒ *natural* ⇒ *Random.seed* ⇒ ′*a Limited-Sequence.pos-dseq*

**definition** *pos-empty* :: ′*a pos-random-dseq*
**where**
*pos-empty* = (%*nrandom size seed*. *Limited-Sequence.pos-empty*)

**definition** *pos-single* :: ′*a* => ′*a pos-random-dseq*
**where**
*pos-single x* = (%*nrandom size seed*. *Limited-Sequence.pos-single x*)

**definition** *pos-bind* :: ′*a pos-random-dseq* => (′*a* ⇒ ′*b pos-random-dseq*) ⇒ ′*b pos-random-dseq*
**where**
*pos-bind R f* = (λ*nrandom size seed*. *Limited-Sequence.pos-bind* (*R nrandom size seed*) (%*a*. *f a nrandom size seed*))

**definition** *pos-decr-bind* :: ′*a pos-random-dseq* => (′*a* ⇒ ′*b pos-random-dseq*) ⇒ ′*b pos-random-dseq*
**where**
*pos-decr-bind R f* = (λ*nrandom size seed*. *Limited-Sequence.pos-decr-bind* (*R nrandom size seed*) (%*a*. *f a nrandom size seed*))

**definition** *pos-union* :: ′*a pos-random-dseq* => ′*a pos-random-dseq* => ′*a pos-random-dseq*
**where**
*pos-union R1 R2* = (λ*nrandom size seed*. *Limited-Sequence.pos-union* (*R1 nrandom size seed*) (*R2 nrandom size seed*))

**definition** *pos-if-random-dseq* :: *bool* => *unit pos-random-dseq*
**where**
*pos-if-random-dseq b* = (*if b then pos-single* () *else pos-empty*)

**definition** *pos-iterate-upto* :: (*natural* => ′*a*) => *natural* => *natural* => ′*a pos-random-dseq*
**where**
*pos-iterate-upto f n m* = (λ*nrandom size seed*. *Limited-Sequence.pos-iterate-upto f n m*)

**definition** *pos-map* :: (′*a* => ′*b*) => ′*a pos-random-dseq* => ′*b pos-random-dseq*
**where**
*pos-map f P* = *pos-bind P* (*pos-single* ∘ *f*)

**fun** *iter* :: (*Random.seed* ⇒ (′*a* × (*unit* ⇒ *term*)) × *Random.seed*)
⇒ *natural* ⇒ *Random.seed* ⇒ ′*a Lazy-Sequence.lazy-sequence*

**where**
*iter random nrandom seed =*
*(if nrandom = 0 then Lazy-Sequence.empty else Lazy-Sequence.Lazy-Sequence (%u. let ((x, -), seed′) = random seed in Some (x, iter random (nrandom − 1) seed′)))*

**definition** *pos-Random* :: (*natural* ⇒ *Random.seed* ⇒ (′*a* × (*unit* ⇒ *term*)) × *Random.seed*)
⇒ ′*a pos-random-dseq*
**where**
*pos-Random g = (%nrandom size seed depth. iter (g size) nrandom seed)*

**type-synonym** ′*a neg-random-dseq = natural ⇒ natural ⇒ Random.seed ⇒* ′*a Limited-Sequence.neg-dseq*

**definition** *neg-empty* :: ′*a neg-random-dseq*
**where**
*neg-empty = (%nrandom size seed. Limited-Sequence.neg-empty)*

**definition** *neg-single* :: ′*a* => ′*a neg-random-dseq*
**where**
*neg-single x = (%nrandom size seed. Limited-Sequence.neg-single x)*

**definition** *neg-bind* :: ′*a neg-random-dseq* => (′*a* ⇒ ′*b neg-random-dseq*) ⇒ ′*b neg-random-dseq*
**where**
*neg-bind R f = (λnrandom size seed. Limited-Sequence.neg-bind (R nrandom size seed) (%a. f a nrandom size seed))*

**definition** *neg-decr-bind* :: ′*a neg-random-dseq* => (′*a* ⇒ ′*b neg-random-dseq*) ⇒ ′*b neg-random-dseq*
**where**
*neg-decr-bind R f = (λnrandom size seed. Limited-Sequence.neg-decr-bind (R nrandom size seed) (%a. f a nrandom size seed))*

**definition** *neg-union* :: ′*a neg-random-dseq* => ′*a neg-random-dseq* => ′*a neg-random-dseq*
**where**
*neg-union R1 R2 = (λnrandom size seed. Limited-Sequence.neg-union (R1 nrandom size seed) (R2 nrandom size seed))*

**definition** *neg-if-random-dseq* :: *bool* => *unit neg-random-dseq*
**where**
*neg-if-random-dseq b = (if b then neg-single () else neg-empty)*

**definition** *neg-iterate-upto* :: (*natural* => ′*a*) => *natural* => *natural* => ′*a neg-random-dseq*
**where**
*neg-iterate-upto f n m = (λnrandom size seed. Limited-Sequence.neg-iterate-upto*

*f n m*)

**definition** *neg-not-random-dseq* :: *unit pos-random-dseq* => *unit neg-random-dseq*
**where**
*neg-not-random-dseq R* = ($\lambda$*nrandom size seed. Limited-Sequence.neg-not-seq* (*R nrandom size seed*))

**definition** *neg-map* :: (*'a* => *'b*) => *'a neg-random-dseq* => *'b neg-random-dseq*
**where**
*neg-map f P* = *neg-bind P* (*neg-single* ∘ *f*)

**definition** *pos-not-random-dseq* :: *unit neg-random-dseq* => *unit pos-random-dseq*
**where**
*pos-not-random-dseq R* = ($\lambda$*nrandom size seed. Limited-Sequence.pos-not-seq* (*R nrandom size seed*))

**hide-const** (**open**)
*empty single bind union if-random-dseq not-random-dseq map Random*
*pos-empty pos-single pos-bind pos-decr-bind pos-union pos-if-random-dseq pos-iterate-upto*
*pos-not-random-dseq pos-map iter pos-Random*
*neg-empty neg-single neg-bind neg-decr-bind neg-union neg-if-random-dseq neg-iterate-upto*
*neg-not-random-dseq neg-map*

**hide-fact** (**open**) *empty-def single-def bind-def union-def if-random-dseq-def not-random-dseq-def*
*map-def Random.simps*
*pos-empty-def pos-single-def pos-bind-def pos-decr-bind-def pos-union-def pos-if-random-dseq-def*
*pos-iterate-upto-def pos-not-random-dseq-def pos-map-def iter.simps pos-Random-def*
*neg-empty-def neg-single-def neg-bind-def neg-decr-bind-def neg-union-def neg-if-random-dseq-def*
*neg-iterate-upto-def neg-not-random-dseq-def neg-map-def*

**end**

# 82 A simple counterexample generator performing exhaustive testing

**theory** *Quickcheck-Exhaustive*
**imports** *Quickcheck-Random*
**keywords** *quickcheck-generator* :: *thy-decl*
**begin**

## 82.1 Basic operations for exhaustive generators

**definition** *orelse* :: *'a option* ⇒ *'a option* ⇒ *'a option* (**infixr** ‹*orelse*› *55*)
**where** [*code-unfold*]: *x orelse y* = (*case x of Some x'* ⇒ *Some x'* | *None* ⇒ *y*)

## 82.2 Exhaustive generator type classes

**class** *exhaustive* = *term-of* +

**fixes** *exhaustive* :: $('a \Rightarrow (bool \times term\ list)\ option) \Rightarrow natural \Rightarrow (bool \times term\ list)\ option$

**class** *full-exhaustive* = *term-of* +
**fixes** *full-exhaustive* ::
$('a \times (unit \Rightarrow term) \Rightarrow (bool \times term\ list)\ option) \Rightarrow natural \Rightarrow (bool \times term\ list)\ option$

**instantiation** *natural* :: *full-exhaustive*
**begin**

**function** *full-exhaustive-natural'* ::
$(natural \times (unit \Rightarrow term) \Rightarrow (bool \times term\ list)\ option) \Rightarrow$
$natural \Rightarrow natural \Rightarrow (bool \times term\ list)\ option$
**where** *full-exhaustive-natural' f d i* =
(*if d* < *i then None*
*else* (*f* (*i*, λ-. *Code-Evaluation.term-of i*)) *orelse* (*full-exhaustive-natural' f d* (*i* + *1*)))
⟨*proof*⟩

**termination**
⟨*proof*⟩

**definition** *full-exhaustive f d* = *full-exhaustive-natural' f d 0*

**instance** ⟨*proof*⟩

**end**

**instantiation** *natural* :: *exhaustive*
**begin**

**function** *exhaustive-natural'* ::
$(natural \Rightarrow (bool \times term\ list)\ option) \Rightarrow natural \Rightarrow natural \Rightarrow (bool \times term\ list)\ option$
**where** *exhaustive-natural' f d i* =
(*if d* < *i then None*
*else* (*f i orelse exhaustive-natural' f d* (*i* + *1*)))
⟨*proof*⟩

**termination**
⟨*proof*⟩

**definition** *exhaustive f d* = *exhaustive-natural' f d 0*

**instance** ⟨*proof*⟩

**end**

**instantiation** *integer* :: *exhaustive*
**begin**

**function** *exhaustive-integer'* ::
 $(integer \Rightarrow (bool \times term\ list)\ option) \Rightarrow integer \Rightarrow integer \Rightarrow (bool \times term\ list)$
*option*
 **where** *exhaustive-integer' f d i* =
 $(if\ d < i\ then\ None\ else\ (f\ i\ orelse\ exhaustive\text{-}integer'\ f\ d\ (i + 1)))$
⟨*proof*⟩

**termination**
 ⟨*proof*⟩

**definition** *exhaustive f d* = *exhaustive-integer' f* (*integer-of-natural d*) (− (*integer-of-natural d*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *integer* :: *full-exhaustive*
**begin**

**function** *full-exhaustive-integer'* ::
 $(integer \times (unit \Rightarrow term) \Rightarrow (bool \times term\ list)\ option) \Rightarrow$
 $integer \Rightarrow integer \Rightarrow (bool \times term\ list)\ option$
 **where** *full-exhaustive-integer' f d i* =
 (*if d* < *i then None*
 *else*
 (*case f* (*i*, λ-. *Code-Evaluation.term-of i*) *of*
 *Some t* ⇒ *Some t*
 | *None* ⇒ *full-exhaustive-integer' f d* (*i* + *1*)))
⟨*proof*⟩

**termination**
 ⟨*proof*⟩

**definition** *full-exhaustive f d* =
 *full-exhaustive-integer' f* (*integer-of-natural d*) (− (*integer-of-natural d*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *nat* :: *exhaustive*
**begin**

**definition** *exhaustive f d* = *exhaustive* (λ*x*. *f* (*nat-of-natural x*)) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *nat* :: *full-exhaustive*
**begin**

**definition** *full-exhaustive f d* =
*full-exhaustive* ($\lambda$(*x*, *xt*). *f* (*nat-of-natural x*, $\lambda$-. *Code-Evaluation.term-of* (*nat-of-natural x*))) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *int* :: *exhaustive*
**begin**

**function** *exhaustive-int'* ::
(*int* ⇒ (*bool* × *term list*) *option*) ⇒ *int* ⇒ *int* ⇒ (*bool* × *term list*) *option*
**where** *exhaustive-int' f d i* =
(*if d* < *i then None else* (*f i orelse exhaustive-int' f d* (*i* + *1*)))
⟨*proof*⟩

**termination**
⟨*proof*⟩

**definition** *exhaustive f d* =
*exhaustive-int' f* (*int-of-integer* (*integer-of-natural d*))
(− (*int-of-integer* (*integer-of-natural d*)))

**instance** ⟨*proof*⟩

**end**

**instantiation** *int* :: *full-exhaustive*
**begin**

**function** *full-exhaustive-int'* ::
(*int* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒
*int* ⇒ *int* ⇒ (*bool* × *term list*) *option*
**where** *full-exhaustive-int' f d i* =
(*if d* < *i then None*
*else*
(*case f* (*i*, $\lambda$-. *Code-Evaluation.term-of i*) *of*
*Some t* ⇒ *Some t*
| *None* ⇒ *full-exhaustive-int' f d* (*i* + *1*)))
⟨*proof*⟩

**termination**
⟨*proof*⟩

**definition** *full-exhaustive f d* =
*full-exhaustive-int′ f* (*int-of-integer* (*integer-of-natural d*))
(− (*int-of-integer* (*integer-of-natural d*)))

**instance** ⟨*proof*⟩

**end**

**instantiation** *prod* :: (*exhaustive*, *exhaustive*) *exhaustive*
**begin**

**definition** *exhaustive f d* = *exhaustive* (λ*x*. *exhaustive* (λ*y*. *f* ((*x*, *y*))) *d*) *d*

**instance** ⟨*proof*⟩

**end**

**context**
**includes** *term-syntax*
**begin**

**definition**
[*code-unfold*]: *valtermify-pair x y* =
*Code-Evaluation.valtermify* (*Pair* :: ′*a*::*typerep* ⇒ ′*b*::*typerep* ⇒ ′*a* × ′*b*) {·} *x*
{·} *y*

**end**

**instantiation** *prod* :: (*full-exhaustive*, *full-exhaustive*) *full-exhaustive*
**begin**

**definition** *full-exhaustive f d* =
*full-exhaustive* (λ*x*. *full-exhaustive* (λ*y*. *f* (*valtermify-pair x y*)) *d*) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *set* :: (*exhaustive*) *exhaustive*
**begin**

**fun** *exhaustive-set*
**where**
*exhaustive-set f i* =
(*if i* = *0 then None*
*else*

$f$ {} *orelse*
*exhaustive-set*
($\lambda A$. $f$ $A$ *orelse exhaustive* ($\lambda x$. *if* $x \in A$ *then None else* $f$ (*insert* $x$ $A$)) ($i - 1$)) ($i - 1$))

**instance** ⟨*proof*⟩

**end**

**instantiation** *set* :: (*full-exhaustive*) *full-exhaustive*
**begin**

**fun** *full-exhaustive-set*
**where**
*full-exhaustive-set* $f$ $i$ =
(*if* $i = 0$ *then None*
*else*
$f$ *valterm-emptyset orelse*
*full-exhaustive-set*
($\lambda A$. $f$ $A$ *orelse Quickcheck-Exhaustive.full-exhaustive*
($\lambda x$. *if fst* $x \in$ *fst* $A$ *then None else* $f$ (*valtermify-insert* $x$ $A$)) ($i - 1$)) ($i - 1$))

**instance** ⟨*proof*⟩

**end**

**instantiation** *fun* :: ({*equal*,*exhaustive*}, *exhaustive*) *exhaustive*
**begin**

**fun** *exhaustive-fun'* ::
(('$a$ ⇒ '$b$) ⇒ (*bool* × *term list*) *option*) ⇒ *natural* ⇒ *natural* ⇒ (*bool* × *term list*) *option*
**where**
*exhaustive-fun'* $f$ $i$ $d$ =
(*exhaustive* ($\lambda b$. $f$ ($\lambda$-. $b$)) $d$) *orelse*
(*if* $i > 1$ *then*
*exhaustive-fun'*
($\lambda g$. *exhaustive* ($\lambda a$. *exhaustive* ($\lambda b$. $f$ ($g$($a$ := $b$))) $d$) $d$) ($i - 1$) $d$ *else None*)

**definition** *exhaustive-fun* ::
(('$a$ ⇒ '$b$) ⇒ (*bool* × *term list*) *option*) ⇒ *natural* ⇒ (*bool* × *term list*) *option*
**where** *exhaustive-fun* $f$ $d$ = *exhaustive-fun'* $f$ $d$ $d$

**instance** ⟨*proof*⟩

**end**

**definition** [*code-unfold*]:
*valtermify-absdummy* =
($\lambda$(*v*, *t*).
($\lambda$-::'*a*. *v*,
$\lambda$*u*::*unit*. *Code-Evaluation.Abs* (*STR* ''*x*'') (*Typerep.typerep* *TYPE*('*a*::*typerep*)) (*t* ())))

**context**
**includes** *term-syntax*
**begin**

**definition**
[*code-unfold*]: *valtermify-fun-upd* *g* *a* *b* =
*Code-Evaluation.valtermify*
(*fun-upd* :: ('*a*::*typerep* ⇒ '*b*::*typerep*) ⇒ '*a* ⇒ '*b* ⇒ '*a* ⇒ '*b*) {·} *g* {·} *a* {·} *b*

**end**

**instantiation** *fun* :: ({*equal*,*full-exhaustive*}, *full-exhaustive*) *full-exhaustive*
**begin**

**fun** *full-exhaustive-fun'* ::
(('*a* ⇒ '*b*) × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒
*natural* ⇒ *natural* ⇒ (*bool* × *term list*) *option*
**where**
*full-exhaustive-fun'* *f* *i* *d* =
*full-exhaustive* ($\lambda$*v*. *f* (*valtermify-absdummy* *v*)) *d* *orelse*
(*if* *i* > *1* *then*
*full-exhaustive-fun'*
($\lambda$*g*. *full-exhaustive*
($\lambda$*a*. *full-exhaustive* ($\lambda$*b*. *f* (*valtermify-fun-upd* *g* *a* *b*)) *d*) *d*) (*i* − *1*) *d*
*else* *None*)

**definition** *full-exhaustive-fun* ::
(('*a* ⇒ '*b*) × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒
*natural* ⇒ (*bool* × *term list*) *option*
**where** *full-exhaustive-fun* *f* *d* = *full-exhaustive-fun'* *f* *d* *d*

**instance** ⟨*proof*⟩

**end**

### 82.2.1 A smarter enumeration scheme for functions over finite datatypes

**class** *check-all* = *enum* + *term-of* +
**fixes** *check-all* :: ('*a* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒ (*bool* ∗ *term list*) *option*
**fixes** *enum-term-of* :: '*a* *itself* ⇒ *unit* ⇒ *term list*

**fun** *check-all-n-lists* :: ('a::*check-all list* × (*unit* ⇒ *term list*) ⇒
(*bool* × *term list*) *option*) ⇒ *natural* ⇒ (*bool* ∗ *term list*) *option*
**where**
*check-all-n-lists f n* =
(*if n* = *0 then f* ([], (λ-. []))
*else check-all* (λ(*x*, *xt*).
*check-all-n-lists* (λ(*xs*, *xst*). *f* ((*x* # *xs*), (λ-. (*xt* () # *xst* ()))))) (*n* − *1*)))

**context**
**includes** *term-syntax*
**begin**

**definition**
[*code-unfold*]: *termify-fun-upd g a b* =
(*Code-Evaluation.termify*
(*fun-upd* :: ('a::*typerep* ⇒ 'b::*typerep*) ⇒ 'a ⇒ 'b ⇒ 'a ⇒ 'b) <·> *g* <·> *a*
<·> *b*)

**end**

**definition** *mk-map-term* ::
(*unit* ⇒ *typerep*) ⇒ (*unit* ⇒ *typerep*) ⇒
(*unit* ⇒ *term list*) ⇒ (*unit* ⇒ *term list*) ⇒ *unit* ⇒ *term*
**where** *mk-map-term T1 T2 domm rng* =
(λ-.
*let*
*T1* = *T1* ();
*T2* = *T2* ();
*update-term* =
(λ*g* (*a*, *b*).
*Code-Evaluation.App* (*Code-Evaluation.App* (*Code-Evaluation.App*
(*Code-Evaluation.Const* (*STR* ''*Fun.fun-upd*'')
(*Typerep.Typerep* (*STR* ''*fun*'') [*Typerep.Typerep* (*STR* ''*fun*'') [*T1*,
*T2*],
*Typerep.Typerep* (*STR* ''*fun*'') [*T1*,
*Typerep.Typerep* (*STR* ''*fun*'') [*T2*, *Typerep.Typerep* (*STR* ''*fun*'')
[*T1*, *T2*]]]]))
*g*) *a*) *b*)
*in*
*List.foldl update-term*
(*Code-Evaluation.Abs* (*STR* ''*x*'') *T1*
(*Code-Evaluation.Const* (*STR* ''*HOL.undefined*'') *T2*)) (*zip* (*domm* ())
(*rng* ())))

**instantiation** *fun* :: ({*equal*,*check-all*}, *check-all*) *check-all*
**begin**

**definition**

*check-all f* =
  (*let*
    *mk-term* =
      *mk-map-term*
        ($\lambda$-. *Typerep.typerep* (*TYPE*(*'a*)))
        ($\lambda$-. *Typerep.typerep* (*TYPE*(*'b*)))
        (*enum-term-of* (*TYPE*(*'a*)));
    *enum* = (*Enum.enum* :: *'a list*)
  *in*
    *check-all-n-lists*
      ($\lambda$(*ys*, *yst*). *f* (*the* ∘ *map-of* (*zip enum ys*), *mk-term yst*))
      (*natural-of-nat* (*length enum*)))

**definition** *enum-term-of-fun* :: (*'a* ⇒ *'b*) *itself* ⇒ *unit* ⇒ *term list*
  **where** *enum-term-of-fun* =
    ($\lambda$- -.
      *let*
        *enum-term-of-a* = *enum-term-of* (*TYPE*(*'a*));
        *mk-term* =
          *mk-map-term*
            ($\lambda$-. *Typerep.typerep* (*TYPE*(*'a*)))
            ($\lambda$-. *Typerep.typerep* (*TYPE*(*'b*)))
            *enum-term-of-a*
      *in*
        *map* ($\lambda$*ys*. *mk-term* ($\lambda$-. *ys*) ())
          (*List.n-lists* (*length* (*enum-term-of-a* ())) (*enum-term-of* (*TYPE*(*'b*)) ())))

**instance** ⟨*proof*⟩

**end**

**context**
  **includes** *term-syntax*
**begin**

**fun** *check-all-subsets* ::
  ((*'a*::*typerep*) *set* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒
    (*'a* × (*unit* ⇒ *term*)) *list* ⇒ (*bool* × *term list*) *option*
**where**
  *check-all-subsets f* [] = *f valterm-emptyset*
| *check-all-subsets f* (*x* # *xs*) =
   *check-all-subsets* ($\lambda$*s*. *case f s of Some ts* ⇒ *Some ts* | *None* ⇒ *f* (*valtermify-insert x s*)) *xs*

**definition**
  [*code-unfold*]: *term-emptyset* = *Code-Evaluation.termify* ({} :: (*'a*::*typerep*) *set*)

**definition**
  [*code-unfold*]: *termify-insert x s* =

*Code-Evaluation.termify* (*insert* :: ('*a*::*typerep*) ⇒ '*a set* ⇒ '*a set*) <·> *x* <·> *s*

**definition** *setify* :: ('*a*::*typerep*) *itself* ⇒ *term list* ⇒ *term*
**where**
*setify T ts* = *foldr* (*termify-insert T*) *ts* (*term-emptyset T*)

**end**

**instantiation** *set* :: (*check-all*) *check-all*
**begin**

**definition**
*check-all-set f* =
*check-all-subsets f*
(*zip* (*Enum.enum* :: '*a list*)
(*map* (λ*a*. λ*u* :: *unit*. *a*) (*Quickcheck-Exhaustive.enum-term-of* (*TYPE* ('*a*)) ())))

**definition** *enum-term-of-set* :: '*a set itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-set* - - =
*map* (*setify* (*TYPE*('*a*))) (*subseqs* (*Quickcheck-Exhaustive.enum-term-of* (*TYPE*('*a*)) ()))

**instance** ⟨*proof*⟩

**end**

**instantiation** *unit* :: *check-all*
**begin**

**definition** *check-all f* = *f* (*Code-Evaluation.valtermify* ())

**definition** *enum-term-of-unit* :: *unit itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-unit* = (λ- -. [*Code-Evaluation.term-of* ()])

**instance** ⟨*proof*⟩

**end**

**instantiation** *bool* :: *check-all*
**begin**

**definition**
*check-all f* =
(*case f* (*Code-Evaluation.valtermify False*) *of*
*Some x'* ⇒ *Some x'*
| *None* ⇒ *f* (*Code-Evaluation.valtermify True*))

**definition** *enum-term-of-bool* :: *bool itself* ⇒ *unit* ⇒ *term list*
 **where** *enum-term-of-bool* = (λ- -. *map Code-Evaluation.term-of* (*Enum.enum* :: *bool list*))

**instance** ⟨*proof*⟩

**end**

**context**
 **includes** *term-syntax*
**begin**

**definition** [*code-unfold*]:
 *termify-pair x y* =
  *Code-Evaluation.termify* (*Pair* :: 'a::*typerep* ⇒ 'b :: *typerep* ⇒ 'a ∗ 'b) <·> *x* <·> *y*

**end**

**instantiation** *prod* :: (*check-all*, *check-all*) *check-all*
**begin**

**definition** *check-all f* = *check-all* (λ*x*. *check-all* (λ*y*. *f* (*valtermify-pair x y*)))

**definition** *enum-term-of-prod* :: ('a ∗ 'b) *itself* ⇒ *unit* ⇒ *term list*
 **where** *enum-term-of-prod* =
  (λ- -.
   *map* (λ(*x*, *y*). *termify-pair TYPE*('a) *TYPE*('b) *x y*)
    (*List.product* (*enum-term-of* (*TYPE*('a)) ()) (*enum-term-of* (*TYPE*('b)) ())))

**instance** ⟨*proof*⟩

**end**

**context**
 **includes** *term-syntax*
**begin**

**definition**
 [*code-unfold*]: *valtermify-Inl x* =
  *Code-Evaluation.valtermify* (*Inl* :: 'a::*typerep* ⇒ 'a + 'b :: *typerep*) {·} *x*

**definition**
 [*code-unfold*]: *valtermify-Inr x* =
  *Code-Evaluation.valtermify* (*Inr* :: 'b::*typerep* ⇒ 'a::*typerep* + 'b) {·} *x*

**end**

**instantiation** *sum* :: (*check-all*, *check-all*) *check-all*
**begin**

**definition**
*check-all f* = *check-all* ($\lambda a$. *f* (*valtermify-Inl a*)) *orelse check-all* ($\lambda b$. *f* (*valtermify-Inr b*))

**definition** *enum-term-of-sum* :: ('*a* + '*b*) *itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-sum* =
($\lambda$- -.
*let*
*T1* = *Typerep.typerep* (*TYPE*('*a*));
*T2* = *Typerep.typerep* (*TYPE*('*b*))
*in*
*map*
(*Code-Evaluation.App* (*Code-Evaluation.Const* (*STR* ''*Sum-Type.Inl*'')
(*Typerep.Typerep* (*STR* ''*fun*'') [*T1*, *Typerep.Typerep* (*STR* ''*Sum-Type.sum*'') [*T1*, *T2*]])))
(*enum-term-of* (*TYPE*('*a*)) ()) @
*map*
(*Code-Evaluation.App* (*Code-Evaluation.Const* (*STR* ''*Sum-Type.Inr*'')
(*Typerep.Typerep* (*STR* ''*fun*'') [*T2*, *Typerep.Typerep* (*STR* ''*Sum-Type.sum*'') [*T1*, *T2*]])))
(*enum-term-of* (*TYPE*('*b*)) ()))

**instance** ⟨*proof*⟩

**end**

**instantiation** *char* :: *check-all*
**begin**

**primrec** *check-all-char'* ::
(*char* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒ *char list* ⇒ (*bool* × *term list*) *option*
**where** *check-all-char' f* [] = *None*
| *check-all-char' f* (*c* # *cs*) = *f* (*c*, $\lambda$-. *Code-Evaluation.term-of c*)
*orelse check-all-char' f cs*

**definition** *check-all-char* ::
(*char* × (*unit* ⇒ *term*) ⇒ (*bool* × *term list*) *option*) ⇒ (*bool* × *term list*) *option*
**where** *check-all f* = *check-all-char' f Enum.enum*

**definition** *enum-term-of-char* :: *char itself* ⇒ *unit* ⇒ *term list*
**where**
*enum-term-of-char* = ($\lambda$- -. *map Code-Evaluation.term-of* (*Enum.enum* :: *char list*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *option* :: (*check-all*) *check-all*
**begin**

**definition**
  *check-all f* =
    *f* (*Code-Evaluation.valtermify* (*None* :: ′*a option*)) *orelse*
    *check-all*
      (λ(*x*, *t*).
        *f*
          (*Some x*,
            λ-. *Code-Evaluation.App*
              (*Code-Evaluation.Const* (*STR* ′′*Option.option.Some*′′)
                (*Typerep.Typerep* (*STR* ′′*fun*′′)
                [*Typerep.typerep TYPE*(′*a*),
              *Typerep.Typerep* (*STR* ′′*Option.option*′′) [*Typerep.typerep TYPE*(′*a*)]]]))
(*t* ())))

**definition** *enum-term-of-option* :: ′*a option itself* ⇒ *unit* ⇒ *term list*
  **where** *enum-term-of-option* =
    (λ - -.
      *Code-Evaluation.term-of* (*None* :: ′*a option*) #
      (*map*
        (*Code-Evaluation.App*
          (*Code-Evaluation.Const* (*STR* ′′*Option.option.Some*′′)
            (*Typerep.Typerep* (*STR* ′′*fun*′′)
              [*Typerep.typerep TYPE*(′*a*),
            *Typerep.Typerep* (*STR* ′′*Option.option*′′) [*Typerep.typerep TYPE*(′*a*)]]])))
        (*enum-term-of* (*TYPE*(′*a*)) ())))

**instance** ⟨*proof*⟩

**end**

**instantiation** *Enum.finite-1* :: *check-all*
**begin**

**definition** *check-all f* = *f* (*Code-Evaluation.valtermify Enum.finite-1*.$a_1$)

**definition** *enum-term-of-finite-1* :: *Enum.finite-1 itself* ⇒ *unit* ⇒ *term list*
 **where** *enum-term-of-finite-1* = (λ- -. [*Code-Evaluation.term-of Enum.finite-1*.$a_1$])

**instance** ⟨*proof*⟩

**end**

**instantiation** *Enum.finite-2* :: *check-all*
**begin**

**definition**
*check-all f* =
($f$ (*Code-Evaluation.valtermify Enum.finite-2*.$a_1$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-2*.$a_2$))

**definition** *enum-term-of-finite-2* :: *Enum.finite-2 itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-finite-2* =
(λ- -. *map Code-Evaluation.term-of* (*Enum.enum* :: *Enum.finite-2 list*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *Enum.finite-3* :: *check-all*
**begin**

**definition**
*check-all f* =
($f$ (*Code-Evaluation.valtermify Enum.finite-3*.$a_1$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-3*.$a_2$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-3*.$a_3$))

**definition** *enum-term-of-finite-3* :: *Enum.finite-3 itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-finite-3* =
(λ- -. *map Code-Evaluation.term-of* (*Enum.enum* :: *Enum.finite-3 list*))

**instance** ⟨*proof*⟩

**end**

**instantiation** *Enum.finite-4* :: *check-all*
**begin**

**definition**
*check-all f* =
$f$ (*Code-Evaluation.valtermify Enum.finite-4*.$a_1$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-4*.$a_2$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-4*.$a_3$) *orelse*
$f$ (*Code-Evaluation.valtermify Enum.finite-4*.$a_4$)

**definition** *enum-term-of-finite-4* :: *Enum.finite-4 itself* ⇒ *unit* ⇒ *term list*
**where** *enum-term-of-finite-4* =
(λ- -. *map Code-Evaluation.term-of* (*Enum.enum* :: *Enum.finite-4 list*))

**instance** ⟨*proof*⟩

**end**

## 82.3 Bounded universal quantifiers

**class** *bounded-forall* =
**fixes** *bounded-forall* :: ('*a* ⇒ *bool*) ⇒ *natural* ⇒ *bool*

## 82.4 Fast exhaustive combinators

**class** *fast-exhaustive* = *term-of* +
**fixes** *fast-exhaustive* :: ('*a* ⇒ *unit*) ⇒ *natural* ⇒ *unit*

**axiomatization** *throw-Counterexample* :: *term list* ⇒ *unit*
**axiomatization** *catch-Counterexample* :: *unit* ⇒ *term list option*

**code-printing**
**constant** *throw-Counterexample* ⇀
(*Quickcheck*) *raise* (*Exhaustive'-Generators.Counterexample* -)
| **constant** *catch-Counterexample* ⇀
(*Quickcheck*) (((-); *NONE*) *handle Exhaustive'-Generators.Counterexample ts* ⇒ *SOME ts*)

## 82.5 Continuation passing style functions as plus monad

**type-synonym** '*a cps* = ('*a* ⇒ *term list option*) ⇒ *term list option*

**definition** *cps-empty* :: '*a cps*
**where** *cps-empty* = (λ*cont*. *None*)

**definition** *cps-single* :: '*a* ⇒ '*a cps*
**where** *cps-single v* = (λ*cont*. *cont v*)

**definition** *cps-bind* :: '*a cps* ⇒ ('*a* ⇒ '*b cps*) ⇒ '*b cps*
**where** *cps-bind m f* = (λ*cont*. *m* (λ*a*. (*f a*) *cont*))

**definition** *cps-plus* :: '*a cps* ⇒ '*a cps* ⇒ '*a cps*
**where** *cps-plus a b* = (λ*c*. *case a c of None* ⇒ *b c* | *Some x* ⇒ *Some x*)

**definition** *cps-if* :: *bool* ⇒ *unit cps*
**where** *cps-if b* = (*if b then cps-single* () *else cps-empty*)

**definition** *cps-not* :: *unit cps* ⇒ *unit cps*
**where** *cps-not n* = (λ*c*. *case n* (λ*u*. *Some* []) *of None* ⇒ *c* () | *Some* - ⇒ *None*)

**type-synonym** '*a pos-bound-cps* =
('*a* ⇒ (*bool* ∗ *term list*) *option*) ⇒ *natural* ⇒ (*bool* ∗ *term list*) *option*

**definition** *pos-bound-cps-empty* :: '*a pos-bound-cps*
**where** *pos-bound-cps-empty* = (λ*cont i*. *None*)

**definition** *pos-bound-cps-single* :: $'a \Rightarrow 'a$ *pos-bound-cps*
**where** *pos-bound-cps-single* $v = (\lambda cont\ i.\ cont\ v)$

**definition** *pos-bound-cps-bind* :: $'a$ *pos-bound-cps* $\Rightarrow$ ($'a \Rightarrow 'b$ *pos-bound-cps*) $\Rightarrow$ $'b$ *pos-bound-cps*
**where** *pos-bound-cps-bind* $m\ f = (\lambda cont\ i.\ if\ i = 0\ then\ None\ else\ (m\ (\lambda a.\ (f\ a)\ cont\ i)\ (i - 1)))$

**definition** *pos-bound-cps-plus* :: $'a$ *pos-bound-cps* $\Rightarrow$ $'a$ *pos-bound-cps* $\Rightarrow$ $'a$ *pos-bound-cps*
**where** *pos-bound-cps-plus* $a\ b = (\lambda c\ i.\ case\ a\ c\ i\ of\ None \Rightarrow b\ c\ i\ |\ Some\ x \Rightarrow Some\ x)$

**definition** *pos-bound-cps-if* :: $bool \Rightarrow unit$ *pos-bound-cps*
**where** *pos-bound-cps-if* $b$ = (*if b then pos-bound-cps-single* () *else pos-bound-cps-empty*)

**datatype** (*plugins only*: *code extraction*) (*dead* $'a$) *unknown* =
*Unknown* | *Known* $'a$

**datatype** (*plugins only*: *code extraction*) (*dead* $'a$) *three-valued* =
*Unknown-value* | *Value* $'a$ | *No-value*

**type-synonym** $'a$ *neg-bound-cps* =
($'a$ *unknown* $\Rightarrow$ *term list three-valued*) $\Rightarrow$ *natural* $\Rightarrow$ *term list three-valued*

**definition** *neg-bound-cps-empty* :: $'a$ *neg-bound-cps*
**where** *neg-bound-cps-empty* = ($\lambda$*cont i. No-value*)

**definition** *neg-bound-cps-single* :: $'a \Rightarrow 'a$ *neg-bound-cps*
**where** *neg-bound-cps-single* $v$ = ($\lambda$*cont i. cont* (*Known* $v$))

**definition** *neg-bound-cps-bind* :: $'a$ *neg-bound-cps* $\Rightarrow$ ($'a \Rightarrow 'b$ *neg-bound-cps*) $\Rightarrow$ $'b$ *neg-bound-cps*
**where** *neg-bound-cps-bind* $m\ f$ =
($\lambda$*cont i.*
*if* $i = 0$ *then cont Unknown*
*else m* ($\lambda a$. *case a of Unknown* $\Rightarrow$ *cont Unknown* | *Known* $a' \Rightarrow f\ a'$ *cont i*) ($i - 1$))

**definition** *neg-bound-cps-plus* :: $'a$ *neg-bound-cps* $\Rightarrow$ $'a$ *neg-bound-cps* $\Rightarrow$ $'a$ *neg-bound-cps*
**where** *neg-bound-cps-plus* $a\ b$ =
($\lambda c\ i$.
*case a c i of*
*No-value* $\Rightarrow$ *b c i*
| *Value x* $\Rightarrow$ *Value x*
| *Unknown-value* $\Rightarrow$
(*case b c i of*
*No-value* $\Rightarrow$ *Unknown-value*
| *Value x* $\Rightarrow$ *Value x*

| *Unknown-value* ⇒ *Unknown-value*))

**definition** *neg-bound-cps-if* :: *bool* ⇒ *unit neg-bound-cps*
**where** *neg-bound-cps-if b* = (*if b then neg-bound-cps-single* () *else neg-bound-cps-empty*)

**definition** *neg-bound-cps-not* :: *unit pos-bound-cps* ⇒ *unit neg-bound-cps*
**where** *neg-bound-cps-not n* =
(λ*c i. case n* (λ*u. Some* (*True*, [])) *i of None* ⇒ *c* (*Known* ()) | *Some* - ⇒ *No-value*)

**definition** *pos-bound-cps-not* :: *unit neg-bound-cps* ⇒ *unit pos-bound-cps*
**where** *pos-bound-cps-not n* =
(λ*c i. case n* (λ*u. Value* []) *i of No-value* ⇒ *c* () | *Value* - ⇒ *None* | *Unknown-value* ⇒ *None*)

## 82.6 Defining generators for any first-order data type

**axiomatization** *unknown* :: *'a*

**notation** (**output**) *unknown* (‹?›)

⟨*ML*⟩

**declare** [[*quickcheck-batch-tester* = *exhaustive*]]

## 82.7 Defining generators for abstract types

⟨*ML*⟩

**hide-fact** (**open**) *orelse-def*
**no-notation** *orelse* (**infixr** ‹*orelse*› *55*)

**hide-const** *valtermify-absdummy valtermify-fun-upd*
*valterm-emptyset valtermify-insert*
*valtermify-pair valtermify-Inl valtermify-Inr*
*termify-fun-upd term-emptyset termify-insert termify-pair setify*

**hide-const** (**open**)
*exhaustive full-exhaustive*
*exhaustive-int' full-exhaustive-int'*
*exhaustive-integer' full-exhaustive-integer'*
*exhaustive-natural' full-exhaustive-natural'*
*throw-Counterexample catch-Counterexample*
*check-all enum-term-of*
*orelse unknown mk-map-term check-all-n-lists check-all-subsets*

**hide-type** (**open**) *cps pos-bound-cps neg-bound-cps unknown three-valued*

**hide-const** (**open**) *cps-empty cps-single cps-bind cps-plus cps-if cps-not*
*pos-bound-cps-empty pos-bound-cps-single pos-bound-cps-bind*

*pos-bound-cps-plus pos-bound-cps-if pos-bound-cps-not*
*neg-bound-cps-empty neg-bound-cps-single neg-bound-cps-bind*
*neg-bound-cps-plus neg-bound-cps-if neg-bound-cps-not*
*Unknown Known Unknown-value Value No-value*

**end**

# 83 A compiler for predicates defined by introduction rules

**theory** *Predicate-Compile*
**imports** *Random-Sequence Quickcheck-Exhaustive*
**keywords**
*code-pred* :: *thy-goal* **and**
*values* :: *diag*
**begin**

⟨*ML*⟩

## 83.1 Set membership as a generator predicate

Introduce a new constant for membership to allow fine-grained control in code equations.

**definition** *contains* :: *'a set => 'a => bool*
**where** *contains A x* $\longleftrightarrow$ $x \in A$

**definition** *contains-pred* :: *'a set => 'a => unit Predicate.pred*
**where** *contains-pred A x* = (*if* $x \in A$ *then Predicate.single* () *else bot*)

**lemma** *pred-of-setE*:
**assumes** *Predicate.eval* (*pred-of-set A*) *x*
**obtains** *contains A x*
⟨*proof*⟩

**lemma** *pred-of-setI*: *contains A x ==> Predicate.eval* (*pred-of-set A*) *x*
⟨*proof*⟩

**lemma** *pred-of-set-eq*: *pred-of-set* ≡ $\lambda A$. *Predicate.Pred* (*contains A*)
⟨*proof*⟩

**lemma** *containsI*: $x \in A$ *==> contains A x*
⟨*proof*⟩

**lemma** *containsE*: **assumes** *contains A x*
**obtains** $A'$ $x'$ **where** $A = A'$ $x = x'$ $x \in A$
⟨*proof*⟩

**lemma** *contains-predI*: *contains A x ==> Predicate.eval* (*contains-pred A x*) ()

⟨*proof*⟩

**lemma** *contains-predE*:
**assumes** *Predicate.eval* (*contains-pred A x*) *y*
**obtains** *contains A x*
⟨*proof*⟩

**lemma** *contains-pred-eq*: *contains-pred* $\equiv$ $\lambda A\ x.$ *Predicate.Pred* ($\lambda y.$ *contains A x*)
⟨*proof*⟩

**lemma** *contains-pred-notI*:
$\neg$ *contains A x* ==> *Predicate.eval* (*Predicate.not-pred* (*contains-pred A x*)) ()
⟨*proof*⟩

⟨*ML*⟩

**hide-const** (**open**) *contains contains-pred*
**hide-fact** (**open**) *pred-of-setE pred-of-setI pred-of-set-eq*
*containsI containsE contains-predI contains-predE contains-pred-eq contains-pred-notI*

**end**

# 84 Counterexample generator performing narrowing-based testing

**theory** *Quickcheck-Narrowing*
**imports** *Quickcheck-Random*
**keywords** *find-unused-assms* :: *diag*
**begin**

## 84.1 Counterexample generator

### 84.1.1 Code generation setup

⟨*ML*⟩

**code-printing**
**code-module** *Typerep* ⇀ (*Haskell-Quickcheck*) ‹
*module Typerep*(*Typerep*(..)) *where*

*data Typerep* = *Typerep String* [*Typerep*]
› **for type-constructor** *typerep* **constant** *Typerep.Typerep*
| **type-constructor** *typerep* ⇀ (*Haskell-Quickcheck*) *Typerep.Typerep*
| **constant** *Typerep.Typerep* ⇀ (*Haskell-Quickcheck*) *Typerep.Typerep*

**code-reserved**
(*Haskell-Quickcheck*) *Typerep*

**code-printing**

**type-constructor** *integer* ⇀ (*Haskell-Quickcheck*) *Prelude.Int*
| **constant** *0*::*integer* ⇀
(*Haskell-Quickcheck*) !(*0*/ ::/ *Prelude.Int*)

⟨*ML*⟩

**code-printing**
**constant** *Code-Numeral.push-bit* ⇀
(*Haskell-Quickcheck*) *Bit′-Shifts.drop′*
| **constant** *Code-Numeral.drop-bit* ⇀
(*Haskell-Quickcheck*) *Bit′-Shifts.push′*

### 84.1.2 Narrowing's deep representation of types and terms

**datatype** (*plugins only*: *code extraction*) *narrowing-type* =
*Narrowing-sum-of-products narrowing-type list list*

**datatype** (*plugins only*: *code extraction*) *narrowing-term* =
*Narrowing-variable integer list narrowing-type*
| *Narrowing-constructor integer narrowing-term list*

**datatype** (*plugins only*: *code extraction*) (*dead ′a*) *narrowing-cons* =
*Narrowing-cons narrowing-type* (*narrowing-term list* ⇒ *′a*) *list*

**primrec** *map-cons* :: (*′a* => *′b*) => *′a narrowing-cons* => *′b narrowing-cons*
**where**
*map-cons f* (*Narrowing-cons ty cs*) = *Narrowing-cons ty* (*map* ($\lambda c.\ f \circ c$) *cs*)

### 84.1.3 From narrowing's deep representation of terms to *HOL.Code-Evaluation*'s terms

**class** *partial-term-of* = *typerep* +
**fixes** *partial-term-of* :: *′a itself* => *narrowing-term* => *Code-Evaluation.term*

**lemma** *partial-term-of-anything*: *partial-term-of x nt* ≡ *t*
⟨*proof*⟩

### 84.1.4 Auxilary functions for Narrowing

**consts** *nth* :: *′a list* => *integer* => *′a*

**code-printing constant** *nth* ⇀ (*Haskell-Quickcheck*) **infixl** *9* !!

**consts** *error* :: *char list* => *′a*

**code-printing constant** *error* ⇀ (*Haskell-Quickcheck*) *error*

**consts** *toEnum* :: *integer* => *char*

**code-printing constant** *toEnum* ⇀ (*Haskell-Quickcheck*) *Prelude.toEnum*

**consts** *marker* :: *char*

**code-printing constant** *marker* ⇀ (*Haskell-Quickcheck*) ''\0''

### 84.1.5 Narrowing's basic operations

**type-synonym** *'a narrowing* = *integer* => *'a narrowing-cons*

**definition** *cons* :: *'a* => *'a narrowing*
**where**
*cons a d* = (*Narrowing-cons* (*Narrowing-sum-of-products* [[]]) [(λ-. *a*)])

**fun** *conv* :: (*narrowing-term list* => *'a*) *list* => *narrowing-term* => *'a*
**where**
*conv cs* (*Narrowing-variable p* -) = *error* (*marker* # *map toEnum p*)
| *conv cs* (*Narrowing-constructor i xs*) = (*nth cs i*) *xs*

**fun** *non-empty* :: *narrowing-type* => *bool*
**where**
*non-empty* (*Narrowing-sum-of-products ps*) = (¬ (*List.null ps*))

**definition** *apply* :: (*'a* => *'b*) *narrowing* => *'a narrowing* => *'b narrowing*
**where**
*apply f a d* = (*if d* > *0 then*
(*case f d of Narrowing-cons* (*Narrowing-sum-of-products ps*) *cfs* ⇒
*case a* (*d* − *1*) *of Narrowing-cons ta cas* ⇒
*let*
*shallow* = *non-empty ta*;
*cs* = [(λ(*x* # *xs*) ⇒ *cf xs* (*conv cas x*)). *shallow*, *cf* ← *cfs*]
*in Narrowing-cons* (*Narrowing-sum-of-products* [*ta* # *p*. *shallow*, *p* ← *ps*]) *cs*)
*else Narrowing-cons* (*Narrowing-sum-of-products* []) [])

**definition** *sum* :: *'a narrowing* => *'a narrowing* => *'a narrowing*
**where**
*sum a b d* =
(*case a d of Narrowing-cons* (*Narrowing-sum-of-products ssa*) *ca* ⇒
*case b d of Narrowing-cons* (*Narrowing-sum-of-products ssb*) *cb* ⇒
*Narrowing-cons* (*Narrowing-sum-of-products* (*ssa* @ *ssb*)) (*ca* @ *cb*))

**lemma** [*fundef-cong*]:
**assumes** $a\ d = a'\ d$ $b\ d = b'\ d$ $d = d'$
**shows** $sum\ a\ b\ d = sum\ a'\ b'\ d'$
⟨*proof*⟩

**lemma** [*fundef-cong*]:
**assumes** $f\ d = f'\ d$ $(\bigwedge d'.\ 0 \le d' \wedge d' < d \Longrightarrow a\ d' = a'\ d')$
**assumes** $d = d'$

**shows** *apply f a d = apply f′ a′ d′*
⟨*proof*⟩

### 84.1.6 Narrowing generator type class

**class** *narrowing* =
**fixes** *narrowing* :: *integer* => ′*a narrowing-cons*

**datatype** (*plugins only*: *code extraction*) *property* =
*Universal narrowing-type* (*narrowing-term* => *property*) *narrowing-term* => *Code-Evaluation.term*
| *Existential narrowing-type* (*narrowing-term* => *property*) *narrowing-term* => *Code-Evaluation.term*
| *Property bool*

**definition** *exists* :: (′*a* :: {*narrowing*, *partial-term-of*} => *property*) => *property*
**where**
*exists f* = (*case narrowing* (*100* :: *integer*) *of Narrowing-cons ty cs* ⇒ *Existential ty* (λ *t*. *f* (*conv cs t*)) (*partial-term-of* (*TYPE*(′*a*))))

**definition** *all* :: (′*a* :: {*narrowing*, *partial-term-of*} => *property*) => *property*
**where**
*all f* = (*case narrowing* (*100* :: *integer*) *of Narrowing-cons ty cs* ⇒ *Universal ty* (λ*t*. *f* (*conv cs t*)) (*partial-term-of* (*TYPE*(′*a*))))

### 84.1.7 class *is-testable*

The class *is-testable* ensures that all necessary type instances are generated.

**class** *is-testable*

**instance** *bool* :: *is-testable* ⟨*proof*⟩

**instance** *fun* :: ({*term-of*, *narrowing*, *partial-term-of*}, *is-testable*) *is-testable* ⟨*proof*⟩

**definition** *ensure-testable* :: ′*a* :: *is-testable* => ′*a* :: *is-testable*
**where**
*ensure-testable f* = *f*

### 84.1.8 Defining a simple datatype to represent functions in an incomplete and redundant way

**datatype** (*plugins only*: *code quickcheck-narrowing extraction*) (*dead* ′*a*, *dead* ′*b*) *ffun* =
*Constant* ′*b*
| *Update* ′*a* ′*b* (′*a*, ′*b*) *ffun*

**primrec** *eval-ffun* :: (′*a*, ′*b*) *ffun* => ′*a* => ′*b*
**where**

*eval-ffun* (*Constant c*) $x = c$
| *eval-ffun* (*Update* $x'$ *y f*) $x$ = (*if* $x = x'$ *then y else eval-ffun f x*)

**hide-type** (**open**) *ffun*
**hide-const** (**open**) *Constant Update eval-ffun*

**datatype** (*plugins only*: *code quickcheck-narrowing extraction*) (*dead 'b*) *cfun* = *Constant 'b*

**primrec** *eval-cfun* :: *'b cfun => 'a => 'b*
**where**
*eval-cfun* (*Constant c*) $y = c$

**hide-type** (**open**) *cfun*
**hide-const** (**open**) *Constant eval-cfun Abs-cfun Rep-cfun*

### 84.1.9 Setting up the counterexample generator

**external-file** ‹~~/*src/HOL/Tools/Quickcheck/Narrowing-Engine.hs*›
**external-file** ‹~~/*src/HOL/Tools/Quickcheck/PNF-Narrowing-Engine.hs*›
⟨*ML*⟩

**definition** *narrowing-dummy-partial-term-of* :: (*'a* :: *partial-term-of*) *itself* => *narrowing-term* => *term*
**where**
*narrowing-dummy-partial-term-of* = *partial-term-of*

**definition** *narrowing-dummy-narrowing* :: *integer* => (*'a* :: *narrowing*) *narrowing-cons*
**where**
*narrowing-dummy-narrowing* = *narrowing*

**lemma** [*code*]:
*ensure-testable f* =
(*let*
*x* = *narrowing-dummy-narrowing* :: *integer* => *bool narrowing-cons*;
*y* = *narrowing-dummy-partial-term-of* :: *bool itself* => *narrowing-term* => *term*;
*z* = (*conv* :: - => - => *unit*) *in f*)
⟨*proof*⟩

## 84.2 Narrowing for sets

**instantiation** *set* :: (*narrowing*) *narrowing*
**begin**

**definition** *narrowing-set* = *Quickcheck-Narrowing.apply* (*Quickcheck-Narrowing.cons set*) *narrowing*

**instance** ⟨*proof*⟩

**end**

## 84.3 Narrowing for integers

**definition** *drawn-from* :: *'a list* ⇒ *'a narrowing-cons*
**where**
*drawn-from xs* =
*Narrowing-cons* (*Narrowing-sum-of-products* (*map* (λ-. []) *xs*)) (*map* (λ*x* -. *x*) *xs*)

**function** *around-zero* :: *int* ⇒ *int list*
**where**
*around-zero i* = (*if i* < *0 then* [] *else* (*if i* = *0 then* [*0*] *else around-zero* (*i* − *1*) @ [*i*, −*i*]))
⟨*proof*⟩
**termination** ⟨*proof*⟩

**declare** *around-zero.simps* [*simp del*]

**lemma** *length-around-zero*:
**assumes** *i* >= *0*
**shows** *length* (*around-zero i*) = *2* ∗ *nat i* + *1*
⟨*proof*⟩

**instantiation** *int* :: *narrowing*
**begin**

**definition**
*narrowing-int d* = (*let* (*u* :: - ⇒ - ⇒ *unit*) = *conv*; *i* = *int-of-integer d*
*in drawn-from* (*around-zero i*))

**instance** ⟨*proof*⟩

**end**

**lemma** [*code*]:
*partial-term-of* (*ty* :: *int itself*) (*Narrowing-variable p t*) ≡
*Code-Evaluation.Free* (*STR* ''-'') (*Typerep.Typerep* (*STR* ''Int.int'') [])
*partial-term-of* (*ty* :: *int itself*) (*Narrowing-constructor i* []) ≡
(*if i mod 2* = *0*
*then Code-Evaluation.term-of* (− (*int-of-integer i*) *div 2*)
*else Code-Evaluation.term-of* ((*int-of-integer i* + *1*) *div 2*))
⟨*proof*⟩

**instantiation** *integer* :: *narrowing*
**begin**

**definition**

*narrowing-integer d* = (*let* (*u* :: - ⇒ - ⇒ *unit*) = *conv*; *i* = *int-of-integer d*
*in drawn-from* (*map integer-of-int* (*around-zero i*)))

**instance** ⟨*proof*⟩

**end**

**lemma** [*code*]:
*partial-term-of* (*ty* :: *integer itself*) (*Narrowing-variable p t*) ≡
*Code-Evaluation.Free* (*STR ''-''*) (*Typerep.Typerep* (*STR ''Code-Numeral.integer''*) [])
*partial-term-of* (*ty* :: *integer itself*) (*Narrowing-constructor i* []) ≡
(*if i mod 2 = 0*
*then Code-Evaluation.term-of* (− *i div 2*)
*else Code-Evaluation.term-of* ((*i* + *1*) *div 2*))
⟨*proof*⟩

**code-printing constant** *Code-Evaluation.term-of* :: *integer* ⇒ *term* ⇀ (*Haskell-Quickcheck*)

(*let* { *t* = *Typerep.Typerep Code′-Numeral.integer* [];
*mkFunT s t* = *Typerep.Typerep fun* [*s*, *t*];
*numT* = *Typerep.Typerep Num.num* [];
*mkBit 0* = *Generated′-Code.Const Num.num.Bit0* (*mkFunT numT numT*);
*mkBit 1* = *Generated′-Code.Const Num.num.Bit1* (*mkFunT numT numT*);
*mkNumeral 1* = *Generated′-Code.Const Num.num.One numT*;
*mkNumeral i* = *let* { *q* = *i* '*Prelude.div*' *2*; *r* = *i* '*Prelude.mod*' *2* }
*in Generated′-Code.App* (*mkBit r*) (*mkNumeral q*);
*mkNumber 0* = *Generated′-Code.Const Groups.zero′-class.zero t*;
*mkNumber 1* = *Generated′-Code.Const Groups.one′-class.one t*;
*mkNumber i* = *if i* > *0 then*
*Generated′-Code.App*
(*Generated′-Code.Const Num.numeral′-class.numeral*
(*mkFunT numT t*))
(*mkNumeral i*)
*else*
*Generated′-Code.App*
(*Generated′-Code.Const Groups.uminus′-class.uminus* (*mkFunT t t*))
(*mkNumber* (− *i*)); } *in mkNumber*)

## 84.4 The *find-unused-assms* command

⟨*ML*⟩

## 84.5 Closing up

**hide-type** *narrowing-type narrowing-term narrowing-cons property*
**hide-const** *map-cons nth error toEnum marker empty Narrowing-cons conv non-empty ensure-testable all exists drawn-from around-zero*
**hide-const** (**open**) *Narrowing-variable Narrowing-constructor apply sum cons*

**hide-fact** *empty-def cons-def conv.simps non-empty.simps apply-def sum-def ensure-testable-def all-def exists-def*

**end**

**theory** *Mirabelle*
**imports** *Sledgehammer Predicate-Compile Presburger*
**begin**

⟨*ML*⟩

**end**

# 85 Program extraction for HOL

**theory** *Extraction*
**imports** *Option*
**begin**

## 85.1 Setup

⟨*ML*⟩

**lemmas** [*extraction-expand*] =
*meta-spec atomize-eq atomize-all atomize-imp atomize-conj*
*allE rev-mp conjE Eq-TrueI Eq-FalseI eqTrueI eqTrueE eq-cong2*
*notE′ impE′ impE iffE imp-cong simp-thms eq-True eq-False*
*induct-forall-eq induct-implies-eq induct-equal-eq induct-conj-eq*
*induct-atomize induct-atomize′ induct-rulify induct-rulify′*
*induct-rulify-fallback induct-trueI*
*True-implies-equals implies-True-equals TrueE*
*False-implies-equals implies-False-swap*

**lemmas** [*extraction-expand-def*] =
*HOL.induct-forall-def HOL.induct-implies-def HOL.induct-equal-def HOL.induct-conj-def*
*HOL.induct-true-def HOL.induct-false-def*

**datatype** (*plugins only*: *code extraction*) *sumbool* = *Left* | *Right*

## 85.2 Type of extracted program

**extract-type**
$\mathit{typeof}\ (\mathit{Trueprop}\ P) \equiv \mathit{typeof}\ P$

$\mathit{typeof}\ P \equiv \mathit{Type}\ (\mathit{TYPE}(\mathit{Null})) \Longrightarrow \mathit{typeof}\ Q \equiv \mathit{Type}\ (\mathit{TYPE}('Q)) \Longrightarrow$
$\quad \mathit{typeof}\ (P \longrightarrow Q) \equiv \mathit{Type}\ (\mathit{TYPE}('Q))$

$\mathit{typeof}\ Q \equiv \mathit{Type}\ (\mathit{TYPE}(\mathit{Null})) \Longrightarrow \mathit{typeof}\ (P \longrightarrow Q) \equiv \mathit{Type}\ (\mathit{TYPE}(\mathit{Null}))$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE('Q)) \Longrightarrow$
$typeof\ (P \longrightarrow Q) \equiv Type\ (TYPE('P \Rightarrow 'Q))$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE(Null))) \Longrightarrow$
$typeof\ (\forall x.\ P\ x) \equiv Type\ (TYPE(Null))$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE('P))) \Longrightarrow$
$typeof\ (\forall x::'a.\ P\ x) \equiv Type\ (TYPE('a \Rightarrow 'P))$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE(Null))) \Longrightarrow$
$typeof\ (\exists x::'a.\ P\ x) \equiv Type\ (TYPE('a))$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE('P))) \Longrightarrow$
$typeof\ (\exists x::'a.\ P\ x) \equiv Type\ (TYPE('a \times 'P))$

$typeof\ P \equiv Type\ (TYPE(Null)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE(Null)) \Longrightarrow$
$typeof\ (P \vee Q) \equiv Type\ (TYPE(sumbool))$

$typeof\ P \equiv Type\ (TYPE(Null)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE('Q)) \Longrightarrow$
$typeof\ (P \vee Q) \equiv Type\ (TYPE('Q\ option))$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE(Null)) \Longrightarrow$
$typeof\ (P \vee Q) \equiv Type\ (TYPE('P\ option))$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE('Q)) \Longrightarrow$
$typeof\ (P \vee Q) \equiv Type\ (TYPE('P + 'Q))$

$typeof\ P \equiv Type\ (TYPE(Null)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE('Q)) \Longrightarrow$
$typeof\ (P \wedge Q) \equiv Type\ (TYPE('Q))$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE(Null)) \Longrightarrow$
$typeof\ (P \wedge Q) \equiv Type\ (TYPE('P))$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow typeof\ Q \equiv Type\ (TYPE('Q)) \Longrightarrow$
$typeof\ (P \wedge Q) \equiv Type\ (TYPE('P \times 'Q))$

$typeof\ (P = Q) \equiv typeof\ ((P \longrightarrow Q) \wedge (Q \longrightarrow P))$

$typeof\ (x \in P) \equiv typeof\ P$

## 85.3 Realizability

**realizability**
$(realizes\ t\ (Trueprop\ P)) \equiv (Trueprop\ (realizes\ t\ P))$

$(typeof\ P) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \longrightarrow Q)) \equiv (realizes\ Null\ P \longrightarrow realizes\ t\ Q)$

$(typeof\ P) \equiv (Type\ (TYPE('P))) \Longrightarrow$
$(typeof\ Q) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \longrightarrow Q)) \equiv (\forall\, x::'P.\ realizes\ x\ P \longrightarrow realizes\ Null\ Q)$

$(realizes\ t\ (P \longrightarrow Q)) \equiv (\forall\, x.\ realizes\ x\ P \longrightarrow realizes\ (t\ x)\ Q)$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (\forall\, x.\ P\ x)) \equiv (\forall\, x.\ realizes\ Null\ (P\ x))$

$(realizes\ t\ (\forall\, x.\ P\ x)) \equiv (\forall\, x.\ realizes\ (t\ x)\ (P\ x))$

$(\lambda x.\ typeof\ (P\ x)) \equiv (\lambda x.\ Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (\exists\, x.\ P\ x)) \equiv (realizes\ Null\ (P\ t))$

$(realizes\ t\ (\exists\, x.\ P\ x)) \equiv (realizes\ (snd\ t)\ (P\ (fst\ t)))$

$(typeof\ P) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(typeof\ Q) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \vee Q)) \equiv$
$(case\ t\ of\ Left \Rightarrow realizes\ Null\ P \mid Right \Rightarrow realizes\ Null\ Q)$

$(typeof\ P) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \vee Q)) \equiv$
$(case\ t\ of\ None \Rightarrow realizes\ Null\ P \mid Some\ q \Rightarrow realizes\ q\ Q)$

$(typeof\ Q) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \vee Q)) \equiv$
$(case\ t\ of\ None \Rightarrow realizes\ Null\ Q \mid Some\ p \Rightarrow realizes\ p\ P)$

$(realizes\ t\ (P \vee Q)) \equiv$
$(case\ t\ of\ Inl\ p \Rightarrow realizes\ p\ P \mid Inr\ q \Rightarrow realizes\ q\ Q)$

$(typeof\ P) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \wedge Q)) \equiv (realizes\ Null\ P \wedge realizes\ t\ Q)$

$(typeof\ Q) \equiv (Type\ (TYPE(Null))) \Longrightarrow$
$(realizes\ t\ (P \wedge Q)) \equiv (realizes\ t\ P \wedge realizes\ Null\ Q)$

$(realizes\ t\ (P \wedge Q)) \equiv (realizes\ (fst\ t)\ P \wedge realizes\ (snd\ t)\ Q)$

$typeof\ P \equiv Type\ (TYPE(Null)) \Longrightarrow$
$realizes\ t\ (\neg\ P) \equiv \neg\ realizes\ Null\ P$

$typeof\ P \equiv Type\ (TYPE('P)) \Longrightarrow$
$realizes\ t\ (\neg\ P) \equiv (\forall\, x::'P.\ \neg\ realizes\ x\ P)$

$typeof\ (P::bool) \equiv Type\ (TYPE(Null)) \Longrightarrow$
$typeof\ Q \equiv Type\ (TYPE(Null)) \Longrightarrow$
$realizes\ t\ (P = Q) \equiv realizes\ Null\ P = realizes\ Null\ Q$

$(realizes\ t\ (P = Q)) \equiv (realizes\ t\ ((P \longrightarrow Q) \wedge (Q \longrightarrow P)))$

## 85.4 Computational content of basic inference rules

**theorem** *disjE-realizer*:
**assumes** $r$: $case\ x\ of\ Inl\ p \Rightarrow P\ p \mid Inr\ q \Rightarrow Q\ q$
**and** $r1$: $\bigwedge p.\ P\ p \Longrightarrow R\ (f\ p)$ **and** $r2$: $\bigwedge q.\ Q\ q \Longrightarrow R\ (g\ q)$
**shows** $R\ (case\ x\ of\ Inl\ p \Rightarrow f\ p \mid Inr\ q \Rightarrow g\ q)$
⟨*proof*⟩

**theorem** *disjE-realizer2*:
**assumes** $r$: $case\ x\ of\ None \Rightarrow P \mid Some\ q \Rightarrow Q\ q$
**and** $r1$: $P \Longrightarrow R\ f$ **and** $r2$: $\bigwedge q.\ Q\ q \Longrightarrow R\ (g\ q)$
**shows** $R\ (case\ x\ of\ None \Rightarrow f \mid Some\ q \Rightarrow g\ q)$
⟨*proof*⟩

**theorem** *disjE-realizer3*:
**assumes** $r$: $case\ x\ of\ Left \Rightarrow P \mid Right \Rightarrow Q$
**and** $r1$: $P \Longrightarrow R\ f$ **and** $r2$: $Q \Longrightarrow R\ g$
**shows** $R\ (case\ x\ of\ Left \Rightarrow f \mid Right \Rightarrow g)$
⟨*proof*⟩

**theorem** *conjI-realizer*:
$P\ p \Longrightarrow Q\ q \Longrightarrow P\ (fst\ (p,\ q)) \wedge Q\ (snd\ (p,\ q))$
⟨*proof*⟩

**theorem** *exI-realizer*:
$P\ y\ x \Longrightarrow P\ (snd\ (x,\ y))\ (fst\ (x,\ y))$ ⟨*proof*⟩

**theorem** *exE-realizer*: $P\ (snd\ p)\ (fst\ p) \Longrightarrow$
$(\bigwedge x\ y.\ P\ y\ x \Longrightarrow Q\ (f\ x\ y)) \Longrightarrow Q\ (let\ (x,\ y) = p\ in\ f\ x\ y)$
⟨*proof*⟩

**theorem** *exE-realizer'*: $P\ (snd\ p)\ (fst\ p) \Longrightarrow$
$(\bigwedge x\ y.\ P\ y\ x \Longrightarrow Q) \Longrightarrow Q$ ⟨*proof*⟩

**realizers**
$impI\ (P,\ Q)$: $\lambda pq.\ pq$
$\boldsymbol{\lambda}(c\text{: -})\ (d\text{: -})\ P\ Q\ pq\ (h\text{: -}).\ allI \cdot \text{-} \cdot c \cdot (\boldsymbol{\lambda}x.\ impI \cdot \text{-} \cdot \text{-} \cdot (h \cdot x))$

$impI\ (P)$: $Null$
$\boldsymbol{\lambda}(c\text{: -})\ P\ Q\ (h\text{: -}).\ allI \cdot \text{-} \cdot c \cdot (\boldsymbol{\lambda}x.\ impI \cdot \text{-} \cdot \text{-} \cdot (h \cdot x))$

$impI\ (Q)$: $\lambda q.\ q\ \boldsymbol{\lambda}(c\text{: -})\ P\ Q\ q.\ impI \cdot \text{-} \cdot \text{-}$

$impI$: $Null\ impI$

$mp\ (P,\ Q)$: $\lambda pq.\ pq$

$\boldsymbol{\lambda}(c\colon \text{-})\ (d\colon \text{-})\ P\ Q\ pq\ (h\colon \text{-})\ p.\ mp \cdot \text{-} \cdot \text{-} \bullet (spec \cdot \text{-} \cdot p \bullet c \bullet h)$

*mp* (*P*): *Null*
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ (h\colon \text{-})\ p.\ mp \cdot \text{-} \cdot \text{-} \bullet (spec \cdot \text{-} \cdot p \bullet c \bullet h)$

*mp* (*Q*): $\lambda q.\ q$ $\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ q.\ mp \cdot \text{-} \cdot \text{-}$

*mp*: *Null mp*

*allI* (*P*): $\lambda p.\ p$ $\boldsymbol{\lambda}(c\colon \text{-})\ P\ (d\colon \text{-})\ p.\ allI \cdot \text{-} \bullet d$

*allI*: *Null allI*

*spec* (*P*): $\lambda x\ p.\ p\ x$ $\boldsymbol{\lambda}(c\colon \text{-})\ P\ x\ (d\colon \text{-})\ p.\ spec \cdot \text{-} \cdot x \bullet d$

*spec*: *Null spec*

*exI* (*P*): $\lambda x\ p.\ (x,\ p)$ $\boldsymbol{\lambda}(c\colon \text{-})\ P\ x\ (d\colon \text{-})\ p.\ exI\text{-}realizer \cdot P \cdot p \cdot x \bullet c \bullet d$

*exI*: $\lambda x.\ x$ $\boldsymbol{\lambda}P\ x\ (c\colon \text{-})\ (h\colon \text{-}).\ h$

*exE* (*P*, *Q*): $\lambda p\ pq.\ let\ (x,\ y) = p\ in\ pq\ x\ y$
$\boldsymbol{\lambda}(c\colon \text{-})\ (d\colon \text{-})\ P\ Q\ (e\colon \text{-})\ p\ (h\colon \text{-})\ pq.\ exE\text{-}realizer \cdot P \cdot p \cdot Q \cdot pq \bullet c \bullet e \bullet d \bullet h$

*exE* (*P*): *Null*
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ (d\colon \text{-})\ p.\ exE\text{-}realizer' \cdot \text{-} \cdot \text{-} \cdot \text{-} \bullet c \bullet d$

*exE* (*Q*): $\lambda x\ pq.\ pq\ x$
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ (d\colon \text{-})\ x\ (h1\colon \text{-})\ pq\ (h2\colon \text{-}).\ h2 \cdot x \bullet h1$

*exE*: *Null*
$\boldsymbol{\lambda}P\ Q\ (c\colon \text{-})\ x\ (h1\colon \text{-})\ (h2\colon \text{-}).\ h2 \cdot x \bullet h1$

*conjI* (*P*, *Q*): *Pair*
$\boldsymbol{\lambda}(c\colon \text{-})\ (d\colon \text{-})\ P\ Q\ p\ (h\colon \text{-})\ q.\ conjI\text{-}realizer \cdot P \cdot p \cdot Q \cdot q \bullet c \bullet d \bullet h$

*conjI* (*P*): $\lambda p.\ p$
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ p.\ conjI \cdot \text{-} \cdot \text{-}$

*conjI* (*Q*): $\lambda q.\ q$
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ (h\colon \text{-})\ q.\ conjI \cdot \text{-} \cdot \text{-} \bullet h$

*conjI*: *Null conjI*

*conjunct1* (*P*, *Q*): *fst*
$\boldsymbol{\lambda}(c\colon \text{-})\ (d\colon \text{-})\ P\ Q\ pq.\ conjunct1 \cdot \text{-} \cdot \text{-}$

*conjunct1* (*P*): $\lambda p.\ p$
$\boldsymbol{\lambda}(c\colon \text{-})\ P\ Q\ p.\ conjunct1 \cdot \text{-} \cdot \text{-}$

*conjunct1* (*Q*): *Null*
  **λ**(*c*: -) *P Q q*. *conjunct1* · - · -

*conjunct1*: *Null conjunct1*

*conjunct2* (*P*, *Q*): *snd*
  **λ**(*c*: -) (*d*: -) *P Q pq*. *conjunct2* · - · -

*conjunct2* (*P*): *Null*
  **λ**(*c*: -) *P Q p*. *conjunct2* · - · -

*conjunct2* (*Q*): *λp. p*
  **λ**(*c*: -) *P Q p*. *conjunct2* · - · -

*conjunct2*: *Null conjunct2*

*disjI1* (*P*, *Q*): *Inl*
  **λ**(*c*: -) (*d*: -) *P Q p*. *iffD2* · - · - • (*sum.case-1* · *P* · - · *p* • *arity-type-bool* • *c* • *d*)

*disjI1* (*P*): *Some*
  **λ**(*c*: -) *P Q p*. *iffD2* · - · - • (*option.case-2* · - · *P* · *p* • *arity-type-bool* • *c*)

*disjI1* (*Q*): *None*
  **λ**(*c*: -) *P Q*. *iffD2* · - · - • (*option.case-1* · - · - • *arity-type-bool* • *c*)

*disjI1*: *Left*
  **λ***P Q*. *iffD2* · - · - • (*sumbool.case-1* · - · - • *arity-type-bool*)

*disjI2* (*P*, *Q*): *Inr*
  **λ**(*d*: -) (*c*: -) *Q P q*. *iffD2* · - · - • (*sum.case-2* · - · *Q* · *q* • *arity-type-bool* • *c* • *d*)

*disjI2* (*P*): *None*
  **λ**(*c*: -) *Q P*. *iffD2* · - · - • (*option.case-1* · - · - • *arity-type-bool* • *c*)

*disjI2* (*Q*): *Some*
  **λ**(*c*: -) *Q P q*. *iffD2* · - · - • (*option.case-2* · - · *Q* · *q* • *arity-type-bool* • *c*)

*disjI2*: *Right*
  **λ***Q P*. *iffD2* · - · - • (*sumbool.case-2* · - · - • *arity-type-bool*)

*disjE* (*P*, *Q*, *R*): *λpq pr qr*.
  (*case pq of Inl p* ⇒ *pr p* | *Inr q* ⇒ *qr q*)
  **λ**(*c*: -) (*d*: -) (*e*: -) *P Q R pq* (*h1*: -) *pr* (*h2*: -) *qr*.
    *disjE-realizer* · - · - • *pq* · *R* · *pr* · *qr* • *c* • *d* • *e* • *h1* • *h2*

*disjE* (*Q*, *R*): *λpq pr qr*.

(*case pq of None* ⇒ *pr* | *Some q* ⇒ *qr q*)
$\boldsymbol{\lambda}$(*c*: -) (*d*: -) *P Q R pq* (*h1*: -) *pr* (*h2*: -) *qr*.
*disjE-realizer2* · - · - · *pq* · *R* · *pr* · *qr* • *c* • *d* • *h1* • *h2*

*disjE* (*P*, *R*): $\lambda$*pq pr qr*.
(*case pq of None* ⇒ *qr* | *Some p* ⇒ *pr p*)
$\boldsymbol{\lambda}$(*c*: -) (*d*: -) *P Q R pq* (*h1*: -) *pr* (*h2*: -) *qr* (*h3*: -).
*disjE-realizer2* · - · - · *pq* · *R* · *qr* · *pr* • *c* • *d* • *h1* • *h3* • *h2*

*disjE* (*R*): $\lambda$*pq pr qr*.
(*case pq of Left* ⇒ *pr* | *Right* ⇒ *qr*)
$\boldsymbol{\lambda}$(*c*: -) *P Q R pq* (*h1*: -) *pr* (*h2*: -) *qr*.
*disjE-realizer3* · - · - · *pq* · *R* · *pr* · *qr* • *c* • *h1* • *h2*

*disjE* (*P*, *Q*): *Null*
$\boldsymbol{\lambda}$(*c*: -) (*d*: -) *P Q R pq*. *disjE-realizer* · - · - · *pq* · ($\lambda$*x*. *R*) · - · - • *c* • *d* • *arity-type-bool*

*disjE* (*Q*): *Null*
$\boldsymbol{\lambda}$(*c*: -) *P Q R pq*. *disjE-realizer2* · - · - · *pq* · ($\lambda$*x*. *R*) · - · - • *c* • *arity-type-bool*

*disjE* (*P*): *Null*
$\boldsymbol{\lambda}$(*c*: -) *P Q R pq* (*h1*: -) (*h2*: -) (*h3*: -).
*disjE-realizer2* · - · - · *pq* · ($\lambda$*x*. *R*) · - · - • *c* • *arity-type-bool* • *h1* • *h3* • *h2*

*disjE*: *Null*
$\boldsymbol{\lambda}$*P Q R pq*. *disjE-realizer3* · - · - · *pq* · ($\lambda$*x*. *R*) · - · - • *arity-type-bool*

*FalseE* (*P*): *default*
$\boldsymbol{\lambda}$(*c*: -) *P*. *FalseE* · -

*FalseE*: *Null FalseE*

*notI* (*P*): *Null*
$\boldsymbol{\lambda}$(*c*: -) *P* (*h*: -). *allI* · - • *c* • ($\boldsymbol{\lambda}$*x*. *notI* · - • (*h* · *x*))

*notI*: *Null notI*

*notE* (*P*, *R*): $\lambda$*p*. *default*
$\boldsymbol{\lambda}$(*c*: -) (*d*: -) *P R* (*h*: -) *p*. *notE* · - · - • (*spec* · - · *p* • *c* • *h*)

*notE* (*P*): *Null*
$\boldsymbol{\lambda}$(*c*: -) *P R* (*h*: -) *p*. *notE* · - · - • (*spec* · - · *p* • *c* • *h*)

*notE* (*R*): *default*
$\boldsymbol{\lambda}$(*c*: -) *P R*. *notE* · - · -

*notE*: *Null notE*

$\textit{subst}$ $(P)$: $\lambda s\ t\ ps.\ ps$
$\boldsymbol{\lambda}(c\!: \text{-})\ s\ t\ P\ (d\!: \text{-})\ (h\!: \text{-})\ ps.\ \textit{subst} \cdot s \cdot t \cdot P\ ps \bullet d \bullet h$

$\textit{subst}$: $\textit{Null subst}$

$\textit{iffD1}$ $(P, Q)$: $\textit{fst}$
$\boldsymbol{\lambda}(d\!: \text{-})\ (c\!: \text{-})\ Q\ P\ pq\ (h\!: \text{-})\ p.$
$\quad mp \cdot \text{-} \cdot \text{-} \bullet (\textit{spec} \cdot \text{-} \cdot p \bullet d \bullet (\textit{conjunct1} \cdot \text{-} \cdot \text{-} \bullet h))$

$\textit{iffD1}$ $(P)$: $\lambda p.\ p$
$\boldsymbol{\lambda}(c\!: \text{-})\ Q\ P\ p\ (h\!: \text{-}).\ mp \cdot \text{-} \cdot \text{-} \bullet (\textit{conjunct1} \cdot \text{-} \cdot \text{-} \bullet h)$

$\textit{iffD1}$ $(Q)$: $\textit{Null}$
$\boldsymbol{\lambda}(c\!: \text{-})\ Q\ P\ q1\ (h\!: \text{-})\ q2.$
$\quad mp \cdot \text{-} \cdot \text{-} \bullet (\textit{spec} \cdot \text{-} \cdot q2 \bullet c \bullet (\textit{conjunct1} \cdot \text{-} \cdot \text{-} \bullet h))$

$\textit{iffD1}$: $\textit{Null iffD1}$

$\textit{iffD2}$ $(P, Q)$: $\textit{snd}$
$\boldsymbol{\lambda}(c\!: \text{-})\ (d\!: \text{-})\ P\ Q\ pq\ (h\!: \text{-})\ q.$
$\quad mp \cdot \text{-} \cdot \text{-} \bullet (\textit{spec} \cdot \text{-} \cdot q \bullet d \bullet (\textit{conjunct2} \cdot \text{-} \cdot \text{-} \bullet h))$

$\textit{iffD2}$ $(P)$: $\lambda p.\ p$
$\boldsymbol{\lambda}(c\!: \text{-})\ P\ Q\ p\ (h\!: \text{-}).\ mp \cdot \text{-} \cdot \text{-} \bullet (\textit{conjunct2} \cdot \text{-} \cdot \text{-} \bullet h)$

$\textit{iffD2}$ $(Q)$: $\textit{Null}$
$\boldsymbol{\lambda}(c\!: \text{-})\ P\ Q\ q1\ (h\!: \text{-})\ q2.$
$\quad mp \cdot \text{-} \cdot \text{-} \bullet (\textit{spec} \cdot \text{-} \cdot q2 \bullet c \bullet (\textit{conjunct2} \cdot \text{-} \cdot \text{-} \bullet h))$

$\textit{iffD2}$: $\textit{Null iffD2}$

$\textit{iffI}$ $(P, Q)$: $\textit{Pair}$
$\boldsymbol{\lambda}(c\!: \text{-})\ (d\!: \text{-})\ P\ Q\ pq\ (h1 : \text{-})\ qp\ (h2 : \text{-}).\ \textit{conjI-realizer} \cdot$
$\quad (\lambda pq.\ \forall x.\ P\ x \longrightarrow Q\ (pq\ x)) \cdot pq \cdot$
$\quad (\lambda qp.\ \forall x.\ Q\ x \longrightarrow P\ (qp\ x)) \cdot qp \bullet$
$\quad (\textit{arity-type-fun} \bullet c \bullet d) \bullet$
$\quad (\textit{arity-type-fun} \bullet d \bullet c) \bullet$
$\quad (\textit{allI} \cdot \text{-} \bullet c \bullet (\boldsymbol{\lambda} x.\ \textit{impI} \cdot \text{-} \cdot \text{-} \bullet (h1 \cdot x))) \bullet$
$\quad (\textit{allI} \cdot \text{-} \bullet d \bullet (\boldsymbol{\lambda} x.\ \textit{impI} \cdot \text{-} \cdot \text{-} \bullet (h2 \cdot x)))$

$\textit{iffI}$ $(P)$: $\lambda p.\ p$
$\boldsymbol{\lambda}(c\!: \text{-})\ P\ Q\ (h1 : \text{-})\ p\ (h2 : \text{-}).\ \textit{conjI} \cdot \text{-} \cdot \text{-} \bullet$
$\quad (\textit{allI} \cdot \text{-} \bullet c \bullet (\boldsymbol{\lambda} x.\ \textit{impI} \cdot \text{-} \cdot \text{-} \bullet (h1 \cdot x))) \bullet$
$\quad (\textit{impI} \cdot \text{-} \cdot \text{-} \bullet h2)$

$\textit{iffI}$ $(Q)$: $\lambda q.\ q$
$\boldsymbol{\lambda}(c\!: \text{-})\ P\ Q\ q\ (h1 : \text{-})\ (h2 : \text{-}).\ \textit{conjI} \cdot \text{-} \cdot \text{-} \bullet$
$\quad (\textit{impI} \cdot \text{-} \cdot \text{-} \bullet h1) \bullet$
$\quad (\textit{allI} \cdot \text{-} \bullet c \bullet (\boldsymbol{\lambda} x.\ \textit{impI} \cdot \text{-} \cdot \text{-} \bullet (h2 \cdot x)))$

*iffI*: *Null iffI*

**end**

# 86 Extensible records with structural subtyping

**theory** *Record*
**imports** *Quickcheck-Exhaustive*
**keywords**
*record* :: *thy-defn* **and**
*print-record* :: *diag*
**begin**

## 86.1 Introduction

Records are isomorphic to compound tuple types. To implement efficient records, we make this isomorphism explicit. Consider the record access/update simplification *alpha* (*beta-update f rec*) = *alpha rec* for distinct fields alpha and beta of some record rec with n fields. There are $n \hat{~} 2$ such theorems, which prohibits storage of all of them for large n. The rules can be proved on the fly by case decomposition and simplification in O(n) time. By creating O(n) isomorphic-tuple types while defining the record, however, we can prove the access/update simplification in $O(log(n)\hat{~}2)$ time.

The O(n) cost of case decomposition is not because O(n) steps are taken, but rather because the resulting rule must contain O(n) new variables and an O(n) size concrete record construction. To sidestep this cost, we would like to avoid case decomposition in proving access/update theorems.

Record types are defined as isomorphic to tuple types. For instance, a record type with fields $'a$, $'b$, $'c$ and $'d$ might be introduced as isomorphic to $'a \times ('b \times ('c \times 'd))$. If we balance the tuple tree to $('a \times 'b) \times ('c \times 'd)$ then accessors can be defined by converting to the underlying type then using O(log(n)) fst or snd operations. Updators can be defined similarly, if we introduce a *fst-update* and *snd-update* function. Furthermore, we can prove the access/update theorem in O(log(n)) steps by using simple rewrites on fst, snd, *fst-update* and *snd-update*.

The catch is that, although O(log(n)) steps were taken, the underlying type we converted to is a tuple tree of size O(n). Processing this term type wastes performance. We avoid this for large n by taking each subtree of size K and defining a new type isomorphic to that tuple subtree. A record can now be defined as isomorphic to a tuple tree of these O(n/K) new types, or, if $n > K*K$, we can repeat the process, until the record can be defined in terms of a tuple tree of complexity less than the constant K.

If we prove the access/update theorem on this type with the analogous steps to the tuple tree, we consume $O(log(n)^2)$ time as the intermediate terms are $O(log(n))$ in size and the types needed have size bounded by K. To enable this analogous traversal, we define the functions seen below: *iso-tuple-fst*, *iso-tuple-snd*, *iso-tuple-fst-update* and *iso-tuple-snd-update*. These functions generalise tuple operations by taking a parameter that encapsulates a tuple isomorphism. The rewrites needed on these functions now need an additional assumption which is that the isomorphism works.

These rewrites are typically used in a structured way. They are here presented as the introduction rule *isomorphic-tuple.intros* rather than as a rewrite rule set. The introduction form is an optimisation, as net matching can be performed at one term location for each step rather than the simplifier searching the term for possible pattern matches. The rule set is used as it is viewed outside the locale, with the locale assumption (that the isomorphism is valid) left as a rule assumption. All rules are structured to aid net matching, using either a point-free form or an encapsulating predicate.

## 86.2 Operators and lemmas for types isomorphic to tuples

**datatype** (*dead 'a*, *dead 'b*, *dead 'c*) *tuple-isomorphism* =
  *Tuple-Isomorphism 'a* ⇒ *'b* × *'c* *'b* × *'c* ⇒ *'a*

**primrec**
  *repr* :: (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ *'a* ⇒ *'b* × *'c* **where**
  *repr* (*Tuple-Isomorphism r a*) = *r*

**primrec**
  *abst* :: (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ *'b* × *'c* ⇒ *'a* **where**
  *abst* (*Tuple-Isomorphism r a*) = *a*

**definition**
  *iso-tuple-fst* :: (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ *'a* ⇒ *'b* **where**
  *iso-tuple-fst isom* = *fst* ∘ *repr isom*

**definition**
  *iso-tuple-snd* :: (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ *'a* ⇒ *'c* **where**
  *iso-tuple-snd isom* = *snd* ∘ *repr isom*

**definition**
  *iso-tuple-fst-update* ::
    (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ (*'b* ⇒ *'b*) ⇒ (*'a* ⇒ *'a*) **where**
  *iso-tuple-fst-update isom f* = *abst isom* ∘ *apfst f* ∘ *repr isom*

**definition**
  *iso-tuple-snd-update* ::
    (*'a*, *'b*, *'c*) *tuple-isomorphism* ⇒ (*'c* ⇒ *'c*) ⇒ (*'a* ⇒ *'a*) **where**
  *iso-tuple-snd-update isom f* = *abst isom* ∘ *apsnd f* ∘ *repr isom*

**definition**
  *iso-tuple-cons* ::
    $('a, 'b, 'c)$ *tuple-isomorphism* $\Rightarrow 'b \Rightarrow 'c \Rightarrow 'a$ **where**
  *iso-tuple-cons isom* = *curry* (*abst isom*)

## 86.3 Logical infrastructure for records

**definition**
  *iso-tuple-surjective-proof-assist* :: $'a \Rightarrow 'b \Rightarrow ('a \Rightarrow 'b) \Rightarrow$ *bool* **where**
  *iso-tuple-surjective-proof-assist* $x\ y\ f \longleftrightarrow f\ x = y$

**definition**
  *iso-tuple-update-accessor-cong-assist* ::
    $(('b \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'a)) \Rightarrow ('a \Rightarrow 'b) \Rightarrow$ *bool* **where**
  *iso-tuple-update-accessor-cong-assist upd ac* $\longleftrightarrow$
    $(\forall f\ v.\ upd\ (\lambda x.\ f\ (ac\ v))\ v = upd\ f\ v) \wedge (\forall v.\ upd\ id\ v = v)$

**definition**
  *iso-tuple-update-accessor-eq-assist* ::
    $(('b \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'a)) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow ('b \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow$ *bool*
**where**
  *iso-tuple-update-accessor-eq-assist upd ac v f v' x* $\longleftrightarrow$
    $upd\ f\ v = v' \wedge ac\ v = x \wedge$ *iso-tuple-update-accessor-cong-assist upd ac*

**lemma** *update-accessor-congruence-foldE*:
  **assumes** *uac*: *iso-tuple-update-accessor-cong-assist upd ac*
    **and** $r$: $r = r'$ **and** $v$: $ac\ r' = v'$
    **and** $f$: $\bigwedge v.\ v' = v \Longrightarrow f\ v = f'\ v$
  **shows** $upd\ f\ r = upd\ f'\ r'$
  ⟨*proof*⟩

**lemma** *update-accessor-congruence-unfoldE*:
  *iso-tuple-update-accessor-cong-assist upd ac* $\Longrightarrow$
    $r = r' \Longrightarrow ac\ r' = v' \Longrightarrow (\bigwedge v.\ v = v' \Longrightarrow f\ v = f'\ v) \Longrightarrow$
    $upd\ f\ r = upd\ f'\ r'$
  ⟨*proof*⟩

**lemma** *iso-tuple-update-accessor-cong-assist-id*:
  *iso-tuple-update-accessor-cong-assist upd ac* $\Longrightarrow upd\ id = id$
  ⟨*proof*⟩

**lemma** *update-accessor-noopE*:
  **assumes** *uac*: *iso-tuple-update-accessor-cong-assist upd ac*
    **and** *ac*: $f\ (ac\ x) = ac\ x$
  **shows** $upd\ f\ x = x$
  ⟨*proof*⟩

**lemma** *update-accessor-noop-compE*:
  **assumes** *uac*: *iso-tuple-update-accessor-cong-assist upd ac*

**and** *ac*: $f$ $(ac\ x) = ac\ x$
**shows** *upd* $(g \circ f)\ x = upd\ g\ x$
⟨*proof*⟩

**lemma** *update-accessor-cong-assist-idI*:
*iso-tuple-update-accessor-cong-assist id id*
⟨*proof*⟩

**lemma** *update-accessor-cong-assist-triv*:
*iso-tuple-update-accessor-cong-assist upd ac* $\Longrightarrow$
*iso-tuple-update-accessor-cong-assist upd ac*
⟨*proof*⟩

**lemma** *update-accessor-accessor-eqE*:
*iso-tuple-update-accessor-eq-assist upd ac* $v\ f\ v'\ x \Longrightarrow ac\ v = x$
⟨*proof*⟩

**lemma** *update-accessor-updator-eqE*:
*iso-tuple-update-accessor-eq-assist upd ac* $v\ f\ v'\ x \Longrightarrow upd\ f\ v = v'$
⟨*proof*⟩

**lemma** *iso-tuple-update-accessor-eq-assist-idI*:
$v' = f\ v \Longrightarrow$ *iso-tuple-update-accessor-eq-assist id id* $v\ f\ v'\ v$
⟨*proof*⟩

**lemma** *iso-tuple-update-accessor-eq-assist-triv*:
*iso-tuple-update-accessor-eq-assist upd ac* $v\ f\ v'\ x \Longrightarrow$
*iso-tuple-update-accessor-eq-assist upd ac* $v\ f\ v'\ x$
⟨*proof*⟩

**lemma** *iso-tuple-update-accessor-cong-from-eq*:
*iso-tuple-update-accessor-eq-assist upd ac* $v\ f\ v'\ x \Longrightarrow$
*iso-tuple-update-accessor-cong-assist upd ac*
⟨*proof*⟩

**lemma** *iso-tuple-surjective-proof-assistI*:
$f\ x = y \Longrightarrow$ *iso-tuple-surjective-proof-assist* $x\ y\ f$
⟨*proof*⟩

**lemma** *iso-tuple-surjective-proof-assist-idE*:
*iso-tuple-surjective-proof-assist* $x\ y$ *id* $\Longrightarrow x = y$
⟨*proof*⟩

**locale** *isomorphic-tuple* =
**fixes** *isom* :: $('a, 'b, 'c)$ *tuple-isomorphism*
**assumes** *repr-inv*: $\bigwedge x.$ *abst isom* (*repr isom* $x$) $= x$
**and** *abst-inv*: $\bigwedge y.$ *repr isom* (*abst isom* $y$) $= y$
**begin**

**lemma** *repr-inj*: $repr\ isom\ x = repr\ isom\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *abst-inj*: $abst\ isom\ x = abst\ isom\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemmas** *simps* = *Let-def repr-inv abst-inv repr-inj abst-inj*

**lemma** *iso-tuple-access-update-fst-fst*:
$f \circ h\ g = j \circ f \Longrightarrow$
$(f \circ iso\text{-}tuple\text{-}fst\ isom) \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ g =$
$j \circ (f \circ iso\text{-}tuple\text{-}fst\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-access-update-snd-snd*:
$f \circ h\ g = j \circ f \Longrightarrow$
$(f \circ iso\text{-}tuple\text{-}snd\ isom) \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ g =$
$j \circ (f \circ iso\text{-}tuple\text{-}snd\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-access-update-fst-snd*:
$(f \circ iso\text{-}tuple\text{-}fst\ isom) \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ g =$
$id \circ (f \circ iso\text{-}tuple\text{-}fst\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-access-update-snd-fst*:
$(f \circ iso\text{-}tuple\text{-}snd\ isom) \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ g =$
$id \circ (f \circ iso\text{-}tuple\text{-}snd\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-update-swap-fst-fst*:
$h\ f \circ j\ g = j\ g \circ h\ f \Longrightarrow$
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ j)\ g =$
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ j)\ g \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ f$
⟨*proof*⟩

**lemma** *iso-tuple-update-swap-snd-snd*:
$h\ f \circ j\ g = j\ g \circ h\ f \Longrightarrow$
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ j)\ g =$
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ j)\ g \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ f$
⟨*proof*⟩

**lemma** *iso-tuple-update-swap-fst-snd*:
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ j)\ g =$
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ j)\ g \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ f$
⟨*proof*⟩

**lemma** *iso-tuple-update-swap-snd-fst*:
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ j)\ g =$

$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ j)\ g \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ f$
⟨*proof*⟩

**lemma** *iso-tuple-update-compose-fst-fst*:
$h\ f \circ j\ g = k\ (f \circ g) \Longrightarrow$
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ j)\ g =$
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ k)\ (f \circ g)$
⟨*proof*⟩

**lemma** *iso-tuple-update-compose-snd-snd*:
$h\ f \circ j\ g = k\ (f \circ g) \Longrightarrow$
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ h)\ f \circ (iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ j)\ g =$
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ k)\ (f \circ g)$
⟨*proof*⟩

**lemma** *iso-tuple-surjective-proof-assist-step*:
$iso\text{-}tuple\text{-}surjective\text{-}proof\text{-}assist\ v\ a\ (iso\text{-}tuple\text{-}fst\ isom \circ f) \Longrightarrow$
$iso\text{-}tuple\text{-}surjective\text{-}proof\text{-}assist\ v\ b\ (iso\text{-}tuple\text{-}snd\ isom \circ f) \Longrightarrow$
$iso\text{-}tuple\text{-}surjective\text{-}proof\text{-}assist\ v\ (iso\text{-}tuple\text{-}cons\ isom\ a\ b)\ f$
⟨*proof*⟩

**lemma** *iso-tuple-fst-update-accessor-cong-assist*:
**assumes** *iso-tuple-update-accessor-cong-assist f g*
**shows** *iso-tuple-update-accessor-cong-assist*
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ f)\ (g \circ iso\text{-}tuple\text{-}fst\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-snd-update-accessor-cong-assist*:
**assumes** *iso-tuple-update-accessor-cong-assist f g*
**shows** *iso-tuple-update-accessor-cong-assist*
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ f)\ (g \circ iso\text{-}tuple\text{-}snd\ isom)$
⟨*proof*⟩

**lemma** *iso-tuple-fst-update-accessor-eq-assist*:
**assumes** *iso-tuple-update-accessor-eq-assist f g* $a\ u\ a'\ v$
**shows** *iso-tuple-update-accessor-eq-assist*
$(iso\text{-}tuple\text{-}fst\text{-}update\ isom \circ f)\ (g \circ iso\text{-}tuple\text{-}fst\ isom)$
$(iso\text{-}tuple\text{-}cons\ isom\ a\ b)\ u\ (iso\text{-}tuple\text{-}cons\ isom\ a'\ b)\ v$
⟨*proof*⟩

**lemma** *iso-tuple-snd-update-accessor-eq-assist*:
**assumes** *iso-tuple-update-accessor-eq-assist f g* $b\ u\ b'\ v$
**shows** *iso-tuple-update-accessor-eq-assist*
$(iso\text{-}tuple\text{-}snd\text{-}update\ isom \circ f)\ (g \circ iso\text{-}tuple\text{-}snd\ isom)$
$(iso\text{-}tuple\text{-}cons\ isom\ a\ b)\ u\ (iso\text{-}tuple\text{-}cons\ isom\ a\ b')\ v$
⟨*proof*⟩

**lemma** *iso-tuple-cons-conj-eqI*:
$a = c \wedge b = d \wedge P \longleftrightarrow Q \Longrightarrow$

$iso\text{-}tuple\text{-}cons\ isom\ a\ b = iso\text{-}tuple\text{-}cons\ isom\ c\ d \wedge P \longleftrightarrow Q$
⟨*proof*⟩

**lemmas** *intros* =
*iso-tuple-access-update-fst-fst*
*iso-tuple-access-update-snd-snd*
*iso-tuple-access-update-fst-snd*
*iso-tuple-access-update-snd-fst*
*iso-tuple-update-swap-fst-fst*
*iso-tuple-update-swap-snd-snd*
*iso-tuple-update-swap-fst-snd*
*iso-tuple-update-swap-snd-fst*
*iso-tuple-update-compose-fst-fst*
*iso-tuple-update-compose-snd-snd*
*iso-tuple-surjective-proof-assist-step*
*iso-tuple-fst-update-accessor-eq-assist*
*iso-tuple-snd-update-accessor-eq-assist*
*iso-tuple-fst-update-accessor-cong-assist*
*iso-tuple-snd-update-accessor-cong-assist*
*iso-tuple-cons-conj-eqI*

**end**

**lemma** *isomorphic-tuple-intro*:
**fixes** *repr abst*
**assumes** *repr-inj*: $\bigwedge x\ y.\ repr\ x = repr\ y \longleftrightarrow x = y$
**and** *abst-inv*: $\bigwedge z.\ repr\ (abst\ z) = z$
**and** *v*: $v \equiv Tuple\text{-}Isomorphism\ repr\ abst$
**shows** *isomorphic-tuple v*
⟨*proof*⟩

**definition**
$tuple\text{-}iso\text{-}tuple \equiv Tuple\text{-}Isomorphism\ id\ id$

**lemma** *tuple-iso-tuple*:
*isomorphic-tuple tuple-iso-tuple*
⟨*proof*⟩

**lemma** *refl-conj-eq*: $Q = R \Longrightarrow P \wedge Q \longleftrightarrow P \wedge R$
⟨*proof*⟩

**lemma** *iso-tuple-UNIV-I*: $x \in UNIV \equiv True$
⟨*proof*⟩

**lemma** *iso-tuple-True-simp*: $(True \Longrightarrow PROP\ P) \equiv PROP\ P$
⟨*proof*⟩

**lemma** *prop-subst*: $s = t \Longrightarrow PROP\ P\ t \Longrightarrow PROP\ P\ s$
⟨*proof*⟩

**lemma** *K-record-comp*: $(\lambda x.\ c) \circ f = (\lambda x.\ c)$
⟨*proof*⟩

## 86.4 Concrete record syntax

**nonterminal**
*ident* **and**
*field-type* **and**
*field-types* **and**
*field* **and**
*fields* **and**
*field-update* **and**
*field-updates*

**open-bundle** *record-syntax*
**begin**

**syntax**
*-constify* :: *id => ident* (‹-›)
*-constify* :: *longid => ident* (‹-›)

*-field-type* :: *ident => type => field-type* (‹(‹indent=2 notation=‹infix field type››- ::/ -)›)
:: *field-type => field-types* (‹-›)
*-field-types* :: *field-type => field-types => field-types* (‹-,/ -›)
*-record-type* :: *field-types => type* (‹(‹indent=3 notation=‹mixfix record type››⦇-⦈)›)
*-record-type-scheme* :: *field-types => type => type* (‹(‹indent=3 notation=‹mixfix record type››⦇-,/ (‹indent=2 notation=‹infix more type››. . . ::/ -)⦈)›)

*-field* :: *ident => 'a => field* (‹(‹indent=2 notation=‹infix field value››(‹open-block markup=‹const››-) =/ -)›)
:: *field => fields* (‹-›)
*-fields* :: *field => fields => fields* (‹-,/ -›)
*-record* :: *fields => 'a* (‹(‹indent=3 notation=‹mixfix record value››⦇-⦈)›)
*-record-scheme* :: *fields => 'a => 'a* (‹(‹indent=3 notation=‹mixfix record value››⦇-,/ (‹indent=2 notation=‹infix more value››. . . =/ -)⦈)›)

*-field-update* :: *ident => 'a => field-update* (‹(‹indent=2 notation=‹infix field update››(‹open-block markup=‹const››-) :=/ -)›)
:: *field-update => field-updates* (‹-›)
*-field-updates* :: *field-update => field-updates => field-updates* (‹-,/ -›)
*-record-update* :: *'a => field-updates => 'b* (‹(‹open-block notation=‹mixfix record update››-/(3⦇-⦈))› [900, 0] 900)

**syntax** (*ASCII*)
*-record-type* :: *field-types => type* (‹(‹indent=3 notation=‹mixfix

*record type*›› ′(| - |′))›)
*-record-type-scheme* :: *field-types* => *type* => *type* (‹(‹*indent=3 notation=*‹*mixfix record type*›› ′(| -,/ (‹*indent=2 notation=*‹*infix more type*››... ::/ -) |′))›)
*-record* :: *fields* => ′*a* (‹(‹*indent=3 notation=*‹*mixfix record value*›› ′(| - |′))›)
*-record-scheme* :: *fields* => ′*a* => ′*a* (‹(‹*indent=3 notation=*‹*mixfix record value*›› ′(| -,/ (‹*indent=2 notation=*‹*infix more value*››... =/ -) |′))›)
*-record-update* :: ′*a* => *field-updates* => ′*b* (‹(‹*open-block notation=*‹*mixfix record update*››-/(*3* ′(| - |′)))› [*900*, *0*] *900*)

**end**

### 86.5 Record package

⟨*ML*⟩

**hide-const** (**open**) *Tuple-Isomorphism repr abst iso-tuple-fst iso-tuple-snd iso-tuple-fst-update iso-tuple-snd-update iso-tuple-cons iso-tuple-surjective-proof-assist iso-tuple-update-accessor-cong-assist iso-tuple-update-accessor-eq-assist tuple-iso-tuple*

**end**

## 87 Greatest common divisor and least common multiple

**theory** *GCD*
  **imports** *Groups-List Code-Numeral*
**begin**

### 87.1 Abstract bounded quasi semilattices as common foundation

**locale** *bounded-quasi-semilattice* = *abel-semigroup* +
  **fixes** *top* :: ′*a* (‹⊤›) **and** *bot* :: ′*a* (‹⊥›)
    **and** *normalize* :: ′*a* ⇒ ′*a*
  **assumes** *idem-normalize* [*simp*]: $a * a = normalize\ a$
    **and** *normalize-left-idem* [*simp*]: $normalize\ a * b = a * b$
    **and** *normalize-idem* [*simp*]: $normalize\ (a * b) = a * b$
    **and** *normalize-top* [*simp*]: $normalize\ \top = \top$
    **and** *normalize-bottom* [*simp*]: $normalize\ \bot = \bot$
    **and** *top-left-normalize* [*simp*]: $\top * a = normalize\ a$
    **and** *bottom-left-bottom* [*simp*]: $\bot * a = \bot$
**begin**

**lemma** *left-idem* [*simp*]:
  $a * (a * b) = a * b$

⟨*proof*⟩

**lemma** *right-idem* [*simp*]:
$(a * b) * b = a * b$
⟨*proof*⟩

**lemma** *comp-fun-idem*: *comp-fun-idem f*
⟨*proof*⟩

**interpretation** *comp-fun-idem f*
⟨*proof*⟩

**lemma** *top-right-normalize* [*simp*]:
$a * \top = normalize\ a$
⟨*proof*⟩

**lemma** *bottom-right-bottom* [*simp*]:
$a * \bot = \bot$
⟨*proof*⟩

**lemma** *normalize-right-idem* [*simp*]:
$a * normalize\ b = a * b$
⟨*proof*⟩

**end**

**locale** *bounded-quasi-semilattice-set* = *bounded-quasi-semilattice*
**begin**

**interpretation** *comp-fun-idem f*
⟨*proof*⟩

**definition** $F :: 'a\ set \Rightarrow 'a$
**where**
*eq-fold*: $F\ A = (if\ finite\ A\ then\ Finite\text{-}Set.fold\ f\ \top\ A\ else\ \bot)$

**lemma** *infinite* [*simp*]:
$infinite\ A \Longrightarrow F\ A = \bot$
⟨*proof*⟩

**lemma** *set-eq-fold* [*code*]:
$F\ (set\ xs) = fold\ f\ xs\ \top$
⟨*proof*⟩

**lemma** *empty* [*simp*]:
$F\ \{\} = \top$
⟨*proof*⟩

**lemma** *insert* [*simp*]:

$F$ (*insert* $a$ $A$) $= a * F\ A$
⟨*proof*⟩

**lemma** *normalize* [*simp*]:
*normalize* $(F\ A) = F\ A$
⟨*proof*⟩

**lemma** *in-idem*:
**assumes** $a \in A$
**shows** $a * F\ A = F\ A$
⟨*proof*⟩

**lemma** *union*:
$F\ (A \cup B) = F\ A * F\ B$
⟨*proof*⟩

**lemma** *remove*:
**assumes** $a \in A$
**shows** $F\ A = a * F\ (A - \{a\})$
⟨*proof*⟩

**lemma** *insert-remove*:
$F$ (*insert* $a$ $A$) $= a * F\ (A - \{a\})$
⟨*proof*⟩

**lemma** *subset*:
**assumes** $B \subseteq A$
**shows** $F\ B * F\ A = F\ A$
⟨*proof*⟩

**end**

## 87.2 Abstract GCD and LCM

**class** *gcd* = *zero* + *one* + *dvd* +
**fixes** *gcd* :: $'a \Rightarrow 'a \Rightarrow 'a$
**and** *lcm* :: $'a \Rightarrow 'a \Rightarrow 'a$

**class** *Gcd* = *gcd* +
**fixes** *Gcd* :: $'a$ *set* $\Rightarrow 'a$
**and** *Lcm* :: $'a$ *set* $\Rightarrow 'a$

**syntax**
*-GCD1* :: *pttrns* $\Rightarrow 'b \Rightarrow 'b$ (‹(‹*indent=3 notation=*‹*binder GCD*››*GCD* -./ -)› [*0*, *10*] *10*)
*-GCD* :: *pttrn* $\Rightarrow 'a$ *set* $\Rightarrow 'b \Rightarrow 'b$ (‹(‹*indent=3 notation=*‹*binder GCD*››*GCD* -∈-./ -)› [*0*, *0*, *10*] *10*)
*-LCM1* :: *pttrns* $\Rightarrow 'b \Rightarrow 'b$ (‹(‹*indent=3 notation=*‹*binder LCM*››*LCM* -./ -)› [*0*, *10*] *10*)

*-LCM* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder LCM››LCM -∈-./ -)› [0, 0, 10] 10)

**syntax-consts**
*-GCD1 -GCD* ⇌ *Gcd* **and**
*-LCM1 -LCM* ⇌ *Lcm*

**translations**
*GCD x y. f* ⇌ *GCD x. GCD y. f*
*GCD x. f* ⇌ *CONST Gcd* (*CONST range* (λ*x. f*))
*GCD x*∈*A. f* ⇌ *CONST Gcd* ((λ*x. f*) ' *A*)
*LCM x y. f* ⇌ *LCM x. LCM y. f*
*LCM x. f* ⇌ *CONST Lcm* (*CONST range* (λ*x. f*))
*LCM x*∈*A. f* ⇌ *CONST Lcm* ((λ*x. f*) ' *A*)

**class** *semiring-gcd* = *normalization-semidom* + *gcd* +
**assumes** *gcd-dvd1* [*iff*]: *gcd a b dvd a*
**and** *gcd-dvd2* [*iff*]: *gcd a b dvd b*
**and** *gcd-greatest*: *c dvd a* ⟹ *c dvd b* ⟹ *c dvd gcd a b*
**and** *normalize-gcd* [*simp*]: *normalize* (*gcd a b*) = *gcd a b*
**and** *lcm-gcd*: *lcm a b* = *normalize* (*a* ∗ *b div gcd a b*)
**begin**

**lemma** *gcd-greatest-iff* [*simp*]: *a dvd gcd b c* ⟷ *a dvd b* ∧ *a dvd c*
⟨*proof*⟩

**lemma** *gcd-dvdI1*: *a dvd c* ⟹ *gcd a b dvd c*
⟨*proof*⟩

**lemma** *gcd-dvdI2*: *b dvd c* ⟹ *gcd a b dvd c*
⟨*proof*⟩

**lemma** *dvd-gcdD1*: *a dvd gcd b c* ⟹ *a dvd b*
⟨*proof*⟩

**lemma** *dvd-gcdD2*: *a dvd gcd b c* ⟹ *a dvd c*
⟨*proof*⟩

**lemma** *gcd-0-left* [*simp*]: *gcd 0 a* = *normalize a*
⟨*proof*⟩

**lemma** *gcd-0-right* [*simp*]: *gcd a 0* = *normalize a*
⟨*proof*⟩

**lemma** *gcd-eq-0-iff* [*simp*]: *gcd a b* = *0* ⟷ *a* = *0* ∧ *b* = *0*
(**is** *?P* ⟷ *?Q*)
⟨*proof*⟩

**lemma** *unit-factor-gcd*: *unit-factor* (*gcd a b*) = (*if a* = *0* ∧ *b* = *0 then 0 else 1*)

⟨*proof*⟩

**lemma** *is-unit-gcd-iff* [*simp*]:
*is-unit* (*gcd a b*) ⟷ *gcd a b* = *1*
⟨*proof*⟩

**sublocale** *gcd*: *abel-semigroup gcd*
⟨*proof*⟩

**sublocale** *gcd*: *bounded-quasi-semilattice gcd 0 1 normalize*
⟨*proof*⟩

**lemma** *gcd-self*: *gcd a a* = *normalize a*
⟨*proof*⟩

**lemma** *gcd-left-idem*: *gcd a* (*gcd a b*) = *gcd a b*
⟨*proof*⟩

**lemma** *gcd-right-idem*: *gcd* (*gcd a b*) *b* = *gcd a b*
⟨*proof*⟩

**lemma** *gcdI*:
**assumes** *c dvd a* **and** *c dvd b*
**and** *greatest*: $\bigwedge d.$ *d dvd a* ⟹ *d dvd b* ⟹ *d dvd c*
**and** *normalize c* = *c*
**shows** *c* = *gcd a b*
⟨*proof*⟩

**lemma** *gcd-unique*:
*d dvd a* ∧ *d dvd b* ∧ *normalize d* = *d* ∧ (∀ *e*. *e dvd a* ∧ *e dvd b* ⟶ *e dvd d*) ⟷
*d* = *gcd a b*
⟨*proof*⟩

**lemma** *gcd-dvd-prod*: *gcd a b dvd k* ∗ *b*
⟨*proof*⟩

**lemma** *gcd-proj2-if-dvd*: *b dvd a* ⟹ *gcd a b* = *normalize b*
⟨*proof*⟩

**lemma** *gcd-proj1-if-dvd*: *a dvd b* ⟹ *gcd a b* = *normalize a*
⟨*proof*⟩

**lemma** *gcd-proj1-iff*: *gcd m n* = *normalize m* ⟷ *m dvd n*
⟨*proof*⟩

**lemma** *gcd-proj2-iff*: *gcd m n* = *normalize n* ⟷ *n dvd m*
⟨*proof*⟩

**lemma** *gcd-mult-left*: *gcd* (*c* ∗ *a*) (*c* ∗ *b*) = *normalize* (*c* ∗ *gcd a b*)

⟨*proof*⟩

**lemma** *gcd-mult-right*: $gcd\ (a * c)\ (b * c) = normalize\ (gcd\ b\ a * c)$
⟨*proof*⟩

**lemma** *dvd-lcm1* [*iff*]: $a\ dvd\ lcm\ a\ b$
⟨*proof*⟩

**lemma** *dvd-lcm2* [*iff*]: $b\ dvd\ lcm\ a\ b$
⟨*proof*⟩

**lemma** *dvd-lcmI1*: $a\ dvd\ b \Longrightarrow a\ dvd\ lcm\ b\ c$
⟨*proof*⟩

**lemma** *dvd-lcmI2*: $a\ dvd\ c \Longrightarrow a\ dvd\ lcm\ b\ c$
⟨*proof*⟩

**lemma** *lcm-dvdD1*: $lcm\ a\ b\ dvd\ c \Longrightarrow a\ dvd\ c$
⟨*proof*⟩

**lemma** *lcm-dvdD2*: $lcm\ a\ b\ dvd\ c \Longrightarrow b\ dvd\ c$
⟨*proof*⟩

**lemma** *lcm-least*:
**assumes** $a\ dvd\ c$ **and** $b\ dvd\ c$
**shows** $lcm\ a\ b\ dvd\ c$
⟨*proof*⟩

**lemma** *lcm-least-iff* [*simp*]: $lcm\ a\ b\ dvd\ c \longleftrightarrow a\ dvd\ c \wedge b\ dvd\ c$
⟨*proof*⟩

**lemma** *normalize-lcm* [*simp*]: $normalize\ (lcm\ a\ b) = lcm\ a\ b$
⟨*proof*⟩

**lemma** *lcm-0-left* [*simp*]: $lcm\ 0\ a = 0$
⟨*proof*⟩

**lemma** *lcm-0-right* [*simp*]: $lcm\ a\ 0 = 0$
⟨*proof*⟩

**lemma** *lcm-eq-0-iff*: $lcm\ a\ b = 0 \longleftrightarrow a = 0 \vee b = 0$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *zero-eq-lcm-iff*: $0 = lcm\ a\ b \longleftrightarrow a = 0 \vee b = 0$
⟨*proof*⟩

**lemma** *lcm-eq-1-iff* [*simp*]: $lcm\ a\ b = 1 \longleftrightarrow is\text{-}unit\ a \wedge is\text{-}unit\ b$
⟨*proof*⟩

**lemma** *unit-factor-lcm*: *unit-factor* (*lcm a b*) = (*if a* = *0* ∨ *b* = *0 then 0 else 1*)
⟨*proof*⟩

**sublocale** *lcm*: *abel-semigroup lcm*
⟨*proof*⟩

**sublocale** *lcm*: *bounded-quasi-semilattice lcm 1 0 normalize*
⟨*proof*⟩

**lemma** *lcm-self*: *lcm a a* = *normalize a*
⟨*proof*⟩

**lemma** *lcm-left-idem*: *lcm a* (*lcm a b*) = *lcm a b*
⟨*proof*⟩

**lemma** *lcm-right-idem*: *lcm* (*lcm a b*) *b* = *lcm a b*
⟨*proof*⟩

**lemma** *gcd-lcm*:
**assumes** *a* ≠ *0* **and** *b* ≠ *0*
**shows** *gcd a b* = *normalize* (*a* ∗ *b div lcm a b*)
⟨*proof*⟩

**lemma** *lcm-1-left*: *lcm 1 a* = *normalize a*
⟨*proof*⟩

**lemma** *lcm-1-right*: *lcm a 1* = *normalize a*
⟨*proof*⟩

**lemma** *lcm-mult-left*: *lcm* (*c* ∗ *a*) (*c* ∗ *b*) = *normalize* (*c* ∗ *lcm a b*)
⟨*proof*⟩

**lemma** *lcm-mult-right*: *lcm* (*a* ∗ *c*) (*b* ∗ *c*) = *normalize* (*lcm b a* ∗ *c*)
⟨*proof*⟩

**lemma** *lcm-mult-unit1*: *is-unit a* ⟹ *lcm* (*b* ∗ *a*) *c* = *lcm b c*
⟨*proof*⟩

**lemma** *lcm-mult-unit2*: *is-unit a* ⟹ *lcm b* (*c* ∗ *a*) = *lcm b c*
⟨*proof*⟩

**lemma** *lcm-div-unit1*:
*is-unit a* ⟹ *lcm* (*b div a*) *c* = *lcm b c*
⟨*proof*⟩

**lemma** *lcm-div-unit2*: *is-unit a* ⟹ *lcm b* (*c div a*) = *lcm b c*
⟨*proof*⟩

**lemma** *normalize-lcm-left*: $lcm\ (normalize\ a)\ b = lcm\ a\ b$
⟨*proof*⟩

**lemma** *normalize-lcm-right*: $lcm\ a\ (normalize\ b) = lcm\ a\ b$
⟨*proof*⟩

**lemma** *comp-fun-idem-gcd*: *comp-fun-idem gcd*
⟨*proof*⟩

**lemma** *comp-fun-idem-lcm*: *comp-fun-idem lcm*
⟨*proof*⟩

**lemma** *gcd-dvd-antisym*: $gcd\ a\ b\ dvd\ gcd\ c\ d \Longrightarrow gcd\ c\ d\ dvd\ gcd\ a\ b \Longrightarrow gcd\ a\ b = gcd\ c\ d$
⟨*proof*⟩

**declare** *unit-factor-lcm* [*simp*]

**lemma** *lcmI*:
**assumes** $a\ dvd\ c$ **and** $b\ dvd\ c$ **and** $\bigwedge d.\ a\ dvd\ d \Longrightarrow b\ dvd\ d \Longrightarrow c\ dvd\ d$
**and** $normalize\ c = c$
**shows** $c = lcm\ a\ b$
⟨*proof*⟩

**lemma** *gcd-dvd-lcm* [*simp*]: $gcd\ a\ b\ dvd\ lcm\ a\ b$
⟨*proof*⟩

**lemmas** *lcm-0* = *lcm-0-right*

**lemma** *lcm-unique*:
$a\ dvd\ d \wedge b\ dvd\ d \wedge normalize\ d = d \wedge (\forall e.\ a\ dvd\ e \wedge b\ dvd\ e \longrightarrow d\ dvd\ e) \longleftrightarrow d = lcm\ a\ b$
⟨*proof*⟩

**lemma** *lcm-proj1-if-dvd*:
**assumes** $b\ dvd\ a$ **shows** $lcm\ a\ b = normalize\ a$
⟨*proof*⟩

**lemma** *lcm-proj2-if-dvd*: $a\ dvd\ b \Longrightarrow lcm\ a\ b = normalize\ b$
⟨*proof*⟩

**lemma** *lcm-proj1-iff*: $lcm\ m\ n = normalize\ m \longleftrightarrow n\ dvd\ m$
⟨*proof*⟩

**lemma** *lcm-proj2-iff*: $lcm\ m\ n = normalize\ n \longleftrightarrow m\ dvd\ n$
⟨*proof*⟩

**lemma** *gcd-mono*: $a\ dvd\ c \Longrightarrow b\ dvd\ d \Longrightarrow gcd\ a\ b\ dvd\ gcd\ c\ d$
⟨*proof*⟩

**lemma** *lcm-mono*: $a\ dvd\ c \Longrightarrow b\ dvd\ d \Longrightarrow lcm\ a\ b\ dvd\ lcm\ c\ d$
⟨*proof*⟩

**lemma** *dvd-productE*:
**assumes** $p\ dvd\ a * b$
**obtains** $x\ y$ **where** $p = x * y\ x\ dvd\ a\ y\ dvd\ b$
⟨*proof*⟩

**lemma** *gcd-mult-unit1*:
**assumes** *is-unit a* **shows** $gcd\ (b * a)\ c = gcd\ b\ c$
⟨*proof*⟩

**lemma** *gcd-mult-unit2*: $is\text{-}unit\ a \Longrightarrow gcd\ b\ (c * a) = gcd\ b\ c$
⟨*proof*⟩

**lemma** *gcd-div-unit1*: $is\text{-}unit\ a \Longrightarrow gcd\ (b\ div\ a)\ c = gcd\ b\ c$
⟨*proof*⟩

**lemma** *gcd-div-unit2*: $is\text{-}unit\ a \Longrightarrow gcd\ b\ (c\ div\ a) = gcd\ b\ c$
⟨*proof*⟩

**lemma** *normalize-gcd-left*: $gcd\ (normalize\ a)\ b = gcd\ a\ b$
⟨*proof*⟩

**lemma** *normalize-gcd-right*: $gcd\ a\ (normalize\ b) = gcd\ a\ b$
⟨*proof*⟩

**lemma** *gcd-add1* [*simp*]: $gcd\ (m + n)\ n = gcd\ m\ n$
⟨*proof*⟩

**lemma** *gcd-add2* [*simp*]: $gcd\ m\ (m + n) = gcd\ m\ n$
⟨*proof*⟩

**lemma** *gcd-add-mult*: $gcd\ m\ (k * m + n) = gcd\ m\ n$
⟨*proof*⟩

**end**

**class** *ring-gcd* = *comm-ring-1* + *semiring-gcd*
**begin**

**lemma** *gcd-neg1* [*simp*]: $gcd\ (-a)\ b = gcd\ a\ b$
⟨*proof*⟩

**lemma** *gcd-neg2* [*simp*]: $gcd\ a\ (-b) = gcd\ a\ b$
⟨*proof*⟩

**lemma** *gcd-neg-numeral-1* [*simp*]: $gcd\ (-\ numeral\ n)\ a = gcd\ (numeral\ n)\ a$

⟨proof⟩

**lemma** *gcd-neg-numeral-2* [*simp*]: $gcd\ a\ (-\ numeral\ n) = gcd\ a\ (numeral\ n)$
⟨proof⟩

**lemma** *gcd-diff1*: $gcd\ (m - n)\ n = gcd\ m\ n$
⟨proof⟩

**lemma** *gcd-diff2*: $gcd\ (n - m)\ n = gcd\ m\ n$
⟨proof⟩

**lemma** *lcm-neg1* [*simp*]: $lcm\ (-a)\ b = lcm\ a\ b$
⟨proof⟩

**lemma** *lcm-neg2* [*simp*]: $lcm\ a\ (-b) = lcm\ a\ b$
⟨proof⟩

**lemma** *lcm-neg-numeral-1* [*simp*]: $lcm\ (-\ numeral\ n)\ a = lcm\ (numeral\ n)\ a$
⟨proof⟩

**lemma** *lcm-neg-numeral-2* [*simp*]: $lcm\ a\ (-\ numeral\ n) = lcm\ a\ (numeral\ n)$
⟨proof⟩

**end**

**class** *semiring-Gcd* = *semiring-gcd* + *Gcd* +
**assumes** *Gcd-dvd*: $a \in A \Longrightarrow Gcd\ A\ dvd\ a$
**and** *Gcd-greatest*: $(\bigwedge b.\ b \in A \Longrightarrow a\ dvd\ b) \Longrightarrow a\ dvd\ Gcd\ A$
**and** *normalize-Gcd* [*simp*]: $normalize\ (Gcd\ A) = Gcd\ A$
**assumes** *dvd-Lcm*: $a \in A \Longrightarrow a\ dvd\ Lcm\ A$
**and** *Lcm-least*: $(\bigwedge b.\ b \in A \Longrightarrow b\ dvd\ a) \Longrightarrow Lcm\ A\ dvd\ a$
**and** *normalize-Lcm* [*simp*]: $normalize\ (Lcm\ A) = Lcm\ A$
**begin**

**lemma** *Lcm-Gcd*: $Lcm\ A = Gcd\ \{b.\ \forall a \in A.\ a\ dvd\ b\}$
⟨proof⟩

**lemma** *Gcd-Lcm*: $Gcd\ A = Lcm\ \{b.\ \forall a \in A.\ b\ dvd\ a\}$
⟨proof⟩

**lemma** *Gcd-empty* [*simp*]: $Gcd\ \{\} = 0$
⟨proof⟩

**lemma** *Lcm-empty* [*simp*]: $Lcm\ \{\} = 1$
⟨proof⟩

**lemma** *Gcd-insert* [*simp*]: $Gcd\ (insert\ a\ A) = gcd\ a\ (Gcd\ A)$
⟨proof⟩

**lemma** *Lcm-insert* [*simp*]: $Lcm\ (insert\ a\ A) = lcm\ a\ (Lcm\ A)$
⟨*proof*⟩

**lemma** *LcmI*:
**assumes** $\bigwedge a.\ a \in A \Longrightarrow a\ dvd\ b$
**and** $\bigwedge c.\ (\bigwedge a.\ a \in A \Longrightarrow a\ dvd\ c) \Longrightarrow b\ dvd\ c$
**and** $normalize\ b = b$
**shows** $b = Lcm\ A$
⟨*proof*⟩

**lemma** *Lcm-subset*: $A \subseteq B \Longrightarrow Lcm\ A\ dvd\ Lcm\ B$
⟨*proof*⟩

**lemma** *Lcm-Un*: $Lcm\ (A \cup B) = lcm\ (Lcm\ A)\ (Lcm\ B)$
⟨*proof*⟩

**lemma** *Gcd-0-iff* [*simp*]: $Gcd\ A = 0 \longleftrightarrow A \subseteq \{0\}$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *Lcm-1-iff* [*simp*]: $Lcm\ A = 1 \longleftrightarrow (\forall\, a\in A.\ is\text{-}unit\ a)$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

**lemma** *unit-factor-Lcm*: $unit\text{-}factor\ (Lcm\ A) = (if\ Lcm\ A = 0\ then\ 0\ else\ 1)$
⟨*proof*⟩

**lemma** *unit-factor-Gcd*: $unit\text{-}factor\ (Gcd\ A) = (if\ Gcd\ A = 0\ then\ 0\ else\ 1)$
⟨*proof*⟩

**lemma** *GcdI*:
**assumes** $\bigwedge a.\ a \in A \Longrightarrow b\ dvd\ a$
**and** $\bigwedge c.\ (\bigwedge a.\ a \in A \Longrightarrow c\ dvd\ a) \Longrightarrow c\ dvd\ b$
**and** $normalize\ b = b$
**shows** $b = Gcd\ A$
⟨*proof*⟩

**lemma** *Gcd-eq-1-I*:
**assumes** $is\text{-}unit\ a$ **and** $a \in A$
**shows** $Gcd\ A = 1$
⟨*proof*⟩

**lemma** *Lcm-eq-0-I*:
**assumes** $0 \in A$
**shows** $Lcm\ A = 0$
⟨*proof*⟩

**lemma** *Gcd-UNIV* [*simp*]: $Gcd\ UNIV = 1$
⟨*proof*⟩

**lemma** *Lcm-UNIV* [*simp*]: *Lcm UNIV* = *0*
⟨*proof*⟩

**lemma** *Lcm-0-iff*:
**assumes** *finite A*
**shows** *Lcm A* = *0* $\longleftrightarrow$ *0* ∈ *A*
⟨*proof*⟩

**lemma** *Gcd-image-normalize* [*simp*]: *Gcd* (*normalize* ' *A*) = *Gcd A*
⟨*proof*⟩

**lemma** *Gcd-eqI*:
**assumes** *normalize a* = *a*
**assumes** $\bigwedge b$. *b* ∈ *A* $\Longrightarrow$ *a dvd b*
**and** $\bigwedge c$. ($\bigwedge b$. *b* ∈ *A* $\Longrightarrow$ *c dvd b*) $\Longrightarrow$ *c dvd a*
**shows** *Gcd A* = *a*
⟨*proof*⟩

**lemma** *dvd-GcdD*: *x dvd Gcd A* $\Longrightarrow$ *y* ∈ *A* $\Longrightarrow$ *x dvd y*
⟨*proof*⟩

**lemma** *dvd-Gcd-iff*: *x dvd Gcd A* $\longleftrightarrow$ (∀ *y*∈*A*. *x dvd y*)
⟨*proof*⟩

**lemma** *Gcd-mult*: *Gcd* ((∗) *c* ' *A*) = *normalize* (*c* ∗ *Gcd A*)
⟨*proof*⟩

**lemma** *Lcm-eqI*:
**assumes** *normalize a* = *a*
**and** $\bigwedge b$. *b* ∈ *A* $\Longrightarrow$ *b dvd a*
**and** $\bigwedge c$. ($\bigwedge b$. *b* ∈ *A* $\Longrightarrow$ *b dvd c*) $\Longrightarrow$ *a dvd c*
**shows** *Lcm A* = *a*
⟨*proof*⟩

**lemma** *Lcm-dvdD*: *Lcm A dvd x* $\Longrightarrow$ *y* ∈ *A* $\Longrightarrow$ *y dvd x*
⟨*proof*⟩

**lemma** *Lcm-dvd-iff*: *Lcm A dvd x* $\longleftrightarrow$ (∀ *y*∈*A*. *y dvd x*)
⟨*proof*⟩

**lemma** *Lcm-mult*:
**assumes** *A* ≠ {}
**shows** *Lcm* ((∗) *c* ' *A*) = *normalize* (*c* ∗ *Lcm A*)
⟨*proof*⟩

**lemma** *Lcm-no-units*: *Lcm A* = *Lcm* (*A* − {*a*. *is-unit a*})
⟨*proof*⟩

**lemma** *Lcm-0-iff'*: $Lcm\ A = 0 \longleftrightarrow (\nexists l.\ l \neq 0 \wedge (\forall a \in A.\ a\ dvd\ l))$
⟨*proof*⟩

**lemma** *Lcm-no-multiple*: $(\forall m.\ m \neq 0 \longrightarrow (\exists a \in A.\ \neg\ a\ dvd\ m)) \Longrightarrow Lcm\ A = 0$
⟨*proof*⟩

**lemma** *Lcm-singleton* [*simp*]: $Lcm\ \{a\} = normalize\ a$
⟨*proof*⟩

**lemma** *Lcm-2* [*simp*]: $Lcm\ \{a,\ b\} = lcm\ a\ b$
⟨*proof*⟩

**lemma** *Gcd-1*: $1 \in A \Longrightarrow Gcd\ A = 1$
⟨*proof*⟩

**lemma** *Gcd-singleton* [*simp*]: $Gcd\ \{a\} = normalize\ a$
⟨*proof*⟩

**lemma** *Gcd-2* [*simp*]: $Gcd\ \{a,\ b\} = gcd\ a\ b$
⟨*proof*⟩

**lemma** *Gcd-mono*:
**assumes** $\bigwedge x.\ x \in A \Longrightarrow f\ x\ dvd\ g\ x$
**shows** $(GCD\ x{\in}A.\ f\ x)\ dvd\ (GCD\ x{\in}A.\ g\ x)$
⟨*proof*⟩

**lemma** *Lcm-mono*:
**assumes** $\bigwedge x.\ x \in A \Longrightarrow f\ x\ dvd\ g\ x$
**shows** $(LCM\ x{\in}A.\ f\ x)\ dvd\ (LCM\ x{\in}A.\ g\ x)$
⟨*proof*⟩

**end**

## 87.3 An aside: GCD and LCM on finite sets for incomplete gcd rings

**context** *semiring-gcd*
**begin**

**sublocale** *Gcd-fin*: *bounded-quasi-semilattice-set gcd 0 1 normalize*
**defines**
*Gcd-fin* (‹$Gcd_{fin}$›) = *Gcd-fin.F* :: *'a set* ⇒ *'a* ⟨*proof*⟩

**abbreviation** *gcd-list* :: *'a list* ⇒ *'a*
**where** *gcd-list xs* ≡ $Gcd_{fin}$ (*set xs*)

**sublocale** *Lcm-fin*: *bounded-quasi-semilattice-set lcm 1 0 normalize*
**defines**
*Lcm-fin* (‹$Lcm_{fin}$›) = *Lcm-fin.F* ⟨*proof*⟩

**abbreviation** *lcm-list* :: $'a$ *list* $\Rightarrow$ $'a$
  **where** *lcm-list xs* $\equiv$ $Lcm_{fin}$ (*set xs*)

**lemma** *Gcd-fin-dvd*:
  $a \in A \Longrightarrow Gcd_{fin}\ A\ dvd\ a$
  ⟨*proof*⟩

**lemma** *dvd-Lcm-fin*:
  $a \in A \Longrightarrow a\ dvd\ Lcm_{fin}\ A$
  ⟨*proof*⟩

**lemma** *Gcd-fin-greatest*:
  $a\ dvd\ Gcd_{fin}\ A$ **if** *finite A* **and** $\bigwedge b.\ b \in A \Longrightarrow a\ dvd\ b$
  ⟨*proof*⟩

**lemma** *Lcm-fin-least*:
  $Lcm_{fin}\ A\ dvd\ a$ **if** *finite A* **and** $\bigwedge b.\ b \in A \Longrightarrow b\ dvd\ a$
  ⟨*proof*⟩

**lemma** *gcd-list-greatest*:
  *a dvd gcd-list bs* **if** $\bigwedge b.\ b \in set\ bs \Longrightarrow a\ dvd\ b$
  ⟨*proof*⟩

**lemma** *lcm-list-least*:
  *lcm-list bs dvd a* **if** $\bigwedge b.\ b \in set\ bs \Longrightarrow b\ dvd\ a$
  ⟨*proof*⟩

**lemma** *dvd-Gcd-fin-iff*:
  $b\ dvd\ Gcd_{fin}\ A \longleftrightarrow (\forall\, a{\in}A.\ b\ dvd\ a)$ **if** *finite A*
  ⟨*proof*⟩

**lemma** *dvd-gcd-list-iff*:
  *b dvd gcd-list xs* $\longleftrightarrow$ ($\forall$ *a*$\in$*set xs*. *b dvd a*)
  ⟨*proof*⟩

**lemma** *Lcm-fin-dvd-iff*:
  $Lcm_{fin}\ A\ dvd\ b \longleftrightarrow (\forall\, a{\in}A.\ a\ dvd\ b)$ **if** *finite A*
  ⟨*proof*⟩

**lemma** *lcm-list-dvd-iff*:
  *lcm-list xs dvd b* $\longleftrightarrow$ ($\forall$ *a*$\in$*set xs*. *a dvd b*)
  ⟨*proof*⟩

**lemma** *Gcd-fin-mult*:
  $Gcd_{fin}$ (*image* (*times b*) *A*) = *normalize* ($b * Gcd_{fin}\ A$) **if** *finite A*
  ⟨*proof*⟩

**lemma** *Lcm-fin-mult*:

$Lcm_{fin}$ (*image* (*times* *b*) *A*) = *normalize* (*b* ∗ $Lcm_{fin}$ *A*) **if** *A* ≠ {}
⟨*proof*⟩

**lemma** *unit-factor-Gcd-fin*:
*unit-factor* ($Gcd_{fin}$ *A*) = *of-bool* ($Gcd_{fin}$ *A* ≠ *0*)
⟨*proof*⟩

**lemma** *unit-factor-Lcm-fin*:
*unit-factor* ($Lcm_{fin}$ *A*) = *of-bool* ($Lcm_{fin}$ *A* ≠ *0*)
⟨*proof*⟩

**lemma** *is-unit-Gcd-fin-iff* [*simp*]:
*is-unit* ($Gcd_{fin}$ *A*) ⟷ $Gcd_{fin}$ *A* = *1*
⟨*proof*⟩

**lemma** *is-unit-Lcm-fin-iff* [*simp*]:
*is-unit* ($Lcm_{fin}$ *A*) ⟷ $Lcm_{fin}$ *A* = *1*
⟨*proof*⟩

**lemma** *Gcd-fin-0-iff*:
$Gcd_{fin}$ *A* = *0* ⟷ *A* ⊆ {*0*} ∧ *finite A*
⟨*proof*⟩

**lemma** *Lcm-fin-0-iff*:
$Lcm_{fin}$ *A* = *0* ⟷ *0* ∈ *A* **if** *finite A*
⟨*proof*⟩

**lemma** *Lcm-fin-1-iff*:
$Lcm_{fin}$ *A* = *1* ⟷ (∀ *a*∈*A*. *is-unit a*) ∧ *finite A*
⟨*proof*⟩

**end**

**context** *semiring-Gcd*
**begin**

**lemma** *Gcd-fin-eq-Gcd* [*simp*]:
$Gcd_{fin}$ *A* = *Gcd A* **if** *finite A* **for** *A* :: *'a set*
⟨*proof*⟩

**lemma** *Gcd-set-eq-fold* [*code-unfold*]:
*Gcd* (*set xs*) = *fold gcd xs 0*
⟨*proof*⟩

**lemma** *Lcm-fin-eq-Lcm* [*simp*]:
$Lcm_{fin}$ *A* = *Lcm A* **if** *finite A* **for** *A* :: *'a set*
⟨*proof*⟩

**lemma** *Lcm-set-eq-fold* [*code-unfold*]:

$Lcm\ (set\ xs) = fold\ lcm\ xs\ 1$
⟨proof⟩

**end**

## 87.4 Coprimality

**context** *semiring-gcd*
**begin**

**lemma** *coprime-imp-gcd-eq-1* [*simp*]:
$gcd\ a\ b = 1$ **if** $coprime\ a\ b$
⟨proof⟩

**lemma** *gcd-eq-1-imp-coprime* [*dest*!]:
$coprime\ a\ b$ **if** $gcd\ a\ b = 1$
⟨proof⟩

**lemma** *coprime-iff-gcd-eq-1* [*presburger*, *code*]:
$coprime\ a\ b \longleftrightarrow gcd\ a\ b = 1$
⟨proof⟩

**lemma** *is-unit-gcd* [*simp*]:
$is\text{-}unit\ (gcd\ a\ b) \longleftrightarrow coprime\ a\ b$
⟨proof⟩

**lemma** *coprime-add-one-left* [*simp*]: $coprime\ (a + 1)\ a$
⟨proof⟩

**lemma** *coprime-add-one-right* [*simp*]: $coprime\ a\ (a + 1)$
⟨proof⟩

**lemma** *coprime-mult-left-iff* [*simp*]:
$coprime\ (a * b)\ c \longleftrightarrow coprime\ a\ c \wedge coprime\ b\ c$
⟨proof⟩

**lemma** *coprime-mult-right-iff* [*simp*]:
$coprime\ c\ (a * b) \longleftrightarrow coprime\ c\ a \wedge coprime\ c\ b$
⟨proof⟩

**lemma** *coprime-power-left-iff* [*simp*]:
$coprime\ (a\ \widehat{}\ n)\ b \longleftrightarrow coprime\ a\ b \vee n = 0$
⟨proof⟩

**lemma** *coprime-power-right-iff* [*simp*]:
$coprime\ a\ (b\ \widehat{}\ n) \longleftrightarrow coprime\ a\ b \vee n = 0$
⟨proof⟩

**lemma** *prod-coprime-left*:

*coprime* $(\prod i{\in}A.\ f\ i)$ *a* **if** $\bigwedge i.\ i \in A \Longrightarrow$ *coprime* $(f\ i)$ *a*
⟨*proof*⟩

**lemma** *prod-coprime-right*:
*coprime a* $(\prod i{\in}A.\ f\ i)$ **if** $\bigwedge i.\ i \in A \Longrightarrow$ *coprime a* $(f\ i)$
⟨*proof*⟩

**lemma** *prod-list-coprime-left*:
*coprime* (*prod-list xs*) *a* **if** $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *coprime x a*
⟨*proof*⟩

**lemma** *prod-list-coprime-right*:
*coprime a* (*prod-list xs*) **if** $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *coprime a x*
⟨*proof*⟩

**lemma** *coprime-dvd-mult-left-iff*:
*a dvd b* $*$ *c* $\longleftrightarrow$ *a dvd b* **if** *coprime a c*
⟨*proof*⟩

**lemma** *coprime-dvd-mult-right-iff*:
*a dvd c* $*$ *b* $\longleftrightarrow$ *a dvd b* **if** *coprime a c*
⟨*proof*⟩

**lemma** *divides-mult*:
*a* $*$ *b dvd c* **if** *a dvd c* **and** *b dvd c* **and** *coprime a b*
⟨*proof*⟩

**lemma** *div-gcd-coprime*:
**assumes** $a \neq 0 \vee b \neq 0$
**shows** *coprime* (*a div gcd a b*) (*b div gcd a b*)
⟨*proof*⟩

**lemma** *gcd-coprime*:
**assumes** *c*: *gcd a b* $\neq 0$
**and** *a*: $a = a' *$ *gcd a b*
**and** *b*: $b = b' *$ *gcd a b*
**shows** *coprime* $a'\ b'$
⟨*proof*⟩

**lemma** *gcd-coprime-exists*:
**assumes** *gcd a b* $\neq 0$
**shows** $\exists\, a'\ b'.\ a = a' *$ *gcd a b* $\wedge\ b = b' *$ *gcd a b* $\wedge$ *coprime* $a'\ b'$
⟨*proof*⟩

**lemma** *pow-divides-pow-iff* [*simp*]:
$a\ \hat{}\ n$ *dvd* $b\ \hat{}\ n \longleftrightarrow$ *a dvd b* **if** $n > 0$
⟨*proof*⟩

**lemma** *coprime-crossproduct*:

**fixes** $a\ b\ c\ d :: 'a$
**assumes** *coprime a d* **and** *coprime b c*
**shows** $normalize\ a * normalize\ c = normalize\ b * normalize\ d \longleftrightarrow$
$normalize\ a = normalize\ b \wedge normalize\ c = normalize\ d$
(**is** $?lhs \longleftrightarrow ?rhs$)
⟨*proof*⟩

**lemma** *gcd-mult-left-left-cancel*:
$gcd\ (c * a)\ b = gcd\ a\ b$ **if** *coprime b c*
⟨*proof*⟩

**lemma** *gcd-mult-left-right-cancel*:
$gcd\ (a * c)\ b = gcd\ a\ b$ **if** *coprime b c*
⟨*proof*⟩

**lemma** *gcd-mult-right-left-cancel*:
$gcd\ a\ (c * b) = gcd\ a\ b$ **if** *coprime a c*
⟨*proof*⟩

**lemma** *gcd-mult-right-right-cancel*:
$gcd\ a\ (b * c) = gcd\ a\ b$ **if** *coprime a c*
⟨*proof*⟩

**lemma** *gcd-exp-weak*:
$gcd\ (a\ \hat{}\ n)\ (b\ \hat{}\ n) = normalize\ (gcd\ a\ b\ \hat{}\ n)$
⟨*proof*⟩

**lemma** *division-decomp*:
**assumes** *a dvd b* $*$ *c*
**shows** $\exists\, b'\ c'.\ a = b' * c' \wedge b'\ dvd\ b \wedge c'\ dvd\ c$
⟨*proof*⟩

**lemma** *lcm-coprime*: $coprime\ a\ b \Longrightarrow lcm\ a\ b = normalize\ (a * b)$
⟨*proof*⟩

**end**

**context** *ring-gcd*
**begin**

**lemma** *coprime-minus-left-iff* [*simp*]:
$coprime\ (-\ a)\ b \longleftrightarrow coprime\ a\ b$
⟨*proof*⟩

**lemma** *coprime-minus-right-iff* [*simp*]:
$coprime\ a\ (-\ b) \longleftrightarrow coprime\ a\ b$
⟨*proof*⟩

**lemma** *coprime-diff-one-left* [*simp*]: $coprime\ (a - 1)\ a$

⟨*proof*⟩

**lemma** *coprime-doff-one-right* [*simp*]: *coprime a* (*a* − *1*)
⟨*proof*⟩

**end**

**context** *semiring-Gcd*
**begin**

**lemma** *Lcm-coprime*:
**assumes** *finite A*
**and** $A \neq \{\}$
**and** $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \neq b \Longrightarrow$ *coprime a b*
**shows** *Lcm A* = *normalize* ($\prod A$)
⟨*proof*⟩

**lemma** *Lcm-coprime'*:
*card* $A \neq 0 \Longrightarrow$
($\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \neq b \Longrightarrow$ *coprime a b*) $\Longrightarrow$
*Lcm A* = *normalize* ($\prod A$)
⟨*proof*⟩

**end**

And some consequences: cancellation modulo $m$

**lemma** *mult-mod-cancel-right*:
**fixes** *m* :: *'a*::{*euclidean-ring-cancel*,*semiring-gcd*}
**assumes** *eq*: (*a* ∗ *n*) *mod m* = (*b* ∗ *n*) *mod m* **and** *coprime m n*
**shows** *a mod m* = *b mod m*
⟨*proof*⟩

**lemma** *mult-mod-cancel-left*:
**fixes** *m* :: *'a*::{*euclidean-ring-cancel*,*semiring-gcd*}
**assumes** (*n* ∗ *a*) *mod m* = (*n* ∗ *b*) *mod m* **and** *coprime m n*
**shows** *a mod m* = *b mod m*
⟨*proof*⟩

## 87.5 GCD and LCM for multiplicative normalisation functions

**class** *semiring-gcd-mult-normalize* = *semiring-gcd* + *normalization-semidom-multiplicative*
**begin**

**lemma** *mult-gcd-left*: *c* ∗ *gcd a b* = *unit-factor c* ∗ *gcd* (*c* ∗ *a*) (*c* ∗ *b*)
⟨*proof*⟩

**lemma** *mult-gcd-right*: *gcd a b* ∗ *c* = *gcd* (*a* ∗ *c*) (*b* ∗ *c*) ∗ *unit-factor c*
⟨*proof*⟩

**lemma** *gcd-mult-distrib'*: *normalize c* $*$ *gcd a b* = *gcd* (*c* $*$ *a*) (*c* $*$ *b*)
⟨*proof*⟩

**lemma** *gcd-mult-distrib*: *k* $*$ *gcd a b* = *gcd* (*k* $*$ *a*) (*k* $*$ *b*) $*$ *unit-factor k*
⟨*proof*⟩

**lemma** *gcd-mult-lcm* [*simp*]: *gcd a b* $*$ *lcm a b* = *normalize a* $*$ *normalize b*
⟨*proof*⟩

**lemma** *lcm-mult-gcd* [*simp*]: *lcm a b* $*$ *gcd a b* = *normalize a* $*$ *normalize b*
⟨*proof*⟩

**lemma** *mult-lcm-left*: *c* $*$ *lcm a b* = *unit-factor c* $*$ *lcm* (*c* $*$ *a*) (*c* $*$ *b*)
⟨*proof*⟩

**lemma** *mult-lcm-right*: *lcm a b* $*$ *c* = *lcm* (*a* $*$ *c*) (*b* $*$ *c*) $*$ *unit-factor c*
⟨*proof*⟩

**lemma** *lcm-gcd-prod*: *lcm a b* $*$ *gcd a b* = *normalize* (*a* $*$ *b*)
⟨*proof*⟩

**lemma** *lcm-mult-distrib'*: *normalize c* $*$ *lcm a b* = *lcm* (*c* $*$ *a*) (*c* $*$ *b*)
⟨*proof*⟩

**lemma** *lcm-mult-distrib*: *k* $*$ *lcm a b* = *lcm* (*k* $*$ *a*) (*k* $*$ *b*) $*$ *unit-factor k*
⟨*proof*⟩

**lemma** *coprime-crossproduct'*:
**fixes** *a b c d*
**assumes** $b \neq 0$
**assumes** *unit-factors*: *unit-factor b* = *unit-factor d*
**assumes** *coprime*: *coprime a b coprime c d*
**shows** $a * d = b * c \longleftrightarrow a = c \wedge b = d$
⟨*proof*⟩

**lemma** *gcd-exp* [*simp*]:
*gcd* (*a* ^ *n*) (*b* ^ *n*) = *gcd a b* ^ *n*
⟨*proof*⟩

**end**

## 87.6 GCD and LCM on *nat* and *int*

**instantiation** *nat* :: *gcd*
**begin**

**fun** *gcd-nat* :: *nat* ⇒ *nat* ⇒ *nat*
**where** *gcd-nat x y* = (*if y* = *0 then x else gcd y* (*x mod y*))

**definition** *lcm-nat* :: *nat* ⇒ *nat* ⇒ *nat*
**where** *lcm-nat x y* = *x* ∗ *y div* (*gcd x y*)

**instance** ⟨*proof*⟩

**end**

**instantiation** *int* :: *gcd*
**begin**

**definition** *gcd-int* :: *int* ⇒ *int* ⇒ *int*
**where** *gcd-int x y* = *int* (*gcd* (*nat* |*x*|) (*nat* |*y*|))

**definition** *lcm-int* :: *int* ⇒ *int* ⇒ *int*
**where** *lcm-int x y* = *int* (*lcm* (*nat* |*x*|) (*nat* |*y*|))

**instance** ⟨*proof*⟩

**end**

**lemma** *gcd-int-int-eq* [*simp*]:
*gcd* (*int m*) (*int n*) = *int* (*gcd m n*)
⟨*proof*⟩

**lemma** *gcd-nat-abs-left-eq* [*simp*]:
*gcd* (*nat* |*k*|) *n* = *nat* (*gcd k* (*int n*))
⟨*proof*⟩

**lemma** *gcd-nat-abs-right-eq* [*simp*]:
*gcd n* (*nat* |*k*|) = *nat* (*gcd* (*int n*) *k*)
⟨*proof*⟩

**lemma** *abs-gcd-int* [*simp*]:
|*gcd x y*| = *gcd x y*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *gcd-abs1-int* [*simp*]:
*gcd* |*x*| *y* = *gcd x y*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *gcd-abs2-int* [*simp*]:
*gcd x* |*y*| = *gcd x y*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *lcm-int-int-eq* [*simp*]:

$lcm\ (int\ m)\ (int\ n) = int\ (lcm\ m\ n)$
⟨*proof*⟩

**lemma** *lcm-nat-abs-left-eq* [*simp*]:
$lcm\ (nat\ |k|)\ n = nat\ (lcm\ k\ (int\ n))$
⟨*proof*⟩

**lemma** *lcm-nat-abs-right-eq* [*simp*]:
$lcm\ n\ (nat\ |k|) = nat\ (lcm\ (int\ n)\ k)$
⟨*proof*⟩

**lemma** *lcm-abs1-int* [*simp*]:
$lcm\ |x|\ y = lcm\ x\ y$
**for** $x\ y$ :: *int*
⟨*proof*⟩

**lemma** *lcm-abs2-int* [*simp*]:
$lcm\ x\ |y| = lcm\ x\ y$
**for** $x\ y$ :: *int*
⟨*proof*⟩

**lemma** *abs-lcm-int* [*simp*]: $|lcm\ i\ j| = lcm\ i\ j$
**for** $i\ j$ :: *int*
⟨*proof*⟩

**lemma** *gcd-nat-induct* [*case-names base step*]:
**fixes** $m\ n$ :: *nat*
**assumes** $\bigwedge m.\ P\ m\ 0$
**and** $\bigwedge m\ n.\ 0 < n \Longrightarrow P\ n\ (m\ mod\ n) \Longrightarrow P\ m\ n$
**shows** $P\ m\ n$
⟨*proof*⟩

**lemma** *gcd-neg1-int* [*simp*]: $gcd\ (-\ x)\ y = gcd\ x\ y$
**for** $x\ y$ :: *int*
⟨*proof*⟩

**lemma** *gcd-neg2-int* [*simp*]: $gcd\ x\ (-\ y) = gcd\ x\ y$
**for** $x\ y$ :: *int*
⟨*proof*⟩

**lemma** *gcd-cases-int*:
**fixes** $x\ y$ :: *int*
**assumes** $x \geq 0 \Longrightarrow y \geq 0 \Longrightarrow P\ (gcd\ x\ y)$
**and** $x \geq 0 \Longrightarrow y \leq 0 \Longrightarrow P\ (gcd\ x\ (-\ y))$
**and** $x \leq 0 \Longrightarrow y \geq 0 \Longrightarrow P\ (gcd\ (-\ x)\ y)$
**and** $x \leq 0 \Longrightarrow y \leq 0 \Longrightarrow P\ (gcd\ (-\ x)\ (-\ y))$
**shows** $P\ (gcd\ x\ y)$
⟨*proof*⟩

**lemma** *gcd-ge-0-int* [*simp*]: *gcd* (*x*::*int*) *y* >= *0*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *lcm-neg1-int*: *lcm* (− *x*) *y* = *lcm x y*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *lcm-neg2-int*: *lcm x* (− *y*) = *lcm x y*
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *lcm-cases-int*:
**fixes** *x y* :: *int*
**assumes** $x \geq 0 \Longrightarrow y \geq 0 \Longrightarrow P\ (lcm\ x\ y)$
**and** $x \geq 0 \Longrightarrow y \leq 0 \Longrightarrow P\ (lcm\ x\ (-\ y))$
**and** $x \leq 0 \Longrightarrow y \geq 0 \Longrightarrow P\ (lcm\ (-\ x)\ y)$
**and** $x \leq 0 \Longrightarrow y \leq 0 \Longrightarrow P\ (lcm\ (-\ x)\ (-\ y))$
**shows** *P* (*lcm x y*)
⟨*proof*⟩

**lemma** *lcm-ge-0-int* [*simp*]: $lcm\ x\ y \geq 0$
**for** *x y* :: *int*
⟨*proof*⟩

**lemma** *gcd-0-nat*: *gcd x 0* = *x*
**for** *x* :: *nat*
⟨*proof*⟩

**lemma** *gcd-0-int* [*simp*]: *gcd x 0* = |*x*|
**for** *x* :: *int*
⟨*proof*⟩

**lemma** *gcd-0-left-nat*: *gcd 0 x* = *x*
**for** *x* :: *nat*
⟨*proof*⟩

**lemma** *gcd-0-left-int* [*simp*]: *gcd 0 x* = |*x*|
**for** *x* :: *int*
⟨*proof*⟩

**lemma** *gcd-red-nat*: *gcd x y* = *gcd y* (*x mod y*)
**for** *x y* :: *nat*
⟨*proof*⟩

Weaker, but useful for the simplifier.

**lemma** *gcd-non-0-nat*: $y \neq 0 \Longrightarrow gcd\ x\ y = gcd\ y\ (x\ mod\ y)$
**for** *x y* :: *nat*
⟨*proof*⟩

**lemma** *gcd-1-nat* [*simp*]: $gcd\ m\ 1 = 1$
  **for** $m :: nat$
  ⟨*proof*⟩

**lemma** *gcd-Suc-0* [*simp*]: $gcd\ m\ (Suc\ 0) = Suc\ 0$
  **for** $m :: nat$
  ⟨*proof*⟩

**lemma** *gcd-1-int* [*simp*]: $gcd\ m\ 1 = 1$
  **for** $m :: int$
  ⟨*proof*⟩

**lemma** *gcd-idem-nat*: $gcd\ x\ x = x$
  **for** $x :: nat$
  ⟨*proof*⟩

**lemma** *gcd-idem-int*: $gcd\ x\ x = |x|$
  **for** $x :: int$
  ⟨*proof*⟩

**declare** *gcd-nat.simps* [*simp del*]

*gcd m n* divides $m$ and $n$. The conjunctions don't seem provable separately.

**instance** $nat :: semiring\text{-}gcd$
⟨*proof*⟩

**instance** $int :: ring\text{-}gcd$
⟨*proof*⟩

**lemma** *gcd-le1-nat* [*simp*]: $a \neq 0 \Longrightarrow gcd\ a\ b \leq a$
  **for** $a\ b :: nat$
  ⟨*proof*⟩

**lemma** *gcd-le2-nat* [*simp*]: $b \neq 0 \Longrightarrow gcd\ a\ b \leq b$
  **for** $a\ b :: nat$
  ⟨*proof*⟩

**lemma** *gcd-le1-int* [*simp*]: $a > 0 \Longrightarrow gcd\ a\ b \leq a$
  **for** $a\ b :: int$
  ⟨*proof*⟩

**lemma** *gcd-le2-int* [*simp*]: $b > 0 \Longrightarrow gcd\ a\ b \leq b$
  **for** $a\ b :: int$
  ⟨*proof*⟩

**lemma** *gcd-pos-nat* [*simp*]: $gcd\ m\ n > 0 \longleftrightarrow m \neq 0 \lor n \neq 0$
  **for** $m\ n :: nat$
  ⟨*proof*⟩

**lemma** *gcd-pos-int* [*simp*]: $gcd\ m\ n > 0 \longleftrightarrow m \neq 0 \vee n \neq 0$
**for** $m\ n :: int$
⟨*proof*⟩

**lemma** *gcd-unique-nat*: $d\ dvd\ a \wedge d\ dvd\ b \wedge (\forall e.\ e\ dvd\ a \wedge e\ dvd\ b \longrightarrow e\ dvd\ d) \longleftrightarrow d = gcd\ a\ b$
**for** $d\ a :: nat$
⟨*proof*⟩

**lemma** *gcd-unique-int*:
$d \geq 0 \wedge d\ dvd\ a \wedge d\ dvd\ b \wedge (\forall e.\ e\ dvd\ a \wedge e\ dvd\ b \longrightarrow e\ dvd\ d) \longleftrightarrow d = gcd\ a\ b$
**for** $d\ a :: int$
⟨*proof*⟩

**interpretation** *gcd-nat*:
*semilattice-neutr-order* $gcd\ 0::nat\ Rings.dvd\ \lambda m\ n.\ m\ dvd\ n \wedge m \neq n$
⟨*proof*⟩

**lemma** *gcd-proj1-if-dvd-int* [*simp*]: $x\ dvd\ y \Longrightarrow gcd\ x\ y = |x|$
**for** $x\ y :: int$
⟨*proof*⟩

**lemma** *gcd-proj2-if-dvd-int* [*simp*]: $y\ dvd\ x \Longrightarrow gcd\ x\ y = |y|$
**for** $x\ y :: int$
⟨*proof*⟩

Multiplication laws.

**lemma** *gcd-mult-distrib-nat*: $k * gcd\ m\ n = gcd\ (k * m)\ (k * n)$
**for** $k\ m\ n :: nat$
— [1, page 27]
⟨*proof*⟩

**lemma** *gcd-mult-distrib-int*: $|k| * gcd\ m\ n = gcd\ (k * m)\ (k * n)$
**for** $k\ m\ n :: int$
⟨*proof*⟩

Addition laws.

**lemma** *gcd-diff1-nat*: $m \geq n \Longrightarrow gcd\ (m - n)\ n = gcd\ m\ n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *gcd-diff2-nat*: $n \geq m \Longrightarrow gcd\ (n - m)\ n = gcd\ m\ n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *gcd-non-0-int*:

**fixes** $x\ y :: int$
**assumes** $y > 0$ **shows** $gcd\ x\ y = gcd\ y\ (x\ mod\ y)$
⟨*proof*⟩

**lemma** *gcd-red-int*: $gcd\ x\ y = gcd\ y\ (x\ mod\ y)$
**for** $x\ y :: int$
⟨*proof*⟩

**lemma** *finite-divisors-nat* [*simp*]:
**fixes** $m :: nat$
**assumes** $m > 0$
**shows** $finite\ \{d.\ d\ dvd\ m\}$
⟨*proof*⟩

**lemma** *finite-divisors-int* [*simp*]:
**fixes** $i :: int$
**assumes** $i \neq 0$
**shows** $finite\ \{d.\ d\ dvd\ i\}$
⟨*proof*⟩

**lemma** *Max-divisors-self-nat* [*simp*]: $n \neq 0 \Longrightarrow Max\ \{d::nat.\ d\ dvd\ n\} = n$
⟨*proof*⟩

**lemma** *Max-divisors-self-int* [*simp*]:
**assumes** $n \neq 0$ **shows** $Max\ \{d::int.\ d\ dvd\ n\} = |n|$
⟨*proof*⟩

**lemma** *gcd-is-Max-divisors-nat*:
**fixes** $m\ n :: nat$
**assumes** $n > 0$ **shows** $gcd\ m\ n = Max\ \{d.\ d\ dvd\ m \wedge d\ dvd\ n\}$
⟨*proof*⟩

**lemma** *gcd-is-Max-divisors-int*:
**fixes** $m\ n :: int$
**assumes** $n \neq 0$ **shows** $gcd\ m\ n = Max\ \{d.\ d\ dvd\ m \wedge d\ dvd\ n\}$
⟨*proof*⟩

**lemma** *gcd-code-int* [*code*]: $gcd\ k\ l = |if\ l = 0\ then\ k\ else\ gcd\ l\ (|k|\ mod\ |l|)|$
**for** $k\ l :: int$
⟨*proof*⟩

**lemma** *coprime-Suc-left-nat* [*simp*]:
$coprime\ (Suc\ n)\ n$
⟨*proof*⟩

**lemma** *coprime-Suc-right-nat* [*simp*]:
  *coprime n* (*Suc n*)
  ⟨*proof*⟩

**lemma** *coprime-diff-one-left-nat* [*simp*]:
  *coprime* ($n - 1$) *n* **if** $n > 0$ **for** *n* :: *nat*
  ⟨*proof*⟩

**lemma** *coprime-diff-one-right-nat* [*simp*]:
  *coprime n* ($n - 1$) **if** $n > 0$ **for** *n* :: *nat*
  ⟨*proof*⟩

**lemma** *coprime-crossproduct-nat*:
  **fixes** *a b c d* :: *nat*
  **assumes** *coprime a d* **and** *coprime b c*
  **shows** $a * c = b * d \longleftrightarrow a = b \land c = d$
  ⟨*proof*⟩

**lemma** *coprime-crossproduct-int*:
  **fixes** *a b c d* :: *int*
  **assumes** *coprime a d* **and** *coprime b c*
  **shows** $|a| * |c| = |b| * |d| \longleftrightarrow |a| = |b| \land |c| = |d|$
  ⟨*proof*⟩

## 87.7 Bezout's theorem

Function *bezw* returns a pair of witnesses to Bezout's theorem – see the theorems that follow the definition.

**fun** *bezw* :: *nat* ⇒ *nat* ⇒ *int* ∗ *int*
  **where** *bezw x y* =
    (*if y* = *0 then* (*1*, *0*)
     *else*
      (*snd* (*bezw y* (*x mod y*)),
       *fst* (*bezw y* (*x mod y*)) − *snd* (*bezw y* (*x mod y*)) ∗ *int*(*x div y*)))

**lemma** *bezw-0* [*simp*]: *bezw x 0* = (*1*, *0*)
  ⟨*proof*⟩

**lemma** *bezw-non-0*:
  $y > 0 \Longrightarrow$ *bezw x y* =
    (*snd* (*bezw y* (*x mod y*)), *fst* (*bezw y* (*x mod y*)) − *snd* (*bezw y* (*x mod y*)) ∗ *int*(*x div y*))
  ⟨*proof*⟩

**declare** *bezw.simps* [*simp del*]

**lemma** *bezw-aux*: *int* (*gcd x y*) = *fst* (*bezw x y*) ∗ *int x* + *snd* (*bezw x y*) ∗ *int y*
⟨*proof*⟩

**lemma** *bezout-int*: $\exists\, u\ v.\ u * x + v * y = gcd\ x\ y$
  **for** $x\ y :: int$
⟨*proof*⟩

Versions of Bezout for *nat*, by Amine Chaieb.

**lemma** *Euclid-induct* [*case-names swap zero add*]:
  **fixes** $P :: nat \Rightarrow nat \Rightarrow bool$
  **assumes** *c*: $\bigwedge a\ b.\ P\ a\ b \longleftrightarrow P\ b\ a$
    **and** *z*: $\bigwedge a.\ P\ a\ 0$
    **and** *add*: $\bigwedge a\ b.\ P\ a\ b \longrightarrow P\ a\ (a + b)$
  **shows** $P\ a\ b$
⟨*proof*⟩

**lemma** *bezout-lemma-nat*:
  **fixes** $d{::}nat$
  **shows** $[\![d\ dvd\ a;\ d\ dvd\ b;\ a * x = b * y + d \lor b * x = a * y + d]\!]$
    $\Longrightarrow \exists\, x\ y.\ d\ dvd\ a \land d\ dvd\ a + b \land (a * x = (a + b) * y + d \lor (a + b) * x = a * y + d)$
  ⟨*proof*⟩

**lemma** *bezout-add-nat*:
  $\exists\, (d{::}nat)\ x\ y.\ d\ dvd\ a \land d\ dvd\ b \land (a * x = b * y + d \lor b * x = a * y + d)$
⟨*proof*⟩

**lemma** *bezout1-nat*: $\exists\, (d{::}nat)\ x\ y.\ d\ dvd\ a \land d\ dvd\ b \land (a * x - b * y = d \lor b * x - a * y = d)$
  ⟨*proof*⟩

**lemma** *bezout-add-strong-nat*:
  **fixes** $a\ b :: nat$
  **assumes** *a*: $a \neq 0$
  **shows** $\exists\, d\ x\ y.\ d\ dvd\ a \land d\ dvd\ b \land a * x = b * y + d$
⟨*proof*⟩

**lemma** *bezout-nat*:
  **fixes** $a :: nat$
  **assumes** *a*: $a \neq 0$
  **shows** $\exists\, x\ y.\ a * x = b * y + gcd\ a\ b$
⟨*proof*⟩

## 87.8 LCM properties on *nat* and *int*

**lemma** *lcm-altdef-int* [*code*]: $lcm\ a\ b = |a| * |b|\ div\ gcd\ a\ b$
  **for** $a\ b :: int$
  ⟨*proof*⟩

**lemma** *prod-gcd-lcm-nat*: $m * n = gcd\ m\ n * lcm\ m\ n$

**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *prod-gcd-lcm-int*: $|m| * |n| = gcd\ m\ n * lcm\ m\ n$
**for** $m\ n :: int$
$\langle proof \rangle$

**lemma** *lcm-pos-nat*: $m > 0 \Longrightarrow n > 0 \Longrightarrow lcm\ m\ n > 0$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *lcm-pos-int*: $m \neq 0 \Longrightarrow n \neq 0 \Longrightarrow lcm\ m\ n > 0$
**for** $m\ n :: int$
$\langle proof \rangle$

**lemma** *dvd-pos-nat*: $n > 0 \Longrightarrow m\ dvd\ n \Longrightarrow m > 0$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *lcm-unique-nat*:
$a\ dvd\ d \wedge b\ dvd\ d \wedge (\forall e.\ a\ dvd\ e \wedge b\ dvd\ e \longrightarrow d\ dvd\ e) \longleftrightarrow d = lcm\ a\ b$
**for** $a\ b\ d :: nat$
$\langle proof \rangle$

**lemma** *lcm-unique-int*:
$d \geq 0 \wedge a\ dvd\ d \wedge b\ dvd\ d \wedge (\forall e.\ a\ dvd\ e \wedge b\ dvd\ e \longrightarrow d\ dvd\ e) \longleftrightarrow d = lcm\ a\ b$
**for** $a\ b\ d :: int$
$\langle proof \rangle$

**lemma** *lcm-proj2-if-dvd-nat* [*simp*]: $x\ dvd\ y \Longrightarrow lcm\ x\ y = y$
**for** $x\ y :: nat$
$\langle proof \rangle$

**lemma** *lcm-proj2-if-dvd-int* [*simp*]: $x\ dvd\ y \Longrightarrow lcm\ x\ y = |y|$
**for** $x\ y :: int$
$\langle proof \rangle$

**lemma** *lcm-proj1-if-dvd-nat* [*simp*]: $x\ dvd\ y \Longrightarrow lcm\ y\ x = y$
**for** $x\ y :: nat$
$\langle proof \rangle$

**lemma** *lcm-proj1-if-dvd-int* [*simp*]: $x\ dvd\ y \Longrightarrow lcm\ y\ x = |y|$
**for** $x\ y :: int$
$\langle proof \rangle$

**lemma** *lcm-proj1-iff-nat* [*simp*]: $lcm\ m\ n = m \longleftrightarrow n\ dvd\ m$
**for** $m\ n :: nat$
$\langle proof \rangle$

**lemma** *lcm-proj2-iff-nat* [*simp*]: $lcm\ m\ n = n \longleftrightarrow m\ dvd\ n$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *lcm-proj1-iff-int* [*simp*]: $lcm\ m\ n = |m| \longleftrightarrow n\ dvd\ m$
**for** $m\ n :: int$
⟨*proof*⟩

**lemma** *lcm-proj2-iff-int* [*simp*]: $lcm\ m\ n = |n| \longleftrightarrow m\ dvd\ n$
**for** $m\ n :: int$
⟨*proof*⟩

**lemma** *lcm-1-iff-nat* [*simp*]: $lcm\ m\ n = Suc\ 0 \longleftrightarrow m = Suc\ 0 \wedge n = Suc\ 0$
**for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *lcm-1-iff-int* [*simp*]: $lcm\ m\ n = 1 \longleftrightarrow (m = 1 \vee m = -1) \wedge (n = 1 \vee n = -1)$
**for** $m\ n :: int$
⟨*proof*⟩

## 87.9 The complete divisibility lattice on *nat* and *int*

Lifting *gcd* and *lcm* to sets (*Gcd* / *Lcm*). *Gcd* is defined via *Lcm* to facilitate the proof that we have a complete lattice.

**instantiation** *nat* :: *semiring-Gcd*
**begin**

**interpretation** *semilattice-neutr-set lcm 1::nat*
⟨*proof*⟩

**definition** $Lcm\ M = (if\ finite\ M\ then\ F\ M\ else\ 0)$ **for** $M :: nat\ set$

**lemma** *Lcm-nat-empty*: $Lcm\ \{\} = (1::nat)$
⟨*proof*⟩

**lemma** *Lcm-nat-insert*: $Lcm\ (insert\ n\ M) = lcm\ n\ (Lcm\ M)$ **for** $n :: nat$
⟨*proof*⟩

**lemma** *Lcm-nat-infinite*: $infinite\ M \Longrightarrow Lcm\ M = 0$ **for** $M :: nat\ set$
⟨*proof*⟩

**lemma** *dvd-Lcm-nat* [*simp*]:
**fixes** $M :: nat\ set$
**assumes** $m \in M$
**shows** $m\ dvd\ Lcm\ M$
⟨*proof*⟩

**lemma** *Lcm-dvd-nat* [*simp*]:
  **fixes** $M$ :: *nat set*
  **assumes** $\forall\, m{\in}M.\ m\ dvd\ n$
  **shows** *Lcm M dvd n*
⟨*proof*⟩

**definition** $Gcd\ M = Lcm\ \{d.\ \forall\, m{\in}M.\ d\ dvd\ m\}$ **for** $M$ :: *nat set*

**instance**
⟨*proof*⟩

**end**

**lemma** *Gcd-nat-eq-one*: $1 \in N \Longrightarrow Gcd\ N = 1$
  **for** $N$ :: *nat set*
  ⟨*proof*⟩

**instance** *nat* :: *semiring-gcd-mult-normalize*
  ⟨*proof*⟩

Alternative characterizations of Gcd:

**lemma** *Gcd-eq-Max*:
  **fixes** $M$ :: *nat set*
  **assumes** *finite* ($M$::*nat set*) **and** $M \neq \{\}$ **and** $0 \notin M$
  **shows** $Gcd\ M = Max\ (\bigcap m{\in}M.\ \{d.\ d\ dvd\ m\})$
⟨*proof*⟩

**lemma** *Gcd-remove0-nat*: $Gcd\ M = Gcd\ (M - \{0\})$
  **for** $M$ :: *nat set*
⟨*proof*⟩

**lemma** *Lcm-in-lcm-closed-set-nat*:
  **fixes** $M$ :: *nat set*
  **assumes** *finite* $M$ $M \neq \{\}$ $\bigwedge m\ n.\ [\![m \in M;\ n \in M]\!] \Longrightarrow lcm\ m\ n \in M$
  **shows** $Lcm\ M \in M$
  ⟨*proof*⟩

**lemma** *Lcm-eq-Max-nat*:
  **fixes** $M$ :: *nat set*
  **assumes** $M$: *finite* $M$ $M \neq \{\}$ $0 \notin M$ **and** *lcm*: $\bigwedge m\ n.\ [\![m \in M;\ n \in M]\!] \Longrightarrow$
$lcm\ m\ n \in M$
  **shows** $Lcm\ M = Max\ M$
⟨*proof*⟩

**lemma** *mult-inj-if-coprime-nat*:
  $inj\text{-}on\ f\ A \Longrightarrow inj\text{-}on\ g\ B \Longrightarrow (\bigwedge a\ b.\ [\![a{\in}A;\ b{\in}B]\!] \Longrightarrow coprime\ (f\ a)\ (g\ b)) \Longrightarrow$
    $inj\text{-}on\ (\lambda(a,\ b).\ f\ a * g\ b)\ (A \times B)$
  **for** $f$ :: $'a \Rightarrow nat$ **and** $g$ :: $'b \Rightarrow nat$
  ⟨*proof*⟩

### 87.9.1 Setwise GCD and LCM for integers

**instantiation** *int* :: *Gcd*
**begin**

**definition** *Gcd-int* :: *int set* ⇒ *int*
**where** *Gcd K* = *int* (*GCD k*∈*K*. (*nat* ∘ *abs*) *k*)

**definition** *Lcm-int* :: *int set* ⇒ *int*
**where** *Lcm K* = *int* (*LCM k*∈*K*. (*nat* ∘ *abs*) *k*)

**instance** ⟨*proof*⟩

**end**

**lemma** *Gcd-int-eq* [*simp*]:
(*GCD n*∈*N*. *int n*) = *int* (*Gcd N*)
⟨*proof*⟩

**lemma** *Gcd-nat-abs-eq* [*simp*]:
(*GCD k*∈*K*. *nat* |*k*|) = *nat* (*Gcd K*)
⟨*proof*⟩

**lemma** *abs-Gcd-eq* [*simp*]:
|*Gcd K*| = *Gcd K* **for** *K* :: *int set*
⟨*proof*⟩

**lemma** *uminus-Gcd-eq* [*simp*]:
**fixes** *K*::*int set*
**shows** *Gcd* (*uminus* ‘ *K*) = *Gcd K*
⟨*proof*⟩

**lemma** *Gcd-int-greater-eq-0* [*simp*]:
*Gcd K* ≥ *0*
**for** *K* :: *int set*
⟨*proof*⟩

**lemma** *Gcd-abs-eq* [*simp*]:
(*GCD k*∈*K*. |*k*|) = *Gcd K*
**for** *K* :: *int set*
⟨*proof*⟩

**lemma** *Lcm-int-eq* [*simp*]:
(*LCM n*∈*N*. *int n*) = *int* (*Lcm N*)
⟨*proof*⟩

**lemma** *Lcm-nat-abs-eq* [*simp*]:
(*LCM k*∈*K*. *nat* |*k*|) = *nat* (*Lcm K*)
⟨*proof*⟩

**lemma** *abs-Lcm-eq* [*simp*]:
$|Lcm\ K| = Lcm\ K$ **for** $K :: int\ set$
⟨*proof*⟩

**lemma** *Lcm-int-greater-eq-0* [*simp*]:
$Lcm\ K \geq 0$
**for** $K :: int\ set$
⟨*proof*⟩

**lemma** *Lcm-abs-eq* [*simp*]:
$(LCM\ k{\in}K.\ |k|) = Lcm\ K$
**for** $K :: int\ set$
⟨*proof*⟩

**instance** *int* :: *semiring-Gcd*
⟨*proof*⟩

**instance** *int* :: *semiring-gcd-mult-normalize*
⟨*proof*⟩

## 87.10 GCD and LCM on *integer*

**instantiation** *integer* :: *gcd*
**begin**

**context**
**includes** *integer.lifting*
**begin**

**lift-definition** *gcd-integer* :: *integer* ⇒ *integer* ⇒ *integer* **is** *gcd* ⟨*proof*⟩

**lift-definition** *lcm-integer* :: *integer* ⇒ *integer* ⇒ *integer* **is** *lcm* ⟨*proof*⟩

**end**

**instance** ⟨*proof*⟩

**end**

**lifting-update** *integer.lifting*
**lifting-forget** *integer.lifting*

**context**
**includes** *integer.lifting*
**begin**

**lemma** *gcd-code-integer* [*code*]: $gcd\ k\ l = |if\ l = (0{::}integer)\ then\ k\ else\ gcd\ l\ (|k|\ mod\ |l|)|$
⟨*proof*⟩

**lemma** *lcm-code-integer* [*code*]: *lcm a b* = |*a*| ∗ |*b*| *div gcd a b*
  **for** *a b* :: *integer*
  ⟨*proof*⟩

**end**

**code-printing**
  **constant** *gcd* :: *integer* ⇒ - ⇀
    (*OCaml*) !(*fun k l −> if Z.equal k Z.zero then/ Z.abs l else if Z.equal/ l Z.zero then Z.abs k else Z.gcd k l*)
  **and** (*Haskell*) *Prelude.gcd*
  **and** (*Scala*) *-.gcd′*((-)′)
  — There is no gcd operation in the SML standard library, so no code setup for SML

Some code equations

**lemmas** *Gcd-nat-set-eq-fold* [*code*] = *Gcd-set-eq-fold* [**where** *?′a* = *nat*]
**lemmas** *Lcm-nat-set-eq-fold* [*code*] = *Lcm-set-eq-fold* [**where** *?′a* = *nat*]
**lemmas** *Gcd-int-set-eq-fold* [*code*] = *Gcd-set-eq-fold* [**where** *?′a* = *int*]
**lemmas** *Lcm-int-set-eq-fold* [*code*] = *Lcm-set-eq-fold* [**where** *?′a* = *int*]

Fact aliases.

**lemma** *lcm-0-iff-nat* [*simp*]: *lcm m n* = *0* ⟷ *m* = *0* ∨ *n* = *0*
  **for** *m n* :: *nat*
  ⟨*proof*⟩

**lemma** *lcm-0-iff-int* [*simp*]: *lcm m n* = *0* ⟷ *m* = *0* ∨ *n* = *0*
  **for** *m n* :: *int*
  ⟨*proof*⟩

**lemma** *dvd-lcm-I1-nat* [*simp*]: *k dvd m* ⟹ *k dvd lcm m n*
  **for** *k m n* :: *nat*
  ⟨*proof*⟩

**lemma** *dvd-lcm-I2-nat* [*simp*]: *k dvd n* ⟹ *k dvd lcm m n*
  **for** *k m n* :: *nat*
  ⟨*proof*⟩

**lemma** *dvd-lcm-I1-int* [*simp*]: *i dvd m* ⟹ *i dvd lcm m n*
  **for** *i m n* :: *int*
  ⟨*proof*⟩

**lemma** *dvd-lcm-I2-int* [*simp*]: *i dvd n* ⟹ *i dvd lcm m n*
  **for** *i m n* :: *int*
  ⟨*proof*⟩

**lemmas** *Gcd-dvd-nat* [*simp*] = *Gcd-dvd* [**where** *?′a* = *nat*]
**lemmas** *Gcd-dvd-int* [*simp*] = *Gcd-dvd* [**where** *?′a* = *int*]

**lemmas** *Gcd-greatest-nat* [*simp*] = *Gcd-greatest* [**where** *?'a* = *nat*]
**lemmas** *Gcd-greatest-int* [*simp*] = *Gcd-greatest* [**where** *?'a* = *int*]

**lemma** *dvd-Lcm-int* [*simp*]: $m \in M \Longrightarrow m\ dvd\ Lcm\ M$
  **for** $M$ :: *int set*
  ⟨*proof*⟩

**lemma** *gcd-neg-numeral-1-int* [*simp*]: $gcd\ (-\ numeral\ n :: int)\ x = gcd\ (numeral\ n)\ x$
  ⟨*proof*⟩

**lemma** *gcd-neg-numeral-2-int* [*simp*]: $gcd\ x\ (-\ numeral\ n :: int) = gcd\ x\ (numeral\ n)$
  ⟨*proof*⟩

**lemma** *gcd-proj1-if-dvd-nat* [*simp*]: $x\ dvd\ y \Longrightarrow gcd\ x\ y = x$
  **for** $x\ y$ :: *nat*
  ⟨*proof*⟩

**lemma** *gcd-proj2-if-dvd-nat* [*simp*]: $y\ dvd\ x \Longrightarrow gcd\ x\ y = y$
  **for** $x\ y$ :: *nat*
  ⟨*proof*⟩

**lemma** *Gcd-in*:
  **fixes** $A$ :: *nat set*
  **assumes** $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow gcd\ a\ b \in A$
  **assumes** $A \neq \{\}$
  **shows** $Gcd\ A \in A$
⟨*proof*⟩

**lemma** *bezout-gcd-nat'*:
  **fixes** $a\ b$ :: *nat*
  **shows** $\exists x\ y.\ b * y \leq a * x \wedge a * x - b * y = gcd\ a\ b \vee a * y \leq b * x \wedge b * x - a * y = gcd\ a\ b$
  ⟨*proof*⟩

**lemmas** *Lcm-eq-0-I-nat* [*simp*] = *Lcm-eq-0-I* [**where** *?'a* = *nat*]
**lemmas** *Lcm-0-iff-nat* [*simp*] = *Lcm-0-iff* [**where** *?'a* = *nat*]
**lemmas** *Lcm-least-int* [*simp*] = *Lcm-least* [**where** *?'a* = *int*]

## 87.11 Characteristic of a semiring

**definition** (**in** *semiring-1*) *semiring-char* :: *'a itself* ⇒ *nat*
  **where** *semiring-char* - = *Gcd* {*n. of-nat n* = (*0* :: *'a*)}

**syntax** *-type-char* :: *type* => *nat* (‹(‹*indent=1 notation=*‹*mixfix CHAR*››*CHAR*/(*1*'(-')))›)
**syntax-consts** *-type-char* ⇌ *semiring-char*
**translations** *CHAR*('*t*) ⇀ *CONST semiring-char* (*CONST Pure.type* :: '*t itself*)
⟨*ML*⟩

**context** *semiring-1*
**begin**

**lemma** *of-nat-CHAR* [*simp*]: $\mathit{of\text{-}nat}\ \mathit{CHAR}('a) = (0 :: 'a)$
⟨*proof*⟩

**lemma** *of-nat-eq-0-iff-char-dvd*: $\mathit{of\text{-}nat}\ n = (0 :: 'a) \longleftrightarrow \mathit{CHAR}('a)\ \mathit{dvd}\ n$
⟨*proof*⟩

**lemma** *CHAR-eqI*:
  **assumes** $\mathit{of\text{-}nat}\ c = (0 :: 'a)$
  **assumes** $\bigwedge x.\ \mathit{of\text{-}nat}\ x = (0 :: 'a) \Longrightarrow c\ \mathit{dvd}\ x$
  **shows** $\mathit{CHAR}('a) = c$
  ⟨*proof*⟩

**lemma** *CHAR-eq0-iff*: $\mathit{CHAR}('a) = 0 \longleftrightarrow (\forall\, n>0.\ \mathit{of\text{-}nat}\ n \neq (0::'a))$
  ⟨*proof*⟩

**lemma** *CHAR-pos-iff*: $\mathit{CHAR}('a) > 0 \longleftrightarrow (\exists\, n>0.\ \mathit{of\text{-}nat}\ n = (0::'a))$
  ⟨*proof*⟩

**lemma** *CHAR-eq-posI*:
  **assumes** $c > 0$ $\mathit{of\text{-}nat}\ c = (0 :: 'a)$ $\bigwedge x.\ x > 0 \Longrightarrow x < c \Longrightarrow \mathit{of\text{-}nat}\ x \neq (0 :: 'a)$
  **shows** $\mathit{CHAR}('a) = c$
⟨*proof*⟩

**end**

**lemma** (**in** *semiring-char-0*) *CHAR-eq-0* [*simp*]: $\mathit{CHAR}('a) = 0$
  ⟨*proof*⟩

**lemma** *CHAR-not-1* [*simp*]: $\mathit{CHAR}('a :: \{\mathit{semiring\text{-}1},\ \mathit{zero\text{-}neq\text{-}one}\}) \neq \mathit{Suc}\ 0$
  ⟨*proof*⟩

**lemma** (**in** *idom*) *CHAR-not-1'* [*simp*]: $\mathit{CHAR}('a) \neq \mathit{Suc}\ 0$
  ⟨*proof*⟩

**lemma** (**in** *ring-1*) *uminus-CHAR-2*:
  **assumes** $\mathit{CHAR}('a) = 2$
  **shows** $-(x :: 'a) = x$
⟨*proof*⟩

**lemma** (**in** *ring-1*) *minus-CHAR-2*:
  **assumes** $\mathit{CHAR}('a) = 2$
  **shows** $(x - y :: 'a) = x + y$
⟨*proof*⟩

**lemma** (**in** *semiring-1-cancel*) *of-nat-eq-iff-char-dvd*:
**assumes** $m < n$
**shows** $\mathit{of\text{-}nat}\ m = (\mathit{of\text{-}nat}\ n :: 'a) \longleftrightarrow \mathit{CHAR}('a)\ \mathit{dvd}\ (n - m)$
⟨*proof*⟩

**lemma** (**in** *ring-1*) *of-int-eq-0-iff-char-dvd*:
$(\mathit{of\text{-}int}\ n = (0 :: 'a)) = (\mathit{int}\ \mathit{CHAR}('a)\ \mathit{dvd}\ n)$
⟨*proof*⟩

**lemma** (**in** *semiring-1-cancel*) *finite-imp-CHAR-pos*:
**assumes** *finite* (*UNIV* :: *'a set*)
**shows** $\mathit{CHAR}('a) > 0$
⟨*proof*⟩

**end**

# 88 Nitpick: Yet Another Counterexample Generator for Isabelle/HOL

**theory** *Nitpick*
**imports** *Record GCD*
**keywords**
*nitpick* :: *diag* **and**
*nitpick-params* :: *thy-decl*
**begin**

**datatype** (*plugins only*: *extraction*) (*dead 'a*, *dead 'b*) *fun-box* = *FunBox* $'a \Rightarrow 'b$
**datatype** (*plugins only*: *extraction*) (*dead 'a*, *dead 'b*) *pair-box* = *PairBox 'a 'b*
**datatype** (*plugins only*: *extraction*) (*dead 'a*) *word* = *Word 'a set*

**typedecl** *bisim-iterator*
**typedecl** *unsigned-bit*
**typedecl** *signed-bit*

**consts**
*unknown* :: *'a*
*is-unknown* :: $'a \Rightarrow \mathit{bool}$
*bisim* :: $\mathit{bisim\text{-}iterator} \Rightarrow 'a \Rightarrow 'a \Rightarrow \mathit{bool}$
*bisim-iterator-max* :: *bisim-iterator*
*Quot* :: $'a \Rightarrow 'b$
*safe-The* :: $('a \Rightarrow \mathit{bool}) \Rightarrow 'a$

Alternative definitions.

**lemma** *Ex1-unfold*[*nitpick-unfold*]: $\mathit{Ex1}\ P \equiv \exists x.\ \{x.\ P\ x\} = \{x\}$
⟨*proof*⟩

**lemma** *rtrancl-unfold*[*nitpick-unfold*]: $r^* \equiv (r^+)^=$

$\langle proof \rangle$

**lemma** *rtranclp-unfold*[*nitpick-unfold*]: $rtranclp\ r\ a\ b \equiv (a = b \lor tranclp\ r\ a\ b)$
$\langle proof \rangle$

**lemma** *tranclp-unfold*[*nitpick-unfold*]:
$tranclp\ r\ a\ b \equiv (a,\ b) \in trancl\ \{(x,\ y).\ r\ x\ y\}$
$\langle proof \rangle$

**lemma** [*nitpick-simp*]:
$of\text{-}nat\ n = (if\ n = 0\ then\ 0\ else\ 1 + of\text{-}nat\ (n - 1))$
$\langle proof \rangle$

**definition** $prod :: {'a}\ set \Rightarrow {'b}\ set \Rightarrow ({'a} \times {'b})\ set$ **where**
$prod\ A\ B = \{(a,\ b).\ a \in A \land b \in B\}$

**definition** $refl' :: ({'a} \times {'a})\ set \Rightarrow bool$ **where**
$refl'\ r \equiv \forall x.\ (x,\ x) \in r$

**definition** $wf' :: ({'a} \times {'a})\ set \Rightarrow bool$ **where**
$wf'\ r \equiv acyclic\ r \land (finite\ r \lor unknown)$

**definition** $card' :: {'a}\ set \Rightarrow nat$ **where**
$card'\ A \equiv if\ finite\ A\ then\ length\ (SOME\ xs.\ set\ xs = A \land distinct\ xs)\ else\ 0$

**definition** $sum' :: ({'a} \Rightarrow {'b}{::}comm\text{-}monoid\text{-}add) \Rightarrow {'a}\ set \Rightarrow {'b}$ **where**
$sum'\ f\ A \equiv if\ finite\ A\ then\ sum\text{-}list\ (map\ f\ (SOME\ xs.\ set\ xs = A \land distinct\ xs))$
$else\ 0$

**inductive** $fold\text{-}graph' :: ({'a} \Rightarrow {'b} \Rightarrow {'b}) \Rightarrow {'b} \Rightarrow {'a}\ set \Rightarrow {'b} \Rightarrow bool$ **where**
$fold\text{-}graph'\ f\ z\ \{\}\ z\ |$
$[\![x \in A;\ fold\text{-}graph'\ f\ z\ (A - \{x\})\ y]\!] \Longrightarrow fold\text{-}graph'\ f\ z\ A\ (f\ x\ y)$

The following lemmas are not strictly necessary but they help the *specialize* optimization.

**lemma** *The-psimp*[*nitpick-psimp*]: $P = (=)\ x \Longrightarrow The\ P = x$
$\langle proof \rangle$

**lemma** *Eps-psimp*[*nitpick-psimp*]:
$[\![P\ x;\ \neg\ P\ y;\ Eps\ P = y]\!] \Longrightarrow Eps\ P = x$
$\langle proof \rangle$

**lemma** *case-unit-unfold*[*nitpick-unfold*]:
$case\text{-}unit\ x\ u \equiv x$
$\langle proof \rangle$

**declare** *unit.case*[*nitpick-simp del*]

**lemma** *case-nat-unfold*[*nitpick-unfold*]:

$case\text{-}nat\ x\ f\ n \equiv if\ n = 0\ then\ x\ else\ f\ (n - 1)$
⟨proof⟩

**declare** $nat.case[nitpick\text{-}simp\ del]$

**lemma** $size\text{-}list\text{-}simp[nitpick\text{-}simp]$:
$size\text{-}list\ f\ xs = (if\ xs = []\ then\ 0\ else\ Suc\ (f\ (hd\ xs) + size\text{-}list\ f\ (tl\ xs)))$
$size\ xs = (if\ xs = []\ then\ 0\ else\ Suc\ (size\ (tl\ xs)))$
⟨proof⟩

Auxiliary definitions used to provide an alternative representation for *rat* and *real*.

**fun** $nat\text{-}gcd :: nat \Rightarrow nat \Rightarrow nat$ **where**
$nat\text{-}gcd\ x\ y = (if\ y = 0\ then\ x\ else\ nat\text{-}gcd\ y\ (x\ mod\ y))$

**declare** $nat\text{-}gcd.simps\ [simp\ del]$

**definition** $nat\text{-}lcm :: nat \Rightarrow nat \Rightarrow nat$ **where**
$nat\text{-}lcm\ x\ y = x * y\ div\ (nat\text{-}gcd\ x\ y)$

**lemma** $gcd\text{-}eq\text{-}nitpick\text{-}gcd\ [nitpick\text{-}unfold]$:
$gcd\ x\ y = Nitpick.nat\text{-}gcd\ x\ y$
⟨proof⟩

**lemma** $lcm\text{-}eq\text{-}nitpick\text{-}lcm\ [nitpick\text{-}unfold]$:
$lcm\ x\ y = Nitpick.nat\text{-}lcm\ x\ y$
⟨proof⟩

**definition** $Frac :: int \times int \Rightarrow bool$ **where**
$Frac \equiv \lambda(a, b).\ b > 0 \wedge coprime\ a\ b$

**consts**
$Abs\text{-}Frac :: int \times int \Rightarrow 'a$
$Rep\text{-}Frac :: 'a \Rightarrow int \times int$

**definition** $zero\text{-}frac :: 'a$ **where**
$zero\text{-}frac \equiv Abs\text{-}Frac\ (0, 1)$

**definition** $one\text{-}frac :: 'a$ **where**
$one\text{-}frac \equiv Abs\text{-}Frac\ (1, 1)$

**definition** $num :: 'a \Rightarrow int$ **where**
$num \equiv fst \circ Rep\text{-}Frac$

**definition** $denom :: 'a \Rightarrow int$ **where**
$denom \equiv snd \circ Rep\text{-}Frac$

**function** $norm\text{-}frac :: int \Rightarrow int \Rightarrow int \times int$ **where**
$norm\text{-}frac\ a\ b =$

 ($if\ b < 0\ then\ norm\text{-}frac\ (-\ a)\ (-\ b)$
 $else\ if\ a = 0 \lor b = 0\ then\ (0,\ 1)$
 $else\ let\ c = gcd\ a\ b\ in\ (a\ div\ c,\ b\ div\ c))$
 ⟨*proof*⟩
 **termination** ⟨*proof*⟩

**declare** *norm-frac.simps*[*simp del*]

**definition** $frac :: int \Rightarrow int \Rightarrow 'a$ **where**
 $frac\ a\ b \equiv Abs\text{-}Frac\ (norm\text{-}frac\ a\ b)$

**definition** $plus\text{-}frac :: 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 [*nitpick-simp*]: $plus\text{-}frac\ q\ r = (let\ d = lcm\ (denom\ q)\ (denom\ r)\ in$
 $frac\ (num\ q * (d\ div\ denom\ q) + num\ r * (d\ div\ denom\ r))\ d)$

**definition** $times\text{-}frac :: 'a \Rightarrow 'a \Rightarrow 'a$ **where**
 [*nitpick-simp*]: $times\text{-}frac\ q\ r = frac\ (num\ q * num\ r)\ (denom\ q * denom\ r)$

**definition** $uminus\text{-}frac :: 'a \Rightarrow 'a$ **where**
 $uminus\text{-}frac\ q \equiv Abs\text{-}Frac\ (-\ num\ q,\ denom\ q)$

**definition** $number\text{-}of\text{-}frac :: int \Rightarrow 'a$ **where**
 $number\text{-}of\text{-}frac\ n \equiv Abs\text{-}Frac\ (n,\ 1)$

**definition** $inverse\text{-}frac :: 'a \Rightarrow 'a$ **where**
 $inverse\text{-}frac\ q \equiv frac\ (denom\ q)\ (num\ q)$

**definition** $less\text{-}frac :: 'a \Rightarrow 'a \Rightarrow bool$ **where**
 [*nitpick-simp*]: $less\text{-}frac\ q\ r \longleftrightarrow num\ (plus\text{-}frac\ q\ (uminus\text{-}frac\ r)) < 0$

**definition** $less\text{-}eq\text{-}frac :: 'a \Rightarrow 'a \Rightarrow bool$ **where**
 [*nitpick-simp*]: $less\text{-}eq\text{-}frac\ q\ r \longleftrightarrow num\ (plus\text{-}frac\ q\ (uminus\text{-}frac\ r)) \leq 0$

**definition** $of\text{-}frac :: 'a \Rightarrow 'b::\{inverse,ring\text{-}1\}$ **where**
 $of\text{-}frac\ q \equiv of\text{-}int\ (num\ q)\ /\ of\text{-}int\ (denom\ q)$

**axiomatization** $wf\text{-}wfrec :: ('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$

**definition** $wf\text{-}wfrec' :: ('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **where**
 [*nitpick-simp*]: $wf\text{-}wfrec'\ R\ F\ x = F\ (cut\ (wf\text{-}wfrec\ R\ F)\ R\ x)\ x$

**definition** $wfrec' :: ('a \times 'a)\ set \Rightarrow (('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b$ **where**
 $wfrec'\ R\ F\ x \equiv if\ wf\ R\ then\ wf\text{-}wfrec'\ R\ F\ x\ else\ THE\ y.\ wfrec\text{-}rel\ R\ (\lambda f\ x.\ F\ (cut\ f\ R\ x)\ x)\ x\ y$

⟨*ML*⟩

**hide-const** (**open**) *unknown is-unknown bisim bisim-iterator-max Quot safe-The FunBox PairBox Word prod*

*refl′ wf′ card′ sum′ fold-graph′ nat-gcd nat-lcm Frac Abs-Frac Rep-Frac*
*zero-frac one-frac num denom norm-frac frac plus-frac times-frac uminus-frac number-of-frac*
*inverse-frac less-frac less-eq-frac of-frac wf-wfrec wf-wfrec wfrec′*

**hide-type** (**open**) *bisim-iterator fun-box pair-box unsigned-bit signed-bit word*

**hide-fact** (**open**) *Ex1-unfold rtrancl-unfold rtranclp-unfold tranclp-unfold prod-def refl′-def wf′-def*
*card′-def sum′-def The-psimp Eps-psimp case-unit-unfold case-nat-unfold*
*size-list-simp nat-lcm-def Frac-def zero-frac-def one-frac-def*
*num-def denom-def frac-def plus-frac-def times-frac-def uminus-frac-def*
*number-of-frac-def inverse-frac-def less-frac-def less-eq-frac-def of-frac-def wf-wfrec′-def*
*wfrec′-def*

**end**

**theory** *Nunchaku*
**imports** *Nitpick*
**keywords**
*nunchaku* :: *diag* **and**
*nunchaku-params* :: *thy-decl*
**begin**

**consts** *unreachable* :: $'a$

**definition** *The-unsafe* :: $('a \Rightarrow bool) \Rightarrow 'a$ **where**
$The\text{-}unsafe = The$

**definition** *rmember* :: $'a\ set \Rightarrow 'a \Rightarrow bool$ **where**
$rmember\ A\ x \longleftrightarrow x \in A$

⟨*ML*⟩

**hide-const** (**open**) *unreachable The-unsafe rmember*

**end**

# 89 Greatest Fixpoint (Codatatype) Operation on Bounded Natural Functors

**theory** *BNF-Greatest-Fixpoint*
**imports** *BNF-Fixpoint-Base String*
**keywords**
*codatatype* :: *thy-defn* **and**
*primcorecursive* :: *thy-goal-defn* **and**
*primcorec* :: *thy-defn*

**begin**

**alias** *proj* = *Equiv-Relations.proj*

**lemma** *one-pointE*: $[\![\bigwedge x.\ s = x \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *obj-sumE*: $[\![\forall x.\ s = Inl\ x \longrightarrow P; \forall x.\ s = Inr\ x \longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *not-TrueE*: $\neg\ True \Longrightarrow P$
⟨*proof*⟩

**lemma** *neq-eq-eq-contradict*: $[\![t \neq u;\ s = t;\ s = u]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *converse-Times*: $(A \times B)^{-1} = B \times A$
⟨*proof*⟩

**lemma** *equiv-proj*:
**assumes** *e*: *equiv A R* **and** *m*: $z \in R$
**shows** $(proj\ R \circ fst)\ z = (proj\ R \circ snd)\ z$
⟨*proof*⟩

**definition** *image2* **where** $image2\ A\ f\ g = \{(f\ a,\ g\ a) \mid a.\ a \in A\}$

**lemma** *Id-on-Gr*: *Id-on A = Gr A id*
⟨*proof*⟩

**lemma** *image2-eqI*: $[\![b = f\ x;\ c = g\ x;\ x \in A]\!] \Longrightarrow (b,\ c) \in image2\ A\ f\ g$
⟨*proof*⟩

**lemma** *IdD*: $(a,\ b) \in Id \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *image2-Gr*: $image2\ A\ f\ g = (Gr\ A\ f)^{-1}\ O\ (Gr\ A\ g)$
⟨*proof*⟩

**lemma** *GrD1*: $(x,\ fx) \in Gr\ A\ f \Longrightarrow x \in A$
⟨*proof*⟩

**lemma** *GrD2*: $(x,\ fx) \in Gr\ A\ f \Longrightarrow f\ x = fx$
⟨*proof*⟩

**lemma** *Gr-incl*: $Gr\ A\ f \subseteq A \times B \longleftrightarrow f\ `\ A \subseteq B$
⟨*proof*⟩

**lemma** *subset-Collect-iff*: $B \subseteq A \Longrightarrow (B \subseteq \{x \in A.\ P\ x\}) = (\forall x \in B.\ P\ x)$

⟨*proof*⟩

**lemma** *subset-CollectI*: $B \subseteq A \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow Q\ x \Longrightarrow P\ x) \Longrightarrow (\{x \in B.\ Q\ x\} \subseteq \{x \in A.\ P\ x\})$
⟨*proof*⟩

**lemma** *in-rel-Collect-case-prod-eq*: *in-rel* (*Collect* (*case-prod* $X$)) $= X$
⟨*proof*⟩

**lemma** *Collect-case-prod-in-rel-leI*: $X \subseteq Y \Longrightarrow X \subseteq$ *Collect* (*case-prod* (*in-rel* $Y$))
⟨*proof*⟩

**lemma** *Collect-case-prod-in-rel-leE*: $X \subseteq$ *Collect* (*case-prod* (*in-rel* $Y$)) $\Longrightarrow (X \subseteq Y \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

**lemma** *conversep-in-rel*: (*in-rel* $R)^{-1-1}$ = *in-rel* $(R^{-1})$
⟨*proof*⟩

**lemma** *relcompp-in-rel*: *in-rel* $R$ *OO* *in-rel* $S$ = *in-rel* ($R$ $O$ $S$)
⟨*proof*⟩

**lemma** *in-rel-Gr*: *in-rel* (*Gr* $A$ $f$) = *Grp* $A$ $f$
⟨*proof*⟩

**definition** *relImage* **where**
*relImage* $R\ f \equiv \{(f\ a1,\ f\ a2) \mid a1\ a2.\ (a1,a2) \in R\}$

**definition** *relInvImage* **where**
*relInvImage* $A\ R\ f \equiv \{(a1,\ a2) \mid a1\ a2.\ a1 \in A \wedge a2 \in A \wedge (f\ a1,\ f\ a2) \in R\}$

**lemma** *relImage-Gr*:
$[\![R \subseteq A \times A]\!] \Longrightarrow$ *relImage* $R\ f = ($*Gr* $A\ f)^{-1}\ O\ R\ O$ *Gr* $A\ f$
⟨*proof*⟩

**lemma** *relInvImage-Gr*: $[\![R \subseteq B \times B]\!] \Longrightarrow$ *relInvImage* $A\ R\ f =$ *Gr* $A\ f\ O\ R\ O$ $($*Gr* $A\ f)^{-1}$
⟨*proof*⟩

**lemma** *relImage-mono*:
$R1 \subseteq R2 \Longrightarrow$ *relImage* $R1\ f \subseteq$ *relImage* $R2\ f$
⟨*proof*⟩

**lemma** *relInvImage-mono*:
$R1 \subseteq R2 \Longrightarrow$ *relInvImage* $A\ R1\ f \subseteq$ *relInvImage* $A\ R2\ f$
⟨*proof*⟩

**lemma** *relInvImage-Id-on*:

$(\bigwedge a1\ a2.\ f\ a1 = f\ a2 \longleftrightarrow a1 = a2) \Longrightarrow relInvImage\ A\ (Id\text{-}on\ B)\ f \subseteq Id$
⟨proof⟩

**lemma** *relInvImage-UNIV-relImage*:
$R \subseteq relInvImage\ UNIV\ (relImage\ R\ f)\ f$
⟨proof⟩

**lemma** *relImage-proj*:
**assumes** *equiv A R*
**shows** $relImage\ R\ (proj\ R) \subseteq Id\text{-}on\ (A//R)$
⟨proof⟩

**lemma** *relImage-relInvImage*:
**assumes** $R \subseteq f\ `\ A \times f\ `\ A$
**shows** $relImage\ (relInvImage\ A\ R\ f)\ f = R$
⟨proof⟩

**lemma** *subst-Pair*: $P\ x\ y \Longrightarrow a = (x, y) \Longrightarrow P\ (fst\ a)\ (snd\ a)$
⟨proof⟩

**lemma** *fst-diag-id*: $(fst \circ (\lambda x.\ (x, x)))\ z = id\ z$ ⟨proof⟩
**lemma** *snd-diag-id*: $(snd \circ (\lambda x.\ (x, x)))\ z = id\ z$ ⟨proof⟩

**lemma** *fst-diag-fst*: $fst \circ ((\lambda x.\ (x, x)) \circ fst) = fst$ ⟨proof⟩
**lemma** *snd-diag-fst*: $snd \circ ((\lambda x.\ (x, x)) \circ fst) = fst$ ⟨proof⟩
**lemma** *fst-diag-snd*: $fst \circ ((\lambda x.\ (x, x)) \circ snd) = snd$ ⟨proof⟩
**lemma** *snd-diag-snd*: $snd \circ ((\lambda x.\ (x, x)) \circ snd) = snd$ ⟨proof⟩

**definition** *Succ* **where** $Succ\ Kl\ kl = \{k\ .\ kl\ @\ [k] \in Kl\}$
**definition** *Shift* **where** $Shift\ Kl\ k = \{kl.\ k\ \#\ kl \in Kl\}$
**definition** *shift* **where** $shift\ lab\ k = (\lambda kl.\ lab\ (k\ \#\ kl))$

**lemma** *empty-Shift*: $[\![[] \in Kl;\ k \in Succ\ Kl\ []]\!] \Longrightarrow [] \in Shift\ Kl\ k$
⟨proof⟩

**lemma** *SuccD*: $k \in Succ\ Kl\ kl \Longrightarrow kl\ @\ [k] \in Kl$
⟨proof⟩

**lemmas** *SuccE* = *SuccD*[*elim-format*]

**lemma** *SuccI*: $kl\ @\ [k] \in Kl \Longrightarrow k \in Succ\ Kl\ kl$
⟨proof⟩

**lemma** *ShiftD*: $kl \in Shift\ Kl\ k \Longrightarrow k\ \#\ kl \in Kl$
⟨proof⟩

**lemma** *Succ-Shift*: $Succ\ (Shift\ Kl\ k)\ kl = Succ\ Kl\ (k\ \#\ kl)$
⟨proof⟩

**lemma** *length-Cons*: $length\ (x\ \#\ xs) = Suc\ (length\ xs)$
⟨*proof*⟩

**lemma** *length-append-singleton*: $length\ (xs\ @\ [x]) = Suc\ (length\ xs)$
⟨*proof*⟩

**definition** $toCard\text{-}pred\ A\ r\ f \equiv inj\text{-}on\ f\ A \wedge f\ `\ A \subseteq Field\ r \wedge Card\text{-}order\ r$
**definition** $toCard\ A\ r \equiv SOME\ f.\ toCard\text{-}pred\ A\ r\ f$

**lemma** *ex-toCard-pred*:
$[\![|A| \leq o\ r;\ Card\text{-}order\ r]\!] \Longrightarrow \exists\ f.\ toCard\text{-}pred\ A\ r\ f$
⟨*proof*⟩

**lemma** *toCard-pred-toCard*:
$[\![|A| \leq o\ r;\ Card\text{-}order\ r]\!] \Longrightarrow toCard\text{-}pred\ A\ r\ (toCard\ A\ r)$
⟨*proof*⟩

**lemma** *toCard-inj*: $[\![|A| \leq o\ r;\ Card\text{-}order\ r;\ x \in A;\ y \in A]\!] \Longrightarrow toCard\ A\ r\ x = toCard\ A\ r\ y \longleftrightarrow x = y$
⟨*proof*⟩

**definition** $fromCard\ A\ r\ k \equiv SOME\ b.\ b \in A \wedge toCard\ A\ r\ b = k$

**lemma** *fromCard-toCard*:
$[\![|A| \leq o\ r;\ Card\text{-}order\ r;\ b \in A]\!] \Longrightarrow fromCard\ A\ r\ (toCard\ A\ r\ b) = b$
⟨*proof*⟩

**lemma** *Inl-Field-csum*: $a \in Field\ r \Longrightarrow Inl\ a \in Field\ (r\ +c\ s)$
⟨*proof*⟩

**lemma** *Inr-Field-csum*: $a \in Field\ s \Longrightarrow Inr\ a \in Field\ (r\ +c\ s)$
⟨*proof*⟩

**lemma** *rec-nat-0-imp*: $f = rec\text{-}nat\ f1\ (\lambda n\ rec.\ f2\ n\ rec) \Longrightarrow f\ 0 = f1$
⟨*proof*⟩

**lemma** *rec-nat-Suc-imp*: $f = rec\text{-}nat\ f1\ (\lambda n\ rec.\ f2\ n\ rec) \Longrightarrow f\ (Suc\ n) = f2\ n\ (f\ n)$
⟨*proof*⟩

**lemma** *rec-list-Nil-imp*: $f = rec\text{-}list\ f1\ (\lambda x\ xs\ rec.\ f2\ x\ xs\ rec) \Longrightarrow f\ [] = f1$
⟨*proof*⟩

**lemma** *rec-list-Cons-imp*: $f = rec\text{-}list\ f1\ (\lambda x\ xs\ rec.\ f2\ x\ xs\ rec) \Longrightarrow f\ (x\ \#\ xs) = f2\ x\ xs\ (f\ xs)$
⟨*proof*⟩

**lemma** *not-arg-cong-Inr*: $x \neq y \Longrightarrow Inr\ x \neq Inr\ y$

⟨*proof*⟩

**definition** *image2p* **where**
$image2p\ f\ g\ R = (\lambda x\ y.\ \exists x'\ y'.\ R\ x'\ y' \wedge f\ x' = x \wedge g\ y' = y)$

**lemma** *image2pI*: $R\ x\ y \Longrightarrow image2p\ f\ g\ R\ (f\ x)\ (g\ y)$
⟨*proof*⟩

**lemma** *image2pE*: $[\![image2p\ f\ g\ R\ fx\ gy;\ (\bigwedge x\ y.\ fx = f\ x \Longrightarrow gy = g\ y \Longrightarrow R\ x\ y \Longrightarrow P)]\!] \Longrightarrow P$
⟨*proof*⟩

**lemma** *rel-fun-iff-geq-image2p*: $rel\text{-}fun\ R\ S\ f\ g = (image2p\ f\ g\ R \leq S)$
⟨*proof*⟩

**lemma** *rel-fun-image2p*: $rel\text{-}fun\ R\ (image2p\ f\ g\ R)\ f\ g$
⟨*proof*⟩

## 89.1 Equivalence relations, quotients, and Hilbert's choice

**lemma** *equiv-Eps-in*:
$[\![equiv\ A\ r;\ X \in A//r]\!] \Longrightarrow Eps\ (\lambda x.\ x \in X) \in X$
⟨*proof*⟩

**lemma** *equiv-Eps-preserves*:
**assumes** *ECH*: $equiv\ A\ r$ **and** *X*: $X \in A//r$
**shows** $Eps\ (\lambda x.\ x \in X) \in A$
⟨*proof*⟩

**lemma** *proj-Eps*:
**assumes** $equiv\ A\ r$ **and** $X \in A//r$
**shows** $proj\ r\ (Eps\ (\lambda x.\ x \in X)) = X$
⟨*proof*⟩

**definition** *univ* **where** $univ\ f\ X == f\ (Eps\ (\lambda x.\ x \in X))$

**lemma** *univ-commute*:
**assumes** *ECH*: $equiv\ A\ r$ **and** *RES*: $f\ respects\ r$ **and** *x*: $x \in A$
**shows** $(univ\ f)\ (proj\ r\ x) = f\ x$
⟨*proof*⟩

**lemma** *univ-preserves*:
**assumes** *ECH*: $equiv\ A\ r$ **and** *RES*: $f\ respects\ r$ **and** *PRES*: $\forall x \in A.\ f\ x \in B$
**shows** $\forall X \in A//r.\ univ\ f\ X \in B$
⟨*proof*⟩

**lemma** *card-suc-ordLess-imp-ordLeq*:
**assumes** *ORD*: $Card\text{-}order\ r\ Card\text{-}order\ r'\ card\text{-}order\ r'$
**and** *LESS*: $r <o\ card\text{-}suc\ r'$

**shows** $r \leq o\ r'$
⟨*proof*⟩

**lemma** *natLeq-ordLess-cinfinite*: $[\![$*Cinfinite r*; *card-order r*$]\!] \Longrightarrow$ *natLeq* $<o$ *card-suc r*
⟨*proof*⟩

**corollary** *natLeq-ordLess-cinfinite'*: $[\![$*Cinfinite r'*; *card-order r'*; $r \equiv$ *card-suc r'*$]\!] \Longrightarrow$ *natLeq* $<o\ r$
⟨*proof*⟩

⟨*ML*⟩

**end**

# 90 Filters on predicates

**theory** *Filter*
**imports** *Set-Interval Lifting-Set*
**begin**

## 90.1 Filters

This definition also allows non-proper filters.

**locale** *is-filter* =
**fixes** $F :: ('a \Rightarrow bool) \Rightarrow bool$
**assumes** *True*: $F\ (\lambda x.\ True)$
**assumes** *conj*: $F\ (\lambda x.\ P\ x) \Longrightarrow F\ (\lambda x.\ Q\ x) \Longrightarrow F\ (\lambda x.\ P\ x \wedge Q\ x)$
**assumes** *mono*: $\forall x.\ P\ x \longrightarrow Q\ x \Longrightarrow F\ (\lambda x.\ P\ x) \Longrightarrow F\ (\lambda x.\ Q\ x)$

**typedef** *'a filter* = $\{F :: ('a \Rightarrow bool) \Rightarrow bool.$ *is-filter* $F\}$
⟨*proof*⟩

**lemma** *is-filter-Rep-filter*: *is-filter* (*Rep-filter F*)
⟨*proof*⟩

**lemma** *Abs-filter-inverse'*:
**assumes** *is-filter F* **shows** *Rep-filter* (*Abs-filter F*) = *F*
⟨*proof*⟩

### 90.1.1 Eventually

**definition** *eventually* :: $('a \Rightarrow bool) \Rightarrow$ *'a filter* $\Rightarrow bool$
**where** *eventually P F* $\longleftrightarrow$ *Rep-filter F P*

**syntax**
*-eventually* :: *pttrn* => *'a filter* => *bool* => *bool* (‹(‹indent=3 notation=‹binder $\forall_F$››$\forall_F$ - in -./ -)› [0, 0, 10] 10)

**syntax-consts**
*-eventually* == *eventually*
**translations**
$\forall_F x$ *in* $F$. $P$ == *CONST eventually* $(\lambda x.\ P)\ F$

**lemma** *eventually-Abs-filter*:
**assumes** *is-filter* $F$ **shows** *eventually* $P$ (*Abs-filter* $F$) = $F\ P$
⟨*proof*⟩

**lemma** *filter-eq-iff*:
**shows** $F = F' \longleftrightarrow (\forall P.\ eventually\ P\ F = eventually\ P\ F')$
⟨*proof*⟩

**lemma** *eventually-True* [*simp*]: *eventually* $(\lambda x.\ True)\ F$
⟨*proof*⟩

**lemma** *always-eventually*: $\forall x.\ P\ x \Longrightarrow eventually\ P\ F$
⟨*proof*⟩

**lemma** *eventuallyI*: $(\bigwedge x.\ P\ x) \Longrightarrow eventually\ P\ F$
⟨*proof*⟩

**lemma** *filter-eqI*: $(\bigwedge P.\ eventually\ P\ F \longleftrightarrow eventually\ P\ G) \Longrightarrow F = G$
⟨*proof*⟩

**lemma** *eventually-mono*:
$[\![eventually\ P\ F;\ \bigwedge x.\ P\ x \Longrightarrow Q\ x]\!] \Longrightarrow eventually\ Q\ F$
⟨*proof*⟩

**lemma** *eventually-conj*:
**assumes** $P$: *eventually* $(\lambda x.\ P\ x)\ F$
**assumes** $Q$: *eventually* $(\lambda x.\ Q\ x)\ F$
**shows** *eventually* $(\lambda x.\ P\ x \wedge Q\ x)\ F$
⟨*proof*⟩

**lemma** *eventually-mp*:
**assumes** *eventually* $(\lambda x.\ P\ x \longrightarrow Q\ x)\ F$
**assumes** *eventually* $(\lambda x.\ P\ x)\ F$
**shows** *eventually* $(\lambda x.\ Q\ x)\ F$
⟨*proof*⟩

**lemma** *eventually-rev-mp*:
**assumes** *eventually* $(\lambda x.\ P\ x)\ F$
**assumes** *eventually* $(\lambda x.\ P\ x \longrightarrow Q\ x)\ F$
**shows** *eventually* $(\lambda x.\ Q\ x)\ F$
⟨*proof*⟩

**lemma** *eventually-conj-iff*:
*eventually* $(\lambda x.\ P\ x \wedge Q\ x)\ F \longleftrightarrow eventually\ P\ F \wedge eventually\ Q\ F$

⟨proof⟩

**lemma** *eventually-elim2*:
**assumes** *eventually* $(\lambda i.\ P\ i)\ F$
**assumes** *eventually* $(\lambda i.\ Q\ i)\ F$
**assumes** $\bigwedge i.\ P\ i \Longrightarrow Q\ i \Longrightarrow R\ i$
**shows** *eventually* $(\lambda i.\ R\ i)\ F$
⟨proof⟩

**lemma** *eventually-cong*:
**assumes** *eventually* $P\ F$ **and** $\bigwedge x.\ P\ x \Longrightarrow Q\ x \longleftrightarrow R\ x$
**shows** *eventually* $Q\ F \longleftrightarrow$ *eventually* $R\ F$
⟨proof⟩

**lemma** *eventually-ball-finite-distrib*:
*finite* $A \Longrightarrow$ (*eventually* $(\lambda x.\ \forall y{\in}A.\ P\ x\ y)$ *net*) $\longleftrightarrow$ $(\forall y{\in}A.$ *eventually* $(\lambda x.\ P\ x\ y)$ *net*)
⟨proof⟩

**lemma** *eventually-ball-finite*:
*finite* $A \Longrightarrow \forall y{\in}A.$ *eventually* $(\lambda x.\ P\ x\ y)$ *net* $\Longrightarrow$ *eventually* $(\lambda x.\ \forall y{\in}A.\ P\ x\ y)$ *net*
⟨proof⟩

**lemma** *eventually-all-finite*:
**fixes** $P :: 'a \Rightarrow 'b{::}finite \Rightarrow bool$
**assumes** $\bigwedge y.$ *eventually* $(\lambda x.\ P\ x\ y)$ *net*
**shows** *eventually* $(\lambda x.\ \forall y.\ P\ x\ y)$ *net*
⟨proof⟩

**lemma** *eventually-ex*: $(\forall_F x\ in\ F.\ \exists y.\ P\ x\ y) \longleftrightarrow (\exists Y.\ \forall_F x\ in\ F.\ P\ x\ (Y\ x))$
⟨proof⟩

**lemma** *not-eventually-impI*: *eventually* $P\ F \Longrightarrow \neg$ *eventually* $Q\ F \Longrightarrow \neg$ *eventually* $(\lambda x.\ P\ x \longrightarrow Q\ x)\ F$
⟨proof⟩

**lemma** *not-eventuallyD*: $\neg$ *eventually* $P\ F \Longrightarrow \exists x.\ \neg\ P\ x$
⟨proof⟩

**lemma** *eventually-subst*:
**assumes** *eventually* $(\lambda n.\ P\ n = Q\ n)\ F$
**shows** *eventually* $P\ F =$ *eventually* $Q\ F$ (**is** $?L = ?R$)
⟨proof⟩

## 90.2 Frequently as dual to eventually

**definition** *frequently* :: $('a \Rightarrow bool) \Rightarrow 'a\ filter \Rightarrow bool$
**where** *frequently* $P\ F \longleftrightarrow \neg$ *eventually* $(\lambda x.\ \neg\ P\ x)\ F$

**syntax**
*-frequently* :: *pttrn* ⇒ *'a filter* ⇒ *bool* ⇒ *bool* (‹(‹indent=3 notation=‹binder $\exists_F$››$\exists_F$ - in -./ -)› [0, 0, 10] 10)
**syntax-consts**
*-frequently* == *frequently*
**translations**
$\exists_F x$ *in* $F$. $P$ == *CONST frequently* ($\lambda x$. $P$) $F$

**lemma** *not-frequently-False* [*simp*]: ¬ ($\exists_F x$ *in* $F$. *False*)
⟨*proof*⟩

**lemma** *frequently-ex*: $\exists_F x$ *in* $F$. $P\ x \Longrightarrow \exists x.\ P\ x$
⟨*proof*⟩

**lemma** *frequentlyE*: **assumes** *frequently* $P\ F$ **obtains** $x$ **where** $P\ x$
⟨*proof*⟩

**lemma** *frequently-mp*:
**assumes** *ev*: $\forall_F x$ *in* $F$. $P\ x \longrightarrow Q\ x$ **and** $P$: $\exists_F x$ *in* $F$. $P\ x$ **shows** $\exists_F x$ *in* $F$. $Q\ x$
⟨*proof*⟩

**lemma** *frequently-rev-mp*:
**assumes** $\exists_F x$ *in* $F$. $P\ x$
**assumes** $\forall_F x$ *in* $F$. $P\ x \longrightarrow Q\ x$
**shows** $\exists_F x$ *in* $F$. $Q\ x$
⟨*proof*⟩

**lemma** *frequently-mono*: $(\forall x.\ P\ x \longrightarrow Q\ x) \Longrightarrow$ *frequently* $P\ F \Longrightarrow$ *frequently* $Q\ F$
⟨*proof*⟩

**lemma** *frequently-elim1*: $\exists_F x$ *in* $F$. $P\ x \Longrightarrow (\bigwedge i.\ P\ i \Longrightarrow Q\ i) \Longrightarrow \exists_F x$ *in* $F$. $Q\ x$
⟨*proof*⟩

**lemma** *frequently-disj-iff*: ($\exists_F x$ *in* $F$. $P\ x \lor Q\ x$) $\longleftrightarrow$ ($\exists_F x$ *in* $F$. $P\ x$) $\lor$ ($\exists_F x$ *in* $F$. $Q\ x$)
⟨*proof*⟩

**lemma** *frequently-disj*: $\exists_F x$ *in* $F$. $P\ x \Longrightarrow \exists_F x$ *in* $F$. $Q\ x \Longrightarrow \exists_F x$ *in* $F$. $P\ x \lor Q\ x$
⟨*proof*⟩

**lemma** *frequently-bex-finite-distrib*:
**assumes** *finite* $A$ **shows** ($\exists_F x$ *in* $F$. $\exists y{\in}A$. $P\ x\ y$) $\longleftrightarrow$ ($\exists y{\in}A$. $\exists_F x$ *in* $F$. $P\ x\ y$)
⟨*proof*⟩

**lemma** *frequently-bex-finite*: *finite* $A \Longrightarrow \exists_F x$ *in* $F.\ \exists y{\in}A.\ P\ x\ y \Longrightarrow \exists y{\in}A.\ \exists_F x$ *in* $F.\ P\ x\ y$
⟨*proof*⟩

**lemma** *frequently-all*: $(\exists_F x$ *in* $F.\ \forall y.\ P\ x\ y) \longleftrightarrow (\forall Y.\ \exists_F x$ *in* $F.\ P\ x\ (Y\ x))$
⟨*proof*⟩

**lemma**
**shows** *not-eventually*: $\neg$ *eventually* $P\ F \longleftrightarrow (\exists_F x$ *in* $F.\ \neg\ P\ x)$
**and** *not-frequently*: $\neg$ *frequently* $P\ F \longleftrightarrow (\forall_F x$ *in* $F.\ \neg\ P\ x)$
⟨*proof*⟩

**lemma** *frequently-imp-iff*:
$(\exists_F x$ *in* $F.\ P\ x \longrightarrow Q\ x) \longleftrightarrow$ (*eventually* $P\ F \longrightarrow$ *frequently* $Q\ F$)
⟨*proof*⟩

**lemma** *frequently-eventually-conj*:
**assumes** $\exists_F x$ *in* $F.\ P\ x$
**assumes** $\forall_F x$ *in* $F.\ Q\ x$
**shows** $\exists_F x$ *in* $F.\ Q\ x \wedge P\ x$
⟨*proof*⟩

**lemma** *frequently-cong*:
**assumes** *ev*: *eventually* $P\ F$ **and** *QR*: $\bigwedge x.\ P\ x \Longrightarrow Q\ x \longleftrightarrow R\ x$
**shows** *frequently* $Q\ F \longleftrightarrow$ *frequently* $R\ F$
⟨*proof*⟩

**lemma** *frequently-eventually-frequently*:
*frequently* $P\ F \Longrightarrow$ *eventually* $Q\ F \Longrightarrow$ *frequently* $(\lambda x.\ P\ x \wedge Q\ x)\ F$
⟨*proof*⟩

**lemma** *eventually-frequently-const-simps* [*simp*]:
$(\exists_F x$ *in* $F.\ P\ x \wedge C) \longleftrightarrow (\exists_F x$ *in* $F.\ P\ x) \wedge C$
$(\exists_F x$ *in* $F.\ C \wedge P\ x) \longleftrightarrow C \wedge (\exists_F x$ *in* $F.\ P\ x)$
$(\forall_F x$ *in* $F.\ P\ x \vee C) \longleftrightarrow (\forall_F x$ *in* $F.\ P\ x) \vee C$
$(\forall_F x$ *in* $F.\ C \vee P\ x) \longleftrightarrow C \vee (\forall_F x$ *in* $F.\ P\ x)$
$(\forall_F x$ *in* $F.\ P\ x \longrightarrow C) \longleftrightarrow ((\exists_F x$ *in* $F.\ P\ x) \longrightarrow C)$
$(\forall_F x$ *in* $F.\ C \longrightarrow P\ x) \longleftrightarrow (C \longrightarrow (\forall_F x$ *in* $F.\ P\ x))$
⟨*proof*⟩

**lemmas** *eventually-frequently-simps* =
*eventually-frequently-const-simps*
*not-eventually*
*eventually-conj-iff*
*eventually-ball-finite-distrib*
*eventually-ex*
*not-frequently*
*frequently-disj-iff*

*frequently-bex-finite-distrib*
*frequently-all*
*frequently-imp-iff*

⟨*ML*⟩

### 90.2.1 Finer-than relation

$F \leq F'$ means that filter $F$ is finer than filter $F'$.

**instantiation** *filter* :: (*type*) *complete-lattice*
**begin**

**definition** *le-filter-def*:
$F \leq F' \longleftrightarrow (\forall P.\ eventually\ P\ F' \longrightarrow eventually\ P\ F)$

**definition**
$(F :: 'a\ filter) < F' \longleftrightarrow F \leq F' \wedge \neg\ F' \leq F$

**definition**
$top = Abs\text{-}filter\ (\lambda P.\ \forall x.\ P\ x)$

**definition**
$bot = Abs\text{-}filter\ (\lambda P.\ True)$

**definition**
$sup\ F\ F' = Abs\text{-}filter\ (\lambda P.\ eventually\ P\ F \wedge eventually\ P\ F')$

**definition**
$inf\ F\ F' = Abs\text{-}filter$
$(\lambda P.\ \exists\ Q\ R.\ eventually\ Q\ F \wedge eventually\ R\ F' \wedge (\forall x.\ Q\ x \wedge R\ x \longrightarrow P\ x))$

**definition**
$Sup\ S = Abs\text{-}filter\ (\lambda P.\ \forall F \in S.\ eventually\ P\ F)$

**definition**
$Inf\ S = Sup\ \{F::'a\ filter.\ \forall F' \in S.\ F \leq F'\}$

**lemma** *eventually-top* [*simp*]: $eventually\ P\ top \longleftrightarrow (\forall x.\ P\ x)$
⟨*proof*⟩

**lemma** *eventually-bot* [*simp*]: $eventually\ P\ bot$
⟨*proof*⟩

**lemma** *eventually-sup*:
$eventually\ P\ (sup\ F\ F') \longleftrightarrow eventually\ P\ F \wedge eventually\ P\ F'$
⟨*proof*⟩

**lemma** *eventually-inf*:
$eventually\ P\ (inf\ F\ F') \longleftrightarrow$

$(\exists\, Q\ R.\ eventually\ Q\ F \wedge eventually\ R\ F' \wedge (\forall\, x.\ Q\ x \wedge R\ x \longrightarrow P\ x))$
⟨*proof*⟩

**lemma** *eventually-Sup*:
$eventually\ P\ (Sup\ S) \longleftrightarrow (\forall\, F{\in}S.\ eventually\ P\ F)$
⟨*proof*⟩

**instance** ⟨*proof*⟩

**end**

**instance** *filter* :: (*type*) *distrib-lattice*
⟨*proof*⟩

**lemma** *filter-leD*:
$F \le F' \Longrightarrow eventually\ P\ F' \Longrightarrow eventually\ P\ F$
⟨*proof*⟩

**lemma** *filter-leI*:
$(\bigwedge P.\ eventually\ P\ F' \Longrightarrow eventually\ P\ F) \Longrightarrow F \le F'$
⟨*proof*⟩

**lemma** *eventually-False*:
$eventually\ (\lambda x.\ False)\ F \longleftrightarrow F = bot$
⟨*proof*⟩

**lemma** *eventually-frequently*: $F \neq bot \Longrightarrow eventually\ P\ F \Longrightarrow frequently\ P\ F$
⟨*proof*⟩

**lemma** *eventually-frequentlyE*:
**assumes** $eventually\ P\ F$
**assumes** $eventually\ (\lambda x.\ \neg\ P\ x \vee Q\ x)\ F\ F{\neq}bot$
**shows** $frequently\ Q\ F$
⟨*proof*⟩

**lemma** *eventually-const-iff*: $eventually\ (\lambda x.\ P)\ F \longleftrightarrow P \vee F = bot$
⟨*proof*⟩

**lemma** *eventually-const*[*simp*]: $F \neq bot \Longrightarrow eventually\ (\lambda x.\ P)\ F \longleftrightarrow P$
⟨*proof*⟩

**lemma** *frequently-const-iff*: $frequently\ (\lambda x.\ P)\ F \longleftrightarrow P \wedge F \neq bot$
⟨*proof*⟩

**lemma** *frequently-const*[*simp*]: $F \neq bot \Longrightarrow frequently\ (\lambda x.\ P)\ F \longleftrightarrow P$
⟨*proof*⟩

**lemma** *eventually-happens*: $eventually\ P\ net \Longrightarrow net = bot \vee (\exists\, x.\ P\ x)$

⟨*proof*⟩

**lemma** *eventually-happens'*:
**assumes** $F \neq bot$ *eventually P F*
**shows** $\exists x.\ P\ x$
⟨*proof*⟩

**abbreviation** (*input*) *trivial-limit* :: *'a filter* ⇒ *bool*
**where** *trivial-limit* $F \equiv F = bot$

**lemma** *trivial-limit-def*: *trivial-limit* $F \longleftrightarrow$ *eventually* $(\lambda x.\ False)\ F$
⟨*proof*⟩

**lemma** *False-imp-not-eventually*: $(\forall x.\ \neg\ P\ x\ ) \Longrightarrow \neg$ *trivial-limit net* $\Longrightarrow \neg$ *eventually* $(\lambda x.\ P\ x)$ *net*
⟨*proof*⟩

**lemma** *trivial-limit-eventually*: *trivial-limit net* $\Longrightarrow$ *eventually P net*
⟨*proof*⟩

**lemma** *trivial-limit-eq*: *trivial-limit net* $\longleftrightarrow$ $(\forall P.$ *eventually P net*$)$
⟨*proof*⟩

**lemma** *eventually-Inf*: *eventually P* (*Inf B*) $\longleftrightarrow$ $(\exists X \subseteq B.$ *finite X* $\wedge$ *eventually P* (*Inf X*)$)$
⟨*proof*⟩

**lemma** *eventually-INF*: *eventually P* $(\bigsqcap b \in B.\ F\ b) \longleftrightarrow (\exists X \subseteq B.$ *finite X* $\wedge$ *eventually P* $(\bigsqcap b \in X.\ F\ b))$
⟨*proof*⟩

**lemma** *Inf-filter-not-bot*:
**fixes** $B$ :: *'a filter set*
**shows** $(\bigwedge X.\ X \subseteq B \Longrightarrow$ *finite X* $\Longrightarrow$ *Inf X* $\neq bot) \Longrightarrow$ *Inf B* $\neq bot$
⟨*proof*⟩

**lemma** *INF-filter-not-bot*:
**fixes** $F$ :: *'i* ⇒ *'a filter*
**shows** $(\bigwedge X.\ X \subseteq B \Longrightarrow$ *finite X* $\Longrightarrow (\bigsqcap b \in X.\ F\ b) \neq bot) \Longrightarrow (\bigsqcap b \in B.\ F\ b) \neq bot$
⟨*proof*⟩

**lemma** *eventually-Inf-base*:
**assumes** $B \neq \{\}$ **and** *base*: $\bigwedge F\ G.\ F \in B \Longrightarrow G \in B \Longrightarrow \exists x \in B.\ x \leq inf\ F\ G$
**shows** *eventually P* (*Inf B*) $\longleftrightarrow$ $(\exists b \in B.$ *eventually P b*$)$
⟨*proof*⟩

**lemma** *eventually-INF-base*:
$B \neq \{\} \Longrightarrow (\bigwedge a\ b.\ a \in B \Longrightarrow b \in B \Longrightarrow \exists x \in B.\ F\ x \leq inf\ (F\ a)\ (F\ b)) \Longrightarrow$

*eventually* $P$ $(\bigsqcap b{\in}B.\ F\ b) \longleftrightarrow (\exists\, b{\in}B.\ \textit{eventually}\ P\ (F\ b))$
⟨*proof*⟩

**lemma** *eventually-INF1*: $i \in I \Longrightarrow \textit{eventually}\ P\ (F\ i) \Longrightarrow \textit{eventually}\ P\ (\bigsqcap i{\in}I.\ F\ i)$
⟨*proof*⟩

**lemma** *eventually-INF-finite*:
**assumes** *finite A*
**shows** $\textit{eventually}\ P\ (\bigsqcap\ x{\in}A.\ F\ x) \longleftrightarrow$
$(\exists\, Q.\ (\forall\, x{\in}A.\ \textit{eventually}\ (Q\ x)\ (F\ x)) \wedge (\forall\, y.\ (\forall\, x{\in}A.\ Q\ x\ y) \longrightarrow P\ y))$
⟨*proof*⟩

**lemma** *eventually-le-le*:
**fixes** $P :: {'a} \Rightarrow ({'b} :: \textit{preorder})$
**assumes** $\textit{eventually}\ (\lambda x.\ P\ x \leq Q\ x)\ F$
**assumes** $\textit{eventually}\ (\lambda x.\ Q\ x \leq R\ \ x)\ F$
**shows** $\textit{eventually}\ (\lambda x.\ P\ x \leq R\ x)\ F$
⟨*proof*⟩

### 90.2.2 Map function for filters

**definition** *filtermap* $:: ({'a} \Rightarrow {'b}) \Rightarrow {'a}\ \textit{filter} \Rightarrow {'b}\ \textit{filter}$
**where** $\textit{filtermap}\ f\ F = \textit{Abs-filter}\ (\lambda P.\ \textit{eventually}\ (\lambda x.\ P\ (f\ x))\ F)$

**lemma** *eventually-filtermap*:
$\textit{eventually}\ P\ (\textit{filtermap}\ f\ F) = \textit{eventually}\ (\lambda x.\ P\ (f\ x))\ F$
⟨*proof*⟩

**lemma** *eventually-comp-filtermap*:
$\textit{eventually}\ (P \circ f)\ F \longleftrightarrow \textit{eventually}\ P\ (\textit{filtermap}\ f\ F)$
⟨*proof*⟩

**lemma** *filtermap-compose*: $\textit{filtermap}\ (f \circ g)\ F = \textit{filtermap}\ f\ (\textit{filtermap}\ g\ F)$
⟨*proof*⟩

**lemma** *filtermap-ident*: $\textit{filtermap}\ (\lambda x.\ x)\ F = F$
⟨*proof*⟩

**lemma** *filtermap-filtermap*:
$\textit{filtermap}\ f\ (\textit{filtermap}\ g\ F) = \textit{filtermap}\ (\lambda x.\ f\ (g\ x))\ F$
⟨*proof*⟩

**lemma** *filtermap-mono*: $F \leq F' \Longrightarrow \textit{filtermap}\ f\ F \leq \textit{filtermap}\ f\ F'$
⟨*proof*⟩

**lemma** *filtermap-bot* [*simp*]: $\textit{filtermap}\ f\ \textit{bot} = \textit{bot}$
⟨*proof*⟩

**lemma** *filtermap-bot-iff*: $filtermap\ f\ F = bot \longleftrightarrow F = bot$
⟨*proof*⟩

**lemma** *filtermap-sup*: $filtermap\ f\ (sup\ F1\ F2) = sup\ (filtermap\ f\ F1)\ (filtermap\ f\ F2)$
⟨*proof*⟩

**lemma** *filtermap-SUP*: $filtermap\ f\ (\bigsqcup b{\in}B.\ F\ b) = (\bigsqcup b{\in}B.\ filtermap\ f\ (F\ b))$
⟨*proof*⟩

**lemma** *filtermap-inf*: $filtermap\ f\ (inf\ F1\ F2) \le inf\ (filtermap\ f\ F1)\ (filtermap\ f\ F2)$
⟨*proof*⟩

**lemma** *filtermap-INF*: $filtermap\ f\ (\bigsqcap b{\in}B.\ F\ b) \le (\bigsqcap b{\in}B.\ filtermap\ f\ (F\ b))$
⟨*proof*⟩

**lemma** *frequently-filtermap*:
$frequently\ P\ (filtermap\ f\ F) = frequently\ (\lambda x.\ P\ (f\ x))\ F$
⟨*proof*⟩

### 90.2.3 Contravariant map function for filters

**definition** *filtercomap* :: $('a \Rightarrow 'b) \Rightarrow 'b\ filter \Rightarrow 'a\ filter$ **where**
$filtercomap\ f\ F = Abs\text{-}filter\ (\lambda P.\ \exists Q.\ eventually\ Q\ F \land (\forall x.\ Q\ (f\ x) \longrightarrow P\ x))$

**lemma** *eventually-filtercomap*:
$eventually\ P\ (filtercomap\ f\ F) \longleftrightarrow (\exists Q.\ eventually\ Q\ F \land (\forall x.\ Q\ (f\ x) \longrightarrow P\ x))$
⟨*proof*⟩

**lemma** *filtercomap-ident*: $filtercomap\ (\lambda x.\ x)\ F = F$
⟨*proof*⟩

**lemma** *filtercomap-filtercomap*: $filtercomap\ f\ (filtercomap\ g\ F) = filtercomap\ (\lambda x.\ g\ (f\ x))\ F$
⟨*proof*⟩

**lemma** *filtercomap-mono*: $F \le F' \Longrightarrow filtercomap\ f\ F \le filtercomap\ f\ F'$
⟨*proof*⟩

**lemma** *filtercomap-bot* [*simp*]: $filtercomap\ f\ bot = bot$
⟨*proof*⟩

**lemma** *filtercomap-top* [*simp*]: $filtercomap\ f\ top = top$
⟨*proof*⟩

**lemma** *filtercomap-inf*: $filtercomap\ f\ (inf\ F1\ F2) = inf\ (filtercomap\ f\ F1)\ (filtercomap\ f\ F2)$

⟨*proof*⟩

**lemma** *filtercomap-sup*: *filtercomap f* (*sup F1 F2*) $\geq$ *sup* (*filtercomap f F1*) (*filtercomap f F2*)
⟨*proof*⟩

**lemma** *filtercomap-INF*: *filtercomap f* ($\bigsqcap b\in B.\ F\ b$) = ($\bigsqcap b\in B.$ *filtercomap f* ($F$ $b$))
⟨*proof*⟩

**lemma** *filtercomap-SUP*:
*filtercomap f* ($\bigsqcup b\in B.\ F\ b$) $\geq$ ($\bigsqcup b\in B.$ *filtercomap f* ($F\ b$))
⟨*proof*⟩

**lemma** *filtermap-le-iff-le-filtercomap*: *filtermap f F* $\leq$ $G$ $\longleftrightarrow$ $F$ $\leq$ *filtercomap f G*
⟨*proof*⟩

**lemma** *filtercomap-neq-bot*:
**assumes** $\bigwedge P.$ *eventually P F* $\Longrightarrow$ $\exists\, x.\ P\ (f\ x)$
**shows** *filtercomap f F* $\neq$ *bot*
⟨*proof*⟩

**lemma** *filtercomap-neq-bot-surj*:
**assumes** $F \neq$ *bot* **and** *surj f*
**shows** *filtercomap f F* $\neq$ *bot*
⟨*proof*⟩

**lemma** *eventually-filtercomapI* [*intro*]:
**assumes** *eventually P F*
**shows** *eventually* ($\lambda x.\ P\ (f\ x)$) (*filtercomap f F*)
⟨*proof*⟩

**lemma** *filtermap-filtercomap*: *filtermap f* (*filtercomap f F*) $\leq F$
⟨*proof*⟩

**lemma** *filtercomap-filtermap*: *filtercomap f* (*filtermap f F*) $\geq F$
⟨*proof*⟩

### 90.2.4 Standard filters

**definition** *principal* :: *'a set* $\Rightarrow$ *'a filter* **where**
*principal S* = *Abs-filter* ($\lambda P.\ \forall\, x\in S.\ P\ x$)

**lemma** *eventually-principal*: *eventually P* (*principal S*) $\longleftrightarrow$ ($\forall\, x\in S.\ P\ x$)
⟨*proof*⟩

**lemma** *eventually-inf-principal*: *eventually P* (*inf F* (*principal s*)) $\longleftrightarrow$ *eventually* ($\lambda x.\ x \in s \longrightarrow P\ x$) $F$
⟨*proof*⟩

**lemma** *principal-UNIV*[*simp*]: *principal UNIV* = *top*
⟨*proof*⟩

**lemma** *principal-empty*[*simp*]: *principal* {} = *bot*
⟨*proof*⟩

**lemma** *principal-eq-bot-iff*: *principal X* = *bot* ⟷ *X* = {}
⟨*proof*⟩

**lemma** *principal-le-iff*[*iff*]: *principal A* ≤ *principal B* ⟷ *A* ⊆ *B*
⟨*proof*⟩

**lemma** *le-principal*: *F* ≤ *principal A* ⟷ *eventually* (λ*x*. *x* ∈ *A*) *F*
⟨*proof*⟩

**lemma** *principal-inject*[*iff*]: *principal A* = *principal B* ⟷ *A* = *B*
⟨*proof*⟩

**lemma** *sup-principal*[*simp*]: *sup* (*principal A*) (*principal B*) = *principal* (*A* ∪ *B*)
⟨*proof*⟩

**lemma** *inf-principal*[*simp*]: *inf* (*principal A*) (*principal B*) = *principal* (*A* ∩ *B*)
⟨*proof*⟩

**lemma** *SUP-principal*[*simp*]: (⨆*i*∈*I*. *principal* (*A i*)) = *principal* (⋃*i*∈*I*. *A i*)
⟨*proof*⟩

**lemma** *INF-principal-finite*: *finite X* ⟹ (⨅*x*∈*X*. *principal* (*f x*)) = *principal* (⋂*x*∈*X*. *f x*)
⟨*proof*⟩

**lemma** *filtermap-principal*[*simp*]: *filtermap f* (*principal A*) = *principal* (*f* ‘ *A*)
⟨*proof*⟩

**lemma** *filtercomap-principal*[*simp*]: *filtercomap f* (*principal A*) = *principal* (*f* −‘ *A*)
⟨*proof*⟩

### 90.2.5 Order filters

**definition** *at-top* :: (′*a*::*order*) *filter*
**where** *at-top* = (⨅*k*. *principal* {*k* ..})

**lemma** *at-top-sub*: *at-top* = (⨅*k*∈{*c*::′*a*::*linorder*..}. *principal* {*k* ..})
⟨*proof*⟩

**lemma** *eventually-at-top-linorder*: *eventually P at-top* ⟷ (∃ *N*::′*a*::*linorder*. ∀ *n*≥*N*. *P n*)

⟨*proof*⟩

**lemma** *eventually-filtercomap-at-top-linorder*:
*eventually* $P$ (*filtercomap* $f$ *at-top*) $\longleftrightarrow$ ($\exists N::'a::linorder.\ \forall x.\ f\ x \geq N \longrightarrow P\ x$)
⟨*proof*⟩

**lemma** *eventually-at-top-linorderI*:
**fixes** $c::'a::linorder$
**assumes** $\bigwedge x.\ c \leq x \Longrightarrow P\ x$
**shows** *eventually* $P$ *at-top*
⟨*proof*⟩

**lemma** *eventually-ge-at-top* [*simp*]:
*eventually* ($\lambda x.\ (c::\text{-}::linorder) \leq x$) *at-top*
⟨*proof*⟩

**lemma** *eventually-at-top-dense*: *eventually* $P$ *at-top* $\longleftrightarrow$ ($\exists N::'a::\{no\text{-}top, linorder\}.$ $\forall n{>}N.\ P\ n$)
⟨*proof*⟩

**lemma** *eventually-filtercomap-at-top-dense*:
*eventually* $P$ (*filtercomap* $f$ *at-top*) $\longleftrightarrow$ ($\exists N::'a::\{no\text{-}top, linorder\}.\ \forall x.\ f\ x > N$ $\longrightarrow P\ x$)
⟨*proof*⟩

**lemma** *eventually-at-top-not-equal* [*simp*]: *eventually* ($\lambda x::'a::\{no\text{-}top, linorder\}.\ x$ $\neq c$) *at-top*
⟨*proof*⟩

**lemma** *eventually-gt-at-top* [*simp*]: *eventually* ($\lambda x.\ (c::\text{-}::\{no\text{-}top, linorder\}) < x$) *at-top*
⟨*proof*⟩

**lemma** *eventually-all-ge-at-top*:
**assumes** *eventually* $P$ (*at-top* :: ($'a$ :: *linorder*) *filter*)
**shows** *eventually* ($\lambda x.\ \forall y{\geq}x.\ P\ y$) *at-top*
⟨*proof*⟩

**definition** *at-bot* :: ($'a$::*order*) *filter*
**where** *at-bot* = ($\prod k.$ *principal* $\{..\ k\}$)

**lemma** *at-bot-sub*: *at-bot* = ($\prod k{\in}\{..\ c::'a::linorder\}.$ *principal* $\{..\ k\}$)
⟨*proof*⟩

**lemma** *eventually-at-bot-linorder*:
**fixes** $P$ :: $'a::linorder \Rightarrow bool$ **shows** *eventually* $P$ *at-bot* $\longleftrightarrow$ ($\exists N.\ \forall n{\leq}N.\ P\ n$)
⟨*proof*⟩

**lemma** *eventually-at-bot-linorderI*:

**fixes** $c::'a::linorder$
**assumes** $\bigwedge x.\ x \leq c \Longrightarrow P\ x$
**shows** *eventually P at-bot*
⟨*proof*⟩

**lemma** *eventually-filtercomap-at-bot-linorder*:
*eventually P* (*filtercomap f at-bot*) $\longleftrightarrow$ ($\exists\, N::'a::linorder.\ \forall\, x.\ f\ x \leq N \longrightarrow P\ x$)
⟨*proof*⟩

**lemma** *eventually-le-at-bot* [*simp*]:
*eventually* ($\lambda x.\ x \leq (c::-::linorder)$) *at-bot*
⟨*proof*⟩

**lemma** *eventually-at-bot-dense*: *eventually P at-bot* $\longleftrightarrow$ ($\exists\, N::'a::\{no\text{-}bot,\ linorder\}.$ $\forall\, n<N.\ P\ n$)
⟨*proof*⟩

**lemma** *eventually-filtercomap-at-bot-dense*:
*eventually P* (*filtercomap f at-bot*) $\longleftrightarrow$ ($\exists\, N::'a::\{no\text{-}bot,\ linorder\}.\ \forall\, x.\ f\ x < N$ $\longrightarrow P\ x$)
⟨*proof*⟩

**lemma** *eventually-at-bot-not-equal* [*simp*]: *eventually* ($\lambda x::'a::\{no\text{-}bot,\ linorder\}.\ x$ $\neq c$) *at-bot*
⟨*proof*⟩

**lemma** *eventually-gt-at-bot* [*simp*]:
*eventually* ($\lambda x.\ x < (c::-::unbounded\text{-}dense\text{-}linorder)$) *at-bot*
⟨*proof*⟩

**lemma** *trivial-limit-at-bot-linorder* [*simp*]: $\neg$ *trivial-limit* (*at-bot* ::('*a*::*linorder*) *filter*)
⟨*proof*⟩

**lemma** *trivial-limit-at-top-linorder* [*simp*]: $\neg$ *trivial-limit* (*at-top* ::('*a*::*linorder*) *filter*)
⟨*proof*⟩

## 90.3 Sequentially

**abbreviation** *sequentially* :: *nat filter*
**where** *sequentially* $\equiv$ *at-top*

**lemma** *eventually-sequentially*:
*eventually P sequentially* $\longleftrightarrow$ ($\exists\, N.\ \forall\, n \geq N.\ P\ n$)
⟨*proof*⟩

**lemma** *frequently-sequentially*:
*frequently P sequentially* $\longleftrightarrow$ ($\forall\, N.\ \exists\, n \geq N.\ P\ n$)

⟨*proof*⟩

**lemma** *sequentially-bot* [*simp*, *intro*]: *sequentially* $\neq$ *bot*
⟨*proof*⟩

**lemmas** *trivial-limit-sequentially* = *sequentially-bot*

**lemma** *eventually-False-sequentially* [*simp*]:
$\neg$ *eventually* ($\lambda n$. *False*) *sequentially*
⟨*proof*⟩

**lemma** *le-sequentially*:
$F \leq$ *sequentially* $\longleftrightarrow$ ($\forall N$. *eventually* ($\lambda n$. $N \leq n$) $F$)
⟨*proof*⟩

**lemma** *eventually-sequentiallyI* [*intro?*]:
**assumes** $\bigwedge x$. $c \leq x \Longrightarrow P\ x$
**shows** *eventually* $P$ *sequentially*
⟨*proof*⟩

**lemma** *eventually-sequentially-Suc* [*simp*]: *eventually* ($\lambda i$. $P$ (*Suc* $i$)) *sequentially* $\longleftrightarrow$ *eventually* $P$ *sequentially*
⟨*proof*⟩

**lemma** *eventually-sequentially-seg* [*simp*]: *eventually* ($\lambda n$. $P$ ($n + k$)) *sequentially* $\longleftrightarrow$ *eventually* $P$ *sequentially*
⟨*proof*⟩

**lemma** *filtermap-sequentually-ne-bot*: *filtermap f sequentially* $\neq$ *bot*
⟨*proof*⟩

## 90.4 Increasing finite subsets

**definition** *finite-subsets-at-top* **where**
*finite-subsets-at-top* $A$ = ($\bigsqcap$ $X \in \{X$. *finite* $X \wedge X \subseteq A\}$. *principal* $\{Y$. *finite* $Y \wedge X \subseteq Y \wedge Y \subseteq A\}$)

**abbreviation** *finite-sets-at-top* $\equiv$ *finite-subsets-at-top UNIV*

**lemma** *eventually-finite-subsets-at-top*:
*eventually* $P$ (*finite-subsets-at-top* $A$) $\longleftrightarrow$
($\exists X$. *finite* $X \wedge X \subseteq A \wedge$ ($\forall Y$. *finite* $Y \wedge X \subseteq Y \wedge Y \subseteq A \longrightarrow P\ Y$))
⟨*proof*⟩

**lemma** *eventually-finite-subsets-at-top-weakI* [*intro*]:
**assumes** $\bigwedge X$. *finite* $X \Longrightarrow X \subseteq A \Longrightarrow P\ X$
**shows** *eventually* $P$ (*finite-subsets-at-top* $A$)
⟨*proof*⟩

**lemma** *finite-subsets-at-top-neq-bot* [*simp*]: *finite-subsets-at-top* $A \neq$ *bot*
⟨*proof*⟩

**lemma** *filtermap-image-finite-subsets-at-top*:
**assumes** *inj-on f A*
**shows** *filtermap* ((\`) *f*) (*finite-subsets-at-top A*) = *finite-subsets-at-top* (*f* \` *A*)
⟨*proof*⟩

**lemma** *eventually-finite-subsets-at-top-finite*:
**assumes** *finite A*
**shows** *eventually P* (*finite-subsets-at-top A*) $\longleftrightarrow$ *P A*
⟨*proof*⟩

**lemma** *finite-subsets-at-top-finite*: *finite A* $\Longrightarrow$ *finite-subsets-at-top A* = *principal* {*A*}
⟨*proof*⟩

## 90.5 The cofinite filter

**definition** *cofinite* = *Abs-filter* ($\lambda P$. *finite* {*x*. $\neg$ *P x*})

**abbreviation** *Inf-many* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* (**binder** ‹$\exists_\infty$› *10*)
**where** *Inf-many P* $\equiv$ *frequently P cofinite*

**abbreviation** *Alm-all* :: ('*a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* (**binder** ‹$\forall_\infty$› *10*)
**where** *Alm-all P* $\equiv$ *eventually P cofinite*

**notation** (*ASCII*)
*Inf-many* (**binder** ‹*INFM* › *10*) **and**
*Alm-all* (**binder** ‹*MOST* › *10*)

**lemma** *eventually-cofinite*: *eventually P cofinite* $\longleftrightarrow$ *finite* {*x*. $\neg$ *P x*}
⟨*proof*⟩

**lemma** *frequently-cofinite*: *frequently P cofinite* $\longleftrightarrow$ $\neg$ *finite* {*x*. *P x*}
⟨*proof*⟩

**lemma** *cofinite-bot*[*simp*]: *cofinite* = (*bot*::'*a filter*) $\longleftrightarrow$ *finite* (*UNIV* :: '*a set*)
⟨*proof*⟩

**lemma** *cofinite-eq-sequentially*: *cofinite* = *sequentially*
⟨*proof*⟩

### 90.5.1 Product of filters

**definition** *prod-filter* :: '*a filter* $\Rightarrow$ '*b filter* $\Rightarrow$ ('*a* $\times$ '*b*) *filter* (**infixr** ‹$\times_F$› *80*)
**where**
*prod-filter F G* =
($\bigsqcap$(*P*, *Q*)$\in${(*P*, *Q*). *eventually P F* $\wedge$ *eventually Q G*}. *principal* {(*x*, *y*). *P x* $\wedge$ *Q y*})

**lemma** *eventually-prod-filter*: *eventually* $P$ $(F \times_F G) \longleftrightarrow$
$(\exists Pf\ Pg.\ eventually\ Pf\ F \wedge eventually\ Pg\ G \wedge (\forall x\ y.\ Pf\ x \longrightarrow Pg\ y \longrightarrow P\ (x, y)))$
⟨*proof*⟩

**lemma** *eventually-prod1*:
**assumes** $B \neq bot$
**shows** $(\forall_F\ (x, y)\ in\ A \times_F B.\ P\ x) \longleftrightarrow (\forall_F\ x\ in\ A.\ P\ x)$
⟨*proof*⟩

**lemma** *eventually-prod2*:
**assumes** $A \neq bot$
**shows** $(\forall_F\ (x, y)\ in\ A \times_F B.\ P\ y) \longleftrightarrow (\forall_F\ y\ in\ B.\ P\ y)$
⟨*proof*⟩

**lemma** *eventually-eventually-prod-filter1*:
**assumes** *eventually* $P$ $(F \times_F G)$
**shows** *eventually* $(\lambda x.\ eventually\ (\lambda y.\ P\ (x, y))\ G)\ F$
⟨*proof*⟩

**lemma** *eventually-eventually-prod-filter2*:
**assumes** *eventually* $P$ $(F \times_F G)$
**shows** *eventually* $(\lambda y.\ eventually\ (\lambda x.\ P\ (x, y))\ F)\ G$
⟨*proof*⟩

**lemma** *INF-filter-bot-base*:
**fixes** $F$ :: $'a \Rightarrow 'b$ *filter*
**assumes** $*$: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow \exists k{\in}I.\ F\ k \leq F\ i \sqcap F\ j$
**shows** $(\bigsqcap i{\in}I.\ F\ i) = bot \longleftrightarrow (\exists i{\in}I.\ F\ i = bot)$
⟨*proof*⟩

**lemma** *Collect-empty-eq-bot*: *Collect* $P = \{\} \longleftrightarrow P = \bot$
⟨*proof*⟩

**lemma** *prod-filter-eq-bot*: $A \times_F B = bot \longleftrightarrow A = bot \vee B = bot$
⟨*proof*⟩

**lemma** *prod-filter-mono*: $F \leq F' \Longrightarrow G \leq G' \Longrightarrow F \times_F G \leq F' \times_F G'$
⟨*proof*⟩

**lemma** *prod-filter-mono-iff*:
**assumes** *nAB*: $A \neq bot\ B \neq bot$
**shows** $A \times_F B \leq C \times_F D \longleftrightarrow A \leq C \wedge B \leq D$
⟨*proof*⟩

**lemma** *eventually-prod-same*: *eventually* $P$ $(F \times_F F) \longleftrightarrow$
$(\exists Q.\ eventually\ Q\ F \wedge (\forall x\ y.\ Q\ x \longrightarrow Q\ y \longrightarrow P\ (x, y)))$
⟨*proof*⟩

**lemma** *eventually-prod-sequentially*:
*eventually* $P$ (*sequentially* $\times_F$ *sequentially*) $\longleftrightarrow$ ($\exists N. \forall m \geq N. \forall n \geq N. P\ (n, m)$)
⟨*proof*⟩

**lemma** *principal-prod-principal*: *principal* $A \times_F$ *principal* $B$ = *principal* $(A \times B)$
⟨*proof*⟩

**lemma** *le-prod-filterI*:
*filtermap fst* $F \leq A \Longrightarrow$ *filtermap snd* $F \leq B \Longrightarrow F \leq A \times_F B$
⟨*proof*⟩

**lemma** *filtermap-fst-prod-filter*: *filtermap fst* $(A \times_F B) \leq A$
⟨*proof*⟩

**lemma** *filtermap-snd-prod-filter*: *filtermap snd* $(A \times_F B) \leq B$
⟨*proof*⟩

**lemma** *prod-filter-INF*:
**assumes** $I \neq \{\}$ **and** $J \neq \{\}$
**shows** $(\bigsqcap i\in I.\ A\ i) \times_F (\bigsqcap j\in J.\ B\ j) = (\bigsqcap i\in I.\ \bigsqcap j\in J.\ A\ i \times_F B\ j)$
⟨*proof*⟩

**lemma** *filtermap-Pair*: *filtermap* ($\lambda x.$ ($f\ x$, $g\ x$)) $F \leq$ *filtermap* $f\ F \times_F$ *filtermap* $g\ F$
⟨*proof*⟩

**lemma** *eventually-prodI*: *eventually* $P\ F \Longrightarrow$ *eventually* $Q\ G \Longrightarrow$ *eventually* ($\lambda x.$ $P$ (*fst* $x$) $\wedge$ $Q$ (*snd* $x$)) $(F \times_F G)$
⟨*proof*⟩

**lemma** *prod-filter-INF1*: $I \neq \{\} \Longrightarrow (\bigsqcap i\in I.\ A\ i) \times_F B = (\bigsqcap i\in I.\ A\ i \times_F B)$
⟨*proof*⟩

**lemma** *prod-filter-INF2*: $J \neq \{\} \Longrightarrow A \times_F (\bigsqcap i\in J.\ B\ i) = (\bigsqcap i\in J.\ A \times_F B\ i)$
⟨*proof*⟩

**lemma** *prod-filtermap1*: *prod-filter* (*filtermap* $f\ F$) $G$ = *filtermap* (*apfst* $f$) (*prod-filter* $F\ G$)
⟨*proof*⟩

**lemma** *prod-filtermap2*: *prod-filter* $F$ (*filtermap* $g\ G$) = *filtermap* (*apsnd* $g$) (*prod-filter* $F\ G$)
⟨*proof*⟩

**lemma** *prod-filter-assoc*:
*prod-filter* (*prod-filter* $F\ G$) $H$ = *filtermap* ($\lambda(x, y, z).\ ((x, y), z)$) (*prod-filter* $F$ (*prod-filter* $G\ H$))

⟨*proof*⟩

**lemma** *prod-filter-principal-singleton*: *prod-filter* (*principal* $\{x\}$) $F$ = *filtermap* (*Pair* $x$) $F$
⟨*proof*⟩

**lemma** *prod-filter-principal-singleton2*: *prod-filter* $F$ (*principal* $\{x\}$) = *filtermap* ($\lambda a.\ (a,\ x)$) $F$
⟨*proof*⟩

**lemma** *prod-filter-commute*: *prod-filter* $F$ $G$ = *filtermap* *prod.swap* (*prod-filter* $G$ $F$)
⟨*proof*⟩

## 90.6 Limits

**definition** *filterlim* :: ($'a \Rightarrow 'b$) $\Rightarrow$ $'b$ *filter* $\Rightarrow$ $'a$ *filter* $\Rightarrow$ *bool* **where**
*filterlim f F2 F1* $\longleftrightarrow$ *filtermap f F1* $\leq$ *F2*

**syntax**
*-LIM* :: *pttrns* $\Rightarrow$ $'a \Rightarrow 'b \Rightarrow 'a \Rightarrow$ *bool* (‹(‹*indent=3 notation=*‹*binder LIM*››*LIM* (-)/ (-)./ (-) :> (-))› [*1000*, *10*, *0*, *10*] *10*)
**syntax-consts**
*-LIM* == *filterlim*
**translations**
*LIM x F1. f* :> *F2* == *CONST filterlim* ($\lambda x.\ f$) *F2 F1*

**lemma** *filterlim-filtercomapI*: *filterlim f F G* $\Longrightarrow$ *filterlim* ($\lambda x.\ f\ (g\ x)$) *F* (*filtercomap* *g G*)
⟨*proof*⟩

**lemma** *filterlim-top* [*simp*]: *filterlim f top F*
⟨*proof*⟩

**lemma** *filterlim-iff*:
(*LIM x F1. f x* :> *F2*) $\longleftrightarrow$ ($\forall P$. *eventually P F2* $\longrightarrow$ *eventually* ($\lambda x.\ P\ (f\ x)$) *F1*)
⟨*proof*⟩

**lemma** *filterlim-compose*:
*filterlim g F3 F2* $\Longrightarrow$ *filterlim f F2 F1* $\Longrightarrow$ *filterlim* ($\lambda x.\ g\ (f\ x)$) *F3 F1*
⟨*proof*⟩

**lemma** *filterlim-mono*:
*filterlim f F2 F1* $\Longrightarrow$ *F2* $\leq$ *F2'* $\Longrightarrow$ *F1'* $\leq$ *F1* $\Longrightarrow$ *filterlim f F2' F1'*
⟨*proof*⟩

**lemma** *filterlim-ident*: *LIM x F. x* :> *F*
⟨*proof*⟩

**lemma** *filterlim-cong*:
$F1 = F1' \Longrightarrow F2 = F2' \Longrightarrow$ *eventually* $(\lambda x.\ f\ x = g\ x)\ F2 \Longrightarrow$ *filterlim* $f\ F1\ F2 =$ *filterlim* $g\ F1'\ F2'$
⟨*proof*⟩

**lemma** *filterlim-mono-eventually*:
**assumes** *filterlim* $f\ F\ G$ **and** *ord*: $F \le F'\ G' \le G$
**assumes** *eq*: *eventually* $(\lambda x.\ f\ x = f'\ x)\ G'$
**shows** *filterlim* $f'\ F'\ G'$
⟨*proof*⟩

**lemma** *filtermap-mono-strong*: *inj* $f \Longrightarrow$ *filtermap* $f\ F \le$ *filtermap* $f\ G \longleftrightarrow F \le G$
⟨*proof*⟩

**lemma** *eventually-compose-filterlim*:
**assumes** *eventually* $P\ F$ *filterlim* $f\ F\ G$
**shows** *eventually* $(\lambda x.\ P\ (f\ x))\ G$
⟨*proof*⟩

**lemma** *filtermap-eq-strong*: *inj* $f \Longrightarrow$ *filtermap* $f\ F =$ *filtermap* $f\ G \longleftrightarrow F = G$
⟨*proof*⟩

**lemma** *filtermap-fun-inverse*:
**assumes** *g*: *filterlim* $g\ F\ G$
**assumes** *f*: *filterlim* $f\ G\ F$
**assumes** *ev*: *eventually* $(\lambda x.\ f\ (g\ x) = x)\ G$
**shows** *filtermap* $f\ F = G$
⟨*proof*⟩

**lemma** *filterlim-principal*:
$(LIM\ x\ F.\ f\ x :> principal\ S) \longleftrightarrow (eventually\ (\lambda x.\ f\ x \in S)\ F)$
⟨*proof*⟩

**lemma** *filterlim-filtercomap* [*intro*]: *filterlim* $f\ F$ (*filtercomap* $f\ F$)
⟨*proof*⟩

**lemma** *filterlim-inf*:
$(LIM\ x\ F1.\ f\ x :> inf\ F2\ F3) \longleftrightarrow ((LIM\ x\ F1.\ f\ x :> F2) \wedge (LIM\ x\ F1.\ f\ x :> F3))$
⟨*proof*⟩

**lemma** *filterlim-INF*:
$(LIM\ x\ F.\ f\ x :> (\bigsqcap b{\in}B.\ G\ b)) \longleftrightarrow (\forall\, b{\in}B.\ LIM\ x\ F.\ f\ x :> G\ b)$
⟨*proof*⟩

**lemma** *filterlim-INF-INF*:
$(\bigwedge m.\ m \in J \Longrightarrow \exists\, i{\in}I.\ filtermap\ f\ (F\ i) \le G\ m) \Longrightarrow LIM\ x\ (\bigsqcap i{\in}I.\ F\ i).\ f\ x$

$:> (\prod j{\in}J.\ G\ j)$
⟨proof⟩

**lemma** *filterlim-INF'*: $x \in A \Longrightarrow$ *filterlim f F* $(G\ x) \Longrightarrow$ *filterlim f F* $(\prod\ x{\in}A.\ G\ x)$
⟨proof⟩

**lemma** *filterlim-filtercomap-iff*: *filterlim f* (*filtercomap g G*) $F \longleftrightarrow$ *filterlim* $(g \circ f)\ G\ F$
⟨proof⟩

**lemma** *filterlim-iff-le-filtercomap*: *filterlim f F G* $\longleftrightarrow G \leq$ *filtercomap f F*
⟨proof⟩

**lemma** *filterlim-base*:
$(\bigwedge m\ x.\ m \in J \Longrightarrow i\ m \in I) \Longrightarrow (\bigwedge m\ x.\ m \in J \Longrightarrow x \in F\ (i\ m) \Longrightarrow f\ x \in G\ m) \Longrightarrow$
$LIM\ x\ (\prod i{\in}I.\ principal\ (F\ i)).\ f\ x :> (\prod j{\in}J.\ principal\ (G\ j))$
⟨proof⟩

**lemma** *filterlim-base-iff*:
**assumes** $I \neq \{\}$ **and** *chain*: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow F\ i \subseteq F\ j \lor F\ j \subseteq F\ i$
**shows** $(LIM\ x\ (\prod i{\in}I.\ principal\ (F\ i)).\ f\ x :> \prod j{\in}J.\ principal\ (G\ j)) \longleftrightarrow$
$(\forall j{\in}J.\ \exists i{\in}I.\ \forall x{\in}F\ i.\ f\ x \in G\ j)$
⟨proof⟩

**lemma** *filterlim-filtermap*: *filterlim f F1* (*filtermap g F2*) = *filterlim* $(\lambda x.\ f\ (g\ x))$ *F1 F2*
⟨proof⟩

**lemma** *filterlim-sup*:
*filterlim f F F1* $\Longrightarrow$ *filterlim f F F2* $\Longrightarrow$ *filterlim f F* (*sup F1 F2*)
⟨proof⟩

**lemma** *filterlim-sequentially-Suc*:
(*LIM x sequentially. f* (*Suc x*) :> *F*) $\longleftrightarrow$ (*LIM x sequentially. f x* :> *F*)
⟨proof⟩

**lemma** *filterlim-Suc*: *filterlim Suc sequentially sequentially*
⟨proof⟩

**lemma** *filterlim-If*:
*LIM x inf F* (*principal* $\{x.\ P\ x\}$). *f x* :> $G \Longrightarrow$
*LIM x inf F* (*principal* $\{x.\ \neg\ P\ x\}$). *g x* :> $G \Longrightarrow$
*LIM x F. if P x then f x else g x* :> *G*
⟨proof⟩

**lemma** *filterlim-Pair*:
$LIM\ x\ F.\ f\ x :> G \Longrightarrow LIM\ x\ F.\ g\ x :> H \Longrightarrow LIM\ x\ F.\ (f\ x,\ g\ x) :> G \times_F H$

⟨*proof*⟩

## 90.7 Limits to *at-top* and *at-bot*

**lemma** *filterlim-at-top*:
**fixes** $f :: 'a \Rightarrow ('b::linorder)$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}top) \longleftrightarrow (\forall Z.\ eventually\ (\lambda x.\ Z \le f\ x)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-top-mono*:
$LIM\ x\ F.\ f\ x :> at\text{-}top \Longrightarrow eventually\ (\lambda x.\ f\ x \le (g\ x::'a::linorder))\ F \Longrightarrow$
$LIM\ x\ F.\ g\ x :> at\text{-}top$
⟨*proof*⟩

**lemma** *filterlim-at-top-dense*:
**fixes** $f :: 'a \Rightarrow ('b::unbounded\text{-}dense\text{-}linorder)$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}top) \longleftrightarrow (\forall Z.\ eventually\ (\lambda x.\ Z < f\ x)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-top-ge*:
**fixes** $f :: 'a \Rightarrow ('b::linorder)$ **and** $c :: 'b$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}top) \longleftrightarrow (\forall Z \ge c.\ eventually\ (\lambda x.\ Z \le f\ x)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-top-at-top*:
**fixes** $f :: 'a::linorder \Rightarrow 'b::linorder$
**assumes** *mono*: $\bigwedge x\ y.\ Q\ x \Longrightarrow Q\ y \Longrightarrow x \le y \Longrightarrow f\ x \le f\ y$
**assumes** *bij*: $\bigwedge x.\ P\ x \Longrightarrow f\ (g\ x) = x\ \bigwedge x.\ P\ x \Longrightarrow Q\ (g\ x)$
**assumes** *Q*: *eventually Q at-top*
**assumes** *P*: *eventually P at-top*
**shows** *filterlim f at-top at-top*
⟨*proof*⟩

**lemma** *filterlim-at-top-gt*:
**fixes** $f :: 'a \Rightarrow ('b::unbounded\text{-}dense\text{-}linorder)$ **and** $c :: 'b$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}top) \longleftrightarrow (\forall Z > c.\ eventually\ (\lambda x.\ Z \le f\ x)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-bot*:
**fixes** $f :: 'a \Rightarrow ('b::linorder)$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}bot) \longleftrightarrow (\forall Z.\ eventually\ (\lambda x.\ f\ x \le Z)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-bot-dense*:
**fixes** $f :: 'a \Rightarrow ('b::\{dense\text{-}linorder,\ no\text{-}bot\})$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}bot) \longleftrightarrow (\forall Z.\ eventually\ (\lambda x.\ f\ x < Z)\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-bot-le*:

**fixes** $f$ :: $'a \Rightarrow ('b$::*linorder*) **and** $c$ :: $'b$
**shows** (*LIM x F. f x* :> *at-bot*) $\longleftrightarrow$ ($\forall Z \leq c$. *eventually* ($\lambda x$. $Z \geq f\ x$) $F$)
⟨*proof*⟩

**lemma** *filterlim-at-bot-lt*:
**fixes** $f$ :: $'a \Rightarrow ('b$::*unbounded-dense-linorder*) **and** $c$ :: $'b$
**shows** (*LIM x F. f x* :> *at-bot*) $\longleftrightarrow$ ($\forall Z < c$. *eventually* ($\lambda x$. $Z \geq f\ x$) $F$)
⟨*proof*⟩

**lemma** *filterlim-at-top-div-const-nat*:
**assumes** $c > 0$
**shows** *filterlim* ($\lambda x$::*nat*. *x div c*) *at-top at-top*
⟨*proof*⟩

**lemma** *filterlim-finite-subsets-at-top*:
*filterlim f* (*finite-subsets-at-top A*) $F$ $\longleftrightarrow$
($\forall X$. *finite X* $\wedge$ $X \subseteq A$ $\longrightarrow$ *eventually* ($\lambda y$. *finite* ($f\ y$) $\wedge$ $X \subseteq f\ y$ $\wedge$ $f\ y \subseteq A$) $F$)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *filterlim-atMost-at-top*:
*filterlim* ($\lambda n$. {..n}) (*finite-subsets-at-top* (*UNIV* :: *nat set*)) *at-top*
⟨*proof*⟩

**lemma** *filterlim-lessThan-at-top*:
*filterlim* ($\lambda n$. {..<n}) (*finite-subsets-at-top* (*UNIV* :: *nat set*)) *at-top*
⟨*proof*⟩

**lemma** *filterlim-minus-const-nat-at-top*:
*filterlim* ($\lambda n$. $n - c$) *sequentially sequentially*
⟨*proof*⟩

**lemma** *filterlim-add-const-nat-at-top*:
*filterlim* ($\lambda n$. $n + c$) *sequentially sequentially*
⟨*proof*⟩

## 90.8 Setup *'a filter* for lifting and transfer

**lemma** *filtermap-id* [*simp*, *id-simps*]: *filtermap id* = *id*
⟨*proof*⟩

**lemma** *filtermap-id'* [*simp*]: *filtermap* ($\lambda x$. $x$) = ($\lambda F$. $F$)
⟨*proof*⟩

**context includes** *lifting-syntax*
**begin**

**definition** *map-filter-on* :: *'a set* $\Rightarrow$ ($'a \Rightarrow 'b$) $\Rightarrow$ *'a filter* $\Rightarrow$ *'b filter* **where**

*map-filter-on* $X\ f\ F = \textit{Abs-filter}\ (\lambda P.\ \textit{eventually}\ (\lambda x.\ P\ (f\ x) \wedge x \in X)\ F)$

**lemma** *is-filter-map-filter-on*:
*is-filter* $(\lambda P.\ \forall_F\ x\ \textit{in}\ F.\ P\ (f\ x) \wedge x \in X) \longleftrightarrow \textit{eventually}\ (\lambda x.\ x \in X)\ F$
⟨*proof*⟩

**lemma** *eventually-map-filter-on*: *eventually* $P$ (*map-filter-on* $X\ f\ F) = (\forall_F\ x\ \textit{in}\ F.\ P\ (f\ x) \wedge x \in X)$
**if** *eventually* $(\lambda x.\ x \in X)\ F$
⟨*proof*⟩

**lemma** *map-filter-on-UNIV*: *map-filter-on UNIV* = *filtermap*
⟨*proof*⟩

**lemma** *map-filter-on-comp*: *map-filter-on* $X\ f$ (*map-filter-on* $Y\ g\ F$) = *map-filter-on* $Y\ (f \circ g)\ F$
**if** $g\ `\ Y \subseteq X$ **and** *eventually* $(\lambda x.\ x \in Y)\ F$
⟨*proof*⟩

**inductive** *rel-filter* :: $('a \Rightarrow 'b \Rightarrow \textit{bool}) \Rightarrow 'a\ \textit{filter} \Rightarrow 'b\ \textit{filter} \Rightarrow \textit{bool}$ **for** $R\ F\ G$ **where**
*rel-filter* $R\ F\ G$ **if** *eventually* (*case-prod* $R$) $Z$ *map-filter-on* $\{(x, y).\ R\ x\ y\}$ *fst* $Z = F$ *map-filter-on* $\{(x, y).\ R\ x\ y\}$ *snd* $Z = G$

**lemma** *rel-filter-eq* [*relator-eq*]: *rel-filter* $(=) = (=)$
⟨*proof*⟩

**lemma** *rel-filter-mono* [*relator-mono*]: *rel-filter* $A \leq$ *rel-filter* $B$ **if** *le*: $A \leq B$
⟨*proof*⟩

**lemma** *rel-filter-conversep*: *rel-filter* $A^{-1-1} = (\textit{rel-filter}\ A)^{-1-1}$
⟨*proof*⟩

**lemma** *rel-filter-distr* [*relator-distr*]:
*rel-filter* $A$ *OO* *rel-filter* $B$ = *rel-filter* $(A\ \textit{OO}\ B)$
⟨*proof*⟩

**lemma** *filtermap-parametric*: $((A ===> B) ===> \textit{rel-filter}\ A ===> \textit{rel-filter}\ B)$ *filtermap filtermap*
⟨*proof*⟩

**lemma** *rel-filter-Grp*: *rel-filter* (*Grp UNIV* $f$) = *Grp UNIV* (*filtermap* $f$)
⟨*proof*⟩

**lemma** *Quotient-filter* [*quot-map*]:
*Quotient* $R$ *Abs Rep* $T \Longrightarrow$ *Quotient* (*rel-filter* $R$) (*filtermap Abs*) (*filtermap Rep*) (*rel-filter* $T$)
⟨*proof*⟩

**lemma** *left-total-rel-filter* [*transfer-rule*]: *left-total* $A \Longrightarrow$ *left-total* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *right-total-rel-filter* [*transfer-rule*]: *right-total* $A \Longrightarrow$ *right-total* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *bi-total-rel-filter* [*transfer-rule*]: *bi-total* $A \Longrightarrow$ *bi-total* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *left-unique-rel-filter* [*transfer-rule*]: *left-unique* $A \Longrightarrow$ *left-unique* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *right-unique-rel-filter* [*transfer-rule*]:
*right-unique* $A \Longrightarrow$ *right-unique* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *bi-unique-rel-filter* [*transfer-rule*]: *bi-unique* $A \Longrightarrow$ *bi-unique* (*rel-filter* $A$)
⟨*proof*⟩

**lemma** *eventually-parametric* [*transfer-rule*]:
(($A$ ===> (=)) ===> *rel-filter* $A$ ===> (=)) *eventually eventually*
⟨*proof*⟩

**lemma** *frequently-parametric* [*transfer-rule*]: (($A$ ===> (=)) ===> *rel-filter* $A$ ===> (=)) *frequently frequently*
⟨*proof*⟩

**lemma** *is-filter-parametric* [*transfer-rule*]:
**assumes** [*transfer-rule*]: *bi-total* $A$
**assumes** [*transfer-rule*]: *bi-unique* $A$
**shows** ((($A$ ===> (=)) ===> (=)) ===> (=)) *is-filter is-filter*
⟨*proof*⟩

**lemma** *top-filter-parametric* [*transfer-rule*]: *rel-filter* $A$ *top top* **if** *bi-total* $A$
⟨*proof*⟩

**lemma** *bot-filter-parametric* [*transfer-rule*]: *rel-filter* $A$ *bot bot*
⟨*proof*⟩

**lemma** *principal-parametric* [*transfer-rule*]: (*rel-set* $A$ ===> *rel-filter* $A$) *principal principal*
⟨*proof*⟩

**lemma** *sup-filter-parametric* [*transfer-rule*]:
(*rel-filter* $A$ ===> *rel-filter* $A$ ===> *rel-filter* $A$) *sup sup*
⟨*proof*⟩

**lemma** *Sup-filter-parametric* [*transfer-rule*]: (*rel-set* (*rel-filter* $A$) ===> *rel-filter* $A$) *Sup Sup*
⟨*proof*⟩

**context**
  **fixes** $A$ :: $'a \Rightarrow 'b \Rightarrow bool$
  **assumes** [*transfer-rule*]: *bi-unique* $A$
**begin**

**lemma** *le-filter-parametric* [*transfer-rule*]:
  (*rel-filter* $A$ ===> *rel-filter* $A$ ===> (=)) (≤) (≤)
⟨*proof*⟩

**lemma** *less-filter-parametric* [*transfer-rule*]:
  (*rel-filter* $A$ ===> *rel-filter* $A$ ===> (=)) (<) (<)
⟨*proof*⟩

**context**
  **assumes** [*transfer-rule*]: *bi-total* $A$
**begin**

**lemma** *Inf-filter-parametric* [*transfer-rule*]:
  (*rel-set* (*rel-filter* $A$) ===> *rel-filter* $A$) *Inf Inf*
⟨*proof*⟩

**lemma** *inf-filter-parametric* [*transfer-rule*]:
  (*rel-filter* $A$ ===> *rel-filter* $A$ ===> *rel-filter* $A$) *inf inf*
⟨*proof*⟩

**end**

**end**

**end**

**context**
  **includes** *lifting-syntax*
**begin**

**lemma** *prod-filter-parametric* [*transfer-rule*]:
  (*rel-filter* $R$ ===> *rel-filter* $S$ ===> *rel-filter* (*rel-prod* $R$ $S$)) *prod-filter prod-filter*
⟨*proof*⟩

**end**

Code generation for filters

**definition** *abstract-filter* :: ($unit \Rightarrow 'a$ *filter*) $\Rightarrow 'a$ *filter*
  **where** [*simp*]: *abstract-filter* $f = f$ ()

**code-datatype** *principal abstract-filter*

**hide-const** (**open**) *abstract-filter*

**declare** *filterlim-principal* [*code*]
**declare** *principal-prod-principal* [*code*]
**declare** *filtermap-principal* [*code*]
**declare** *filtercomap-principal* [*code*]
**declare** *eventually-principal* [*code*]
**declare** *inf-principal* [*code*]
**declare** *sup-principal* [*code*]
**declare** *principal-le-iff* [*code*]

**lemma** *Rep-filter-iff-eventually* [*simp*, *code*]:
  *Rep-filter F P* $\longleftrightarrow$ *eventually P F*
  ⟨*proof*⟩

**lemma** *bot-eq-principal-empty* [*code*]:
  *bot* = *principal* {}
  ⟨*proof*⟩

**lemma** *top-eq-principal-UNIV* [*code*]:
  *top* = *principal UNIV*
  ⟨*proof*⟩

**instantiation** *filter* :: (*equal*) *equal*
**begin**

**definition** *equal-filter* :: *'a filter* $\Rightarrow$ *'a filter* $\Rightarrow$ *bool*
  **where** *equal-filter F F'* $\longleftrightarrow$ *F* = *F'*

**lemma** *equal-filter* [*code*]:
  *HOL.equal* (*principal A*) (*principal B*) $\longleftrightarrow$ *A* = *B*
  ⟨*proof*⟩

**instance**
  ⟨*proof*⟩

**end**

**end**

# 91 Conditionally-complete Lattices

**theory** *Conditionally-Complete-Lattices*
**imports** *Finite-Set Lattices-Big Set-Interval*
**begin**

**locale** *preordering-bdd* = *preordering*

**begin**

**definition** *bdd* :: ‹$'a$ *set* ⇒ *bool*›
**where** *unfold*: ‹*bdd* $A \longleftrightarrow (\exists M.\ \forall x \in A.\ x \leq M)$›

**lemma** *empty* [*simp*, *intro*]:
‹*bdd* {}›
⟨*proof*⟩

**lemma** *I* [*intro*]:
‹*bdd* $A$› **if** ‹$\bigwedge x.\ x \in A \Longrightarrow x \leq M$›
⟨*proof*⟩

**lemma** *E*:
**assumes** ‹*bdd* $A$›
**obtains** $M$ **where** ‹$\bigwedge x.\ x \in A \Longrightarrow x \leq M$›
⟨*proof*⟩

**lemma** *I2*:
‹*bdd* ($f$ ‘ $A$)› **if** ‹$\bigwedge x.\ x \in A \Longrightarrow f\, x \leq M$›
⟨*proof*⟩

**lemma** *mono*:
‹*bdd* $A$› **if** ‹*bdd* $B$› ‹$A \subseteq B$›
⟨*proof*⟩

**lemma** *Int1* [*simp*]:
‹*bdd* $(A \cap B)$› **if** ‹*bdd* $A$›
⟨*proof*⟩

**lemma** *Int2* [*simp*]:
‹*bdd* $(A \cap B)$› **if** ‹*bdd* $B$›
⟨*proof*⟩

**end**

## 91.1 Preorders

**context** *preorder*
**begin**

**sublocale** *bdd-above*: *preordering-bdd* ‹($\leq$)› ‹($<$)›
**defines** *bdd-above-primitive-def*: *bdd-above* = *bdd-above.bdd* ⟨*proof*⟩

**sublocale** *bdd-below*: *preordering-bdd* ‹($\geq$)› ‹($>$)›
**defines** *bdd-below-primitive-def*: *bdd-below* = *bdd-below.bdd* ⟨*proof*⟩

**lemma** *bdd-above-def*: ‹*bdd-above* $A \longleftrightarrow (\exists M.\ \forall x \in A.\ x \leq M)$›
⟨*proof*⟩

**lemma** *bdd-below-def*: ‹*bdd-below* $A \longleftrightarrow (\exists M.\ \forall x \in A.\ M \leq x)$›
⟨*proof*⟩

**lemma** *bdd-aboveI*: $(\bigwedge x.\ x \in A \Longrightarrow x \leq M) \Longrightarrow$ *bdd-above* $A$
⟨*proof*⟩

**lemma** *bdd-belowI*: $(\bigwedge x.\ x \in A \Longrightarrow m \leq x) \Longrightarrow$ *bdd-below* $A$
⟨*proof*⟩

**lemma** *bdd-aboveI2*: $(\bigwedge x.\ x \in A \Longrightarrow f\,x \leq M) \Longrightarrow$ *bdd-above* $(f`A)$
⟨*proof*⟩

**lemma** *bdd-belowI2*: $(\bigwedge x.\ x \in A \Longrightarrow m \leq f\,x) \Longrightarrow$ *bdd-below* $(f`A)$
⟨*proof*⟩

**lemma** *bdd-above-empty*: *bdd-above* $\{\}$
⟨*proof*⟩

**lemma** *bdd-below-empty*: *bdd-below* $\{\}$
⟨*proof*⟩

**lemma** *bdd-above-mono*: *bdd-above* $B \Longrightarrow A \subseteq B \Longrightarrow$ *bdd-above* $A$
⟨*proof*⟩

**lemma** *bdd-below-mono*: *bdd-below* $B \Longrightarrow A \subseteq B \Longrightarrow$ *bdd-below* $A$
⟨*proof*⟩

**lemma** *bdd-above-Int1*: *bdd-above* $A \Longrightarrow$ *bdd-above* $(A \cap B)$
⟨*proof*⟩

**lemma** *bdd-above-Int2*: *bdd-above* $B \Longrightarrow$ *bdd-above* $(A \cap B)$
⟨*proof*⟩

**lemma** *bdd-below-Int1*: *bdd-below* $A \Longrightarrow$ *bdd-below* $(A \cap B)$
⟨*proof*⟩

**lemma** *bdd-below-Int2*: *bdd-below* $B \Longrightarrow$ *bdd-below* $(A \cap B)$
⟨*proof*⟩

**lemma** *bdd-above-Ioo* [*simp*, *intro*]: *bdd-above* $\{a <..< b\}$
⟨*proof*⟩

**lemma** *bdd-above-Ico* [*simp*, *intro*]: *bdd-above* $\{a ..< b\}$
⟨*proof*⟩

**lemma** *bdd-above-Iio* [*simp*, *intro*]: *bdd-above* $\{..< b\}$
⟨*proof*⟩

**lemma** *bdd-above-Ioc* [*simp*, *intro*]: *bdd-above* $\{a <.. b\}$
⟨*proof*⟩

**lemma** *bdd-above-Icc* [*simp*, *intro*]: *bdd-above* $\{a .. b\}$
⟨*proof*⟩

**lemma** *bdd-above-Iic* [*simp*, *intro*]: *bdd-above* $\{.. b\}$
⟨*proof*⟩

**lemma** *bdd-below-Ioo* [*simp*, *intro*]: *bdd-below* $\{a <..< b\}$
⟨*proof*⟩

**lemma** *bdd-below-Ioc* [*simp*, *intro*]: *bdd-below* $\{a <.. b\}$
⟨*proof*⟩

**lemma** *bdd-below-Ioi* [*simp*, *intro*]: *bdd-below* $\{a <..\}$
⟨*proof*⟩

**lemma** *bdd-below-Ico* [*simp*, *intro*]: *bdd-below* $\{a ..< b\}$
⟨*proof*⟩

**lemma** *bdd-below-Icc* [*simp*, *intro*]: *bdd-below* $\{a .. b\}$
⟨*proof*⟩

**lemma** *bdd-below-Ici* [*simp*, *intro*]: *bdd-below* $\{a ..\}$
⟨*proof*⟩

**end**

**context** *order-top*
**begin**

**lemma** *bdd-above-top* [*simp*, *intro*!]: *bdd-above* $A$
⟨*proof*⟩

**end**

**context** *order-bot*
**begin**

**lemma** *bdd-below-bot* [*simp*, *intro*!]: *bdd-below* $A$
⟨*proof*⟩

**end**

**lemma** *bdd-above-image-mono*: *mono* $f \Longrightarrow$ *bdd-above* $A \Longrightarrow$ *bdd-above* $(f`A)$
⟨*proof*⟩

**lemma** *bdd-below-image-mono*: *mono* $f \Longrightarrow$ *bdd-below* $A \Longrightarrow$ *bdd-below* $(f`A)$

⟨*proof*⟩

**lemma** *bdd-above-image-antimono*: *antimono f* $\Longrightarrow$ *bdd-below A* $\Longrightarrow$ *bdd-above* $(f`A)$
⟨*proof*⟩

**lemma** *bdd-below-image-antimono*: *antimono f* $\Longrightarrow$ *bdd-above A* $\Longrightarrow$ *bdd-below* $(f`A)$
⟨*proof*⟩

**lemma**
**fixes** $X$ :: *'a::ordered-ab-group-add set*
**shows** *bdd-above-uminus*[*simp*]: *bdd-above* (*uminus* ` $X$) $\longleftrightarrow$ *bdd-below X*
**and** *bdd-below-uminus*[*simp*]: *bdd-below* (*uminus* ` $X$) $\longleftrightarrow$ *bdd-above X*
⟨*proof*⟩

## 91.2 Lattices

**context** *lattice*
**begin**

**lemma** *bdd-above-insert* [*simp*]: *bdd-above* (*insert a A*) = *bdd-above A*
⟨*proof*⟩

**lemma** *bdd-below-insert* [*simp*]: *bdd-below* (*insert a A*) = *bdd-below A*
⟨*proof*⟩

**lemma** *bdd-finite* [*simp*]:
**assumes** *finite A* **shows** *bdd-above-finite*: *bdd-above A* **and** *bdd-below-finite*: *bdd-below A*
⟨*proof*⟩

**lemma** *bdd-above-Un* [*simp*]: *bdd-above* $(A \cup B)$ = (*bdd-above A* $\wedge$ *bdd-above B*)
⟨*proof*⟩

**lemma** *bdd-below-Un* [*simp*]: *bdd-below* $(A \cup B)$ = (*bdd-below A* $\wedge$ *bdd-below B*)
⟨*proof*⟩

**lemma** *bdd-above-image-sup*[*simp*]:
*bdd-above* (($\lambda x$. *sup* $(f\ x)$ $(g\ x)$) ` $A$) $\longleftrightarrow$ *bdd-above* $(f`A)$ $\wedge$ *bdd-above* $(g`A)$
⟨*proof*⟩

**lemma** *bdd-below-image-inf*[*simp*]:
*bdd-below* (($\lambda x$. *inf* $(f\ x)$ $(g\ x)$) ` $A$) $\longleftrightarrow$ *bdd-below* $(f`A)$ $\wedge$ *bdd-below* $(g`A)$
⟨*proof*⟩

**lemma** *bdd-below-UN*[*simp*]: *finite I* $\Longrightarrow$ *bdd-below* $(\bigcup i{\in}I.\ A\ i)$ = $(\forall\ i \in I.$ *bdd-below* $(A\ i))$
⟨*proof*⟩

**lemma** *bdd-above-UN*[*simp*]: *finite* $I \Longrightarrow$ *bdd-above* $(\bigcup i{\in}I.\ A\ i) = (\forall\, i \in I.$ *bdd-above* $(A\ i))$
⟨*proof*⟩

**end**

To avoid name classes with the *complete-lattice*-class we prefix *Sup* and *Inf* in theorem names with c.

## 91.3 Conditionally complete lattices

**class** *conditionally-complete-lattice* = *lattice* + *Sup* + *Inf* +
**assumes** *cInf-lower*: $x \in X \Longrightarrow$ *bdd-below* $X \Longrightarrow$ *Inf* $X \leq x$
**and** *cInf-greatest*: $X \neq \{\} \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow z \leq x) \Longrightarrow z \leq$ *Inf* $X$
**assumes** *cSup-upper*: $x \in X \Longrightarrow$ *bdd-above* $X \Longrightarrow x \leq$ *Sup* $X$
**and** *cSup-least*: $X \neq \{\} \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow x \leq z) \Longrightarrow$ *Sup* $X \leq z$
**begin**

**lemma** *cSup-upper2*: $x \in X \Longrightarrow y \leq x \Longrightarrow$ *bdd-above* $X \Longrightarrow y \leq$ *Sup* $X$
⟨*proof*⟩

**lemma** *cInf-lower2*: $x \in X \Longrightarrow x \leq y \Longrightarrow$ *bdd-below* $X \Longrightarrow$ *Inf* $X \leq y$
⟨*proof*⟩

**lemma** *cSup-mono*: $B \neq \{\} \Longrightarrow$ *bdd-above* $A \Longrightarrow (\bigwedge b.\ b \in B \Longrightarrow \exists\, a{\in}A.\ b \leq a)$ $\Longrightarrow$ *Sup* $B \leq$ *Sup* $A$
⟨*proof*⟩

**lemma** *cInf-mono*: $B \neq \{\} \Longrightarrow$ *bdd-below* $A \Longrightarrow (\bigwedge b.\ b \in B \Longrightarrow \exists\, a{\in}A.\ a \leq b)$ $\Longrightarrow$ *Inf* $A \leq$ *Inf* $B$
⟨*proof*⟩

**lemma** *cSup-subset-mono*: $A \neq \{\} \Longrightarrow$ *bdd-above* $B \Longrightarrow A \subseteq B \Longrightarrow$ *Sup* $A \leq$ *Sup* $B$
⟨*proof*⟩

**lemma** *cInf-superset-mono*: $A \neq \{\} \Longrightarrow$ *bdd-below* $B \Longrightarrow A \subseteq B \Longrightarrow$ *Inf* $B \leq$ *Inf* $A$
⟨*proof*⟩

**lemma** *cSup-eq-maximum*: $z \in X \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow x \leq z) \Longrightarrow$ *Sup* $X = z$
⟨*proof*⟩

**lemma** *cInf-eq-minimum*: $z \in X \Longrightarrow (\bigwedge x.\ x \in X \Longrightarrow z \leq x) \Longrightarrow$ *Inf* $X = z$
⟨*proof*⟩

**lemma** *cSup-le-iff*: $S \neq \{\} \Longrightarrow$ *bdd-above* $S \Longrightarrow$ *Sup* $S \leq a \longleftrightarrow (\forall\, x{\in}S.\ x \leq a)$
⟨*proof*⟩

**lemma** *le-cInf-iff*: $S \neq \{\} \Longrightarrow$ *bdd-below* $S \Longrightarrow a \leq$ *Inf* $S \longleftrightarrow (\forall x{\in}S.\ a \leq x)$
⟨*proof*⟩

**lemma** *cSup-eq-non-empty*:
**assumes** *1*: $X \neq \{\}$
**assumes** *2*: $\bigwedge x.\ x \in X \Longrightarrow x \leq a$
**assumes** *3*: $\bigwedge y.\ (\bigwedge x.\ x \in X \Longrightarrow x \leq y) \Longrightarrow a \leq y$
**shows** *Sup* $X = a$
⟨*proof*⟩

**lemma** *cInf-eq-non-empty*:
**assumes** *1*: $X \neq \{\}$
**assumes** *2*: $\bigwedge x.\ x \in X \Longrightarrow a \leq x$
**assumes** *3*: $\bigwedge y.\ (\bigwedge x.\ x \in X \Longrightarrow y \leq x) \Longrightarrow y \leq a$
**shows** *Inf* $X = a$
⟨*proof*⟩

**lemma** *cInf-cSup*: $S \neq \{\} \Longrightarrow$ *bdd-below* $S \Longrightarrow$ *Inf* $S =$ *Sup* $\{x.\ \forall s{\in}S.\ x \leq s\}$
⟨*proof*⟩

**lemma** *cSup-cInf*: $S \neq \{\} \Longrightarrow$ *bdd-above* $S \Longrightarrow$ *Sup* $S =$ *Inf* $\{x.\ \forall s{\in}S.\ s \leq x\}$
⟨*proof*⟩

**lemma** *cSup-insert*: $X \neq \{\} \Longrightarrow$ *bdd-above* $X \Longrightarrow$ *Sup* (*insert* $a$ $X$) $=$ *sup* $a$ (*Sup* $X$)
⟨*proof*⟩

**lemma** *cInf-insert*: $X \neq \{\} \Longrightarrow$ *bdd-below* $X \Longrightarrow$ *Inf* (*insert* $a$ $X$) $=$ *inf* $a$ (*Inf* $X$)
⟨*proof*⟩

**lemma** *cSup-singleton* [*simp*]: *Sup* $\{x\} = x$
⟨*proof*⟩

**lemma** *cInf-singleton* [*simp*]: *Inf* $\{x\} = x$
⟨*proof*⟩

**lemma** *cSup-insert-If*: *bdd-above* $X \Longrightarrow$ *Sup* (*insert* $a$ $X$) $=$ (*if* $X = \{\}$ *then* $a$ *else sup* $a$ (*Sup* $X$))
⟨*proof*⟩

**lemma** *cInf-insert-If*: *bdd-below* $X \Longrightarrow$ *Inf* (*insert* $a$ $X$) $=$ (*if* $X = \{\}$ *then* $a$ *else inf* $a$ (*Inf* $X$))
⟨*proof*⟩

**lemma** *le-cSup-finite*: *finite* $X \Longrightarrow x \in X \Longrightarrow x \leq$ *Sup* $X$
⟨*proof*⟩

**lemma** *cInf-le-finite*: *finite* $X \Longrightarrow x \in X \Longrightarrow$ *Inf* $X \leq x$

⟨*proof*⟩

**lemma** *cSup-eq-Sup-fin*: $\mathit{finite}\ X \Longrightarrow X \neq \{\} \Longrightarrow \mathit{Sup}\ X = \mathit{Sup\text{-}fin}\ X$
⟨*proof*⟩

**lemma** *cInf-eq-Inf-fin*: $\mathit{finite}\ X \Longrightarrow X \neq \{\} \Longrightarrow \mathit{Inf}\ X = \mathit{Inf\text{-}fin}\ X$
⟨*proof*⟩

**lemma** *cSup-atMost*[*simp*]: $\mathit{Sup}\ \{..x\} = x$
⟨*proof*⟩

**lemma** *cSup-greaterThanAtMost*[*simp*]: $y < x \Longrightarrow \mathit{Sup}\ \{y{<}..x\} = x$
⟨*proof*⟩

**lemma** *cSup-atLeastAtMost*[*simp*]: $y \leq x \Longrightarrow \mathit{Sup}\ \{y..x\} = x$
⟨*proof*⟩

**lemma** *cInf-atLeast*[*simp*]: $\mathit{Inf}\ \{x..\} = x$
⟨*proof*⟩

**lemma** *cInf-atLeastLessThan*[*simp*]: $y < x \Longrightarrow \mathit{Inf}\ \{y..{<}x\} = y$
⟨*proof*⟩

**lemma** *cInf-atLeastAtMost*[*simp*]: $y \leq x \Longrightarrow \mathit{Inf}\ \{y..x\} = y$
⟨*proof*⟩

**lemma** *cINF-lower*: $\mathit{bdd\text{-}below}\ (f\ `\ A) \Longrightarrow x \in A \Longrightarrow \bigsqcap(f\ `\ A) \leq f\ x$
⟨*proof*⟩

**lemma** *cINF-greatest*: $A \neq \{\} \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow m \leq f\ x) \Longrightarrow m \leq \bigsqcap(f\ `\ A)$
⟨*proof*⟩

**lemma** *cSUP-upper*: $x \in A \Longrightarrow \mathit{bdd\text{-}above}\ (f\ `\ A) \Longrightarrow f\ x \leq \bigsqcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *cSUP-least*: $A \neq \{\} \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\ x \leq M) \Longrightarrow \bigsqcup(f\ `\ A) \leq M$
⟨*proof*⟩

**lemma** *cINF-lower2*: $\mathit{bdd\text{-}below}\ (f\ `\ A) \Longrightarrow x \in A \Longrightarrow f\ x \leq u \Longrightarrow \bigsqcap(f\ `\ A) \leq u$
⟨*proof*⟩

**lemma** *cSUP-upper2*: $\mathit{bdd\text{-}above}\ (f\ `\ A) \Longrightarrow x \in A \Longrightarrow u \leq f\ x \Longrightarrow u \leq \bigsqcup(f\ `\ A)$
⟨*proof*⟩

**lemma** *cSUP-const* [*simp*]: $A \neq \{\} \Longrightarrow (\bigsqcup x{\in}A.\ c) = c$
⟨*proof*⟩

**lemma** *cINF-const* [*simp*]: $A \neq \{\} \Longrightarrow (\bigsqcap x{\in}A.\ c) = c$
⟨*proof*⟩

**lemma** *le-cINF-iff*: $A \neq \{\} \Longrightarrow$ *bdd-below* $(f\ `\ A) \Longrightarrow u \leq \bigsqcap(f\ `\ A) \longleftrightarrow (\forall\, x{\in}A.\ u \leq f\ x)$
⟨*proof*⟩

**lemma** *cSUP-le-iff*: $A \neq \{\} \Longrightarrow$ *bdd-above* $(f\ `\ A) \Longrightarrow \bigsqcup(f\ `\ A) \leq u \longleftrightarrow (\forall\, x{\in}A.\ f\ x \leq u)$
⟨*proof*⟩

**lemma** *less-cINF-D*: *bdd-below* $(f`A) \Longrightarrow y < (\bigsqcap i{\in}A.\ f\ i) \Longrightarrow i \in A \Longrightarrow y < f\ i$
⟨*proof*⟩

**lemma** *cSUP-lessD*: *bdd-above* $(f`A) \Longrightarrow (\bigsqcup i{\in}A.\ f\ i) < y \Longrightarrow i \in A \Longrightarrow f\ i < y$
⟨*proof*⟩

**lemma** *cINF-insert*: $A \neq \{\} \Longrightarrow$ *bdd-below* $(f\ `\ A) \Longrightarrow \bigsqcap(f\ `\ \mathit{insert}\ a\ A) = \mathit{inf}\ (f\ a)\ (\bigsqcap(f\ `\ A))$
⟨*proof*⟩

**lemma** *cSUP-insert*: $A \neq \{\} \Longrightarrow$ *bdd-above* $(f\ `\ A) \Longrightarrow \bigsqcup(f\ `\ \mathit{insert}\ a\ A) = \mathit{sup}\ (f\ a)\ (\bigsqcup(f\ `\ A))$
⟨*proof*⟩

**lemma** *cINF-mono*: $B \neq \{\} \Longrightarrow$ *bdd-below* $(f\ `\ A) \Longrightarrow (\bigwedge m.\ m \in B \Longrightarrow \exists\, n{\in}A.\ f\ n \leq g\ m) \Longrightarrow \bigsqcap(f\ `\ A) \leq \bigsqcap(g\ `\ B)$
⟨*proof*⟩

**lemma** *cSUP-mono*: $A \neq \{\} \Longrightarrow$ *bdd-above* $(g\ `\ B) \Longrightarrow (\bigwedge n.\ n \in A \Longrightarrow \exists\, m{\in}B.\ f\ n \leq g\ m) \Longrightarrow \bigsqcup(f\ `\ A) \leq \bigsqcup(g\ `\ B)$
⟨*proof*⟩

**lemma** *cINF-superset-mono*: $A \neq \{\} \Longrightarrow$ *bdd-below* $(g\ `\ B) \Longrightarrow A \subseteq B \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow g\ x \leq f\ x) \Longrightarrow \bigsqcap(g\ `\ B) \leq \bigsqcap(f\ `\ A)$
⟨*proof*⟩

**lemma** *cSUP-subset-mono*:
$[\![A \neq \{\};$ *bdd-above* $(g\ `\ B);\ A \subseteq B;\ \bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x]\!] \Longrightarrow \bigsqcup\ (f\ `\ A) \leq \bigsqcup\ (g\ `\ B)$
⟨*proof*⟩

**lemma** *less-eq-cInf-inter*: *bdd-below* $A \Longrightarrow$ *bdd-below* $B \Longrightarrow A \cap B \neq \{\} \Longrightarrow \mathit{inf}\ (\mathit{Inf}\ A)\ (\mathit{Inf}\ B) \leq \mathit{Inf}\ (A \cap B)$
⟨*proof*⟩

**lemma** *cSup-inter-less-eq*: *bdd-above* $A \Longrightarrow$ *bdd-above* $B \Longrightarrow A \cap B \neq \{\} \Longrightarrow \mathit{Sup}\ (A \cap B) \leq \mathit{sup}\ (\mathit{Sup}\ A)\ (\mathit{Sup}\ B)$
⟨*proof*⟩

**lemma** *cInf-union-distrib*: $A \neq \{\} \Longrightarrow$ *bdd-below* $A \Longrightarrow B \neq \{\} \Longrightarrow$ *bdd-below* $B$

$\Longrightarrow$ *Inf* $(A \cup B)$ = *inf* (*Inf* $A$) (*Inf* $B$)
⟨*proof*⟩

**lemma** *cINF-union*: $A \neq \{\} \Longrightarrow$ *bdd-below* $(f$ ‘ $A) \Longrightarrow B \neq \{\} \Longrightarrow$ *bdd-below* $(f$ ‘ $B) \Longrightarrow \bigsqcap (f$ ‘ $(A \cup B)) = \bigsqcap (f$ ‘ $A) \sqcap \bigsqcap (f$ ‘ $B)$
⟨*proof*⟩

**lemma** *cSup-union-distrib*: $A \neq \{\} \Longrightarrow$ *bdd-above* $A \Longrightarrow B \neq \{\} \Longrightarrow$ *bdd-above* $B \Longrightarrow$ *Sup* $(A \cup B)$ = *sup* (*Sup* $A$) (*Sup* $B$)
⟨*proof*⟩

**lemma** *cSUP-union*: $A \neq \{\} \Longrightarrow$ *bdd-above* $(f$ ‘ $A) \Longrightarrow B \neq \{\} \Longrightarrow$ *bdd-above* $(f$ ‘ $B) \Longrightarrow \bigsqcup (f$ ‘ $(A \cup B)) = \bigsqcup (f$ ‘ $A) \sqcup \bigsqcup (f$ ‘ $B)$
⟨*proof*⟩

**lemma** *cINF-inf-distrib*: $A \neq \{\} \Longrightarrow$ *bdd-below* $(f$‘$A) \Longrightarrow$ *bdd-below* $(g$‘$A) \Longrightarrow \bigsqcap (f$ ‘ $A) \sqcap \bigsqcap (g$ ‘ $A) = (\bigsqcap a{\in}A.$ *inf* $(f\ a)\ (g\ a))$
⟨*proof*⟩

**lemma** *SUP-sup-distrib*: $A \neq \{\} \Longrightarrow$ *bdd-above* $(f$‘$A) \Longrightarrow$ *bdd-above* $(g$‘$A) \Longrightarrow \bigsqcup (f$ ‘ $A) \sqcup \bigsqcup (g$ ‘ $A) = (\bigsqcup a{\in}A.$ *sup* $(f\ a)\ (g\ a))$
⟨*proof*⟩

**lemma** *cInf-le-cSup*:
$A \neq \{\} \Longrightarrow$ *bdd-above* $A \Longrightarrow$ *bdd-below* $A \Longrightarrow$ *Inf* $A \leq$ *Sup* $A$
⟨*proof*⟩

**context**
**fixes** $f :: 'a \Rightarrow 'b$::*conditionally-complete-lattice*
**assumes** *mono* $f$
**begin**

**lemma** *mono-cInf*: ⟦*bdd-below* $A$; $A{\neq}\{\}$⟧ $\Longrightarrow f$ (*Inf* $A$) $\leq$ (*INF* $x{\in}A.\ f\ x$)
⟨*proof*⟩

**lemma** *mono-cSup*: ⟦*bdd-above* $A$; $A{\neq}\{\}$⟧ $\Longrightarrow$ (*SUP* $x{\in}A.\ f\ x$) $\leq f$ (*Sup* $A$)
⟨*proof*⟩

**lemma** *mono-cINF*: ⟦*bdd-below* $(A$‘$I)$; $I{\neq}\{\}$⟧ $\Longrightarrow f$ (*INF* $i{\in}I.\ A\ i$) $\leq$ (*INF* $x{\in}I.\ f\ (A\ x)$)
⟨*proof*⟩

**lemma** *mono-cSUP*: ⟦*bdd-above* $(A$‘$I)$; $I{\neq}\{\}$⟧ $\Longrightarrow$ (*SUP* $x{\in}I.\ f\ (A\ x)$) $\leq f$ (*SUP* $i{\in}I.\ A\ i$)
⟨*proof*⟩

**end**

**end**

The special case of well-orderings

**lemma** *wellorder-InfI*:
**fixes** $k$ :: $'a$::{*wellorder*,*conditionally-complete-lattice*}
**assumes** $k \in A$ **shows** $Inf\ A \in A$
⟨*proof*⟩

**lemma** *wellorder-Inf-le1*:
**fixes** $k$ :: $'a$::{*wellorder*,*conditionally-complete-lattice*}
**assumes** $k \in A$ **shows** $Inf\ A \leq k$
⟨*proof*⟩

## 91.4 Complete lattices

**instance** *complete-lattice* ⊆ *conditionally-complete-lattice*
⟨*proof*⟩

**lemma** *cSup-eq*:
**fixes** $a$ :: $'a$ :: {*conditionally-complete-lattice*, *no-bot*}
**assumes** *upper*: $\bigwedge x.\ x \in X \Longrightarrow x \leq a$
**assumes** *least*: $\bigwedge y.\ (\bigwedge x.\ x \in X \Longrightarrow x \leq y) \Longrightarrow a \leq y$
**shows** $Sup\ X = a$
⟨*proof*⟩

**lemma** *cSup-unique*:
**fixes** $b$ :: $'a$ :: {*conditionally-complete-lattice*, *no-bot*}
**assumes** $\bigwedge c.\ (\forall x \in s.\ x \leq c) \longleftrightarrow b \leq c$
**shows** $Sup\ s = b$
⟨*proof*⟩

**lemma** *cInf-eq*:
**fixes** $a$ :: $'a$ :: {*conditionally-complete-lattice*, *no-top*}
**assumes** *upper*: $\bigwedge x.\ x \in X \Longrightarrow a \leq x$
**assumes** *least*: $\bigwedge y.\ (\bigwedge x.\ x \in X \Longrightarrow y \leq x) \Longrightarrow y \leq a$
**shows** $Inf\ X = a$
⟨*proof*⟩

**lemma** *cInf-unique*:
**fixes** $b$ :: $'a$ :: {*conditionally-complete-lattice*, *no-top*}
**assumes** $\bigwedge c.\ (\forall x \in s.\ x \geq c) \longleftrightarrow b \geq c$
**shows** $Inf\ s = b$
⟨*proof*⟩

**class** *conditionally-complete-linorder* = *conditionally-complete-lattice* + *linorder*
**begin**

**lemma** *less-cSup-iff*:
$X \neq \{\} \Longrightarrow bdd\text{-}above\ X \Longrightarrow y < Sup\ X \longleftrightarrow (\exists x \in X.\ y < x)$
⟨*proof*⟩

**lemma** *cInf-less-iff*: $X \neq \{\} \Longrightarrow bdd\text{-}below\ X \Longrightarrow Inf\ X < y \longleftrightarrow (\exists x \in X.\ x < y)$
⟨*proof*⟩

**lemma** *cINF-less-iff*: $A \neq \{\} \Longrightarrow bdd\text{-}below\ (f`A) \Longrightarrow (\bigsqcap i \in A.\ f\ i) < a \longleftrightarrow (\exists x \in A.\ f\ x < a)$
⟨*proof*⟩

**lemma** *less-cSUP-iff*: $A \neq \{\} \Longrightarrow bdd\text{-}above\ (f`A) \Longrightarrow a < (\bigsqcup i \in A.\ f\ i) \longleftrightarrow (\exists x \in A.\ a < f\ x)$
⟨*proof*⟩

**lemma** *less-cSupE*:
**assumes** $y < Sup\ X\ X \neq \{\}$ **obtains** $x$ **where** $x \in X\ y < x$
⟨*proof*⟩

**lemma** *less-cSupD*:
$X \neq \{\} \Longrightarrow z < Sup\ X \Longrightarrow \exists x \in X.\ z < x$
⟨*proof*⟩

**lemma** *cInf-lessD*:
$X \neq \{\} \Longrightarrow Inf\ X < z \Longrightarrow \exists x \in X.\ x < z$
⟨*proof*⟩

**lemma** *complete-interval*:
**assumes** $a < b$ **and** $P\ a$ **and** $\neg\ P\ b$
**shows** $\exists c.\ a \leq c \wedge c \leq b \wedge (\forall x.\ a \leq x \wedge x < c \longrightarrow P\ x) \wedge$
$(\forall d.\ (\forall x.\ a \leq x \wedge x < d \longrightarrow P\ x) \longrightarrow d \leq c)$
⟨*proof*⟩

**end**

## 91.5 Instances

**instance** *complete-linorder* < *conditionally-complete-linorder*
⟨*proof*⟩

**lemma** *cSup-eq-Max*: $finite\ (X::'a::conditionally\text{-}complete\text{-}linorder\ set) \Longrightarrow X \neq \{\} \Longrightarrow Sup\ X = Max\ X$
⟨*proof*⟩

**lemma** *cInf-eq-Min*: $finite\ (X::'a::conditionally\text{-}complete\text{-}linorder\ set) \Longrightarrow X \neq \{\} \Longrightarrow Inf\ X = Min\ X$
⟨*proof*⟩

**lemma** *cSup-lessThan*[*simp*]: $Sup\ \{..<x::'a::\{conditionally\text{-}complete\text{-}linorder, no\text{-}bot, dense\text{-}linorder\}\} = x$
⟨*proof*⟩

**lemma** *cSup-greaterThanLessThan*[*simp*]: $y < x \Longrightarrow Sup\ \{y<..<x::'a::\{conditionally\text{-}complete\text{-}linorder,$

*dense-linorder*}} = *x*
⟨*proof*⟩

**lemma** *cSup-atLeastLessThan*[*simp*]: $y < x \Longrightarrow$ *Sup* {*y..<x::'a::*{*conditionally-complete-linorder*, *dense-linorder*}} = *x*
⟨*proof*⟩

**lemma** *cInf-greaterThan*[*simp*]: *Inf* {*x::'a::*{*conditionally-complete-linorder*, *no-top*, *dense-linorder*} <..} = *x*
⟨*proof*⟩

**lemma** *cInf-greaterThanAtMost*[*simp*]: $y < x \Longrightarrow$ *Inf* {*y<..x::'a::*{*conditionally-complete-linorder*, *dense-linorder*}} = *y*
⟨*proof*⟩

**lemma** *cInf-greaterThanLessThan*[*simp*]: $y < x \Longrightarrow$ *Inf* {*y<..<x::'a::*{*conditionally-complete-linorder*, *dense-linorder*}} = *y*
⟨*proof*⟩

**lemma** *Sup-inverse-eq-inverse-Inf*:
**fixes** *f::'b ⇒ 'a::*{*conditionally-complete-linorder*,*linordered-field*}
**assumes** *bdd-above* (*range f*) $L > 0$ **and** *geL*: ⋀*x. f x* ≥ *L*
**shows** (*SUP x. 1 / f x*) = *1* / (*INF x. f x*)
⟨*proof*⟩

**lemma** *Inf-inverse-eq-inverse-Sup*:
**fixes** *f::'b ⇒ 'a::*{*conditionally-complete-linorder*,*linordered-field*}
**assumes** *bdd-above* (*range f*) $L > 0$ **and** *geL*: ⋀*x. f x* ≥ *L*
**shows** (*INF x. 1 / f x*) = *1* / (*SUP x. f x*)
⟨*proof*⟩

**lemma** *Inf-insert-finite*:
**fixes** *S :: 'a::conditionally-complete-linorder set*
**shows** *finite S* ⟹ *Inf* (*insert x S*) = (*if S* = {} *then x else min x* (*Inf S*))
⟨*proof*⟩

**lemma** *Sup-insert-finite*:
**fixes** *S :: 'a::conditionally-complete-linorder set*
**shows** *finite S* ⟹ *Sup* (*insert x S*) = (*if S* = {} *then x else max x* (*Sup S*))
⟨*proof*⟩

**lemma** *finite-imp-less-Inf*:
**fixes** *a :: 'a::conditionally-complete-linorder*
**shows** ⟦*finite X*; *x* ∈ *X*; ⋀*x. x*∈*X* ⟹ *a* < *x*⟧ ⟹ *a* < *Inf X*
⟨*proof*⟩

**lemma** *finite-less-Inf-iff*:
**fixes** *a :: 'a :: conditionally-complete-linorder*
**shows** ⟦*finite X*; *X* ≠ {}⟧ ⟹ *a* < *Inf X* ⟷ (∀ *x* ∈ *X*. *a* < *x*)

⟨*proof*⟩

**lemma** *finite-imp-Sup-less*:
**fixes** $a :: 'a::conditionally\text{-}complete\text{-}linorder$
**shows** $[\![finite\ X;\ x \in X;\ \bigwedge x.\ x{\in}X \Longrightarrow a > x]\!] \Longrightarrow a > Sup\ X$
⟨*proof*⟩

**lemma** *finite-Sup-less-iff*:
**fixes** $a :: 'a :: conditionally\text{-}complete\text{-}linorder$
**shows** $[\![finite\ X;\ X \neq \{\}]\!] \Longrightarrow a > Sup\ X \longleftrightarrow (\forall x \in X.\ a > x)$
⟨*proof*⟩

**class** *linear-continuum* = *conditionally-complete-linorder* + *dense-linorder* +
**assumes** *UNIV-not-singleton*: $\exists a\ b::'a.\ a \neq b$
**begin**

**lemma** *ex-gt-or-lt*: $\exists b.\ a < b \lor b < a$
⟨*proof*⟩

**end**

**context**
**fixes** $f::'a \Rightarrow 'b::\{conditionally\text{-}complete\text{-}linorder, ordered\text{-}ab\text{-}group\text{-}add\}$
**begin**

**lemma** *bdd-above-uminus-image*: $bdd\text{-}above\ ((\lambda x.\ -\ f\ x)\ `\ A) \longleftrightarrow bdd\text{-}below\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *bdd-below-uminus-image*: $bdd\text{-}below\ ((\lambda x.\ -\ f\ x)\ `\ A) \longleftrightarrow bdd\text{-}above\ (f\ `\ A)$
⟨*proof*⟩

**lemma** *uminus-cSUP*:
**assumes** $bdd\text{-}above\ (f\ `\ A)\ A \neq \{\}$
**shows** $-\ (SUP\ x{\in}A.\ f\ x) = (INF\ x{\in}A.\ -\ f\ x)$
⟨*proof*⟩

**end**

**context**
**fixes** $f::'a \Rightarrow 'b::\{conditionally\text{-}complete\text{-}linorder, ordered\text{-}ab\text{-}group\text{-}add\}$
**begin**

**lemma** *uminus-cINF*:
**assumes** $bdd\text{-}below\ (f\ `\ A)\ A \neq \{\}$
**shows** $-\ (INF\ x{\in}A.\ f\ x) = (SUP\ x{\in}A.\ -\ f\ x)$
⟨*proof*⟩

**lemma** *Sup-add-eq*:
  **assumes** *bdd-above* $(f\ `\ A)$ $A \neq \{\}$
  **shows** $(SUP\ x{\in}A.\ a + f\ x) = a + (SUP\ x{\in}A.\ f\ x)$ (**is** *?L=?R*)
⟨*proof*⟩

**lemma** *Inf-add-eq*: — you don't get a shorter proof by duality
  **assumes** *bdd-below* $(f\ `\ A)$ $A \neq \{\}$
  **shows** $(INF\ x{\in}A.\ a + f\ x) = a + (INF\ x{\in}A.\ f\ x)$ (**is** *?L=?R*)
⟨*proof*⟩

**end**

**instantiation** *nat* :: *conditionally-complete-linorder*
**begin**

**definition** *Sup* $(X{::}nat\ set) = (if\ X{=}\{\}\ then\ 0\ else\ Max\ X)$
**definition** *Inf* $(X{::}nat\ set) = (LEAST\ n.\ n \in X)$

**lemma** *bdd-above-nat*: *bdd-above* $X \longleftrightarrow$ *finite* $(X{::}nat\ set)$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**lemma** *Inf-nat-def1*:
  **fixes** $K{::}nat\ set$
  **assumes** $K \neq \{\}$
  **shows** $Inf\ K \in K$
⟨*proof*⟩

**lemma** *Sup-nat-empty* [*simp*]: $Sup\ \{\} = (0{::}nat)$
  ⟨*proof*⟩

**instantiation** *int* :: *conditionally-complete-linorder*
**begin**

**definition** *Sup* $(X{::}int\ set) = (THE\ x.\ x \in X \wedge (\forall\ y{\in}X.\ y \leq x))$
**definition** *Inf* $(X{::}int\ set) = -\ (Sup\ (uminus\ `\ X))$

**instance**
⟨*proof*⟩
**end**

**lemma** *interval-cases*:

**fixes** $S$ :: $'a$ :: *conditionally-complete-linorder set*
**assumes** *ivl*: $\bigwedge a\ b\ x.\ a \in S \Longrightarrow b \in S \Longrightarrow a \leq x \Longrightarrow x \leq b \Longrightarrow x \in S$
**shows** $\exists\, a\ b.\ S = \{\} \vee$
$S = UNIV \vee$
$S = \{..{<}b\} \vee$
$S = \{..b\} \vee$
$S = \{a{<}..\} \vee$
$S = \{a..\} \vee$
$S = \{a{<}..{<}b\} \vee$
$S = \{a{<}..b\} \vee$
$S = \{a..{<}b\} \vee$
$S = \{a..b\}$
⟨*proof*⟩

**lemma** *cSUP-eq-cINF-D*:
**fixes** $f$ :: - $\Rightarrow$ $'b$::*conditionally-complete-lattice*
**assumes** *eq*: $(\bigsqcup x{\in}A.\ f\ x) = (\bigsqcap x{\in}A.\ f\ x)$
**and** *bdd*: *bdd-above* ($f$ ` $A$) *bdd-below* ($f$ ` $A$)
**and** *a*: $a \in A$
**shows** $f\ a = (\bigsqcap x{\in}A.\ f\ x)$
⟨*proof*⟩

**lemma** *cSUP-UNION*:
**fixes** $f$ :: - $\Rightarrow$ $'b$::*conditionally-complete-lattice*
**assumes** *ne*: $A \neq \{\}\ \bigwedge x.\ x \in A \Longrightarrow B(x) \neq \{\}$
**and** *bdd-UN*: *bdd-above* ($\bigcup x{\in}A.\ f$ ` $B\ x$)
**shows** $(\bigsqcup z \in \bigcup x{\in}A.\ B\ x.\ f\ z) = (\bigsqcup x{\in}A.\ \bigsqcup z{\in}B\ x.\ f\ z)$
⟨*proof*⟩

**lemma** *cINF-UNION*:
**fixes** $f$ :: - $\Rightarrow$ $'b$::*conditionally-complete-lattice*
**assumes** *ne*: $A \neq \{\}\ \bigwedge x.\ x \in A \Longrightarrow B(x) \neq \{\}$
**and** *bdd-UN*: *bdd-below* ($\bigcup x{\in}A.\ f$ ` $B\ x$)
**shows** $(\bigsqcap z \in \bigcup x{\in}A.\ B\ x.\ f\ z) = (\bigsqcap x{\in}A.\ \bigsqcap z{\in}B\ x.\ f\ z)$
⟨*proof*⟩

**lemma** *cSup-abs-le*:
**fixes** $S$ :: ($'a$::{*linordered-idom,conditionally-complete-linorder*}) *set*
**shows** $S \neq \{\} \Longrightarrow (\bigwedge x.\ x{\in}S \Longrightarrow |x| \leq a) \Longrightarrow |Sup\ S| \leq a$
⟨*proof*⟩

**end**

# 92 Factorial Function, Rising Factorials

**theory** *Factorial*
**imports** *Groups-List*
**begin**

## 92.1 Factorial Function

**context** *semiring-char-0*
**begin**

**definition** *fact* :: $nat \Rightarrow 'a$
  **where** *fact-prod*: $fact\ n = of\text{-}nat\ (\prod \{1..n\})$

**lemma** *fact-prod-Suc*: $fact\ n = of\text{-}nat\ (prod\ Suc\ \{0..<n\})$
  ⟨*proof*⟩

**lemma** *fact-prod-rev*: $fact\ n = of\text{-}nat\ (\prod i = 0..<n.\ n - i)$
⟨*proof*⟩

**lemma** *fact-0* [*simp*]: $fact\ 0 = 1$
  ⟨*proof*⟩

**lemma** *fact-1* [*simp*]: $fact\ 1 = 1$
  ⟨*proof*⟩

**lemma** *fact-Suc-0* [*simp*]: $fact\ (Suc\ 0) = 1$
  ⟨*proof*⟩

**lemma** *fact-Suc* [*simp*]: $fact\ (Suc\ n) = of\text{-}nat\ (Suc\ n) * fact\ n$
  ⟨*proof*⟩

**lemma** *fact-2* [*simp*]: $fact\ 2 = 2$
  ⟨*proof*⟩

**lemma** *fact-split*: $k \le n \Longrightarrow fact\ n = of\text{-}nat\ (prod\ Suc\ \{n - k..<n\}) * fact\ (n - k)$
  ⟨*proof*⟩

**end**

**lemma** *of-nat-fact* [*simp*]: $of\text{-}nat\ (fact\ n) = fact\ n$
  ⟨*proof*⟩

**lemma** *of-int-fact* [*simp*]: $of\text{-}int\ (fact\ n) = fact\ n$
  ⟨*proof*⟩

**lemma** *fact-reduce*: $n > 0 \Longrightarrow fact\ n = of\text{-}nat\ n * fact\ (n - 1)$
  ⟨*proof*⟩

**lemma** *fact-nonzero* [*simp*]: $fact\ n \ne (0::'a::\{semiring\text{-}char\text{-}0,semiring\text{-}no\text{-}zero\text{-}divisors\})$
  ⟨*proof*⟩

**lemma** *fact-mono-nat*: $m \le n \Longrightarrow fact\ m \le (fact\ n :: nat)$
  ⟨*proof*⟩

**lemma** *fact-in-Nats*: $fact\ n \in \mathbb{N}$
⟨*proof*⟩

**lemma** *fact-in-Ints*: $fact\ n \in \mathbb{Z}$
⟨*proof*⟩

**context**
**assumes** *SORT-CONSTRAINT*('*a*::*linordered-semidom*)
**begin**

**lemma** *fact-mono*: $m \leq n \Longrightarrow fact\ m \leq (fact\ n :: 'a)$
⟨*proof*⟩

**lemma** *fact-ge-1* [*simp*]: $fact\ n \geq (1 :: 'a)$
⟨*proof*⟩

**lemma** *fact-gt-zero* [*simp*]: $fact\ n > (0 :: 'a)$
⟨*proof*⟩

**lemma** *fact-ge-zero* [*simp*]: $fact\ n \geq (0 :: 'a)$
⟨*proof*⟩

**lemma** *fact-not-neg* [*simp*]: $\neg\ fact\ n < (0 :: 'a)$
⟨*proof*⟩

**lemma** *fact-le-power*: $fact\ n \leq (of\text{-}nat\ (n\hat{}n) :: 'a)$
⟨*proof*⟩

**end**

**lemma** *fact-less-mono-nat*: $0 < m \Longrightarrow m < n \Longrightarrow fact\ m < (fact\ n :: nat)$
⟨*proof*⟩

**lemma** *fact-less-mono*: $0 < m \Longrightarrow m < n \Longrightarrow fact\ m < (fact\ n :: 'a::linordered\text{-}semidom)$
⟨*proof*⟩

**lemma** *fact-ge-Suc-0-nat* [*simp*]: $fact\ n \geq Suc\ 0$
⟨*proof*⟩

**lemma** *dvd-fact*: $1 \leq m \Longrightarrow m \leq n \Longrightarrow m\ dvd\ fact\ n$
⟨*proof*⟩

**lemma** *fact-ge-self*: $fact\ n \geq n$
⟨*proof*⟩

**lemma** *fact-dvd*: $n \leq m \Longrightarrow fact\ n\ dvd\ (fact\ m :: 'a::linordered\text{-}semidom)$
⟨*proof*⟩

**lemma** *fact-mod*: $m \leq n \Longrightarrow fact\ n\ mod\ (fact\ m :: 'a::\{semidom\text{-}modulo,\ linordered\text{-}semidom\})$

$= 0$
⟨*proof*⟩

**lemma** *fact-eq-fact-times*:
**assumes** $m \geq n$
**shows** $fact\ m = fact\ n * \prod\{Suc\ n..m\}$
⟨*proof*⟩

**lemma** *fact-div-fact*:
**assumes** $m \geq n$
**shows** $fact\ m\ div\ fact\ n = \prod\{n + 1..m\}$
⟨*proof*⟩

**lemma** *fact-num-eq-if*: $fact\ m = (if\ m = 0\ then\ 1\ else\ of\text{-}nat\ m * fact\ (m - 1))$
⟨*proof*⟩

**lemma** *fact-div-fact-le-pow*:
**assumes** $r \leq n$
**shows** $fact\ n\ div\ fact\ (n - r) \leq n\ \hat{}\ r$
⟨*proof*⟩

**lemma** *prod-Suc-fact*: $prod\ Suc\ \{0..<n\} = fact\ n$
⟨*proof*⟩

**lemma** *prod-Suc-Suc-fact*: $prod\ Suc\ \{Suc\ 0..<n\} = fact\ n$
⟨*proof*⟩

**lemma** *fact-numeral*: $fact\ (numeral\ k) = numeral\ k * fact\ (pred\text{-}numeral\ k)$
— Evaluation for specific numerals
⟨*proof*⟩

## 92.2 Pochhammer's symbol: generalized rising factorial

See https://en.wikipedia.org/wiki/Pochhammer_symbol.

**context** *comm-semiring-1*
**begin**

**definition** *pochhammer* :: $'a \Rightarrow nat \Rightarrow 'a$
**where** *pochhammer-prod*: $pochhammer\ a\ n = prod\ (\lambda i.\ a + of\text{-}nat\ i)\ \{0..<n\}$

**lemma** *pochhammer-prod-rev*: $pochhammer\ a\ n = prod\ (\lambda i.\ a + of\text{-}nat\ (n - i))\ \{1..n\}$
⟨*proof*⟩

**lemma** *pochhammer-Suc-prod*: $pochhammer\ a\ (Suc\ n) = prod\ (\lambda i.\ a + of\text{-}nat\ i)\ \{0..n\}$
⟨*proof*⟩

**lemma** *pochhammer-Suc-prod-rev*: $pochhammer\ a\ (Suc\ n) = prod\ (\lambda i.\ a + of\text{-}nat$

$(n - i))$ $\{0..n\}$
⟨proof⟩

**lemma** *pochhammer-0* [*simp*]: *pochhammer a 0 = 1*
⟨proof⟩

**lemma** *pochhammer-1* [*simp*]: *pochhammer a 1 = a*
⟨proof⟩

**lemma** *pochhammer-Suc0* [*simp*]: *pochhammer a (Suc 0) = a*
⟨proof⟩

**lemma** *pochhammer-Suc*: *pochhammer a (Suc n) = pochhammer a n * (a + of-nat n)*
⟨proof⟩

**end**

**lemma** *pochhammer-nonneg*:
**fixes** *x* :: *'a* :: *linordered-semidom*
**shows** $x > 0 \Longrightarrow$ *pochhammer x n* $\geq 0$
⟨proof⟩

**lemma** *pochhammer-pos*:
**fixes** *x* :: *'a* :: *linordered-semidom*
**shows** $x > 0 \Longrightarrow$ *pochhammer x n* $> 0$
⟨proof⟩

**context** *comm-semiring-1*
**begin**

**lemma** *pochhammer-of-nat*: *pochhammer (of-nat x) n = of-nat (pochhammer x n)*
⟨proof⟩

**end**

**context** *comm-ring-1*
**begin**

**lemma** *pochhammer-of-int*: *pochhammer (of-int x) n = of-int (pochhammer x n)*
⟨proof⟩

**end**

**lemma** *pochhammer-rec*: *pochhammer a (Suc n) = a * pochhammer (a + 1) n*
⟨proof⟩

**lemma** *pochhammer-rec'*: *pochhammer z (Suc n) = (z + of-nat n) * pochhammer z n*

⟨*proof*⟩

**lemma** *pochhammer-fact*: *fact n* = *pochhammer 1 n*
⟨*proof*⟩

**lemma** *pochhammer-of-nat-eq-0-lemma*: $k > n \Longrightarrow$ *pochhammer* (− (*of-nat n* :: ′*a*:: *idom*)) *k* = *0*
⟨*proof*⟩

**lemma** *pochhammer-of-nat-eq-0-lemma′*:
**assumes** *kn*: $k \leq n$
**shows** *pochhammer* (− (*of-nat n* :: ′*a*::{*idom*,*ring-char-0*})) *k* ≠ *0*
⟨*proof*⟩

**lemma** *pochhammer-of-nat-eq-0-iff*:
*pochhammer* (− (*of-nat n* :: ′*a*::{*idom*,*ring-char-0*})) *k* = *0* ⟷ $k > n$
(**is** *?l* = *?r*)
⟨*proof*⟩

**lemma** *pochhammer-0-left*:
*pochhammer 0 n* = (*if n = 0 then 1 else 0*)
⟨*proof*⟩

**lemma** *pochhammer-eq-0-iff*: *pochhammer a n* = (*0*::′*a*::*field-char-0*) ⟷ (∃ *k* < *n*. *a* = − *of-nat k*)
⟨*proof*⟩

**lemma** *pochhammer-eq-0-mono*:
*pochhammer a n* = (*0*::′*a*::*field-char-0*) ⟹ $m \geq n$ ⟹ *pochhammer a m* = *0*
⟨*proof*⟩

**lemma** *pochhammer-neq-0-mono*:
*pochhammer a m* ≠ (*0*::′*a*::*field-char-0*) ⟹ $m \geq n$ ⟹ *pochhammer a n* ≠ *0*
⟨*proof*⟩

**lemma** *pochhammer-minus*:
*pochhammer* (− *b*) *k* = ((− *1*) ^ *k* :: ′*a*::*comm-ring-1*) ∗ *pochhammer* (*b* − *of-nat k* + *1*) *k*
⟨*proof*⟩

**lemma** *pochhammer-minus′*:
*pochhammer* (*b* − *of-nat k* + *1*) *k* = ((− *1*) ^ *k* :: ′*a*::*comm-ring-1*) ∗ *pochhammer* (− *b*) *k*
⟨*proof*⟩

**lemma** *pochhammer-same*: *pochhammer* (− *of-nat n*) *n* =
((− *1*) ^ *n* :: ′*a*::{*semiring-char-0*,*comm-ring-1*,*semiring-no-zero-divisors*}) ∗ *fact n*
⟨*proof*⟩

**lemma** *pochhammer-product'*: *pochhammer z* $(n + m)$ = *pochhammer z n* $*$ *pochhammer* $(z +$ *of-nat n*$)$ *m*
⟨*proof*⟩

**lemma** *pochhammer-product*:
$m \le n \Longrightarrow$ *pochhammer z n* = *pochhammer z m* $*$ *pochhammer* $(z +$ *of-nat m*$)$ $(n - m)$
⟨*proof*⟩

**lemma** *pochhammer-times-pochhammer-half*:
**fixes** *z* :: *'a::field-char-0*
**shows** *pochhammer z* (*Suc n*) $*$ *pochhammer* $(z + 1/2)$ (*Suc n*) = ($\prod$ *k*=*0*..*2*$*$*n*+*1*. *z* + *of-nat k* / *2*)
⟨*proof*⟩

**lemma** *pochhammer-double*:
**fixes** *z* :: *'a::field-char-0*
**shows** *pochhammer* $(2 * z)$ $(2 * n)$ = *of-nat* $(2\hat{}(2*n))$ $*$ *pochhammer z n* $*$ *pochhammer* $(z+1/2)$ *n*
⟨*proof*⟩

**lemma** *fact-double*:
*fact* $(2 * n) = (2 \hat{} (2 * n)$ $*$ *pochhammer* $(1 / 2)$ *n* $*$ *fact n* :: *'a::field-char-0*)
⟨*proof*⟩

**lemma** *pochhammer-absorb-comp*: $(r -$ *of-nat k*$)$ $*$ *pochhammer* $(- r)$ $k = r *$ *pochhammer* $(-r + 1)$ *k*
(**is** *?lhs* = *?rhs*)
**for** *r* :: *'a::comm-ring-1*
⟨*proof*⟩

## 92.3 Misc

**lemma** *fact-code* [*code*]:
*fact n* = (*of-nat* (*fold-atLeastAtMost-nat* $((*))$ *2 n 1*) :: *'a::semiring-char-0*)
⟨*proof*⟩

**lemma** *pochhammer-code* [*code*]:
*pochhammer a n* =
(*if n* = *0 then 1*
*else fold-atLeastAtMost-nat* ($\lambda$*n acc.* $(a +$ *of-nat n*$)$ $*$ *acc*) *0* $(n - 1)$ *1*)
⟨*proof*⟩

**end**

# 93 Binomial Coefficients, Binomial Theorem, Inclusion-exclusion Principle

**theory** *Binomial*
**imports** *Presburger Factorial*
**begin**

## 93.1 Binomial coefficients

This development is based on the work of Andy Gordon and Florian Kammueller.

Combinatorial definition

**definition** *binomial* :: *nat* ⇒ *nat* ⇒ *nat*
**where** *binomial n k* = *card* {*K*∈*Pow* {*0..*<*n*}. *card K* = *k*}

**open-bundle** *binomial-syntax*
**begin**
**notation** *binomial* (**infix** ‹*choose*› *64*)
**end**

**lemma** *binomial-right-mono*:
**assumes** $m \leq n$ **shows** $m$ *choose* $k \leq n$ *choose* $k$
⟨*proof*⟩

**theorem** *n-subsets*:
**assumes** *finite A*
**shows** *card* {*B*. $B \subseteq A \land$ *card* $B = k$} = *card A choose k*
⟨*proof*⟩

Recursive characterization

**lemma** *binomial-n-0* [*simp*]: *n choose 0* = *1*
⟨*proof*⟩

**lemma** *binomial-0-Suc* [*simp*]: *0 choose Suc k* = *0*
⟨*proof*⟩

**lemma** *binomial-Suc-Suc* [*simp*]: *Suc n choose Suc k* = (*n choose k*) + (*n choose Suc k*)
⟨*proof*⟩

**lemma** *binomial-eq-0*: $n < k \Longrightarrow n$ *choose* $k = 0$
⟨*proof*⟩

**lemma** *zero-less-binomial*: $k \leq n \Longrightarrow n$ *choose* $k > 0$
⟨*proof*⟩

**lemma** *binomial-eq-0-iff* [*simp*]: *n choose k* = *0* $\longleftrightarrow n < k$

$\langle proof \rangle$

**lemma** *zero-less-binomial-iff* [*simp*]: $n\ choose\ k > 0 \longleftrightarrow k \leq n$
$\langle proof \rangle$

**lemma** *binomial-n-n* [*simp*]: $n\ choose\ n = 1$
$\langle proof \rangle$

**lemma** *binomial-Suc-n* [*simp*]: $Suc\ n\ choose\ n = Suc\ n$
$\langle proof \rangle$

**lemma** *binomial-1* [*simp*]: $n\ choose\ Suc\ 0 = n$
$\langle proof \rangle$

**lemma** *choose-one*: $n\ choose\ 1 = n$ **for** $n :: nat$
$\langle proof \rangle$

**lemma** *choose-reduce-nat*:
$0 < n \Longrightarrow 0 < k \Longrightarrow$
$n\ choose\ k = ((n - 1)\ choose\ (k - 1)) + ((n - 1)\ choose\ k)$
$\langle proof \rangle$

**lemma** *Suc-times-binomial-eq*: $Suc\ n * (n\ choose\ k) = (Suc\ n\ choose\ Suc\ k) * Suc\ k$
$\langle proof \rangle$

**lemma** *binomial-le-pow2*: $n\ choose\ k \leq 2\hat{}n$
$\langle proof \rangle$

The absorption property.

**lemma** *Suc-times-binomial*: $Suc\ k * (Suc\ n\ choose\ Suc\ k) = Suc\ n * (n\ choose\ k)$
$\langle proof \rangle$

This is the well-known version of absorption, but it's harder to use because of the need to reason about division.

**lemma** *binomial-Suc-Suc-eq-times*: $(Suc\ n\ choose\ Suc\ k) = (Suc\ n * (n\ choose\ k))\ div\ Suc\ k$
$\langle proof \rangle$

Another version of absorption, with $-1$ instead of *Suc*.

**lemma** *times-binomial-minus1-eq*: $0 < k \Longrightarrow k * (n\ choose\ k) = n * ((n - 1)\ choose\ (k - 1))$
$\langle proof \rangle$

## 93.2 The binomial theorem (courtesy of Tobias Nipkow):

Avigad's version, generalized to any commutative ring

**theorem** (**in** *comm-semiring-1*) *binomial-ring*:

$(a + b :: 'a)\hat{\ }n = (\sum k\le n.\ (\mathit{of\text{-}nat}\ (n\ \mathit{choose}\ k)) * a\hat{\ }k * b\hat{\ }(n{-}k))$
⟨proof⟩

Original version for the naturals.

**corollary** *binomial*: $(a + b :: \mathit{nat})\hat{\ }n = (\sum k\le n.\ (\mathit{of\text{-}nat}\ (n\ \mathit{choose}\ k)) * a\hat{\ }k * b\hat{\ }(n - k))$
⟨proof⟩

**lemma** *binomial-fact-lemma*: $k \le n \Longrightarrow \mathit{fact}\ k * \mathit{fact}\ (n - k) * (n\ \mathit{choose}\ k) = \mathit{fact}\ n$
⟨proof⟩

**lemma** *binomial-fact'*:
**assumes** $k \le n$
**shows** $n\ \mathit{choose}\ k = \mathit{fact}\ n\ \mathit{div}\ (\mathit{fact}\ k * \mathit{fact}\ (n - k))$
⟨proof⟩

**lemma** *binomial-fact*:
**assumes** *kn*: $k \le n$
**shows** $(\mathit{of\text{-}nat}\ (n\ \mathit{choose}\ k) :: 'a{::}\mathit{field\text{-}char\text{-}0}) = \mathit{fact}\ n\ /\ (\mathit{fact}\ k * \mathit{fact}\ (n - k))$
⟨proof⟩

**lemma** *fact-binomial*:
**assumes** $k \le n$
**shows** $\mathit{fact}\ k * \mathit{of\text{-}nat}\ (n\ \mathit{choose}\ k) = (\mathit{fact}\ n\ /\ \mathit{fact}\ (n - k) :: 'a{::}\mathit{field\text{-}char\text{-}0})$
⟨proof⟩

**lemma** *binomial-fact-pow*: $(n\ \mathit{choose}\ s) * \mathit{fact}\ s \le n\hat{\ }s$
⟨proof⟩

**lemma** *choose-two*: $n\ \mathit{choose}\ 2 = n * (n - 1)\ \mathit{div}\ 2$
⟨proof⟩

**lemma** *choose-row-sum*: $(\sum k\le n.\ n\ \mathit{choose}\ k) = 2\hat{\ }n$
⟨proof⟩

**lemma** *sum-choose-lower*: $(\sum k\le n.\ (r{+}k)\ \mathit{choose}\ k) = \mathit{Suc}\ (r{+}n)\ \mathit{choose}\ n$
⟨proof⟩

**lemma** *sum-choose-upper*: $(\sum k\le n.\ k\ \mathit{choose}\ m) = \mathit{Suc}\ n\ \mathit{choose}\ \mathit{Suc}\ m$
⟨proof⟩

**lemma** *choose-alternating-sum*:
$n > 0 \Longrightarrow (\sum i\le n.\ (-1)\hat{\ }i * \mathit{of\text{-}nat}\ (n\ \mathit{choose}\ i)) = (0 :: 'a{::}\mathit{comm\text{-}ring\text{-}1})$
⟨proof⟩

**lemma** *choose-even-sum*:
**assumes** $n > 0$
**shows** $2 * (\sum i\le n.\ \mathit{if}\ \mathit{even}\ i\ \mathit{then}\ \mathit{of\text{-}nat}\ (n\ \mathit{choose}\ i)\ \mathit{else}\ 0) = (2\ \hat{\ }\ n ::$

$'a::comm\text{-}ring\text{-}1)$
⟨proof⟩

**lemma** *choose-odd-sum*:
**assumes** $n > 0$
**shows** $2 * (\sum i \le n.\ if\ odd\ i\ then\ of\text{-}nat\ (n\ choose\ i)\ else\ 0) = (2\ \hat{}\ n :: 'a::comm\text{-}ring\text{-}1)$
⟨proof⟩

NW diagonal sum property

**lemma** *sum-choose-diagonal*:
**assumes** $m \le n$
**shows** $(\sum k \le m.\ (n - k)\ choose\ (m - k)) = Suc\ n\ choose\ m$
⟨proof⟩

## 93.3 Generalized binomial coefficients

**definition** *gbinomial* :: $'a::\{semidom\text{-}divide,semiring\text{-}char\text{-}0\} \Rightarrow nat \Rightarrow 'a$ (**infix** ‹gchoose› 64)
**where** *gbinomial-prod-rev*: $a\ gchoose\ k = (\prod i=0..<k.\ a - of\text{-}nat\ i)\ div\ fact\ k$

**lemma** *gbinomial-0* [*simp*]:
$a\ gchoose\ 0 = 1$
$0\ gchoose\ (Suc\ k) = 0$
⟨proof⟩

**lemma** *gbinomial-Suc*: $a\ gchoose\ (Suc\ k) = prod\ (\lambda i.\ a - of\text{-}nat\ i)\ \{0..k\}\ div\ fact\ (Suc\ k)$
⟨proof⟩

**lemma** *gbinomial-1* [*simp*]: $a\ gchoose\ 1 = a$
⟨proof⟩

**lemma** *gbinomial-Suc0* [*simp*]: $a\ gchoose\ Suc\ 0 = a$
⟨proof⟩

**lemma** *gbinomial-0-left*: $0\ gchoose\ k = (if\ k = 0\ then\ 1\ else\ 0)$
⟨proof⟩

**lemma** *gbinomial-mult-fact*: $fact\ k * (a\ gchoose\ k) = (\prod i = 0..<k.\ a - of\text{-}nat\ i)$
**for** $a :: 'a::field\text{-}char\text{-}0$
⟨proof⟩

**lemma** *gbinomial-mult-fact'*: $(a\ gchoose\ k) * fact\ k = (\prod i = 0..<k.\ a - of\text{-}nat\ i)$
**for** $a :: 'a::field\text{-}char\text{-}0$
⟨proof⟩

**lemma** *gbinomial-pochhammer*: $a\ gchoose\ k = (- 1)\ \hat{}\ k * pochhammer\ (- a)\ k\ /\ fact\ k$

**for** $a$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *gbinomial-pochhammer'*: $a$ *gchoose* $k$ = *pochhammer* $(a - \textit{of-nat}\ k + 1)$ $k$ / *fact* $k$
**for** $a$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *gbinomial-binomial*: $n$ *gchoose* $k$ = $n$ *choose* $k$
⟨*proof*⟩

**lemma** *of-nat-gbinomial*: *of-nat* ($n$ *gchoose* $k$) = (*of-nat* $n$ *gchoose* $k$ :: $'a$::*field-char-0*)
⟨*proof*⟩

**lemma** *binomial-gbinomial*: *of-nat* ($n$ *choose* $k$) = (*of-nat* $n$ *gchoose* $k$ :: $'a$::*field-char-0*)
⟨*proof*⟩

⟨*ML*⟩

**lemma** *gbinomial-mult-1*:
**fixes** $a$ :: $'a$::*field-char-0*
**shows** $a * (a$ *gchoose* $k)$ = *of-nat* $k * (a$ *gchoose* $k)$ + *of-nat* (*Suc* $k$) $* (a$ *gchoose* (*Suc* $k$))
(**is** $?l = ?r$)
⟨*proof*⟩

**lemma** *gbinomial-mult-1'*:
($a$ *gchoose* $k$) $* a$ = *of-nat* $k * (a$ *gchoose* $k)$ + *of-nat* (*Suc* $k$) $* (a$ *gchoose* (*Suc* $k$))
**for** $a$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *gbinomial-Suc-Suc*: $(a + 1)$ *gchoose* (*Suc* $k$) = ($a$ *gchoose* $k$) + ($a$ *gchoose* (*Suc* $k$))
**for** $a$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *gbinomial-reduce-nat*: $0 < k \Longrightarrow a$ *gchoose* $k$ = ($a-1$ *gchoose* $k-1$) + ($a-1$ *gchoose* $k$)
**for** $a$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *gchoose-row-sum-weighted*:
$(\sum k = 0..m.$ ($r$ *gchoose* $k$) $* (r/2 - \textit{of-nat}\ k))$ = *of-nat*(*Suc* $m$) / $2 * (r$ *gchoose* (*Suc* $m$))
**for** $r$ :: $'a$::*field-char-0*
⟨*proof*⟩

**lemma** *binomial-symmetric*:

**assumes** *kn*: $k \leq n$
**shows** $n$ *choose* $k = n$ *choose* $(n - k)$
⟨*proof*⟩

**lemma** *choose-rising-sum*:
$(\sum j \leq m.\ ((n + j)\ choose\ n)) = ((n + m + 1)\ choose\ (n + 1))$
$(\sum j \leq m.\ ((n + j)\ choose\ n)) = ((n + m + 1)\ choose\ m)$
⟨*proof*⟩

**lemma** *choose-linear-sum*: $(\sum i \leq n.\ i * (n\ choose\ i)) = n * 2\ \hat{}\ (n - 1)$
⟨*proof*⟩

**lemma** *choose-alternating-linear-sum*:
**assumes** $n \neq 1$
**shows** $(\sum i \leq n.\ (-1)\hat{}i * of\text{-}nat\ i * of\text{-}nat\ (n\ choose\ i) :: 'a::comm\text{-}ring\text{-}1) = 0$
⟨*proof*⟩

**lemma** *vandermonde*: $(\sum k \leq r.\ (m\ choose\ k) * (n\ choose\ (r - k))) = (m + n)$ *choose* $r$
⟨*proof*⟩

**lemma** *choose-square-sum*: $(\sum k \leq n.\ (n\ choose\ k)\hat{}2) = ((2*n)\ choose\ n)$
⟨*proof*⟩

**lemma** *pochhammer-binomial-sum*:
**fixes** $a\ b :: 'a::comm\text{-}ring\text{-}1$
**shows** *pochhammer* $(a + b)\ n = (\sum k \leq n.\ of\text{-}nat\ (n\ choose\ k) * pochhammer\ a\ k * pochhammer\ b\ (n - k))$
⟨*proof*⟩

Contributed by Manuel Eberl, generalised by LCP. Alternative definition of the binomial coefficient as $\prod i<k.\ (n - i) / (k - i)$.

**lemma** *gbinomial-altdef-of-nat*: $a$ *gchoose* $k = (\prod i = 0..<k.\ (a - of\text{-}nat\ i) / of\text{-}nat\ (k - i) :: 'a)$
**for** $k :: nat$ **and** $a :: 'a::field\text{-}char\text{-}0$
⟨*proof*⟩

**lemma** *gbinomial-ge-n-over-k-pow-k*:
**fixes** $k :: nat$
**and** $a :: 'a::linordered\text{-}field$
**assumes** *of-nat* $k \leq a$
**shows** $(a / of\text{-}nat\ k :: 'a)\ \hat{}\ k \leq a$ *gchoose* $k$
⟨*proof*⟩

**lemma** *gbinomial-negated-upper*: $(a\ gchoose\ k) = (-1)\ \hat{}\ k * ((of\text{-}nat\ k - a - 1)\ gchoose\ k)$
⟨*proof*⟩

**lemma** *gbinomial-minus*: $((-a)\ gchoose\ k) = (-1)\ \hat{}\ k * ((a + of\text{-}nat\ k - 1)$

*gchoose k*)
⟨*proof*⟩

**lemma** *Suc-times-gbinomial*: *of-nat* (*Suc k*) ∗ ((*a* + *1*) *gchoose* (*Suc k*)) = (*a* + *1*) ∗ (*a gchoose k*)
⟨*proof*⟩

**lemma** *gbinomial-factors*: ((*a* + *1*) *gchoose* (*Suc k*)) = (*a* + *1*) / *of-nat* (*Suc k*) ∗ (*a gchoose k*)
⟨*proof*⟩

**lemma** *gbinomial-rec*: ((*a* + *1*) *gchoose* (*Suc k*)) = (*a gchoose k*) ∗ ((*a* + *1*) / *of-nat* (*Suc k*))
⟨*proof*⟩

**lemma** *gbinomial-of-nat-symmetric*: *k* ≤ *n* ⟹ (*of-nat n*) *gchoose k* = (*of-nat n*) *gchoose* (*n* − *k*)
⟨*proof*⟩

The absorption identity (equation 5.5 [3, p. 157]):

$$\binom{r}{k} = \frac{r}{k}\binom{r-1}{k-1}, \quad \text{integer } k \neq 0.$$

**lemma** *gbinomial-absorption'*: *k* > *0* ⟹ *a gchoose k* = (*a* / *of-nat k*) ∗ (*a* − *1* *gchoose* (*k* − *1*))
⟨*proof*⟩

The absorption identity is written in the following form to avoid division by $k$ (the lower index) and therefore remove the $k \neq 0$ restriction [3, p. 157]:

$$k\binom{r}{k} = r\binom{r-1}{k-1}, \quad \text{integer } k.$$

**lemma** *gbinomial-absorption*: *of-nat* (*Suc k*) ∗ (*a gchoose Suc k*) = *a* ∗ ((*a* − *1*) *gchoose k*)
⟨*proof*⟩

The absorption identity for natural number binomial coefficients:

**lemma** *binomial-absorption*: *Suc k* ∗ (*n choose Suc k*) = *n* ∗ ((*n* − *1*) *choose k*)
⟨*proof*⟩

The absorption companion identity for natural number coefficients, following the proof by GKP [3, p. 157]:

**lemma** *binomial-absorb-comp*: (*n* − *k*) ∗ (*n choose k*) = *n* ∗ ((*n* − *1*) *choose k*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

The generalised absorption companion identity:

**lemma** *gbinomial-absorb-comp*: $(a - \textit{of-nat } k) * (a \textit{ gchoose } k) = a * ((a - 1) \textit{ gchoose } k)$
⟨*proof*⟩

**lemma** *gbinomial-addition-formula*:
$a \textit{ gchoose } (\textit{Suc } k) = ((a - 1) \textit{ gchoose } (\textit{Suc } k)) + ((a - 1) \textit{ gchoose } k)$
⟨*proof*⟩

**lemma** *binomial-addition-formula*:
$0 < n \Longrightarrow n \textit{ choose } (\textit{Suc } k) = ((n - 1) \textit{ choose } (\textit{Suc } k)) + ((n - 1) \textit{ choose } k)$
⟨*proof*⟩

Equation 5.9 of the reference material [3, p. 159] is a useful summation formula, operating on both indices:

$$\sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}, \quad \text{integer } n.$$

**lemma** *gbinomial-parallel-sum*: $(\sum k \le n. (a + \textit{of-nat } k) \textit{ gchoose } k) = (a + \textit{of-nat } n + 1) \textit{ gchoose } n$
⟨*proof*⟩

## 93.4 Summation on the upper index

Another summation formula is equation 5.10 of the reference material [3, p. 160], aptly named *summation on the upper index*:

$$\sum_{0 \le k \le n} \binom{k}{m} = \binom{n+1}{m+1}, \quad \text{integers } m, n \ge 0.$$

**lemma** *gbinomial-sum-up-index*:
$(\sum j = 0..n. (\textit{of-nat } j \textit{ gchoose } k) :: {'a}::\textit{field-char-0}) = (\textit{of-nat } n + 1) \textit{ gchoose } (k + 1)$
⟨*proof*⟩

**lemma** *gbinomial-index-swap*:
$((-1) \mathbin{\hat{}} k) * ((- (\textit{of-nat } n) - 1) \textit{ gchoose } k) = ((-1) \mathbin{\hat{}} n) * ((- (\textit{of-nat } k) - 1) \textit{ gchoose } n)$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *gbinomial-sum-lower-neg*: $(\sum k \le m. (a \textit{ gchoose } k) * (- 1) \mathbin{\hat{}} k) = (- 1) \mathbin{\hat{}} m * (a - 1 \textit{ gchoose } m)$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *gbinomial-partial-row-sum*:

$(\sum k{\le}m.\ (a\ gchoose\ k) * ((a\ /\ 2) - of\text{-}nat\ k)) = ((of\text{-}nat\ m + 1)/2) * (a\ gchoose\ (m + 1))$
⟨proof⟩

**lemma** *sum-bounds-lt-plus1*: $(\sum k{<}mm.\ f\ (Suc\ k)) = (\sum k{=}1..mm.\ f\ k)$
⟨proof⟩

**lemma** *gbinomial-partial-sum-poly*:
$(\sum k{\le}m.\ (of\text{-}nat\ m + a\ gchoose\ k) * x\hat{}k * y\hat{}(m-k)) =$
$(\sum k{\le}m.\ (-a\ gchoose\ k) * (-x)\hat{}k * (x + y)\hat{}(m-k))$
(**is** $?lhs\ m = ?rhs\ m$)
⟨proof⟩

**lemma** *gbinomial-partial-sum-poly-xpos*:
$(\sum k{\le}m.\ (of\text{-}nat\ m + a\ gchoose\ k) * x\hat{}k * y\hat{}(m-k)) =$
$(\sum k{\le}m.\ (of\text{-}nat\ k + a - 1\ gchoose\ k) * x\hat{}k * (x + y)\hat{}(m-k))$ (**is** $?lhs = ?rhs$)
⟨proof⟩

**lemma** *binomial-r-part-sum*: $(\sum k{\le}m.\ (2 * m + 1\ choose\ k)) = 2\ \hat{}\ (2 * m)$
⟨proof⟩

**lemma** *gbinomial-r-part-sum*: $(\sum k{\le}m.\ (2 * (of\text{-}nat\ m) + 1\ gchoose\ k)) = 2\ \hat{}\ (2 * m)$
(**is** $?lhs = ?rhs$)
⟨proof⟩

**lemma** *gbinomial-sum-nat-pow2*:
$(\sum k{\le}m.\ (of\text{-}nat\ (m + k)\ gchoose\ k :: 'a::field\text{-}char\text{-}0)\ /\ 2\ \hat{}\ k) = 2\ \hat{}\ m$
(**is** $?lhs = ?rhs$)
⟨proof⟩

**lemma** *gbinomial-trinomial-revision*:
**assumes** $k \le m$
**shows** $(a\ gchoose\ m) * (of\text{-}nat\ m\ gchoose\ k) = (a\ gchoose\ k) * (a - of\text{-}nat\ k\ gchoose\ (m - k))$
⟨proof⟩

Versions of the theorems above for the natural-number version of "choose"

**lemma** *binomial-altdef-of-nat*:
$k \le n \Longrightarrow of\text{-}nat\ (n\ choose\ k) = (\prod i = 0..{<}k.\ of\text{-}nat\ (n - i)\ /\ of\text{-}nat\ (k - i) :: 'a)$
**for** $n\ k :: nat$ **and** $x :: 'a::field\text{-}char\text{-}0$
⟨proof⟩

**lemma** *binomial-ge-n-over-k-pow-k*: $k \le n \Longrightarrow (of\text{-}nat\ n\ /\ of\text{-}nat\ k :: 'a)\ \hat{}\ k \le of\text{-}nat\ (n\ choose\ k)$
**for** $k\ n :: nat$ **and** $x :: 'a::linordered\text{-}field$
⟨proof⟩

**lemma** *binomial-le-pow*:
**assumes** $r \le n$
**shows** $n$ *choose* $r \le n$ ^ $r$
⟨*proof*⟩

**lemma** *choose-dvd*:
**assumes** $k \le n$ **shows** *fact* $k$ ∗ *fact* $(n - k)$ *dvd* (*fact* $n$)
⟨*proof*⟩

**lemma** *fact-fact-dvd-fact*:
*fact* $k$ ∗ *fact* $n$ *dvd* (*fact* $(k + n)$)
⟨*proof*⟩

**lemma** *choose-mult-lemma*:
$((m + r + k)$ *choose* $(m + k)) * ((m + k)$ *choose* $k) = ((m + r + k)$ *choose* $k) * ((m + r)$ *choose* $m)$
(**is** *?lhs* = -)
⟨*proof*⟩

The "Subset of a Subset" identity.

**lemma** *choose-mult*:
$k \le m \Longrightarrow m \le n \Longrightarrow (n$ *choose* $m) * (m$ *choose* $k) = (n$ *choose* $k) * ((n - k)$ *choose* $(m - k))$
⟨*proof*⟩

**lemma** *of-nat-binomial-eq-mult-binomial-Suc*:
**assumes** $k \le n$
**shows** (*of-nat* :: (*nat* ⇒ ('*a* :: *field-char-0*))) ($n$ *choose* $k$) = *of-nat* $(n + 1 - k)$ / *of-nat* $(n + 1)$ ∗ *of-nat* (*Suc* $n$ *choose* $k$)
⟨*proof*⟩

## 93.5 More on Binomial Coefficients

The number of nat lists of length $m$ summing to $N$ is $N + m - 1$ *choose* $N$:

**lemma** *card-length-sum-list-rec*:
**assumes** $m \ge 1$
**shows** *card* {*l*::*nat list*. *length* $l = m$ ∧ *sum-list* $l = N$} =
*card* {*l*. *length* $l = (m - 1)$ ∧ *sum-list* $l = N$} +
*card* {*l*. *length* $l = m$ ∧ *sum-list* $l + 1 = N$}
(**is** *card ?C* = *card ?A* + *card ?B*)
⟨*proof*⟩

**lemma** *card-length-sum-list*: *card* {*l*::*nat list*. *size* $l = m$ ∧ *sum-list* $l = N$} = $(N + m - 1)$ *choose* $N$
— by Holden Lee, tidied by Tobias Nipkow
⟨*proof*⟩

**lemma** *card-disjoint-shuffles*:
**assumes** $set\ xs \cap set\ ys = \{\}$
**shows** $card\ (shuffles\ xs\ ys) = (length\ xs + length\ ys)\ choose\ length\ xs$
$\langle proof \rangle$

**lemma** *Suc-times-binomial-add*: $Suc\ a * (Suc\ (a + b)\ choose\ Suc\ a) = Suc\ b * (Suc\ (a + b)\ choose\ a)$
— by Lukas Bulwahn
$\langle proof \rangle$

## 93.6 Inclusion-exclusion principle

Ported from HOL Light by lcp

**lemma** *Inter-over-Union*:
$\bigcap\ \{\bigcup\ (\mathcal{F}\ x)\ |x.\ x \in S\} = \bigcup\ \{\bigcap\ (G\ `\ S)\ |G.\ \forall x{\in}S.\ G\ x \in \mathcal{F}\ x\}$
$\langle proof \rangle$

**lemma** *subset-insert-lemma*:
$\{T.\ T \subseteq (insert\ a\ S) \wedge P\ T\} = \{T.\ T \subseteq S \wedge P\ T\} \cup \{insert\ a\ T\ |T.\ T \subseteq S \wedge P(insert\ a\ T)\}$ (**is** $?L{=}?R$)
$\langle proof \rangle$

Versions for additive real functions, where the additivity applies only to some specific subsets (e.g. cardinality of finite sets, measurable sets with bounded measure. (From HOL Light)

**locale** *Incl-Excl* =
**fixes** $P :: 'a\ set \Rightarrow bool$ **and** $f :: 'a\ set \Rightarrow 'b{::}ring\text{-}1$
**assumes** *disj-add*: $[\![P\ S;\ P\ T;\ disjnt\ S\ T]\!] \Longrightarrow f(S \cup T) = f\ S + f\ T$
**and** *empty*: $P\{\}$
**and** *Int*: $[\![P\ S;\ P\ T]\!] \Longrightarrow P(S \cap T)$
**and** *Un*: $[\![P\ S;\ P\ T]\!] \Longrightarrow P(S \cup T)$
**and** *Diff*: $[\![P\ S;\ P\ T]\!] \Longrightarrow P(S - T)$

**begin**

**lemma** *f-empty* [*simp*]: $f\{\} = 0$
$\langle proof \rangle$

**lemma** *f-Un-Int*: $[\![P\ S;\ P\ T]\!] \Longrightarrow f(S \cup T) + f(S \cap T) = f\ S + f\ T$
$\langle proof \rangle$

**lemma** *restricted-indexed*:
**assumes** *finite* $A$ **and** $X$: $\bigwedge a.\ a \in A \Longrightarrow P(X\ a)$
**shows** $f(\bigcup(X\ `\ A)) = (\sum B\ |\ B \subseteq A \wedge B \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ B + 1) * f\ (\bigcap\ (X\ `\ B)))$
$\langle proof \rangle$

**lemma** *restricted*:
**assumes** *finite A* $\bigwedge a.\ a \in A \Longrightarrow P\ a$
**shows** $f(\bigcup\ A) = (\sum B \mid B \subseteq A \land B \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ B + 1) * f\ (\bigcap\ B))$
⟨*proof*⟩

**end**

## 93.7 Versions for unrestrictedly additive functions

**lemma** *Incl-Excl-UN*:
**fixes** $f$ :: *'a set* ⇒ *'b::ring-1*
**assumes** $\bigwedge S\ T.\ disjnt\ S\ T \Longrightarrow f(S \cup T) = f\ S + f\ T$ *finite A*
**shows** $f(\bigcup (G\ `\ A)) = (\sum B \mid B \subseteq A \land B \neq \{\}.\ (-1)\ \hat{}\ (card\ B + 1) * f\ (\bigcap (G\ `\ B)))$
⟨*proof*⟩

**lemma** *Incl-Excl-Union*:
**fixes** $f$ :: *'a set* ⇒ *'b::ring-1*
**assumes** $\bigwedge S\ T.\ disjnt\ S\ T \Longrightarrow f(S \cup T) = f\ S + f\ T$ *finite A*
**shows** $f(\bigcup\ A) = (\sum B \mid B \subseteq A \land B \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ B + 1) * f\ (\bigcap\ B))$
⟨*proof*⟩

The famous inclusion-exclusion formula for the cardinality of a union

**lemma** *int-card-UNION*:
**assumes** *finite A* $\bigwedge K.\ K \in A \Longrightarrow$ *finite K*
**shows** $int\ (card\ (\bigcup A)) = (\sum I \mid I \subseteq A \land I \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ I + 1) * int\ (card\ (\bigcap I)))$
⟨*proof*⟩

A more conventional form

**lemma** *inclusion-exclusion*:
**assumes** *finite A* $\bigwedge K.\ K \in A \Longrightarrow$ *finite K*
**shows** $int(card(\bigcup\ A)) =$
$(\sum n{=}1..card\ A.\ (-1)\ \hat{}\ (Suc\ n) * (\sum B \mid B \subseteq A \land card\ B = n.\ int\ (card\ (\bigcap\ B))))$ (**is** *-=?R*)
⟨*proof*⟩

**lemma** *card-UNION*:
**assumes** *finite A* **and** $\bigwedge K.\ K \in A \Longrightarrow$ *finite K*
**shows** $card\ (\bigcup A) = nat\ (\sum I \mid I \subseteq A \land I \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ I + 1) * int\ (card\ (\bigcap I)))$
⟨*proof*⟩

**lemma** *card-UNION-nonneg*:
**assumes** *finite A* **and** $\bigwedge K.\ K \in A \Longrightarrow$ *finite K*
**shows** $(\sum I \mid I \subseteq A \land I \neq \{\}.\ (-\ 1)\ \hat{}\ (card\ I + 1) * int\ (card\ (\bigcap I))) \geq 0$
⟨*proof*⟩

## 93.8 General "Moebius inversion" inclusion-exclusion principle

This "symmetric" form is from Ira Gessel: "Symmetric Inclusion-Exclusion"

**lemma** *sum-Un-eq*:
$[\![S \cap T = \{\}; S \cup T = U; \textit{finite } U]\!]$
$\Longrightarrow (\textit{sum } f\ S + \textit{sum } f\ T = \textit{sum } f\ U)$
⟨*proof*⟩

**lemma** *card-adjust-lemma*: $[\![\textit{inj-on } f\ S;\ x = y + \textit{card } (f\ `\ S)]\!] \Longrightarrow x = y + \textit{card } S$
⟨*proof*⟩

**lemma** *card-subsets-step*:
**assumes** *finite* $S$ $x \notin S$ $U \subseteq S$
**shows** $\textit{card } \{T.\ T \subseteq (\textit{insert } x\ S) \wedge U \subseteq T \wedge \textit{odd}(\textit{card } T)\}$
$= \textit{card } \{T.\ T \subseteq S \wedge U \subseteq T \wedge \textit{odd}(\textit{card } T)\} + \textit{card } \{T.\ T \subseteq S \wedge U \subseteq T$
$\wedge\ \textit{even}(\textit{card } T)\} \wedge$
$\textit{card } \{T.\ T \subseteq (\textit{insert } x\ S) \wedge U \subseteq T \wedge \textit{even}(\textit{card } T)\}$
$= \textit{card } \{T.\ T \subseteq S \wedge U \subseteq T \wedge \textit{even}(\textit{card } T)\} + \textit{card } \{T.\ T \subseteq S \wedge U \subseteq T$
$\wedge\ \textit{odd}(\textit{card } T)\}$
⟨*proof*⟩

**lemma** *card-subsupersets-even-odd*:
**assumes** *finite* $S$ $U \subset S$
**shows** $\textit{card } \{T.\ T \subseteq S \wedge U \subseteq T \wedge \textit{even}(\textit{card } T)\}$
$= \textit{card } \{T.\ T \subseteq S \wedge U \subseteq T \wedge \textit{odd}(\textit{card } T)\}$
⟨*proof*⟩

**lemma** *sum-alternating-cancels*:
**assumes** *finite* $S$ $\textit{card } \{x.\ x \in S \wedge \textit{even}(f\ x)\} = \textit{card } \{x.\ x \in S \wedge \textit{odd}(f\ x)\}$
**shows** $(\sum x{\in}S.\ (-1)\ \hat{}\ f\ x) = (0::'b::\textit{ring-1})$
⟨*proof*⟩

**lemma** *inclusion-exclusion-symmetric*:
**fixes** $f :: 'a\ \textit{set} \Rightarrow 'b::\textit{ring-1}$
**assumes** §: $\bigwedge S.\ \textit{finite } S \Longrightarrow g\ S = (\sum T \in \textit{Pow } S.\ (-1)\ \hat{}\ \textit{card } T * f\ T)$
**and** *finite* $S$
**shows** $f\ S = (\sum T \in \textit{Pow } S.\ (-1)\ \hat{}\ \textit{card } T * g\ T)$
⟨*proof*⟩

The more typical non-symmetric version.

**lemma** *inclusion-exclusion-mobius*:
**fixes** $f :: 'a\ \textit{set} \Rightarrow 'b::\textit{ring-1}$
**assumes** §: $\bigwedge S.\ \textit{finite } S \Longrightarrow g\ S = \textit{sum } f\ (\textit{Pow } S)$ **and** *finite* $S$
**shows** $f\ S = (\sum T \in \textit{Pow } S.\ (-1)\ \hat{}\ (\textit{card } S - \textit{card } T) * g\ T)$ (**is** - = *?rhs*)
⟨*proof*⟩

### 93.9 Executable code

**lemma** *gbinomial-code* [*code*]:
*a gchoose k* =
(*if k* = *0 then 1*
*else fold-atLeastAtMost-nat* ($\lambda$*k acc.* (*a* − *of-nat k*) ∗ *acc*) *0* (*k* − *1*) *1* / *fact k*)
⟨*proof*⟩

**lemma** *binomial-code* [*code*]:
*n choose k* =
(*if k* > *n then 0*
*else if 2* ∗ *k* > *n then n choose* (*n* − *k*)
*else* (*fold-atLeastAtMost-nat* (∗) (*n* − *k* + *1*) *n 1 div fact k*))
⟨*proof*⟩

**end**

# 94 Main HOL

Classical Higher-order Logic – only "Main", excluding real and complex numbers etc.

**theory** *Main*
**imports**
*Predicate-Compile*
*Quickcheck-Narrowing*
*Mirabelle*
*Extraction*
*Nunchaku*
*BNF-Greatest-Fixpoint*
*Filter*
*Conditionally-Complete-Lattices*
*Binomial*
*GCD*
**begin**

### 94.1 Namespace cleanup

**hide-const** (**open**)
*czero cinfinite cfinite csum cone ctwo Csum cprod cexp image2 image2p vimage2p Gr Grp collect*
*fsts snds setl setr convol pick-middlep fstOp sndOp csquare relImage relInvImage Succ Shift*
*shift proj id-bnf*

**hide-fact** (**open**) *id-bnf-def type-definition-id-bnf-UNIV*

## 94.2 Syntax cleanup

**no-notation**
*ordLeq2* (**infix** ‹<=o› *50*) **and**
*ordLeq3* (**infix** ‹≤o› *50*) **and**
*ordLess2* (**infix** ‹<o› *50*) **and**
*ordIso2* (**infix** ‹=o› *50*) **and**
*card-of* (‹(‹open-block notation=‹mixfix card-of››|-|)›) **and**
*BNF-Cardinal-Arithmetic.csum* (**infixr** ‹+c› *65*) **and**
*BNF-Cardinal-Arithmetic.cprod* (**infixr** ‹∗c› *80*) **and**
*BNF-Cardinal-Arithmetic.cexp* (**infixr** ‹^c› *90*) **and**
*BNF-Def.convol* (‹(‹indent=1 notation=‹mixfix convol››⟨-,/ -⟩)›)

**bundle** *cardinal-syntax*
**begin**

**notation**
*ordLeq2* (**infix** ‹<=o› *50*) **and**
*ordLeq3* (**infix** ‹≤o› *50*) **and**
*ordLess2* (**infix** ‹<o› *50*) **and**
*ordIso2* (**infix** ‹=o› *50*) **and**
*card-of* (‹(‹open-block notation=‹mixfix card-of››|-|)›) **and**
*BNF-Cardinal-Arithmetic.csum* (**infixr** ‹+c› *65*) **and**
*BNF-Cardinal-Arithmetic.cprod* (**infixr** ‹∗c› *80*) **and**
*BNF-Cardinal-Arithmetic.cexp* (**infixr** ‹^c› *90*)

**alias** *cinfinite* = *BNF-Cardinal-Arithmetic.cinfinite*
**alias** *czero* = *BNF-Cardinal-Arithmetic.czero*
**alias** *cone* = *BNF-Cardinal-Arithmetic.cone*
**alias** *ctwo* = *BNF-Cardinal-Arithmetic.ctwo*

**end**

## 94.3 Lattice syntax

**bundle** *lattice-syntax*
**begin**

**notation**
*bot* (‹⊥›) **and**
*top* (‹⊤›) **and**
*inf* (**infixl** ‹⊓› *70*) **and**
*sup* (**infixl** ‹⊔› *65*) **and**
*Inf* (‹(‹open-block notation=‹prefix ⨅››⨅ -)› [*900*] *900*) **and**
*Sup* (‹(‹open-block notation=‹prefix ⨆››⨆ -)› [*900*] *900*)

**syntax**
*-INF1* :: *pttrns* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder ⨅››⨅-./ -)› [*0*, *10*] *10*)
*-INF* :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* (‹(‹indent=3 notation=‹binder ⨅››⨅-∈-./

-)› [0, 0, 10] 10)
  -SUP1 :: pttrns ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ⨆››⨆-./
-)› [0, 10] 10)
  -SUP :: pttrn ⇒ 'a set ⇒ 'b ⇒ 'b (‹(‹indent=3 notation=‹binder ⨆››⨆-∈-./
-)› [0, 0, 10] 10)

**end**

**unbundle** *no lattice-syntax*

**end**

# 95 Archimedean Fields, Floor and Ceiling Functions

**theory** *Archimedean-Field*
**imports** *Main*
**begin**

**lemma** *cInf-abs-ge*:
  **fixes** $S$ :: $'a$::{*linordered-idom*,*conditionally-complete-linorder*} *set*
  **assumes** $S \neq \{\}$
    **and** *bdd*: $\bigwedge x.\ x{\in}S \Longrightarrow |x| \leq a$
  **shows** $|Inf\ S| \leq a$
⟨*proof*⟩

**lemma** *cSup-asclose*:
  **fixes** $S$ :: $'a$::{*linordered-idom*,*conditionally-complete-linorder*} *set*
  **assumes** *S*: $S \neq \{\}$
    **and** *b*: $\forall x{\in}S.\ |x - l| \leq e$
  **shows** $|Sup\ S - l| \leq e$
⟨*proof*⟩

**lemma** *cInf-asclose*:
  **fixes** $S$ :: $'a$::{*linordered-idom*,*conditionally-complete-linorder*} *set*
  **assumes** *S*: $S \neq \{\}$
    **and** *b*: $\forall x{\in}S.\ |x - l| \leq e$
  **shows** $|Inf\ S - l| \leq e$
⟨*proof*⟩

## 95.1 Class of Archimedean fields

Archimedean fields have no infinite elements.

**class** *archimedean-field* = *linordered-field* +
  **assumes** *ex-le-of-int*: $\exists z.\ x \leq of\text{-}int\ z$

**lemma** *ex-less-of-int*: $\exists z.\ x < of\text{-}int\ z$
  **for** $x$ :: $'a$::*archimedean-field*

⟨*proof*⟩

**lemma** *ex-of-int-less*: $\exists z.\ \textit{of-int}\ z < x$
**for** $x :: {'a}::\textit{archimedean-field}$
⟨*proof*⟩

**lemma** *reals-Archimedean2*: $\exists n.\ x < \textit{of-nat}\ n$
**for** $x :: {'a}::\textit{archimedean-field}$
⟨*proof*⟩

**lemma** *real-arch-simple*: $\exists n.\ x \le \textit{of-nat}\ n$
**for** $x :: {'a}::\textit{archimedean-field}$
⟨*proof*⟩

Archimedean fields have no infinitesimal elements.

**lemma** *reals-Archimedean*:
**fixes** $x :: {'a}::\textit{archimedean-field}$
**assumes** $0 < x$
**shows** $\exists n.\ \textit{inverse}\ (\textit{of-nat}\ (\textit{Suc}\ n)) < x$
⟨*proof*⟩

**lemma** *ex-inverse-of-nat-less*:
**fixes** $x :: {'a}::\textit{archimedean-field}$
**assumes** $0 < x$
**shows** $\exists n>0.\ \textit{inverse}\ (\textit{of-nat}\ n) < x$
⟨*proof*⟩

**lemma** *ex-less-of-nat-mult*:
**fixes** $x :: {'a}::\textit{archimedean-field}$
**assumes** $0 < x$
**shows** $\exists n.\ y < \textit{of-nat}\ n * x$
⟨*proof*⟩

## 95.2 Existence and uniqueness of floor function

**lemma** *exists-least-lemma*:
**assumes** $\neg P\ 0$ **and** $\exists n.\ P\ n$
**shows** $\exists n.\ \neg P\ n \wedge P\ (\textit{Suc}\ n)$
⟨*proof*⟩

**lemma** *floor-exists*:
**fixes** $x :: {'a}::\textit{archimedean-field}$
**shows** $\exists z.\ \textit{of-int}\ z \le x \wedge x < \textit{of-int}\ (z + 1)$
⟨*proof*⟩

**lemma** *floor-exists1*: $\exists!z.\ \textit{of-int}\ z \le x \wedge x < \textit{of-int}\ (z + 1)$
**for** $x :: {'a}::\textit{archimedean-field}$
⟨*proof*⟩

## 95.3 Floor function

**class** *floor-ceiling* = *archimedean-field* +
  **fixes** *floor* :: $'a \Rightarrow int$ (‹(‹*open-block notation*=‹*mixfix floor*››$\lfloor$-$\rfloor$)›)
  **assumes** *floor-correct*: $\mathit{of\text{-}int}\ \lfloor x \rfloor \le x \land x < \mathit{of\text{-}int}\ (\lfloor x \rfloor + 1)$

**lemma** *floor-unique*: $\mathit{of\text{-}int}\ z \le x \Longrightarrow x < \mathit{of\text{-}int}\ z + 1 \Longrightarrow \lfloor x \rfloor = z$
  ⟨*proof*⟩

**lemma** *floor-eq-iff*: $\lfloor x \rfloor = a \longleftrightarrow \mathit{of\text{-}int}\ a \le x \land x < \mathit{of\text{-}int}\ a + 1$
  ⟨*proof*⟩

**lemma** *of-int-floor-le* [*simp*]: $\mathit{of\text{-}int}\ \lfloor x \rfloor \le x$
  ⟨*proof*⟩

**lemma** *le-floor-iff*: $z \le \lfloor x \rfloor \longleftrightarrow \mathit{of\text{-}int}\ z \le x$
⟨*proof*⟩

**lemma** *floor-less-iff*: $\lfloor x \rfloor < z \longleftrightarrow x < \mathit{of\text{-}int}\ z$
  ⟨*proof*⟩

**lemma** *less-floor-iff*: $z < \lfloor x \rfloor \longleftrightarrow \mathit{of\text{-}int}\ z + 1 \le x$
  ⟨*proof*⟩

**lemma** *floor-le-iff*: $\lfloor x \rfloor \le z \longleftrightarrow x < \mathit{of\text{-}int}\ z + 1$
  ⟨*proof*⟩

**lemma** *floor-split*[*linarith-split*]: $P\ \lfloor t \rfloor \longleftrightarrow (\forall\, i.\ \mathit{of\text{-}int}\ i \le t \land t < \mathit{of\text{-}int}\ i + 1$
$\longrightarrow P\ i)$
  ⟨*proof*⟩

**lemma** *floor-eq-imp-diff-1*: $\lfloor x \rfloor = \lfloor y \rfloor \Longrightarrow |x - y| < 1$
  ⟨*proof*⟩

**lemma** *floor-mono*:
  **assumes** $x \le y$
  **shows** $\lfloor x \rfloor \le \lfloor y \rfloor$
  ⟨*proof*⟩

**lemma** *floor-less-cancel*: $\lfloor x \rfloor < \lfloor y \rfloor \Longrightarrow x < y$
  ⟨*proof*⟩

**lemma** *floor-of-int* [*simp*]: $\lfloor \mathit{of\text{-}int}\ z \rfloor = z$
  ⟨*proof*⟩

**lemma** *floor-of-nat* [*simp*]: $\lfloor \mathit{of\text{-}nat}\ n \rfloor = \mathit{int}\ n$
  ⟨*proof*⟩

**lemma** *le-floor-add*: $\lfloor x \rfloor + \lfloor y \rfloor \le \lfloor x + y \rfloor$
  ⟨*proof*⟩

Floor with numerals.

**lemma** *floor-zero* [*simp*]: $\lfloor 0 \rfloor = 0$
⟨*proof*⟩

**lemma** *floor-one* [*simp*]: $\lfloor 1 \rfloor = 1$
⟨*proof*⟩

**lemma** *floor-numeral* [*simp*]: $\lfloor numeral\ v \rfloor = numeral\ v$
⟨*proof*⟩

**lemma** *floor-neg-numeral* [*simp*]: $\lfloor - numeral\ v \rfloor = - numeral\ v$
⟨*proof*⟩

**lemma** *zero-le-floor* [*simp*]: $0 \le \lfloor x \rfloor \longleftrightarrow 0 \le x$
⟨*proof*⟩

**lemma** *one-le-floor* [*simp*]: $1 \le \lfloor x \rfloor \longleftrightarrow 1 \le x$
⟨*proof*⟩

**lemma** *numeral-le-floor* [*simp*]: $numeral\ v \le \lfloor x \rfloor \longleftrightarrow numeral\ v \le x$
⟨*proof*⟩

**lemma** *neg-numeral-le-floor* [*simp*]: $- numeral\ v \le \lfloor x \rfloor \longleftrightarrow - numeral\ v \le x$
⟨*proof*⟩

**lemma** *zero-less-floor* [*simp*]: $0 < \lfloor x \rfloor \longleftrightarrow 1 \le x$
⟨*proof*⟩

**lemma** *one-less-floor* [*simp*]: $1 < \lfloor x \rfloor \longleftrightarrow 2 \le x$
⟨*proof*⟩

**lemma** *numeral-less-floor* [*simp*]: $numeral\ v < \lfloor x \rfloor \longleftrightarrow numeral\ v + 1 \le x$
⟨*proof*⟩

**lemma** *neg-numeral-less-floor* [*simp*]: $- numeral\ v < \lfloor x \rfloor \longleftrightarrow - numeral\ v + 1 \le x$
⟨*proof*⟩

**lemma** *floor-le-zero* [*simp*]: $\lfloor x \rfloor \le 0 \longleftrightarrow x < 1$
⟨*proof*⟩

**lemma** *floor-le-one* [*simp*]: $\lfloor x \rfloor \le 1 \longleftrightarrow x < 2$
⟨*proof*⟩

**lemma** *floor-le-numeral* [*simp*]: $\lfloor x \rfloor \le numeral\ v \longleftrightarrow x < numeral\ v + 1$
⟨*proof*⟩

**lemma** *floor-le-neg-numeral* [*simp*]: $\lfloor x \rfloor \le - numeral\ v \longleftrightarrow x < - numeral\ v + 1$

⟨*proof*⟩

**lemma** *floor-less-zero* [*simp*]: $\lfloor x \rfloor < 0 \longleftrightarrow x < 0$
⟨*proof*⟩

**lemma** *floor-less-one* [*simp*]: $\lfloor x \rfloor < 1 \longleftrightarrow x < 1$
⟨*proof*⟩

**lemma** *floor-less-numeral* [*simp*]: $\lfloor x \rfloor < \mathit{numeral}\ v \longleftrightarrow x < \mathit{numeral}\ v$
⟨*proof*⟩

**lemma** *floor-less-neg-numeral* [*simp*]: $\lfloor x \rfloor < -\ \mathit{numeral}\ v \longleftrightarrow x < -\ \mathit{numeral}\ v$
⟨*proof*⟩

**lemma** *le-mult-floor-Ints*:
**assumes** $0 \leq a\ a \in \mathit{Ints}$
**shows** $\mathit{of\text{-}int}\ (\lfloor a \rfloor * \lfloor b \rfloor) \leq (\mathit{of\text{-}int}\lfloor a * b \rfloor :: 'a :: \mathit{linordered\text{-}idom})$
⟨*proof*⟩

Addition and subtraction of integers.

**lemma** *floor-add-int*: $\lfloor x \rfloor + z = \lfloor x + \mathit{of\text{-}int}\ z \rfloor$
⟨*proof*⟩

**lemma** *int-add-floor*: $z + \lfloor x \rfloor = \lfloor \mathit{of\text{-}int}\ z + x \rfloor$
⟨*proof*⟩

**lemma** *one-add-floor*: $\lfloor x \rfloor + 1 = \lfloor x + 1 \rfloor$
⟨*proof*⟩

**lemma** *floor-diff-of-int* [*simp*]: $\lfloor x - \mathit{of\text{-}int}\ z \rfloor = \lfloor x \rfloor - z$
⟨*proof*⟩

**lemma** *floor-uminus-of-int* [*simp*]: $\lfloor -\ (\mathit{of\text{-}int}\ z) \rfloor = -\ z$
⟨*proof*⟩

**lemma** *floor-diff-numeral* [*simp*]: $\lfloor x - \mathit{numeral}\ v \rfloor = \lfloor x \rfloor - \mathit{numeral}\ v$
⟨*proof*⟩

**lemma** *floor-diff-one* [*simp*]: $\lfloor x - 1 \rfloor = \lfloor x \rfloor - 1$
⟨*proof*⟩

**lemma** *le-mult-floor*:
**assumes** $0 \leq a$ **and** $0 \leq b$
**shows** $\lfloor a \rfloor * \lfloor b \rfloor \leq \lfloor a * b \rfloor$
⟨*proof*⟩

**lemma** *floor-divide-of-int-eq*: $\lfloor \mathit{of\text{-}int}\ k\ /\ \mathit{of\text{-}int}\ l \rfloor = k\ \mathit{div}\ l$
**for** $k\ l :: \mathit{int}$
⟨*proof*⟩

**lemma** *floor-divide-of-nat-eq*: $\lfloor$*of-nat m / of-nat n*$\rfloor$ = *of-nat* (*m div n*)
**for** *m n* :: *nat*
⟨*proof*⟩

**lemma** *floor-divide-lower*:
**fixes** *q* :: *'a::floor-ceiling*
**shows** $q > 0 \Longrightarrow$ *of-int* $\lfloor p\ /\ q \rfloor * q \le p$
⟨*proof*⟩

**lemma** *floor-divide-upper*:
**fixes** *q* :: *'a::floor-ceiling*
**shows** $q > 0 \Longrightarrow p <$ (*of-int* $\lfloor p\ /\ q \rfloor + 1) * q$
⟨*proof*⟩

## 95.4 Ceiling function

**definition** *ceiling* :: *'a::floor-ceiling* ⇒ *int* (‹(‹*open-block notation*=‹*mixfix ceiling*››⌈-⌉)›)
**where** $\lceil x \rceil = - \lfloor - x \rfloor$

**lemma** *ceiling-correct*: *of-int* $\lceil x \rceil - 1 < x \wedge x \le$ *of-int* $\lceil x \rceil$
⟨*proof*⟩

**lemma** *ceiling-unique*: *of-int* $z - 1 < x \Longrightarrow x \le$ *of-int* $z \Longrightarrow \lceil x \rceil = z$
⟨*proof*⟩

**lemma** *ceiling-eq-iff*: $\lceil x \rceil = a \longleftrightarrow$ *of-int* $a - 1 < x \wedge x \le$ *of-int* $a$
⟨*proof*⟩

**lemma** *le-of-int-ceiling* [*simp*]: $x \le$ *of-int* $\lceil x \rceil$
⟨*proof*⟩

**lemma** *ceiling-le-iff*: $\lceil x \rceil \le z \longleftrightarrow x \le$ *of-int* $z$
⟨*proof*⟩

**lemma** *less-ceiling-iff*: $z < \lceil x \rceil \longleftrightarrow$ *of-int* $z < x$
⟨*proof*⟩

**lemma** *ceiling-less-iff*: $\lceil x \rceil < z \longleftrightarrow x \le$ *of-int* $z - 1$
⟨*proof*⟩

**lemma** *le-ceiling-iff*: $z \le \lceil x \rceil \longleftrightarrow$ *of-int* $z - 1 < x$
⟨*proof*⟩

**lemma** *ceiling-mono*: $x \ge y \Longrightarrow \lceil x \rceil \ge \lceil y \rceil$
⟨*proof*⟩

**lemma** *ceiling-less-cancel*: $\lceil x \rceil < \lceil y \rceil \Longrightarrow x < y$

⟨*proof*⟩

**lemma** *ceiling-of-int* [*simp*]: $\lceil$*of-int z*$\rceil$ = *z*
⟨*proof*⟩

**lemma** *ceiling-of-nat* [*simp*]: $\lceil$*of-nat n*$\rceil$ = *int n*
⟨*proof*⟩

**lemma** *ceiling-add-le*: $\lceil x + y\rceil \leq \lceil x\rceil + \lceil y\rceil$
⟨*proof*⟩

**lemma** *mult-ceiling-le*:
**assumes** $0 \leq a$ **and** $0 \leq b$
**shows** $\lceil a * b\rceil \leq \lceil a\rceil * \lceil b\rceil$
⟨*proof*⟩

**lemma** *mult-ceiling-le-Ints*:
**assumes** $0 \leq a$ $a \in$ *Ints*
**shows** (*of-int* $\lceil a * b\rceil$ :: $'a$ :: *linordered-idom*) $\leq$ *of-int*($\lceil a\rceil * \lceil b\rceil$)
⟨*proof*⟩

**lemma** *finite-int-segment*:
**fixes** *a* :: $'a$::*floor-ceiling*
**shows** *finite* $\{x \in \mathbb{Z}.\ a \leq x \wedge x \leq b\}$
⟨*proof*⟩

**corollary** *finite-abs-int-segment*:
**fixes** *a* :: $'a$::*floor-ceiling*
**shows** *finite* $\{k \in \mathbb{Z}.\ |k| \leq a\}$
⟨*proof*⟩

### 95.4.1 Ceiling with numerals.

**lemma** *ceiling-zero* [*simp*]: $\lceil 0\rceil = 0$
⟨*proof*⟩

**lemma** *ceiling-one* [*simp*]: $\lceil 1\rceil = 1$
⟨*proof*⟩

**lemma** *ceiling-numeral* [*simp*]: $\lceil$*numeral v*$\rceil$ = *numeral v*
⟨*proof*⟩

**lemma** *ceiling-neg-numeral* [*simp*]: $\lceil-$ *numeral v*$\rceil$ = $-$ *numeral v*
⟨*proof*⟩

**lemma** *ceiling-le-zero* [*simp*]: $\lceil x\rceil \leq 0 \longleftrightarrow x \leq 0$
⟨*proof*⟩

**lemma** *ceiling-le-one* [*simp*]: $\lceil x\rceil \leq 1 \longleftrightarrow x \leq 1$

⟨proof⟩

**lemma** *ceiling-le-numeral* [*simp*]: $\lceil x \rceil \leq$ *numeral v* $\longleftrightarrow$ $x \leq$ *numeral v*
⟨proof⟩

**lemma** *ceiling-le-neg-numeral* [*simp*]: $\lceil x \rceil \leq -$ *numeral v* $\longleftrightarrow$ $x \leq -$ *numeral v*
⟨proof⟩

**lemma** *ceiling-less-zero* [*simp*]: $\lceil x \rceil < 0 \longleftrightarrow x \leq -1$
⟨proof⟩

**lemma** *ceiling-less-one* [*simp*]: $\lceil x \rceil < 1 \longleftrightarrow x \leq 0$
⟨proof⟩

**lemma** *ceiling-less-numeral* [*simp*]: $\lceil x \rceil <$ *numeral v* $\longleftrightarrow$ $x \leq$ *numeral v* $- 1$
⟨proof⟩

**lemma** *ceiling-less-neg-numeral* [*simp*]: $\lceil x \rceil < -$ *numeral v* $\longleftrightarrow$ $x \leq -$ *numeral v* $- 1$
⟨proof⟩

**lemma** *zero-le-ceiling* [*simp*]: $0 \leq \lceil x \rceil \longleftrightarrow -1 < x$
⟨proof⟩

**lemma** *one-le-ceiling* [*simp*]: $1 \leq \lceil x \rceil \longleftrightarrow 0 < x$
⟨proof⟩

**lemma** *numeral-le-ceiling* [*simp*]: *numeral v* $\leq \lceil x \rceil \longleftrightarrow$ *numeral v* $- 1 < x$
⟨proof⟩

**lemma** *neg-numeral-le-ceiling* [*simp*]: $-$ *numeral v* $\leq \lceil x \rceil \longleftrightarrow -$ *numeral v* $- 1 < x$
⟨proof⟩

**lemma** *zero-less-ceiling* [*simp*]: $0 < \lceil x \rceil \longleftrightarrow 0 < x$
⟨proof⟩

**lemma** *one-less-ceiling* [*simp*]: $1 < \lceil x \rceil \longleftrightarrow 1 < x$
⟨proof⟩

**lemma** *numeral-less-ceiling* [*simp*]: *numeral v* $< \lceil x \rceil \longleftrightarrow$ *numeral v* $< x$
⟨proof⟩

**lemma** *neg-numeral-less-ceiling* [*simp*]: $-$ *numeral v* $< \lceil x \rceil \longleftrightarrow -$ *numeral v* $< x$
⟨proof⟩

**lemma** *ceiling-altdef*: $\lceil x \rceil = ($*if* $x =$ *of-int* $\lfloor x \rfloor$ *then* $\lfloor x \rfloor$ *else* $\lfloor x \rfloor + 1)$
⟨proof⟩

**lemma** *floor-le-ceiling* [*simp*]: $\lfloor x \rfloor \le \lceil x \rceil$
⟨*proof*⟩

### 95.4.2 Addition and subtraction of integers.

**lemma** *ceiling-add-of-int* [*simp*]: $\lceil x + \textit{of-int } z \rceil = \lceil x \rceil + z$
⟨*proof*⟩

**lemma** *ceiling-add-numeral* [*simp*]: $\lceil x + \textit{numeral } v \rceil = \lceil x \rceil + \textit{numeral } v$
⟨*proof*⟩

**lemma** *ceiling-add-one* [*simp*]: $\lceil x + 1 \rceil = \lceil x \rceil + 1$
⟨*proof*⟩

**lemma** *ceiling-diff-of-int* [*simp*]: $\lceil x - \textit{of-int } z \rceil = \lceil x \rceil - z$
⟨*proof*⟩

**lemma** *ceiling-diff-numeral* [*simp*]: $\lceil x - \textit{numeral } v \rceil = \lceil x \rceil - \textit{numeral } v$
⟨*proof*⟩

**lemma** *ceiling-diff-one* [*simp*]: $\lceil x - 1 \rceil = \lceil x \rceil - 1$
⟨*proof*⟩

**lemma** *ceiling-split*[*linarith-split*]: $P \lceil t \rceil \longleftrightarrow (\forall i.\ \textit{of-int } i - 1 < t \wedge t \le \textit{of-int } i \longrightarrow P\ i)$
⟨*proof*⟩

**lemma** *ceiling-eq-imp-diff-1*: $\lceil x \rceil = \lceil y \rceil \Longrightarrow |x-y| < 1$
⟨*proof*⟩

**lemma** *ceiling-diff-floor-le-1*: $\lceil x \rceil - \lfloor x \rfloor \le 1$
⟨*proof*⟩

**lemma** *floor-eq-ceiling-imp-diff-2*: $\lfloor x \rfloor = \lceil y \rceil \Longrightarrow |x-y| < 2$
⟨*proof*⟩

**lemma** *nat-approx-posE*:
**fixes** *e*:: *'a*::{*archimedean-field*,*floor-ceiling*}
**assumes** $0 < e$
**obtains** *n* :: *nat* **where** $1 / \textit{of-nat}(\textit{Suc } n) < e$
⟨*proof*⟩

**lemma** *ceiling-divide-upper*:
**fixes** *q* :: *'a*::*floor-ceiling*
**shows** $q > 0 \Longrightarrow p \le \textit{of-int } (\textit{ceiling } (p / q)) * q$
⟨*proof*⟩

**lemma** *ceiling-divide-lower*:
**fixes** *q* :: *'a*::*floor-ceiling*

**shows** $q > 0 \Longrightarrow (\mathit{of\text{-}int}\ \lceil p\ /\ q \rceil - 1) * q < p$
⟨proof⟩

## 95.5 Negation

**lemma** *floor-minus*: $\lfloor - x \rfloor = - \lceil x \rceil$
⟨proof⟩

**lemma** *ceiling-minus*: $\lceil - x \rceil = - \lfloor x \rfloor$
⟨proof⟩

## 95.6 Natural numbers

**lemma** *of-nat-floor*: $r \geq 0 \Longrightarrow \mathit{of\text{-}nat}\ (\mathit{nat}\ \lfloor r \rfloor) \leq r$
⟨proof⟩

**lemma** *of-nat-ceiling*: $\mathit{of\text{-}nat}\ (\mathit{nat}\ \lceil r \rceil) \geq r$
⟨proof⟩

**lemma** *of-nat-int-floor* [*simp*]: $x \geq 0 \Longrightarrow \mathit{of\text{-}nat}\ (\mathit{nat} \lfloor x \rfloor) = \mathit{of\text{-}int}\ \lfloor x \rfloor$
⟨proof⟩

**lemma** *of-nat-int-ceiling* [*simp*]: $x \geq 0 \Longrightarrow \mathit{of\text{-}nat}\ (\mathit{nat}\ \lceil x \rceil) = \mathit{of\text{-}int}\ \lceil x \rceil$
⟨proof⟩

## 95.7 Frac Function

**definition** *frac* :: $'a \Rightarrow 'a$::*floor-ceiling*
**where** $\mathit{frac}\ x \equiv x - \mathit{of\text{-}int}\ \lfloor x \rfloor$

**lemma** *frac-lt-1*: $\mathit{frac}\ x < 1$
⟨proof⟩

**lemma** *frac-eq-0-iff* [*simp*]: $\mathit{frac}\ x = 0 \longleftrightarrow x \in \mathbb{Z}$
⟨proof⟩

**lemma** *frac-ge-0* [*simp*]: $\mathit{frac}\ x \geq 0$
⟨proof⟩

**lemma** *frac-gt-0-iff* [*simp*]: $\mathit{frac}\ x > 0 \longleftrightarrow x \notin \mathbb{Z}$
⟨proof⟩

**lemma** *frac-of-int* [*simp*]: $\mathit{frac}\ (\mathit{of\text{-}int}\ z) = 0$
⟨proof⟩

**lemma** *frac-frac* [*simp*]: $\mathit{frac}\ (\mathit{frac}\ x) = \mathit{frac}\ x$
⟨proof⟩

**lemma** *floor-add*: $\lfloor x + y \rfloor = (\mathit{if}\ \mathit{frac}\ x + \mathit{frac}\ y < 1\ \mathit{then}\ \lfloor x \rfloor + \lfloor y \rfloor\ \mathit{else}\ (\lfloor x \rfloor + \lfloor y \rfloor) + 1)$

⟨*proof*⟩

**lemma** *floor-add2*[*simp*]: $x \in \mathbb{Z} \vee y \in \mathbb{Z} \Longrightarrow \lfloor x + y \rfloor = \lfloor x \rfloor + \lfloor y \rfloor$
⟨*proof*⟩

**lemma** *frac-add*:
$frac\ (x + y) = (if\ frac\ x + frac\ y < 1\ then\ frac\ x + frac\ y\ else\ (frac\ x + frac\ y) - 1)$
⟨*proof*⟩

**lemma** *frac-unique-iff*: $frac\ x = a \longleftrightarrow x - a \in \mathbb{Z} \wedge 0 \leq a \wedge a < 1$
**for** $x :: 'a::floor\text{-}ceiling$
⟨*proof*⟩

**lemma** *frac-eq*: $frac\ x = x \longleftrightarrow 0 \leq x \wedge x < 1$
⟨*proof*⟩

**lemma** *frac-eq-id* [*simp*]: $x \in \{0..<1\} \Longrightarrow frac\ x = x$
⟨*proof*⟩

**lemma** *frac-in-Ints-iff* [*simp*]: $frac\ x \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *frac-neg*: $frac\ (-\ x) = (if\ x \in \mathbb{Z}\ then\ 0\ else\ 1 - frac\ x)$
**for** $x :: 'a::floor\text{-}ceiling$
⟨*proof*⟩

**lemma** *frac-1-eq*: $frac\ (x+1) = frac\ x$
⟨*proof*⟩

## 95.8 Fractional part arithmetic

Many thanks to Stepan Holub

**lemma** *frac-non-zero*: $frac\ x \neq 0 \Longrightarrow frac\ (-x) = 1 - frac\ x$
⟨*proof*⟩

**lemma** *frac-add-simps* [*simp*]:
$frac\ (frac\ a + b) = frac\ (a + b)$
$frac\ (a + frac\ b) = frac\ (a + b)$
⟨*proof*⟩

**lemma** *frac-neg-frac*: $frac\ (-\ frac\ x) = frac\ (-x)$
⟨*proof*⟩

**lemma** *frac-diff-simp*: $frac\ (y - frac\ x) = frac\ (y - x)$
⟨*proof*⟩

**lemma** *frac-diff*: $frac\ (a - b) = frac\ (frac\ a + (-\ frac\ b))$
⟨*proof*⟩

**lemma** *frac-diff-pos*: $frac\ x \leq frac\ y \Longrightarrow frac\ (y - x) = frac\ y - frac\ x$
⟨*proof*⟩

**lemma** *frac-diff-neg*: **assumes** $frac\ y < frac\ x$
**shows** $frac\ (y - x) = frac\ y + 1 - frac\ x$
⟨*proof*⟩

**lemma** *frac-diff-eq*: **assumes** $frac\ y = frac\ x$
**shows** $frac\ (y - x) = 0$
⟨*proof*⟩

**lemma** *frac-diff-zero*: **assumes** $frac\ (x - y) = 0$
**shows** $frac\ x = frac\ y$
⟨*proof*⟩

**lemma** *frac-neg-eq-iff*: $frac\ (-x) = frac\ (-y) \longleftrightarrow frac\ x = frac\ y$
⟨*proof*⟩

**lemma** *frac-eqE*:
**assumes** $frac\ x = frac\ y$
**obtains** $n$ **where** $x = y + of\text{-}int\ n$
⟨*proof*⟩

**lemma** *frac-add-of-int-right* [*simp*]: $frac\ (x + of\text{-}int\ n) = frac\ x$
⟨*proof*⟩

**lemma** *frac-add-of-int-left* [*simp*]: $frac\ (of\text{-}int\ n + x) = frac\ x$
⟨*proof*⟩

**lemma** *frac-add-int-right*: $y \in \mathbb{Z} \Longrightarrow frac\ (x + y) = frac\ x$
⟨*proof*⟩

**lemma** *frac-add-int-left*: $x \in \mathbb{Z} \Longrightarrow frac\ (x + y) = frac\ y$
⟨*proof*⟩

**lemma** *bij-betw-frac*: $bij\text{-}betw\ frac\ \{x..<x+1\}\ \{0..<1\}$
⟨*proof*⟩

## 95.9 Rounding to the nearest integer

**definition** $round :: 'a::floor\text{-}ceiling \Rightarrow int$
**where** $round\ x = \lfloor x + 1/2 \rfloor$

**lemma** *round-eq-imp-diff-1*: $round\ x = round\ y \Longrightarrow |x-y| < 1$
⟨*proof*⟩

**lemma** *of-int-round-ge*: $of\text{-}int\ (round\ x) \geq x - 1/2$
**and** *of-int-round-le*: $of\text{-}int\ (round\ x) \leq x + 1/2$

**and** *of-int-round-abs-le*: $|of\text{-}int\ (round\ x) - x| \le 1/2$
**and** *of-int-round-gt*: $of\text{-}int\ (round\ x) > x - 1/2$
⟨*proof*⟩

**lemma** *round-of-int* [*simp*]: $round\ (of\text{-}int\ n) = n$
⟨*proof*⟩

**lemma** *round-0* [*simp*]: $round\ 0 = 0$
⟨*proof*⟩

**lemma** *round-1* [*simp*]: $round\ 1 = 1$
⟨*proof*⟩

**lemma** *round-numeral* [*simp*]: $round\ (numeral\ n) = numeral\ n$
⟨*proof*⟩

**lemma** *round-neg-numeral* [*simp*]: $round\ (-numeral\ n) = -numeral\ n$
⟨*proof*⟩

**lemma** *round-of-nat* [*simp*]: $round\ (of\text{-}nat\ n) = of\text{-}nat\ n$
⟨*proof*⟩

**lemma** *round-mono*: $x \le y \Longrightarrow round\ x \le round\ y$
⟨*proof*⟩

**lemma** *round-unique*: $of\text{-}int\ y > x - 1/2 \Longrightarrow of\text{-}int\ y \le x + 1/2 \Longrightarrow round\ x = y$
⟨*proof*⟩

**lemma** *round-unique'*: $|x - of\text{-}int\ n| < 1/2 \Longrightarrow round\ x = n$
⟨*proof*⟩

**lemma** *round-altdef*: $round\ x = (if\ frac\ x \ge 1/2\ then\ \lceil x\rceil\ else\ \lfloor x\rfloor)$
⟨*proof*⟩

**lemma** *floor-le-round*: $\lfloor x\rfloor \le round\ x$
⟨*proof*⟩

**lemma** *ceiling-ge-round*: $\lceil x\rceil \ge round\ x$
⟨*proof*⟩

**lemma** *round-diff-minimal*: $|z - of\text{-}int\ (round\ z)| \le |z - of\text{-}int\ m|$
**for** *z* :: *'a::floor-ceiling*
⟨*proof*⟩

**bundle** *floor-ceiling-syntax*
**begin**
**notation** *floor* (‹(‹*open-block notation*=‹*mixfix floor*››⌊-⌋)›)
**and** *ceiling* (‹(‹*open-block notation*=‹*mixfix ceiling*››⌈-⌉)›)

**end**

**end**

# 96 Rational numbers

**theory** *Rat*
**imports** *Archimedean-Field*
**begin**

## 96.1 Rational numbers as quotient

### 96.1.1 Construction of the type of rational numbers

**definition** *ratrel* :: $(int \times int) \Rightarrow (int \times int) \Rightarrow bool$
**where** $ratrel = (\lambda x\ y.\ snd\ x \neq 0 \wedge snd\ y \neq 0 \wedge fst\ x * snd\ y = fst\ y * snd\ x)$

**lemma** *ratrel-iff* [*simp*]: $ratrel\ x\ y \longleftrightarrow snd\ x \neq 0 \wedge snd\ y \neq 0 \wedge fst\ x * snd\ y = fst\ y * snd\ x$
⟨*proof*⟩

**lemma** *exists-ratrel-refl*: $\exists x.\ ratrel\ x\ x$
⟨*proof*⟩

**lemma** *symp-ratrel*: *symp ratrel*
⟨*proof*⟩

**lemma** *transp-ratrel*: *transp ratrel*
⟨*proof*⟩

**lemma** *part-equivp-ratrel*: *part-equivp ratrel*
⟨*proof*⟩

**quotient-type** $rat = int \times int\ /\ partial$: *ratrel*
**morphisms** *Rep-Rat Abs-Rat*
⟨*proof*⟩

**lemma** *Domainp-cr-rat* [*transfer-domain-rule*]: $Domainp\ pcr\text{-}rat = (\lambda x.\ snd\ x \neq 0)$
⟨*proof*⟩

### 96.1.2 Representation and basic operations

**lift-definition** *Fract* :: $int \Rightarrow int \Rightarrow rat$
**is** $\lambda a\ b.\ if\ b = 0\ then\ (0,\ 1)\ else\ (a,\ b)$
⟨*proof*⟩

**lemma** *eq-rat*:
$\bigwedge a\ b\ c\ d.\ b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow Fract\ a\ b = Fract\ c\ d \longleftrightarrow a * d = c * b$

$\bigwedge a.\ Fract\ a\ 0 = Fract\ 0\ 1$
$\bigwedge a\ c.\ Fract\ 0\ a = Fract\ 0\ c$
⟨proof⟩

**lemma** *Rat-cases* [*case-names Fract*, *cases type*: *rat*]:
**assumes** *that*: $\bigwedge a\ b.\ q = Fract\ a\ b \Longrightarrow b > 0 \Longrightarrow coprime\ a\ b \Longrightarrow C$
**shows** $C$
⟨proof⟩

**lemma** *Rat-induct* [*case-names Fract*, *induct type*: *rat*]:
**assumes** $\bigwedge a\ b.\ b > 0 \Longrightarrow coprime\ a\ b \Longrightarrow P\ (Fract\ a\ b)$
**shows** $P\ q$
⟨proof⟩

**instantiation** *rat* :: *field*
**begin**

**lift-definition** *zero-rat* :: *rat* **is** $(0,\ 1)$
⟨proof⟩

**lift-definition** *one-rat* :: *rat* **is** $(1,\ 1)$
⟨proof⟩

**lemma** *Zero-rat-def*: $0 = Fract\ 0\ 1$
⟨proof⟩

**lemma** *One-rat-def*: $1 = Fract\ 1\ 1$
⟨proof⟩

**lift-definition** *plus-rat* :: $rat \Rightarrow rat \Rightarrow rat$
**is** $\lambda x\ y.\ (fst\ x * snd\ y + fst\ y * snd\ x,\ snd\ x * snd\ y)$
⟨proof⟩

**lemma** *add-rat* [*simp*]:
**assumes** $b \neq 0$ **and** $d \neq 0$
**shows** $Fract\ a\ b + Fract\ c\ d = Fract\ (a * d + c * b)\ (b * d)$
⟨proof⟩

**lift-definition** *uminus-rat* :: $rat \Rightarrow rat$ **is** $\lambda x.\ (-\ fst\ x,\ snd\ x)$
⟨proof⟩

**lemma** *minus-rat* [*simp*]: $-\ Fract\ a\ b = Fract\ (-\ a)\ b$
⟨proof⟩

**lemma** *minus-rat-cancel* [*simp*]: $Fract\ (-\ a)\ (-\ b) = Fract\ a\ b$
⟨proof⟩

**definition** *diff-rat-def*: $q - r = q + -\ r$ **for** $q\ r$ :: *rat*

**lemma** *diff-rat* [*simp*]:
$b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow Fract\ a\ b - Fract\ c\ d = Fract\ (a * d - c * b)\ (b * d)$
⟨*proof*⟩

**lift-definition** *times-rat* :: $rat \Rightarrow rat \Rightarrow rat$
**is** $\lambda x\ y.\ (fst\ x * fst\ y,\ snd\ x * snd\ y)$
⟨*proof*⟩

**lemma** *mult-rat* [*simp*]: $Fract\ a\ b * Fract\ c\ d = Fract\ (a * c)\ (b * d)$
⟨*proof*⟩

**lemma** *mult-rat-cancel*: $c \neq 0 \Longrightarrow Fract\ (c * a)\ (c * b) = Fract\ a\ b$
⟨*proof*⟩

**lift-definition** *inverse-rat* :: $rat \Rightarrow rat$
**is** $\lambda x.$ *if* $fst\ x = 0$ *then* $(0,\ 1)$ *else* $(snd\ x,\ fst\ x)$
⟨*proof*⟩

**lemma** *inverse-rat* [*simp*]: $inverse\ (Fract\ a\ b) = Fract\ b\ a$
⟨*proof*⟩

**definition** *divide-rat-def*: $q\ div\ r = q * inverse\ r$ **for** $q\ r$ :: *rat*

**lemma** *divide-rat* [*simp*]: $Fract\ a\ b\ div\ Fract\ c\ d = Fract\ (a * d)\ (b * c)$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**lemma** *div-add-self1-no-field* [*simp*]:
**assumes** *NO-MATCH* $(x :: 'b :: field)\ b\ (b :: 'a :: euclidean\text{-}semiring\text{-}cancel) \neq 0$
**shows** $(b + a)\ div\ b = a\ div\ b + 1$
⟨*proof*⟩

**lemma** *div-add-self2-no-field* [*simp*]:
**assumes** *NO-MATCH* $(x :: 'b :: field)\ b\ (b :: 'a :: euclidean\text{-}semiring\text{-}cancel) \neq 0$
**shows** $(a + b)\ div\ b = a\ div\ b + 1$
⟨*proof*⟩

**lemma** *of-nat-rat*: $of\text{-}nat\ k = Fract\ (of\text{-}nat\ k)\ 1$
⟨*proof*⟩

**lemma** *of-int-rat*: $of\text{-}int\ k = Fract\ k\ 1$
⟨*proof*⟩

**lemma** *Fract-of-nat-eq*: $Fract\ (of\text{-}nat\ k)\ 1 = of\text{-}nat\ k$
⟨*proof*⟩

**lemma** *Fract-of-int-eq*: $Fract\ k\ 1 = of\text{-}int\ k$
⟨*proof*⟩

**lemma** *rat-number-collapse*:
$Fract\ 0\ k = 0$
$Fract\ 1\ 1 = 1$
$Fract\ (numeral\ w)\ 1 = numeral\ w$
$Fract\ (-\ numeral\ w)\ 1 = -\ numeral\ w$
$Fract\ (-\ 1)\ 1 = -\ 1$
$Fract\ k\ 0 = 0$
⟨*proof*⟩

**lemma** *rat-number-expand*:
$0 = Fract\ 0\ 1$
$1 = Fract\ 1\ 1$
$numeral\ k = Fract\ (numeral\ k)\ 1$
$-\ 1 = Fract\ (-\ 1)\ 1$
$-\ numeral\ k = Fract\ (-\ numeral\ k)\ 1$
⟨*proof*⟩

**lemma** *Rat-cases-nonzero* [*case-names Fract 0*]:
**assumes** *Fract*: $\bigwedge a\ b.\ q = Fract\ a\ b \Longrightarrow b > 0 \Longrightarrow a \neq 0 \Longrightarrow coprime\ a\ b \Longrightarrow C$
**and** *0*: $q = 0 \Longrightarrow C$
**shows** $C$
⟨*proof*⟩

### 96.1.3 Function *normalize*

**lemma** *Fract-coprime*: $Fract\ (a\ div\ gcd\ a\ b)\ (b\ div\ gcd\ a\ b) = Fract\ a\ b$
⟨*proof*⟩

**definition** $normalize :: int \times int \Rightarrow int \times int$
**where** $normalize\ p =$
$(if\ snd\ p > 0\ then\ (let\ a = gcd\ (fst\ p)\ (snd\ p)\ in\ (fst\ p\ div\ a,\ snd\ p\ div\ a))$
$else\ if\ snd\ p = 0\ then\ (0,\ 1)$
$else\ (let\ a = -\ gcd\ (fst\ p)\ (snd\ p)\ in\ (fst\ p\ div\ a,\ snd\ p\ div\ a)))$

**lemma** *normalize-crossproduct*:
**assumes** $q \neq 0\ s \neq 0$
**assumes** $normalize\ (p,\ q) = normalize\ (r,\ s)$
**shows** $p * s = r * q$
⟨*proof*⟩

**lemma** *normalize-eq*: $normalize\ (a,\ b) = (p,\ q) \Longrightarrow Fract\ p\ q = Fract\ a\ b$

⟨*proof*⟩

**lemma** *normalize-denom-pos*: *normalize r* = (*p*, *q*) ⟹ *q* > *0*
⟨*proof*⟩

**lemma** *normalize-coprime*: *normalize r* = (*p*, *q*) ⟹ *coprime p q*
⟨*proof*⟩

**lemma** *normalize-stable* [*simp*]: *q* > *0* ⟹ *coprime p q* ⟹ *normalize* (*p*, *q*) = (*p*, *q*)
⟨*proof*⟩

**lemma** *normalize-denom-zero* [*simp*]: *normalize* (*p*, *0*) = (*0*, *1*)
⟨*proof*⟩

**lemma** *normalize-negative* [*simp*]: *q* < *0* ⟹ *normalize* (*p*, *q*) = *normalize* (− *p*, − *q*)
⟨*proof*⟩

Decompose a fraction into normalized, i.e. coprime numerator and denominator:

**definition** *quotient-of* :: *rat* ⇒ *int* × *int*
**where** *quotient-of x* =
(*THE pair*. *x* = *Fract* (*fst pair*) (*snd pair*) ∧ *snd pair* > *0* ∧ *coprime* (*fst pair*) (*snd pair*))

**lemma** *quotient-of-unique*: ∃!*p*. *r* = *Fract* (*fst p*) (*snd p*) ∧ *snd p* > *0* ∧ *coprime* (*fst p*) (*snd p*)
⟨*proof*⟩

**lemma** *quotient-of-Fract* [*code*]: *quotient-of* (*Fract a b*) = *normalize* (*a*, *b*)
⟨*proof*⟩

**lemma** *quotient-of-number* [*simp*]:
*quotient-of 0* = (*0*, *1*)
*quotient-of 1* = (*1*, *1*)
*quotient-of* (*numeral k*) = (*numeral k*, *1*)
*quotient-of* (− *1*) = (− *1*, *1*)
*quotient-of* (− *numeral k*) = (− *numeral k*, *1*)
⟨*proof*⟩

**lemma** *quotient-of-eq*: *quotient-of* (*Fract a b*) = (*p*, *q*) ⟹ *Fract p q* = *Fract a b*
⟨*proof*⟩

**lemma** *quotient-of-denom-pos*: *quotient-of r* = (*p*, *q*) ⟹ *q* > *0*
⟨*proof*⟩

**lemma** *quotient-of-denom-pos'*: *snd* (*quotient-of r*) > *0*
⟨*proof*⟩

**lemma** *quotient-of-coprime*: *quotient-of* $r = (p, q) \Longrightarrow$ *coprime* $p$ $q$
⟨*proof*⟩

**lemma** *quotient-of-inject*:
**assumes** *quotient-of* $a$ = *quotient-of* $b$
**shows** $a = b$
⟨*proof*⟩

**lemma** *quotient-of-inject-eq*: *quotient-of* $a$ = *quotient-of* $b \longleftrightarrow a = b$
⟨*proof*⟩

### 96.1.4 Various

**lemma** *Fract-of-int-quotient*: *Fract* $k$ $l$ = *of-int* $k$ / *of-int* $l$
⟨*proof*⟩

**lemma** *Fract-add-one*: $n \neq 0 \Longrightarrow$ *Fract* $(m + n)$ $n$ = *Fract* $m$ $n + 1$
⟨*proof*⟩

**lemma** *quotient-of-div*:
**assumes** $r$: *quotient-of* $r = (n,d)$
**shows** $r$ = *of-int* $n$ / *of-int* $d$
⟨*proof*⟩

**lemma** *Fract-quotient-of* [*simp*]: *Fract* (*fst* (*quotient-of* $r$)) (*snd* (*quotient-of* $r$)) $= r$
⟨*proof*⟩

### 96.1.5 The ordered field of rational numbers

**lift-definition** *positive* :: *rat* ⇒ *bool*
**is** $\lambda x.\ 0 <$ *fst* $x$ ∗ *snd* $x$
⟨*proof*⟩

**lemma** *positive-zero*: ¬ *positive* *0*
⟨*proof*⟩

**lemma** *positive-add*: *positive* $x \Longrightarrow$ *positive* $y \Longrightarrow$ *positive* $(x + y)$
⟨*proof*⟩

**lemma** *positive-mult*: *positive* $x \Longrightarrow$ *positive* $y \Longrightarrow$ *positive* $(x * y)$
⟨*proof*⟩

**lemma** *positive-minus*: ¬ *positive* $x \Longrightarrow x \neq 0 \Longrightarrow$ *positive* $(- x)$
⟨*proof*⟩

**instantiation** *rat* :: *linordered-field*
**begin**

**definition** $x < y \longleftrightarrow$ *positive* $(y - x)$

**definition** $x \le y \longleftrightarrow x < y \vee x = y$ **for** $x\ y :: rat$

**definition** $|a| = (\mathit{if}\ a < 0\ \mathit{then} - a\ \mathit{else}\ a)$ **for** $a :: rat$

**definition** $sgn\ a = (\mathit{if}\ a = 0\ \mathit{then}\ 0\ \mathit{else}\ \mathit{if}\ 0 < a\ \mathit{then}\ 1\ \mathit{else} - 1)$ **for** $a :: rat$

**instance**
⟨*proof*⟩

**end**

**instantiation** *rat* :: *distrib-lattice*
**begin**

**definition** $(\mathit{inf} :: rat \Rightarrow rat \Rightarrow rat) = min$

**definition** $(\mathit{sup} :: rat \Rightarrow rat \Rightarrow rat) = max$

**instance**
⟨*proof*⟩

**end**

**lemma** *positive-rat*: *positive* $(\mathit{Fract}\ a\ b) \longleftrightarrow 0 < a * b$
⟨*proof*⟩

**lemma** *less-rat* [*simp*]:
$b \ne 0 \Longrightarrow d \ne 0 \Longrightarrow \mathit{Fract}\ a\ b < \mathit{Fract}\ c\ d \longleftrightarrow (a * d) * (b * d) < (c * b) * (b * d)$
⟨*proof*⟩

**lemma** *le-rat* [*simp*]:
$b \ne 0 \Longrightarrow d \ne 0 \Longrightarrow \mathit{Fract}\ a\ b \le \mathit{Fract}\ c\ d \longleftrightarrow (a * d) * (b * d) \le (c * b) * (b * d)$
⟨*proof*⟩

**lemma** *abs-rat* [*simp*, *code*]: $|\mathit{Fract}\ a\ b| = \mathit{Fract}\ |a|\ |b|$
⟨*proof*⟩

**lemma** *sgn-rat* [*simp*, *code*]: $sgn\ (\mathit{Fract}\ a\ b) = \mathit{of\text{-}int}\ (sgn\ a * sgn\ b)$
⟨*proof*⟩

**lemma** *Rat-induct-pos* [*case-names Fract*, *induct type*: *rat*]:
**assumes** *step*: $\bigwedge a\ b.\ 0 < b \Longrightarrow P\ (\mathit{Fract}\ a\ b)$
**shows** $P\ q$
⟨*proof*⟩

**lemma** *zero-less-Fract-iff*: $0 < b \Longrightarrow 0 < \mathit{Fract}\ a\ b \longleftrightarrow 0 < a$
⟨*proof*⟩

**lemma** *Fract-less-zero-iff*: $0 < b \Longrightarrow \mathit{Fract}\ a\ b < 0 \longleftrightarrow a < 0$
⟨*proof*⟩

**lemma** *zero-le-Fract-iff*: $0 < b \Longrightarrow 0 \leq \mathit{Fract}\ a\ b \longleftrightarrow 0 \leq a$
⟨*proof*⟩

**lemma** *Fract-le-zero-iff*: $0 < b \Longrightarrow \mathit{Fract}\ a\ b \leq 0 \longleftrightarrow a \leq 0$
⟨*proof*⟩

**lemma** *one-less-Fract-iff*: $0 < b \Longrightarrow 1 < \mathit{Fract}\ a\ b \longleftrightarrow b < a$
⟨*proof*⟩

**lemma** *Fract-less-one-iff*: $0 < b \Longrightarrow \mathit{Fract}\ a\ b < 1 \longleftrightarrow a < b$
⟨*proof*⟩

**lemma** *one-le-Fract-iff*: $0 < b \Longrightarrow 1 \leq \mathit{Fract}\ a\ b \longleftrightarrow b \leq a$
⟨*proof*⟩

**lemma** *Fract-le-one-iff*: $0 < b \Longrightarrow \mathit{Fract}\ a\ b \leq 1 \longleftrightarrow a \leq b$
⟨*proof*⟩

### 96.1.6 Rationals are an Archimedean field

**lemma** *rat-floor-lemma*: $\mathit{of\text{-}int}\ (a\ \mathit{div}\ b) \leq \mathit{Fract}\ a\ b \wedge \mathit{Fract}\ a\ b < \mathit{of\text{-}int}\ (a\ \mathit{div}\ b + 1)$
⟨*proof*⟩

**instance** *rat* :: *archimedean-field*
⟨*proof*⟩

**instantiation** *rat* :: *floor-ceiling*
**begin**

**definition** *floor-rat* :: *rat* ⇒ *int*
**where** $\lfloor x \rfloor = (\mathit{THE}\ z.\ \mathit{of\text{-}int}\ z \leq x \wedge x < \mathit{of\text{-}int}\ (z + 1))$ **for** *x* :: *rat*

**instance**
⟨*proof*⟩

**end**

**lemma** *floor-Fract* [*simp*]: $\lfloor \mathit{Fract}\ a\ b \rfloor = a\ \mathit{div}\ b$
⟨*proof*⟩

## 96.2 Linear arithmetic setup

⟨*ML*⟩

## 96.3 Embedding from Rationals to other Fields

**context** *field-char-0*
**begin**

**lift-definition** *of-rat* :: *rat* ⇒ *'a*
  **is** $\lambda x.$ *of-int* (*fst x*) / *of-int* (*snd x*)
  ⟨*proof*⟩

**end**

**lemma** *of-rat-rat*: $b \neq 0 \Longrightarrow$ *of-rat* (*Fract a b*) = *of-int a* / *of-int b*
  ⟨*proof*⟩

**lemma** *of-rat-0* [*simp*]: *of-rat 0* = *0*
  ⟨*proof*⟩

**lemma** *of-rat-1* [*simp*]: *of-rat 1* = *1*
  ⟨*proof*⟩

**lemma** *of-rat-add*: *of-rat* (*a* + *b*) = *of-rat a* + *of-rat b*
  ⟨*proof*⟩

**lemma** *of-rat-minus*: *of-rat* (− *a*) = − *of-rat a*
  ⟨*proof*⟩

**lemma** *of-rat-neg-one* [*simp*]: *of-rat* (− *1*) = − *1*
  ⟨*proof*⟩

**lemma** *of-rat-diff*: *of-rat* (*a* − *b*) = *of-rat a* − *of-rat b*
  ⟨*proof*⟩

**lemma** *of-rat-mult*: *of-rat* (*a* ∗ *b*) = *of-rat a* ∗ *of-rat b*
  ⟨*proof*⟩

**lemma** *of-rat-sum*: *of-rat* ($\sum$ *a*∈*A*. *f a*) = ($\sum$ *a*∈*A*. *of-rat* (*f a*))
  ⟨*proof*⟩

**lemma** *of-rat-prod*: *of-rat* ($\prod$ *a*∈*A*. *f a*) = ($\prod$ *a*∈*A*. *of-rat* (*f a*))
  ⟨*proof*⟩

**lemma** *nonzero-of-rat-inverse*: $a \neq 0 \Longrightarrow$ *of-rat* (*inverse a*) = *inverse* (*of-rat a*)
  ⟨*proof*⟩

**lemma** *of-rat-inverse*: (*of-rat* (*inverse a*) :: *'a::field-char-0*) = *inverse* (*of-rat a*)
  ⟨*proof*⟩

**lemma** *nonzero-of-rat-divide*: $b \neq 0 \Longrightarrow$ *of-rat* (*a* / *b*) = *of-rat a* / *of-rat b*
  ⟨*proof*⟩

**lemma** *of-rat-divide*: $(\textit{of-rat}\ (a\ /\ b) :: {}'a::\textit{field-char-0}) = \textit{of-rat}\ a\ /\ \textit{of-rat}\ b$
⟨*proof*⟩

**lemma** *of-rat-power*: $(\textit{of-rat}\ (a\ \widehat{}\ n) :: {}'a::\textit{field-char-0}) = \textit{of-rat}\ a\ \widehat{}\ n$
⟨*proof*⟩

**lemma** *of-rat-eq-iff* [*simp*]: $\textit{of-rat}\ a = \textit{of-rat}\ b \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** *of-rat-eq-0-iff* [*simp*]: $\textit{of-rat}\ a = \mathit{0} \longleftrightarrow a = \mathit{0}$
⟨*proof*⟩

**lemma** *zero-eq-of-rat-iff* [*simp*]: $\mathit{0} = \textit{of-rat}\ a \longleftrightarrow \mathit{0} = a$
⟨*proof*⟩

**lemma** *of-rat-eq-1-iff* [*simp*]: $\textit{of-rat}\ a = \mathit{1} \longleftrightarrow a = \mathit{1}$
⟨*proof*⟩

**lemma** *one-eq-of-rat-iff* [*simp*]: $\mathit{1} = \textit{of-rat}\ a \longleftrightarrow \mathit{1} = a$
⟨*proof*⟩

**lemma** *of-rat-less*: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) < \textit{of-rat}\ s \longleftrightarrow r < s$
⟨*proof*⟩

**lemma** *of-rat-less-eq*: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \le \textit{of-rat}\ s \longleftrightarrow r \le s$
⟨*proof*⟩

**lemma** *of-rat-le-0-iff* [*simp*]: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \le \mathit{0} \longleftrightarrow r \le \mathit{0}$
⟨*proof*⟩

**lemma** *zero-le-of-rat-iff* [*simp*]: $\mathit{0} \le (\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \longleftrightarrow \mathit{0} \le r$
⟨*proof*⟩

**lemma** *of-rat-le-1-iff* [*simp*]: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \le \mathit{1} \longleftrightarrow r \le \mathit{1}$
⟨*proof*⟩

**lemma** *one-le-of-rat-iff* [*simp*]: $\mathit{1} \le (\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \longleftrightarrow \mathit{1} \le r$
⟨*proof*⟩

**lemma** *of-rat-less-0-iff* [*simp*]: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) < \mathit{0} \longleftrightarrow r < \mathit{0}$
⟨*proof*⟩

**lemma** *zero-less-of-rat-iff* [*simp*]: $\mathit{0} < (\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \longleftrightarrow \mathit{0} < r$
⟨*proof*⟩

**lemma** *of-rat-less-1-iff* [*simp*]: $(\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) < \mathit{1} \longleftrightarrow r < \mathit{1}$
⟨*proof*⟩

**lemma** *one-less-of-rat-iff* [*simp*]: $\mathit{1} < (\textit{of-rat}\ r :: {}'a::\textit{linordered-field}) \longleftrightarrow \mathit{1} < r$

⟨*proof*⟩

**lemma** *of-rat-eq-id* [*simp*]: *of-rat* = *id*
⟨*proof*⟩

**lemma** *abs-of-rat* [*simp*]:
$|$*of-rat r*$|$ = (*of-rat* $|r|$ :: $'a$ :: *linordered-field*)
⟨*proof*⟩

Collapse nested embeddings.

**lemma** *of-rat-of-nat-eq* [*simp*]: *of-rat* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

**lemma** *of-rat-of-int-eq* [*simp*]: *of-rat* (*of-int z*) = *of-int z*
⟨*proof*⟩

**lemma** *of-rat-numeral-eq* [*simp*]: *of-rat* (*numeral w*) = *numeral w*
⟨*proof*⟩

**lemma** *of-rat-neg-numeral-eq* [*simp*]: *of-rat* (− *numeral w*) = − *numeral w*
⟨*proof*⟩

**lemma** *of-rat-floor* [*simp*]:
$\lfloor$*of-rat r*$\rfloor$ = $\lfloor r \rfloor$
⟨*proof*⟩

**lemma** *of-rat-ceiling* [*simp*]:
$\lceil$*of-rat r*$\rceil$ = $\lceil r \rceil$
⟨*proof*⟩

**lemmas** *zero-rat* = *Zero-rat-def*
**lemmas** *one-rat* = *One-rat-def*

**abbreviation** *rat-of-nat* :: *nat* ⇒ *rat*
**where** *rat-of-nat* ≡ *of-nat*

**abbreviation** *rat-of-int* :: *int* ⇒ *rat*
**where** *rat-of-int* ≡ *of-int*

## 96.4 The Set of Rational Numbers

**context** *field-char-0*
**begin**

**definition** *Rats* :: $'a$ *set* (‹ℚ›)
**where** ℚ = *range of-rat*

**end**

**lemma** *Rats-cases* [*cases set*: *Rats*]:
**assumes** $q \in \mathbb{Q}$
**obtains** (*of-rat*) $r$ **where** $q = \textit{of-rat}\ r$
⟨*proof*⟩

**lemma** *Rats-cases'*:
**assumes** $(x :: 'a :: \textit{field-char-0}) \in \mathbb{Q}$
**obtains** $a\ b$ **where** $b > 0$ *coprime* $a\ b$ $x = \textit{of-int}\ a\ /\ \textit{of-int}\ b$
⟨*proof*⟩

**lemma** *Rats-of-rat* [*simp*]: $\textit{of-rat}\ r \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-of-int* [*simp*]: $\textit{of-int}\ z \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Ints-subset-Rats*: $\mathbb{Z} \subseteq \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-of-nat* [*simp*]: $\textit{of-nat}\ n \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Nats-subset-Rats*: $\mathbb{N} \subseteq \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-number-of* [*simp*]: $\textit{numeral}\ w \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-0* [*simp*]: $0 \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-1* [*simp*]: $1 \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-add* [*simp*]: $a \in \mathbb{Q} \Longrightarrow b \in \mathbb{Q} \Longrightarrow a + b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-minus-iff* [*simp*]: $-\ a \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-diff* [*simp*]: $a \in \mathbb{Q} \Longrightarrow b \in \mathbb{Q} \Longrightarrow a - b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-mult* [*simp*]: $a \in \mathbb{Q} \Longrightarrow b \in \mathbb{Q} \Longrightarrow a * b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-inverse* [*simp*]: $a \in \mathbb{Q} \Longrightarrow \textit{inverse}\ a \in \mathbb{Q}$
**for** $a :: 'a{::}\textit{field-char-0}$
⟨*proof*⟩

**lemma** *Rats-divide* [*simp*]: $a \in \mathbb{Q} \Longrightarrow b \in \mathbb{Q} \Longrightarrow a \mathbin{/} b \in \mathbb{Q}$
**for** $a\ b$ :: *'a::field-char-0*
⟨*proof*⟩

**lemma** *Rats-power* [*simp*]: $a \in \mathbb{Q} \Longrightarrow a \mathbin{\hat{}} n \in \mathbb{Q}$
**for** $a$ :: *'a::field-char-0*
⟨*proof*⟩

**lemma** *Rats-sum* [*intro*]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}) \Longrightarrow \mathit{sum}\ f\ A \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-prod* [*intro*]: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}) \Longrightarrow \mathit{prod}\ f\ A \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-induct* [*case-names of-rat*, *induct set*: *Rats*]: $q \in \mathbb{Q} \Longrightarrow (\bigwedge r.\ P\ (\mathit{of\text{-}rat}\ r)) \Longrightarrow P\ q$
⟨*proof*⟩

**lemma** *Rats-infinite*: $\neg\ \mathit{finite}\ \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-add-iff*: $a \in \mathbb{Q} \lor b \in \mathbb{Q} \Longrightarrow a{+}b \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q} \land b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-diff-iff*: $a \in \mathbb{Q} \lor b \in \mathbb{Q} \Longrightarrow a{-}b \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q} \land b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-mult-iff*: $a \in \mathbb{Q}{-}\{0\} \lor b \in \mathbb{Q}{-}\{0\} \Longrightarrow a{*}b \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q} \land b \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-inverse-iff* [*simp*]: $\mathit{inverse}\ a \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-divide-iff*: $a \in \mathbb{Q}{-}\{0\} \lor b \in \mathbb{Q}{-}\{0\} \Longrightarrow a/b \in \mathbb{Q} \longleftrightarrow a \in \mathbb{Q} \land b \in \mathbb{Q}$
⟨*proof*⟩

## 96.5 Implementation of rational numbers as pairs of integers

Formal constructor

**definition** *Frct* :: $\mathit{int} \times \mathit{int} \Rightarrow \mathit{rat}$
**where** [*simp*]: $\mathit{Frct}\ p = \mathit{Fract}\ (\mathit{fst}\ p)\ (\mathit{snd}\ p)$

**lemma** [*code abstype*]: $\mathit{Frct}\ (\mathit{quotient\text{-}of}\ q) = q$
⟨*proof*⟩

Numerals

**declare** *quotient-of-Fract* [*code abstract*]

**definition** *of-int* :: *int* ⇒ *rat*
**where** [*code-abbrev*]: *of-int* = *Int.of-int*

**hide-const** (**open**) *of-int*

**lemma** *quotient-of-int* [*code abstract*]: *quotient-of* (*Rat.of-int a*) = (*a*, *1*)
⟨*proof*⟩

**lemma** *quotient-of-rat-of-int* [*simp*]: *quotient-of* (*rat-of-int i*) = (*i*, *1*)
⟨*proof*⟩

**lemma** *quotient-of-rat-of-nat* [*simp*]: *quotient-of* (*rat-of-nat i*) = (*int i*, *1*)
⟨*proof*⟩

**lemma** [*code-unfold*]: *numeral k* = *Rat.of-int* (*numeral k*)
⟨*proof*⟩

**lemma** [*code-unfold*]: − *numeral k* = *Rat.of-int* (− *numeral k*)
⟨*proof*⟩

**lemma** *Frct-code-post* [*code-post*]:
*Frct* (*0*, *a*) = *0*
*Frct* (*a*, *0*) = *0*
*Frct* (*1*, *1*) = *1*
*Frct* (*numeral k*, *1*) = *numeral k*
*Frct* (*1*, *numeral k*) = *1* / *numeral k*
*Frct* (*numeral k*, *numeral l*) = *numeral k* / *numeral l*
*Frct* (− *a*, *b*) = − *Frct* (*a*, *b*)
*Frct* (*a*, − *b*) = − *Frct* (*a*, *b*)
− (− *Frct q*) = *Frct q*
⟨*proof*⟩

Operations

**lemma** *rat-zero-code* [*code abstract*]: *quotient-of 0* = (*0*, *1*)
⟨*proof*⟩

**lemma** *rat-one-code* [*code abstract*]: *quotient-of 1* = (*1*, *1*)
⟨*proof*⟩

**lemma** *rat-plus-code* [*code abstract*]:
*quotient-of* (*p* + *q*) = (*let* (*a*, *c*) = *quotient-of p*; (*b*, *d*) = *quotient-of q*
*in normalize* (*a* ∗ *d* + *b* ∗ *c*, *c* ∗ *d*))
⟨*proof*⟩

**lemma** *rat-uminus-code* [*code abstract*]:
*quotient-of* (− *p*) = (*let* (*a*, *b*) = *quotient-of p in* (− *a*, *b*))
⟨*proof*⟩

**lemma** *rat-minus-code* [*code abstract*]:
  *quotient-of* $(p - q) =$
    $(let\ (a,\ c) = quotient\text{-}of\ p;\ (b,\ d) = quotient\text{-}of\ q$
     $in\ normalize\ (a * d - b * c,\ c * d))$
  ⟨*proof*⟩

**lemma** *rat-times-code* [*code abstract*]:
  *quotient-of* $(p * q) =$
    $(let\ (a,\ c) = quotient\text{-}of\ p;\ (b,\ d) = quotient\text{-}of\ q$
     $in\ normalize\ (a * b,\ c * d))$
  ⟨*proof*⟩

**lemma** *rat-inverse-code* [*code abstract*]:
  *quotient-of* $(inverse\ p) =$
    $(let\ (a,\ b) = quotient\text{-}of\ p$
     $in\ if\ a = 0\ then\ (0,\ 1)\ else\ (sgn\ a * b,\ |a|))$
⟨*proof*⟩

**lemma** *rat-divide-code* [*code abstract*]:
  *quotient-of* $(p\ /\ q) =$
    $(let\ (a,\ c) = quotient\text{-}of\ p;\ (b,\ d) = quotient\text{-}of\ q$
     $in\ normalize\ (a * d,\ c * b))$
  ⟨*proof*⟩

**lemma** *rat-abs-code* [*code abstract*]:
  *quotient-of* $|p| = (let\ (a,\ b) = quotient\text{-}of\ p\ in\ (|a|,\ b))$
  ⟨*proof*⟩

**lemma** *rat-sgn-code* [*code abstract*]: *quotient-of* $(sgn\ p) = (sgn\ (fst\ (quotient\text{-}of\ p)),\ 1)$
⟨*proof*⟩

**lemma** *rat-floor-code* [*code*]: $\lfloor p \rfloor = (let\ (a,\ b) = quotient\text{-}of\ p\ in\ a\ div\ b)$
  ⟨*proof*⟩

**instantiation** *rat* :: *equal*
**begin**

**definition** [*code*]: $HOL.equal\ a\ b \longleftrightarrow quotient\text{-}of\ a = quotient\text{-}of\ b$

**instance**
  ⟨*proof*⟩

**lemma** *rat-eq-refl* [*code nbe*]: $HOL.equal\ (r{::}rat)\ r \longleftrightarrow True$
  ⟨*proof*⟩

**end**

**lemma** *rat-less-eq-code* [*code*]:

$p \le q \longleftrightarrow (let\ (a,\ c) = quotient\text{-}of\ p;\ (b,\ d) = quotient\text{-}of\ q\ in\ a * d \le c * b)$
$\langle proof \rangle$

**lemma** *rat-less-code* [*code*]:
$p < q \longleftrightarrow (let\ (a,\ c) = quotient\text{-}of\ p;\ (b,\ d) = quotient\text{-}of\ q\ in\ a * d < c * b)$
$\langle proof \rangle$

**lemma** [*code*]: $of\text{-}rat\ p = (let\ (a,\ b) = quotient\text{-}of\ p\ in\ of\text{-}int\ a\ /\ of\text{-}int\ b)$
$\langle proof \rangle$

Quickcheck

**context**
**includes** *term-syntax*
**begin**

**definition**
$valterm\text{-}fract :: int \times (unit \Rightarrow Code\text{-}Evaluation.term) \Rightarrow$
$int \times (unit \Rightarrow Code\text{-}Evaluation.term) \Rightarrow$
$rat \times (unit \Rightarrow Code\text{-}Evaluation.term)$
**where** [*code-unfold*]: $valterm\text{-}fract\ k\ l = Code\text{-}Evaluation.valtermify\ Fract\ \{\cdot\}\ k$
$\{\cdot\}\ l$

**end**

**instantiation** *rat* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition**
$Quickcheck\text{-}Random.random\ i =$
$Quickcheck\text{-}Random.random\ i \circ\!\!\rightarrow (\lambda num.\ Random.range\ i \circ\!\!\rightarrow (\lambda denom.\ Pair$
$(let\ j = int\text{-}of\text{-}integer\ (integer\text{-}of\text{-}natural\ (denom + 1))$
$in\ valterm\text{-}fract\ num\ (j,\ \lambda u.\ Code\text{-}Evaluation.term\text{-}of\ j))))$

**instance** $\langle proof \rangle$

**end**

**end**

**instantiation** *rat* :: *exhaustive*
**begin**

**definition**
$exhaustive\text{-}rat\ f\ d =$
$Quickcheck\text{-}Exhaustive.exhaustive$

(λl. *Quickcheck-Exhaustive.exhaustive*
(λk. *f* (*Fract k* (*int-of-integer* (*integer-of-natural l*) + *1*))) *d*) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *rat* :: *full-exhaustive*
**begin**

**definition**
*full-exhaustive-rat f d* =
*Quickcheck-Exhaustive.full-exhaustive*
(λ(*l*, -). *Quickcheck-Exhaustive.full-exhaustive*
(λk. *f*
(*let j* = *int-of-integer* (*integer-of-natural l*) + *1*
*in valterm-fract k* (*j*, λ-. *Code-Evaluation.term-of j*))) *d*) *d*

**instance** ⟨*proof*⟩

**end**

**instance** *rat* :: *partial-term-of* ⟨*proof*⟩

**lemma** [*code*]:
*partial-term-of* (*ty* :: *rat itself*) (*Quickcheck-Narrowing.Narrowing-constructor 0* [*l*, *k*]) ≡
*Code-Evaluation.App*
(*Code-Evaluation.Const* (*STR* ''*Rat.Frct*'')
(*Typerep.Typerep* (*STR* ''*fun*'')
[*Typerep.Typerep* (*STR* ''*Product-Type.prod*'')
[*Typerep.Typerep* (*STR* ''*Int.int*'') [], *Typerep.Typerep* (*STR* ''*Int.int*'') []],
*Typerep.Typerep* (*STR* ''*Rat.rat*'') []]))
(*Code-Evaluation.App*
(*Code-Evaluation.App*
(*Code-Evaluation.Const* (*STR* ''*Product-Type.Pair*'')
(*Typerep.Typerep* (*STR* ''*fun*'')
[*Typerep.Typerep* (*STR* ''*Int.int*'') [],
*Typerep.Typerep* (*STR* ''*fun*'')
[*Typerep.Typerep* (*STR* ''*Int.int*'') [],
*Typerep.Typerep* (*STR* ''*Product-Type.prod*'')
[*Typerep.Typerep* (*STR* ''*Int.int*'') [], *Typerep.Typerep* (*STR* ''*Int.int*'') []]]]))
(*partial-term-of* (*TYPE*(*int*)) *l*)) (*partial-term-of* (*TYPE*(*int*)) *k*))
*partial-term-of* (*ty* :: *rat itself*) (*Quickcheck-Narrowing.Narrowing-variable p tt*) ≡
*Code-Evaluation.Free* (*STR* ''-'') (*Typerep.Typerep* (*STR* ''*Rat.rat*'') [])
⟨*proof*⟩

**instantiation** *rat* :: *narrowing*
**begin**

**definition**
  *narrowing* =
    *Quickcheck-Narrowing.apply*
      (*Quickcheck-Narrowing.apply*
        (*Quickcheck-Narrowing.cons* ($\lambda$*nom denom. Fract nom denom*)) *narrowing*)
*narrowing*

**instance** ⟨*proof*⟩

**end**

## 96.6 Setup for Nitpick

⟨*ML*⟩

**lemmas** [*nitpick-unfold*] =
  *inverse-rat-inst.inverse-rat*
  *one-rat-inst.one-rat ord-rat-inst.less-rat*
  *ord-rat-inst.less-eq-rat plus-rat-inst.plus-rat times-rat-inst.times-rat*
  *uminus-rat-inst.uminus-rat zero-rat-inst.zero-rat*

## 96.7 Float syntax

**syntax** *-Float* :: *float-const* ⇒ *'a*    (‹(‹*open-block notation*=‹*literal number*››-)›)

⟨*ML*⟩

Test:

**lemma** $123.456 = -111.111 + 200 + 30 + 4 + 5/10 + 6/100 + (7/1000::rat)$
  ⟨*proof*⟩

## 96.8 Hiding implementation details

**hide-const** (**open**) *normalize positive*

**lifting-update** *rat.lifting*
**lifting-forget** *rat.lifting*

**end**

# 97 Development of the Reals using Cauchy Sequences

**theory** *Real*
**imports** *Rat*

**begin**

This theory contains a formalization of the real numbers as equivalence classes of Cauchy sequences of rationals. See the AFP entry *Dedekind-Real* for an alternative construction using Dedekind cuts.

## 97.1 Preliminary lemmas

Useful in convergence arguments

**lemma** *inverse-of-nat-le*:
**fixes** $n::nat$ **shows** $[\![n \leq m;\ n\neq 0]\!] \Longrightarrow 1\ /\ \textit{of-nat}\ m \leq (1::'a::\textit{linordered-field})\ /\ \textit{of-nat}\ n$
⟨*proof*⟩

**lemma** *add-diff-add*: $(a + c) - (b + d) = (a - b) + (c - d)$
**for** $a\ b\ c\ d :: 'a::\textit{ab-group-add}$
⟨*proof*⟩

**lemma** *minus-diff-minus*: $-\ a - -\ b = -\ (a - b)$
**for** $a\ b :: 'a::\textit{ab-group-add}$
⟨*proof*⟩

**lemma** *mult-diff-mult*: $(x * y - a * b) = x * (y - b) + (x - a) * b$
**for** $x\ y\ a\ b :: 'a::\textit{ring}$
⟨*proof*⟩

**lemma** *inverse-diff-inverse*:
**fixes** $a\ b :: 'a::\textit{division-ring}$
**assumes** $a \neq 0$ **and** $b \neq 0$
**shows** $\textit{inverse}\ a - \textit{inverse}\ b = -\ (\textit{inverse}\ a * (a - b) * \textit{inverse}\ b)$
⟨*proof*⟩

**lemma** *obtain-pos-sum*:
**fixes** $r :: rat$ **assumes** $r$: $0 < r$
**obtains** $s\ t$ **where** $0 < s$ **and** $0 < t$ **and** $r = s + t$
⟨*proof*⟩

## 97.2 Sequences that converge to zero

**definition** *vanishes* :: $(nat \Rightarrow rat) \Rightarrow bool$
**where** $\textit{vanishes}\ X \longleftrightarrow (\forall r>0.\ \exists k.\ \forall n\geq k.\ |X\ n| < r)$

**lemma** *vanishesI*: $(\bigwedge r.\ 0 < r \Longrightarrow \exists k.\ \forall n\geq k.\ |X\ n| < r) \Longrightarrow \textit{vanishes}\ X$
⟨*proof*⟩

**lemma** *vanishesD*: $\textit{vanishes}\ X \Longrightarrow 0 < r \Longrightarrow \exists k.\ \forall n\geq k.\ |X\ n| < r$
⟨*proof*⟩

**lemma** *vanishes-const* [*simp*]: *vanishes* ($\lambda n.\ c$) $\longleftrightarrow$ $c = 0$
⟨*proof*⟩

**lemma** *vanishes-minus*: *vanishes* $X \Longrightarrow$ *vanishes* ($\lambda n.\ -\ X\ n$)
⟨*proof*⟩

**lemma** *vanishes-add*:
**assumes** $X$: *vanishes* $X$
**and** $Y$: *vanishes* $Y$
**shows** *vanishes* ($\lambda n.\ X\ n\ +\ Y\ n$)
⟨*proof*⟩

**lemma** *vanishes-diff*:
**assumes** *vanishes* $X$ *vanishes* $Y$
**shows** *vanishes* ($\lambda n.\ X\ n\ -\ Y\ n$)
⟨*proof*⟩

**lemma** *vanishes-mult-bounded*:
**assumes** $X$: $\exists\, a{>}0.\ \forall\, n.\ |X\ n| < a$
**assumes** $Y$: *vanishes* ($\lambda n.\ Y\ n$)
**shows** *vanishes* ($\lambda n.\ X\ n\ *\ Y\ n$)
⟨*proof*⟩

## 97.3 Cauchy sequences

**definition** *cauchy* :: (*nat* $\Rightarrow$ *rat*) $\Rightarrow$ *bool*
**where** *cauchy* $X \longleftrightarrow (\forall\, r{>}0.\ \exists\, k.\ \forall\, m{\geq}k.\ \forall\, n{\geq}k.\ |X\ m\ -\ X\ n| < r)$

**lemma** *cauchyI*: $(\bigwedge r.\ 0 < r \Longrightarrow \exists\, k.\ \forall\, m{\geq}k.\ \forall\, n{\geq}k.\ |X\ m\ -\ X\ n| < r) \Longrightarrow$ *cauchy* $X$
⟨*proof*⟩

**lemma** *cauchyD*: *cauchy* $X \Longrightarrow 0 < r \Longrightarrow \exists\, k.\ \forall\, m{\geq}k.\ \forall\, n{\geq}k.\ |X\ m\ -\ X\ n| < r$
⟨*proof*⟩

**lemma** *cauchy-const* [*simp*]: *cauchy* ($\lambda n.\ x$)
⟨*proof*⟩

**lemma** *cauchy-add* [*simp*]:
**assumes** $X$: *cauchy* $X$ **and** $Y$: *cauchy* $Y$
**shows** *cauchy* ($\lambda n.\ X\ n\ +\ Y\ n$)
⟨*proof*⟩

**lemma** *cauchy-minus* [*simp*]:
**assumes** $X$: *cauchy* $X$
**shows** *cauchy* ($\lambda n.\ -\ X\ n$)
⟨*proof*⟩

**lemma** *cauchy-diff* [*simp*]:

**assumes** *cauchy* $X$ *cauchy* $Y$
**shows** *cauchy* $(\lambda n.\ X\ n - Y\ n)$
⟨*proof*⟩

**lemma** *cauchy-imp-bounded*:
**assumes** *cauchy* $X$
**shows** $\exists\, b>0.\ \forall\, n.\ |X\ n| < b$
⟨*proof*⟩

**lemma** *cauchy-mult* [*simp*]:
**assumes** $X$: *cauchy* $X$ **and** $Y$: *cauchy* $Y$
**shows** *cauchy* $(\lambda n.\ X\ n * Y\ n)$
⟨*proof*⟩

**lemma** *cauchy-not-vanishes-cases*:
**assumes** $X$: *cauchy* $X$
**assumes** *nz*: $\neg$ *vanishes* $X$
**shows** $\exists\, b>0.\ \exists\, k.\ (\forall\, n\geq k.\ b < -\ X\ n) \vee (\forall\, n\geq k.\ b < X\ n)$
⟨*proof*⟩

**lemma** *cauchy-not-vanishes*:
**assumes** $X$: *cauchy* $X$
**and** *nz*: $\neg$ *vanishes* $X$
**shows** $\exists\, b>0.\ \exists\, k.\ \forall\, n\geq k.\ b < |X\ n|$
⟨*proof*⟩

**lemma** *cauchy-inverse* [*simp*]:
**assumes** $X$: *cauchy* $X$
**and** *nz*: $\neg$ *vanishes* $X$
**shows** *cauchy* $(\lambda n.$ *inverse* $(X\ n))$
⟨*proof*⟩

**lemma** *vanishes-diff-inverse*:
**assumes** $X$: *cauchy* $X$ $\neg$ *vanishes* $X$
**and** $Y$: *cauchy* $Y$ $\neg$ *vanishes* $Y$
**and** $XY$: *vanishes* $(\lambda n.\ X\ n - Y\ n)$
**shows** *vanishes* $(\lambda n.$ *inverse* $(X\ n) -$ *inverse* $(Y\ n))$
⟨*proof*⟩

## 97.4 Equivalence relation on Cauchy sequences

**definition** *realrel* :: $(nat \Rightarrow rat) \Rightarrow (nat \Rightarrow rat) \Rightarrow bool$
**where** *realrel* $= (\lambda X\ Y.$ *cauchy* $X \wedge$ *cauchy* $Y \wedge$ *vanishes* $(\lambda n.\ X\ n - Y\ n))$

**lemma** *realrelI* [*intro?*]: *cauchy* $X \Longrightarrow$ *cauchy* $Y \Longrightarrow$ *vanishes* $(\lambda n.\ X\ n - Y\ n) \Longrightarrow$ *realrel* $X\ Y$
⟨*proof*⟩

**lemma** *realrel-refl*: *cauchy* $X \Longrightarrow$ *realrel* $X\ X$

⟨*proof*⟩

**lemma** *symp-realrel*: *symp realrel*
⟨*proof*⟩

**lemma** *transp-realrel*: *transp realrel*
⟨*proof*⟩

**lemma** *part-equivp-realrel*: *part-equivp realrel*
⟨*proof*⟩

## 97.5 The field of real numbers

**quotient-type** *real* = *nat* ⇒ *rat* / *partial*: *realrel*
**morphisms** *rep-real Real*
⟨*proof*⟩

**lemma** *cr-real-eq*: *pcr-real* = ($\lambda x\ y.$ *cauchy x* ∧ *Real x* = *y*)
⟨*proof*⟩

**lemma** *Real-induct* [*induct type*: *real*]:
**assumes** $\bigwedge X.$ *cauchy X* ⟹ *P* (*Real X*)
**shows** *P x*
⟨*proof*⟩

**lemma** *eq-Real*: *cauchy X* ⟹ *cauchy Y* ⟹ *Real X* = *Real Y* ⟷ *vanishes* ($\lambda n.$ *X n* − *Y n*)
⟨*proof*⟩

**lemma** *Domainp-pcr-real* [*transfer-domain-rule*]: *Domainp pcr-real* = *cauchy*
⟨*proof*⟩

**instantiation** *real* :: *field*
**begin**

**lift-definition** *zero-real* :: *real* **is** $\lambda n.$ *0*
⟨*proof*⟩

**lift-definition** *one-real* :: *real* **is** $\lambda n.$ *1*
⟨*proof*⟩

**lift-definition** *plus-real* :: *real* ⇒ *real* ⇒ *real* **is** $\lambda X\ Y\ n.\ X\ n + Y\ n$
⟨*proof*⟩

**lift-definition** *uminus-real* :: *real* ⇒ *real* **is** $\lambda X\ n.\ -\ X\ n$
⟨*proof*⟩

**lift-definition** *times-real* :: *real* ⇒ *real* ⇒ *real* **is** $\lambda X\ Y\ n.\ X\ n * Y\ n$
⟨*proof*⟩

**lift-definition** *inverse-real* :: *real* ⇒ *real*
**is** $\lambda X.$ *if vanishes X then* $(\lambda n.\ 0)$ *else* $(\lambda n.$ *inverse* $(X\ n))$
⟨*proof*⟩

**definition** $x - y = x + - y$ **for** $x\ y$ :: *real*

**definition** $x$ *div* $y = x *$ *inverse* $y$ **for** $x\ y$ :: *real*

**lemma** *add-Real*: *cauchy* $X \Longrightarrow$ *cauchy* $Y \Longrightarrow$ *Real* $X +$ *Real* $Y =$ *Real* $(\lambda n.\ X\ n + Y\ n)$
⟨*proof*⟩

**lemma** *minus-Real*: *cauchy* $X \Longrightarrow -$ *Real* $X =$ *Real* $(\lambda n.\ - X\ n)$
⟨*proof*⟩

**lemma** *diff-Real*: *cauchy* $X \Longrightarrow$ *cauchy* $Y \Longrightarrow$ *Real* $X -$ *Real* $Y =$ *Real* $(\lambda n.\ X\ n - Y\ n)$
⟨*proof*⟩

**lemma** *mult-Real*: *cauchy* $X \Longrightarrow$ *cauchy* $Y \Longrightarrow$ *Real* $X *$ *Real* $Y =$ *Real* $(\lambda n.\ X\ n * Y\ n)$
⟨*proof*⟩

**lemma** *inverse-Real*:
*cauchy* $X \Longrightarrow$ *inverse* (*Real* $X$) = (*if vanishes X then 0 else Real* $(\lambda n.$ *inverse* $(X\ n)))$
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

## 97.6 Positive reals

**lift-definition** *positive* :: *real* ⇒ *bool*
**is** $\lambda X.\ \exists r>0.\ \exists k.\ \forall n\geq k.\ r < X\ n$
⟨*proof*⟩

**lemma** *positive-Real*: *cauchy* $X \Longrightarrow$ *positive* (*Real* $X$) $\longleftrightarrow (\exists r>0.\ \exists k.\ \forall n\geq k.\ r < X\ n)$
⟨*proof*⟩

**lemma** *positive-zero*: $\neg$ *positive 0*
⟨*proof*⟩

**lemma** *positive-add*:
**assumes** *positive x positive y* **shows** *positive* $(x + y)$

⟨*proof*⟩

**lemma** *positive-mult*:
**assumes** *positive* $x$ *positive* $y$ **shows** *positive* $(x * y)$
⟨*proof*⟩

**lemma** *positive-minus*: $\neg$ *positive* $x \Longrightarrow x \neq 0 \Longrightarrow$ *positive* $(- x)$
⟨*proof*⟩

**instantiation** *real* :: *linordered-field*
**begin**

**definition** $x < y \longleftrightarrow$ *positive* $(y - x)$

**definition** $x \leq y \longleftrightarrow x < y \vee x = y$ **for** $x$ $y$ :: *real*

**definition** $|a| = (\mathit{if}\ a < 0\ \mathit{then} - a\ \mathit{else}\ a)$ **for** $a$ :: *real*

**definition** $\mathit{sgn}\ a = (\mathit{if}\ a = 0\ \mathit{then}\ 0\ \mathit{else}\ \mathit{if}\ 0 < a\ \mathit{then}\ 1\ \mathit{else} - 1)$ **for** $a$ :: *real*

**instance**
⟨*proof*⟩

**end**

**instantiation** *real* :: *distrib-lattice*
**begin**

**definition** $(\mathit{inf} :: \mathit{real} \Rightarrow \mathit{real} \Rightarrow \mathit{real}) = \mathit{min}$

**definition** $(\mathit{sup} :: \mathit{real} \Rightarrow \mathit{real} \Rightarrow \mathit{real}) = \mathit{max}$

**instance**
⟨*proof*⟩

**end**

**lemma** *of-nat-Real*: *of-nat* $x$ = *Real* $(\lambda n.\ \mathit{of\text{-}nat}\ x)$
⟨*proof*⟩

**lemma** *of-int-Real*: *of-int* $x$ = *Real* $(\lambda n.\ \mathit{of\text{-}int}\ x)$
⟨*proof*⟩

**lemma** *of-rat-Real*: *of-rat* $x$ = *Real* $(\lambda n.\ x)$
⟨*proof*⟩

**instance** *real* :: *archimedean-field*
⟨*proof*⟩

**instantiation** *real* :: *floor-ceiling*
**begin**

**definition** [*code del*]: $\lfloor x::real \rfloor = (THE\ z.\ of\text{-}int\ z \leq x \wedge x < of\text{-}int\ (z + 1))$

**instance**
⟨*proof*⟩

**end**

## 97.7 Completeness

**lemma** *not-positive-Real*:
**assumes** *cauchy X* **shows** $\neg$ *positive* (*Real X*) $\longleftrightarrow (\forall r>0.\ \exists k.\ \forall n\geq k.\ X\ n \leq r)$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *le-Real*:
**assumes** *cauchy X cauchy Y*
**shows** $Real\ X \leq Real\ Y = (\forall r>0.\ \exists k.\ \forall n\geq k.\ X\ n \leq Y\ n + r)$
⟨*proof*⟩

**lemma** *le-RealI*:
**assumes** *Y*: *cauchy Y*
**shows** $\forall n.\ x \leq of\text{-}rat\ (Y\ n) \Longrightarrow x \leq Real\ Y$
⟨*proof*⟩

**lemma** *Real-leI*:
**assumes** *X*: *cauchy X*
**assumes** *le*: $\forall n.\ of\text{-}rat\ (X\ n) \leq y$
**shows** $Real\ X \leq y$
⟨*proof*⟩

**lemma** *less-RealD*:
**assumes** *cauchy Y*
**shows** $x < Real\ Y \Longrightarrow \exists n.\ x < of\text{-}rat\ (Y\ n)$
⟨*proof*⟩

**lemma** *of-nat-less-two-power* [*simp*]: $of\text{-}nat\ n < (2::'a::linordered\text{-}idom)\ \hat{}\ n$
⟨*proof*⟩

**lemma** *complete-real*:
**fixes** *S* :: *real set*
**assumes** $\exists x.\ x \in S$ **and** $\exists z.\ \forall x\in S.\ x \leq z$
**shows** $\exists y.\ (\forall x\in S.\ x \leq y) \wedge (\forall z.\ (\forall x\in S.\ x \leq z) \longrightarrow y \leq z)$
⟨*proof*⟩

**instantiation** *real* :: *linear-continuum*
**begin**

## 97.8 Supremum of a set of reals

**definition** *Sup* $X = (LEAST\ z{::}real.\ \forall x{\in}X.\ x \le z)$
**definition** *Inf* $X = -\ Sup\ (uminus\ `\ X)$ **for** $X :: real\ set$

**instance**
⟨*proof*⟩

**end**

## 97.9 Hiding implementation details

**hide-const** (**open**) *vanishes cauchy positive Real*

**declare** *Real-induct* [*induct del*]
**declare** *Abs-real-induct* [*induct del*]
**declare** *Abs-real-cases* [*cases del*]

**lifting-update** *real.lifting*
**lifting-forget** *real.lifting*

## 97.10 Embedding numbers into the Reals

**abbreviation** *real-of-nat* :: $nat \Rightarrow real$
  **where** *real-of-nat* $\equiv$ *of-nat*

**abbreviation** *real* :: $nat \Rightarrow real$
  **where** *real* $\equiv$ *of-nat*

**abbreviation** *real-of-int* :: $int \Rightarrow real$
  **where** *real-of-int* $\equiv$ *of-int*

**abbreviation** *real-of-rat* :: $rat \Rightarrow real$
  **where** *real-of-rat* $\equiv$ *of-rat*

**declare** [[*coercion-enabled*]]

**declare** [[*coercion of-nat* :: $nat \Rightarrow int$]]
**declare** [[*coercion of-nat* :: $nat \Rightarrow real$]]
**declare** [[*coercion of-int* :: $int \Rightarrow real$]]

**declare** [[*coercion-map map*]]
**declare** [[*coercion-map* $\lambda f\ g\ h\ x.\ g\ (h\ (f\ x))$]]
**declare** [[*coercion-map* $\lambda f\ g\ (x,y).\ (f\ x,\ g\ y)$]]

**declare** *of-int-eq-0-iff* [*algebra*, *presburger*]
**declare** *of-int-eq-1-iff* [*algebra*, *presburger*]
**declare** *of-int-eq-iff* [*algebra*, *presburger*]

**declare** *of-int-less-0-iff* [*algebra*, *presburger*]
**declare** *of-int-less-1-iff* [*algebra*, *presburger*]
**declare** *of-int-less-iff* [*algebra*, *presburger*]
**declare** *of-int-le-0-iff* [*algebra*, *presburger*]
**declare** *of-int-le-1-iff* [*algebra*, *presburger*]
**declare** *of-int-le-iff* [*algebra*, *presburger*]
**declare** *of-int-0-less-iff* [*algebra*, *presburger*]
**declare** *of-int-0-le-iff* [*algebra*, *presburger*]
**declare** *of-int-1-less-iff* [*algebra*, *presburger*]
**declare** *of-int-1-le-iff* [*algebra*, *presburger*]

**lemma** *int-less-real-le*: $n < m \longleftrightarrow \mathit{real\text{-}of\text{-}int}\ n + 1 \leq \mathit{real\text{-}of\text{-}int}\ m$
⟨*proof*⟩

**lemma** *int-le-real-less*: $n \leq m \longleftrightarrow \mathit{real\text{-}of\text{-}int}\ n < \mathit{real\text{-}of\text{-}int}\ m + 1$
⟨*proof*⟩

**lemma** (**in** *field-char-0*) *of-int-div-aux*:
$(\mathit{of\text{-}int}\ x)\ /\ (\mathit{of\text{-}int}\ d) =$
$\mathit{of\text{-}int}\ (x\ \mathit{div}\ d) + (\mathit{of\text{-}int}\ (x\ \mathit{mod}\ d))\ /\ (\mathit{of\text{-}int}\ d)$
⟨*proof*⟩

**lemma** *real-of-int-div*:
$d\ \mathit{dvd}\ n \Longrightarrow \mathit{real\text{-}of\text{-}int}\ (n\ \mathit{div}\ d) = \mathit{real\text{-}of\text{-}int}\ n\ /\ \mathit{real\text{-}of\text{-}int}\ d$ **for** $d\ n :: \mathit{int}$
⟨*proof*⟩

**lemma** *real-of-int-div2*: $0 \leq \mathit{real\text{-}of\text{-}int}\ n\ /\ \mathit{real\text{-}of\text{-}int}\ x - \mathit{real\text{-}of\text{-}int}\ (n\ \mathit{div}\ x)$
⟨*proof*⟩

**lemma** *real-of-int-div3*: $\mathit{real\text{-}of\text{-}int}\ n\ /\ \mathit{real\text{-}of\text{-}int}\ x - \mathit{real\text{-}of\text{-}int}\ (n\ \mathit{div}\ x) \leq 1$
⟨*proof*⟩

**lemma** *real-of-int-div4*: $\mathit{real\text{-}of\text{-}int}\ (n\ \mathit{div}\ x) \leq \mathit{real\text{-}of\text{-}int}\ n\ /\ \mathit{real\text{-}of\text{-}int}\ x$
⟨*proof*⟩

## 97.11 Embedding the Naturals into the Reals

**lemma** (**in** *field-char-0*) *of-nat-of-nat-div-aux*:
$\mathit{of\text{-}nat}\ x\ /\ \mathit{of\text{-}nat}\ d = \mathit{of\text{-}nat}\ (x\ \mathit{div}\ d) + \mathit{of\text{-}nat}\ (x\ \mathit{mod}\ d)\ /\ \mathit{of\text{-}nat}\ d$
⟨*proof*⟩

**lemma**(**in** *field-char-0*) *of-nat-of-nat-div*: $d\ \mathit{dvd}\ n \Longrightarrow \mathit{of\text{-}nat}(n\ \mathit{div}\ d) = \mathit{of\text{-}nat}\ n\ /\ \mathit{of\text{-}nat}\ d$
⟨*proof*⟩

**lemma** (**in** *linordered-field*) *of-nat-div-le-of-nat*: $\mathit{of\text{-}nat}\ (n\ \mathit{div}\ x) \leq \mathit{of\text{-}nat}\ n\ /\ \mathit{of\text{-}nat}\ x$
⟨*proof*⟩

**lemma** *real-of-card*: $real\ (card\ A) = sum\ (\lambda x.\ 1)\ A$
⟨*proof*⟩

**lemma** *nat-less-real-le*: $n < m \longleftrightarrow real\ n + 1 \leq real\ m$
⟨*proof*⟩

**lemma** *nat-le-real-less*: $n \leq m \longleftrightarrow real\ n < real\ m + 1$
⟨*proof*⟩

**lemma** *real-of-nat-div*: $d\ dvd\ n \Longrightarrow real(n\ div\ d) = real\ n\ /\ real\ d$
⟨*proof*⟩

**lemma** *real-binomial-eq-mult-binomial-Suc*:
**assumes** $k \leq n$
**shows** $real(n\ choose\ k) = (n + 1 - k)\ /\ (n + 1) * (Suc\ n\ choose\ k)$
⟨*proof*⟩

## 97.12 The Archimedean Property of the Reals

Not actually the reals any more!

**lemma** *real-arch-inverse*:
**fixes** $e::'a::archimedean\text{-}field$
**shows** $0 < e \longleftrightarrow (\exists n::nat.\ n \neq 0 \wedge 0 < inverse\ (real\ n) \wedge inverse\ (of\text{-}nat\ n) < e)$
⟨*proof*⟩

**lemma** *reals-Archimedean3*:
**fixes** $x::'a::archimedean\text{-}field$
**shows** $0 < x \Longrightarrow \forall y.\ \exists n.\ y < of\text{-}nat\ n * x$
⟨*proof*⟩

**lemma** *real-archimedian-rdiv-eq-0*:
**fixes** $x::'a::archimedean\text{-}field$
**assumes** $x \geq 0$ **and** $\bigwedge m::nat.\ m > 0 \Longrightarrow of\text{-}nat\ m * x \leq c$
**shows** $x = 0$
⟨*proof*⟩

**lemma** *inverse-Suc*: $inverse\ (of\text{-}nat\ (Suc\ n)) > (0::'a::archimedean\text{-}field)$
⟨*proof*⟩

**lemma** *Archimedean-eventually-inverse*:
**fixes** $\varepsilon::'a::archimedean\text{-}field$ **shows** $(\forall_F\ n\ in\ sequentially.\ inverse\ (of\text{-}nat\ (Suc\ n)) < \varepsilon) \longleftrightarrow 0 < \varepsilon$
(**is** *?lhs=?rhs*)
⟨*proof*⟩

On the relationship between two different ways of converting to 0

**lemma** *Inter-eq-Inter-inverse-Suc*:
**assumes** $\bigwedge r'\ r.\ r' < r \Longrightarrow A\ r' \subseteq A\ r$

**shows** $\bigcap$ (*A* ' {*0*<..}) = ($\bigcap$ *n*. *A*(*inverse*(*Suc n*)))
⟨*proof*⟩

## 97.13 Rationals

**lemma** *Rats-abs-iff* [*simp*]:
$|(x::real)| \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *Rats-eq-int-div-int*: $\mathbb{Q}$ = {*real-of-int i* / *real-of-int j* | *i j*. *j* ≠ *0*} (**is** - = *?S*)
⟨*proof*⟩

**lemma** *Rats-eq-int-div-nat*: $\mathbb{Q}$ = { *real-of-int i* / *real n* | *i n*. *n* ≠ *0*}
⟨*proof*⟩

**lemma** *Rats-abs-nat-div-natE*:
**assumes** $x \in \mathbb{Q}$
**obtains** *m n* :: *nat* **where** *n* ≠ *0* **and** |*x*| = *real m* / *real n* **and** *coprime m n*
⟨*proof*⟩

## 97.14 Density of the Rational Reals in the Reals

This density proof is due to Stefan Richter and was ported by TN. The original source is *Real Analysis* by H.L. Royden. It employs the Archimedean property of the reals.

**lemma** *Rats-dense-in-real*:
**fixes** *x* :: *real*
**assumes** $x < y$
**shows** $\exists r \in \mathbb{Q}.\ x < r \land r < y$
⟨*proof*⟩

**lemma** *of-rat-dense*:
**fixes** *x y* :: *real*
**assumes** $x < y$
**shows** ∃ *q* :: *rat*. *x* < *of-rat q* ∧ *of-rat q* < *y*
⟨*proof*⟩

## 97.15 Numerals and Arithmetic

⟨*ML*⟩

## 97.16 Simprules combining $x + y$ and $0$

**lemma** *real-add-minus-iff* [*simp*]: $x + - a = 0 \longleftrightarrow x = a$
**for** *x a* :: *real*
⟨*proof*⟩

**lemma** *real-add-less-0-iff*: $x + y < 0 \longleftrightarrow y < - x$

**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *real-0-less-add-iff*: $0 < x + y \longleftrightarrow - x < y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *real-add-le-0-iff*: $x + y \leq 0 \longleftrightarrow y \leq - x$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *real-0-le-add-iff*: $0 \leq x + y \longleftrightarrow - x \leq y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *mult-ge1-I*: $[\![x\geq 1;\ y\geq 1]\!] \Longrightarrow x*y \geq (1::real)$
⟨*proof*⟩

## 97.17 Lemmas about powers

**lemma** *two-realpow-ge-one*: $(1::real) \leq 2\ \hat{}\ n$
⟨*proof*⟩

**declare** *sum-squares-eq-zero-iff* [*simp*] *sum-power2-eq-zero-iff* [*simp*]

**lemma** *real-minus-mult-self-le* [*simp*]: $- (u * u) \leq x * x$
**for** $u\ x$ :: *real*
⟨*proof*⟩

**lemma** *realpow-square-minus-le* [*simp*]: $- u^2 \leq x^2$
**for** $u\ x$ :: *real*
⟨*proof*⟩

## 97.18 Density of the Reals

**lemma** *field-lbound-gt-zero*: $0 < d1 \Longrightarrow 0 < d2 \Longrightarrow \exists e.\ 0 < e \wedge e < d1 \wedge e < d2$
**for** $d1\ d2$ :: $'a$::*linordered-field*
⟨*proof*⟩

**lemma** *field-less-half-sum*: $x < y \Longrightarrow x < (x + y) / 2$
**for** $x\ y$ :: $'a$::*linordered-field*
⟨*proof*⟩

**lemma** *field-sum-of-halves*: $x / 2 + x / 2 = x$
**for** $x$ :: $'a$::*linordered-field*
⟨*proof*⟩

## 97.19 Archimedean properties and useful consequences

Bernoulli's inequality

**proposition** *Bernoulli-inequality*:
**fixes** $x :: 'a :: linordered\text{-}field$
**assumes** $-1 \leq x$
**shows** $1 + of\text{-}nat\ n * x \leq (1 + x)\ \hat{}\ n$
$\langle proof \rangle$

**corollary** *Bernoulli-inequality-even*:
**fixes** $x :: 'a :: linordered\text{-}field$
**assumes** *even n*
**shows** $1 + of\text{-}nat\ n * x \leq (1 + x)\ \hat{}\ n$
$\langle proof \rangle$

**corollary** *real-arch-pow*:
**fixes** $x :: real$
**assumes** $x$: $1 < x$
**shows** $\exists n.\ y < x\hat{}n$
$\langle proof \rangle$

**corollary** *real-arch-pow-inv*:
**fixes** $x\ y :: real$
**assumes** $y$: $y > 0$
**and** $x1$: $x < 1$
**shows** $\exists n.\ x\hat{}n < y$
$\langle proof \rangle$

**lemma** *forall-pos-mono*:
$(\bigwedge d\ e::real.\ d < e \Longrightarrow P\ d \Longrightarrow P\ e) \Longrightarrow$
$(\bigwedge n::nat.\ n \neq 0 \Longrightarrow P\ (inverse\ (real\ n))) \Longrightarrow (\bigwedge e.\ 0 < e \Longrightarrow P\ e)$
$\langle proof \rangle$

**lemma** *forall-pos-mono-1*:
$(\bigwedge d\ e::real.\ d < e \Longrightarrow P\ d \Longrightarrow P\ e) \Longrightarrow$
$(\bigwedge n.\ P\ (inverse\ (real\ (Suc\ n)))) \Longrightarrow 0 < e \Longrightarrow P\ e$
$\langle proof \rangle$

**lemma** *Archimedean-eventually-pow*:
**fixes** $x::real$
**assumes** $1 < x$
**shows** $\forall_F\ n\ in\ sequentially.\ b < x\ \hat{}\ n$
$\langle proof \rangle$

**lemma** *Archimedean-eventually-pow-inverse*:
**fixes** $x::real$
**assumes** $|x| < 1\ \varepsilon > 0$
**shows** $\forall_F\ n\ in\ sequentially.\ |x\hat{}n| < \varepsilon$
$\langle proof \rangle$

## 97.20 Floor and Ceiling Functions from the Reals to the Integers

**lemma** *real-of-nat-less-numeral-iff* [*simp*]: *real n* $<$ *numeral w* $\longleftrightarrow$ *n* $<$ *numeral w*
**for** *n* :: *nat*
⟨*proof*⟩

**lemma** *numeral-less-real-of-nat-iff* [*simp*]: *numeral w* $<$ *real n* $\longleftrightarrow$ *numeral w* $<$ *n*
**for** *n* :: *nat*
⟨*proof*⟩

**lemma** *numeral-le-real-of-nat-iff* [*simp*]: *numeral n* $\leq$ *real m* $\longleftrightarrow$ *numeral n* $\leq$ *m*
**for** *m* :: *nat*
⟨*proof*⟩

**lemma** *of-int-floor-cancel* [*simp*]: *of-int* $\lfloor x \rfloor = x \longleftrightarrow (\exists\, n{::}int.\ x =$ *of-int n*$)$
⟨*proof*⟩

**lemma** *of-int-floor* [*simp*]: $a \in \mathbb{Z} \Longrightarrow$ *of-int* (*floor a*) $= a$
⟨*proof*⟩

**lemma** *floor-frac* [*simp*]: $\lfloor$*frac r*$\rfloor = 0$
⟨*proof*⟩

**lemma** *frac-1* [*simp*]: *frac 1* $= 0$
⟨*proof*⟩

**lemma** *frac-in-Rats-iff* [*simp*]:
**fixes** *r*::'*a*::{*floor-ceiling*,*field-char-0*}
**shows** *frac r* $\in \mathbb{Q} \longleftrightarrow r \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *floor-eq*: *real-of-int n* $< x \Longrightarrow x <$ *real-of-int n* $+ 1 \Longrightarrow \lfloor x \rfloor = n$
⟨*proof*⟩

**lemma** *floor-eq2*: *real-of-int n* $\leq x \Longrightarrow x <$ *real-of-int n* $+ 1 \Longrightarrow \lfloor x \rfloor = n$
⟨*proof*⟩

**lemma** *floor-eq3*: *real n* $< x \Longrightarrow x <$ *real* (*Suc n*) $\Longrightarrow$ *nat* $\lfloor x \rfloor = n$
⟨*proof*⟩

**lemma** *floor-eq4*: *real n* $\leq x \Longrightarrow x <$ *real* (*Suc n*) $\Longrightarrow$ *nat* $\lfloor x \rfloor = n$
⟨*proof*⟩

**lemma** *real-of-int-floor-ge-diff-one* [*simp*]: $r - 1 \leq$ *real-of-int* $\lfloor r \rfloor$
⟨*proof*⟩

**lemma** *real-of-int-floor-gt-diff-one* [*simp*]: $r - 1 <$ *real-of-int* $\lfloor r \rfloor$

⟨*proof*⟩

**lemma** *real-of-int-floor-add-one-ge* [*simp*]: $r \leq$ *real-of-int* $\lfloor r \rfloor + 1$
⟨*proof*⟩

**lemma** *real-of-int-floor-add-one-gt* [*simp*]: $r <$ *real-of-int* $\lfloor r \rfloor + 1$
⟨*proof*⟩

**lemma** *floor-divide-real-eq-div*:
**assumes** $0 \leq b$
**shows** $\lfloor a$ / *real-of-int* $b \rfloor = \lfloor a \rfloor$ *div* $b$
⟨*proof*⟩

**lemma** *floor-one-divide-eq-div-numeral* [*simp*]:
$\lfloor 1$ / *numeral* $b$::*real*$\rfloor = 1$ *div numeral* $b$
⟨*proof*⟩

**lemma** *floor-minus-one-divide-eq-div-numeral* [*simp*]:
$\lfloor -$ ($1$ / *numeral* $b$)::*real*$\rfloor = - 1$ *div numeral* $b$
⟨*proof*⟩

**lemma** *floor-divide-eq-div-numeral* [*simp*]:
$\lfloor$*numeral* $a$ / *numeral* $b$::*real*$\rfloor$ = *numeral* $a$ *div numeral* $b$
⟨*proof*⟩

**lemma** *floor-minus-divide-eq-div-numeral* [*simp*]:
$\lfloor -$ (*numeral* $a$ / *numeral* $b$)::*real*$\rfloor = -$ *numeral* $a$ *div numeral* $b$
⟨*proof*⟩

**lemma** *of-int-ceiling-cancel* [*simp*]: *of-int* $\lceil x \rceil = x \longleftrightarrow (\exists\, n$::*int*. $x =$ *of-int* $n$)
⟨*proof*⟩

**lemma** *of-int-ceiling* [*simp*]: $a \in \mathbb{Z} \Longrightarrow$ *of-int* (*ceiling* $a$) $= a$
⟨*proof*⟩

**lemma** *ceiling-eq*: *of-int* $n < x \Longrightarrow x \leq$ *of-int* $n + 1 \Longrightarrow \lceil x \rceil = n + 1$
⟨*proof*⟩

**lemma** *of-int-ceiling-diff-one-le* [*simp*]: *of-int* $\lceil r \rceil - 1 \leq r$
⟨*proof*⟩

**lemma** *of-int-ceiling-le-add-one* [*simp*]: *of-int* $\lceil r \rceil \leq r + 1$
⟨*proof*⟩

**lemma** *ceiling-le*: $x \leq$ *of-int* $a \Longrightarrow \lceil x \rceil \leq a$
⟨*proof*⟩

**lemma** *ceiling-divide-eq-div*: $\lceil$*of-int* $a$ / *of-int* $b\rceil = - (- a$ *div* $b$)
⟨*proof*⟩

**lemma** *ceiling-divide-eq-div-numeral* [*simp*]:
$\lceil$*numeral a* / *numeral b* :: *real*$\rceil$ = − (− *numeral a div numeral b*)
⟨*proof*⟩

**lemma** *ceiling-minus-divide-eq-div-numeral* [*simp*]:
$\lceil$− (*numeral a* / *numeral b* :: *real*)$\rceil$ = − (*numeral a div numeral b*)
⟨*proof*⟩

The following lemmas are remnants of the erstwhile functions natfloor and natceiling.

**lemma** *nat-floor-neg*: $x \le 0 \Longrightarrow nat\ \lfloor x\rfloor = 0$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *le-nat-floor*: $real\ x \le a \Longrightarrow x \le nat\ \lfloor a\rfloor$
⟨*proof*⟩

**lemma** *le-mult-nat-floor*: $nat\ \lfloor a\rfloor * nat\ \lfloor b\rfloor \le nat\ \lfloor a * b\rfloor$
⟨*proof*⟩

**lemma** *nat-ceiling-le-eq* [*simp*]: $nat\ \lceil x\rceil \le a \longleftrightarrow x \le real\ a$
⟨*proof*⟩

**lemma** *real-nat-ceiling-ge*: $x \le real\ (nat\ \lceil x\rceil)$
⟨*proof*⟩

**lemma** *Rats-no-top-le*: $\exists\, q \in \mathbb{Q}.\ x \le q$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *Rats-no-bot-less*: $\exists\, q \in \mathbb{Q}.\ q < x$ **for** $x$ :: *real*
⟨*proof*⟩

**lemma** *floor-ceiling-diff-le*: $0 \le r \Longrightarrow nat\lfloor real\ k - r\rfloor \le k - nat\lceil r\rceil$
⟨*proof*⟩

**lemma** *floor-ceiling-diff-le'*: $nat\lfloor r - real\ k\rfloor \le nat\lceil r\rceil - k$
⟨*proof*⟩

**lemma** *ceiling-floor-diff-ge*: $nat\lceil r - real\ k\rceil \ge nat\lfloor r\rfloor - k$
⟨*proof*⟩

**lemma** *ceiling-floor-diff-ge'*: $r \le k \Longrightarrow nat\lceil r - real\ k\rceil \le k - nat\lfloor r\rfloor$
⟨*proof*⟩

## 97.21 Exponentiation with floor

**lemma** *floor-power*:

**assumes** $x = \textit{of-int}\ \lfloor x \rfloor$
**shows** $\lfloor x \mathbin{\hat{}} n \rfloor = \lfloor x \rfloor \mathbin{\hat{}} n$
⟨*proof*⟩

**lemma** *floor-numeral-power* [*simp*]: $\lfloor \textit{numeral}\ x \mathbin{\hat{}} n \rfloor = \textit{numeral}\ x \mathbin{\hat{}} n$
⟨*proof*⟩

**lemma** *ceiling-numeral-power* [*simp*]: $\lceil \textit{numeral}\ x \mathbin{\hat{}} n \rceil = \textit{numeral}\ x \mathbin{\hat{}} n$
⟨*proof*⟩

## 97.22 Implementation of rational real numbers

Formal constructor

**definition** *Ratreal* :: *rat* ⇒ *real*
**where** [*code-abbrev*, *simp*]: *Ratreal* = *real-of-rat*

**code-datatype** *Ratreal*

Quasi-Numerals

**lemma** [*code-abbrev*]:
*real-of-rat* (*numeral k*) = *numeral k*
*real-of-rat* (− *numeral k*) = − *numeral k*
*real-of-rat* (*rat-of-int a*) = *real-of-int a*
⟨*proof*⟩

**lemma** [*code-post*]:
*real-of-rat 0* = *0*
*real-of-rat 1* = *1*
*real-of-rat* (− *1*) = − *1*
*real-of-rat* (*1* / *numeral k*) = *1* / *numeral k*
*real-of-rat* (*numeral k* / *numeral l*) = *numeral k* / *numeral l*
*real-of-rat* (− (*1* / *numeral k*)) = − (*1* / *numeral k*)
*real-of-rat* (− (*numeral k* / *numeral l*)) = − (*numeral k* / *numeral l*)
⟨*proof*⟩

Operations

**lemma** *zero-real-code* [*code*]: *0* = *Ratreal 0*
⟨*proof*⟩

**lemma** *one-real-code* [*code*]: *1* = *Ratreal 1*
⟨*proof*⟩

**instantiation** *real* :: *equal*
**begin**

**definition** *HOL.equal x y* ⟷ *x* − *y* = *0* **for** *x* :: *real*

**instance** ⟨*proof*⟩

**lemma** *real-equal-code* [*code*]: *HOL.equal* (*Ratreal* $x$) (*Ratreal* $y$) $\longleftrightarrow$ *HOL.equal* $x$ $y$
⟨*proof*⟩

**lemma** [*code nbe*]: *HOL.equal* $x$ $x$ $\longleftrightarrow$ *True*
**for** $x$ :: *real*
⟨*proof*⟩

**end**

**lemma** *real-less-eq-code* [*code*]: *Ratreal* $x \leq$ *Ratreal* $y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *real-less-code* [*code*]: *Ratreal* $x <$ *Ratreal* $y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *real-plus-code* [*code*]: *Ratreal* $x +$ *Ratreal* $y =$ *Ratreal* $(x + y)$
⟨*proof*⟩

**lemma** *real-times-code* [*code*]: *Ratreal* $x *$ *Ratreal* $y =$ *Ratreal* $(x * y)$
⟨*proof*⟩

**lemma** *real-uminus-code* [*code*]: $-$ *Ratreal* $x =$ *Ratreal* $(- x)$
⟨*proof*⟩

**lemma** *real-minus-code* [*code*]: *Ratreal* $x -$ *Ratreal* $y =$ *Ratreal* $(x - y)$
⟨*proof*⟩

**lemma** *real-inverse-code* [*code*]: *inverse* (*Ratreal* $x$) $=$ *Ratreal* (*inverse* $x$)
⟨*proof*⟩

**lemma** *real-divide-code* [*code*]: *Ratreal* $x$ / *Ratreal* $y =$ *Ratreal* $(x / y)$
⟨*proof*⟩

**lemma** *real-floor-code* [*code*]: $\lfloor$*Ratreal* $x\rfloor = \lfloor x\rfloor$
⟨*proof*⟩

Quickcheck

**context**
**includes** *term-syntax*
**begin**

**definition**
*valterm-ratreal* :: *rat* $\times$ (*unit* $\Rightarrow$ *Code-Evaluation.term*) $\Rightarrow$ *real* $\times$ (*unit* $\Rightarrow$ *Code-Evaluation.term*)
**where** [*code-unfold*]: *valterm-ratreal* $k$ = *Code-Evaluation.valtermify* *Ratreal* $\{\cdot\}$ $k$

**end**

**instantiation** *real* :: *random*
**begin**

**context**
**includes** *state-combinator-syntax*
**begin**

**definition**
*Quickcheck-Random.random i* = *Quickcheck-Random.random i* ∘→ (λ*r*. *Pair* (*valterm-ratreal r*))

**instance** ⟨*proof*⟩

**end**

**end**

**instantiation** *real* :: *exhaustive*
**begin**

**definition**
*exhaustive-real f d* = *Quickcheck-Exhaustive.exhaustive* (λ*r*. *f* (*Ratreal r*)) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *real* :: *full-exhaustive*
**begin**

**definition**
*full-exhaustive-real f d* = *Quickcheck-Exhaustive.full-exhaustive* (λ*r*. *f* (*valterm-ratreal r*)) *d*

**instance** ⟨*proof*⟩

**end**

**instantiation** *real* :: *narrowing*
**begin**

**definition**
*narrowing-real* = *Quickcheck-Narrowing.apply* (*Quickcheck-Narrowing.cons Ratreal*) *narrowing*

**instance** ⟨*proof*⟩

**end**

## 97.23 Setup for Nitpick

⟨*ML*⟩

**lemmas** [*nitpick-unfold*] = *inverse-real-inst.inverse-real one-real-inst.one-real ord-real-inst.less-real ord-real-inst.less-eq-real plus-real-inst.plus-real times-real-inst.times-real uminus-real-inst.uminus-real zero-real-inst.zero-real*

## 97.24 Setup for SMT

⟨*ML*⟩

**lemma** [*z3-rule*]:
$0 + x = x$
$x + 0 = x$
$0 * x = 0$
$1 * x = x$
$-x = -1 * x$
$x + y = y + x$
**for** $x$ $y$ :: *real*
⟨*proof*⟩

**lemma** [*smt-arith-multiplication*]:
**fixes** $A$ $B$ :: *real* **and** $p$ $n$ :: *real*
**assumes** $A \leq B$ $0 < n$ $p > 0$
**shows** $(A / n) * p \leq (B / n) * p$
⟨*proof*⟩

**lemma** [*smt-arith-multiplication*]:
**fixes** $A$ $B$ :: *real* **and** $p$ $n$ :: *real*
**assumes** $A < B$ $0 < n$ $p > 0$
**shows** $(A / n) * p < (B / n) * p$
⟨*proof*⟩

**lemma** [*smt-arith-multiplication*]:
**fixes** $A$ $B$ :: *real* **and** $p$ $n$ :: *int*
**assumes** $A \leq B$ $0 < n$ $p > 0$
**shows** $(A / n) * p \leq (B / n) * p$
⟨*proof*⟩

**lemma** [*smt-arith-multiplication*]:
**fixes** $A$ $B$ :: *real* **and** $p$ $n$ :: *int*
**assumes** $A < B$ $0 < n$ $p > 0$
**shows** $(A / n) * p < (B / n) * p$
⟨*proof*⟩

**lemmas** [*smt-arith-multiplication*] =

*verit-le-mono-div*[*THEN mult-left-mono*, *unfolded int-distrib*, *of* - - ‹*nat* (*floor* (- :: *real*))› ‹*nat* (*floor* (- :: *real*))›]
*div-le-mono*[*THEN mult-left-mono*, *unfolded int-distrib*, *of* - - ‹*nat* (*floor* (- :: *real*))› ‹*nat* (*floor* (- :: *real*))›]
*verit-le-mono-div-int*[*THEN mult-left-mono*, *unfolded int-distrib*, *of* - - ‹*floor* (- :: *real*)› ‹*floor* (- :: *real*)›]
*zdiv-mono1*[*THEN mult-left-mono*, *unfolded int-distrib*, *of* - - ‹*floor* (- :: *real*)› ‹*floor* (- :: *real*)›]
*arg-cong*[*of* - - ‹$\lambda a$ :: *real*. *a* / *real* (*n*::*nat*) $*$ *real* (*p*::*nat*)› **for** *n p* :: *nat*, *THEN sym*]
*arg-cong*[*of* - - ‹$\lambda a$ :: *real*. *a* / *real-of-int n* $*$ *real-of-int p*› **for** *n p* :: *int*, *THEN sym*]
*arg-cong*[*of* - - ‹$\lambda a$ :: *real*. *a* / *n* $*$ *p*› **for** *n p* :: *real*, *THEN sym*]

**lemmas** [*smt-arith-simplify*] =
*floor-one floor-numeral div-by-1 times-divide-eq-right*
*nonzero-mult-div-cancel-left division-ring-divide-zero div-0*
*divide-minus-left zero-less-divide-iff*

### 97.25 Setup for Argo

⟨*ML*⟩

**end**

# 98 Topological Spaces

**theory** *Topological-Spaces*
**imports** *Main*
**begin**

**named-theorems** *continuous-intros structural introduction rules for continuity*

### 98.1 Topological space

**class** *open* =
**fixes** *open* :: *'a set* ⇒ *bool*

**class** *topological-space* = *open* +
**assumes** *open-UNIV* [*simp*, *intro*]: *open UNIV*
**assumes** *open-Int* [*intro*]: *open S* ⟹ *open T* ⟹ *open* (*S* ∩ *T*)
**assumes** *open-Union* [*intro*]: $\forall S \in K$. *open S* ⟹ *open* ($\bigcup K$)
**begin**

**definition** *closed* :: *'a set* ⇒ *bool*
**where** *closed S* ⟷ *open* (− *S*)

**lemma** *open-empty* [*continuous-intros*, *intro*, *simp*]: *open* {}
⟨*proof*⟩

**lemma** *open-Un* [*continuous-intros*, *intro*]: *open* $S \Longrightarrow$ *open* $T \Longrightarrow$ *open* $(S \cup T)$
⟨*proof*⟩

**lemma** *open-UN* [*continuous-intros*, *intro*]: $\forall x \in A$. *open* $(B\ x) \Longrightarrow$ *open* $(\bigcup x \in A.\ B\ x)$
⟨*proof*⟩

**lemma** *open-Inter* [*continuous-intros*, *intro*]: *finite* $S \Longrightarrow \forall T \in S$. *open* $T \Longrightarrow$ *open* $(\bigcap S)$
⟨*proof*⟩

**lemma** *open-INT* [*continuous-intros*, *intro*]: *finite* $A \Longrightarrow \forall x \in A$. *open* $(B\ x) \Longrightarrow$ *open* $(\bigcap x \in A.\ B\ x)$
⟨*proof*⟩

**lemma** *openI*:
**assumes** $\bigwedge x.\ x \in S \Longrightarrow \exists T$. *open* $T \wedge x \in T \wedge T \subseteq S$
**shows** *open* $S$
⟨*proof*⟩

**lemma** *open-subopen*: *open* $S \longleftrightarrow (\forall x \in S.\ \exists T.$ *open* $T \wedge x \in T \wedge T \subseteq S)$
⟨*proof*⟩

**lemma** *closed-empty* [*continuous-intros*, *intro*, *simp*]: *closed* $\{\}$
⟨*proof*⟩

**lemma** *closed-Un* [*continuous-intros*, *intro*]: *closed* $S \Longrightarrow$ *closed* $T \Longrightarrow$ *closed* $(S \cup T)$
⟨*proof*⟩

**lemma** *closed-UNIV* [*continuous-intros*, *intro*, *simp*]: *closed UNIV*
⟨*proof*⟩

**lemma** *closed-Int* [*continuous-intros*, *intro*]: *closed* $S \Longrightarrow$ *closed* $T \Longrightarrow$ *closed* $(S \cap T)$
⟨*proof*⟩

**lemma** *closed-INT* [*continuous-intros*, *intro*]: $\forall x \in A$. *closed* $(B\ x) \Longrightarrow$ *closed* $(\bigcap x \in A.\ B\ x)$
⟨*proof*⟩

**lemma** *closed-Inter* [*continuous-intros*, *intro*]: $\forall S \in K$. *closed* $S \Longrightarrow$ *closed* $(\bigcap K)$
⟨*proof*⟩

**lemma** *closed-Union* [*continuous-intros*, *intro*]: *finite* $S \Longrightarrow \forall T \in S$. *closed* $T \Longrightarrow$ *closed* $(\bigcup S)$
⟨*proof*⟩

**lemma** *closed-UN* [*continuous-intros*, *intro*]:
$finite\ A \Longrightarrow \forall x{\in}A.\ closed\ (B\ x) \Longrightarrow closed\ (\bigcup x{\in}A.\ B\ x)$
⟨*proof*⟩

**lemma** *open-closed*: $open\ S \longleftrightarrow closed\ (-\ S)$
⟨*proof*⟩

**lemma** *closed-open*: $closed\ S \longleftrightarrow open\ (-\ S)$
⟨*proof*⟩

**lemma** *open-Diff* [*continuous-intros*, *intro*]: $open\ S \Longrightarrow closed\ T \Longrightarrow open\ (S -\ T)$
⟨*proof*⟩

**lemma** *closed-Diff* [*continuous-intros*, *intro*]: $closed\ S \Longrightarrow open\ T \Longrightarrow closed\ (S -\ T)$
⟨*proof*⟩

**lemma** *open-Compl* [*continuous-intros*, *intro*]: $closed\ S \Longrightarrow open\ (-\ S)$
⟨*proof*⟩

**lemma** *closed-Compl* [*continuous-intros*, *intro*]: $open\ S \Longrightarrow closed\ (-\ S)$
⟨*proof*⟩

**lemma** *open-Collect-neg*: $closed\ \{x.\ P\ x\} \Longrightarrow open\ \{x.\ \neg\ P\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-conj*:
**assumes** $open\ \{x.\ P\ x\}\ open\ \{x.\ Q\ x\}$
**shows** $open\ \{x.\ P\ x \wedge Q\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-disj*:
**assumes** $open\ \{x.\ P\ x\}\ open\ \{x.\ Q\ x\}$
**shows** $open\ \{x.\ P\ x \vee Q\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-ex*: $(\bigwedge i.\ open\ \{x.\ P\ i\ x\}) \Longrightarrow open\ \{x.\ \exists\, i.\ P\ i\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-imp*: $closed\ \{x.\ P\ x\} \Longrightarrow open\ \{x.\ Q\ x\} \Longrightarrow open\ \{x.\ P\ x \longrightarrow Q\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-const*: $open\ \{x.\ P\}$
⟨*proof*⟩

**lemma** *closed-Collect-neg*: $open\ \{x.\ P\ x\} \Longrightarrow closed\ \{x.\ \neg\ P\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-conj*:
**assumes** *closed* $\{x.\ P\ x\}$ *closed* $\{x.\ Q\ x\}$
**shows** *closed* $\{x.\ P\ x \wedge Q\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-disj*:
**assumes** *closed* $\{x.\ P\ x\}$ *closed* $\{x.\ Q\ x\}$
**shows** *closed* $\{x.\ P\ x \vee Q\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-all*: $(\bigwedge i.\ closed\ \{x.\ P\ i\ x\}) \Longrightarrow closed\ \{x.\ \forall i.\ P\ i\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-imp*: $open\ \{x.\ P\ x\} \Longrightarrow closed\ \{x.\ Q\ x\} \Longrightarrow closed\ \{x.\ P\ x \longrightarrow Q\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-const*: *closed* $\{x.\ P\}$
⟨*proof*⟩

**end**

## 98.2 Hausdorff and other separation properties

**class** *t0-space* = *topological-space* +
**assumes** *t0-space*: $x \neq y \Longrightarrow \exists U.\ open\ U \wedge \neg\ (x \in U \longleftrightarrow y \in U)$

**class** *t1-space* = *topological-space* +
**assumes** *t1-space*: $x \neq y \Longrightarrow \exists U.\ open\ U \wedge x \in U \wedge y \notin U$

**instance** *t1-space* ⊆ *t0-space*
⟨*proof*⟩

**context** *t1-space* **begin**

**lemma** *separation-t1*: $x \neq y \longleftrightarrow (\exists U.\ open\ U \wedge x \in U \wedge y \notin U)$
⟨*proof*⟩

**lemma** *closed-singleton* [*iff*]: *closed* $\{a\}$
⟨*proof*⟩

**lemma** *closed-insert* [*continuous-intros*, *simp*]:
**assumes** *closed* $S$
**shows** *closed* (*insert* $a$ $S$)
⟨*proof*⟩

**lemma** *finite-imp-closed*: $finite\ S \Longrightarrow closed\ S$
⟨*proof*⟩

**end**

T2 spaces are also known as Hausdorff spaces.

**class** *t2-space* = *topological-space* +
**assumes** *hausdorff*: $x \neq y \Longrightarrow \exists\, U\ V.\ open\ U \wedge open\ V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\}$

**instance** *t2-space* $\subseteq$ *t1-space*
⟨*proof*⟩

**lemma** (**in** *t2-space*) *separation-t2*: $x \neq y \longleftrightarrow (\exists\, U\ V.\ open\ U \wedge open\ V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\})$
⟨*proof*⟩

**lemma** (**in** *t0-space*) *separation-t0*: $x \neq y \longleftrightarrow (\exists\, U.\ open\ U \wedge \neg\ (x \in U \longleftrightarrow y \in U))$
⟨*proof*⟩

A classical separation axiom for topological space, the T3 axiom – also called regularity: if a point is not in a closed set, then there are open sets separating them.

**class** *t3-space* = *t2-space* +
**assumes** *t3-space*: $closed\ S \Longrightarrow y \notin S \Longrightarrow \exists\, U\ V.\ open\ U \wedge open\ V \wedge y \in U \wedge S \subseteq V \wedge U \cap V = \{\}$

A classical separation axiom for topological space, the T4 axiom – also called normality: if two closed sets are disjoint, then there are open sets separating them.

**class** *t4-space* = *t2-space* +
**assumes** *t4-space*: $closed\ S \Longrightarrow closed\ T \Longrightarrow S \cap T = \{\} \Longrightarrow \exists\, U\ V.\ open\ U \wedge open\ V \wedge S \subseteq U \wedge T \subseteq V \wedge U \cap V = \{\}$

T4 is stronger than T3, and weaker than metric.

**instance** *t4-space* $\subseteq$ *t3-space*
⟨*proof*⟩

A perfect space is a topological space with no isolated points.

**class** *perfect-space* = *topological-space* +
**assumes** *not-open-singleton*: $\neg\ open\ \{x\}$

**lemma** (**in** *perfect-space*) *UNIV-not-singleton*: $UNIV \neq \{x\}$
**for** $x{::}'a$
⟨*proof*⟩

## 98.3 Generators for topologies

**inductive** *generate-topology* :: $'a\ set\ set \Rightarrow 'a\ set \Rightarrow bool$ **for** $S$ :: $'a\ set\ set$

**where**
*UNIV*: *generate-topology S UNIV*
| *Int*: *generate-topology S* $(a \cap b)$ **if** *generate-topology S a* **and** *generate-topology S b*
| *UN*: *generate-topology S* $(\bigcup K)$ **if** $(\bigwedge k.\ k \in K \Longrightarrow$ *generate-topology S k*$)$
| *Basis*: *generate-topology S s* **if** $s \in S$

**hide-fact** (**open**) *UNIV Int UN Basis*

**lemma** *generate-topology-Union*:
$(\bigwedge k.\ k \in I \Longrightarrow$ *generate-topology S* $(K\ k)) \Longrightarrow$ *generate-topology S* $(\bigcup k{\in}I.\ K\ k)$
⟨*proof*⟩

**lemma** *topological-space-generate-topology*: *class.topological-space* (*generate-topology S*)
⟨*proof*⟩

## 98.4 Order topologies

**class** *order-topology* = *order* + *open* +
**assumes** *open-generated-order*: *open* = *generate-topology* (*range* $(\lambda a.\ \{..< a\})$ $\cup$ *range* $(\lambda a.\ \{a <..\})$)
**begin**

**subclass** *topological-space*
⟨*proof*⟩

**lemma** *open-greaterThan* [*continuous-intros*, *simp*]: *open* $\{a <..\}$
⟨*proof*⟩

**lemma** *open-lessThan* [*continuous-intros*, *simp*]: *open* $\{..< a\}$
⟨*proof*⟩

**lemma** *open-greaterThanLessThan* [*continuous-intros*, *simp*]: *open* $\{a <..< b\}$
⟨*proof*⟩

**end**

**class** *linorder-topology* = *linorder* + *order-topology*

**lemma** *closed-atMost* [*continuous-intros*, *simp*]: *closed* $\{..a\}$
**for** $a$ :: *'a::linorder-topology*
⟨*proof*⟩

**lemma** *closed-atLeast* [*continuous-intros*, *simp*]: *closed* $\{a..\}$
**for** $a$ :: *'a::linorder-topology*
⟨*proof*⟩

**lemma** *closed-atLeastAtMost* [*continuous-intros*, *simp*]: *closed* $\{a..b\}$

**for** $a\ b :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** (**in** *order*) *less-separate*:
**assumes** $x < y$
**shows** $\exists\, a\ b.\ x \in \{..< a\} \wedge y \in \{b <..\} \wedge \{..< a\} \cap \{b <..\} = \{\}$
⟨*proof*⟩

**instance** *linorder-topology* $\subseteq$ *t2-space*
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *open-right*:
**assumes** *open* $S\ x \in S$
**and** *gt-ex*: $x < y$
**shows** $\exists\, b{>}x.\ \{x\ ..< b\} \subseteq S$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *open-left*:
**assumes** *open* $S\ x \in S$
**and** *lt-ex*: $y < x$
**shows** $\exists\, b{<}x.\ \{b <..\ x\} \subseteq S$
⟨*proof*⟩

**lemma** *filterlim-atLeastAtMost-at-bot-at-top*:
**fixes** $f\ g :: 'a \Rightarrow 'b :: linorder\text{-}topology$
**assumes** *filterlim f at-bot F filterlim g at-top F*
**assumes** [*simp*]: $\bigwedge a\ b.\ finite\ \{a..b::'b\}$
**shows** *filterlim* $(\lambda x.\ \{f\ x..g\ x\})$ *finite-sets-at-top F*
⟨*proof*⟩

## 98.5 Setup some topologies

### 98.5.1 Boolean is an order topology

**class** *discrete-topology* = *topological-space* +
**assumes** *open-discrete*: $\bigwedge A.\ open\ A$

**instance** *discrete-topology* < *t2-space*
⟨*proof*⟩

**instantiation** *bool* :: *linorder-topology*
**begin**

**definition** *open-bool* :: *bool set* $\Rightarrow$ *bool*
**where** *open-bool* = *generate-topology* (*range* $(\lambda a.\ \{..< a\})$ $\cup$ *range* $(\lambda a.\ \{a <..\})$)

**instance**
⟨*proof*⟩

**end**

**instance** *bool* :: *discrete-topology*
⟨*proof*⟩

**instantiation** *nat* :: *linorder-topology*
**begin**

**definition** *open-nat* :: *nat set* ⇒ *bool*
  **where** $open\text{-}nat = generate\text{-}topology\ (range\ (\lambda a.\ \{..< a\}) \cup range\ (\lambda a.\ \{a <..\}))$

**instance**
  ⟨*proof*⟩

**end**

**instance** *nat* :: *discrete-topology*
⟨*proof*⟩

**instantiation** *int* :: *linorder-topology*
**begin**

**definition** *open-int* :: *int set* ⇒ *bool*
  **where** $open\text{-}int = generate\text{-}topology\ (range\ (\lambda a.\ \{..< a\}) \cup range\ (\lambda a.\ \{a <..\}))$

**instance**
  ⟨*proof*⟩

**end**

**instance** *int* :: *discrete-topology*
⟨*proof*⟩

### 98.5.2 Topological filters

**definition** (**in** *topological-space*) *nhds* :: *'a* ⇒ *'a filter*
  **where** $nhds\ a = (INF\ S{\in}\{S.\ open\ S \wedge a \in S\}.\ principal\ S)$

**definition** (**in** *topological-space*) *at-within* :: *'a* ⇒ *'a set* ⇒ *'a filter*
    (‹*at* (-)/ *within* (-)› [*1000*, *60*] *60*)
  **where** $at\ a\ within\ s = inf\ (nhds\ a)\ (principal\ (s - \{a\}))$

**abbreviation** (**in** *topological-space*) *at* :: *'a* ⇒ *'a filter*  (‹*at*›)
  **where** $at\ x \equiv at\ x\ within\ (CONST\ UNIV)$

**abbreviation** (**in** *order-topology*) *at-right* :: *'a* ⇒ *'a filter*
  **where** $at\text{-}right\ x \equiv at\ x\ within\ \{x <..\}$

**abbreviation** (**in** *order-topology*) *at-left* :: *'a* ⇒ *'a filter*
  **where** $at\text{-}left\ x \equiv at\ x\ within\ \{..< x\}$

**lemma** (**in** *topological-space*) *nhds-generated-topology*:
$open = generate\text{-}topology\ T \Longrightarrow nhds\ x = (INF\ S{\in}\{S{\in}T.\ x \in S\}.\ principal\ S)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *eventually-nhds*:
$eventually\ P\ (nhds\ a) \longleftrightarrow (\exists S.\ open\ S \wedge a \in S \wedge (\forall x{\in}S.\ P\ x))$
⟨*proof*⟩

**lemma** *eventually-eventually*:
$eventually\ (\lambda y.\ eventually\ P\ (nhds\ y))\ (nhds\ x) = eventually\ P\ (nhds\ x)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *eventually-nhds-in-open*:
$open\ s \Longrightarrow x \in s \Longrightarrow eventually\ (\lambda y.\ y \in s)\ (nhds\ x)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *eventually-nhds-x-imp-x*: $eventually\ P\ (nhds\ x) \Longrightarrow P\ x$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *nhds-neq-bot* [*simp*]: $nhds\ a \neq bot$
⟨*proof*⟩

**lemma** (**in** *t1-space*) *t1-space-nhds*: $x \neq y \Longrightarrow (\forall_F\ x\ in\ nhds\ x.\ x \neq y)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *nhds-discrete-open*: $open\ \{x\} \Longrightarrow nhds\ x = principal\ \{x\}$
⟨*proof*⟩

**lemma** (**in** *discrete-topology*) *nhds-discrete*: $nhds\ x = principal\ \{x\}$
⟨*proof*⟩

**lemma** (**in** *discrete-topology*) *at-discrete*: $at\ x\ within\ S = bot$
⟨*proof*⟩

**lemma** (**in** *discrete-topology*) *tendsto-discrete*:
$filterlim\ (f :: 'b \Rightarrow 'a)\ (nhds\ y)\ F \longleftrightarrow eventually\ (\lambda x.\ f\ x = y)\ F$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-eq*:
$at\ x\ within\ s = (INF\ S{\in}\{S.\ open\ S \wedge x \in S\}.\ principal\ (S \cap s - \{x\}))$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *eventually-at-filter*:
$eventually\ P\ (at\ a\ within\ s) \longleftrightarrow eventually\ (\lambda x.\ x \neq a \longrightarrow x \in s \longrightarrow P\ x)\ (nhds\ a)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-le*: $s \subseteq t \Longrightarrow at\ x\ within\ s \leq at\ x\ within\ t$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *eventually-at-topological*:
$eventually\ P\ (at\ a\ within\ s) \longleftrightarrow (\exists S.\ open\ S \wedge a \in S \wedge (\forall x{\in}S.\ x \neq a \longrightarrow x \in s \longrightarrow P\ x))$
⟨*proof*⟩

**lemma** *eventually-nhds-conv-at*:
$eventually\ P\ (nhds\ x) \longleftrightarrow eventually\ P\ (at\ x) \wedge P\ x$
⟨*proof*⟩

**lemma** *eventually-at-in-open*:
**assumes** $open\ A\ x \in A$
**shows** $eventually\ (\lambda y.\ y \in A - \{x\})\ (at\ x)$
⟨*proof*⟩

**lemma** *eventually-at-in-open'*:
**assumes** $open\ A\ x \in A$
**shows** $eventually\ (\lambda y.\ y \in A)\ (at\ x)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-open*: $a \in S \Longrightarrow open\ S \Longrightarrow at\ a\ within\ S = at\ a$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-open-NO-MATCH*:
$a \in s \Longrightarrow open\ s \Longrightarrow NO\text{-}MATCH\ UNIV\ s \Longrightarrow at\ a\ within\ s = at\ a$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-open-subset*:
$a \in S \Longrightarrow open\ S \Longrightarrow S \subseteq T \Longrightarrow at\ a\ within\ T = at\ a$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-nhd*:
**assumes** $x \in S\ open\ S\ T \cap S - \{x\} = U \cap S - \{x\}$
**shows** $at\ x\ within\ T = at\ x\ within\ U$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-empty* [*simp*]: $at\ a\ within\ \{\} = bot$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-within-union*:
$at\ x\ within\ (S \cup T) = sup\ (at\ x\ within\ S)\ (at\ x\ within\ T)$
⟨*proof*⟩

**lemma** (**in** *topological-space*) *at-eq-bot-iff*: $at\ a = bot \longleftrightarrow open\ \{a\}$
⟨*proof*⟩

**lemma** (**in** *t1-space*) *eventually-neq-at-within*:
*eventually* ($\lambda w.\ w \neq x$) (*at z within A*)
⟨*proof*⟩

**lemma** (**in** *perfect-space*) *at-neq-bot* [*simp*]: *at a* $\neq$ *bot*
⟨*proof*⟩

**lemma** (**in** *order-topology*) *nhds-order*:
*nhds x* = *inf* (*INF* $a \in \{x <..\}$. *principal* $\{..< a\}$) (*INF* $a \in \{..< x\}$. *principal* $\{a <..\}$)
⟨*proof*⟩

**lemma** (**in** *topological-space*) *filterlim-at-within-If*:
**assumes** *filterlim f G* (*at x within* ($A \cap \{x.\ P\ x\}$))
**and** *filterlim g G* (*at x within* ($A \cap \{x.\ \neg P\ x\}$))
**shows** *filterlim* ($\lambda x.$ *if P x then f x else g x*) *G* (*at x within A*)
⟨*proof*⟩

**lemma** (**in** *topological-space*) *filterlim-at-If*:
**assumes** *filterlim f G* (*at x within* $\{x.\ P\ x\}$)
**and** *filterlim g G* (*at x within* $\{x.\ \neg P\ x\}$)
**shows** *filterlim* ($\lambda x.$ *if P x then f x else g x*) *G* (*at x*)
⟨*proof*⟩
**lemma** (**in** *linorder-topology*) *at-within-order*:
**assumes** *UNIV* $\neq \{x\}$
**shows** *at x within s* =
*inf* (*INF* $a \in \{x <..\}$. *principal* ($\{..< a\} \cap s - \{x\}$))
(*INF* $a \in \{..< x\}$. *principal* ($\{a <..\} \cap s - \{x\}$))
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *at-left-eq*:
$y < x \Longrightarrow$ *at-left x* = (*INF* $a \in \{..< x\}$. *principal* $\{a <..< x\}$)
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *eventually-at-left*:
$y < x \Longrightarrow$ *eventually P* (*at-left x*) $\longleftrightarrow$ ($\exists\, b<x.\ \forall\, y>b.\ y < x \longrightarrow P\ y$)
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *at-right-eq*:
$x < y \Longrightarrow$ *at-right x* = (*INF* $a \in \{x <..\}$. *principal* $\{x <..< a\}$)
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *eventually-at-right*:
$x < y \Longrightarrow$ *eventually P* (*at-right x*) $\longleftrightarrow$ ($\exists\, b>x.\ \forall\, y>x.\ y < b \longrightarrow P\ y$)
⟨*proof*⟩

**lemma** *eventually-at-right-less*: $\forall_F$ *y in at-right* (*x*::'*a*::{*linorder-topology*, *no-top*}). $x < y$

⟨proof⟩

**lemma** *trivial-limit-at-right-top*: *at-right* (*top*::-::{*order-top*,*linorder-topology*}) = *bot*
⟨proof⟩

**lemma** *trivial-limit-at-left-bot*: *at-left* (*bot*::-::{*order-bot*,*linorder-topology*}) = *bot*
⟨proof⟩

**lemma** *trivial-limit-at-left-real* [*simp*]: ¬ *trivial-limit* (*at-left* *x*)
**for** *x* :: '*a*::{*no-bot*,*dense-order*,*linorder-topology*}
⟨proof⟩

**lemma** *trivial-limit-at-right-real* [*simp*]: ¬ *trivial-limit* (*at-right* *x*)
**for** *x* :: '*a*::{*no-top*,*dense-order*,*linorder-topology*}
⟨proof⟩

**lemma** (**in** *linorder-topology*) *at-eq-sup-left-right*: *at* *x* = *sup* (*at-left* *x*) (*at-right* *x*)
⟨proof⟩

**lemma** (**in** *linorder-topology*) *eventually-at-split*:
*eventually* *P* (*at* *x*) ⟷ *eventually* *P* (*at-left* *x*) ∧ *eventually* *P* (*at-right* *x*)
⟨proof⟩

**lemma** (**in** *order-topology*) *eventually-at-leftI*:
**assumes** $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow P\ x\ a < b$
**shows** *eventually* *P* (*at-left* *b*)
⟨proof⟩

**lemma** (**in** *order-topology*) *eventually-at-rightI*:
**assumes** $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow P\ x\ a < b$
**shows** *eventually* *P* (*at-right* *a*)
⟨proof⟩

**lemma** *eventually-filtercomap-nhds*:
*eventually* *P* (*filtercomap* *f* (*nhds* *x*)) ⟷ (∃ *S*. *open* *S* ∧ *x* ∈ *S* ∧ (∀ *x*. *f* *x* ∈ *S* ⟶ *P* *x*))
⟨proof⟩

**lemma** *eventually-filtercomap-at-topological*:
*eventually* *P* (*filtercomap* *f* (*at* *A* *within* *B*)) ⟷
(∃ *S*. *open* *S* ∧ *A* ∈ *S* ∧ (∀ *x*. *f* *x* ∈ *S* ∩ *B* − {*A*} ⟶ *P* *x*)) (**is** *?lhs* = *?rhs*)
⟨proof⟩

**lemma** *eventually-at-right-field*:
*eventually* *P* (*at-right* *x*) ⟷ (∃ *b*>*x*. ∀ *y*>*x*. *y* < *b* ⟶ *P* *y*)
**for** *x* :: '*a*::{*linordered-field*, *linorder-topology*}
⟨proof⟩

**lemma** *eventually-at-left-field*:
*eventually P* (*at-left x*) $\longleftrightarrow$ ($\exists\, b<x.\ \forall\, y>b.\ y < x \longrightarrow P\ y$)
**for** $x$ :: $'a$::{*linordered-field*, *linorder-topology*}
⟨*proof*⟩

**lemma** *filtermap-nhds-eq-imp-filtermap-at-eq*:
**assumes** *filtermap f* (*nhds z*) = *nhds* ($f\ z$)
**assumes** *eventually* ($\lambda x.\ f\ x = f\ z \longrightarrow x = z$) (*at z*)
**shows** *filtermap f* (*at z*) = *at* ($f\ z$)
⟨*proof*⟩

### 98.5.3 Tendsto

**abbreviation** (**in** *topological-space*)
*tendsto* :: ($'b \Rightarrow 'a$) $\Rightarrow$ $'a \Rightarrow 'b$ *filter* $\Rightarrow$ *bool* (**infixr** ‹$\longrightarrow$› *55*)
**where** ($f \longrightarrow l$) $F \equiv$ *filterlim f* (*nhds l*) $F$

**definition** (**in** *t2-space*) *Lim* :: $'f$ *filter* $\Rightarrow$ ($'f \Rightarrow 'a$) $\Rightarrow$ $'a$
**where** *Lim A f* = (*THE l.* ($f \longrightarrow l$) $A$)

**lemma** (**in** *topological-space*) *tendsto-eq-rhs*: ($f \longrightarrow x$) $F \Longrightarrow x = y \Longrightarrow$ ($f \longrightarrow y$) $F$
⟨*proof*⟩

**named-theorems** *tendsto-intros introduction rules for tendsto*
⟨*ML*⟩

**context** *topological-space* **begin**

**lemma** *tendsto-def*:
($f \longrightarrow l$) $F \longleftrightarrow$ ($\forall\, S.$ *open* $S \longrightarrow l \in S \longrightarrow$ *eventually* ($\lambda x.\ f\ x \in S$) $F$)
⟨*proof*⟩

**lemma** *tendsto-cong*: ($f \longrightarrow c$) $F \longleftrightarrow$ ($g \longrightarrow c$) $F$ **if** *eventually* ($\lambda x.\ f\ x = g\ x$) $F$
⟨*proof*⟩

**lemma** *tendsto-mono*: $F \leq F' \Longrightarrow$ ($f \longrightarrow l$) $F' \Longrightarrow$ ($f \longrightarrow l$) $F$
⟨*proof*⟩

**lemma** *tendsto-ident-at* [*tendsto-intros*, *simp*, *intro*]: (($\lambda x.\ x$) $\longrightarrow a$) (*at a within s*)
⟨*proof*⟩

**lemma** *tendsto-const* [*tendsto-intros*, *simp*, *intro*]: (($\lambda x.\ k$) $\longrightarrow k$) $F$
⟨*proof*⟩

**lemma** *filterlim-at*:

$(LIM\ x\ F.\ f\ x\ :>\ at\ b\ within\ s) \longleftrightarrow eventually\ (\lambda x.\ f\ x \in s \wedge f\ x \neq b)\ F \wedge (f \longrightarrow b)\ F$
⟨*proof*⟩

**lemma** (**in** −)
**assumes** *filterlim* $f$ (*nhds* $L$) $F$
**shows** *tendsto-imp-filterlim-at-right*:
$eventually\ (\lambda x.\ f\ x > L)\ F \Longrightarrow filterlim\ f\ (at\text{-}right\ L)\ F$
**and** *tendsto-imp-filterlim-at-left*:
$eventually\ (\lambda x.\ f\ x < L)\ F \Longrightarrow filterlim\ f\ (at\text{-}left\ L)\ F$
⟨*proof*⟩

**lemma** *filterlim-at-withinI*:
**assumes** *filterlim* $f$ (*nhds* $c$) $F$
**assumes** $eventually\ (\lambda x.\ f\ x \in A - \{c\})\ F$
**shows** *filterlim* $f$ (*at* $c$ *within* $A$) $F$
⟨*proof*⟩

**lemma** *filterlim-atI*:
**assumes** *filterlim* $f$ (*nhds* $c$) $F$
**assumes** $eventually\ (\lambda x.\ f\ x \neq c)\ F$
**shows** *filterlim* $f$ (*at* $c$) $F$
⟨*proof*⟩

**lemma** *topological-tendstoI*:
$(\bigwedge S.\ open\ S \Longrightarrow l \in S \Longrightarrow eventually\ (\lambda x.\ f\ x \in S)\ F) \Longrightarrow (f \longrightarrow l)\ F$
⟨*proof*⟩

**lemma** *topological-tendstoD*:
$(f \longrightarrow l)\ F \Longrightarrow open\ S \Longrightarrow l \in S \Longrightarrow eventually\ (\lambda x.\ f\ x \in S)\ F$
⟨*proof*⟩

**lemma** *tendsto-bot* [*simp*]: $(f \longrightarrow a)\ bot$
⟨*proof*⟩

**lemma** *tendsto-eventually*: $eventually\ (\lambda x.\ f\ x = l)\ net \Longrightarrow ((\lambda x.\ f\ x) \longrightarrow l)\ net$
⟨*proof*⟩

**lemma** *tendsto-principal-singleton*[*simp*]:
**shows** $(f \longrightarrow f\ x)\ (principal\ \{x\})$
⟨*proof*⟩

**end**

**lemma** (**in** *topological-space*) *filterlim-within-subset*:
$filterlim\ f\ l\ (at\ x\ within\ S) \Longrightarrow T \subseteq S \Longrightarrow filterlim\ f\ l\ (at\ x\ within\ T)$
⟨*proof*⟩

**lemmas** *tendsto-within-subset* = *filterlim-within-subset*

**lemma** (**in** *order-topology*) *order-tendsto-iff*:
$(f \longrightarrow x)\ F \longleftrightarrow (\forall\, l{<}x.\ eventually\ (\lambda x.\ l < f\ x)\ F) \wedge (\forall\, u{>}x.\ eventually\ (\lambda x.\ f\ x < u)\ F)$
⟨*proof*⟩

**lemma** (**in** *order-topology*) *order-tendstoI*:
$(\bigwedge a.\ a < y \Longrightarrow eventually\ (\lambda x.\ a < f\ x)\ F) \Longrightarrow (\bigwedge a.\ y < a \Longrightarrow eventually\ (\lambda x.\ f\ x < a)\ F) \Longrightarrow$
$(f \longrightarrow y)\ F$
⟨*proof*⟩

**lemma** (**in** *order-topology*) *order-tendstoD*:
**assumes** $(f \longrightarrow y)\ F$
**shows** $a < y \Longrightarrow eventually\ (\lambda x.\ a < f\ x)\ F$
**and** $y < a \Longrightarrow eventually\ (\lambda x.\ f\ x < a)\ F$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *tendsto-max*[*tendsto-intros*]:
**assumes** $X$: $(X \longrightarrow x)\ net$
**and** $Y$: $(Y \longrightarrow y)\ net$
**shows** $((\lambda x.\ max\ (X\ x)\ (Y\ x)) \longrightarrow max\ x\ y)\ net$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *tendsto-min*[*tendsto-intros*]:
**assumes** $X$: $(X \longrightarrow x)\ net$
**and** $Y$: $(Y \longrightarrow y)\ net$
**shows** $((\lambda x.\ min\ (X\ x)\ (Y\ x)) \longrightarrow min\ x\ y)\ net$
⟨*proof*⟩

**lemma** (**in** *order-topology*)
**assumes** $a < b$
**shows** *at-within-Icc-at-right*: $at\ a\ within\ \{a..b\} = at\text{-}right\ a$
**and** *at-within-Icc-at-left*: $at\ b\ within\ \{a..b\} = at\text{-}left\ b$
⟨*proof*⟩

**lemma** (**in** *order-topology*)
**shows** *at-within-Ici-at-right*: $at\ a\ within\ \{a..\} = at\text{-}right\ a$
**and** *at-within-Iic-at-left*: $at\ a\ within\ \{..a\} = at\text{-}left\ a$
⟨*proof*⟩

**lemma** (**in** *order-topology*) *at-within-Icc-at*: $a < x \Longrightarrow x < b \Longrightarrow at\ x\ within\ \{a..b\} = at\ x$
⟨*proof*⟩

**lemma** (**in** *t2-space*) *tendsto-unique*:
**assumes** $F \neq bot$
**and** $(f \longrightarrow a)\ F$

**and** $(f \longrightarrow b)$ $F$
**shows** $a = b$
⟨*proof*⟩

**lemma** (**in** *t2-space*) *tendsto-const-iff*:
**fixes** $a$ $b$ :: $'a$
**assumes** $\neg$ *trivial-limit* $F$
**shows** $((\lambda x.\ a) \longrightarrow b)$ $F \longleftrightarrow a = b$
⟨*proof*⟩

**lemma** (**in** *t2-space*) *tendsto-unique'*:
**assumes** $F \neq bot$
**shows** $\exists_{\leq 1} l.\ (f \longrightarrow l)$ $F$
⟨*proof*⟩

**lemma** *Lim-in-closed-set*:
**assumes** *closed* $S$ *eventually* $(\lambda x.\ f(x) \in S)$ $F$ $F \neq bot$ $(f \longrightarrow l)$ $F$
**shows** $l \in S$
⟨*proof*⟩

**lemma** (**in** *t3-space*) *nhds-closed*:
**assumes** $x \in A$ **and** *open* $A$
**shows** $\exists A'.\ x \in A' \wedge$ *closed* $A' \wedge A' \subseteq A \wedge$ *eventually* $(\lambda y.\ y \in A')$ (*nhds* $x$)
⟨*proof*⟩

**lemma** (**in** *order-topology*) *increasing-tendsto*:
**assumes** *bdd*: *eventually* $(\lambda n.\ f\ n \leq l)$ $F$
**and** *en*: $\bigwedge x.\ x < l \Longrightarrow$ *eventually* $(\lambda n.\ x < f\ n)$ $F$
**shows** $(f \longrightarrow l)$ $F$
⟨*proof*⟩

**lemma** (**in** *order-topology*) *decreasing-tendsto*:
**assumes** *bdd*: *eventually* $(\lambda n.\ l \leq f\ n)$ $F$
**and** *en*: $\bigwedge x.\ l < x \Longrightarrow$ *eventually* $(\lambda n.\ f\ n < x)$ $F$
**shows** $(f \longrightarrow l)$ $F$
⟨*proof*⟩

**lemma** (**in** *order-topology*) *tendsto-sandwich*:
**assumes** *ev*: *eventually* $(\lambda n.\ f\ n \leq g\ n)$ *net* *eventually* $(\lambda n.\ g\ n \leq h\ n)$ *net*
**assumes** *lim*: $(f \longrightarrow c)$ *net* $(h \longrightarrow c)$ *net*
**shows** $(g \longrightarrow c)$ *net*
⟨*proof*⟩

**lemma** (**in** *t1-space*) *limit-frequently-eq*:
**assumes** $F \neq bot$
**and** *frequently* $(\lambda x.\ f\ x = c)$ $F$
**and** $(f \longrightarrow d)$ $F$
**shows** $d = c$
⟨*proof*⟩

**lemma** (**in** *t1-space*) *tendsto-imp-eventually-ne*:
**assumes** $(f \longrightarrow c)\ F\ c \neq c'$
**shows** *eventually* $(\lambda z.\ f\ z \neq c')\ F$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *tendsto-le*:
**assumes** $F$: $\neg$ *trivial-limit* $F$
**and** $x$: $(f \longrightarrow x)\ F$
**and** $y$: $(g \longrightarrow y)\ F$
**and** *ev*: *eventually* $(\lambda x.\ g\ x \leq f\ x)\ F$
**shows** $y \leq x$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *tendsto-lowerbound*:
**assumes** $x$: $(f \longrightarrow x)\ F$
**and** *ev*: *eventually* $(\lambda i.\ a \leq f\ i)\ F$
**and** $F$: $\neg$ *trivial-limit* $F$
**shows** $a \leq x$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *tendsto-upperbound*:
**assumes** $x$: $(f \longrightarrow x)\ F$
**and** *ev*: *eventually* $(\lambda i.\ a \geq f\ i)\ F$
**and** $F$: $\neg$ *trivial-limit* $F$
**shows** $a \geq x$
⟨*proof*⟩

**lemma** *filterlim-at-within-not-equal*:
**fixes** $f$::$'a \Rightarrow$ $'b$::*t2-space*
**assumes** *filterlim* $f$ (*at* $a$ *within* $s$) $F$
**shows** *eventually* $(\lambda w.\ f\ w \in s \wedge f\ w \neq b)\ F$
⟨*proof*⟩

### 98.5.4 Rules about *Lim*

**lemma** *tendsto-Lim*: $\neg$ *trivial-limit net* $\Longrightarrow (f \longrightarrow l)$ *net* $\Longrightarrow$ *Lim net* $f = l$
⟨*proof*⟩

**lemma** *Lim-ident-at*: $\neg$ *trivial-limit* (*at* $x$ *within* $s$) $\Longrightarrow$ *Lim* (*at* $x$ *within* $s$) $(\lambda x.\ x) = x$
⟨*proof*⟩

**lemma** *Lim-cong*:
**assumes** $\forall_F\ x$ *in* $F$. $f\ x = g\ x\ F = G$
**shows** *Lim* $F\ f =$ *Lim* $F\ g$
⟨*proof*⟩

**lemma** *eventually-Lim-ident-at*:

$(\forall_F\ y\ in\ at\ x\ within\ X.\ P\ (Lim\ (at\ x\ within\ X)\ (\lambda x.\ x))\ y) \longleftrightarrow$
$(\forall_F\ y\ in\ at\ x\ within\ X.\ P\ x\ y)$ **for** $x::'a::t2\text{-}space$
⟨*proof*⟩

**lemma** *filterlim-at-bot-at-right*:
**fixes** $f :: 'a::linorder\text{-}topology \Rightarrow 'b::linorder$
**assumes** *mono*: $\bigwedge x\ y.\ Q\ x \Longrightarrow Q\ y \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$
**and** *bij*: $\bigwedge x.\ P\ x \Longrightarrow f\ (g\ x) = x\ \bigwedge x.\ P\ x \Longrightarrow Q\ (g\ x)$
**and** $Q$: *eventually* $Q$ (*at-right* $a$)
**and** *bound*: $\bigwedge b.\ Q\ b \Longrightarrow a < b$
**and** $P$: *eventually* $P$ *at-bot*
**shows** *filterlim* $f$ *at-bot* (*at-right* $a$)
⟨*proof*⟩

**lemma** *filterlim-at-top-at-left*:
**fixes** $f :: 'a::linorder\text{-}topology \Rightarrow 'b::linorder$
**assumes** *mono*: $\bigwedge x\ y.\ Q\ x \Longrightarrow Q\ y \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$
**and** *bij*: $\bigwedge x.\ P\ x \Longrightarrow f\ (g\ x) = x\ \bigwedge x.\ P\ x \Longrightarrow Q\ (g\ x)$
**and** $Q$: *eventually* $Q$ (*at-left* $a$)
**and** *bound*: $\bigwedge b.\ Q\ b \Longrightarrow b < a$
**and** $P$: *eventually* $P$ *at-top*
**shows** *filterlim* $f$ *at-top* (*at-left* $a$)
⟨*proof*⟩

**lemma** *filterlim-split-at*:
*filterlim* $f$ $F$ (*at-left* $x$) $\Longrightarrow$ *filterlim* $f$ $F$ (*at-right* $x$) $\Longrightarrow$
*filterlim* $f$ $F$ (*at* $x$)
**for** $x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *filterlim-at-split*:
*filterlim* $f$ $F$ (*at* $x$) $\longleftrightarrow$ *filterlim* $f$ $F$ (*at-left* $x$) $\wedge$ *filterlim* $f$ $F$ (*at-right* $x$)
**for** $x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *eventually-nhds-top*:
**fixes** $P :: 'a :: \{order\text{-}top, linorder\text{-}topology\} \Rightarrow bool$
**and** $b :: 'a$
**assumes** $b < top$
**shows** *eventually* $P$ (*nhds top*) $\longleftrightarrow (\exists\, b<top.\ (\forall\, z.\ b < z \longrightarrow P\ z))$
⟨*proof*⟩

**lemma** *tendsto-at-within-iff-tendsto-nhds*:
$(g \longrightarrow g\ l)\ (at\ l\ within\ S) \longleftrightarrow (g \longrightarrow g\ l)\ (inf\ (nhds\ l)\ (principal\ S))$
⟨*proof*⟩

## 98.6 Limits on sequences

**abbreviation** (**in** *topological-space*)

*LIMSEQ* :: [*nat* ⇒ ′*a*, ′*a*] ⇒ *bool* (‹(‹notation=‹infix LIMSEQ››(-)/ ⟶ (-))›
[*60*, *60*] *60*)
**where** $X \longrightarrow L \equiv (X \longrightarrow L)$ *sequentially*

**abbreviation** (**in** *t2-space*) *lim* :: (*nat* ⇒ ′*a*) ⇒ ′*a*
**where** *lim X* ≡ *Lim sequentially X*

**definition** (**in** *topological-space*) *convergent* :: (*nat* ⇒ ′*a*) ⇒ *bool*
**where** *convergent X* = (∃ *L*. $X \longrightarrow L$)

**lemma** *lim-def*: *lim X* = (*THE L*. $X \longrightarrow L$)
⟨*proof*⟩

**lemma** *lim-explicit*:
$f \longrightarrow f0 \longleftrightarrow (\forall S.\ open\ S \longrightarrow f0 \in S \longrightarrow (\exists N.\ \forall n \geq N.\ f\ n \in S))$
⟨*proof*⟩

**lemma** *closed-sequentially*:
**assumes** *closed S* **and** $\bigwedge n.\ f\ n \in S$ **and** $f \longrightarrow l$
**shows** $l \in S$
⟨*proof*⟩

## 98.7 Monotone sequences and subsequences

Definition of monotonicity. The use of disjunction here complicates proofs considerably. One alternative is to add a Boolean argument to indicate the direction. Another is to develop the notions of increasing and decreasing first.

**definition** *monoseq* :: (*nat* ⇒ ′*a*::*order*) ⇒ *bool*
**where** *monoseq X* $\longleftrightarrow (\forall m.\ \forall n \geq m.\ X\ m \leq X\ n) \lor (\forall m.\ \forall n \geq m.\ X\ n \leq X\ m)$

**abbreviation** *incseq* :: (*nat* ⇒ ′*a*::*order*) ⇒ *bool*
**where** *incseq X* ≡ *mono X*

**lemma** *incseq-def*: *incseq X* $\longleftrightarrow (\forall m.\ \forall n \geq m.\ X\ n \geq X\ m)$
⟨*proof*⟩

**abbreviation** *decseq* :: (*nat* ⇒ ′*a*::*order*) ⇒ *bool*
**where** *decseq X* ≡ *antimono X*

**lemma** *decseq-def*: *decseq X* $\longleftrightarrow (\forall m.\ \forall n \geq m.\ X\ n \leq X\ m)$
⟨*proof*⟩

### 98.7.1 Definition of subsequence.

**lemma** *strict-mono-leD*: *strict-mono r* $\Longrightarrow m \leq n \Longrightarrow r\ m \leq r\ n$
⟨*proof*⟩

**lemma** *strict-mono-id*: *strict-mono id*

⟨proof⟩

**lemma** *incseq-SucI*: $(\bigwedge n.\ X\ n \leq X\ (Suc\ n)) \Longrightarrow incseq\ X$
⟨proof⟩

**lemma** *incseqD*: $incseq\ f \Longrightarrow i \leq j \Longrightarrow f\ i \leq f\ j$
⟨proof⟩

**lemma** *incseq-SucD*: $incseq\ A \Longrightarrow A\ i \leq A\ (Suc\ i)$
⟨proof⟩

**lemma** *incseq-Suc-iff*: $incseq\ f \longleftrightarrow (\forall n.\ f\ n \leq f\ (Suc\ n))$
⟨proof⟩

**lemma** *incseq-const*[*simp*, *intro*]: $incseq\ (\lambda x.\ k)$
⟨proof⟩

**lemma** *decseq-SucI*: $(\bigwedge n.\ X\ (Suc\ n) \leq X\ n) \Longrightarrow decseq\ X$
⟨proof⟩

**lemma** *decseqD*: $decseq\ f \Longrightarrow i \leq j \Longrightarrow f\ j \leq f\ i$
⟨proof⟩

**lemma** *decseq-SucD*: $decseq\ A \Longrightarrow A\ (Suc\ i) \leq A\ i$
⟨proof⟩

**lemma** *decseq-Suc-iff*: $decseq\ f \longleftrightarrow (\forall n.\ f\ (Suc\ n) \leq f\ n)$
⟨proof⟩

**lemma** *decseq-const*[*simp*, *intro*]: $decseq\ (\lambda x.\ k)$
⟨proof⟩

**lemma** *monoseq-iff*: $monoseq\ X \longleftrightarrow incseq\ X \lor decseq\ X$
⟨proof⟩

**lemma** *monoseq-Suc*: $monoseq\ X \longleftrightarrow (\forall n.\ X\ n \leq X\ (Suc\ n)) \lor (\forall n.\ X\ (Suc\ n) \leq X\ n)$
⟨proof⟩

**lemma** *monoI1*: $\forall m.\ \forall n \geq m.\ X\ m \leq X\ n \Longrightarrow monoseq\ X$
⟨proof⟩

**lemma** *monoI2*: $\forall m.\ \forall n \geq m.\ X\ n \leq X\ m \Longrightarrow monoseq\ X$
⟨proof⟩

**lemma** *mono-SucI1*: $\forall n.\ X\ n \leq X\ (Suc\ n) \Longrightarrow monoseq\ X$
⟨proof⟩

**lemma** *mono-SucI2*: $\forall n.\ X\ (Suc\ n) \leq X\ n \Longrightarrow monoseq\ X$

⟨*proof*⟩

**lemma** *monoseq-minus*:
**fixes** $a :: nat \Rightarrow 'a::ordered\text{-}ab\text{-}group\text{-}add$
**assumes** *monoseq a*
**shows** *monoseq* $(\lambda\ n.\ -\ a\ n)$
⟨*proof*⟩

### 98.7.2 Subsequence (alternative definition, (e.g. Hoskins)

For any sequence, there is a monotonic subsequence.

**lemma** *seq-monosub*:
**fixes** $s :: nat \Rightarrow 'a::linorder$
**shows** $\exists f.\ strict\text{-}mono\ f \wedge monoseq\ (\lambda n.\ (s\ (f\ n)))$
⟨*proof*⟩

**lemma** *seq-suble*:
**assumes** *sf*: $strict\text{-}mono\ (f :: nat \Rightarrow nat)$
**shows** $n \leq f\ n$
⟨*proof*⟩

**lemma** *eventually-subseq*:
$strict\text{-}mono\ r \Longrightarrow eventually\ P\ sequentially \Longrightarrow eventually\ (\lambda n.\ P\ (r\ n))\ sequentially$
⟨*proof*⟩

**lemma** *not-eventually-sequentiallyD*:
**assumes** $\neg$ *eventually P sequentially*
**shows** $\exists\ r::nat \Rightarrow nat.\ strict\text{-}mono\ r \wedge (\forall\ n.\ \neg\ P\ (r\ n))$
⟨*proof*⟩

**lemma** *sequentially-offset*:
**assumes** *eventually* $(\lambda i.\ P\ i)$ *sequentially*
**shows** *eventually* $(\lambda i.\ P\ (i + k))$ *sequentially*
⟨*proof*⟩

**lemma** *seq-offset-neg*:
$(f \longrightarrow l)\ sequentially \Longrightarrow ((\lambda i.\ f(i - k)) \longrightarrow l)\ sequentially$
⟨*proof*⟩

**lemma** *filterlim-subseq*: $strict\text{-}mono\ f \Longrightarrow filterlim\ f\ sequentially\ sequentially$
⟨*proof*⟩

**lemma** *strict-mono-o*: $strict\text{-}mono\ r \Longrightarrow strict\text{-}mono\ s \Longrightarrow strict\text{-}mono\ (r \circ s)$
⟨*proof*⟩

**lemma** *strict-mono-compose*: $strict\text{-}mono\ r \Longrightarrow strict\text{-}mono\ s \Longrightarrow strict\text{-}mono\ (\lambda x.\ r\ (s\ x))$
⟨*proof*⟩

**lemma** *incseq-imp-monoseq*: $incseq\ X \Longrightarrow monoseq\ X$
⟨*proof*⟩

**lemma** *decseq-imp-monoseq*: $decseq\ X \Longrightarrow monoseq\ X$
⟨*proof*⟩

**lemma** *decseq-eq-incseq*: $decseq\ X = incseq\ (\lambda n.\ -\ X\ n)$
**for** $X :: nat \Rightarrow 'a::ordered\text{-}ab\text{-}group\text{-}add$
⟨*proof*⟩

**lemma** *INT-decseq-offset*:
**assumes** *decseq F*
**shows** $(\bigcap i.\ F\ i) = (\bigcap i\in\{n..\}.\ F\ i)$
⟨*proof*⟩

**lemma** *LIMSEQ-const-iff*: $(\lambda n.\ k) \longrightarrow l \longleftrightarrow k = l$
**for** $k\ l :: 'a::t2\text{-}space$
⟨*proof*⟩

**lemma** *LIMSEQ-SUP*: $incseq\ X \Longrightarrow X \longrightarrow (SUP\ i.\ X\ i :: 'a::\{complete\text{-}linorder,linorder\text{-}topology\})$
⟨*proof*⟩

**lemma** *LIMSEQ-INF*: $decseq\ X \Longrightarrow X \longrightarrow (INF\ i.\ X\ i :: 'a::\{complete\text{-}linorder,linorder\text{-}topology\})$
⟨*proof*⟩

**lemma** *LIMSEQ-ignore-initial-segment*: $f \longrightarrow a \Longrightarrow (\lambda n.\ f\ (n + k)) \longrightarrow a$
⟨*proof*⟩

**lemma** *LIMSEQ-offset*: $(\lambda n.\ f\ (n + k)) \longrightarrow a \Longrightarrow f \longrightarrow a$
⟨*proof*⟩

**lemma** *LIMSEQ-Suc*: $f \longrightarrow l \Longrightarrow (\lambda n.\ f\ (Suc\ n)) \longrightarrow l$
⟨*proof*⟩

**lemma** *LIMSEQ-imp-Suc*: $(\lambda n.\ f\ (Suc\ n)) \longrightarrow l \Longrightarrow f \longrightarrow l$
⟨*proof*⟩

**lemma** *LIMSEQ-lessThan-iff-atMost*:
**shows** $(\lambda n.\ f\ \{..<n\}) \longrightarrow x \longleftrightarrow (\lambda n.\ f\ \{..n\}) \longrightarrow x$
⟨*proof*⟩

**lemma** (**in** *t2-space*) *LIMSEQ-Uniq*: $\exists_{\leq 1} l.\ X \longrightarrow l$
⟨*proof*⟩

**lemma** (**in** *t2-space*) *LIMSEQ-unique*: $X \longrightarrow a \Longrightarrow X \longrightarrow b \Longrightarrow a = b$
⟨*proof*⟩

**lemma** *LIMSEQ-le-const*: $X \longrightarrow x \Longrightarrow \exists N.\ \forall n\geq N.\ a \leq X\ n \Longrightarrow a \leq x$

**for** $a\ x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *LIMSEQ-le*: $X \longrightarrow x \Longrightarrow Y \longrightarrow y \Longrightarrow \exists N.\ \forall n\geq N.\ X\ n \leq Y\ n \Longrightarrow x \leq y$
**for** $x\ y :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *LIMSEQ-le-const2*: $X \longrightarrow x \Longrightarrow \exists N.\ \forall n\geq N.\ X\ n \leq a \Longrightarrow x \leq a$
**for** $a\ x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *Lim-bounded*: $f \longrightarrow l \Longrightarrow \forall n\geq M.\ f\ n \leq C \Longrightarrow l \leq C$
**for** $l :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *Lim-bounded2*:
**fixes** $f :: nat \Rightarrow 'a::linorder\text{-}topology$
**assumes** $lim{:}f \longrightarrow l$ **and** $ge{:}\ \forall n\geq N.\ f\ n \geq C$
**shows** $l \geq C$
⟨*proof*⟩

**lemma** *lim-mono*:
**fixes** $X\ Y :: nat \Rightarrow 'a::linorder\text{-}topology$
**assumes** $\bigwedge n.\ N \leq n \Longrightarrow X\ n \leq Y\ n$
**and** $X \longrightarrow x$
**and** $Y \longrightarrow y$
**shows** $x \leq y$
⟨*proof*⟩

**lemma** *Sup-lim*:
**fixes** $a :: 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
**assumes** $\bigwedge n.\ b\ n \in s$
**and** $b \longrightarrow a$
**shows** $a \leq Sup\ s$
⟨*proof*⟩

**lemma** *Inf-lim*:
**fixes** $a :: 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
**assumes** $\bigwedge n.\ b\ n \in s$
**and** $b \longrightarrow a$
**shows** $Inf\ s \leq a$
⟨*proof*⟩

**lemma** *SUP-Lim*:
**fixes** $X :: nat \Rightarrow 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
**assumes** $inc{:}\ incseq\ X$
**and** $l{:}\ X \longrightarrow l$
**shows** $(SUP\ n.\ X\ n) = l$

⟨*proof*⟩

**lemma** *INF-Lim*:
**fixes** $X :: nat \Rightarrow 'a::\{complete\text{-}linorder,linorder\text{-}topology\}$
**assumes** *dec*: *decseq X*
**and** $l$: $X \longlonglongrightarrow l$
**shows** $(INF\ n.\ X\ n) = l$
⟨*proof*⟩

**lemma** *convergentD*: $convergent\ X \Longrightarrow \exists L.\ X \longlonglongrightarrow L$
⟨*proof*⟩

**lemma** *convergentI*: $X \longlonglongrightarrow L \Longrightarrow convergent\ X$
⟨*proof*⟩

**lemma** *convergent-LIMSEQ-iff*: $convergent\ X \longleftrightarrow X \longlonglongrightarrow lim\ X$
⟨*proof*⟩

**lemma** *convergent-const*: $convergent\ (\lambda n.\ c)$
⟨*proof*⟩

**lemma** *monoseq-le*:
$monoseq\ a \Longrightarrow a \longlonglongrightarrow x \Longrightarrow$
$(\forall n.\ a\ n \le x) \wedge (\forall m.\ \forall n{\ge}m.\ a\ m \le a\ n) \vee$
$(\forall n.\ x \le a\ n) \wedge (\forall m.\ \forall n{\ge}m.\ a\ n \le a\ m)$
**for** $x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *LIMSEQ-subseq-LIMSEQ*: $X \longlonglongrightarrow L \Longrightarrow strict\text{-}mono\ f \Longrightarrow (X \circ f) \longlonglongrightarrow L$
⟨*proof*⟩

**lemma** *convergent-subseq-convergent*: $convergent\ X \Longrightarrow strict\text{-}mono\ f \Longrightarrow convergent\ (X \circ f)$
⟨*proof*⟩

**lemma** *limI*: $X \longlonglongrightarrow L \Longrightarrow lim\ X = L$
⟨*proof*⟩

**lemma** *lim-le*: $convergent\ f \Longrightarrow (\bigwedge n.\ f\ n \le x) \Longrightarrow lim\ f \le x$
**for** $x :: 'a::linorder\text{-}topology$
⟨*proof*⟩

**lemma** *lim-const* [*simp*]: $lim\ (\lambda m.\ a) = a$
⟨*proof*⟩

### 98.7.3 Increasing and Decreasing Series

**lemma** *incseq-le*: $incseq\ X \Longrightarrow X \longlonglongrightarrow L \Longrightarrow X\ n \le L$

**for** $L$ :: $'a$::*linorder-topology*
⟨*proof*⟩

**lemma** *decseq-ge*: $decseq\ X \Longrightarrow X \longrightarrow L \Longrightarrow L \leq X\ n$
**for** $L$ :: $'a$::*linorder-topology*
⟨*proof*⟩

## 98.8 First countable topologies

**class** *first-countable-topology* = *topological-space* +
**assumes** *first-countable-basis*:
$\exists A::nat \Rightarrow 'a\ set.\ (\forall i.\ x \in A\ i \wedge open\ (A\ i)) \wedge (\forall S.\ open\ S \wedge x \in S \longrightarrow (\exists i.\ A\ i \subseteq S))$

**lemma** (**in** *first-countable-topology*) *countable-basis-at-decseq*:
**obtains** $A$ :: $nat \Rightarrow 'a\ set$ **where**
$\bigwedge i.\ open\ (A\ i)\ \bigwedge i.\ x \in (A\ i)$
$\bigwedge S.\ open\ S \Longrightarrow x \in S \Longrightarrow eventually\ (\lambda i.\ A\ i \subseteq S)\ sequentially$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *nhds-countable*:
**obtains** $X$ :: $nat \Rightarrow 'a\ set$
**where** $decseq\ X\ \bigwedge n.\ open\ (X\ n)\ \bigwedge n.\ x \in X\ n\ nhds\ x = (INF\ n.\ principal\ (X\ n))$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *countable-basis*:
**obtains** $A$ :: $nat \Rightarrow 'a\ set$ **where**
$\bigwedge i.\ open\ (A\ i)\ \bigwedge i.\ x \in A\ i$
$\bigwedge F.\ (\forall n.\ F\ n \in A\ n) \Longrightarrow F \longrightarrow x$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *sequentially-imp-eventually-nhds-within*:
**assumes** $\forall f.\ (\forall n.\ f\,n \in s) \wedge f \longrightarrow a \longrightarrow eventually\ (\lambda n.\ P\ (f\,n))\ sequentially$
**shows** $eventually\ P\ (inf\ (nhds\ a)\ (principal\ s))$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *eventually-nhds-within-iff-sequentially*:
$eventually\ P\ (inf\ (nhds\ a)\ (principal\ s)) \longleftrightarrow$
$(\forall f.\ (\forall n.\ f\,n \in s) \wedge f \longrightarrow a \longrightarrow eventually\ (\lambda n.\ P\ (f\,n))\ sequentially)$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *eventually-nhds-iff-sequentially*:
$eventually\ P\ (nhds\ a) \longleftrightarrow (\forall f.\ f \longrightarrow a \longrightarrow eventually\ (\lambda n.\ P\ (f\,n))\ sequentially)$
⟨*proof*⟩

**lemma** *Inf-as-limit*:

**fixes** $A$::$'a$::{*linorder-topology*, *first-countable-topology*, *complete-linorder*} *set*
**assumes** $A \neq \{\}$
**shows** $\exists\, u.\ (\forall\, n.\ u\ n \in A) \wedge u \longrightarrow \mathit{Inf}\ A$
⟨*proof*⟩

**lemma** *tendsto-at-iff-sequentially*:
$(f \longrightarrow a)\ (at\ x\ within\ s) \longleftrightarrow (\forall\, X.\ (\forall\, i.\ X\ i \in s - \{x\}) \longrightarrow X \longrightarrow x \longrightarrow ((f \circ X) \longrightarrow a))$
**for** $f$ :: $'a$::*first-countable-topology* $\Rightarrow$ -
⟨*proof*⟩

**lemma** *approx-from-above-dense-linorder*:
**fixes** $x$::$'a$::{*dense-linorder*, *linorder-topology*, *first-countable-topology*}
**assumes** $x < y$
**shows** $\exists\, u.\ (\forall\, n.\ u\ n > x) \wedge (u \longrightarrow x)$
⟨*proof*⟩

**lemma** *approx-from-below-dense-linorder*:
**fixes** $x$::$'a$::{*dense-linorder*, *linorder-topology*, *first-countable-topology*}
**assumes** $x > y$
**shows** $\exists\, u.\ (\forall\, n.\ u\ n < x) \wedge (u \longrightarrow x)$
⟨*proof*⟩

## 98.9 Function limit at a point

**abbreviation** *LIM* :: ($'a$::*topological-space* $\Rightarrow$ $'b$::*topological-space*) $\Rightarrow$ $'a \Rightarrow$ $'b \Rightarrow$ *bool*
(‹(‹notation=‹infix LIM››(-)/ −(-)/→ (-))› [60, 0, 60] 60)
**where** $f\ {-a\rightarrow}\ L \equiv (f \longrightarrow L)\ (at\ a)$

**lemma** *tendsto-within-open*: $a \in S \Longrightarrow open\ S \Longrightarrow (f \longrightarrow l)\ (at\ a\ within\ S) \longleftrightarrow (f\ {-a\rightarrow}\ l)$
⟨*proof*⟩

**lemma** *tendsto-within-open-NO-MATCH*:
$a \in S \Longrightarrow$ *NO-MATCH UNIV* $S \Longrightarrow open\ S \Longrightarrow (f \longrightarrow l)(at\ a\ within\ S) \longleftrightarrow (f \longrightarrow l)(at\ a)$
**for** $f$ :: $'a$::*topological-space* $\Rightarrow$ $'b$::*topological-space*
⟨*proof*⟩

**lemma** *LIM-const-not-eq*[*tendsto-intros*]: $k \neq L \Longrightarrow \neg\ (\lambda x.\ k)\ {-a\rightarrow}\ L$
**for** $a$ :: $'a$::*perfect-space* **and** $k\ L$ :: $'b$::*t2-space*
⟨*proof*⟩

**lemmas** *LIM-not-zero* = *LIM-const-not-eq* [**where** $L = 0$]

**lemma** *LIM-const-eq*: $(\lambda x.\ k)\ {-a\rightarrow}\ L \Longrightarrow k = L$
**for** $a$ :: $'a$::*perfect-space* **and** $k\ L$ :: $'b$::*t2-space*
⟨*proof*⟩

**lemma** *LIM-unique*: $f$ $-a\rightarrow$ $L \Longrightarrow f$ $-a\rightarrow$ $M \Longrightarrow L = M$
**for** $a$ :: $'a$::*perfect-space* **and** $L$ $M$ :: $'b$::*t2-space*
⟨*proof*⟩

**lemma** *LIM-Uniq*: $\exists_{\leq 1}L$::$'b$::*t2-space*. $f$ $-a\rightarrow$ $L$
**for** $a$ :: $'a$::*perfect-space*
⟨*proof*⟩

Limits are equal for functions equal except at limit point.

**lemma** *LIM-equal*: $\forall x.\ x \neq a \longrightarrow f\,x = g\,x \Longrightarrow (f\ -a\rightarrow\ l) \longleftrightarrow (g\ -a\rightarrow\ l)$
⟨*proof*⟩

**lemma** *LIM-cong*: $a = b \Longrightarrow (\bigwedge x.\ x \neq b \Longrightarrow f\,x = g\,x) \Longrightarrow l = m \Longrightarrow (f\ -a\rightarrow\ l) \longleftrightarrow (g\ -b\rightarrow\ m)$
⟨*proof*⟩

**lemma** *tendsto-cong-limit*: $(f \longrightarrow l)\ F \Longrightarrow k = l \Longrightarrow (f \longrightarrow k)\ F$
⟨*proof*⟩

**lemma** *tendsto-at-iff-tendsto-nhds*: $g\ -l\rightarrow\ g\ l \longleftrightarrow (g \longrightarrow g\ l)\ (nhds\ l)$
⟨*proof*⟩

**lemma** *tendsto-compose*: $g\ -l\rightarrow\ g\ l \Longrightarrow (f \longrightarrow l)\ F \Longrightarrow ((\lambda x.\ g\ (f\,x)) \longrightarrow g\ l)\ F$
⟨*proof*⟩

**lemma** *tendsto-compose-eventually*:
$g\ -l\rightarrow\ m \Longrightarrow (f \longrightarrow l)\ F \Longrightarrow eventually\ (\lambda x.\ f\,x \neq l)\ F \Longrightarrow ((\lambda x.\ g\ (f\,x)) \longrightarrow m)\ F$
⟨*proof*⟩

**lemma** *LIM-compose-eventually*:
**assumes** $f$ $-a\rightarrow$ $b$
**and** $g$ $-b\rightarrow$ $c$
**and** *eventually* $(\lambda x.\ f\,x \neq b)\ (at\ a)$
**shows** $(\lambda x.\ g\ (f\,x))$ $-a\rightarrow$ $c$
⟨*proof*⟩

**lemma** *tendsto-compose-filtermap*: $((g \circ f) \longrightarrow T)\ F \longleftrightarrow (g \longrightarrow T)\ (filtermap\ f\ F)$
⟨*proof*⟩

**lemma** *tendsto-compose-at*:
**assumes** $f$: $(f \longrightarrow y)\ F$ **and** $g$: $(g \longrightarrow z)\ (at\ y)$ **and** $fg$: *eventually* $(\lambda w.\ f\,w = y \longrightarrow g\ y = z)\ F$
**shows** $((g \circ f) \longrightarrow z)\ F$
⟨*proof*⟩

**lemma** *tendsto-nhds-iff*: $(f \longrightarrow (c :: 'a :: t1\text{-}space))\ (nhds\ x) \longleftrightarrow f\ {-x}{\rightarrow}\ c \wedge f\ x = c$
⟨*proof*⟩

### 98.9.1 Relation of *LIM* and *LIMSEQ*

**lemma** (**in** *first-countable-topology*) *sequentially-imp-eventually-within*:
$(\forall f.\ (\forall n.\ f\ n \in s \wedge f\ n \neq a) \wedge f \longrightarrow a \longrightarrow eventually\ (\lambda n.\ P\ (f\ n))\ sequentially) \Longrightarrow$
$eventually\ P\ (at\ a\ within\ s)$
⟨*proof*⟩

**lemma** (**in** *first-countable-topology*) *sequentially-imp-eventually-at*:
$(\forall f.\ (\forall n.\ f\ n \neq a) \wedge f \longrightarrow a \longrightarrow eventually\ (\lambda n.\ P\ (f\ n))\ sequentially) \Longrightarrow eventually\ P\ (at\ a)$
⟨*proof*⟩

**lemma** *LIMSEQ-SEQ-conv*:
$(\forall S.\ (\forall n.\ S\ n \neq a) \wedge S \longrightarrow a \longrightarrow (\lambda n.\ X\ (S\ n)) \longrightarrow L) \longleftrightarrow X\ {-a}{\rightarrow}\ L$ (**is** *?lhs=?rhs*)
**for** $a :: 'a::first\text{-}countable\text{-}topology$ **and** $L :: 'b::topological\text{-}space$
⟨*proof*⟩

**lemma** *sequentially-imp-eventually-at-left*:
**fixes** $a :: 'a::\{linorder\text{-}topology,first\text{-}countable\text{-}topology\}$
**assumes** $b[simp]$: $b < a$
**and** $*$: $\bigwedge f.\ (\bigwedge n.\ b < f\ n) \Longrightarrow (\bigwedge n.\ f\ n < a) \Longrightarrow incseq\ f \Longrightarrow f \longrightarrow a \Longrightarrow$
$eventually\ (\lambda n.\ P\ (f\ n))\ sequentially$
**shows** $eventually\ P\ (at\text{-}left\ a)$
⟨*proof*⟩

**lemma** *tendsto-at-left-sequentially*:
**fixes** $a\ b :: 'b::\{linorder\text{-}topology,first\text{-}countable\text{-}topology\}$
**assumes** $b < a$
**assumes** $*$: $\bigwedge S.\ (\bigwedge n.\ S\ n < a) \Longrightarrow (\bigwedge n.\ b < S\ n) \Longrightarrow incseq\ S \Longrightarrow S \longrightarrow a \Longrightarrow$
$(\lambda n.\ X\ (S\ n)) \longrightarrow L$
**shows** $(X \longrightarrow L)\ (at\text{-}left\ a)$
⟨*proof*⟩

**lemma** *sequentially-imp-eventually-at-right*:
**fixes** $a\ b :: 'a::\{linorder\text{-}topology,first\text{-}countable\text{-}topology\}$
**assumes** $b[simp]$: $a < b$
**assumes** $*$: $\bigwedge f.\ (\bigwedge n.\ a < f\ n) \Longrightarrow (\bigwedge n.\ f\ n < b) \Longrightarrow decseq\ f \Longrightarrow f \longrightarrow a \Longrightarrow$
$eventually\ (\lambda n.\ P\ (f\ n))\ sequentially$
**shows** $eventually\ P\ (at\text{-}right\ a)$
⟨*proof*⟩

**lemma** *tendsto-at-right-sequentially*:
**fixes** $a$ :: - :: {*linorder-topology*, *first-countable-topology*}
**assumes** $a < b$
**and** $*$: $\bigwedge S.\ (\bigwedge n.\ a < S\ n) \Longrightarrow (\bigwedge n.\ S\ n < b) \Longrightarrow decseq\ S \Longrightarrow S \longrightarrow a \Longrightarrow$
$(\lambda n.\ X\ (S\ n)) \longrightarrow L$
**shows** $(X \longrightarrow L)$ (*at-right* $a$)
⟨*proof*⟩

## 98.10 Continuity

### 98.10.1 Continuity on a set

**definition** *continuous-on* :: $'a\ set \Rightarrow ('a::topological\text{-}space \Rightarrow 'b::topological\text{-}space) \Rightarrow bool$
**where** *continuous-on* $s\ f \longleftrightarrow (\forall x\in s.\ (f \longrightarrow f\ x)$ (*at* $x$ *within* $s$))

**lemma** *continuous-on-cong* [*cong*]:
$s = t \Longrightarrow (\bigwedge x.\ x \in t \Longrightarrow f\ x = g\ x) \Longrightarrow$ *continuous-on* $s\ f \longleftrightarrow$ *continuous-on* $t\ g$
⟨*proof*⟩

**lemma** *continuous-on-cong-simp*:
$s = t \Longrightarrow (\bigwedge x.\ x \in t =simp=> f\ x = g\ x) \Longrightarrow$ *continuous-on* $s\ f \longleftrightarrow$ *continuous-on* $t\ g$
⟨*proof*⟩

**lemma** *continuous-on-topological*:
*continuous-on* $s\ f \longleftrightarrow$
$(\forall x\in s.\ \forall B.\ open\ B \longrightarrow f\ x \in B \longrightarrow (\exists A.\ open\ A \wedge x \in A \wedge (\forall y\in s.\ y \in A \longrightarrow f\ y \in B)))$
⟨*proof*⟩

**lemma** *continuous-on-open-invariant*:
*continuous-on* $s\ f \longleftrightarrow (\forall B.\ open\ B \longrightarrow (\exists A.\ open\ A \wedge A \cap s = f -` B \cap s))$
⟨*proof*⟩

**lemma** *continuous-on-open-vimage*:
$open\ s \Longrightarrow$ *continuous-on* $s\ f \longleftrightarrow (\forall B.\ open\ B \longrightarrow open\ (f -` B \cap s))$
⟨*proof*⟩

**corollary** *continuous-imp-open-vimage*:
**assumes** *continuous-on* $s\ f$ *open* $s$ *open* $B\ f -` B \subseteq s$
**shows** *open* $(f -` B)$
⟨*proof*⟩

**corollary** *open-vimage*[*continuous-intros*]:
**assumes** *open* $s$
**and** *continuous-on UNIV* $f$
**shows** *open* $(f -` s)$

⟨*proof*⟩

**lemma** *continuous-on-closed-invariant*:
$continuous\text{-}on\ s\ f \longleftrightarrow (\forall B.\ closed\ B \longrightarrow (\exists A.\ closed\ A \wedge A \cap s = f -` B \cap s))$
⟨*proof*⟩

**lemma** *continuous-on-closed-vimage*:
$closed\ s \Longrightarrow continuous\text{-}on\ s\ f \longleftrightarrow (\forall B.\ closed\ B \longrightarrow closed\ (f -` B \cap s))$
⟨*proof*⟩

**corollary** *closed-vimage-Int*[*continuous-intros*]:
**assumes** *closed s*
**and** *continuous-on t f*
**and** *t*: *closed t*
**shows** $closed\ (f -` s \cap t)$
⟨*proof*⟩

**corollary** *closed-vimage*[*continuous-intros*]:
**assumes** *closed s*
**and** *continuous-on UNIV f*
**shows** $closed\ (f -` s)$
⟨*proof*⟩

**lemma** *continuous-on-empty* [*simp*]: $continuous\text{-}on\ \{\}\ f$
⟨*proof*⟩

**lemma** *continuous-on-sing* [*simp*]: $continuous\text{-}on\ \{x\}\ f$
⟨*proof*⟩

**lemma** *continuous-on-open-Union*:
$(\bigwedge s.\ s \in S \Longrightarrow open\ s) \Longrightarrow (\bigwedge s.\ s \in S \Longrightarrow continuous\text{-}on\ s\ f) \Longrightarrow continuous\text{-}on\ (\bigcup S)\ f$
⟨*proof*⟩

**lemma** *continuous-on-open-UN*:
$(\bigwedge s.\ s \in S \Longrightarrow open\ (A\ s)) \Longrightarrow (\bigwedge s.\ s \in S \Longrightarrow continuous\text{-}on\ (A\ s)\ f) \Longrightarrow continuous\text{-}on\ (\bigcup s{\in}S.\ A\ s)\ f$
⟨*proof*⟩

**lemma** *continuous-on-open-Un*:
$open\ s \Longrightarrow open\ t \Longrightarrow continuous\text{-}on\ s\ f \Longrightarrow continuous\text{-}on\ t\ f \Longrightarrow continuous\text{-}on\ (s \cup t)\ f$
⟨*proof*⟩

**lemma** *continuous-on-closed-Un*:
$closed\ s \Longrightarrow closed\ t \Longrightarrow continuous\text{-}on\ s\ f \Longrightarrow continuous\text{-}on\ t\ f \Longrightarrow continuous\text{-}on\ (s \cup t)\ f$
⟨*proof*⟩

**lemma** *continuous-on-closed-Union*:
  **assumes** *finite* $I$
    $\bigwedge i.\ i \in I \Longrightarrow$ *closed* $(U\ i)$
    $\bigwedge i.\ i \in I \Longrightarrow$ *continuous-on* $(U\ i)\ f$
  **shows** *continuous-on* $(\bigcup\ i \in I.\ U\ i)\ f$
  $\langle proof \rangle$

**lemma** *continuous-on-If*:
  **assumes** *closed*: *closed* $s$ *closed* $t$
    **and** *cont*: *continuous-on* $s\ f$ *continuous-on* $t\ g$
    **and** $P$: $\bigwedge x.\ x \in s \Longrightarrow \neg\ P\ x \Longrightarrow f\ x = g\ x\ \bigwedge x.\ x \in t \Longrightarrow P\ x \Longrightarrow f\ x = g\ x$
  **shows** *continuous-on* $(s \cup t)$ $(\lambda x.$ *if* $P\ x$ *then* $f\ x$ *else* $g\ x)$
    (**is** *continuous-on* - *?h*)
$\langle proof \rangle$

**lemma** *continuous-on-cases*:
  *closed* $s \Longrightarrow$ *closed* $t \Longrightarrow$ *continuous-on* $s\ f \Longrightarrow$ *continuous-on* $t\ g \Longrightarrow$
    $\forall x.\ (x{\in}s \wedge \neg\ P\ x) \vee (x \in t \wedge P\ x) \longrightarrow f\ x = g\ x \Longrightarrow$
    *continuous-on* $(s \cup t)$ $(\lambda x.$ *if* $P\ x$ *then* $f\ x$ *else* $g\ x)$
  $\langle proof \rangle$

**lemma** *continuous-on-id*[*continuous-intros*,*simp*]: *continuous-on* $s$ $(\lambda x.\ x)$
  $\langle proof \rangle$

**lemma** *continuous-on-id'*[*continuous-intros*,*simp*]: *continuous-on* $s$ *id*
  $\langle proof \rangle$

**lemma** *continuous-on-const*[*continuous-intros*,*simp*]: *continuous-on* $s$ $(\lambda x.\ c)$
  $\langle proof \rangle$

**lemma** *continuous-on-subset*: *continuous-on* $s\ f \Longrightarrow t \subseteq s \Longrightarrow$ *continuous-on* $t\ f$
  $\langle proof \rangle$

**lemma** *continuous-on-compose*[*continuous-intros*]:
  *continuous-on* $s\ f \Longrightarrow$ *continuous-on* $(f\ `\ s)\ g \Longrightarrow$ *continuous-on* $s\ (g \circ f)$
  $\langle proof \rangle$

**lemma** *continuous-on-compose2*:
  *continuous-on* $t\ g \Longrightarrow$ *continuous-on* $s\ f \Longrightarrow f\ `\ s \subseteq t \Longrightarrow$ *continuous-on* $s$ $(\lambda x.$
$g\ (f\ x))$
  $\langle proof \rangle$

**lemma** *continuous-on-generate-topology*:
  **assumes** $*$: *open* = *generate-topology* $X$
    **and** $**$: $\bigwedge B.\ B \in X \Longrightarrow \exists C.$ *open* $C \wedge C \cap A = f\ -`\ B \cap A$
  **shows** *continuous-on* $A\ f$
  $\langle proof \rangle$

**lemma** *continuous-onI-mono*:

**fixes** $f$ :: $'a$::*linorder-topology* $\Rightarrow$ $'b$::{*dense-order*,*linorder-topology*}
**assumes** *open* $(f`A)$
**and** *mono*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$
**shows** *continuous-on* $A$ $f$
⟨*proof*⟩

**lemma** *continuous-on-IccI*:
$[\![(f \longrightarrow f\ a)$ (*at-right* $a$);
$(f \longrightarrow f\ b)$ (*at-left* $b$);
$(\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow f\ {-x\rightarrow}\ f\ x)$; $a < b]\!] \Longrightarrow$
*continuous-on* $\{a\ ..\ b\}$ $f$
**for** $a$::$'a$::*linorder-topology*
⟨*proof*⟩

**lemma**
**fixes** $a$ $b$::$'a$::*linorder-topology*
**assumes** *continuous-on* $\{a\ ..\ b\}$ $f$ $a < b$
**shows** *continuous-on-Icc-at-rightD*: $(f \longrightarrow f\ a)$ (*at-right* $a$)
**and** *continuous-on-Icc-at-leftD*: $(f \longrightarrow f\ b)$ (*at-left* $b$)
⟨*proof*⟩

**lemma** *continuous-on-discrete* [*simp*]:
*continuous-on* $A$ ($f$ :: $'a$ :: *discrete-topology* $\Rightarrow$ -)
⟨*proof*⟩

**lemma** *continuous-on-of-nat* [*continuous-intros*]:
**assumes** *continuous-on* $A$ $f$
**shows** *continuous-on* $A$ ($\lambda n.$ *of-nat* $(f\ n)$)
⟨*proof*⟩

**lemma** *continuous-on-of-int* [*continuous-intros*]:
**assumes** *continuous-on* $A$ $f$
**shows** *continuous-on* $A$ ($\lambda n.$ *of-int* $(f\ n)$)
⟨*proof*⟩

### 98.10.2 Continuity at a point

**definition** *continuous* :: $'a$::*t2-space filter* $\Rightarrow$ ($'a \Rightarrow 'b$::*topological-space*) $\Rightarrow$ *bool*
**where** *continuous* $F\ f \longleftrightarrow (f \longrightarrow f$ (*Lim* $F$ $(\lambda x.\ x)))$ $F$

**lemma** *continuous-bot*[*continuous-intros*, *simp*]: *continuous bot* $f$
⟨*proof*⟩

**lemma** *continuous-trivial-limit*: *trivial-limit net* $\Longrightarrow$ *continuous net* $f$
⟨*proof*⟩

**lemma** *continuous-within*: *continuous* (*at* $x$ *within* $s$) $f \longleftrightarrow (f \longrightarrow f\ x)$ (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *continuous-within-topological*:
$continuous\ (at\ x\ within\ s)\ f \longleftrightarrow$
$(\forall B.\ open\ B \longrightarrow f\ x \in B \longrightarrow (\exists A.\ open\ A \wedge x \in A \wedge (\forall y{\in}s.\ y \in A \longrightarrow f\ y \in B)))$
⟨*proof*⟩

**lemma** *continuous-within-compose*[*continuous-intros*]:
$continuous\ (at\ x\ within\ s)\ f \Longrightarrow continuous\ (at\ (f\ x)\ within\ f\ `\ s)\ g \Longrightarrow$
$continuous\ (at\ x\ within\ s)\ (g \circ f)$
⟨*proof*⟩

**lemma** *continuous-within-compose2*:
$continuous\ (at\ x\ within\ s)\ f \Longrightarrow continuous\ (at\ (f\ x)\ within\ f\ `\ s)\ g \Longrightarrow$
$continuous\ (at\ x\ within\ s)\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-at*: $continuous\ (at\ x)\ f \longleftrightarrow f\ {-x\rightarrow}\ f\ x$
⟨*proof*⟩

**lemma** *continuous-ident*[*continuous-intros*, *simp*]: $continuous\ (at\ x\ within\ S)\ (\lambda x.\ x)$
⟨*proof*⟩

**lemma** *continuous-id*[*continuous-intros*, *simp*]: $continuous\ (at\ x\ within\ S)\ id$
⟨*proof*⟩

**lemma** *continuous-const*[*continuous-intros*, *simp*]: $continuous\ F\ (\lambda x.\ c)$
⟨*proof*⟩

**lemma** *continuous-on-eq-continuous-within*:
$continuous\text{-}on\ s\ f \longleftrightarrow (\forall x{\in}s.\ continuous\ (at\ x\ within\ s)\ f)$
⟨*proof*⟩

**lemma** *continuous-discrete* [*simp*]:
$continuous\ (at\ x\ within\ A)\ (f :: 'a :: discrete\text{-}topology \Rightarrow \text{-})$
⟨*proof*⟩

Continuity in terms of open preimages.

**lemma** *continuous-at-open*:
$continuous\ (at\ x)\ f \longleftrightarrow (\forall t.\ open\ t \wedge f\ x \in t \longrightarrow (\exists S.\ open\ S \wedge x \in S \wedge (\forall x' \in S.\ (f\ x') \in t)))$
⟨*proof*⟩

**lemma** *continuous-imp-tendsto*:
**assumes** $continuous\ (at\ x0)\ f$ **and** $x \longlonglongrightarrow x0$
**shows** $(f \circ x) \longlonglongrightarrow (f\ x0)$
⟨*proof*⟩

**abbreviation** *isCont* :: ($'a$::*t2-space* $\Rightarrow$ $'b$::*topological-space*) $\Rightarrow$ $'a \Rightarrow$ *bool*
**where** *isCont f a* $\equiv$ *continuous* (*at a*) *f*

**lemma** *isCont-def*: *isCont f a* $\longleftrightarrow$ *f* $-a\rightarrow$ *f a*
⟨*proof*⟩

**lemma** *isContD*: *isCont f x* $\Longrightarrow$ *f* $-x\rightarrow$ *f x*
⟨*proof*⟩

**lemma** *isCont-cong*:
**assumes** *eventually* ($\lambda x$. *f x* = *g x*) (*nhds x*)
**shows** *isCont f x* $\longleftrightarrow$ *isCont g x*
⟨*proof*⟩

**lemma** *continuous-at-imp-continuous-at-within*: *isCont f x* $\Longrightarrow$ *continuous* (*at x within s*) *f*
⟨*proof*⟩

**lemma** *continuous-on-eq-continuous-at*: *open s* $\Longrightarrow$ *continuous-on s f* $\longleftrightarrow$ ($\forall x \in s$. *isCont f x*)
⟨*proof*⟩

**lemma** *continuous-within-open*: $a \in A \Longrightarrow$ *open A* $\Longrightarrow$ *continuous* (*at a within A*) *f* $\longleftrightarrow$ *isCont f a*
⟨*proof*⟩

**lemma** *continuous-at-imp-continuous-on*: $\forall x \in s$. *isCont f x* $\Longrightarrow$ *continuous-on s f*
⟨*proof*⟩

**lemma** *isCont-o2*: *isCont f a* $\Longrightarrow$ *isCont g* (*f a*) $\Longrightarrow$ *isCont* ($\lambda x$. *g* (*f x*)) *a*
⟨*proof*⟩

**lemma** *continuous-at-compose*[*continuous-intros*]: *isCont f a* $\Longrightarrow$ *isCont g* (*f a*) $\Longrightarrow$ *isCont* ($g \circ f$) *a*
⟨*proof*⟩

**lemma** *isCont-tendsto-compose*: *isCont g l* $\Longrightarrow$ (*f* $\longrightarrow$ *l*) *F* $\Longrightarrow$ (($\lambda x$. *g* (*f x*)) $\longrightarrow$ *g l*) *F*
⟨*proof*⟩

**lemma** *continuous-on-tendsto-compose*:
**assumes** *f-cont*: *continuous-on s f*
**and** *g*: (*g* $\longrightarrow$ *l*) *F*
**and** *l*: $l \in s$
**and** *ev*: $\forall_F x$ *in F*. *g x* $\in$ *s*
**shows** (($\lambda x$. *f* (*g x*)) $\longrightarrow$ *f l*) *F*
⟨*proof*⟩

**lemma** *continuous-within-compose3*:

$isCont\ g\ (f\ x) \Longrightarrow continuous\ (at\ x\ within\ s)\ f \Longrightarrow continuous\ (at\ x\ within\ s)\ (\lambda x.\ g\ (f\ x))$
⟨*proof*⟩

**lemma** *at-within-isCont-imp-nhds*:
**fixes** $f$:: $'a$:: {*t2-space*,*perfect-space*} ⇒ $'b$:: *t2-space*
**assumes** $\forall_F\ w\ in\ at\ z.\ f\ w = g\ w\ isCont\ f\ z\ isCont\ g\ z$
**shows** $\forall_F\ w\ in\ nhds\ z.\ f\ w = g\ w$
⟨*proof*⟩

**lemma** *filtermap-nhds-open-map'*:
**assumes** *cont*: *isCont f a*
**and** *open A a* ∈ *A*
**and** *open-map*: $\bigwedge S.\ open\ S \Longrightarrow S \subseteq A \Longrightarrow open\ (f\ `\ S)$
**shows** *filtermap f* (*nhds a*) = *nhds* (*f a*)
⟨*proof*⟩

**lemma** *filtermap-nhds-open-map*:
**assumes** *cont*: *isCont f a*
**and** *open-map*: $\bigwedge S.\ open\ S \Longrightarrow open\ (f`S)$
**shows** *filtermap f* (*nhds a*) = *nhds* (*f a*)
⟨*proof*⟩

**lemma** *continuous-at-split*:
$continuous\ (at\ x)\ f \longleftrightarrow continuous\ (at\text{-}left\ x)\ f \land continuous\ (at\text{-}right\ x)\ f$
**for** $x$ :: $'a$::*linorder-topology*
⟨*proof*⟩

**lemma** *continuous-on-max* [*continuous-intros*]:
**fixes** *f g* :: $'a$::*topological-space* ⇒ $'b$::*linorder-topology*
**shows** $continuous\text{-}on\ A\ f \Longrightarrow continuous\text{-}on\ A\ g \Longrightarrow continuous\text{-}on\ A\ (\lambda x.\ max\ (f\ x)\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-on-min* [*continuous-intros*]:
**fixes** *f g* :: $'a$::*topological-space* ⇒ $'b$::*linorder-topology*
**shows** $continuous\text{-}on\ A\ f \Longrightarrow continuous\text{-}on\ A\ g \Longrightarrow continuous\text{-}on\ A\ (\lambda x.\ min\ (f\ x)\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-max* [*continuous-intros*]:
**fixes** *f* :: $'a$::*t2-space* ⇒ $'b$::*linorder-topology*
**shows** $[\![continuous\ F\ f;\ continuous\ F\ g]\!] \Longrightarrow continuous\ F\ (\lambda x.\ (max\ (f\ x)\ (g\ x)))$
⟨*proof*⟩

**lemma** *continuous-min* [*continuous-intros*]:
**fixes** *f* :: $'a$::*t2-space* ⇒ $'b$::*linorder-topology*
**shows** $[\![continuous\ F\ f;\ continuous\ F\ g]\!] \Longrightarrow continuous\ F\ (\lambda x.\ (min\ (f\ x)\ (g\ x)))$

⟨*proof*⟩

The following open/closed Collect lemmas are ported from Sébastien Gouëzel's *Ergodic-Theory*.

**lemma** *open-Collect-neq*:
**fixes** $f\ g :: {'a}::\mathit{topological\text{-}space} \Rightarrow {'b}::\mathit{t2\text{-}space}$
**assumes** $f$: *continuous-on UNIV f* **and** $g$: *continuous-on UNIV g*
**shows** *open* $\{x.\ f\ x \neq g\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-eq*:
**fixes** $f\ g :: {'a}::\mathit{topological\text{-}space} \Rightarrow {'b}::\mathit{t2\text{-}space}$
**assumes** $f$: *continuous-on UNIV f* **and** $g$: *continuous-on UNIV g*
**shows** *closed* $\{x.\ f\ x = g\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-less*:
**fixes** $f\ g :: {'a}::\mathit{topological\text{-}space} \Rightarrow {'b}::\mathit{linorder\text{-}topology}$
**assumes** $f$: *continuous-on UNIV f* **and** $g$: *continuous-on UNIV g*
**shows** *open* $\{x.\ f\ x < g\ x\}$
⟨*proof*⟩

**lemma** *closed-Collect-le*:
**fixes** $f\ g :: {'a} :: \mathit{topological\text{-}space} \Rightarrow {'b}::\mathit{linorder\text{-}topology}$
**assumes** $f$: *continuous-on UNIV f*
**and** $g$: *continuous-on UNIV g*
**shows** *closed* $\{x.\ f\ x \leq g\ x\}$
⟨*proof*⟩

### 98.10.3 Open-cover compactness

**context** *topological-space*
**begin**

**definition** *compact* :: *'a set* ⇒ *bool* **where**
*compact-eq-Heine-Borel*:
$\mathit{compact}\ S \longleftrightarrow (\forall\ C.\ (\forall\ c{\in}C.\ \mathit{open}\ c) \wedge S \subseteq \bigcup C \longrightarrow (\exists\ D{\subseteq}C.\ \mathit{finite}\ D \wedge S \subseteq \bigcup D))$

**lemma** *compactI*:
**assumes** $\bigwedge C.\ \forall\ t{\in}C.\ \mathit{open}\ t \Longrightarrow s \subseteq \bigcup C \Longrightarrow \exists\ C'.\ C' \subseteq C \wedge \mathit{finite}\ C' \wedge s \subseteq \bigcup C'$
**shows** *compact s*
⟨*proof*⟩

**lemma** *compact-empty*[*simp*]: *compact* {}
⟨*proof*⟩

**lemma** *compactE*:

**assumes** *compact* $S$ $S \subseteq \bigcup\mathcal{T}$ $\bigwedge B.\ B \in \mathcal{T} \Longrightarrow$ *open* $B$
**obtains** $\mathcal{T}'$ **where** $\mathcal{T}' \subseteq \mathcal{T}$ *finite* $\mathcal{T}'$ $S \subseteq \bigcup\mathcal{T}'$
⟨*proof*⟩

**lemma** *compactE-image*:
**assumes** *compact* $S$
**and** *opn*: $\bigwedge T.\ T \in C \Longrightarrow$ *open* $(f\ T)$
**and** $S$: $S \subseteq (\bigcup c{\in}C.\ f\ c)$
**obtains** $C'$ **where** $C' \subseteq C$ **and** *finite* $C'$ **and** $S \subseteq (\bigcup c{\in}C'.\ f\ c)$
⟨*proof*⟩

**lemma** *compact-Int-closed* [*intro*]:
**assumes** *compact* $S$
**and** *closed* $T$
**shows** *compact* $(S \cap T)$
⟨*proof*⟩

**lemma** *compact-diff*: $[\![$*compact* $S$; *open* $T]\!] \Longrightarrow$ *compact*$(S - T)$
⟨*proof*⟩

**lemma** *inj-setminus*: *inj-on uminus* $(A::'a\ set\ set)$
⟨*proof*⟩

## 98.11 Finite intersection property

**lemma** *compact-fip*:
*compact* $U \longleftrightarrow$
$(\forall A.\ (\forall a{\in}A.\ closed\ a) \longrightarrow (\forall B \subseteq A.\ finite\ B \longrightarrow U \cap \bigcap B \neq \{\}) \longrightarrow U \cap \bigcap A \neq \{\})$
(**is** - $\longleftrightarrow$ *?R*)
⟨*proof*⟩

**lemma** *compact-imp-fip*:
**assumes** *compact* $S$
**and** $\bigwedge T.\ T \in F \Longrightarrow$ *closed* $T$
**and** $\bigwedge F'.$ *finite* $F' \Longrightarrow F' \subseteq F \Longrightarrow S \cap (\bigcap F') \neq \{\}$
**shows** $S \cap (\bigcap F) \neq \{\}$
⟨*proof*⟩

**lemma** *compact-imp-fip-image*:
**assumes** *compact* $s$
**and** $P$: $\bigwedge i.\ i \in I \Longrightarrow$ *closed* $(f\ i)$
**and** $Q$: $\bigwedge I'.$ *finite* $I' \Longrightarrow I' \subseteq I \Longrightarrow (s \cap (\bigcap i{\in}I'.\ f\ i) \neq \{\})$
**shows** $s \cap (\bigcap i{\in}I.\ f\ i) \neq \{\}$
⟨*proof*⟩

**end**

**lemma** (**in** *t2-space*) *compact-imp-closed*:

**assumes** *compact s*
**shows** *closed s*
⟨*proof*⟩

**lemma** *compact-continuous-image*:
**assumes** $f$: *continuous-on s f*
**and** $s$: *compact s*
**shows** *compact* ($f$ ' $s$)
⟨*proof*⟩

**lemma** *continuous-on-inv*:
**fixes** $f$ :: $'a$::*topological-space* ⇒ $'b$::*t2-space*
**assumes** *continuous-on s f*
**and** *compact s*
**and** $\forall x \in s.\ g\ (f\ x) = x$
**shows** *continuous-on* ($f$ ' $s$) $g$
⟨*proof*⟩

**lemma** *continuous-on-inv-into*:
**fixes** $f$ :: $'a$::*topological-space* ⇒ $'b$::*t2-space*
**assumes** $s$: *continuous-on s f compact s*
**and** $f$: *inj-on f s*
**shows** *continuous-on* ($f$ ' $s$) (*the-inv-into s f*)
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *compact-attains-sup*:
**assumes** *compact S* $S \neq \{\}$
**shows** $\exists s \in S.\ \forall t \in S.\ t \leq s$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *compact-attains-inf*:
**assumes** *compact S* $S \neq \{\}$
**shows** $\exists s \in S.\ \forall t \in S.\ s \leq t$
⟨*proof*⟩

**lemma** *continuous-attains-sup*:
**fixes** $f$ :: $'a$::*topological-space* ⇒ $'b$::*linorder-topology*
**shows** *compact s* $\Longrightarrow s \neq \{\} \Longrightarrow$ *continuous-on s f* $\Longrightarrow (\exists x \in s.\ \forall y \in s.\ f\ y \leq f\ x)$
⟨*proof*⟩

**lemma** *continuous-attains-inf*:
**fixes** $f$ :: $'a$::*topological-space* ⇒ $'b$::*linorder-topology*
**shows** *compact s* $\Longrightarrow s \neq \{\} \Longrightarrow$ *continuous-on s f* $\Longrightarrow (\exists x \in s.\ \forall y \in s.\ f\ x \leq f\ y)$
⟨*proof*⟩

## 98.12 Connectedness

**context** *topological-space*

**begin**

**definition** *connected* $S \longleftrightarrow$
$\neg\ (\exists\, A\ B.\ open\ A \wedge open\ B \wedge S \subseteq A \cup B \wedge A \cap B \cap S = \{\} \wedge A \cap S \neq \{\} \wedge B \cap S \neq \{\})$

**lemma** *connectedI*:
$(\bigwedge A\ B.\ open\ A \Longrightarrow open\ B \Longrightarrow A \cap U \neq \{\} \Longrightarrow B \cap U \neq \{\} \Longrightarrow A \cap B \cap U = \{\} \Longrightarrow U \subseteq A \cup B \Longrightarrow False)$
$\Longrightarrow connected\ U$
⟨*proof*⟩

**lemma** *connected-empty* [*simp*]: *connected* $\{\}$
⟨*proof*⟩

**lemma** *connected-sing* [*simp*]: *connected* $\{x\}$
⟨*proof*⟩

**lemma** *connectedD*:
$connected\ A \Longrightarrow open\ U \Longrightarrow open\ V \Longrightarrow U \cap V \cap A = \{\} \Longrightarrow A \subseteq U \cup V \Longrightarrow U \cap A = \{\} \vee V \cap A = \{\}$
⟨*proof*⟩

**end**

**lemma** *connected-closed*:
$connected\ s \longleftrightarrow$
$\neg\ (\exists\, A\ B.\ closed\ A \wedge closed\ B \wedge s \subseteq A \cup B \wedge A \cap B \cap s = \{\} \wedge A \cap s \neq \{\} \wedge B \cap s \neq \{\})$
⟨*proof*⟩

**lemma** *connected-closedD*:
$[\![connected\ s;\ A \cap B \cap s = \{\};\ s \subseteq A \cup B;\ closed\ A;\ closed\ B]\!] \Longrightarrow A \cap s = \{\} \vee B \cap s = \{\}$
⟨*proof*⟩

**lemma** *connected-Union*:
**assumes** *cs*: $\bigwedge s.\ s \in S \Longrightarrow connected\ s$
**and** *ne*: $\bigcap S \neq \{\}$
**shows** $connected(\bigcup S)$
⟨*proof*⟩

**lemma** *connected-Un*: $connected\ s \Longrightarrow connected\ t \Longrightarrow s \cap t \neq \{\} \Longrightarrow connected\ (s \cup t)$
⟨*proof*⟩

**lemma** *connected-diff-open-from-closed*:
**assumes** *st*: $s \subseteq t$
**and** *tu*: $t \subseteq u$

**and** *s*: *open s*
**and** *t*: *closed t*
**and** *u*: *connected u*
**and** *ts*: *connected* $(t - s)$
**shows** *connected*$(u - s)$
⟨*proof*⟩

**lemma** *connected-iff-const*:
**fixes** $S$ :: $'a$::*topological-space set*
**shows** *connected* $S \longleftrightarrow (\forall P::'a \Rightarrow bool.\ continuous\text{-}on\ S\ P \longrightarrow (\exists c.\ \forall s{\in}S.\ P\ s = c))$
⟨*proof*⟩

**lemma** *connectedD-const*: *connected* $S \Longrightarrow$ *continuous-on* $S\ P \Longrightarrow \exists c.\ \forall s{\in}S.\ P\ s = c$
**for** $P$ :: $'a$::*topological-space* $\Rightarrow$ *bool*
⟨*proof*⟩

**lemma** *connectedI-const*:
$(\bigwedge P::'a::topological\text{-}space \Rightarrow bool.\ continuous\text{-}on\ S\ P \Longrightarrow \exists c.\ \forall s{\in}S.\ P\ s = c) \Longrightarrow$ *connected* $S$
⟨*proof*⟩

**lemma** *connected-local-const*:
**assumes** *connected* $A$ $a \in A$ $b \in A$
**and** $*$: $\forall a{\in}A.$ *eventually* $(\lambda b.\ f\ a = f\ b)$ (*at* $a$ *within* $A$)
**shows** $f\ a = f\ b$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *connectedD-interval*:
**assumes** *connected* $U$
**and** *xy*: $x \in U$ $y \in U$
**and** $x \le z$ $z \le y$
**shows** $z \in U$
⟨*proof*⟩

**lemma** (**in** *linorder-topology*) *not-in-connected-cases*:
**assumes** *conn*: *connected* $S$
**assumes** *nbdd*: $x \notin S$
**assumes** *ne*: $S \neq \{\}$
**obtains** *bdd-above* $S$ $\bigwedge y.\ y \in S \Longrightarrow x \ge y$ | *bdd-below* $S$ $\bigwedge y.\ y \in S \Longrightarrow x \le y$
⟨*proof*⟩

**lemma** *connected-continuous-image*:
**assumes** $*$: *continuous-on* $s\ f$
**and** *connected* $s$
**shows** *connected* $(f\ `\ s)$
⟨*proof*⟩

**lemma** *connected-Un-UN*:
  **assumes** *connected* $A$ $\bigwedge X.\ X \in B \Longrightarrow$ *connected* $X$ $\bigwedge X.\ X \in B \Longrightarrow A \cap X \neq \{\}$
  **shows** *connected* $(A \cup \bigcup B)$
⟨*proof*⟩

# 99 Linear Continuum Topologies

**class** *linear-continuum-topology* = *linorder-topology* + *linear-continuum*
**begin**

**lemma** *Inf-notin-open*:
  **assumes** *A*: *open* $A$
    **and** *bnd*: $\forall a \in A.\ x < a$
  **shows** *Inf* $A \notin A$
⟨*proof*⟩

**lemma** *Sup-notin-open*:
  **assumes** *A*: *open* $A$
    **and** *bnd*: $\forall a \in A.\ a < x$
  **shows** *Sup* $A \notin A$
⟨*proof*⟩

**end**

**instance** *linear-continuum-topology* ⊆ *perfect-space*
⟨*proof*⟩

**lemma** *connectedI-interval*:
  **fixes** $U$ :: *'a* :: *linear-continuum-topology set*
  **assumes** $*$: $\bigwedge x\ y\ z.\ x \in U \Longrightarrow y \in U \Longrightarrow x \leq z \Longrightarrow z \leq y \Longrightarrow z \in U$
  **shows** *connected* $U$
⟨*proof*⟩

**lemma** *connected-iff-interval*: *connected* $U \longleftrightarrow (\forall x \in U.\ \forall y \in U.\ \forall z.\ x \leq z \longrightarrow z \leq y \longrightarrow z \in U)$
  **for** $U$ :: *'a::linear-continuum-topology set*
  ⟨*proof*⟩

**lemma** *connected-UNIV*[*simp*]: *connected* (*UNIV*::*'a::linear-continuum-topology set*)
  ⟨*proof*⟩

**lemma** *connected-Ioi*[*simp*]: *connected* $\{a<..\}$
  **for** $a$ :: *'a::linear-continuum-topology*
  ⟨*proof*⟩

**lemma** *connected-Ici*[*simp*]: *connected* $\{a..\}$
  **for** $a$ :: *'a::linear-continuum-topology*
  ⟨*proof*⟩

**lemma** *connected-Iio*[*simp*]: *connected* $\{..<a\}$
**for** $a$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-Iic*[*simp*]: *connected* $\{..a\}$
**for** $a$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-Ioo*[*simp*]: *connected* $\{a<..<b\}$
**for** $a$ $b$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-Ioc*[*simp*]: *connected* $\{a<..b\}$
**for** $a$ $b$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-Ico*[*simp*]: *connected* $\{a..<b\}$
**for** $a$ $b$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-Icc*[*simp*]: *connected* $\{a..b\}$
**for** $a$ $b$ :: *'a::linear-continuum-topology*
⟨*proof*⟩

**lemma** *connected-contains-Ioo*:
**fixes** $A$ :: *'a :: linorder-topology set*
**assumes** *connected* $A$ $a \in A$ $b \in A$ **shows** $\{a <..< b\} \subseteq A$
⟨*proof*⟩

**lemma** *connected-contains-Icc*:
**fixes** $A$ :: *'a::linorder-topology set*
**assumes** *connected* $A$ $a \in A$ $b \in A$
**shows** $\{a..b\} \subseteq A$
⟨*proof*⟩

## 99.1 Intermediate Value Theorem

**lemma** *IVT'*:
**fixes** $f$ :: *'a::linear-continuum-topology* ⇒ *'b::linorder-topology*
**assumes** $y$: $f\ a \le y$ $y \le f\ b$ $a \le b$
**and** $*$: *continuous-on* $\{a .. b\}$ $f$
**shows** $\exists x.\ a \le x \wedge x \le b \wedge f\ x = y$
⟨*proof*⟩

**lemma** *IVT2'*:
**fixes** $f$ :: *'a :: linear-continuum-topology* ⇒ *'b :: linorder-topology*
**assumes** $y$: $f\ b \le y$ $y \le f\ a$ $a \le b$
**and** $*$: *continuous-on* $\{a .. b\}$ $f$

**shows** $\exists x.\ a \leq x \wedge x \leq b \wedge f\,x = y$
$\langle proof \rangle$

**lemma** *IVT*:
**fixes** $f$ :: $'a$::*linear-continuum-topology* $\Rightarrow$ $'b$::*linorder-topology*
**shows** $f\,a \leq y \Longrightarrow y \leq f\,b \Longrightarrow a \leq b \Longrightarrow (\forall x.\ a \leq x \wedge x \leq b \longrightarrow isCont\ f\,x)$
$\Longrightarrow$
$\exists x.\ a \leq x \wedge x \leq b \wedge f\,x = y$
$\langle proof \rangle$

**lemma** *IVT2*:
**fixes** $f$ :: $'a$::*linear-continuum-topology* $\Rightarrow$ $'b$::*linorder-topology*
**shows** $f\,b \leq y \Longrightarrow y \leq f\,a \Longrightarrow a \leq b \Longrightarrow (\forall x.\ a \leq x \wedge x \leq b \longrightarrow isCont\ f\,x)$
$\Longrightarrow$
$\exists x.\ a \leq x \wedge x \leq b \wedge f\,x = y$
$\langle proof \rangle$

**lemma** *continuous-inj-imp-mono*:
**fixes** $f$ :: $'a$::*linear-continuum-topology* $\Rightarrow$ $'b$::*linorder-topology*
**assumes** $x$: $a < x$ $x < b$
**and** *cont*: *continuous-on* $\{a..b\}$ $f$
**and** *inj*: *inj-on* $f$ $\{a..b\}$
**shows** $(f\,a < f\,x \wedge f\,x < f\,b) \vee (f\,b < f\,x \wedge f\,x < f\,a)$
$\langle proof \rangle$

**lemma** *continuous-at-Sup-mono*:
**fixes** $f$ :: $'a$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$ $\Rightarrow$
$'b$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$
**assumes** *mono* $f$
**and** *cont*: *continuous* (*at-left* (*Sup* $S$)) $f$
**and** $S$: $S \neq \{\}$ *bdd-above* $S$
**shows** $f$ (*Sup* $S$) = (*SUP* $s \in S.\ f\,s$)
$\langle proof \rangle$

**lemma** *continuous-at-Sup-antimono*:
**fixes** $f$ :: $'a$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$ $\Rightarrow$
$'b$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$
**assumes** *antimono* $f$
**and** *cont*: *continuous* (*at-left* (*Sup* $S$)) $f$
**and** $S$: $S \neq \{\}$ *bdd-above* $S$
**shows** $f$ (*Sup* $S$) = (*INF* $s \in S.\ f\,s$)
$\langle proof \rangle$

**lemma** *continuous-at-Inf-mono*:
**fixes** $f$ :: $'a$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$ $\Rightarrow$
$'b$::$\{$*linorder-topology*,*conditionally-complete-linorder*$\}$
**assumes** *mono* $f$
**and** *cont*: *continuous* (*at-right* (*Inf* $S$)) $f$
**and** $S$: $S \neq \{\}$ *bdd-below* $S$

**shows** $f$ $(Inf\ S) = (INF\ s{\in}S.\ f\ s)$
$\langle proof \rangle$

**lemma** *continuous-at-Inf-antimono*:
**fixes** $f$ :: $'a$::{*linorder-topology*,*conditionally-complete-linorder*} $\Rightarrow$
$'b$::{*linorder-topology*,*conditionally-complete-linorder*}
**assumes** *antimono f*
**and** *cont*: *continuous* (*at-right* (*Inf S*)) *f*
**and** *S*: $S \neq \{\}$ *bdd-below S*
**shows** $f$ $(Inf\ S) = (SUP\ s{\in}S.\ f\ s)$
$\langle proof \rangle$

## 99.2 Uniform spaces

**class** *uniformity* =
**fixes** *uniformity* :: $('a \times 'a)$ *filter*
**begin**

**abbreviation** *uniformity-on* :: $'a$ *set* $\Rightarrow$ $('a \times 'a)$ *filter*
**where** *uniformity-on s* $\equiv$ *inf uniformity* (*principal* $(s{\times}s)$)

**end**

**lemma** *uniformity-Abort*:
*uniformity* =
*Filter.abstract-filter* ($\lambda u.$ *Code.abort* (*STR ''uniformity is not executable''*) ($\lambda u.$ *uniformity*))
$\langle proof \rangle$

**class** *open-uniformity* = *open* + *uniformity* +
**assumes** *open-uniformity*:
$\bigwedge U.$ *open* $U \longleftrightarrow (\forall x{\in}U.$ *eventually* $(\lambda(x', y).\ x' = x \longrightarrow y \in U)$ *uniformity*)
**begin**

**subclass** *topological-space*
$\langle proof \rangle$

**end**

**class** *uniform-space* = *open-uniformity* +
**assumes** *uniformity-refl*: *eventually E uniformity* $\Longrightarrow E\ (x, x)$
**and** *uniformity-sym*: *eventually E uniformity* $\Longrightarrow$ *eventually* $(\lambda(x, y).\ E\ (y, x))$ *uniformity*
**and** *uniformity-trans*:
*eventually E uniformity* $\Longrightarrow$
$\exists D.$ *eventually D uniformity* $\wedge (\forall x\ y\ z.\ D\ (x, y) \longrightarrow D\ (y, z) \longrightarrow E\ (x, z))$
**begin**

**lemma** *uniformity-bot*: *uniformity* $\neq$ *bot*

⟨*proof*⟩

**lemma** *uniformity-trans'*:
*eventually E uniformity* $\Longrightarrow$
*eventually* $(\lambda((x, y), (y', z)).\ y = y' \longrightarrow E\ (x, z))$ (*uniformity* $\times_F$ *uniformity*)
⟨*proof*⟩

**lemma** *uniformity-transE*:
**assumes** *eventually E uniformity*
**obtains** *D* **where** *eventually D uniformity* $\bigwedge x\ y\ z.\ D\ (x, y) \Longrightarrow D\ (y, z) \Longrightarrow E\ (x, z)$
⟨*proof*⟩

**lemma** *eventually-nhds-uniformity*:
*eventually P* (*nhds x*) $\longleftrightarrow$ *eventually* $(\lambda(x', y).\ x' = x \longrightarrow P\ y)$ *uniformity*
(**is** - $\longleftrightarrow$ *?N P x*)
⟨*proof*⟩

### 99.2.1 Totally bounded sets

**definition** *totally-bounded* :: *'a set* $\Rightarrow$ *bool*
**where** *totally-bounded S* $\longleftrightarrow$
$(\forall E.$ *eventually E uniformity* $\longrightarrow (\exists X.$ *finite* $X \wedge (\forall s\in S.\ \exists x\in X.\ E\ (x, s))))$

**lemma** *totally-bounded-empty*[*iff*]: *totally-bounded* {}
⟨*proof*⟩

**lemma** *totally-bounded-subset*: *totally-bounded S* $\Longrightarrow T \subseteq S \Longrightarrow$ *totally-bounded T*
⟨*proof*⟩

**lemma** *totally-bounded-Union*[*intro*]:
**assumes** *M*: *finite M* $\bigwedge S.\ S \in M \Longrightarrow$ *totally-bounded S*
**shows** *totally-bounded* $(\bigcup M)$
⟨*proof*⟩

### 99.2.2 Cauchy filter

**definition** *cauchy-filter* :: *'a filter* $\Rightarrow$ *bool*
**where** *cauchy-filter F* $\longleftrightarrow F \times_F F \leq$ *uniformity*

**definition** *Cauchy* :: $(nat \Rightarrow 'a) \Rightarrow bool$
**where** *Cauchy-uniform*: *Cauchy X* = *cauchy-filter* (*filtermap X sequentially*)

**lemma** *Cauchy-uniform-iff*:
*Cauchy X* $\longleftrightarrow (\forall P.$ *eventually P uniformity* $\longrightarrow (\exists N.\ \forall n\geq N.\ \forall m\geq N.\ P\ (X\ n, X\ m)))$
⟨*proof*⟩

**lemma** *nhds-imp-cauchy-filter*:

**assumes** $*$: $F \leq nhds\ x$
**shows** *cauchy-filter F*
⟨*proof*⟩

**lemma** *LIMSEQ-imp-Cauchy*: $X \longrightarrow x \Longrightarrow Cauchy\ X$
⟨*proof*⟩

**lemma** *Cauchy-subseq-Cauchy*:
**assumes** *Cauchy X strict-mono f*
**shows** *Cauchy* $(X \circ f)$
⟨*proof*⟩

**lemma** *convergent-Cauchy*: *convergent X* $\Longrightarrow$ *Cauchy X*
⟨*proof*⟩

**definition** *complete* :: $'a\ set \Rightarrow bool$
**where** *complete-uniform*: *complete S* $\longleftrightarrow$
$(\forall F \leq principal\ S.\ F \neq bot \longrightarrow cauchy\text{-}filter\ F \longrightarrow (\exists x{\in}S.\ F \leq nhds\ x))$

**lemma** (**in** *uniform-space*) *cauchy-filter-complete-converges*:
**assumes** *cauchy-filter F complete A* $F \leq principal\ A\ F \neq bot$
**shows** $\exists c.\ F \leq nhds\ c$
⟨*proof*⟩

**end**

### 99.2.3 Uniformly continuous functions

**definition** *uniformly-continuous-on* :: $'a\ set \Rightarrow ('a::uniform\text{-}space \Rightarrow 'b::uniform\text{-}space)$ $\Rightarrow bool$
**where** *uniformly-continuous-on-uniformity*: *uniformly-continuous-on s f* $\longleftrightarrow$
$(LIM\ (x,\ y)\ (uniformity\text{-}on\ s).\ (f\ x,\ f\ y) :> uniformity)$

**lemma** *uniformly-continuous-onD*:
*uniformly-continuous-on s f* $\Longrightarrow$ *eventually E uniformity* $\Longrightarrow$
*eventually* $(\lambda(x,\ y).\ x \in s \longrightarrow y \in s \longrightarrow E\ (f\ x,\ f\ y))$ *uniformity*
⟨*proof*⟩

**lemma** *uniformly-continuous-on-const*[*continuous-intros*]: *uniformly-continuous-on* $s\ (\lambda x.\ c)$
⟨*proof*⟩

**lemma** *uniformly-continuous-on-id*[*continuous-intros*]: *uniformly-continuous-on s* $(\lambda x.\ x)$
⟨*proof*⟩

**lemma** *uniformly-continuous-on-compose*:
*uniformly-continuous-on s g* $\Longrightarrow$ *uniformly-continuous-on* $(g`s)\ f \Longrightarrow$
*uniformly-continuous-on s* $(\lambda x.\ f\ (g\ x))$

⟨*proof*⟩

**lemma** *uniformly-continuous-imp-continuous*:
**assumes** *f*: *uniformly-continuous-on s f*
**shows** *continuous-on s f*
⟨*proof*⟩

# 100 Product Topology

## 100.1 Product is a topological space

**instantiation** *prod* :: (*topological-space*, *topological-space*) *topological-space*
**begin**

**definition** *open-prod-def*[*code del*]:
$open\ (S :: ('a \times 'b)\ set) \longleftrightarrow$
$(\forall x{\in}S.\ \exists A\ B.\ open\ A \wedge open\ B \wedge x \in A \times B \wedge A \times B \subseteq S)$

**lemma** *open-prod-elim*:
**assumes** *open S* **and** $x \in S$
**obtains** *A B* **where** *open A* **and** *open B* **and** $x \in A \times B$ **and** $A \times B \subseteq S$
⟨*proof*⟩

**lemma** *open-prod-intro*:
**assumes** $\bigwedge x.\ x \in S \Longrightarrow \exists A\ B.\ open\ A \wedge open\ B \wedge x \in A \times B \wedge A \times B \subseteq S$
**shows** *open S*
⟨*proof*⟩

**instance**
⟨*proof*⟩

**end**

**declare** [[*code abort*: $open :: ('a::topological\text{-}space \times 'b::topological\text{-}space)\ set \Rightarrow bool$]]

**lemma** *open-Times*: $open\ S \Longrightarrow open\ T \Longrightarrow open\ (S \times T)$
⟨*proof*⟩

**lemma** *fst-vimage-eq-Times*: $fst\ -`\ S = S \times UNIV$
⟨*proof*⟩

**lemma** *snd-vimage-eq-Times*: $snd\ -`\ S = UNIV \times S$
⟨*proof*⟩

**lemma** *open-vimage-fst*: $open\ S \Longrightarrow open\ (fst\ -`\ S)$
⟨*proof*⟩

**lemma** *open-vimage-snd*: $open\ S \Longrightarrow open\ (snd\ -`\ S)$

⟨*proof*⟩

**lemma** *closed-vimage-fst*: $closed\ S \Longrightarrow closed\ (fst\ -`\ S)$
⟨*proof*⟩

**lemma** *closed-vimage-snd*: $closed\ S \Longrightarrow closed\ (snd\ -`\ S)$
⟨*proof*⟩

**lemma** *closed-Times*: $closed\ S \Longrightarrow closed\ T \Longrightarrow closed\ (S \times T)$
⟨*proof*⟩

**lemma** *subset-fst-imageI*: $A \times B \subseteq S \Longrightarrow y \in B \Longrightarrow A \subseteq fst\ `\ S$
⟨*proof*⟩

**lemma** *subset-snd-imageI*: $A \times B \subseteq S \Longrightarrow x \in A \Longrightarrow B \subseteq snd\ `\ S$
⟨*proof*⟩

**lemma** *open-image-fst*:
**assumes** $open\ S$
**shows** $open\ (fst\ `\ S)$
⟨*proof*⟩

**lemma** *open-image-snd*:
**assumes** $open\ S$
**shows** $open\ (snd\ `\ S)$
⟨*proof*⟩

**lemma** *nhds-prod*: $nhds\ (a,\ b) = nhds\ a \times_F nhds\ b$
⟨*proof*⟩

### 100.1.1 Continuity of operations

**lemma** *tendsto-fst* [*tendsto-intros*]:
**assumes** $(f \longrightarrow a)\ F$
**shows** $((\lambda x.\ fst\ (f\ x)) \longrightarrow fst\ a)\ F$
⟨*proof*⟩

**lemma** *tendsto-snd* [*tendsto-intros*]:
**assumes** $(f \longrightarrow a)\ F$
**shows** $((\lambda x.\ snd\ (f\ x)) \longrightarrow snd\ a)\ F$
⟨*proof*⟩

**lemma** *tendsto-Pair* [*tendsto-intros*]:
**assumes** $(f \longrightarrow a)\ F$ **and** $(g \longrightarrow b)\ F$
**shows** $((\lambda x.\ (f\ x,\ g\ x)) \longrightarrow (a,\ b))\ F$
⟨*proof*⟩

**lemma** *continuous-fst*[*continuous-intros*]: $continuous\ F\ f \Longrightarrow continuous\ F\ (\lambda x.\ fst\ (f\ x))$

⟨*proof*⟩

**lemma** *continuous-snd*[*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* ($\lambda x.$ *snd* (*f x*))
⟨*proof*⟩

**lemma** *continuous-Pair*[*continuous-intros*]:
*continuous F f* $\Longrightarrow$ *continuous F g* $\Longrightarrow$ *continuous F* ($\lambda x.$ (*f x*, *g x*))
⟨*proof*⟩

**lemma** *continuous-on-fst*[*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s* ($\lambda x.$ *fst* (*f x*))
⟨*proof*⟩

**lemma** *continuous-on-snd*[*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s* ($\lambda x.$ *snd* (*f x*))
⟨*proof*⟩

**lemma** *continuous-on-Pair*[*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s g* $\Longrightarrow$ *continuous-on s* ($\lambda x.$ (*f x*, *g x*))
⟨*proof*⟩

**lemma** *continuous-on-swap*[*continuous-intros*]: *continuous-on A prod.swap*
⟨*proof*⟩

**lemma** *continuous-on-swap-args*:
**assumes** *continuous-on* ($A \times B$) ($\lambda(x,y).$ *d x y*)
**shows** *continuous-on* ($B \times A$) ($\lambda(x,y).$ *d y x*)
⟨*proof*⟩

**lemma** *isCont-fst* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* ($\lambda x.$ *fst* (*f x*)) *a*
⟨*proof*⟩

**lemma** *isCont-snd* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* ($\lambda x.$ *snd* (*f x*)) *a*
⟨*proof*⟩

**lemma** *isCont-Pair* [*simp*]: $[\![$*isCont f a*; *isCont g a*$]\!]$ $\Longrightarrow$ *isCont* ($\lambda x.$ (*f x*, *g x*)) *a*
⟨*proof*⟩

**lemma** *continuous-on-compose-Pair*:
**assumes** *f*: *continuous-on* (*Sigma A B*) ($\lambda(a, b).$ *f a b*)
**assumes** *g*: *continuous-on C g*
**assumes** *h*: *continuous-on C h*
**assumes** *subset*: $\bigwedge c.\ c \in C \Longrightarrow g\ c \in A$ $\bigwedge c.\ c \in C \Longrightarrow h\ c \in B\ (g\ c)$
**shows** *continuous-on C* ($\lambda c.$ *f* (*g c*) (*h c*))
⟨*proof*⟩

### 100.1.2 Connectedness of products

**proposition** *connected-Times*:
**assumes** $S$: *connected* $S$ **and** $T$: *connected* $T$
**shows** *connected* $(S \times T)$
⟨*proof*⟩

**corollary** *connected-Times-eq* [*simp*]:
*connected* $(S \times T) \longleftrightarrow S = \{\} \vee T = \{\} \vee$ *connected* $S \wedge$ *connected* $T$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

### 100.1.3 Separation axioms

**instance** *prod* :: (*t0-space*, *t0-space*) *t0-space*
⟨*proof*⟩

**instance** *prod* :: (*t1-space*, *t1-space*) *t1-space*
⟨*proof*⟩

**instance** *prod* :: (*t2-space*, *t2-space*) *t2-space*
⟨*proof*⟩

**lemma** *isCont-swap*[*continuous-intros*]: *isCont prod.swap a*
⟨*proof*⟩

**lemma** *open-diagonal-complement*:
*open* $\{(x,y) \mid x\ y.\ x \neq (y::('a::t2\text{-}space))\}$
⟨*proof*⟩

**lemma** *closed-diagonal*:
*closed* $\{y.\ \exists\ x::('a::t2\text{-}space).\ y = (x,x)\}$
⟨*proof*⟩

**lemma** *open-superdiagonal*:
*open* $\{(x,y) \mid x\ y.\ x > (y::'a::\{linorder\text{-}topology\})\}$
⟨*proof*⟩

**lemma** *closed-subdiagonal*:
*closed* $\{(x,y) \mid x\ y.\ x \le (y::'a::\{linorder\text{-}topology\})\}$
⟨*proof*⟩

**lemma** *open-subdiagonal*:
*open* $\{(x,y) \mid x\ y.\ x < (y::'a::\{linorder\text{-}topology\})\}$
⟨*proof*⟩

**lemma** *closed-superdiagonal*:
*closed* $\{(x,y) \mid x\ y.\ x \ge (y::('a::\{linorder\text{-}topology\}))\}$
⟨*proof*⟩

**end**

**theory** *Hull*
**imports** *Main*
**begin**

## 100.2 A generic notion of the convex, affine, conic hull, or closed "hull".

**definition** $hull :: ('a\ set \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow 'a\ set$ (**infixl** ‹hull› 75)
**where** $S\ hull\ s = \bigcap\{t.\ S\ t \wedge s \subseteq t\}$

**lemma** *hull-same*: $S\ s \Longrightarrow S\ hull\ s = s$
⟨*proof*⟩

**lemma** *hull-in*: $(\bigwedge T.\ Ball\ T\ S \Longrightarrow S\ (\bigcap T)) \Longrightarrow S\ (S\ hull\ s)$
⟨*proof*⟩

**lemma** *hull-eq*: $(\bigwedge T.\ Ball\ T\ S \Longrightarrow S\ (\bigcap T)) \Longrightarrow (S\ hull\ s) = s \longleftrightarrow S\ s$
⟨*proof*⟩

**lemma** *hull-hull* [*simp*]: $S\ hull\ (S\ hull\ s) = S\ hull\ s$
⟨*proof*⟩

**lemma** *hull-subset*[*intro*]: $s \subseteq (S\ hull\ s)$
⟨*proof*⟩

**lemma** *hull-mono*: $s \subseteq t \Longrightarrow (S\ hull\ s) \subseteq (S\ hull\ t)$
⟨*proof*⟩

**lemma** *hull-antimono*: $\forall x.\ S\ x \longrightarrow T\ x \Longrightarrow (T\ hull\ s) \subseteq (S\ hull\ s)$
⟨*proof*⟩

**lemma** *hull-minimal*: $s \subseteq t \Longrightarrow S\ t \Longrightarrow (S\ hull\ s) \subseteq t$
⟨*proof*⟩

**lemma** *subset-hull*: $S\ t \Longrightarrow S\ hull\ s \subseteq t \longleftrightarrow s \subseteq t$
⟨*proof*⟩

**lemma** *hull-UNIV* [*simp*]: $S\ hull\ UNIV = UNIV$
⟨*proof*⟩

**lemma** *hull-unique*: $s \subseteq t \Longrightarrow S\ t \Longrightarrow (\bigwedge t'.\ s \subseteq t' \Longrightarrow S\ t' \Longrightarrow t \subseteq t') \Longrightarrow (S\ hull\ s = t)$
⟨*proof*⟩

**lemma** *hull-induct*: $[\![a \in Q\ hull\ S;\ \bigwedge x.\ x\in S \Longrightarrow P\ x;\ Q\ \{x.\ P\ x\}]\!] \Longrightarrow P\ a$
⟨*proof*⟩

**lemma** *hull-inc*: $x \in S \Longrightarrow x \in P$ *hull* $S$
⟨*proof*⟩

**lemma** *hull-Un-subset*: $(S$ *hull* $s) \cup (S$ *hull* $t) \subseteq (S$ *hull* $(s \cup t))$
⟨*proof*⟩

**lemma** *hull-Un*:
**assumes** $T$: $\bigwedge T$. *Ball* $T$ $S \Longrightarrow S$ $(\bigcap T)$
**shows** $S$ *hull* $(s \cup t) = S$ *hull* $(S$ *hull* $s \cup S$ *hull* $t)$
⟨*proof*⟩

**lemma** *hull-Un-left*: $P$ *hull* $(S \cup T) = P$ *hull* $(P$ *hull* $S \cup T)$
⟨*proof*⟩

**lemma** *hull-Un-right*: $P$ *hull* $(S \cup T) = P$ *hull* $(S \cup P$ *hull* $T)$
⟨*proof*⟩

**lemma** *hull-insert*:
$P$ *hull* (*insert* $a$ $S$) $= P$ *hull* (*insert* $a$ $(P$ *hull* $S))$
⟨*proof*⟩

**lemma** *hull-redundant-eq*: $a \in (S$ *hull* $s) \longleftrightarrow S$ *hull* (*insert* $a$ $s$) $= S$ *hull* $s$
⟨*proof*⟩

**lemma** *hull-redundant*: $a \in (S$ *hull* $s) \Longrightarrow S$ *hull* (*insert* $a$ $s$) $= S$ *hull* $s$
⟨*proof*⟩

**end**

# 101 Modules

Bases of a linear algebra based on modules (i.e. vector spaces of rings).

**theory** *Modules*
**imports** *Hull*
**begin**

## 101.1 Locale for additive functions

**locale** *additive* =
**fixes** $f$ :: $'a$::*ab-group-add* $\Rightarrow$ $'b$::*ab-group-add*
**assumes** *add*: $f\ (x + y) = f\ x + f\ y$
**begin**

**lemma** *zero*: $f\ 0 = 0$
⟨*proof*⟩

**lemma** *minus*: $f\ (-\ x) = -\ f\ x$

⟨*proof*⟩

**lemma** *diff*: $f\ (x - y) = f\ x - f\ y$
⟨*proof*⟩

**lemma** *sum*: $f\ (sum\ g\ A) = (\sum x{\in}A.\ f\ (g\ x))$
⟨*proof*⟩

**end**

Modules form the central spaces in linear algebra. They are a generalization from vector spaces by replacing the scalar field by a scalar ring.

**locale** *module* =
  **fixes** *scale* :: $'a$::*comm-ring-1* ⇒ $'b$::*ab-group-add* ⇒ $'b$ (**infixr** ‹$*s$› *75*)
  **assumes** *scale-right-distrib* [*algebra-simps*, *algebra-split-simps*]:
    $a *s (x + y) = a *s x + a *s y$
  **and** *scale-left-distrib* [*algebra-simps*, *algebra-split-simps*]:
    $(a + b) *s x = a *s x + b *s x$
  **and** *scale-scale* [*simp*]: $a *s (b *s x) = (a * b) *s x$
  **and** *scale-one* [*simp*]: $1 *s x = x$
**begin**

**lemma** *scale-left-commute*: $a *s (b *s x) = b *s (a *s x)$
⟨*proof*⟩

**lemma** *scale-zero-left* [*simp*]: $0 *s x = 0$
  **and** *scale-minus-left* [*simp*]: $(- a) *s x = - (a *s x)$
  **and** *scale-left-diff-distrib* [*algebra-simps*, *algebra-split-simps*]:
    $(a - b) *s x = a *s x - b *s x$
  **and** *scale-sum-left*: $(sum\ f\ A) *s x = (\sum a{\in}A.\ (f\ a) *s x)$
⟨*proof*⟩

**lemma** *scale-zero-right* [*simp*]: $a *s 0 = 0$
  **and** *scale-minus-right* [*simp*]: $a *s (- x) = - (a *s x)$
  **and** *scale-right-diff-distrib* [*algebra-simps*, *algebra-split-simps*]:
    $a *s (x - y) = a *s x - a *s y$
  **and** *scale-sum-right*: $a *s (sum\ f\ A) = (\sum x{\in}A.\ a *s (f\ x))$
⟨*proof*⟩

**lemma** *sum-constant-scale*: $(\sum x{\in}A.\ y) = scale\ (of\text{-}nat\ (card\ A))\ y$
⟨*proof*⟩

**end**

⟨*ML*⟩

**context** *module*
**begin**

**lemma** [*field-simps*, *field-split-simps*]:
**shows** *scale-left-distrib-NO-MATCH*: $NO\text{-}MATCH\ (x\ div\ y)\ c \Longrightarrow (a + b) *s\ x = a *s\ x + b *s\ x$
**and** *scale-right-distrib-NO-MATCH*: $NO\text{-}MATCH\ (x\ div\ y)\ a \Longrightarrow a *s\ (x + y) = a *s\ x + a *s\ y$
**and** *scale-left-diff-distrib-NO-MATCH*: $NO\text{-}MATCH\ (x\ div\ y)\ c \Longrightarrow (a - b) *s\ x = a *s\ x - b *s\ x$
**and** *scale-right-diff-distrib-NO-MATCH*: $NO\text{-}MATCH\ (x\ div\ y)\ a \Longrightarrow a *s\ (x - y) = a *s\ x - a *s\ y$
⟨*proof*⟩

**end**

⟨*ML*⟩

# 102 Subspace

**context** *module*
**begin**

**definition** *subspace* :: $'b\ set \Rightarrow bool$
**where** $subspace\ S \longleftrightarrow 0 \in S \wedge (\forall x{\in}S.\ \forall y{\in}S.\ x + y \in S) \wedge (\forall c.\ \forall x{\in}S.\ c *s\ x \in S)$

**lemma** *subspaceI*:
$0 \in S \Longrightarrow (\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow x + y \in S) \Longrightarrow (\bigwedge c\ x.\ x \in S \Longrightarrow c *s\ x \in S) \Longrightarrow subspace\ S$
⟨*proof*⟩

**lemma** *subspace-UNIV*[*simp*]: *subspace UNIV*
⟨*proof*⟩

**lemma** *subspace-single-0*[*simp*]: $subspace\ \{0\}$
⟨*proof*⟩

**lemma** *subspace-0*: $subspace\ S \Longrightarrow 0 \in S$
⟨*proof*⟩

**lemma** *subspace-add*: $subspace\ S \Longrightarrow x \in S \Longrightarrow y \in S \Longrightarrow x + y \in S$
⟨*proof*⟩

**lemma** *subspace-scale*: $subspace\ S \Longrightarrow x \in S \Longrightarrow c *s\ x \in S$
⟨*proof*⟩

**lemma** *subspace-neg*: $subspace\ S \Longrightarrow x \in S \Longrightarrow - x \in S$
⟨*proof*⟩

**lemma** *subspace-diff*: $subspace\ S \Longrightarrow x \in S \Longrightarrow y \in S \Longrightarrow x - y \in S$
⟨*proof*⟩

**lemma** *subspace-sum*: $subspace\ A \Longrightarrow (\bigwedge x.\ x \in B \Longrightarrow f\, x \in A) \Longrightarrow sum\ f\ B \in A$
⟨*proof*⟩

**lemma** *subspace-Int*: $(\bigwedge i.\ i \in I \Longrightarrow subspace\ (s\ i)) \Longrightarrow subspace\ (\bigcap i{\in}I.\ s\ i)$
⟨*proof*⟩

**lemma** *subspace-Inter*: $\forall\, s \in f.\ subspace\ s \Longrightarrow subspace\ (\bigcap f)$
⟨*proof*⟩

**lemma** *subspace-inter*: $subspace\ A \Longrightarrow subspace\ B \Longrightarrow subspace\ (A \cap B)$
⟨*proof*⟩

# 103 Span: subspace generated by a set

**definition** $span :: 'b\ set \Rightarrow 'b\ set$
**where** *span-explicit*: $span\ b = \{(\sum a{\in}t.\ r\ a *s\ \ a) \mid t\ r.\ finite\ t \wedge t \subseteq b\}$

**lemma** *span-explicit'*:
$span\ b = \{(\sum v \mid f\ v \neq 0.\ f\ v *s\ v) \mid f.\ finite\ \{v.\ f\ v \neq 0\} \wedge (\forall\, v.\ f\ v \neq 0 \longrightarrow v \in b)\}$
⟨*proof*⟩

**lemma** *span-alt*:
$span\ B = \{(\sum x \mid f\, x \neq 0.\ f\, x *s\ x) \mid f.\ \{x.\ f\, x \neq 0\} \subseteq B \wedge finite\ \{x.\ f\, x \neq 0\}\}$
⟨*proof*⟩

**lemma** *span-finite*:
**assumes** *fS*: *finite S*
**shows** $span\ S = range\ (\lambda u.\ \sum v{\in}S.\ u\ v *s\ v)$
⟨*proof*⟩

**lemma** *span-induct-alt* [*consumes 1*, *case-names base step*, *induct set*: *span*]:
**assumes** *x*: $x \in span\ S$
**assumes** *h0*: *h 0* **and** *hS*: $\bigwedge c\ x\ y.\ x \in S \Longrightarrow h\ y \Longrightarrow h\ (c *s\ x + y)$
**shows** *h x*
⟨*proof*⟩

**lemma** *span-mono*: $A \subseteq B \Longrightarrow span\ A \subseteq span\ B$
⟨*proof*⟩

**lemma** *span-base*: $a \in S \Longrightarrow a \in span\ S$
⟨*proof*⟩

**lemma** *span-superset*: $S \subseteq span\ S$
⟨*proof*⟩

**lemma** *span-zero*: $0 \in span\ S$
⟨*proof*⟩

**lemma** *span-UNIV*[*simp*]: *span UNIV* = *UNIV*
⟨*proof*⟩

**lemma** *span-add*: $x \in span\ S \Longrightarrow y \in span\ S \Longrightarrow x + y \in span\ S$
⟨*proof*⟩

**lemma** *span-scale*: $x \in span\ S \Longrightarrow c *s\ x \in span\ S$
⟨*proof*⟩

**lemma** *subspace-span* [*iff*]: *subspace* (*span S*)
⟨*proof*⟩

**lemma** *span-neg*: $x \in span\ S \Longrightarrow -\ x \in span\ S$
⟨*proof*⟩

**lemma** *span-diff*: $x \in span\ S \Longrightarrow y \in span\ S \Longrightarrow x - y \in span\ S$
⟨*proof*⟩

**lemma** *span-sum*: $(\bigwedge x.\ x \in A \Longrightarrow f\ x \in span\ S) \Longrightarrow sum\ f\ A \in span\ S$
⟨*proof*⟩

**lemma** *span-minimal*: $S \subseteq T \Longrightarrow subspace\ T \Longrightarrow span\ S \subseteq T$
⟨*proof*⟩

**lemma** *span-def*: *span S* = *subspace hull S*
⟨*proof*⟩

**lemma** *span-unique*:
$S \subseteq T \Longrightarrow subspace\ T \Longrightarrow (\bigwedge T'.\ S \subseteq T' \Longrightarrow subspace\ T' \Longrightarrow T \subseteq T') \Longrightarrow span\ S = T$
⟨*proof*⟩

**lemma** *span-subspace-induct*[*consumes 2*]:
**assumes** *x*: $x \in span\ S$
**and** *P*: *subspace P*
**and** *SP*: $\bigwedge x.\ x \in S \Longrightarrow x \in P$
**shows** $x \in P$
⟨*proof*⟩

**lemma** (**in** *module*) *span-induct*[*consumes 1*, *case-names base step*, *induct set*: *span*]:
**assumes** *x*: $x \in span\ S$
**and** *P*: *subspace* (*Collect P*)
**and** *SP*: $\bigwedge x.\ x \in S \Longrightarrow P\ x$
**shows** *P x*
⟨*proof*⟩

**lemma** *span-empty*[*simp*]: *span* {} = {*0*}

⟨proof⟩

**lemma** *span-subspace*: $A \subseteq B \Longrightarrow B \subseteq span\ A \Longrightarrow subspace\ B \Longrightarrow span\ A = B$
⟨proof⟩

**lemma** *span-span*: $span\ (span\ A) = span\ A$
⟨proof⟩

**lemma** *span-add-eq*: **assumes** $x$: $x \in span\ S$ **shows** $x + y \in span\ S \longleftrightarrow y \in span\ S$
⟨proof⟩

**lemma** *span-add-eq2*: **assumes** $y$: $y \in span\ S$ **shows** $x + y \in span\ S \longleftrightarrow x \in span\ S$
⟨proof⟩

**lemma** *span-singleton*: $span\ \{x\} = range\ (\lambda k.\ k *s\ x)$
⟨proof⟩

**lemma** *span-Un*: $span\ (S \cup T) = \{x + y \mid x\ y.\ x \in span\ S \wedge y \in span\ T\}$
⟨proof⟩

**lemma** *span-insert*: $span\ (insert\ a\ S) = \{x.\ \exists\, k.\ (x - k *s\ a) \in span\ S\}$
⟨proof⟩

**lemma** *span-breakdown*:
**assumes** $bS$: $b \in S$
**and** $aS$: $a \in span\ S$
**shows** $\exists\, k.\ a - k *s\ b \in span\ (S - \{b\})$
⟨proof⟩

**lemma** *span-breakdown-eq*: $x \in span\ (insert\ a\ S) \longleftrightarrow (\exists\, k.\ x - k *s\ a \in span\ S)$
⟨proof⟩

**lemmas** *span-clauses* = *span-base span-zero span-add span-scale*

**lemma** *span-eq-iff* [*simp*]: $span\ s = s \longleftrightarrow subspace\ s$
⟨proof⟩

**lemma** *span-eq*: $span\ S = span\ T \longleftrightarrow S \subseteq span\ T \wedge T \subseteq span\ S$
⟨proof⟩

**lemma** *eq-span-insert-eq*:
**assumes** $(x - y) \in span\ S$
**shows** $span(insert\ x\ S) = span(insert\ y\ S)$
⟨proof⟩

## 104 Dependent and independent sets

**definition** *dependent* :: $'b$ *set* $\Rightarrow$ *bool*
**where** *dependent-explicit*: *dependent* $s \longleftrightarrow (\exists\, t\ u.\ \mathit{finite}\ t \wedge t \subseteq s \wedge (\sum v{\in}t.\ u\ v *s\ v) = 0 \wedge (\exists\, v{\in}t.\ u\ v \neq 0))$

**abbreviation** *independent* $s \equiv \neg$ *dependent* $s$

**lemma** *dependent-mono*: *dependent* $B \Longrightarrow B \subseteq A \Longrightarrow$ *dependent* $A$
⟨*proof*⟩

**lemma** *independent-mono*: *independent* $A \Longrightarrow B \subseteq A \Longrightarrow$ *independent* $B$
⟨*proof*⟩

**lemma** *dependent-zero*: $0 \in A \Longrightarrow$ *dependent* $A$
⟨*proof*⟩

**lemma** *independent-empty*[*intro*]: *independent* $\{\}$
⟨*proof*⟩

**lemma** *independent-explicit-module*:
*independent* $s \longleftrightarrow (\forall\, t\ u\ v.\ \mathit{finite}\ t \longrightarrow t \subseteq s \longrightarrow (\sum v{\in}t.\ u\ v *s\ v) = 0 \longrightarrow v \in t \longrightarrow u\ v = 0)$
⟨*proof*⟩

**lemma** *independentD*: *independent* $s \Longrightarrow \mathit{finite}\ t \Longrightarrow t \subseteq s \Longrightarrow (\sum v{\in}t.\ u\ v *s\ v) = 0 \Longrightarrow v \in t \Longrightarrow u\ v = 0$
⟨*proof*⟩

**lemma** *independent-Union-directed*:
**assumes** *directed*: $\bigwedge c\ d.\ c \in C \Longrightarrow d \in C \Longrightarrow c \subseteq d \vee d \subseteq c$
**assumes** *indep*: $\bigwedge c.\ c \in C \Longrightarrow$ *independent* $c$
**shows** *independent* $(\bigcup C)$
⟨*proof*⟩

**lemma** *dependent-finite*:
**assumes** *finite* $S$
**shows** *dependent* $S \longleftrightarrow (\exists\, u.\ (\exists\, v \in S.\ u\ v \neq 0) \wedge (\sum v{\in}S.\ u\ v *s\ v) = 0)$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *dependent-alt*:
*dependent* $B \longleftrightarrow$
$(\exists\, X.\ \mathit{finite}\ \{x.\ X\ x \neq 0\} \wedge \{x.\ X\ x \neq 0\} \subseteq B \wedge (\sum x|X\ x \neq 0.\ X\ x *s\ x) = 0 \wedge (\exists\, x.\ X\ x \neq 0))$
⟨*proof*⟩

**lemma** *independent-alt*:
*independent* $B \longleftrightarrow$

$(\forall X.\ \mathit{finite}\ \{x.\ X\ x \neq 0\} \longrightarrow \{x.\ X\ x \neq 0\} \subseteq B \longrightarrow (\sum x|X\ x \neq 0.\ X\ x *s\ x) = 0 \longrightarrow (\forall x.\ X\ x = 0))$
⟨*proof*⟩

**lemma** *independentD-alt*:
$\mathit{independent}\ B \Longrightarrow \mathit{finite}\ \{x.\ X\ x \neq 0\} \Longrightarrow \{x.\ X\ x \neq 0\} \subseteq B \Longrightarrow (\sum x|X\ x \neq 0.\ X\ x *s\ x) = 0 \Longrightarrow X\ x = 0$
⟨*proof*⟩

**lemma** *independentD-unique*:
**assumes** $B$: $\mathit{independent}\ B$
**and** $X$: $\mathit{finite}\ \{x.\ X\ x \neq 0\}\ \{x.\ X\ x \neq 0\} \subseteq B$
**and** $Y$: $\mathit{finite}\ \{x.\ Y\ x \neq 0\}\ \{x.\ Y\ x \neq 0\} \subseteq B$
**and** $(\sum x \mid X\ x \neq 0.\ X\ x *s\ x) = (\sum x|\ Y\ x \neq 0.\ Y\ x *s\ x)$
**shows** $X = Y$
⟨*proof*⟩

# 105 Representation of a vector on a specific basis

**definition** *representation* :: $'b\ set \Rightarrow 'b \Rightarrow 'b \Rightarrow 'a$
**where** $\mathit{representation}\ \mathit{basis}\ v =$
$(\mathit{if}\ \mathit{independent}\ \mathit{basis} \wedge v \in \mathit{span}\ \mathit{basis}\ \mathit{then}$
$\mathit{SOME}\ f.\ (\forall v.\ f\ v \neq 0 \longrightarrow v \in \mathit{basis}) \wedge \mathit{finite}\ \{v.\ f\ v \neq 0\} \wedge (\sum v{\in}\{v.\ f\ v \neq 0\}.\ f\ v *s\ v) = v$
$\mathit{else}\ (\lambda b.\ 0))$

**lemma** *unique-representation*:
**assumes** *basis*: $\mathit{independent}\ \mathit{basis}$
**and** *in-basis*: $\bigwedge v.\ f\ v \neq 0 \Longrightarrow v \in \mathit{basis}\ \bigwedge v.\ g\ v \neq 0 \Longrightarrow v \in \mathit{basis}$
**and** [*simp*]: $\mathit{finite}\ \{v.\ f\ v \neq 0\}\ \mathit{finite}\ \{v.\ g\ v \neq 0\}$
**and** *eq*: $(\sum v{\in}\{v.\ f\ v \neq 0\}.\ f\ v *s\ v) = (\sum v{\in}\{v.\ g\ v \neq 0\}.\ g\ v *s\ v)$
**shows** $f = g$
⟨*proof*⟩

**lemma**
**shows** *representation-ne-zero*: $\bigwedge b.\ \mathit{representation}\ \mathit{basis}\ v\ b \neq 0 \Longrightarrow b \in \mathit{basis}$
**and** *finite-representation*: $\mathit{finite}\ \{b.\ \mathit{representation}\ \mathit{basis}\ v\ b \neq 0\}$
**and** *sum-nonzero-representation-eq*:
$\mathit{independent}\ \mathit{basis} \Longrightarrow v \in \mathit{span}\ \mathit{basis} \Longrightarrow (\sum b \mid \mathit{representation}\ \mathit{basis}\ v\ b \neq 0.\ \mathit{representation}\ \mathit{basis}\ v\ b *s\ b) = v$
⟨*proof*⟩

**lemma** *sum-representation-eq*:
$(\sum b{\in}B.\ \mathit{representation}\ \mathit{basis}\ v\ b *s\ b) = v$
**if** $\mathit{independent}\ \mathit{basis}\ v \in \mathit{span}\ \mathit{basis}\ \mathit{finite}\ B\ \mathit{basis} \subseteq B$
⟨*proof*⟩

**lemma** *representation-eqI*:
**assumes** *basis*: $\mathit{independent}\ \mathit{basis}$ **and** $b$: $v \in \mathit{span}\ \mathit{basis}$

**and** *ne-zero*: $\bigwedge b.\ f\ b \neq 0 \Longrightarrow b \in basis$
**and** *finite*: $finite\ \{b.\ f\ b \neq 0\}$
**and** *eq*: $(\sum b \mid f\ b \neq 0.\ f\ b *s\ b) = v$
**shows** $representation\ basis\ v = f$
⟨*proof*⟩

**lemma** *representation-basis*:
**assumes** *basis*: $independent\ basis$ **and** *b*: $b \in basis$
**shows** $representation\ basis\ b = (\lambda v.\ if\ v = b\ then\ 1\ else\ 0)$
⟨*proof*⟩

**lemma** *representation-zero*: $representation\ basis\ 0 = (\lambda b.\ 0)$
⟨*proof*⟩

**lemma** *representation-diff*:
**assumes** *basis*: $independent\ basis$ **and** *v*: $v \in span\ basis$ **and** *u*: $u \in span\ basis$
**shows** $representation\ basis\ (u - v) = (\lambda b.\ representation\ basis\ u\ b - representation\ basis\ v\ b)$
⟨*proof*⟩

**lemma** *representation-neg*:
$independent\ basis \Longrightarrow v \in span\ basis \Longrightarrow representation\ basis\ (-\ v) = (\lambda b.\ -\ representation\ basis\ v\ b)$
⟨*proof*⟩

**lemma** *representation-add*:
$independent\ basis \Longrightarrow v \in span\ basis \Longrightarrow u \in span\ basis \Longrightarrow$
$representation\ basis\ (u + v) = (\lambda b.\ representation\ basis\ u\ b + representation\ basis\ v\ b)$
⟨*proof*⟩

**lemma** *representation-sum*:
$independent\ basis \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow v\ i \in span\ basis) \Longrightarrow$
$representation\ basis\ (sum\ v\ I) = (\lambda b.\ \sum i{\in}I.\ representation\ basis\ (v\ i)\ b)$
⟨*proof*⟩

**lemma** *representation-scale*:
**assumes** *basis*: $independent\ basis$ **and** *v*: $v \in span\ basis$
**shows** $representation\ basis\ (r *s\ v) = (\lambda b.\ r * representation\ basis\ v\ b)$
⟨*proof*⟩

**lemma** *representation-extend*:
**assumes** *basis*: $independent\ basis$ **and** *v*: $v \in span\ basis'$ **and** *basis'*: $basis' \subseteq basis$
**shows** $representation\ basis\ v = representation\ basis'\ v$
⟨*proof*⟩

The set $B$ is the maximal independent set for *span* $B$, or $A$ is the minimal spanning set

**lemma** *spanning-subset-independent*:
  **assumes** *BA*: $B \subseteq A$
    **and** *iA*: *independent A*
    **and** *AsB*: $A \subseteq$ *span B*
  **shows** $A = B$
⟨*proof*⟩

**end**

A linear function is a mapping between two modules over the same ring.

**locale** *module-hom* = *m1*: *module s1* + *m2*: *module s2*
    **for** *s1* :: *'a::comm-ring-1* ⇒ *'b::ab-group-add* ⇒ *'b* (**infixr** ‹∗a› *75*)
    **and** *s2* :: *'a::comm-ring-1* ⇒ *'c::ab-group-add* ⇒ *'c* (**infixr** ‹∗b› *75*) +
  **fixes** *f* :: *'b* ⇒ *'c*
  **assumes** *add*: *f* (*b1* + *b2*) = *f b1* + *f b2*
    **and** *scale*: *f* (*r* ∗*a* *b*) = *r* ∗*b* *f b*
**begin**

**lemma** *zero*[*simp*]: *f 0* = *0*
  ⟨*proof*⟩

**lemma** *neg*: *f* (− *x*) = − *f x*
  ⟨*proof*⟩

**lemma** *diff*: *f* (*x* − *y*) = *f x* − *f y*
  ⟨*proof*⟩

**lemma** *sum*: *f* (*sum g S*) = ($\sum$ *a*∈*S*. *f* (*g a*))
⟨*proof*⟩

**lemma** *inj-on-iff-eq-0*:
  **assumes** *s*: *m1.subspace s*
  **shows** *inj-on f s* ⟷ (∀ *x*∈*s*. *f x* = *0* ⟶ *x* = *0*)
⟨*proof*⟩

**lemma** *inj-iff-eq-0*: *inj f* = (∀ *x*. *f x* = *0* ⟶ *x* = *0*)
  ⟨*proof*⟩

**lemma** *subspace-image*: **assumes** *S*: *m1.subspace S* **shows** *m2.subspace* (*f* ‘ *S*)
  ⟨*proof*⟩

**lemma** *subspace-vimage*: *m2.subspace S* ⟹ *m1.subspace* (*f* −‘ *S*)
  ⟨*proof*⟩

**lemma** *subspace-kernel*: *m1.subspace* {*x*. *f x* = *0*}
  ⟨*proof*⟩

**lemma** *span-image*: *m2.span* (*f* ‘ *S*) = *f* ‘ (*m1.span S*)
⟨*proof*⟩

**lemma** *dependent-inj-imageD*:
**assumes** *d*: *m2.dependent* ($f$ ' $s$) **and** *i*: *inj-on* $f$ (*m1.span* $s$)
**shows** *m1.dependent* $s$
⟨*proof*⟩

**lemma** *eq-0-on-span*:
**assumes** *f0*: $\bigwedge x.\ x \in b \Longrightarrow f\ x = 0$ **and** *x*: $x \in$ *m1.span* $b$ **shows** $f\ x = 0$
⟨*proof*⟩

**lemma** *independent-injective-image*: *m1.independent* $s \Longrightarrow$ *inj-on* $f$ (*m1.span* $s$) $\Longrightarrow$ *m2.independent* ($f$ ' $s$)
⟨*proof*⟩

**lemma** *inj-on-span-independent-image*:
**assumes** *ifB*: *m2.independent* ($f$ ' $B$) **and** *f*: *inj-on* $f$ $B$ **shows** *inj-on* $f$ (*m1.span* $B$)
⟨*proof*⟩

**lemma** *inj-on-span-iff-independent-image*: *m2.independent* ($f$ ' $B$) $\Longrightarrow$ *inj-on* $f$ (*m1.span* $B$) $\longleftrightarrow$ *inj-on* $f$ $B$
⟨*proof*⟩

**lemma** *subspace-linear-preimage*: *m2.subspace* $S \Longrightarrow$ *m1.subspace* $\{x.\ f\ x \in S\}$
⟨*proof*⟩

**lemma** *spans-image*: $V \subseteq$ *m1.span* $B \Longrightarrow f$ ' $V \subseteq$ *m2.span* ($f$ ' $B$)
⟨*proof*⟩

Relation between bases and injectivity/surjectivity of map.

**lemma** *spanning-surjective-image*:
**assumes** *us*: *UNIV* $\subseteq$ *m1.span* $S$
**and** *sf*: *surj* $f$
**shows** *UNIV* $\subseteq$ *m2.span* ($f$ ' $S$)
⟨*proof*⟩

**lemmas** *independent-inj-on-image* = *independent-injective-image*

**lemma** *independent-inj-image*:
*m1.independent* $S \Longrightarrow$ *inj* $f \Longrightarrow$ *m2.independent* ($f$ ' $S$)
⟨*proof*⟩

**end**

**lemma** *module-hom-iff*:
*module-hom* *s1* *s2* $f \longleftrightarrow$
*module* *s1* $\wedge$ *module* *s2* $\wedge$
$(\forall x\ y.\ f\ (x + y) = f\ x + f\ y) \wedge (\forall c\ x.\ f\ (s1\ c\ x) = s2\ c\ (f\ x))$
⟨*proof*⟩

**locale** *module-pair* = *m1*: *module s1* + *m2*: *module s2*
**for** *s1* :: *'a* :: *comm-ring-1* ⇒ *'b* ⇒ *'b* :: *ab-group-add*
**and** *s2* :: *'a* :: *comm-ring-1* ⇒ *'c* ⇒ *'c* :: *ab-group-add*
**begin**

**lemma** *module-hom-zero*: *module-hom s1 s2* (λ*x*. *0*)
⟨*proof*⟩

**lemma** *module-hom-add*: *module-hom s1 s2 f* ⟹ *module-hom s1 s2 g* ⟹ *module-hom s1 s2* (λ*x*. *f x* + *g x*)
⟨*proof*⟩

**lemma** *module-hom-sub*: *module-hom s1 s2 f* ⟹ *module-hom s1 s2 g* ⟹ *module-hom s1 s2* (λ*x*. *f x* − *g x*)
⟨*proof*⟩

**lemma** *module-hom-neg*: *module-hom s1 s2 f* ⟹ *module-hom s1 s2* (λ*x*. − *f x*)
⟨*proof*⟩

**lemma** *module-hom-scale*: *module-hom s1 s2 f* ⟹ *module-hom s1 s2* (λ*x*. *s2 c* (*f x*))
⟨*proof*⟩

**lemma** *module-hom-compose-scale*:
*module-hom s1 s2* (λ*x*. *s2* (*f x*) (*c*))
**if** *module-hom s1* (∗) *f*
⟨*proof*⟩

**lemma** *bij-module-hom-imp-inv-module-hom*: *module-hom scale1 scale2 f* ⟹ *bij f* ⟹
*module-hom scale2 scale1* (*inv f*)
⟨*proof*⟩

**lemma** *module-hom-sum*: (⋀*i*. *i* ∈ *I* ⟹ *module-hom s1 s2* (*f i*)) ⟹ (*I* = {} ⟹ *module s1* ∧ *module s2*) ⟹ *module-hom s1 s2* (λ*x*. ∑ *i*∈*I*. *f i x*)
⟨*proof*⟩

**lemma** *module-hom-eq-on-span*: *f x* = *g x*
**if** *module-hom s1 s2 f module-hom s1 s2 g*
**and** (⋀*x*. *x* ∈ *B* ⟹ *f x* = *g x*) *x* ∈ *m1.span B*
⟨*proof*⟩

**end**

**context** *module* **begin**

**lemma** *module-hom-scale-self*[*simp*]:
*module-hom scale scale* (λ*x*. *scale c x*)

⟨*proof*⟩

**lemma** *module-hom-scale-left*[*simp*]:
*module-hom* (∗) *scale* (λ*r*. *scale r x*)
⟨*proof*⟩

**lemma** *module-hom-id*: *module-hom scale scale id*
⟨*proof*⟩

**lemma** *module-hom-ident*: *module-hom scale scale* (λ*x*. *x*)
⟨*proof*⟩

**lemma** *module-hom-uminus*: *module-hom scale scale uminus*
⟨*proof*⟩

**end**

**lemma** *module-hom-compose*: *module-hom s1 s2 f* ⟹ *module-hom s2 s3 g* ⟹ *module-hom s1 s3* (*g o f*)
⟨*proof*⟩

**end**

# 106 Vector Spaces

**theory** *Vector-Spaces*
**imports** *Modules*
**begin**

**lemma** *isomorphism-expand*:
$f \circ g = id \wedge g \circ f = id \longleftrightarrow (\forall x.\ f\ (g\ x) = x) \wedge (\forall x.\ g\ (f\ x) = x)$
⟨*proof*⟩

**lemma** *left-right-inverse-eq*:
**assumes** *fg*: $f \circ g = id$
**and** *gh*: $g \circ h = id$
**shows** $f = h$
⟨*proof*⟩

**lemma** *ordLeq3-finite-infinite*:
**assumes** *A*: *finite A* **and** *B*: *infinite B* **shows** *ordLeq3* (*card-of A*) (*card-of B*)
⟨*proof*⟩

**locale** *vector-space* =
**fixes** *scale* :: *'a*::*field* ⇒ *'b*::*ab-group-add* ⇒ *'b* (**infixr** ‹∗*s*› *75*)
**assumes** *vector-space-assms*:— re-stating the assumptions of *module* instead of extending *module* allows us to rewrite in the sublocale.
$a *s (x + y) = a *s x + a *s y$
$(a + b) *s x = a *s x + b *s x$

$a *s (b *s x) = (a * b) *s x$
$1 *s x = x$

**lemma** *module-iff-vector-space*: *module s* $\longleftrightarrow$ *vector-space s*
⟨*proof*⟩

**locale** *linear* = *vs1*: *vector-space s1* + *vs2*: *vector-space s2* + *module-hom s1 s2 f*
**for** *s1* :: *'a::field* ⇒ *'b::ab-group-add* ⇒ *'b* (**infixr** ‹*a› *75*)
**and** *s2* :: *'a::field* ⇒ *'c::ab-group-add* ⇒ *'c* (**infixr** ‹*b› *75*)
**and** *f* :: *'b* ⇒ *'c*

**lemma** *module-hom-iff-linear*: *module-hom s1 s2 f* $\longleftrightarrow$ *linear s1 s2 f*
⟨*proof*⟩
**lemmas** *module-hom-eq-linear* = *module-hom-iff-linear*[*abs-def*, *THEN meta-eq-to-obj-eq*]
**lemmas** *linear-iff-module-hom* = *module-hom-iff-linear*[*symmetric*]
**lemmas** *linear-module-homI* = *module-hom-iff-linear*[*THEN iffD1*]
**and** *module-hom-linearI* = *module-hom-iff-linear*[*THEN iffD2*]

**context** *vector-space* **begin**

**sublocale** *module scale* **rewrites** *module-hom* = *linear*
⟨*proof*⟩

**lemmas**— from *module*
*linear-id* = *module-hom-id*
**and** *linear-ident* = *module-hom-ident*
**and** *linear-scale-self* = *module-hom-scale-self*
**and** *linear-scale-left* = *module-hom-scale-left*
**and** *linear-uminus* = *module-hom-uminus*

**lemma** *linear-representation*:
**assumes** *independent B span B* = *UNIV*
**shows** *linear scale* (∗) (λ*v*. *representation B v b*)
⟨*proof*⟩

**lemma** *linear-imp-scale*:
**fixes** *D*::*'a* ⇒ *'b*
**assumes** *linear* (∗) *scale D*
**obtains** *d* **where** *D* = (λ*x*. *scale x d*)
⟨*proof*⟩

**lemma** *scale-eq-0-iff* [*simp*]: *scale a x* = *0* $\longleftrightarrow$ *a* = *0* ∨ *x* = *0*
⟨*proof*⟩

**lemma** *scale-left-imp-eq*:
**assumes** *nonzero*: *a* ≠ *0*
**and** *scale*: *scale a x* = *scale a y*
**shows** *x* = *y*
⟨*proof*⟩

**lemma** *scale-right-imp-eq*:
  **assumes** *nonzero*: $x \neq 0$
    **and** *scale*: $scale\ a\ x = scale\ b\ x$
  **shows** $a = b$
⟨*proof*⟩

**lemma** *scale-cancel-left* [*simp*]: $scale\ a\ x = scale\ a\ y \longleftrightarrow x = y \lor a = 0$
  ⟨*proof*⟩

**lemma** *scale-cancel-right* [*simp*]: $scale\ a\ x = scale\ b\ x \longleftrightarrow a = b \lor x = 0$
  ⟨*proof*⟩

**lemma** *injective-scale*: $c \neq 0 \Longrightarrow inj\ (\lambda x.\ scale\ c\ x)$
  ⟨*proof*⟩

**lemma** *dependent-def*: $dependent\ P \longleftrightarrow (\exists\, a \in P.\ a \in span\ (P - \{a\}))$
  ⟨*proof*⟩

**lemma** *dependent-single*[*simp*]: $dependent\ \{x\} \longleftrightarrow x = 0$
  ⟨*proof*⟩

**lemma** *in-span-insert*:
  **assumes** *a*: $a \in span\ (insert\ b\ S)$
    **and** *na*: $a \notin span\ S$
  **shows** $b \in span\ (insert\ a\ S)$
⟨*proof*⟩

**lemma** *dependent-insertD*: **assumes** *a*: $a \notin span\ S$ **and** *S*: $dependent\ (insert\ a\ S)$
**shows** $dependent\ S$
⟨*proof*⟩

**lemma** *independent-insertI*: $a \notin span\ S \Longrightarrow independent\ S \Longrightarrow independent\ (insert\ a\ S)$
  ⟨*proof*⟩

**lemma** *independent-insert*:
  $independent\ (insert\ a\ S) \longleftrightarrow (if\ a \in S\ then\ independent\ S\ else\ independent\ S \land a \notin span\ S)$
⟨*proof*⟩

**lemma** *maximal-independent-subset-extend*:
  **assumes** $S \subseteq V$ $independent\ S$
  **obtains** $B$ **where** $S \subseteq B$ $B \subseteq V$ $independent\ B$ $V \subseteq span\ B$
⟨*proof*⟩

**lemma** *maximal-independent-subset*:
  **obtains** $B$ **where** $B \subseteq V$ $independent\ B$ $V \subseteq span\ B$
  ⟨*proof*⟩

Extends a basis from B to a basis of the entire space.

**definition** *extend-basis* :: $'b\ set \Rightarrow 'b\ set$
**where** *extend-basis* $B = (SOME\ B'.\ B \subseteq B' \wedge independent\ B' \wedge span\ B' = UNIV)$

**lemma**
**assumes** $B$: *independent* $B$
**shows** *extend-basis-superset*: $B \subseteq extend\text{-}basis\ B$
**and** *independent-extend-basis*: *independent* $(extend\text{-}basis\ B)$
**and** *span-extend-basis*[*simp*]: $span\ (extend\text{-}basis\ B) = UNIV$
⟨*proof*⟩

**lemma** *in-span-delete*:
**assumes** $a$: $a \in span\ S$ **and** $na$: $a \notin span\ (S - \{b\})$
**shows** $b \in span\ (insert\ a\ (S - \{b\}))$
⟨*proof*⟩

**lemma** *span-redundant*: $x \in span\ S \Longrightarrow span\ (insert\ x\ S) = span\ S$
⟨*proof*⟩

**lemma** *span-trans*: $x \in span\ S \Longrightarrow y \in span\ (insert\ x\ S) \Longrightarrow y \in span\ S$
⟨*proof*⟩

**lemma** *span-insert-0*[*simp*]: $span\ (insert\ 0\ S) = span\ S$
⟨*proof*⟩

**lemma** *span-delete-0* [*simp*]: $span(S - \{0\}) = span\ S$
⟨*proof*⟩

**lemma** *span-image-scale*:
**assumes** *finite* $S$ **and** $nz$: $\bigwedge x.\ x \in S \Longrightarrow c\ x \neq 0$
**shows** $span\ ((\lambda x.\ c\ x *s\ x)\ `\ S) = span\ S$
⟨*proof*⟩

**lemma** *exchange-lemma*:
**assumes** $f$: *finite* $T$
**and** $i$: *independent* $S$
**and** $sp$: $S \subseteq span\ T$
**shows** $\exists\, t'.\ card\ t' = card\ T \wedge finite\ t' \wedge S \subseteq t' \wedge t' \subseteq S \cup T \wedge S \subseteq span\ t'$
⟨*proof*⟩

**lemma** *independent-span-bound*:
**assumes** $f$: *finite* $T$
**and** $i$: *independent* $S$
**and** $sp$: $S \subseteq span\ T$
**shows** $finite\ S \wedge card\ S \leq card\ T$
⟨*proof*⟩

**lemma** *independent-explicit-finite-subsets*:

$independent\ A \longleftrightarrow (\forall\, S \subseteq A.\ finite\ S \longrightarrow (\forall\, u.\ (\sum v{\in}S.\ u\ v\ {*s}\ v) = 0 \longrightarrow (\forall\, v{\in}S.\ u\ v = 0)))$
⟨*proof*⟩

**lemma** *independent-if-scalars-zero*:
**assumes** *fin-A*: $finite\ A$
**and** *sum*: $\bigwedge f\ x.\ (\sum x{\in}A.\ f\ x\ {*s}\ x) = 0 \Longrightarrow x \in A \Longrightarrow f\ x = 0$
**shows** $independent\ A$
⟨*proof*⟩

**lemma** *bij-if-span-eq-span-bases*:
**assumes** *B*: $independent\ B$ **and** *C*: $independent\ C$
**and** *eq*: $span\ B = span\ C$
**shows** $\exists f.\ bij\text{-}betw\ f\ B\ C$
⟨*proof*⟩

**definition** $dim :: {'b}\ set \Rightarrow nat$
**where** $dim\ V = (if\ \exists\, b.\ independent\ b \wedge span\ b = span\ V\ then\ card\ (SOME\ b.\ independent\ b \wedge span\ b = span\ V)\ else\ 0)$

**lemma** *dim-eq-card*:
**assumes** *BV*: $span\ B = span\ V$ **and** *B*: $independent\ B$
**shows** $dim\ V = card\ B$
⟨*proof*⟩

**lemma** *basis-card-eq-dim*: $B \subseteq V \Longrightarrow V \subseteq span\ B \Longrightarrow independent\ B \Longrightarrow card\ B = dim\ V$
⟨*proof*⟩

**lemma** *basis-exists*:
**obtains** *B* **where** $B \subseteq V$ $independent\ B$ $V \subseteq span\ B$ $card\ B = dim\ V$
⟨*proof*⟩

**lemma** *dim-eq-card-independent*: $independent\ B \Longrightarrow dim\ B = card\ B$
⟨*proof*⟩

**lemma** *dim-span*[*simp*]: $dim\ (span\ S) = dim\ S$
⟨*proof*⟩

**lemma** *dim-span-eq-card-independent*: $independent\ B \Longrightarrow dim\ (span\ B) = card\ B$
⟨*proof*⟩

**lemma** *dim-le-card*: **assumes** $V \subseteq span\ W$ $finite\ W$ **shows** $dim\ V \le card\ W$
⟨*proof*⟩

**lemma** *span-eq-dim*: $span\ S = span\ T \Longrightarrow dim\ S = dim\ T$
⟨*proof*⟩

**corollary** *dim-le-card'*:

*finite s* $\Longrightarrow$ *dim s* $\leq$ *card s*
⟨*proof*⟩

**lemma** *span-card-ge-dim*:
$B \subseteq V \Longrightarrow V \subseteq$ *span B* $\Longrightarrow$ *finite B* $\Longrightarrow$ *dim V* $\leq$ *card B*
⟨*proof*⟩

**lemma** *dim-unique*:
$B \subseteq V \Longrightarrow V \subseteq$ *span B* $\Longrightarrow$ *independent B* $\Longrightarrow$ *card B* $= n \Longrightarrow$ *dim V* $= n$
⟨*proof*⟩

**lemma** *subspace-sums*: ⟦*subspace S*; *subspace T*⟧ $\Longrightarrow$ *subspace* $\{x + y|x\ y.\ x \in S \land y \in T\}$
⟨*proof*⟩

**end**

**lemma** *linear-iff*: *linear s1 s2 f* $\longleftrightarrow$
(*vector-space s1* $\land$ *vector-space s2* $\land$ $(\forall x\ y.\ f\ (x + y) = f\ x + f\ y) \land (\forall c\ x.\ f\ (s1\ c\ x) = s2\ c\ (f\ x)))$
⟨*proof*⟩

**context begin**
**qualified lemma** *linear-compose*: *linear s1 s2 f* $\Longrightarrow$ *linear s2 s3 g* $\Longrightarrow$ *linear s1 s3* (*g o f*)
⟨*proof*⟩
**end**

**locale** *vector-space-pair* = *vs1*: *vector-space s1* + *vs2*: *vector-space s2*
**for** *s1* :: *'a::field* ⇒ *'b::ab-group-add* ⇒ *'b* (**infixr** ‹∗a› *75*)
**and** *s2* :: *'a::field* ⇒ *'c::ab-group-add* ⇒ *'c* (**infixr** ‹∗b› *75*)
**begin**

**context fixes** *f* **assumes** *linear s1 s2 f* **begin**
**interpretation** *linear s1 s2 f* ⟨*proof*⟩
**lemmas**— from locale *module-hom*
*linear-0* = *zero*
**and** *linear-add* = *add*
**and** *linear-scale* = *scale*
**and** *linear-neg* = *neg*
**and** *linear-diff* = *diff*
**and** *linear-sum* = *sum*
**and** *linear-inj-on-iff-eq-0* = *inj-on-iff-eq-0*
**and** *linear-inj-iff-eq-0* = *inj-iff-eq-0*
**and** *linear-subspace-image* = *subspace-image*
**and** *linear-subspace-vimage* = *subspace-vimage*
**and** *linear-subspace-kernel* = *subspace-kernel*
**and** *linear-span-image* = *span-image*
**and** *linear-dependent-inj-imageD* = *dependent-inj-imageD*

**and** *linear-eq-0-on-span* = *eq-0-on-span*
**and** *linear-independent-injective-image* = *independent-injective-image*
**and** *linear-inj-on-span-independent-image* = *inj-on-span-independent-image*
**and** *linear-inj-on-span-iff-independent-image* = *inj-on-span-iff-independent-image*
**and** *linear-subspace-linear-preimage* = *subspace-linear-preimage*
**and** *linear-spans-image* = *spans-image*
**and** *linear-spanning-surjective-image* = *spanning-surjective-image*
**end**

**sublocale** *module-pair*
**rewrites** *module-hom* = *linear*
⟨*proof*⟩

**lemmas**— from locale *module-pair*
*linear-eq-on-span* = *module-hom-eq-on-span*
**and** *linear-compose-scale-right* = *module-hom-scale*
**and** *linear-compose-add* = *module-hom-add*
**and** *linear-zero* = *module-hom-zero*
**and** *linear-compose-sub* = *module-hom-sub*
**and** *linear-compose-neg* = *module-hom-neg*
**and** *linear-compose-scale* = *module-hom-compose-scale*

**lemma** *linear-indep-image-lemma*:
**assumes** *lf*: *linear s1 s2 f*
**and** *fB*: *finite B*
**and** *ifB*: *vs2.independent* (*f* ' *B*)
**and** *fi*: *inj-on f B*
**and** *xsB*: $x \in$ *vs1.span B*
**and** *fx*: $f\ x = 0$
**shows** $x = 0$
⟨*proof*⟩

**lemma** *linear-eq-on*:
**assumes** *l*: *linear s1 s2 f linear s1 s2 g*
**assumes** *x*: $x \in$ *vs1.span B* **and** *eq*: $\bigwedge b.\ b \in B \Longrightarrow f\ b = g\ b$
**shows** $f\ x = g\ x$
⟨*proof*⟩

**definition** *construct* :: $'b$ *set* $\Rightarrow ('b \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c)$
**where** *construct B g v* = ($\sum b$ | *vs1.representation* (*vs1.extend-basis B*) *v b* ≠ *0*.
*vs1.representation* (*vs1.extend-basis B*) *v b* $*b$ (*if* $b \in B$ *then g b else 0*))

**lemma** *construct-cong*: ($\bigwedge b.\ b \in B \Longrightarrow f\ b = g\ b$) $\Longrightarrow$ *construct B f* = *construct B g*
⟨*proof*⟩

**lemma** *linear-construct*:
**assumes** *B*[*simp*]: *vs1.independent B*

**shows** *linear s1 s2* (*construct B f*)
⟨*proof*⟩

**lemma** *construct-basis*:
**assumes** *B*[*simp*]: *vs1.independent B* **and** *b*: $b \in B$
**shows** *construct B f b* = *f b*
⟨*proof*⟩

**lemma** *construct-outside*:
**assumes** *B*: *vs1.independent B* **and** *v*: $v \in$ *vs1.span* (*vs1.extend-basis B* − *B*)
**shows** *construct B f v* = *0*
⟨*proof*⟩

**lemma** *construct-add*:
**assumes** *B*[*simp*]: *vs1.independent B*
**shows** *construct B* ($\lambda x$. *f x* + *g x*) *v* = *construct B f v* + *construct B g v*
⟨*proof*⟩

**lemma** *construct-scale*:
**assumes** *B*[*simp*]: *vs1.independent B*
**shows** *construct B* ($\lambda x$. *c* ∗*b f x*) *v* = *c* ∗*b construct B f v*
⟨*proof*⟩

**lemma** *construct-in-span*:
**assumes** *B*[*simp*]: *vs1.independent B*
**shows** *construct B f v* ∈ *vs2.span* (*f* ‘ *B*)
⟨*proof*⟩

**lemma** *linear-compose-sum*:
**assumes** *lS*: ∀ *a* ∈ *S*. *linear s1 s2* (*f a*)
**shows** *linear s1 s2* ($\lambda x$. *sum* ($\lambda a$. *f a x*) *S*)
⟨*proof*⟩

**lemma** *in-span-in-range-construct*:
*x* ∈ *range* (*construct B f*) **if** *i*: *vs1.independent B* **and** *x*: *x* ∈ *vs2.span* (*f* ‘ *B*)
⟨*proof*⟩

**lemma** *range-construct-eq-span*:
*range* (*construct B f*) = *vs2.span* (*f* ‘ *B*)
**if** *vs1.independent B*
⟨*proof*⟩

**lemma** *linear-independent-extend-subspace*:
— legacy: use *construct* instead
**assumes** *vs1.independent B*
**shows** ∃ *g*. *linear s1 s2 g* ∧ (∀ *x*∈*B*. *g x* = *f x*) ∧ *range g* = *vs2.span* (*f*‘*B*)
⟨*proof*⟩

**lemma** *linear-independent-extend*:

*vs1.independent B* $\Longrightarrow \exists g.$ *linear s1 s2 g* $\wedge$ ($\forall x \in B.$ *g x* = *f x*)
⟨*proof*⟩

**lemma** *linear-exists-left-inverse-on*:
**assumes** *lf*: *linear s1 s2 f*
**assumes** *V*: *vs1.subspace V* **and** *f*: *inj-on f V*
**shows** $\exists g.$ *g ‘ UNIV* $\subseteq$ *V* $\wedge$ *linear s2 s1 g* $\wedge$ ($\forall v \in V.$ *g* (*f v*) = *v*)
⟨*proof*⟩

**lemma** *linear-exists-right-inverse-on*:
**assumes** *lf*: *linear s1 s2 f*
**assumes** *vs1.subspace V*
**shows** $\exists g.$ *g ‘ UNIV* $\subseteq$ *V* $\wedge$ *linear s2 s1 g* $\wedge$ ($\forall v \in f$ *‘ V. f* (*g v*) = *v*)
⟨*proof*⟩

**lemma** *linear-inj-on-left-inverse*:
**assumes** *lf*: *linear s1 s2 f*
**assumes** *fi*: *inj-on f* (*vs1.span S*)
**shows** $\exists g.$ *range g* $\subseteq$ *vs1.span S* $\wedge$ *linear s2 s1 g* $\wedge$ ($\forall x \in$ *vs1.span S. g* (*f x*) = *x*)
⟨*proof*⟩

**lemma** *linear-injective-left-inverse*: *linear s1 s2 f* $\Longrightarrow$ *inj f* $\Longrightarrow \exists g.$ *linear s2 s1 g* $\wedge$ *g* $\circ$ *f* = *id*
⟨*proof*⟩

**lemma** *linear-surj-right-inverse*:
**assumes** *lf*: *linear s1 s2 f*
**assumes** *sf*: *vs2.span T* $\subseteq$ *f‘vs1.span S*
**shows** $\exists g.$ *range g* $\subseteq$ *vs1.span S* $\wedge$ *linear s2 s1 g* $\wedge$ ($\forall x \in$ *vs2.span T. f* (*g x*) = *x*)
⟨*proof*⟩

**lemma** *linear-surjective-right-inverse*: *linear s1 s2 f* $\Longrightarrow$ *surj f* $\Longrightarrow \exists g.$ *linear s2 s1 g* $\wedge$ *f* $\circ$ *g* = *id*
⟨*proof*⟩

**lemma** *finite-basis-to-basis-subspace-isomorphism*:
**assumes** *s*: *vs1.subspace S*
**and** *t*: *vs2.subspace T*
**and** *d*: *vs1.dim S* = *vs2.dim T*
**and** *fB*: *finite B*
**and** *B*: *B* $\subseteq$ *S vs1.independent B S* $\subseteq$ *vs1.span B card B* = *vs1.dim S*
**and** *fC*: *finite C*
**and** *C*: *C* $\subseteq$ *T vs2.independent C T* $\subseteq$ *vs2.span C card C* = *vs2.dim T*
**shows** $\exists f.$ *linear s1 s2 f* $\wedge$ *f ‘ B* = *C* $\wedge$ *f ‘ S* = *T* $\wedge$ *inj-on f S*
⟨*proof*⟩

**end**

**locale** *finite-dimensional-vector-space* = *vector-space* +
**fixes** *Basis* :: *'b set*
**assumes** *finite-Basis*: *finite Basis*
**and** *independent-Basis*: *independent Basis*
**and** *span-Basis*: *span Basis* = *UNIV*
**begin**

**definition** *dimension* = *card Basis*

**lemma** *finiteI-independent*: *independent B* $\Longrightarrow$ *finite B*
⟨*proof*⟩

**lemma** *dim-empty* [*simp*]: *dim* {} = *0*
⟨*proof*⟩

**lemma** *dim-insert*:
*dim* (*insert x S*) = (*if x* ∈ *span S then dim S else dim S* + *1*)
⟨*proof*⟩

**lemma** *dim-singleton* [*simp*]: *dim*{*x*} = (*if x* = *0 then 0 else 1*)
⟨*proof*⟩

**proposition** *choose-subspace-of-subspace*:
**assumes** *n* ≤ *dim S*
**obtains** *T* **where** *subspace T T* ⊆ *span S dim T* = *n*
⟨*proof*⟩

**lemma** *basis-subspace-exists*:
**assumes** *subspace S*
**obtains** *B* **where** *finite B B* ⊆ *S independent B span B* = *S card B* = *dim S*
⟨*proof*⟩

**lemma** *dim-mono*: **assumes** *V* ⊆ *span W* **shows** *dim V* ≤ *dim W*
⟨*proof*⟩

**lemma** *dim-subset*: *S* ⊆ *T* $\Longrightarrow$ *dim S* ≤ *dim T*
⟨*proof*⟩

**lemma** *dim-eq-0* [*simp*]:
*dim S* = *0* $\longleftrightarrow$ *S* ⊆ {*0*}
⟨*proof*⟩

**lemma** *dim-UNIV*[*simp*]: *dim UNIV* = *card Basis*
⟨*proof*⟩

**lemma** *independent-card-le-dim*: **assumes** *B* ⊆ *V* **and** *independent B* **shows** *card B* ≤ *dim V*

⟨proof⟩

**lemma** *dim-subset-UNIV*: $dim\ S \leq dimension$
⟨proof⟩

**lemma** *card-ge-dim-independent*:
**assumes** *BV*: $B \subseteq V$
**and** *iB*: *independent B*
**and** *dVB*: $dim\ V \leq card\ B$
**shows** $V \subseteq span\ B$
⟨proof⟩

**lemma** *card-le-dim-spanning*:
**assumes** *BV*: $B \subseteq V$
**and** *VB*: $V \subseteq span\ B$
**and** *fB*: *finite B*
**and** *dVB*: $dim\ V \geq card\ B$
**shows** *independent B*
⟨proof⟩

**lemma** *card-eq-dim*: $B \subseteq V \Longrightarrow card\ B = dim\ V \Longrightarrow finite\ B \Longrightarrow independent\ B \longleftrightarrow V \subseteq span\ B$
⟨proof⟩

**lemma** *subspace-dim-equal*:
**assumes** *subspace S*
**and** *subspace T*
**and** $S \subseteq T$
**and** $dim\ S \geq dim\ T$
**shows** $S = T$
⟨proof⟩

**corollary** *dim-eq-span*:
**shows** $[\![S \subseteq T;\ dim\ T \leq dim\ S]\!] \Longrightarrow span\ S = span\ T$
⟨proof⟩

**lemma** *dim-psubset*:
$span\ S \subset span\ T \Longrightarrow dim\ S < dim\ T$
⟨proof⟩

**lemma** *dim-eq-full*:
**shows** $dim\ S = dimension \longleftrightarrow span\ S = UNIV$
⟨proof⟩

**lemma** *indep-card-eq-dim-span*:
**assumes** *independent B*
**shows** $finite\ B \wedge card\ B = dim\ (span\ B)$
⟨proof⟩

More general size bound lemmas.

**lemma** *independent-bound-general*:
*independent S* $\Longrightarrow$ *finite S* $\wedge$ *card S* $\leq$ *dim S*
⟨*proof*⟩

**lemma** *independent-explicit*:
**shows** *independent B* $\longleftrightarrow$ *finite B* $\wedge$ ($\forall$ *c*. ($\sum$ *v*$\in$*B*. *c v* $*s$ *v*) = *0* $\longrightarrow$ ($\forall$ *v* $\in$ *B*. *c v* = *0*))
⟨*proof*⟩

**proposition** *dim-sums-Int*:
**assumes** *subspace S subspace T*
**shows** *dim* {*x* + *y* |*x y*. *x* $\in$ *S* $\wedge$ *y* $\in$ *T*} + *dim*(*S* $\cap$ *T*) = *dim S* + *dim T* (**is** *dim ?ST* + - = -)
⟨*proof*⟩

**lemma** *dependent-biggerset-general*:
(*finite S* $\Longrightarrow$ *card S* > *dim S*) $\Longrightarrow$ *dependent S*
⟨*proof*⟩

**lemma** *subset-le-dim*:
*S* $\subseteq$ *span T* $\Longrightarrow$ *dim S* $\leq$ *dim T*
⟨*proof*⟩

**lemma** *linear-inj-imp-surj*:
**assumes** *lf*: *linear scale scale f*
**and** *fi*: *inj f*
**shows** *surj f*
⟨*proof*⟩

**end**

**locale** *finite-dimensional-vector-space-pair-1* =
*vs1*: *finite-dimensional-vector-space s1 B1* + *vs2*: *vector-space s2*
**for** *s1* :: *'a*::*field* $\Rightarrow$ *'b*::*ab-group-add* $\Rightarrow$ *'b* (**infixr** ‹$*a$› *75*)
**and** *B1* :: *'b set*
**and** *s2* :: *'a*::*field* $\Rightarrow$ *'c*::*ab-group-add* $\Rightarrow$ *'c* (**infixr** ‹$*b$› *75*)
**begin**

**sublocale** *vector-space-pair s1 s2* ⟨*proof*⟩

**lemma** *dim-image-eq*:
**assumes** *lf*: *linear s1 s2 f*
**and** *fi*: *inj-on f* (*vs1.span S*)
**shows** *vs2.dim* (*f* ‘ *S*) = *vs1.dim S*
⟨*proof*⟩

**lemma** *dim-image-le*:
**assumes** *lf*: *linear s1 s2 f*
**shows** *vs2.dim* (*f* ‘ *S*) $\leq$ *vs1.dim* (*S*)

⟨*proof*⟩

**end**

**locale** *finite-dimensional-vector-space-pair* =
*vs1*: *finite-dimensional-vector-space s1 B1* + *vs2*: *finite-dimensional-vector-space s2 B2*
**for** *s1* :: $'a$::*field* ⇒ $'b$::*ab-group-add* ⇒ $'b$ (**infixr** ‹∗a› *75*)
**and** *B1* :: $'b$ *set*
**and** *s2* :: $'a$::*field* ⇒ $'c$::*ab-group-add* ⇒ $'c$ (**infixr** ‹∗b› *75*)
**and** *B2* :: $'c$ *set*
**begin**

**sublocale** *finite-dimensional-vector-space-pair-1* ⟨*proof*⟩

**lemma** *linear-surjective-imp-injective*:
**assumes** *lf*: *linear s1 s2 f* **and** *sf*: *surj f* **and** *eq*: *vs2.dim UNIV* = *vs1.dim UNIV*
**shows** *inj f*
⟨*proof*⟩

**lemma** *linear-injective-imp-surjective*:
**assumes** *lf*: *linear s1 s2 f* **and** *sf*: *inj f* **and** *eq*: *vs2.dim UNIV* = *vs1.dim UNIV*
**shows** *surj f*
⟨*proof*⟩

**lemma** *linear-injective-isomorphism*:
**assumes** *lf*: *linear s1 s2 f*
**and** *fi*: *inj f*
**and** *dims*: *vs2.dim UNIV* = *vs1.dim UNIV*
**shows** $\exists f'.$ *linear s2 s1* $f' \wedge (\forall x.\ f'\ (f\ x) = x) \wedge (\forall x.\ f\ (f'\ x) = x)$
⟨*proof*⟩

**lemma** *linear-surjective-isomorphism*:
**assumes** *lf*: *linear s1 s2 f*
**and** *sf*: *surj f*
**and** *dims*: *vs2.dim UNIV* = *vs1.dim UNIV*
**shows** $\exists f'.$ *linear s2 s1* $f' \wedge (\forall x.\ f'\ (f\ x) = x) \wedge (\forall x.\ f\ (f'\ x) = x)$
⟨*proof*⟩

**lemma** *basis-to-basis-subspace-isomorphism*:
**assumes** *s*: *vs1.subspace S*
**and** *t*: *vs2.subspace T*
**and** *d*: *vs1.dim S* = *vs2.dim T*
**and** *B*: $B \subseteq S$ *vs1.independent B* $S \subseteq$ *vs1.span B card B* = *vs1.dim S*
**and** *C*: $C \subseteq T$ *vs2.independent C* $T \subseteq$ *vs2.span C card C* = *vs2.dim T*
**shows** $\exists f.$ *linear s1 s2 f* ∧ *f* ‘ *B* = *C* ∧ *f* ‘ *S* = *T* ∧ *inj-on f S*
⟨*proof*⟩

**lemma** *basis-change-exists'*:
**assumes** *vs1.independent B vs2.independent B'*
**assumes** *vs1.span B = UNIV vs2.span B' = UNIV vs1.dimension = vs2.dimension*
**shows** $\exists g.$ *linear s1 s2 g* $\wedge$ *bij g* $\wedge$ *bij-betw g B B'*
⟨*proof*⟩

**lemma** *basis-change-exists*:
**assumes** *vs1.dimension = vs2.dimension*
**shows** $\exists g.$ *linear s1 s2 g* $\wedge$ *bij g* $\wedge$ *bij-betw g B1 B2*
⟨*proof*⟩

**end**

**context** *finite-dimensional-vector-space* **begin**

**lemma** *linear-surj-imp-inj*:
**assumes** *lf*: *linear scale scale f* **and** *sf*: *surj f*
**shows** *inj f*
⟨*proof*⟩

**lemma** *linear-inverse-left*:
**assumes** *lf*: *linear scale scale f*
**and** *lf'*: *linear scale scale f'*
**shows** $f \circ f' = id \longleftrightarrow f' \circ f = id$
⟨*proof*⟩

**lemma** *left-inverse-linear*:
**assumes** *lf*: *linear scale scale f*
**and** *gf*: $g \circ f = id$
**shows** *linear scale scale g*
⟨*proof*⟩

**lemma** *inj-linear-imp-inv-linear*:
**assumes** *linear scale scale f inj f* **shows** *linear scale scale* (*inv f*)
⟨*proof*⟩

**lemma** *right-inverse-linear*:
**assumes** *lf*: *linear scale scale f*
**and** *gf*: $f \circ g = id$
**shows** *linear scale scale g*
⟨*proof*⟩

**lemma** *linear-independent-extend-inj*:
**assumes** *independent B independent* (*f ` B*) *inj-on f B*
**shows** $\exists g.$ *linear scale scale g* $\wedge$ *inj g* $\wedge$ ($\forall x \in B.\ g\ x = f\ x$)
⟨*proof*⟩

**end**

**context** *finite-dimensional-vector-space-pair* **begin**

**lemma** *subspace-isomorphism*:
  **assumes** *s*: *vs1.subspace S*
    **and** *t*: *vs2.subspace T*
    **and** *d*: *vs1.dim S = vs2.dim T*
  **shows** $\exists f.$ *linear s1 s2 f* $\wedge$ *f ' S = T* $\wedge$ *inj-on f S*
⟨*proof*⟩

**end**

**hide-const** (**open**) *linear*

**end**

# 107 Vector Spaces and Algebras over the Reals

**theory** *Real-Vector-Spaces*
**imports** *Real Topological-Spaces Vector-Spaces*
**begin**

## 107.1 Real vector spaces

**class** *scaleR* =
  **fixes** *scaleR* :: *real* ⇒ *'a* ⇒ *'a* (**infixr** ‹$*_R$› *75*)
**begin**

**abbreviation** *divideR* :: *'a* ⇒ *real* ⇒ *'a* (**infixl** ‹$'/_R$› *70*)
  **where** $x /_R r \equiv$ *inverse* $r *_R x$

**end**

**class** *real-vector* = *scaleR* + *ab-group-add* +
  **assumes** *scaleR-add-right*: $a *_R (x + y) = a *_R x + a *_R y$
  **and** *scaleR-add-left*: $(a + b) *_R x = a *_R x + b *_R x$
  **and** *scaleR-scaleR*: $a *_R b *_R x = (a * b) *_R x$
  **and** *scaleR-one*: $1 *_R x = x$

**class** *real-algebra* = *real-vector* + *ring* +
  **assumes** *mult-scaleR-left* [*simp*]: $a *_R x * y = a *_R (x * y)$
    **and** *mult-scaleR-right* [*simp*]: $x * a *_R y = a *_R (x * y)$

**class** *real-algebra-1* = *real-algebra* + *ring-1*

**class** *real-div-algebra* = *real-algebra-1* + *division-ring*

**class** *real-field* = *real-div-algebra* + *field*

**instantiation** *real* :: *real-field*

**begin**

**definition** *real-scaleR-def* [*simp*]: $scaleR\ a\ x = a * x$

**instance**
⟨*proof*⟩

**end**

**locale** *linear* = *Vector-Spaces.linear scaleR::-⇒-⇒'a::real-vector scaleR::-⇒-⇒'b::real-vector*
**begin**

**lemmas** *scaleR* = *scale*

**end**

**global-interpretation** *real-vector?*: *vector-space scaleR* :: *real* ⇒ *'a* ⇒ *'a* :: *real-vector*
**rewrites** *Vector-Spaces.linear* $(*_R)$ $(*_R)$ = *linear*
**and** *Vector-Spaces.linear* $(*)$ $(*_R)$ = *linear*
**defines** *dependent-raw-def*: *dependent* = *real-vector.dependent*
**and** *representation-raw-def*: *representation* = *real-vector.representation*
**and** *subspace-raw-def*: *subspace* = *real-vector.subspace*
**and** *span-raw-def*: *span* = *real-vector.span*
**and** *extend-basis-raw-def*: *extend-basis* = *real-vector.extend-basis*
**and** *dim-raw-def*: *dim* = *real-vector.dim*
⟨*proof*⟩

**hide-const** (**open**)— locale constants
*real-vector.dependent*
*real-vector.independent*
*real-vector.representation*
*real-vector.subspace*
*real-vector.span*
*real-vector.extend-basis*
*real-vector.dim*

**abbreviation** *independent x* ≡ ¬ *dependent x*

**global-interpretation** *real-vector?*: *vector-space-pair scaleR::-⇒-⇒'a::real-vector scaleR::-⇒-⇒'b::real-vector*
**rewrites** *Vector-Spaces.linear* $(*_R)$ $(*_R)$ = *linear*
**and** *Vector-Spaces.linear* $(*)$ $(*_R)$ = *linear*
**defines** *construct-raw-def*: *construct* = *real-vector.construct*
⟨*proof*⟩

**hide-const** (**open**)— locale constants
*real-vector.construct*

**lemma** *linear-compose*: *linear f* ⟹ *linear g* ⟹ *linear* $(g \circ f)$

⟨*proof*⟩

Recover original theorem names

**lemmas** *scaleR-left-commute* = *real-vector.scale-left-commute*
**lemmas** *scaleR-zero-left* = *real-vector.scale-zero-left*
**lemmas** *scaleR-minus-left* = *real-vector.scale-minus-left*
**lemmas** *scaleR-diff-left* = *real-vector.scale-left-diff-distrib*
**lemmas** *scaleR-sum-left* = *real-vector.scale-sum-left*
**lemmas** *scaleR-zero-right* = *real-vector.scale-zero-right*
**lemmas** *scaleR-minus-right* = *real-vector.scale-minus-right*
**lemmas** *scaleR-diff-right* = *real-vector.scale-right-diff-distrib*
**lemmas** *scaleR-sum-right* = *real-vector.scale-sum-right*
**lemmas** *scaleR-eq-0-iff* = *real-vector.scale-eq-0-iff*
**lemmas** *scaleR-left-imp-eq* = *real-vector.scale-left-imp-eq*
**lemmas** *scaleR-right-imp-eq* = *real-vector.scale-right-imp-eq*
**lemmas** *scaleR-cancel-left* = *real-vector.scale-cancel-left*
**lemmas** *scaleR-cancel-right* = *real-vector.scale-cancel-right*

**lemma** [*field-simps*]:
$c \neq 0 \Longrightarrow a = b /_R c \longleftrightarrow c *_R a = b$
$c \neq 0 \Longrightarrow b /_R c = a \longleftrightarrow b = c *_R a$
$c \neq 0 \Longrightarrow a + b /_R c = (c *_R a + b) /_R c$
$c \neq 0 \Longrightarrow a /_R c + b = (a + c *_R b) /_R c$
$c \neq 0 \Longrightarrow a - b /_R c = (c *_R a - b) /_R c$
$c \neq 0 \Longrightarrow a /_R c - b = (a - c *_R b) /_R c$
$c \neq 0 \Longrightarrow - (a /_R c) + b = (- a + c *_R b) /_R c$
$c \neq 0 \Longrightarrow - (a /_R c) - b = (- a - c *_R b) /_R c$
**for** *a b* :: *'a* :: *real-vector*
⟨*proof*⟩

Legacy names

**lemmas** *scaleR-left-distrib* = *scaleR-add-left*
**lemmas** *scaleR-right-distrib* = *scaleR-add-right*
**lemmas** *scaleR-left-diff-distrib* = *scaleR-diff-left*
**lemmas** *scaleR-right-diff-distrib* = *scaleR-diff-right*

**lemmas** *linear-injective-0* = *linear-inj-iff-eq-0*
**and** *linear-injective-on-subspace-0* = *linear-inj-on-iff-eq-0*
**and** *linear-cmul* = *linear-scale*
**and** *linear-scaleR* = *linear-scale-self*
**and** *subspace-mul* = *subspace-scale*
**and** *span-linear-image* = *linear-span-image*
**and** *span-0* = *span-zero*
**and** *span-mul* = *span-scale*
**and** *injective-scaleR* = *injective-scale*

**lemma** *scaleR-minus1-left* [*simp*]: *scaleR* $(-1)$ $x = - x$
**for** *x* :: *'a::real-vector*
⟨*proof*⟩

**lemma** *scaleR-2*:
  **fixes** $x :: 'a::real\text{-}vector$
  **shows** $scaleR\ 2\ x = x + x$
  ⟨*proof*⟩

**lemma** *scaleR-half-double* [*simp*]:
  **fixes** $a :: 'a::real\text{-}vector$
  **shows** $(1\ /\ 2) *_R (a + a) = a$
⟨*proof*⟩

**lemma** *shift-zero-ident* [*simp*]:
  **fixes** $f :: 'a \Rightarrow 'b::real\text{-}vector$
  **shows** $(+)0 \circ f = f$
  ⟨*proof*⟩

**lemma** *linear-scale-real*:
  **fixes** $r::real$ **shows** $linear\ f \Longrightarrow f\ (r * b) = r * f\ b$
  ⟨*proof*⟩

**interpretation** *scaleR-left*: $additive\ (\lambda a.\ scaleR\ a\ x :: 'a::real\text{-}vector)$
  ⟨*proof*⟩

**interpretation** *scaleR-right*: $additive\ (\lambda x.\ scaleR\ a\ x :: 'a::real\text{-}vector)$
  ⟨*proof*⟩

**lemma** *nonzero-inverse-scaleR-distrib*:
  $a \neq 0 \Longrightarrow x \neq 0 \Longrightarrow inverse\ (scaleR\ a\ x) = scaleR\ (inverse\ a)\ (inverse\ x)$
  **for** $x :: 'a::real\text{-}div\text{-}algebra$
  ⟨*proof*⟩

**lemma** *inverse-scaleR-distrib*: $inverse\ (scaleR\ a\ x) = scaleR\ (inverse\ a)\ (inverse\ x)$
  **for** $x :: 'a::\{real\text{-}div\text{-}algebra,division\text{-}ring\}$
  ⟨*proof*⟩

**lemmas** *sum-constant-scaleR* = *real-vector.sum-constant-scale*— legacy name

**named-theorems** *vector-add-divide-simps to simplify sums of scaled vectors*

**lemma** [*vector-add-divide-simps*]:
  $v + (b\ /\ z) *_R w = (if\ z = 0\ then\ v\ else\ (z *_R v + b *_R w)\ /_R\ z)$
  $a *_R v + (b\ /\ z) *_R w = (if\ z = 0\ then\ a *_R v\ else\ ((a * z) *_R v + b *_R w)\ /_R\ z)$
  $(a\ /\ z) *_R v + w = (if\ z = 0\ then\ w\ else\ (a *_R v + z *_R w)\ /_R\ z)$
  $(a\ /\ z) *_R v + b *_R w = (if\ z = 0\ then\ b *_R w\ else\ (a *_R v + (b * z) *_R w)\ /_R\ z)$
  $v - (b\ /\ z) *_R w = (if\ z = 0\ then\ v\ else\ (z *_R v - b *_R w)\ /_R\ z)$
  $a *_R v - (b\ /\ z) *_R w = (if\ z = 0\ then\ a *_R v\ else\ ((a * z) *_R v - b *_R w)\ /_R$

$z$)
$(a / z) *_R v - w = (\mathit{if}\ z = 0\ \mathit{then}\ -w\ \mathit{else}\ (a *_R v - z *_R w) /_R z)$
$(a / z) *_R v - b *_R w = (\mathit{if}\ z = 0\ \mathit{then}\ -b *_R w\ \mathit{else}\ (a *_R v - (b * z) *_R w) /_R z)$
**for** $v :: 'a :: \mathit{real\text{-}vector}$
⟨*proof*⟩

**lemma** *eq-vector-fraction-iff* [*vector-add-divide-simps*]:
**fixes** $x :: 'a :: \mathit{real\text{-}vector}$
**shows** $(x = (u / v) *_R a) \longleftrightarrow (\mathit{if}\ v=0\ \mathit{then}\ x = 0\ \mathit{else}\ v *_R x = u *_R a)$
⟨*proof*⟩

**lemma** *vector-fraction-eq-iff* [*vector-add-divide-simps*]:
**fixes** $x :: 'a :: \mathit{real\text{-}vector}$
**shows** $((u / v) *_R a = x) \longleftrightarrow (\mathit{if}\ v=0\ \mathit{then}\ x = 0\ \mathit{else}\ u *_R a = v *_R x)$
⟨*proof*⟩

**lemma** *real-vector-affinity-eq*:
**fixes** $x :: 'a :: \mathit{real\text{-}vector}$
**assumes** *m0*: $m \neq 0$
**shows** $m *_R x + c = y \longleftrightarrow x = \mathit{inverse}\ m *_R y - (\mathit{inverse}\ m *_R c)$
(**is** *?lhs* ⟷ *?rhs*)
⟨*proof*⟩

**lemma** *real-vector-eq-affinity*: $m \neq 0 \Longrightarrow y = m *_R x + c \longleftrightarrow \mathit{inverse}\ m *_R y - (\mathit{inverse}\ m *_R c) = x$
**for** $x :: 'a::\mathit{real\text{-}vector}$
⟨*proof*⟩

**lemma** *scaleR-eq-iff* [*simp*]: $b + u *_R a = a + u *_R b \longleftrightarrow a = b \lor u = 1$
**for** $a :: 'a::\mathit{real\text{-}vector}$
⟨*proof*⟩

**lemma** *scaleR-collapse* [*simp*]: $(1 - u) *_R a + u *_R a = a$
**for** $a :: 'a::\mathit{real\text{-}vector}$
⟨*proof*⟩

## 107.2 Embedding of the Reals into any *real-algebra-1*: *of-real*

**definition** *of-real* :: *real* ⇒ *'a::real-algebra-1*
**where** *of-real r = scaleR r 1*

**lemma** *scaleR-conv-of-real*: *scaleR r x = of-real r ∗ x*
⟨*proof*⟩

**lemma** *of-real-0* [*simp*]: *of-real 0 = 0*
⟨*proof*⟩

**lemma** *of-real-1* [*simp*]: *of-real 1 = 1*
⟨*proof*⟩

**lemma** *of-real-add* [*simp*]: *of-real* (*x* + *y*) = *of-real x* + *of-real y*
⟨*proof*⟩

**lemma** *of-real-minus* [*simp*]: *of-real* (− *x*) = − *of-real x*
⟨*proof*⟩

**lemma** *of-real-diff* [*simp*]: *of-real* (*x* − *y*) = *of-real x* − *of-real y*
⟨*proof*⟩

**lemma** *of-real-mult* [*simp*]: *of-real* (*x* ∗ *y*) = *of-real x* ∗ *of-real y*
⟨*proof*⟩

**lemma** *of-real-sum*[*simp*]: *of-real* (*sum f s*) = ($\sum$ *x*∈*s*. *of-real* (*f x*))
⟨*proof*⟩

**lemma** *of-real-prod*[*simp*]: *of-real* (*prod f s*) = ($\prod$ *x*∈*s*. *of-real* (*f x*))
⟨*proof*⟩

**lemma** *sum-list-of-real*: *sum-list* (*map of-real xs*) = *of-real* (*sum-list xs*)
⟨*proof*⟩

**lemma** *nonzero-of-real-inverse*:
*x* ≠ *0* ⟹ *of-real* (*inverse x*) = *inverse* (*of-real x* :: ′*a*::*real-div-algebra*)
⟨*proof*⟩

**lemma** *of-real-inverse* [*simp*]:
*of-real* (*inverse x*) = *inverse* (*of-real x* :: ′*a*::{*real-div-algebra*,*division-ring*})
⟨*proof*⟩

**lemma** *nonzero-of-real-divide*:
*y* ≠ *0* ⟹ *of-real* (*x* / *y*) = (*of-real x* / *of-real y* :: ′*a*::*real-field*)
⟨*proof*⟩

**lemma** *of-real-divide* [*simp*]:
*of-real* (*x* / *y*) = (*of-real x* / *of-real y* :: ′*a*::*real-div-algebra*)
⟨*proof*⟩

**lemma** *of-real-power* [*simp*]:
*of-real* (*x* ^ *n*) = (*of-real x* :: ′*a*::{*real-algebra-1*}) ^ *n*
⟨*proof*⟩

**lemma** *of-real-power-int* [*simp*]:
*of-real* (*power-int x n*) = *power-int* (*of-real x* :: ′*a* :: {*real-div-algebra*,*division-ring*})
*n*
⟨*proof*⟩

**lemma** *of-real-eq-iff* [*simp*]: *of-real* $x$ = *of-real* $y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *inj-of-real*: *inj of-real*
⟨*proof*⟩

**lemmas** *of-real-eq-0-iff* [*simp*] = *of-real-eq-iff* [*of - 0*, *simplified*]
**lemmas** *of-real-eq-1-iff* [*simp*] = *of-real-eq-iff* [*of - 1*, *simplified*]

**lemma** *minus-of-real-eq-of-real-iff* [*simp*]: $-$*of-real* $x$ = *of-real* $y \longleftrightarrow -x = y$
⟨*proof*⟩

**lemma** *of-real-eq-minus-of-real-iff* [*simp*]: *of-real* $x$ = $-$*of-real* $y \longleftrightarrow x = -y$
⟨*proof*⟩

**lemma** *of-real-eq-id* [*simp*]: *of-real* = (*id* :: *real* ⇒ *real*)
⟨*proof*⟩

Collapse nested embeddings.

**lemma** *of-real-of-nat-eq* [*simp*]: *of-real* (*of-nat* $n$) = *of-nat* $n$
⟨*proof*⟩

**lemma** *of-real-of-int-eq* [*simp*]: *of-real* (*of-int* $z$) = *of-int* $z$
⟨*proof*⟩

**lemma** *of-real-numeral* [*simp*]: *of-real* (*numeral* $w$) = *numeral* $w$
⟨*proof*⟩

**lemma** *of-real-neg-numeral* [*simp*]: *of-real* ($-$ *numeral* $w$) = $-$ *numeral* $w$
⟨*proof*⟩

**lemma** *numeral-power-int-eq-of-real-cancel-iff* [*simp*]:
*power-int* (*numeral* $x$) $n$ = (*of-real* $y$ :: *'a* :: {*real-div-algebra*, *division-ring*}) ⟷
*power-int* (*numeral* $x$) $n = y$
⟨*proof*⟩

**lemma** *of-real-eq-numeral-power-int-cancel-iff* [*simp*]:
(*of-real* $y$ :: *'a* :: {*real-div-algebra*, *division-ring*}) = *power-int* (*numeral* $x$) $n$ ⟷
$y$ = *power-int* (*numeral* $x$) $n$
⟨*proof*⟩

**lemma** *of-real-eq-of-real-power-int-cancel-iff* [*simp*]:
*power-int* (*of-real* $b$ :: *'a* :: {*real-div-algebra*, *division-ring*}) $w$ = *of-real* $x$ ⟷
*power-int* $b$ $w = x$
⟨*proof*⟩

**lemma** *of-real-in-Ints-iff* [*simp*]: *of-real* $x \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *Ints-of-real* [*intro*]: $x \in \mathbb{Z} \Longrightarrow$ *of-real* $x \in \mathbb{Z}$
⟨*proof*⟩

Every real algebra has characteristic zero.

**instance** *real-algebra-1* < *ring-char-0*
⟨*proof*⟩

**lemma** *fraction-scaleR-times* [*simp*]:
**fixes** $a$ :: *'a::real-algebra-1*
**shows** (*numeral u* / *numeral v*) $*_R$ (*numeral w* $*$ $a$) = (*numeral u* $*$ *numeral w* / *numeral v*) $*_R$ $a$
⟨*proof*⟩

**lemma** *inverse-scaleR-times* [*simp*]:
**fixes** $a$ :: *'a::real-algebra-1*
**shows** (*1* / *numeral v*) $*_R$ (*numeral w* $*$ $a$) = (*numeral w* / *numeral v*) $*_R$ $a$
⟨*proof*⟩

**lemma** *scaleR-times* [*simp*]:
**fixes** $a$ :: *'a::real-algebra-1*
**shows** (*numeral u*) $*_R$ (*numeral w* $*$ $a$) = (*numeral u* $*$ *numeral w*) $*_R$ $a$
⟨*proof*⟩

**instance** *real-field* < *field-char-0* ⟨*proof*⟩

## 107.3 The Set of Real Numbers

**definition** *Reals* :: *'a::real-algebra-1 set* (‹$\mathbb{R}$›)
**where** $\mathbb{R}$ = *range of-real*

**lemma** *Reals-of-real* [*simp*]: *of-real* $r \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-of-int* [*simp*]: *of-int* $z \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-of-nat* [*simp*]: *of-nat* $n \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-numeral* [*simp*]: *numeral* $w \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-0* [*simp*]: $0 \in \mathbb{R}$ **and** *Reals-1* [*simp*]: $1 \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-add* [*simp*]: $a \in \mathbb{R} \Longrightarrow b \in \mathbb{R} \Longrightarrow a + b \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-minus* [*simp*]: $a \in \mathbb{R} \Longrightarrow - a \in \mathbb{R}$

⟨*proof*⟩

**lemma** *Reals-minus-iff* [*simp*]: $- a \in \mathbb{R} \longleftrightarrow a \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-diff* [*simp*]: $a \in \mathbb{R} \Longrightarrow b \in \mathbb{R} \Longrightarrow a - b \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-mult* [*simp*]: $a \in \mathbb{R} \Longrightarrow b \in \mathbb{R} \Longrightarrow a * b \in \mathbb{R}$
⟨*proof*⟩

**lemma** *nonzero-Reals-inverse*: $a \in \mathbb{R} \Longrightarrow a \neq 0 \Longrightarrow$ *inverse* $a \in \mathbb{R}$
**for** $a$ :: *'a::real-div-algebra*
⟨*proof*⟩

**lemma** *Reals-inverse*: $a \in \mathbb{R} \Longrightarrow$ *inverse* $a \in \mathbb{R}$
**for** $a$ :: *'a::{real-div-algebra,division-ring}*
⟨*proof*⟩

**lemma** *Reals-inverse-iff* [*simp*]: *inverse* $x \in \mathbb{R} \longleftrightarrow x \in \mathbb{R}$
**for** $x$ :: *'a::{real-div-algebra,division-ring}*
⟨*proof*⟩

**lemma** *nonzero-Reals-divide*: $a \in \mathbb{R} \Longrightarrow b \in \mathbb{R} \Longrightarrow b \neq 0 \Longrightarrow a / b \in \mathbb{R}$
**for** $a\ b$ :: *'a::real-field*
⟨*proof*⟩

**lemma** *Reals-divide* [*simp*]: $a \in \mathbb{R} \Longrightarrow b \in \mathbb{R} \Longrightarrow a / b \in \mathbb{R}$
**for** $a\ b$ :: *'a::{real-field,field}*
⟨*proof*⟩

**lemma** *Reals-power* [*simp*]: $a \in \mathbb{R} \Longrightarrow a\ \hat{}\ n \in \mathbb{R}$
**for** $a$ :: *'a::real-algebra-1*
⟨*proof*⟩

**lemma** *Reals-cases* [*cases set*: *Reals*]:
**assumes** $q \in \mathbb{R}$
**obtains** (*of-real*) $r$ **where** $q =$ *of-real* $r$
⟨*proof*⟩

**lemma** *sum-in-Reals* [*intro,simp*]: $(\bigwedge i.\ i \in s \Longrightarrow f\ i \in \mathbb{R}) \Longrightarrow$ *sum* $f\ s \in \mathbb{R}$
⟨*proof*⟩

**lemma** *prod-in-Reals* [*intro,simp*]: $(\bigwedge i.\ i \in s \Longrightarrow f\ i \in \mathbb{R}) \Longrightarrow$ *prod* $f\ s \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-induct* [*case-names of-real*, *induct set*: *Reals*]:
$q \in \mathbb{R} \Longrightarrow (\bigwedge r.\ P\ (\textit{of-real}\ r)) \Longrightarrow P\ q$
⟨*proof*⟩

## 107.4 Ordered real vector spaces

**class** *ordered-real-vector* = *real-vector* + *ordered-ab-group-add* +
**assumes** *scaleR-left-mono*: $x \leq y \Longrightarrow 0 \leq a \Longrightarrow a *_R x \leq a *_R y$
**and** *scaleR-right-mono*: $a \leq b \Longrightarrow 0 \leq x \Longrightarrow a *_R x \leq b *_R x$
**begin**

**lemma** *scaleR-mono*:
$a \leq b \Longrightarrow x \leq y \Longrightarrow 0 \leq b \Longrightarrow 0 \leq x \Longrightarrow a *_R x \leq b *_R y$
⟨*proof*⟩

**lemma** *scaleR-mono'*:
$a \leq b \Longrightarrow c \leq d \Longrightarrow 0 \leq a \Longrightarrow 0 \leq c \Longrightarrow a *_R c \leq b *_R d$
⟨*proof*⟩

**lemma** *pos-le-divideR-eq* [*field-simps*]:
$a \leq b /_R c \longleftrightarrow c *_R a \leq b$ (**is** $?P \longleftrightarrow ?Q$) **if** $0 < c$
⟨*proof*⟩

**lemma** *pos-less-divideR-eq* [*field-simps*]:
$a < b /_R c \longleftrightarrow c *_R a < b$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-divideR-le-eq* [*field-simps*]:
$b /_R c \leq a \longleftrightarrow b \leq c *_R a$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-divideR-less-eq* [*field-simps*]:
$b /_R c < a \longleftrightarrow b < c *_R a$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-le-minus-divideR-eq* [*field-simps*]:
$a \leq -(b /_R c) \longleftrightarrow c *_R a \leq -b$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-less-minus-divideR-eq* [*field-simps*]:
$a < -(b /_R c) \longleftrightarrow c *_R a < -b$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-minus-divideR-le-eq* [*field-simps*]:
$-(b /_R c) \leq a \longleftrightarrow -b \leq c *_R a$ **if** $c > 0$
⟨*proof*⟩

**lemma** *pos-minus-divideR-less-eq* [*field-simps*]:
$-(b /_R c) < a \longleftrightarrow -b < c *_R a$ **if** $c > 0$
⟨*proof*⟩

**lemma** *scaleR-image-atLeastAtMost*: $c > 0 \Longrightarrow scaleR\ c\ `\ \{x..y\} = \{c *_R x..c *_R y\}$
⟨*proof*⟩

**end**

**lemma** *neg-le-divideR-eq* [*field-simps*]:
$a \leq b /_R\ c \longleftrightarrow b \leq c *_R\ a$ (**is** *?P* $\longleftrightarrow$ *?Q*) **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-less-divideR-eq* [*field-simps*]:
$a < b /_R\ c \longleftrightarrow b < c *_R\ a$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-divideR-le-eq* [*field-simps*]:
$b /_R\ c \leq a \longleftrightarrow c *_R\ a \leq b$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-divideR-less-eq* [*field-simps*]:
$b /_R\ c < a \longleftrightarrow c *_R\ a < b$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-le-minus-divideR-eq* [*field-simps*]:
$a \leq -\ (b /_R\ c) \longleftrightarrow -\ b \leq c *_R\ a$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-less-minus-divideR-eq* [*field-simps*]:
$a < -\ (b /_R\ c) \longleftrightarrow -\ b < c *_R\ a$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-minus-divideR-le-eq* [*field-simps*]:
$-\ (b /_R\ c) \leq a \longleftrightarrow c *_R\ a \leq -\ b$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** *neg-minus-divideR-less-eq* [*field-simps*]:
$-\ (b /_R\ c) < a \longleftrightarrow c *_R\ a < -\ b$ **if** $c < 0$
**for** $a\ b$ :: $'a$ :: *ordered-real-vector*
⟨*proof*⟩

**lemma** [*field-split-simps*]:
$a = b /_R\ c \longleftrightarrow$ (*if* $c = 0$ *then* $a = 0$ *else* $c *_R\ a = b$)
$b /_R\ c = a \longleftrightarrow$ (*if* $c = 0$ *then* $a = 0$ *else* $b = c *_R\ a$)
$a + b /_R\ c =$ (*if* $c = 0$ *then* $a$ *else* $(c *_R\ a + b) /_R\ c$)
$a /_R\ c + b =$ (*if* $c = 0$ *then* $b$ *else* $(a + c *_R\ b) /_R\ c$)
$a - b /_R\ c =$ (*if* $c = 0$ *then* $a$ *else* $(c *_R\ a - b) /_R\ c$)

$a /_R c - b = (if\ c = 0\ then - b\ else\ (a - c *_R b) /_R c)$
$- (a /_R c) + b = (if\ c = 0\ then\ b\ else\ (- a + c *_R b) /_R c)$
$- (a /_R c) - b = (if\ c = 0\ then - b\ else\ (- a - c *_R b) /_R c)$
**for** $a\ b :: 'a :: real\text{-}vector$
⟨*proof*⟩

**lemma** [*field-split-simps*]:
$0 < c \Longrightarrow a \le b /_R c \longleftrightarrow (if\ c > 0\ then\ c *_R a \le b\ else\ if\ c < 0\ then\ b \le c *_R a\ else\ a \le 0)$
$0 < c \Longrightarrow a < b /_R c \longleftrightarrow (if\ c > 0\ then\ c *_R a < b\ else\ if\ c < 0\ then\ b < c *_R a\ else\ a < 0)$
$0 < c \Longrightarrow b /_R c \le a \longleftrightarrow (if\ c > 0\ then\ b \le c *_R a\ else\ if\ c < 0\ then\ c *_R a \le b\ else\ a \ge 0)$
$0 < c \Longrightarrow b /_R c < a \longleftrightarrow (if\ c > 0\ then\ b < c *_R a\ else\ if\ c < 0\ then\ c *_R a < b\ else\ a > 0)$
$0 < c \Longrightarrow a \le - (b /_R c) \longleftrightarrow (if\ c > 0\ then\ c *_R a \le - b\ else\ if\ c < 0\ then - b \le c *_R a\ else\ a \le 0)$
$0 < c \Longrightarrow a < - (b /_R c) \longleftrightarrow (if\ c > 0\ then\ c *_R a < - b\ else\ if\ c < 0\ then - b < c *_R a\ else\ a < 0)$
$0 < c \Longrightarrow - (b /_R c) \le a \longleftrightarrow (if\ c > 0\ then - b \le c *_R a\ else\ if\ c < 0\ then\ c *_R a \le - b\ else\ a \ge 0)$
$0 < c \Longrightarrow - (b /_R c) < a \longleftrightarrow (if\ c > 0\ then - b < c *_R a\ else\ if\ c < 0\ then\ c *_R a < - b\ else\ a > 0)$
**for** $a\ b :: 'a :: ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *scaleR-nonneg-nonneg*: $0 \le a \Longrightarrow 0 \le x \Longrightarrow 0 \le a *_R x$
**for** $x :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *scaleR-nonneg-nonpos*: $0 \le a \Longrightarrow x \le 0 \Longrightarrow a *_R x \le 0$
**for** $x :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *scaleR-nonpos-nonneg*: $a \le 0 \Longrightarrow 0 \le x \Longrightarrow a *_R x \le 0$
**for** $x :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *split-scaleR-neg-le*: $(0 \le a \wedge x \le 0) \vee (a \le 0 \wedge 0 \le x) \Longrightarrow a *_R x \le 0$
**for** $x :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *le-add-iff1*: $a *_R e + c \le b *_R e + d \longleftrightarrow (a - b) *_R e + c \le d$
**for** $c\ d\ e :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *le-add-iff2*: $a *_R e + c \le b *_R e + d \longleftrightarrow c \le (b - a) *_R e + d$
**for** $c\ d\ e :: 'a::ordered\text{-}real\text{-}vector$
⟨*proof*⟩

**lemma** *scaleR-left-mono-neg*: $b \leq a \Longrightarrow c \leq 0 \Longrightarrow c *_R a \leq c *_R b$
**for** $a\ b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-right-mono-neg*: $b \leq a \Longrightarrow c \leq 0 \Longrightarrow a *_R c \leq b *_R c$
**for** $c$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-nonpos-nonpos*: $a \leq 0 \Longrightarrow b \leq 0 \Longrightarrow 0 \leq a *_R b$
**for** $b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *split-scaleR-pos-le*: $(0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0) \Longrightarrow 0 \leq a *_R b$
**for** $b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *zero-le-scaleR-iff*:
**fixes** $b$ :: $'a$::*ordered-real-vector*
**shows** $0 \leq a *_R b \longleftrightarrow 0 < a \wedge 0 \leq b \vee a < 0 \wedge b \leq 0 \vee a = 0$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *scaleR-le-0-iff*: $a *_R b \leq 0 \longleftrightarrow 0 < a \wedge b \leq 0 \vee a < 0 \wedge 0 \leq b \vee a = 0$
**for** $b$::$'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-le-cancel-left*: $c *_R a \leq c *_R b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
**for** $b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-le-cancel-left-pos*: $0 < c \Longrightarrow c *_R a \leq c *_R b \longleftrightarrow a \leq b$
**for** $b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-le-cancel-left-neg*: $c < 0 \Longrightarrow c *_R a \leq c *_R b \longleftrightarrow b \leq a$
**for** $b$ :: $'a$::*ordered-real-vector*
⟨*proof*⟩

**lemma** *scaleR-left-le-one-le*: $0 \leq x \Longrightarrow a \leq 1 \Longrightarrow a *_R x \leq x$
**for** $x$ :: $'a$::*ordered-real-vector* **and** $a$ :: *real*
⟨*proof*⟩

## 107.5 Real normed vector spaces

**class** *dist* =
**fixes** *dist* :: $'a \Rightarrow 'a \Rightarrow$ *real*

**class** *norm* =
**fixes** *norm* :: $'a \Rightarrow$ *real*

**class** *sgn-div-norm* = *scaleR* + *norm* + *sgn* +
**assumes** *sgn-div-norm*: *sgn* $x = x /_R$ *norm* $x$

**class** *dist-norm* = *dist* + *norm* + *minus* +
**assumes** *dist-norm*: *dist* $x$ $y$ = *norm* $(x - y)$

**class** *uniformity-dist* = *dist* + *uniformity* +
**assumes** *uniformity-dist*: *uniformity* = (*INF* $e \in \{0 <..\}$. *principal* $\{(x, y).$ *dist* $x$ $y < e\}$)
**begin**

**lemma** *eventually-uniformity-metric*:
*eventually* $P$ *uniformity* $\longleftrightarrow (\exists e>0. \forall x\ y.$ *dist* $x\ y < e \longrightarrow P\ (x, y))$
$\langle proof \rangle$

**end**

**class** *real-normed-vector* = *real-vector* + *sgn-div-norm* + *dist-norm* + *uniformity-dist* + *open-uniformity* +
**assumes** *norm-eq-zero* [*simp*]: *norm* $x = 0 \longleftrightarrow x = 0$
**and** *norm-triangle-ineq*: *norm* $(x + y) \leq$ *norm* $x$ + *norm* $y$
**and** *norm-scaleR* [*simp*]: *norm* (*scaleR* $a$ $x$) = $|a| *$ *norm* $x$
**begin**

**lemma** *norm-ge-zero* [*simp*]: $0 \leq$ *norm* $x$
$\langle proof \rangle$

**lemma** *bdd-below-norm-image*: *bdd-below* (*norm* ' $A$)
$\langle proof \rangle$

**end**

**class** *real-normed-algebra* = *real-algebra* + *real-normed-vector* +
**assumes** *norm-mult-ineq*: *norm* $(x * y) \leq$ *norm* $x *$ *norm* $y$

**class** *real-normed-algebra-1* = *real-algebra-1* + *real-normed-algebra* +
**assumes** *norm-one* [*simp*]: *norm* $1 = 1$

**lemma** (**in** *real-normed-algebra-1*) *scaleR-power* [*simp*]: (*scaleR* $x$ $y$) $\hat{}\ n$ = *scaleR* $(x \hat{} n)$ $(y \hat{} n)$
$\langle proof \rangle$

**class** *real-normed-div-algebra* = *real-div-algebra* + *real-normed-vector* +
**assumes** *norm-mult*: *norm* $(x * y)$ = *norm* $x *$ *norm* $y$

**lemma** *divideR-right*:

**fixes** $x\ y :: 'a::real\text{-}normed\text{-}vector$
**shows** $r \neq 0 \Longrightarrow y = x\ /_R\ r \longleftrightarrow r *_R\ y = x$
⟨proof⟩

**class** *real-normed-field* = *real-field* + *real-normed-div-algebra*

**lemma** *dist-mult-left*:
$dist\ (a * b)\ (a * c :: 'a :: real\text{-}normed\text{-}field) = norm\ a * dist\ b\ c$
⟨proof⟩

**lemma** *dist-mult-right*:
$dist\ (b * a)\ (c * a :: 'a :: real\text{-}normed\text{-}field) = norm\ a * dist\ b\ c$
⟨proof⟩

**instance** *real-normed-div-algebra* < *real-normed-algebra-1*
⟨proof⟩

**context** *real-normed-vector* **begin**

**lemma** *norm-zero* [*simp*]: $norm\ (0::'a) = 0$
⟨proof⟩

**lemma** *zero-less-norm-iff* [*simp*]: $norm\ x > 0 \longleftrightarrow x \neq 0$
⟨proof⟩

**lemma** *norm-not-less-zero* [*simp*]: $\neg\ norm\ x < 0$
⟨proof⟩

**lemma** *norm-le-zero-iff* [*simp*]: $norm\ x \leq 0 \longleftrightarrow x = 0$
⟨proof⟩

**lemma** *norm-minus-cancel* [*simp*]: $norm\ (-\ x) = norm\ x$
⟨proof⟩

**lemma** *norm-minus-commute*: $norm\ (a - b) = norm\ (b - a)$
⟨proof⟩

**lemma** *dist-add-cancel* [*simp*]: $dist\ (a + b)\ (a + c) = dist\ b\ c$
⟨proof⟩

**lemma** *dist-add-cancel2* [*simp*]: $dist\ (b + a)\ (c + a) = dist\ b\ c$
⟨proof⟩

**lemma** *norm-uminus-minus*: $norm\ (-\ x - y) = norm\ (x + y)$
⟨proof⟩

**lemma** *norm-triangle-ineq2*: $norm\ a - norm\ b \leq norm\ (a - b)$
⟨proof⟩

**lemma** *norm-triangle-ineq3*: $|norm\ a - norm\ b| \le norm\ (a - b)$
⟨*proof*⟩

**lemma** *norm-triangle-ineq4*: $norm\ (a - b) \le norm\ a + norm\ b$
⟨*proof*⟩

**lemma** *norm-triangle-le-diff*: $norm\ x + norm\ y \le e \Longrightarrow norm\ (x - y) \le e$
⟨*proof*⟩

**lemma** *norm-diff-ineq*: $norm\ a - norm\ b \le norm\ (a + b)$
⟨*proof*⟩

**lemma** *norm-triangle-sub*: $norm\ x \le norm\ y + norm\ (x - y)$
⟨*proof*⟩

**lemma** *norm-triangle-le*: $norm\ x + norm\ y \le e \Longrightarrow norm\ (x + y) \le e$
⟨*proof*⟩

**lemma** *norm-triangle-lt*: $norm\ x + norm\ y < e \Longrightarrow norm\ (x + y) < e$
⟨*proof*⟩

**lemma** *norm-add-leD*: $norm\ (a + b) \le c \Longrightarrow norm\ b \le norm\ a + c$
⟨*proof*⟩

**lemma** *norm-diff-triangle-ineq*: $norm\ ((a + b) - (c + d)) \le norm\ (a - c) + norm\ (b - d)$
⟨*proof*⟩

**lemma** *norm-diff-triangle-le*: $norm\ (x - z) \le e1 + e2$
**if** $norm\ (x - y) \le e1 \quad norm\ (y - z) \le e2$
⟨*proof*⟩

**lemma** *norm-diff-triangle-less*: $norm\ (x - z) < e1 + e2$
**if** $norm\ (x - y) < e1 \quad norm\ (y - z) < e2$
⟨*proof*⟩

**lemma** *norm-triangle-mono*:
$norm\ a \le r \Longrightarrow norm\ b \le s \Longrightarrow norm\ (a + b) \le r + s$
⟨*proof*⟩

**lemma** *norm-sum*: $norm\ (sum\ f\ A) \le (\sum i{\in}A.\ norm\ (f\ i))$
**for** $f::'b \Rightarrow 'a$
⟨*proof*⟩

**lemma** *sum-norm-le*: $norm\ (sum\ f\ S) \le sum\ g\ S$
**if** $\bigwedge x.\ x \in S \Longrightarrow norm\ (f\ x) \le g\ x$
**for** $f::'b \Rightarrow 'a$
⟨*proof*⟩

**lemma** *abs-norm-cancel* [*simp*]: $|norm\ a| = norm\ a$
⟨*proof*⟩

**lemma** *sum-norm-bound*:
*norm* (*sum f S*) $\leq$ *of-nat* (*card S*)$*K$
**if** $\bigwedge x.\ x \in S \Longrightarrow norm\ (f\ x) \leq K$
**for** $f :: 'b \Rightarrow 'a$
⟨*proof*⟩

**lemma** *norm-add-less*: $norm\ x < r \Longrightarrow norm\ y < s \Longrightarrow norm\ (x + y) < r + s$
⟨*proof*⟩

**end**

**lemma** *dist-sum-le*:
**fixes** $f :: 'a \Rightarrow 'b :: real\text{-}normed\text{-}vector$
**shows** $dist\ (\sum x{\in}A.\ f\ x)\ (\sum x{\in}A.\ g\ x) \leq (\sum x{\in}A.\ dist\ (f\ x)\ (g\ x))$
⟨*proof*⟩

**lemma** *dist-scaleR* [*simp*]: $dist\ (x *_R a)\ (y *_R a) = |x - y| * norm\ a$
**for** $a :: 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *norm-mult-less*: $norm\ x < r \Longrightarrow norm\ y < s \Longrightarrow norm\ (x * y) < r * s$
**for** $x\ y :: 'a::real\text{-}normed\text{-}algebra$
⟨*proof*⟩

**lemma** *norm-of-real* [*simp*]: $norm\ (of\text{-}real\ r :: 'a::real\text{-}normed\text{-}algebra\text{-}1) = |r|$
⟨*proof*⟩

**lemma** *norm-numeral* [*simp*]: $norm\ (numeral\ w::'a::real\text{-}normed\text{-}algebra\text{-}1) = numeral\ w$
⟨*proof*⟩

**lemma** *norm-neg-numeral* [*simp*]: $norm\ (-\ numeral\ w::'a::real\text{-}normed\text{-}algebra\text{-}1) = numeral\ w$
⟨*proof*⟩

**lemma** *norm-of-real-add1* [*simp*]: $norm\ (of\text{-}real\ x + 1 :: 'a :: real\text{-}normed\text{-}div\text{-}algebra) = |x + 1|$
⟨*proof*⟩

**lemma** *norm-of-real-addn* [*simp*]:
$norm\ (of\text{-}real\ x + numeral\ b :: 'a :: real\text{-}normed\text{-}div\text{-}algebra) = |x + numeral\ b|$
⟨*proof*⟩

**lemma** *norm-of-int* [*simp*]: $norm\ (of\text{-}int\ z::'a::real\text{-}normed\text{-}algebra\text{-}1) = |of\text{-}int\ z|$
⟨*proof*⟩

**lemma** *norm-of-nat* [*simp*]: *norm* (*of-nat* $n$::'$a$::*real-normed-algebra-1*) = *of-nat* $n$
⟨*proof*⟩

**lemma** *nonzero-norm-inverse*: $a \neq 0 \Longrightarrow$ *norm* (*inverse* $a$) = *inverse* (*norm* $a$)
**for** $a$ :: '$a$::*real-normed-div-algebra*
⟨*proof*⟩

**lemma** *norm-inverse*: *norm* (*inverse* $a$) = *inverse* (*norm* $a$)
**for** $a$ :: '$a$::{*real-normed-div-algebra*,*division-ring*}
⟨*proof*⟩

**lemma** *nonzero-norm-divide*: $b \neq 0 \Longrightarrow$ *norm* ($a$ / $b$) = *norm* $a$ / *norm* $b$
**for** $a$ $b$ :: '$a$::*real-normed-field*
⟨*proof*⟩

**lemma** *norm-divide*: *norm* ($a$ / $b$) = *norm* $a$ / *norm* $b$
**for** $a$ $b$ :: '$a$::{*real-normed-field*,*field*}
⟨*proof*⟩

**lemma** *dist-divide-right*: *dist* ($a/c$) ($b/c$) = *dist* $a$ $b$ / *norm* $c$ **for** $c$ :: '$a$ :: *real-normed-field*
⟨*proof*⟩

**lemma** *norm-inverse-le-norm*:
**fixes** $x$ :: '$a$::*real-normed-div-algebra*
**shows** $r \leq$ *norm* $x \Longrightarrow 0 < r \Longrightarrow$ *norm* (*inverse* $x$) $\leq$ *inverse* $r$
⟨*proof*⟩

**lemma** *norm-power-ineq*: *norm* ($x$ ^ $n$) $\leq$ *norm* $x$ ^ $n$
**for** $x$ :: '$a$::*real-normed-algebra-1*
⟨*proof*⟩

**lemma** *norm-power*: *norm* ($x$ ^ $n$) = *norm* $x$ ^ $n$
**for** $x$ :: '$a$::*real-normed-div-algebra*
⟨*proof*⟩

**lemma** *norm-power-int*: *norm* (*power-int* $x$ $n$) = *power-int* (*norm* $x$) $n$
**for** $x$ :: '$a$::*real-normed-div-algebra*
⟨*proof*⟩

**lemma** *power-eq-imp-eq-norm*:
**fixes** $w$ :: '$a$::*real-normed-div-algebra*
**assumes** *eq*: $w$ ^ $n$ = $z$ ^ $n$ **and** $n > 0$
**shows** *norm* $w$ = *norm* $z$
⟨*proof*⟩

**lemma** *power-eq-1-iff*:
**fixes** $w$ :: '$a$::*real-normed-div-algebra*
**shows** $w$ ^ $n$ = $1 \Longrightarrow$ *norm* $w$ = $1 \vee n = 0$

⟨*proof*⟩

**lemma** *norm-mult-numeral1* [*simp*]: *norm* (*numeral w* $*$ *a*) = *numeral w* $*$ *norm a*
**for** *a b* :: *'a*::{*real-normed-field*,*field*}
⟨*proof*⟩

**lemma** *norm-mult-numeral2* [*simp*]: *norm* (*a* $*$ *numeral w*) = *norm a* $*$ *numeral w*
**for** *a b* :: *'a*::{*real-normed-field*,*field*}
⟨*proof*⟩

**lemma** *norm-divide-numeral* [*simp*]: *norm* (*a* / *numeral w*) = *norm a* / *numeral w*
**for** *a b* :: *'a*::{*real-normed-field*,*field*}
⟨*proof*⟩

**lemma** *norm-of-real-diff* [*simp*]:
*norm* (*of-real b* − *of-real a* :: *'a*::*real-normed-algebra-1*) ≤ |*b* − *a*|
⟨*proof*⟩

Despite a superficial resemblance, *norm-eq-1* is not relevant.

**lemma** *square-norm-one*:
**fixes** *x* :: *'a*::*real-normed-div-algebra*
**assumes** $x^2$ = *1*
**shows** *norm x* = *1*
⟨*proof*⟩

**lemma** *norm-less-p1*: *norm x* < *norm* (*of-real* (*norm x*) + *1* :: *'a*)
**for** *x* :: *'a*::*real-normed-algebra-1*
⟨*proof*⟩

**lemma** *prod-norm*: *prod* (λ*x*. *norm* (*f x*)) *A* = *norm* (*prod f A*)
**for** *f* :: *'a* ⇒ *'b*::{*comm-semiring-1*,*real-normed-div-algebra*}
⟨*proof*⟩

**lemma** *norm-prod-le*:
*norm* (*prod f A*) ≤ (∏ *a*∈*A*. *norm* (*f a* :: *'a* :: {*real-normed-algebra-1*,*comm-monoid-mult*}))
⟨*proof*⟩

**lemma** *norm-prod-diff*:
**fixes** *z w* :: *'i* ⇒ *'a*::{*real-normed-algebra-1*, *comm-monoid-mult*}
**shows** (⋀*i*. *i* ∈ *I* ⟹ *norm* (*z i*) ≤ *1*) ⟹ (⋀*i*. *i* ∈ *I* ⟹ *norm* (*w i*) ≤ *1*) ⟹
*norm* ((∏ *i*∈*I*. *z i*) − (∏ *i*∈*I*. *w i*)) ≤ (∑ *i*∈*I*. *norm* (*z i* − *w i*))
⟨*proof*⟩

**lemma** *norm-power-diff*:
**fixes** *z w* :: *'a*::{*real-normed-algebra-1*, *comm-monoid-mult*}
**assumes** *norm z* ≤ *1* *norm w* ≤ *1*

**shows** *norm* $(z\hat{\ }m - w\hat{\ }m) \le m * norm\ (z - w)$
⟨*proof*⟩

## 107.6 Metric spaces

**class** *metric-space* = *uniformity-dist* + *open-uniformity* +
**assumes** *dist-eq-0-iff* [*simp*]: $dist\ x\ y = 0 \longleftrightarrow x = y$
**and** *dist-triangle2*: $dist\ x\ y \le dist\ x\ z + dist\ y\ z$
**begin**

**lemma** *dist-self* [*simp*]: $dist\ x\ x = 0$
⟨*proof*⟩

**lemma** *zero-le-dist* [*simp*]: $0 \le dist\ x\ y$
⟨*proof*⟩

**lemma** *zero-less-dist-iff*: $0 < dist\ x\ y \longleftrightarrow x \ne y$
⟨*proof*⟩

**lemma** *dist-not-less-zero* [*simp*]: $\neg\ dist\ x\ y < 0$
⟨*proof*⟩

**lemma** *dist-le-zero-iff* [*simp*]: $dist\ x\ y \le 0 \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *dist-commute*: $dist\ x\ y = dist\ y\ x$
⟨*proof*⟩

**lemma** *dist-commute-lessI*: $dist\ y\ x < e \Longrightarrow dist\ x\ y < e$
⟨*proof*⟩

**lemma** *dist-triangle*: $dist\ x\ z \le dist\ x\ y + dist\ y\ z$
⟨*proof*⟩

**lemma** *dist-triangle3*: $dist\ x\ y \le dist\ a\ x + dist\ a\ y$
⟨*proof*⟩

**lemma** *abs-dist-diff-le*: $|dist\ a\ b - dist\ b\ c| \le dist\ a\ c$
⟨*proof*⟩

**lemma** *dist-pos-lt*: $x \ne y \Longrightarrow 0 < dist\ x\ y$
⟨*proof*⟩

**lemma** *dist-nz*: $x \ne y \longleftrightarrow 0 < dist\ x\ y$
⟨*proof*⟩

**declare** *dist-nz* [*symmetric*, *simp*]

**lemma** *dist-triangle-le*: $dist\ x\ z + dist\ y\ z \le e \Longrightarrow dist\ x\ y \le e$

⟨*proof*⟩

**lemma** *dist-triangle-lt*: $dist\ x\ z + dist\ y\ z < e \Longrightarrow dist\ x\ y < e$
⟨*proof*⟩

**lemma** *dist-triangle-less-add*: $dist\ x1\ y < e1 \Longrightarrow dist\ x2\ y < e2 \Longrightarrow dist\ x1\ x2 < e1 + e2$
⟨*proof*⟩

**lemma** *dist-triangle-half-l*: $dist\ x1\ y < e\ /\ 2 \Longrightarrow dist\ x2\ y < e\ /\ 2 \Longrightarrow dist\ x1\ x2 < e$
⟨*proof*⟩

**lemma** *dist-triangle-half-r*: $dist\ y\ x1 < e\ /\ 2 \Longrightarrow dist\ y\ x2 < e\ /\ 2 \Longrightarrow dist\ x1\ x2 < e$
⟨*proof*⟩

**lemma** *dist-triangle-third*:
**assumes** $dist\ x1\ x2 < e/3$ $dist\ x2\ x3 < e/3$ $dist\ x3\ x4 < e/3$
**shows** $dist\ x1\ x4 < e$
⟨*proof*⟩

**subclass** *uniform-space*
⟨*proof*⟩

**lemma** *open-dist*: $open\ S \longleftrightarrow (\forall x \in S.\ \exists e>0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S)$
⟨*proof*⟩

**lemma** *open-ball*: $open\ \{y.\ dist\ x\ y < d\}$
⟨*proof*⟩

**subclass** *first-countable-topology*
⟨*proof*⟩

**end**

**instance** *metric-space* ⊆ *t2-space*
⟨*proof*⟩

Every normed vector space is a metric space.

**instance** *real-normed-vector* < *metric-space*
⟨*proof*⟩

## 107.7 Class instances for real numbers

**instantiation** *real* :: *real-normed-field*
**begin**

**definition** *dist-real-def*: $dist\ x\ y = |x - y|$

**definition** *uniformity-real-def* [*code del*]:
($uniformity$ :: ($real \times real$) *filter*) = (*INF* $e \in \{0 <..\}$. *principal* $\{(x, y).\ dist\ x\ y < e\}$)

**definition** *open-real-def* [*code del*]:
*open* ($U$ :: *real set*) $\longleftrightarrow$ ($\forall x \in U$. *eventually* ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) *uniformity*)

**definition** *real-norm-def* [*simp*]: *norm* $r = |r|$

**instance**
⟨*proof*⟩

**end**

**declare** *uniformity-Abort*[**where** '*a*=*real*, *code*]

**lemma** *dist-of-real* [*simp*]: *dist* (*of-real* $x$ :: '*a*) (*of-real* $y$) = *dist* $x$ $y$
**for** $a$ :: '*a*::*real-normed-div-algebra*
⟨*proof*⟩

**declare** [[*code abort*: *open* :: *real set* ⇒ *bool*]]

**instance** *real* :: *linorder-topology*
⟨*proof*⟩

**instance** *real* :: *linear-continuum-topology* ⟨*proof*⟩

**lemmas** *open-real-greaterThan* = *open-greaterThan*[**where** '*a*=*real*]
**lemmas** *open-real-lessThan* = *open-lessThan*[**where** '*a*=*real*]
**lemmas** *open-real-greaterThanLessThan* = *open-greaterThanLessThan*[**where** '*a*=*real*]
**lemmas** *closed-real-atMost* = *closed-atMost*[**where** '*a*=*real*]
**lemmas** *closed-real-atLeast* = *closed-atLeast*[**where** '*a*=*real*]
**lemmas** *closed-real-atLeastAtMost* = *closed-atLeastAtMost*[**where** '*a*=*real*]

**instance** *real* :: *ordered-real-vector*
⟨*proof*⟩

## 107.8 Extra type constraints

Only allow *open* in class *topological-space*.

⟨*ML*⟩

Only allow *uniformity* in class *uniform-space*.

⟨*ML*⟩

Only allow *dist* in class *metric-space*.

⟨*ML*⟩

Only allow *norm* in class *real-normed-vector*.

⟨*ML*⟩

## 107.9 Sign function

**lemma** *norm-sgn*: $norm\ (sgn\ x) = (if\ x = 0\ then\ 0\ else\ 1)$
  **for** $x$ :: $'a$::*real-normed-vector*
  ⟨*proof*⟩

**lemma** *sgn-zero* [*simp*]: $sgn\ (0::'a::real\text{-}normed\text{-}vector) = 0$
  ⟨*proof*⟩

**lemma** *sgn-zero-iff*: $sgn\ x = 0 \longleftrightarrow x = 0$
  **for** $x$ :: $'a$::*real-normed-vector*
  ⟨*proof*⟩

**lemma** *sgn-minus*: $sgn\ (-\ x) = -\ sgn\ x$
  **for** $x$ :: $'a$::*real-normed-vector*
  ⟨*proof*⟩

**lemma** *sgn-scaleR*: $sgn\ (scaleR\ r\ x) = scaleR\ (sgn\ r)\ (sgn\ x)$
  **for** $x$ :: $'a$::*real-normed-vector*
  ⟨*proof*⟩

**lemma** *sgn-one* [*simp*]: $sgn\ (1::'a::real\text{-}normed\text{-}algebra\text{-}1) = 1$
  ⟨*proof*⟩

**lemma** *sgn-of-real*: $sgn\ (of\text{-}real\ r :: 'a::real\text{-}normed\text{-}algebra\text{-}1) = of\text{-}real\ (sgn\ r)$
  ⟨*proof*⟩

**lemma** *sgn-mult*: $sgn\ (x * y) = sgn\ x * sgn\ y$
  **for** $x\ y$ :: $'a$::*real-normed-div-algebra*
  ⟨*proof*⟩

**hide-fact** (**open**) *sgn-mult*

**lemma** *real-sgn-eq*: $sgn\ x = x\ /\ |x|$
  **for** $x$ :: *real*
  ⟨*proof*⟩

**lemma** *zero-le-sgn-iff* [*simp*]: $0 \leq sgn\ x \longleftrightarrow 0 \leq x$
  **for** $x$ :: *real*
  ⟨*proof*⟩

**lemma** *sgn-le-0-iff* [*simp*]: $sgn\ x \leq 0 \longleftrightarrow x \leq 0$
  **for** $x$ :: *real*
  ⟨*proof*⟩

**lemma** *norm-conv-dist*: $norm\ x = dist\ x\ 0$

⟨proof⟩

**declare** *norm-conv-dist* [*symmetric*, *simp*]

**lemma** *dist-0-norm* [*simp*]: $dist\ 0\ x = norm\ x$
**for** $x$ :: *'a::real-normed-vector*
⟨proof⟩

**lemma** *dist-diff* [*simp*]: $dist\ a\ (a - b) = norm\ b$ $dist\ (a - b)\ a = norm\ b$
⟨proof⟩

**lemma** *dist-of-int*: $dist\ (of\text{-}int\ m)\ (of\text{-}int\ n :: 'a :: real\text{-}normed\text{-}algebra\text{-}1) = of\text{-}int\ |m - n|$
⟨proof⟩

**lemma** *dist-of-nat*:
$dist\ (of\text{-}nat\ m)\ (of\text{-}nat\ n :: 'a :: real\text{-}normed\text{-}algebra\text{-}1) = of\text{-}int\ |int\ m - int\ n|$
⟨proof⟩

## 107.10 Bounded Linear and Bilinear Operators

**lemma** *linearI*: *linear f*
**if** $\bigwedge b1\ b2.\ f\ (b1 + b2) = f\ b1 + f\ b2$
$\bigwedge r\ b.\ f\ (r *_R b) = r *_R f\ b$
⟨proof⟩

**lemma** *linear-iff*:
$linear\ f \longleftrightarrow (\forall x\ y.\ f\ (x + y) = f\ x + f\ y) \wedge (\forall c\ x.\ f\ (c *_R x) = c *_R f\ x)$
(**is** $linear\ f \longleftrightarrow ?rhs$)
⟨proof⟩

**lemma** *linear-of-real* [*simp*]: *linear of-real*
⟨proof⟩

**lemmas** *linear-scaleR-left* = *linear-scale-left*
**lemmas** *linear-imp-scaleR* = *linear-imp-scale*

**corollary** *real-linearD*:
**fixes** $f :: real \Rightarrow real$
**assumes** *linear f* **obtains** $c$ **where** $f = (*)\ c$
⟨proof⟩

**lemma** *linear-times-of-real*: $linear\ (\lambda x.\ a * of\text{-}real\ x)$
⟨proof⟩

**locale** *bounded-linear* = *linear f* **for** $f$ :: *'a::real-normed-vector* ⇒ *'b::real-normed-vector* +
**assumes** *bounded*: $\exists K.\ \forall x.\ norm\ (f\ x) \le norm\ x * K$
**begin**

**lemma** *pos-bounded*: $\exists K>0.\ \forall x.\ norm\ (f\ x) \leq norm\ x * K$
⟨*proof*⟩

**lemma** *nonneg-bounded*: $\exists K\geq 0.\ \forall x.\ norm\ (f\ x) \leq norm\ x * K$
⟨*proof*⟩

**lemma** *linear*: *linear f*
⟨*proof*⟩

**end**

**lemma** *bounded-linear-intro*:
**assumes** $\bigwedge x\ y.\ f\ (x + y) = f\ x + f\ y$
**and** $\bigwedge r\ x.\ f\ (scaleR\ r\ x) = scaleR\ r\ (f\ x)$
**and** $\bigwedge x.\ norm\ (f\ x) \leq norm\ x * K$
**shows** *bounded-linear f*
⟨*proof*⟩

**locale** *bounded-bilinear* =
**fixes** *prod* :: $'a$::*real-normed-vector* ⇒ $'b$::*real-normed-vector* ⇒ $'c$::*real-normed-vector*
(**infixl** ‹**› *70*)
**assumes** *add-left*: $prod\ (a + a')\ b = prod\ a\ b + prod\ a'\ b$
**and** *add-right*: $prod\ a\ (b + b') = prod\ a\ b + prod\ a\ b'$
**and** *scaleR-left*: $prod\ (scaleR\ r\ a)\ b = scaleR\ r\ (prod\ a\ b)$
**and** *scaleR-right*: $prod\ a\ (scaleR\ r\ b) = scaleR\ r\ (prod\ a\ b)$
**and** *bounded*: $\exists K.\ \forall a\ b.\ norm\ (prod\ a\ b) \leq norm\ a * norm\ b * K$
**begin**

**lemma** *pos-bounded*: $\exists K>0.\ \forall a\ b.\ norm\ (a ** b) \leq norm\ a * norm\ b * K$
⟨*proof*⟩

**lemma** *nonneg-bounded*: $\exists K\geq 0.\ \forall a\ b.\ norm\ (a ** b) \leq norm\ a * norm\ b * K$
⟨*proof*⟩

**lemma** *additive-right*: *additive* ($\lambda b.\ prod\ a\ b$)
⟨*proof*⟩

**lemma** *additive-left*: *additive* ($\lambda a.\ prod\ a\ b$)
⟨*proof*⟩

**lemma** *zero-left*: $prod\ 0\ b = 0$
⟨*proof*⟩

**lemma** *zero-right*: $prod\ a\ 0 = 0$
⟨*proof*⟩

**lemma** *minus-left*: $prod\ (-\ a)\ b = -\ prod\ a\ b$
⟨*proof*⟩

**lemma** *minus-right*: *prod a* (− *b*) = − *prod a b*
⟨*proof*⟩

**lemma** *diff-left*: *prod* (*a* − *a'*) *b* = *prod a b* − *prod a' b*
⟨*proof*⟩

**lemma** *diff-right*: *prod a* (*b* − *b'*) = *prod a b* − *prod a b'*
⟨*proof*⟩

**lemma** *sum-left*: *prod* (*sum g S*) *x* = *sum* ((λ*i*. *prod* (*g i*) *x*)) *S*
⟨*proof*⟩

**lemma** *sum-right*: *prod x* (*sum g S*) = *sum* ((λ*i*. (*prod x* (*g i*)))) *S*
⟨*proof*⟩

**lemma** *bounded-linear-left*: *bounded-linear* (λ*a*. *a* ** *b*)
⟨*proof*⟩

**lemma** *bounded-linear-right*: *bounded-linear* (λ*b*. *a* ** *b*)
⟨*proof*⟩

**lemma** *prod-diff-prod*: (*x* ** *y* − *a* ** *b*) = (*x* − *a*) ** (*y* − *b*) + (*x* − *a*) ** *b* + *a* ** (*y* − *b*)
⟨*proof*⟩

**lemma** *flip*: *bounded-bilinear* (λ*x y*. *y* ** *x*)
⟨*proof*⟩

**lemma** *comp1*:
**assumes** *bounded-linear g*
**shows** *bounded-bilinear* (λ*x*. (**) (*g x*))
⟨*proof*⟩

**lemma** *comp*: *bounded-linear f* ⟹ *bounded-linear g* ⟹ *bounded-bilinear* (λ*x y*. *f x* ** *g y*)
⟨*proof*⟩

**end**

**lemma** *bounded-linear-ident*[*simp*]: *bounded-linear* (λ*x*. *x*)
⟨*proof*⟩

**lemma** *bounded-linear-zero*[*simp*]: *bounded-linear* (λ*x*. *0*)
⟨*proof*⟩

**lemma** *bounded-linear-add*:
**assumes** *bounded-linear f*

**and** *bounded-linear g*
**shows** *bounded-linear* $(\lambda x.\ f\ x + g\ x)$
⟨*proof*⟩

**lemma** *bounded-linear-minus*:
**assumes** *bounded-linear f*
**shows** *bounded-linear* $(\lambda x.\ -\ f\ x)$
⟨*proof*⟩

**lemma** *bounded-linear-sub*: *bounded-linear f* $\Longrightarrow$ *bounded-linear g* $\Longrightarrow$ *bounded-linear* $(\lambda x.\ f\ x - g\ x)$
⟨*proof*⟩

**lemma** *bounded-linear-sum*:
**fixes** $f$ :: $'i \Rightarrow 'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow$ *bounded-linear* $(f\ i)) \Longrightarrow$ *bounded-linear* $(\lambda x.\ \sum i{\in}I.\ f\ i\ x)$
⟨*proof*⟩

**lemma** *bounded-linear-compose*:
**assumes** *bounded-linear f*
**and** *bounded-linear g*
**shows** *bounded-linear* $(\lambda x.\ f\ (g\ x))$
⟨*proof*⟩

**lemma** *bounded-bilinear-mult*: *bounded-bilinear* $((*) :: 'a \Rightarrow 'a \Rightarrow 'a$::*real-normed-algebra*)
⟨*proof*⟩

**lemma** *bounded-linear-mult-left*: *bounded-linear* $(\lambda x{::}'a$::*real-normed-algebra*. $x * y)$
⟨*proof*⟩

**lemma** *bounded-linear-mult-right*: *bounded-linear* $(\lambda y{::}'a$::*real-normed-algebra*. $x * y)$
⟨*proof*⟩

**lemmas** *bounded-linear-mult-const* =
*bounded-linear-mult-left* [*THEN bounded-linear-compose*]

**lemmas** *bounded-linear-const-mult* =
*bounded-linear-mult-right* [*THEN bounded-linear-compose*]

**lemma** *bounded-linear-divide*: *bounded-linear* $(\lambda x.\ x\ /\ y)$
**for** $y$ :: $'a$::*real-normed-field*
⟨*proof*⟩

**lemma** *bounded-bilinear-scaleR*: *bounded-bilinear scaleR*
⟨*proof*⟩

**lemma** *bounded-linear-scaleR-left*: *bounded-linear* ($\lambda r.$ *scaleR* $r$ $x$)
⟨*proof*⟩

**lemma** *bounded-linear-scaleR-right*: *bounded-linear* ($\lambda x.$ *scaleR* $r$ $x$)
⟨*proof*⟩

**lemmas** *bounded-linear-scaleR-const* =
*bounded-linear-scaleR-left*[*THEN bounded-linear-compose*]

**lemmas** *bounded-linear-const-scaleR* =
*bounded-linear-scaleR-right*[*THEN bounded-linear-compose*]

**lemma** *bounded-linear-of-real*: *bounded-linear* ($\lambda r.$ *of-real* $r$)
⟨*proof*⟩

**lemma** *real-bounded-linear*: *bounded-linear* $f \longleftrightarrow (\exists\, c{::}real.\ f = (\lambda x.\ x * c))$
**for** $f$ :: *real* ⇒ *real*
⟨*proof*⟩

**instance** *real-normed-algebra-1* ⊆ *perfect-space*
⟨*proof*⟩

## 107.11 Filters and Limits on Metric Space

**lemma** (**in** *metric-space*) *nhds-metric*: *nhds* $x$ = (*INF* $e{\in}\{0 <..\}$. *principal* $\{y.$ *dist* $y$ $x < e\}$)
⟨*proof*⟩

**lemma** *tendsto-iff-uniformity*:
— More general analogus of *tendsto-iff* below. Applies to all uniform spaces, not just metric ones.
**fixes** $l$ :: ‹'b :: *uniform-space*›
**shows** ‹($f \longrightarrow l$) $F \longleftrightarrow$ ($\forall E.$ *eventually* $E$ *uniformity* $\longrightarrow$ ($\forall_F$ $x$ *in* $F$. $E$ ($f$ $x$, $l$)))›
⟨*proof*⟩

**lemma** (**in** *metric-space*) *tendsto-iff*: ($f \longrightarrow l$) $F \longleftrightarrow$ ($\forall e{>}0.$ *eventually* ($\lambda x.$ *dist* ($f$ $x$) $l < e$) $F$)
⟨*proof*⟩

**lemma** *tendsto-dist-iff*:
(($f \longrightarrow l$) $F$) $\longleftrightarrow$ ((($\lambda x.$ *dist* ($f$ $x$) $l$) $\longrightarrow 0$) $F$)
⟨*proof*⟩

**lemma** (**in** *metric-space*) *tendstoI* [*intro?*]:
($\bigwedge e.\ 0 < e \Longrightarrow$ *eventually* ($\lambda x.$ *dist* ($f$ $x$) $l < e$) $F$) $\Longrightarrow$ ($f \longrightarrow l$) $F$
⟨*proof*⟩

**lemma** (**in** *metric-space*) *tendstoD*: $(f \longrightarrow l)\ F \Longrightarrow 0 < e \Longrightarrow$ *eventually* $(\lambda x.\ dist\ (f\ x)\ l < e)\ F$
⟨*proof*⟩

**lemma** (**in** *metric-space*) *eventually-nhds-metric*:
*eventually* $P\ (nhds\ a) \longleftrightarrow (\exists\, d>0.\ \forall\, x.\ dist\ x\ a < d \longrightarrow P\ x)$
⟨*proof*⟩

**lemma** *eventually-at*: *eventually* $P\ (at\ a\ within\ S) \longleftrightarrow (\exists\, d>0.\ \forall\, x\in S.\ x \neq a \wedge dist\ x\ a < d \longrightarrow P\ x)$
**for** $a :: 'a :: metric\text{-}space$
⟨*proof*⟩

**lemma** *frequently-at*: *frequently* $P\ (at\ a\ within\ S) \longleftrightarrow (\forall\, d>0.\ \exists\, x\in S.\ x \neq a \wedge dist\ x\ a < d \wedge P\ x)$
**for** $a :: 'a :: metric\text{-}space$
⟨*proof*⟩

**lemma** *eventually-at-le*: *eventually* $P\ (at\ a\ within\ S) \longleftrightarrow (\exists\, d>0.\ \forall\, x\in S.\ x \neq a \wedge dist\ x\ a \leq d \longrightarrow P\ x)$
**for** $a :: 'a::metric\text{-}space$
⟨*proof*⟩

**lemma** *eventually-at-left-real*: $a > (b :: real) \Longrightarrow$ *eventually* $(\lambda x.\ x \in \{b<..<a\})$ $(at\text{-}left\ a)$
⟨*proof*⟩

**lemma** *eventually-at-right-real*: $a < (b :: real) \Longrightarrow$ *eventually* $(\lambda x.\ x \in \{a<..<b\})$ $(at\text{-}right\ a)$
⟨*proof*⟩

**lemma** *metric-tendsto-imp-tendsto*:
**fixes** $a :: 'a :: metric\text{-}space$
**and** $b :: 'b :: metric\text{-}space$
**assumes** $f$: $(f \longrightarrow a)\ F$
**and** *le*: *eventually* $(\lambda x.\ dist\ (g\ x)\ b \leq dist\ (f\ x)\ a)\ F$
**shows** $(g \longrightarrow b)\ F$
⟨*proof*⟩

**lemma** *filterlim-real-sequentially*: *LIM x sequentially. real x :> at-top*
⟨*proof*⟩

**lemma** *filterlim-nat-sequentially*: *filterlim nat sequentially at-top*
⟨*proof*⟩

**lemma** *filterlim-floor-sequentially*: *filterlim floor at-top at-top*
⟨*proof*⟩

**lemma** *filterlim-sequentially-iff-filterlim-real*:

*filterlim f sequentially F* $\longleftrightarrow$ *filterlim* ($\lambda x.$ *real* (*f x*)) *at-top F* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

### 107.11.1 Limits of Sequences

**lemma** *lim-sequentially*: $X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$
**for** $L$ :: $'a$::*metric-space*
⟨*proof*⟩

**lemmas** *LIMSEQ-def* = *lim-sequentially*

**lemma** *LIMSEQ-iff-nz*: $X \longrightarrow L \longleftrightarrow (\forall r{>}0.\ \exists no{>}0.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r)$
**for** $L$ :: $'a$::*metric-space*
⟨*proof*⟩

**lemma** *metric-LIMSEQ-I*: $(\bigwedge r.\ 0 < r \Longrightarrow \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r) \Longrightarrow X \longrightarrow L$
**for** $L$ :: $'a$::*metric-space*
⟨*proof*⟩

**lemma** *metric-LIMSEQ-D*: $X \longrightarrow L \Longrightarrow 0 < r \Longrightarrow \exists no.\ \forall n{\geq}no.\ dist\ (X\ n)\ L < r$
**for** $L$ :: $'a$::*metric-space*
⟨*proof*⟩

**lemma** *LIMSEQ-norm-0*:
**assumes** $\bigwedge n{::}nat.\ norm\ (f\ n) < 1\ /\ real\ (Suc\ n)$
**shows** $f \longrightarrow 0$
⟨*proof*⟩

### 107.11.2 Limits of Functions

**lemma** *LIM-def*: $f\ {-a}{\rightarrow}\ L \longleftrightarrow (\forall r > 0.\ \exists s > 0.\ \forall x.\ x \neq a \wedge dist\ x\ a < s \longrightarrow dist\ (f\ x)\ L < r)$
**for** $a$ :: $'a$::*metric-space* **and** $L$ :: $'b$::*metric-space*
⟨*proof*⟩

**lemma** *metric-LIM-I*:
$(\bigwedge r.\ 0 < r \Longrightarrow \exists s{>}0.\ \forall x.\ x \neq a \wedge dist\ x\ a < s \longrightarrow dist\ (f\ x)\ L < r) \Longrightarrow f\ {-a}{\rightarrow}\ L$
**for** $a$ :: $'a$::*metric-space* **and** $L$ :: $'b$::*metric-space*
⟨*proof*⟩

**lemma** *metric-LIM-D*: $f\ {-a}{\rightarrow}\ L \Longrightarrow 0 < r \Longrightarrow \exists s{>}0.\ \forall x.\ x \neq a \wedge dist\ x\ a < s \longrightarrow dist\ (f\ x)\ L < r$
**for** $a$ :: $'a$::*metric-space* **and** $L$ :: $'b$::*metric-space*
⟨*proof*⟩

**lemma** *metric-LIM-imp-LIM*:
**fixes** $l$ :: $'a$::*metric-space*
**and** $m$ :: $'b$::*metric-space*
**assumes** $f$: $f$ $-a\rightarrow$ $l$
**and** *le*: $\bigwedge x.\ x \neq a \Longrightarrow dist\ (g\ x)\ m \leq dist\ (f\ x)\ l$
**shows** $g$ $-a\rightarrow$ $m$
⟨*proof*⟩

**lemma** *metric-LIM-equal2*:
**fixes** $a$ :: $'a$::*metric-space*
**assumes** $g$ $-a\rightarrow$ $l$ $0 < R$
**and** $\bigwedge x.\ x \neq a \Longrightarrow dist\ x\ a < R \Longrightarrow f\ x = g\ x$
**shows** $f$ $-a\rightarrow$ $l$
⟨*proof*⟩

**lemma** *metric-LIM-compose2*:
**fixes** $a$ :: $'a$::*metric-space*
**assumes** $f$: $f$ $-a\rightarrow$ $b$
**and** $g$: $g$ $-b\rightarrow$ $c$
**and** *inj*: $\exists d>0.\ \forall x.\ x \neq a \wedge dist\ x\ a < d \longrightarrow f\ x \neq b$
**shows** $(\lambda x.\ g\ (f\ x))$ $-a\rightarrow$ $c$
⟨*proof*⟩

**lemma** *metric-isCont-LIM-compose2*:
**fixes** $f$ :: $'a$ :: *metric-space* $\Rightarrow$ -
**assumes** $f$ [*unfolded isCont-def*]: *isCont f a*
**and** $g$: $g$ $-f\ a\rightarrow$ $l$
**and** *inj*: $\exists d>0.\ \forall x.\ x \neq a \wedge dist\ x\ a < d \longrightarrow f\ x \neq f\ a$
**shows** $(\lambda x.\ g\ (f\ x))$ $-a\rightarrow$ $l$
⟨*proof*⟩

## 107.12 Complete metric spaces

## 107.13 Cauchy sequences

**lemma** (**in** *metric-space*) *Cauchy-def*: *Cauchy X* = $(\forall e>0.\ \exists M.\ \forall m\geq M.\ \forall n\geq M.\ dist\ (X\ m)\ (X\ n) < e)$
⟨*proof*⟩

**lemma** (**in** *metric-space*) *Cauchy-altdef*: *Cauchy f* $\longleftrightarrow$ $(\forall e>0.\ \exists M.\ \forall m\geq M.\ \forall n>m.\ dist\ (f\ m)\ (f\ n) < e)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

**lemma** (**in** *metric-space*) *Cauchy-altdef2*: *Cauchy s* $\longleftrightarrow$ $(\forall e>0.\ \exists N::nat.\ \forall n\geq N.\ dist(s\ n)(s\ N) < e)$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** (**in** *metric-space*) *metric-CauchyI*:
$(\bigwedge e.\ 0 < e \Longrightarrow \exists M.\ \forall m\geq M.\ \forall n\geq M.\ dist\ (X\ m)\ (X\ n) < e) \Longrightarrow Cauchy\ X$

⟨*proof*⟩

**lemma** (**in** *metric-space*) *CauchyI'*:
$(\bigwedge e.\ 0 < e \Longrightarrow \exists M.\ \forall m{\geq}M.\ \forall n{>}m.\ dist\ (X\ m)\ (X\ n) < e) \Longrightarrow Cauchy\ X$
⟨*proof*⟩

**lemma** (**in** *metric-space*) *metric-CauchyD*:
$Cauchy\ X \Longrightarrow 0 < e \Longrightarrow \exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ dist\ (X\ m)\ (X\ n) < e$
⟨*proof*⟩

**lemma** (**in** *metric-space*) *metric-Cauchy-iff2*:
$Cauchy\ X = (\forall j.\ (\exists M.\ \forall m \geq M.\ \forall n \geq M.\ dist\ (X\ m)\ (X\ n) < inverse(real\ (Suc\ j))))$
⟨*proof*⟩

**lemma** *Cauchy-iff2*: $Cauchy\ X \longleftrightarrow (\forall j.\ (\exists M.\ \forall m \geq M.\ \forall n \geq M.\ |X\ m - X\ n| < inverse\ (real\ (Suc\ j))))$
⟨*proof*⟩

**lemma** *lim-1-over-n* [*tendsto-intros*]: $((\lambda n.\ 1\ /\ of\text{-}nat\ n) \longlongrightarrow (0::'a::real\text{-}normed\text{-}field))$ *sequentially*
⟨*proof*⟩

**lemma** (**in** *metric-space*) *complete-def*:
**shows** $complete\ S = (\forall f.\ (\forall n.\ f\ n \in S) \wedge Cauchy\ f \longrightarrow (\exists l{\in}S.\ f \longlongrightarrow l))$
⟨*proof*⟩

apparently unused

**lemma** (**in** *metric-space*) *totally-bounded-metric*:
$totally\text{-}bounded\ S \longleftrightarrow (\forall e{>}0.\ \exists k.\ finite\ k \wedge S \subseteq (\bigcup x{\in}k.\ \{y.\ dist\ x\ y < e\}))$
⟨*proof*⟩

⟨*ML*⟩

**lemma** *cauchy-filter-metric*:
**fixes** $F :: 'a::\{uniformity\text{-}dist,uniform\text{-}space\}\ filter$
**shows** $cauchy\text{-}filter\ F \longleftrightarrow (\forall e.\ e{>}0 \longrightarrow (\exists P.\ eventually\ P\ F \wedge (\forall x\ y.\ P\ x \wedge P\ y \longrightarrow dist\ x\ y < e)))$
⟨*proof*⟩

**lemma** *cauchy-filter-metric-filtermap*:
**fixes** $f :: 'a \Rightarrow 'b::\{uniformity\text{-}dist,uniform\text{-}space\}$
**shows** $cauchy\text{-}filter\ (filtermap\ f\ F) \longleftrightarrow (\forall e.\ e{>}0 \longrightarrow (\exists P.\ eventually\ P\ F \wedge (\forall x\ y.\ P\ x \wedge P\ y \longrightarrow dist\ (f\ x)\ (f\ y) < e)))$
⟨*proof*⟩

⟨*ML*⟩

### 107.13.1 Cauchy Sequences are Convergent

**class** *complete-space* = *metric-space* +
  **assumes** *Cauchy-convergent*: *Cauchy X* $\Longrightarrow$ *convergent X*

**lemma** *Cauchy-convergent-iff*: *Cauchy X* $\longleftrightarrow$ *convergent X*
  **for** *X* :: *nat* $\Rightarrow$ *'a*::*complete-space*
  ⟨*proof*⟩

To prove that a Cauchy sequence converges, it suffices to show that a subsequence converges.

**lemma** *Cauchy-converges-subseq*:
  **fixes** *u*::*nat* $\Rightarrow$ *'a*::*metric-space*
  **assumes** *Cauchy u*
    *strict-mono r*
    $(u \circ r) \longrightarrow l$
  **shows** $u \longrightarrow l$
⟨*proof*⟩

## 107.14 The set of real numbers is a complete metric space

Proof that Cauchy sequences converge based on the one from http://pirate.shu.edu/~wachsmut/ira/numseq/proofs/cauconv.html

If sequence $X$ is Cauchy, then its limit is the lub of $\{r.\ \exists N.\ \forall n{\geq}N.\ r < X\ n\}$

**lemma** *increasing-LIMSEQ*:
  **fixes** *f* :: *nat* $\Rightarrow$ *real*
  **assumes** *inc*: $\bigwedge n.\ f\ n \leq f\ (Suc\ n)$
    **and** *bdd*: $\bigwedge n.\ f\ n \leq l$
    **and** *en*: $\bigwedge e.\ 0 < e \Longrightarrow \exists n.\ l \leq f\ n + e$
  **shows** $f \longrightarrow l$
⟨*proof*⟩

**lemma** *real-Cauchy-convergent*:
  **fixes** *X* :: *nat* $\Rightarrow$ *real*
  **assumes** *X*: *Cauchy X*
  **shows** *convergent X*
⟨*proof*⟩

**instance** *real* :: *complete-space*
  ⟨*proof*⟩

**class** *banach* = *real-normed-vector* + *complete-space*

**instance** *real* :: *banach* ⟨*proof*⟩

**lemma** *tendsto-at-topI-sequentially*:
**fixes** $f$ :: *real* $\Rightarrow$ $'b$::*first-countable-topology*
**assumes** $*$: $\bigwedge X$. *filterlim X at-top sequentially* $\Longrightarrow$ $(\lambda n.\ f\ (X\ n)) \longrightarrow y$
**shows** $(f \longrightarrow y)$ *at-top*
⟨*proof*⟩

**lemma** *tendsto-at-topI-sequentially-real*:
**fixes** $f$ :: *real* $\Rightarrow$ *real*
**assumes** *mono*: *mono f*
**and** *limseq*: $(\lambda n.\ f\ (real\ n)) \longrightarrow y$
**shows** $(f \longrightarrow y)$ *at-top*
⟨*proof*⟩

**end**

# 108 Limits on Real Vector Spaces

**theory** *Limits*
**imports** *Real-Vector-Spaces*
**begin**

**lemma** *range-mult* [*simp*]:
**fixes** *a*::*real* **shows** *range* $((*)\ a) = ($*if* $a=0$ *then* $\{0\}$ *else UNIV*$)$
⟨*proof*⟩

## 108.1 Filter going to infinity norm

**definition** *at-infinity* :: $'a$::*real-normed-vector filter*
**where** *at-infinity* = (*INF r. principal* $\{x.\ r \leq norm\ x\}$)

**lemma** *eventually-at-infinity*: *eventually P at-infinity* $\longleftrightarrow$ $(\exists b.\ \forall x.\ b \leq norm\ x \longrightarrow P\ x)$
⟨*proof*⟩

**lemma** *eventually-at-infinityI*:
**fixes** $P$::$'a$::*real-normed-vector* $\Rightarrow$ *bool*
**assumes** $\bigwedge x.\ c \leq norm\ x \Longrightarrow P\ x$
**shows** *eventually P at-infinity*
⟨*proof*⟩

**corollary** *eventually-at-infinity-pos*:
*eventually p at-infinity* $\longleftrightarrow$ $(\exists b.\ 0 < b \wedge (\forall x.\ norm\ x \geq b \longrightarrow p\ x))$
⟨*proof*⟩

**lemma** *at-infinity-eq-at-top-bot*: (*at-infinity* :: *real filter*) = *sup at-top at-bot*
⟨*proof*⟩

**lemma** *at-top-le-at-infinity*: *at-top* $\leq$ (*at-infinity* :: *real filter*)

⟨*proof*⟩

**lemma** *at-bot-le-at-infinity*: *at-bot* $\leq$ (*at-infinity* :: *real filter*)
⟨*proof*⟩

**lemma** *filterlim-at-top-imp-at-infinity*: *filterlim f at-top F* $\Longrightarrow$ *filterlim f at-infinity F*
**for** *f* :: - $\Rightarrow$ *real*
⟨*proof*⟩

**lemma** *filterlim-real-at-infinity-sequentially*: *filterlim real at-infinity sequentially*
⟨*proof*⟩

**lemma** *lim-infinity-imp-sequentially*: ($f \longrightarrow l$) *at-infinity* $\Longrightarrow$ (($\lambda n.\ f(n)$) $\longrightarrow$ $l$) *sequentially*
⟨*proof*⟩

### 108.1.1 Boundedness

**definition** *Bfun* :: ('*a* $\Rightarrow$ '*b*::*metric-space*) $\Rightarrow$ '*a filter* $\Rightarrow$ *bool*
**where** *Bfun-metric-def*: *Bfun f F* = ($\exists\, y.\ \exists\, K>0.$ *eventually* ($\lambda x.$ *dist* (*f x*) *y* $\leq$ *K*) *F*)

**abbreviation** *Bseq* :: (*nat* $\Rightarrow$ '*a*::*metric-space*) $\Rightarrow$ *bool*
**where** *Bseq X* $\equiv$ *Bfun X sequentially*

**lemma** *Bseq-conv-Bfun*: *Bseq X* $\longleftrightarrow$ *Bfun X sequentially* ⟨*proof*⟩

**lemma** *Bseq-ignore-initial-segment*: *Bseq X* $\Longrightarrow$ *Bseq* ($\lambda n.\ X\ (n + k)$)
⟨*proof*⟩

**lemma** *Bseq-offset*: *Bseq* ($\lambda n.\ X\ (n + k)$) $\Longrightarrow$ *Bseq X*
⟨*proof*⟩

**lemma** *Bfun-def*: *Bfun f F* $\longleftrightarrow$ ($\exists\, K>0.$ *eventually* ($\lambda x.$ *norm* (*f x*) $\leq$ *K*) *F*)
⟨*proof*⟩

**lemma** *BfunI*:
**assumes** *K*: *eventually* ($\lambda x.$ *norm* (*f x*) $\leq$ *K*) *F*
**shows** *Bfun f F*
⟨*proof*⟩

**lemma** *BfunE*:
**assumes** *Bfun f F*
**obtains** *B* **where** *0* < *B* **and** *eventually* ($\lambda x.$ *norm* (*f x*) $\leq$ *B*) *F*
⟨*proof*⟩

**lemma** *Cauchy-Bseq*:
**assumes** *Cauchy X* **shows** *Bseq X*

$\langle proof \rangle$

### 108.1.2 Bounded Sequences

**lemma** *BseqI'*: $(\bigwedge n.\ norm\ (X\ n) \leq K) \Longrightarrow Bseq\ X$
$\langle proof \rangle$

**lemma** *Bseq-def*: $Bseq\ X \longleftrightarrow (\exists K{>}0.\ \forall n.\ norm\ (X\ n) \leq K)$
$\langle proof \rangle$

**lemma** *BseqE*: $Bseq\ X \Longrightarrow (\bigwedge K.\ 0 < K \Longrightarrow \forall n.\ norm\ (X\ n) \leq K \Longrightarrow Q) \Longrightarrow Q$
$\langle proof \rangle$

**lemma** *BseqD*: $Bseq\ X \Longrightarrow \exists K.\ 0 < K \land (\forall n.\ norm\ (X\ n) \leq K)$
$\langle proof \rangle$

**lemma** *BseqI*: $0 < K \Longrightarrow \forall n.\ norm\ (X\ n) \leq K \Longrightarrow Bseq\ X$
$\langle proof \rangle$

**lemma** *Bseq-bdd-above*: $Bseq\ X \Longrightarrow bdd\text{-}above\ (range\ X)$
**for** $X :: nat \Rightarrow real$
$\langle proof \rangle$

**lemma** *Bseq-bdd-above'*: $Bseq\ X \Longrightarrow bdd\text{-}above\ (range\ (\lambda n.\ norm\ (X\ n)))$
**for** $X :: nat \Rightarrow 'a :: real\text{-}normed\text{-}vector$
$\langle proof \rangle$

**lemma** *Bseq-bdd-below*: $Bseq\ X \Longrightarrow bdd\text{-}below\ (range\ X)$
**for** $X :: nat \Rightarrow real$
$\langle proof \rangle$

**lemma** *Bseq-eventually-mono*:
**assumes** $eventually\ (\lambda n.\ norm\ (f\ n) \leq norm\ (g\ n))\ sequentially\ Bseq\ g$
**shows** $Bseq\ f$
$\langle proof \rangle$

**lemma** *lemma-NBseq-def*: $(\exists K > 0.\ \forall n.\ norm\ (X\ n) \leq K) \longleftrightarrow (\exists N.\ \forall n.\ norm\ (X\ n) \leq real(Suc\ N))$
$\langle proof \rangle$

Alternative definition for *Bseq*.

**lemma** *Bseq-iff*: $Bseq\ X \longleftrightarrow (\exists N.\ \forall n.\ norm\ (X\ n) \leq real(Suc\ N))$
$\langle proof \rangle$

**lemma** *lemma-NBseq-def2*: $(\exists K > 0.\ \forall n.\ norm\ (X\ n) \leq K) = (\exists N.\ \forall n.\ norm\ (X\ n) < real(Suc\ N))$
$\langle proof \rangle$

Yet another definition for Bseq.

**lemma** *Bseq-iff1a*: $Bseq\ X \longleftrightarrow (\exists N.\ \forall n.\ norm\ (X\ n) < real\ (Suc\ N))$
⟨*proof*⟩

### 108.1.3 A Few More Equivalence Theorems for Boundedness

Alternative formulation for boundedness.

**lemma** *Bseq-iff2*: $Bseq\ X \longleftrightarrow (\exists k > 0.\ \exists x.\ \forall n.\ norm\ (X\ n + -\ x) \le k)$
⟨*proof*⟩

Alternative formulation for boundedness.

**lemma** *Bseq-iff3*: $Bseq\ X \longleftrightarrow (\exists k{>}0.\ \exists N.\ \forall n.\ norm\ (X\ n + -\ X\ N) \le k)$
(**is** $?P \longleftrightarrow ?Q$)
⟨*proof*⟩

### 108.1.4 Upper Bounds and Lubs of Bounded Sequences

**lemma** *Bseq-minus-iff*: $Bseq\ (\lambda n.\ -\ (X\ n) :: 'a::real\text{-}normed\text{-}vector) \longleftrightarrow Bseq\ X$
⟨*proof*⟩

**lemma** *Bseq-add*:
**fixes** $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
**assumes** *Bseq f*
**shows** $Bseq\ (\lambda x.\ f\ x + c)$
⟨*proof*⟩

**lemma** *Bseq-add-iff*: $Bseq\ (\lambda x.\ f\ x + c) \longleftrightarrow Bseq\ f$
**for** $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *Bseq-mult*:
**fixes** $f\ g :: nat \Rightarrow 'a::real\text{-}normed\text{-}field$
**assumes** *Bseq f* **and** *Bseq g*
**shows** $Bseq\ (\lambda x.\ f\ x * g\ x)$
⟨*proof*⟩

**lemma** *Bfun-const* [*simp*]: $Bfun\ (\lambda\text{-}.\ c)\ F$
⟨*proof*⟩

**lemma** *Bseq-cmult-iff*:
**fixes** $c :: 'a::real\text{-}normed\text{-}field$
**assumes** $c \ne 0$
**shows** $Bseq\ (\lambda x.\ c * f\ x) \longleftrightarrow Bseq\ f$
⟨*proof*⟩

**lemma** *Bseq-subseq*: $Bseq\ f \Longrightarrow Bseq\ (\lambda x.\ f\ (g\ x))$
**for** $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *Bseq-Suc-iff*: $Bseq\ (\lambda n.\ f\ (Suc\ n)) \longleftrightarrow Bseq\ f$

**for** $f$ :: $nat \Rightarrow 'a$::*real-normed-vector*
⟨*proof*⟩

**lemma** *increasing-Bseq-subseq-iff*:
**assumes** $\bigwedge x\ y.\ x \le y \Longrightarrow norm\ (f\ x :: 'a::real\text{-}normed\text{-}vector) \le norm\ (f\ y)$ *strict-mono g*
**shows** $Bseq\ (\lambda x.\ f\ (g\ x)) \longleftrightarrow Bseq\ f$
⟨*proof*⟩

**lemma** *nonneg-incseq-Bseq-subseq-iff*:
**fixes** $f$ :: $nat \Rightarrow real$
**and** $g$ :: $nat \Rightarrow nat$
**assumes** $\bigwedge x.\ f\ x \ge 0$ *incseq f strict-mono g*
**shows** $Bseq\ (\lambda x.\ f\ (g\ x)) \longleftrightarrow Bseq\ f$
⟨*proof*⟩

**lemma** *Bseq-eq-bounded*: $range\ f \subseteq \{a..b\} \Longrightarrow Bseq\ f$
**for** $a\ b$ :: *real*
⟨*proof*⟩

**lemma** *incseq-bounded*: $incseq\ X \Longrightarrow \forall i.\ X\ i \le B \Longrightarrow Bseq\ X$
**for** $B$ :: *real*
⟨*proof*⟩

**lemma** *decseq-bounded*: $decseq\ X \Longrightarrow \forall i.\ B \le X\ i \Longrightarrow Bseq\ X$
**for** $B$ :: *real*
⟨*proof*⟩

### 108.1.5 Polynomal function extremal theorem, from HOL Light

**lemma** *polyfun-extremal-lemma*:
**fixes** $c$ :: $nat \Rightarrow 'a$::*real-normed-div-algebra*
**assumes** $0 < e$
**shows** $\exists M.\ \forall z.\ M \le norm(z) \longrightarrow norm\ (\sum i{\le}n.\ c(i) * z\hat{\ }i) \le e * norm(z)\ \hat{\ }(Suc\ n)$
⟨*proof*⟩

**lemma** *polyfun-extremal*:
**fixes** $c$ :: $nat \Rightarrow 'a$::*real-normed-div-algebra*
**assumes** $k$: $c\ k \ne 0$ $1{\le}k$ **and** $kn$: $k{\le}n$
**shows** *eventually* $(\lambda z.\ norm\ (\sum i{\le}n.\ c(i) * z\hat{\ }i) \ge B)$ *at-infinity*
⟨*proof*⟩

## 108.2 Convergence to Zero

**definition** *Zfun* :: $('a \Rightarrow 'b::real\text{-}normed\text{-}vector) \Rightarrow 'a\ filter \Rightarrow bool$
**where** $Zfun\ f\ F = (\forall r{>}0.\ eventually\ (\lambda x.\ norm\ (f\ x) < r)\ F)$

**lemma** *ZfunI*: $(\bigwedge r.\ 0 < r \Longrightarrow eventually\ (\lambda x.\ norm\ (f\ x) < r)\ F) \Longrightarrow Zfun\ f\ F$
⟨*proof*⟩

**lemma** *ZfunD*: $Zfun\ f\ F \Longrightarrow 0 < r \Longrightarrow eventually\ (\lambda x.\ norm\ (f\ x) < r)\ F$
⟨*proof*⟩

**lemma** *Zfun-ssubst*: $eventually\ (\lambda x.\ f\ x = g\ x)\ F \Longrightarrow Zfun\ g\ F \Longrightarrow Zfun\ f\ F$
⟨*proof*⟩

**lemma** *Zfun-zero*: $Zfun\ (\lambda x.\ 0)\ F$
⟨*proof*⟩

**lemma** *Zfun-norm-iff*: $Zfun\ (\lambda x.\ norm\ (f\ x))\ F = Zfun\ (\lambda x.\ f\ x)\ F$
⟨*proof*⟩

**lemma** *Zfun-imp-Zfun*:
**assumes** *f*: $Zfun\ f\ F$
**and** *g*: $eventually\ (\lambda x.\ norm\ (g\ x) \leq norm\ (f\ x) * K)\ F$
**shows** $Zfun\ (\lambda x.\ g\ x)\ F$
⟨*proof*⟩

**lemma** *Zfun-le*: $Zfun\ g\ F \Longrightarrow \forall x.\ norm\ (f\ x) \leq norm\ (g\ x) \Longrightarrow Zfun\ f\ F$
⟨*proof*⟩

**lemma** *Zfun-add*:
**assumes** *f*: $Zfun\ f\ F$
**and** *g*: $Zfun\ g\ F$
**shows** $Zfun\ (\lambda x.\ f\ x + g\ x)\ F$
⟨*proof*⟩

**lemma** *Zfun-minus*: $Zfun\ f\ F \Longrightarrow Zfun\ (\lambda x.\ -\ f\ x)\ F$
⟨*proof*⟩

**lemma** *Zfun-diff*: $Zfun\ f\ F \Longrightarrow Zfun\ g\ F \Longrightarrow Zfun\ (\lambda x.\ f\ x - g\ x)\ F$
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *Zfun*:
**assumes** *g*: $Zfun\ g\ F$
**shows** $Zfun\ (\lambda x.\ f\ (g\ x))\ F$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *Zfun*:
**assumes** *f*: $Zfun\ f\ F$
**and** *g*: $Zfun\ g\ F$
**shows** $Zfun\ (\lambda x.\ f\ x ** g\ x)\ F$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *Zfun-left*: $Zfun\ f\ F \Longrightarrow Zfun\ (\lambda x.\ f\ x ** a)\ F$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *Zfun-right*: $Zfun\ f\ F \Longrightarrow Zfun\ (\lambda x.\ a ** f\ x)\ F$

⟨*proof*⟩

**lemmas** *Zfun-mult* = *bounded-bilinear.Zfun* [*OF bounded-bilinear-mult*]
**lemmas** *Zfun-mult-right* = *bounded-bilinear.Zfun-right* [*OF bounded-bilinear-mult*]
**lemmas** *Zfun-mult-left* = *bounded-bilinear.Zfun-left* [*OF bounded-bilinear-mult*]

**lemma** *tendsto-Zfun-iff*: $(f \longrightarrow a)\ F = Zfun\ (\lambda x.\ f\ x - a)\ F$
⟨*proof*⟩

**lemma** *tendsto-0-le*:
$(f \longrightarrow 0)\ F \Longrightarrow eventually\ (\lambda x.\ norm\ (g\ x) \le norm\ (f\ x) * K)\ F \Longrightarrow (g \longrightarrow 0)\ F$
⟨*proof*⟩

### 108.2.1 Distance and norms

**lemma** *tendsto-dist* [*tendsto-intros*]:
**fixes** *l m* :: *'a::metric-space*
**assumes** *f*: $(f \longrightarrow l)\ F$
**and** *g*: $(g \longrightarrow m)\ F$
**shows** $((\lambda x.\ dist\ (f\ x)\ (g\ x)) \longrightarrow dist\ l\ m)\ F$
⟨*proof*⟩

**lemma** *continuous-dist*[*continuous-intros*]:
**fixes** *f g* :: - ⇒ *'a* :: *metric-space*
**shows** $continuous\ F\ f \Longrightarrow continuous\ F\ g \Longrightarrow continuous\ F\ (\lambda x.\ dist\ (f\ x)\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-on-dist*[*continuous-intros*]:
**fixes** *f g* :: - ⇒ *'a* :: *metric-space*
**shows** $continuous\text{-}on\ s\ f \Longrightarrow continuous\text{-}on\ s\ g \Longrightarrow continuous\text{-}on\ s\ (\lambda x.\ dist\ (f\ x)\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-at-dist*: *isCont* (*dist a*) *b*
⟨*proof*⟩

**lemma** *tendsto-norm* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ norm\ (f\ x)) \longrightarrow norm\ a)\ F$
⟨*proof*⟩

**lemma** *continuous-norm* [*continuous-intros*]: $continuous\ F\ f \Longrightarrow continuous\ F\ (\lambda x.\ norm\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-norm* [*continuous-intros*]:
$continuous\text{-}on\ s\ f \Longrightarrow continuous\text{-}on\ s\ (\lambda x.\ norm\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-norm-id* [*continuous-intros*]: *continuous-on S norm*
⟨*proof*⟩

**lemma** *tendsto-norm-zero*: $(f \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ norm\ (f\ x)) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-norm-zero-cancel*: $((\lambda x.\ norm\ (f\ x)) \longrightarrow 0)\ F \Longrightarrow (f \longrightarrow 0)$
$F$
⟨*proof*⟩

**lemma** *tendsto-norm-zero-iff*: $((\lambda x.\ norm\ (f\ x)) \longrightarrow 0)\ F \longleftrightarrow (f \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-rabs* [*tendsto-intros*]: $(f \longrightarrow l)\ F \Longrightarrow ((\lambda x.\ |f\ x|) \longrightarrow |l|)\ F$
**for** $l$ :: *real*
⟨*proof*⟩

**lemma** *continuous-rabs* [*continuous-intros*]:
*continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ |f\ x :: real|)$
⟨*proof*⟩

**lemma** *continuous-on-rabs* [*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ |f\ x :: real|)$
⟨*proof*⟩

**lemma** *tendsto-rabs-zero*: $(f \longrightarrow (0::real))\ F \Longrightarrow ((\lambda x.\ |f\ x|) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-rabs-zero-cancel*: $((\lambda x.\ |f\ x|) \longrightarrow (0::real))\ F \Longrightarrow (f \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-rabs-zero-iff*: $((\lambda x.\ |f\ x|) \longrightarrow (0::real))\ F \longleftrightarrow (f \longrightarrow 0)\ F$
⟨*proof*⟩

## 108.3 Topological Monoid

**class** *topological-monoid-add* = *topological-space* + *monoid-add* +
**assumes** *tendsto-add-Pair*: *LIM x* (*nhds a* $\times_F$ *nhds b*). *fst x* + *snd x* :> *nhds* (*a*
+ *b*)

**class** *topological-comm-monoid-add* = *topological-monoid-add* + *comm-monoid-add*

**lemma** *tendsto-add* [*tendsto-intros*]:
**fixes** *a b* :: *'a::topological-monoid-add*
**shows** $(f \longrightarrow a)\ F \Longrightarrow (g \longrightarrow b)\ F \Longrightarrow ((\lambda x.\ f\ x + g\ x) \longrightarrow a + b)\ F$
⟨*proof*⟩

**lemma** *continuous-add* [*continuous-intros*]:

**fixes** $f$ $g$ :: - $\Rightarrow$ $'b$::*topological-monoid-add*
**shows** *continuous* $F$ $f$ $\Longrightarrow$ *continuous* $F$ $g$ $\Longrightarrow$ *continuous* $F$ $(\lambda x.\ f\ x + g\ x)$
⟨*proof*⟩

**lemma** *continuous-on-add* [*continuous-intros*]:
**fixes** $f$ $g$ :: - $\Rightarrow$ $'b$::*topological-monoid-add*
**shows** *continuous-on* $s$ $f$ $\Longrightarrow$ *continuous-on* $s$ $g$ $\Longrightarrow$ *continuous-on* $s$ $(\lambda x.\ f\ x + g\ x)$
⟨*proof*⟩

**lemma** *tendsto-add-zero*:
**fixes** $f$ $g$ :: - $\Rightarrow$ $'b$::*topological-monoid-add*
**shows** $(f \longrightarrow 0)\ F \Longrightarrow (g \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ f\ x + g\ x) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-sum* [*tendsto-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c$::*topological-comm-monoid-add*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow (f\ i \longrightarrow a\ i)\ F) \Longrightarrow ((\lambda x.\ \sum i{\in}I.\ f\ i\ x) \longrightarrow (\sum i{\in}I.\ a\ i))\ F$
⟨*proof*⟩

**lemma** *tendsto-null-sum*:
**fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c$::*topological-comm-monoid-add*
**assumes** $\bigwedge i.\ i \in I \Longrightarrow ((\lambda x.\ f\ x\ i) \longrightarrow 0)\ F$
**shows** $((\lambda i.\ sum\ (f\ i)\ I) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *continuous-sum* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b$::*t2-space* $\Rightarrow$ $'c$::*topological-comm-monoid-add*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow$ *continuous* $F\ (f\ i)) \Longrightarrow$ *continuous* $F\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x)$
⟨*proof*⟩

**lemma** *continuous-on-sum* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b$::*topological-space* $\Rightarrow$ $'c$::*topological-comm-monoid-add*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow$ *continuous-on* $S\ (f\ i)) \Longrightarrow$ *continuous-on* $S\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x)$
⟨*proof*⟩

**instance** *nat* :: *topological-comm-monoid-add*
⟨*proof*⟩

**instance** *int* :: *topological-comm-monoid-add*
⟨*proof*⟩

### 108.3.1 Topological group

**class** *topological-group-add* = *topological-monoid-add* + *group-add* +
**assumes** *tendsto-uminus-nhds*: $(uminus \longrightarrow -\ a)\ (nhds\ a)$
**begin**

**lemma** *tendsto-minus* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ - f\ x) \longrightarrow - a)$
$F$
  ⟨*proof*⟩

**end**

**class** *topological-ab-group-add* = *topological-group-add* + *ab-group-add*

**instance** *topological-ab-group-add* < *topological-comm-monoid-add* ⟨*proof*⟩

**lemma** *continuous-minus* [*continuous-intros*]: *continuous* $F$ $f \Longrightarrow$ *continuous* $F$
$(\lambda x.\ - f\ x)$
  **for** $f$ :: $'a$::*t2-space* ⇒ $'b$::*topological-group-add*
  ⟨*proof*⟩

**lemma** *continuous-on-minus* [*continuous-intros*]: *continuous-on* $s$ $f \Longrightarrow$ *continu-ous-on* $s$ $(\lambda x.\ - f\ x)$
  **for** $f$ :: - ⇒ $'b$::*topological-group-add*
  ⟨*proof*⟩

**lemma** *tendsto-minus-cancel*: $((\lambda x.\ - f\ x) \longrightarrow - a)\ F \Longrightarrow (f \longrightarrow a)\ F$
  **for** $a$ :: $'a$::*topological-group-add*
  ⟨*proof*⟩

**lemma** *tendsto-minus-cancel-left*:
  $(f \longrightarrow - (y$::-::*topological-group-add*$))\ F \longleftrightarrow ((\lambda x.\ - f\ x) \longrightarrow y)\ F$
  ⟨*proof*⟩

**lemma** *tendsto-diff* [*tendsto-intros*]:
  **fixes** $a\ b$ :: $'a$::*topological-group-add*
  **shows** $(f \longrightarrow a)\ F \Longrightarrow (g \longrightarrow b)\ F \Longrightarrow ((\lambda x.\ f\ x - g\ x) \longrightarrow a - b)\ F$
  ⟨*proof*⟩

**lemma** *continuous-diff* [*continuous-intros*]:
  **fixes** $f\ g$ :: $'a$::*t2-space* ⇒ $'b$::*topological-group-add*
  **shows** *continuous* $F$ $f \Longrightarrow$ *continuous* $F$ $g \Longrightarrow$ *continuous* $F$ $(\lambda x.\ f\ x - g\ x)$
  ⟨*proof*⟩

**lemma** *continuous-on-diff* [*continuous-intros*]:
  **fixes** $f\ g$ :: - ⇒ $'b$::*topological-group-add*
  **shows** *continuous-on* $s$ $f \Longrightarrow$ *continuous-on* $s$ $g \Longrightarrow$ *continuous-on* $s$ $(\lambda x.\ f\ x -$
$g\ x)$
  ⟨*proof*⟩

**lemma** *continuous-on-op-minus*: *continuous-on* $(s$::$'a$::*topological-group-add set*$)$
$((-)\ x)$
  ⟨*proof*⟩

**instance** *real-normed-vector* < *topological-ab-group-add*
⟨*proof*⟩

**lemmas** *real-tendsto-sandwich* = *tendsto-sandwich*[**where** *'a*=*real*]

### 108.3.2 Linear operators and multiplication

**lemma** *linear-times* [*simp*]: *linear* ($\lambda x.\ c * x$)
  **for** $c$ :: *'a*::*real-algebra*
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *tendsto*: $(g \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ f\ (g\ x)) \longrightarrow f\ a)\ F$
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *continuous*: *continuous* $F$ $g$ $\Longrightarrow$ *continuous* $F$ $(\lambda x.\ f$ $(g\ x))$
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *continuous-on*: *continuous-on* $s$ $g$ $\Longrightarrow$ *continuous-on* $s$ $(\lambda x.\ f\ (g\ x))$
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *tendsto-zero*: $(g \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ f\ (g\ x)) \longrightarrow 0)\ F$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *tendsto*:
  $(f \longrightarrow a)\ F \Longrightarrow (g \longrightarrow b)\ F \Longrightarrow ((\lambda x.\ f\ x ** g\ x) \longrightarrow a ** b)\ F$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *continuous*:
  *continuous* $F$ $f$ $\Longrightarrow$ *continuous* $F$ $g$ $\Longrightarrow$ *continuous* $F$ $(\lambda x.\ f\ x ** g\ x)$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *continuous-on*:
  *continuous-on* $s$ $f$ $\Longrightarrow$ *continuous-on* $s$ $g$ $\Longrightarrow$ *continuous-on* $s$ $(\lambda x.\ f\ x ** g\ x)$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *tendsto-zero*:
  **assumes** $f$: $(f \longrightarrow 0)\ F$
    **and** $g$: $(g \longrightarrow 0)\ F$
  **shows** $((\lambda x.\ f\ x ** g\ x) \longrightarrow 0)\ F$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *tendsto-left-zero*:
  $(f \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ f\ x ** c) \longrightarrow 0)\ F$
  ⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *tendsto-right-zero*:

$(f \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ c ** f\,x) \longrightarrow 0)\ F$
⟨proof⟩

**lemmas** *tendsto-of-real* [*tendsto-intros*] =
*bounded-linear.tendsto* [*OF bounded-linear-of-real*]

**lemmas** *tendsto-scaleR* [*tendsto-intros*] =
*bounded-bilinear.tendsto* [*OF bounded-bilinear-scaleR*]

Analogous type class for multiplication

**class** *topological-semigroup-mult* = *topological-space* + *semigroup-mult* +
**assumes** *tendsto-mult-Pair*: $LIM\ x\ (nhds\ a \times_F nhds\ b).\ fst\ x * snd\ x :> nhds\ (a * b)$

**instance** *real-normed-algebra* < *topological-semigroup-mult*
⟨proof⟩

**lemma** *tendsto-mult* [*tendsto-intros*]:
**fixes** $a\ b$ :: $'a$::*topological-semigroup-mult*
**shows** $(f \longrightarrow a)\ F \Longrightarrow (g \longrightarrow b)\ F \Longrightarrow ((\lambda x.\ f\,x * g\,x) \longrightarrow a * b)\ F$
⟨proof⟩

**lemma** *tendsto-mult-left*: $(f \longrightarrow l)\ F \Longrightarrow ((\lambda x.\ c * (f\,x)) \longrightarrow c * l)\ F$
**for** $c$ :: $'a$::*topological-semigroup-mult*
⟨proof⟩

**lemma** *tendsto-mult-right*: $(f \longrightarrow l)\ F \Longrightarrow ((\lambda x.\ (f\,x) * c) \longrightarrow l * c)\ F$
**for** $c$ :: $'a$::*topological-semigroup-mult*
⟨proof⟩

**lemma** *tendsto-mult-left-iff* [*simp*]:
$c \neq 0 \Longrightarrow tendsto(\lambda x.\ c * f\,x)\ (c * l)\ F \longleftrightarrow tendsto\ f\ l\ F$ **for** $c$ :: $'a$::{*topological-semigroup-mult*,*field*}
⟨proof⟩

**lemma** *tendsto-mult-right-iff* [*simp*]:
$c \neq 0 \Longrightarrow tendsto(\lambda x.\ f\,x * c)\ (l * c)\ F \longleftrightarrow tendsto\ f\ l\ F$ **for** $c$ :: $'a$::{*topological-semigroup-mult*,*field*}
⟨proof⟩

**lemma** *tendsto-zero-mult-left-iff* [*simp*]:
**fixes** $c$::$'a$::{*topological-semigroup-mult*,*field*} **assumes** $c \neq 0$ **shows** $(\lambda n.\ c * a\ n) \longrightarrow 0 \longleftrightarrow a \longrightarrow 0$
⟨proof⟩

**lemma** *tendsto-zero-mult-right-iff* [*simp*]:
**fixes** $c$::$'a$::{*topological-semigroup-mult*,*field*} **assumes** $c \neq 0$ **shows** $(\lambda n.\ a\ n * c) \longrightarrow 0 \longleftrightarrow a \longrightarrow 0$
⟨proof⟩

**lemma** *tendsto-zero-divide-iff* [*simp*]:

**fixes** $c$::$'a$::{*topological-semigroup-mult,field*} **assumes** $c \neq 0$ **shows** $(\lambda n.\ a\ n\ /\ c) \longrightarrow 0 \longleftrightarrow a \longrightarrow 0$
⟨*proof*⟩

**lemma** *lim-const-over-n* [*tendsto-intros*]:
**fixes** $a$ :: $'a$::*real-normed-field*
**shows** $(\lambda n.\ a\ /\ \textit{of-nat}\ n) \longrightarrow 0$
⟨*proof*⟩

**lemmas** *continuous-of-real* [*continuous-intros*] =
*bounded-linear.continuous* [*OF bounded-linear-of-real*]

**lemmas** *continuous-scaleR* [*continuous-intros*] =
*bounded-bilinear.continuous* [*OF bounded-bilinear-scaleR*]

**lemmas** *continuous-mult* [*continuous-intros*] =
*bounded-bilinear.continuous* [*OF bounded-bilinear-mult*]

**lemmas** *continuous-on-of-real* [*continuous-intros*] =
*bounded-linear.continuous-on* [*OF bounded-linear-of-real*]

**lemmas** *continuous-on-scaleR* [*continuous-intros*] =
*bounded-bilinear.continuous-on* [*OF bounded-bilinear-scaleR*]

**lemmas** *continuous-on-mult* [*continuous-intros*] =
*bounded-bilinear.continuous-on* [*OF bounded-bilinear-mult*]

**lemmas** *tendsto-mult-zero* =
*bounded-bilinear.tendsto-zero* [*OF bounded-bilinear-mult*]

**lemmas** *tendsto-mult-left-zero* =
*bounded-bilinear.tendsto-left-zero* [*OF bounded-bilinear-mult*]

**lemmas** *tendsto-mult-right-zero* =
*bounded-bilinear.tendsto-right-zero* [*OF bounded-bilinear-mult*]

**lemma** *continuous-mult-left*:
**fixes** $c$::$'a$::*real-normed-algebra*
**shows** $\textit{continuous}\ F\ f \Longrightarrow \textit{continuous}\ F\ (\lambda x.\ c * f\ x)$
⟨*proof*⟩

**lemma** *continuous-mult-right*:
**fixes** $c$::$'a$::*real-normed-algebra*
**shows** $\textit{continuous}\ F\ f \Longrightarrow \textit{continuous}\ F\ (\lambda x.\ f\ x * c)$
⟨*proof*⟩

**lemma** *continuous-on-mult-left*:
**fixes** $c$::$'a$::*real-normed-algebra*

**shows** *continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ c * f\ x)$
⟨*proof*⟩

**lemma** *continuous-on-mult-right*:
**fixes** *c*::′*a*::*real-normed-algebra*
**shows** *continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ f\ x * c)$
⟨*proof*⟩

**lemma** *continuous-on-mult-const* [*simp*]:
**fixes** *c*::′*a*::*real-normed-algebra*
**shows** *continuous-on s* $((*)\ c)$
⟨*proof*⟩

**lemma** *tendsto-divide-zero*:
**fixes** *c* :: ′*a*::*real-normed-field*
**shows** $(f \longrightarrow 0)\ F \Longrightarrow ((\lambda x.\ f\ x\ /\ c) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *tendsto-power* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ f\ x\ \hat{}\ n) \longrightarrow a\ \hat{}\ n)\ F$
**for** *f* :: ′*a* ⇒ ′*b*::{*power*,*real-normed-algebra*}
⟨*proof*⟩

**lemma** *tendsto-null-power*: $[\![(f \longrightarrow 0)\ F;\ 0 < n]\!] \Longrightarrow ((\lambda x.\ f\ x\ \hat{}\ n) \longrightarrow 0)\ F$
**for** *f* :: ′*a* ⇒ ′*b*::{*power*,*real-normed-algebra-1*}
⟨*proof*⟩

**lemma** *continuous-power* [*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ (f\ x)\hat{}n)$
**for** *f* :: ′*a*::*t2-space* ⇒ ′*b*::{*power*,*real-normed-algebra*}
⟨*proof*⟩

**lemma** *continuous-on-power* [*continuous-intros*]:
**fixes** *f* :: - ⇒ ′*b*::{*power*,*real-normed-algebra*}
**shows** *continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ (f\ x)\hat{}n)$
⟨*proof*⟩

**lemma** *tendsto-prod* [*tendsto-intros*]:
**fixes** *f* :: ′*a* ⇒ ′*b* ⇒ ′*c*::{*real-normed-algebra*,*comm-ring-1*}
**shows** $(\bigwedge i.\ i \in S \Longrightarrow (f\ i \longrightarrow L\ i)\ F) \Longrightarrow ((\lambda x.\ \prod i{\in}S.\ f\ i\ x) \longrightarrow (\prod i{\in}S.\ L\ i))\ F$
⟨*proof*⟩

**lemma** *continuous-prod* [*continuous-intros*]:
**fixes** *f* :: ′*a* ⇒ ′*b*::*t2-space* ⇒ ′*c*::{*real-normed-algebra*,*comm-ring-1*}
**shows** $(\bigwedge i.\ i \in S \Longrightarrow$ *continuous F* $(f\ i)) \Longrightarrow$ *continuous F* $(\lambda x.\ \prod i{\in}S.\ f\ i\ x)$
⟨*proof*⟩

**lemma** *continuous-on-prod* [*continuous-intros*]:

**fixes** $f$ :: $'a \Rightarrow - \Rightarrow 'c$::{*real-normed-algebra,comm-ring-1*}
**shows** $(\bigwedge i.\ i \in S \Longrightarrow$ *continuous-on* $s$ $(f\ i)) \Longrightarrow$ *continuous-on* $s$ $(\lambda x.\ \prod i{\in}S.\ f\ i\ x)$
⟨*proof*⟩

**lemma** *tendsto-of-real-iff*:
$((\lambda x.$ *of-real* $(f\ x)$ :: $'a$::*real-normed-div-algebra*$) \longrightarrow$ *of-real* $c)$ $F \longleftrightarrow (f \longrightarrow c)$ $F$
⟨*proof*⟩

**lemma** *tendsto-add-const-iff*:
$((\lambda x.\ c + f\ x$ :: $'a$::*topological-group-add*$) \longrightarrow c + d)$ $F \longleftrightarrow (f \longrightarrow d)$ $F$
⟨*proof*⟩

**class** *topological-monoid-mult* = *topological-semigroup-mult* + *monoid-mult*
**class** *topological-comm-monoid-mult* = *topological-monoid-mult* + *comm-monoid-mult*

**lemma** *tendsto-power-strong* [*tendsto-intros*]:
**fixes** $f$ :: $- \Rightarrow 'b$ :: *topological-monoid-mult*
**assumes** $(f \longrightarrow a)$ $F$ $(g \longrightarrow b)$ $F$
**shows** $((\lambda x.\ f\ x\ \widehat{}\ g\ x) \longrightarrow a\ \widehat{}\ b)$ $F$
⟨*proof*⟩

**lemma** *continuous-mult′* [*continuous-intros*]:
**fixes** $f\ g$ :: $- \Rightarrow 'b$::*topological-semigroup-mult*
**shows** *continuous* $F$ $f \Longrightarrow$ *continuous* $F$ $g \Longrightarrow$ *continuous* $F$ $(\lambda x.\ f\ x * g\ x)$
⟨*proof*⟩

**lemma** *continuous-power′* [*continuous-intros*]:
**fixes** $f$ :: $- \Rightarrow 'b$::*topological-monoid-mult*
**shows** *continuous* $F$ $f \Longrightarrow$ *continuous* $F$ $g \Longrightarrow$ *continuous* $F$ $(\lambda x.\ f\ x\ \widehat{}\ g\ x)$
⟨*proof*⟩

**lemma** *continuous-on-mult′* [*continuous-intros*]:
**fixes** $f\ g$ :: $- \Rightarrow 'b$::*topological-semigroup-mult*
**shows** *continuous-on* $A$ $f \Longrightarrow$ *continuous-on* $A$ $g \Longrightarrow$ *continuous-on* $A$ $(\lambda x.\ f\ x * g\ x)$
⟨*proof*⟩

**lemma** *continuous-on-power′* [*continuous-intros*]:
**fixes** $f$ :: $- \Rightarrow 'b$::*topological-monoid-mult*
**shows** *continuous-on* $A$ $f \Longrightarrow$ *continuous-on* $A$ $g \Longrightarrow$ *continuous-on* $A$ $(\lambda x.\ f\ x\ \widehat{}\ g\ x)$
⟨*proof*⟩

**lemma** *tendsto-mult-one*:
**fixes** $f\ g$ :: $- \Rightarrow 'b$::*topological-monoid-mult*
**shows** $(f \longrightarrow 1)$ $F \Longrightarrow (g \longrightarrow 1)$ $F \Longrightarrow ((\lambda x.\ f\ x * g\ x) \longrightarrow 1)$ $F$

⟨proof⟩

**lemma** *tendsto-prod'* [*tendsto-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c$::*topological-comm-monoid-mult*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow (f\ i \longrightarrow a\ i)\ F) \Longrightarrow ((\lambda x.\ \prod i{\in}I.\ f\ i\ x) \longrightarrow (\prod i{\in}I.\ a\ i))\ F$
⟨proof⟩

**lemma** *tendsto-one-prod'*:
**fixes** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c$::*topological-comm-monoid-mult*
**assumes** $\bigwedge i.\ i \in I \Longrightarrow ((\lambda x.\ f\ x\ i) \longrightarrow 1)\ F$
**shows** $((\lambda i.\ prod\ (f\ i)\ I) \longrightarrow 1)\ F$
⟨proof⟩

**lemma** *LIMSEQ-prod-0*:
**fixes** $f$ :: $nat \Rightarrow 'a$::{*semidom*,*topological-space*}
**assumes** $f\ i = 0$
**shows** $(\lambda n.\ prod\ f\ \{..n\}) \longrightarrow 0$
⟨proof⟩

**lemma** *LIMSEQ-prod-nonneg*:
**fixes** $f$ :: $nat \Rightarrow 'a$::{*linordered-semidom*,*linorder-topology*}
**assumes** *0*: $\bigwedge n.\ 0 \le f\ n$ **and** *a*: $(\lambda n.\ prod\ f\ \{..n\}) \longrightarrow a$
**shows** $a \ge 0$
⟨proof⟩

**lemma** *continuous-prod'* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b$::*t2-space* $\Rightarrow 'c$::*topological-comm-monoid-mult*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow continuous\ F\ (f\ i)) \Longrightarrow continuous\ F\ (\lambda x.\ \prod i{\in}I.\ f\ i\ x)$
⟨proof⟩

**lemma** *continuous-on-prod'* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'b$::*topological-space* $\Rightarrow 'c$::*topological-comm-monoid-mult*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow continuous\text{-}on\ S\ (f\ i)) \Longrightarrow continuous\text{-}on\ S\ (\lambda x.\ \prod i{\in}I.\ f\ i\ x)$
⟨proof⟩

**instance** *nat* :: *topological-comm-monoid-mult*
⟨proof⟩

**instance** *int* :: *topological-comm-monoid-mult*
⟨proof⟩

**class** *comm-real-normed-algebra-1* = *real-normed-algebra-1* + *comm-monoid-mult*

**context** *real-normed-field*
**begin**

**subclass** *comm-real-normed-algebra-1*

⟨*proof*⟩

**end**

### 108.3.3 Inverse and division

**lemma** (**in** *bounded-bilinear*) *Zfun-prod-Bfun*:
  **assumes** *f*: *Zfun f F*
    **and** *g*: *Bfun g F*
  **shows** *Zfun* ($\lambda x.$ *f x* $**$ *g x*) *F*
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *Bfun-prod-Zfun*:
  **assumes** *f*: *Bfun f F*
    **and** *g*: *Zfun g F*
  **shows** *Zfun* ($\lambda x.$ *f x* $**$ *g x*) *F*
  ⟨*proof*⟩

**lemma** *Bfun-inverse*:
  **fixes** *a* :: *'a::real-normed-div-algebra*
  **assumes** *f*: (*f* $\longrightarrow$ *a*) *F*
  **assumes** *a*: $a \neq 0$
  **shows** *Bfun* ($\lambda x.$ *inverse* (*f x*)) *F*
⟨*proof*⟩

**lemma** *tendsto-inverse* [*tendsto-intros*]:
  **fixes** *a* :: *'a::real-normed-div-algebra*
  **assumes** *f*: (*f* $\longrightarrow$ *a*) *F*
    **and** *a*: $a \neq 0$
  **shows** (($\lambda x.$ *inverse* (*f x*)) $\longrightarrow$ *inverse a*) *F*
⟨*proof*⟩

**lemma** *continuous-inverse*:
  **fixes** *f* :: *'a::t2-space* $\Rightarrow$ *'b::real-normed-div-algebra*
  **assumes** *continuous F f*
    **and** *f* (*Lim F* ($\lambda x.$ *x*)) $\neq 0$
  **shows** *continuous F* ($\lambda x.$ *inverse* (*f x*))
  ⟨*proof*⟩

**lemma** *continuous-at-within-inverse*[*continuous-intros*]:
  **fixes** *f* :: *'a::t2-space* $\Rightarrow$ *'b::real-normed-div-algebra*
  **assumes** *continuous* (*at a within s*) *f*
    **and** *f a* $\neq 0$
  **shows** *continuous* (*at a within s*) ($\lambda x.$ *inverse* (*f x*))
  ⟨*proof*⟩

**lemma** *continuous-on-inverse*[*continuous-intros*]:
  **fixes** *f* :: *'a::topological-space* $\Rightarrow$ *'b::real-normed-div-algebra*
  **assumes** *continuous-on s f*

**and** $\forall x\in s.\ f\ x \neq 0$
**shows** *continuous-on* $s$ $(\lambda x.\ inverse\ (f\ x))$
⟨*proof*⟩

**lemma** *tendsto-divide* [*tendsto-intros*]:
**fixes** $a\ b :: 'a::real\text{-}normed\text{-}field$
**shows** $(f \longrightarrow a)\ F \Longrightarrow (g \longrightarrow b)\ F \Longrightarrow b \neq 0 \Longrightarrow ((\lambda x.\ f\ x\ /\ g\ x) \longrightarrow a\ /\ b)\ F$
⟨*proof*⟩

**lemma** *continuous-divide*:
**fixes** $f\ g :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}field$
**assumes** *continuous* $F\ f$
**and** *continuous* $F\ g$
**and** $g\ (Lim\ F\ (\lambda x.\ x)) \neq 0$
**shows** *continuous* $F\ (\lambda x.\ (f\ x)\ /\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-at-within-divide*[*continuous-intros*]:
**fixes** $f\ g :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}field$
**assumes** *continuous* $(at\ a\ within\ s)\ f$ *continuous* $(at\ a\ within\ s)\ g$
**and** $g\ a \neq 0$
**shows** *continuous* $(at\ a\ within\ s)\ (\lambda x.\ (f\ x)\ /\ (g\ x))$
⟨*proof*⟩

**lemma** *isCont-divide*[*continuous-intros*, *simp*]:
**fixes** $f\ g :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}field$
**assumes** *isCont* $f\ a$ *isCont* $g\ a$ $g\ a \neq 0$
**shows** *isCont* $(\lambda x.\ (f\ x)\ /\ g\ x)\ a$
⟨*proof*⟩

**lemma** *continuous-on-divide*[*continuous-intros*]:
**fixes** $f :: 'a::topological\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}field$
**assumes** *continuous-on* $s\ f$ *continuous-on* $s\ g$
**and** $\forall x\in s.\ g\ x \neq 0$
**shows** *continuous-on* $s\ (\lambda x.\ (f\ x)\ /\ (g\ x))$
⟨*proof*⟩

**lemma** *continuous-cmult-left-iff*:
**fixes** $c::'a::real\text{-}normed\text{-}field$
**assumes** $c \neq 0$
**shows** *continuous* $F\ (\lambda x.\ c * f\ x) \longleftrightarrow$ *continuous* $F\ f$
⟨*proof*⟩

**lemma** *continuous-cmult-right-iff*:
**fixes** $c::'a::real\text{-}normed\text{-}field$
**assumes** $c \neq 0$
**shows** *continuous* $F\ (\lambda x.\ f\ x * c) \longleftrightarrow$ *continuous* $F\ f$
⟨*proof*⟩

**lemma** *continuous-cdivide-iff*:
**fixes** $c$::$'a$::*real-normed-field*
**assumes** $c \neq 0$
**shows** *continuous* $F$ $(\lambda x.\ f\ x\ /\ c) \longleftrightarrow$ *continuous* $F\ f$
⟨*proof*⟩

**lemma** *continuous-cong*:
**assumes** *eventually* $(\lambda x.\ f\ x = g\ x)$ $F$ $f$ $(Lim\ F\ (\lambda x.\ x)) = g\ (Lim\ F\ (\lambda x.\ x))$
**shows** *continuous* $F\ f \longleftrightarrow$ *continuous* $F\ g$
⟨*proof*⟩

**lemma** *continuous-at-within-cong*:
**assumes** $f\ x = g\ x$ *eventually* $(\lambda x.\ f\ x = g\ x)$ $(at\ x\ within\ S)$
**shows** *continuous* $(at\ x\ within\ S)$ $f \longleftrightarrow$ *continuous* $(at\ x\ within\ S)$ $g$
⟨*proof*⟩

**lemma** *tendsto-power-int* [*tendsto-intros*]:
**fixes** $a$ :: $'a$::*real-normed-div-algebra*
**assumes** $f$: $(f \longrightarrow a)$ $F$
**and** $a$: $a \neq 0$
**shows** $((\lambda x.\ power\text{-}int\ (f\ x)\ n) \longrightarrow power\text{-}int\ a\ n)$ $F$
⟨*proof*⟩

**lemma** *continuous-power-int*:
**fixes** $f$ :: $'a$::*t2-space* $\Rightarrow$ $'b$::*real-normed-div-algebra*
**assumes** *continuous* $F\ f$
**and** $f\ (Lim\ F\ (\lambda x.\ x)) \neq 0$
**shows** *continuous* $F$ $(\lambda x.\ power\text{-}int\ (f\ x)\ n)$
⟨*proof*⟩

**lemma** *continuous-at-within-power-int*[*continuous-intros*]:
**fixes** $f$ :: $'a$::*t2-space* $\Rightarrow$ $'b$::*real-normed-div-algebra*
**assumes** *continuous* $(at\ a\ within\ s)$ $f$
**and** $f\ a \neq 0$
**shows** *continuous* $(at\ a\ within\ s)$ $(\lambda x.\ power\text{-}int\ (f\ x)\ n)$
⟨*proof*⟩

**lemma** *continuous-on-power-int* [*continuous-intros*]:
**fixes** $f$ :: $'a$::*topological-space* $\Rightarrow$ $'b$::*real-normed-div-algebra*
**assumes** *continuous-on* $s\ f$ **and** $n \geq 0 \lor (\forall x \in s.\ f\ x \neq 0)$
**shows** *continuous-on* $s$ $(\lambda x.\ power\text{-}int\ (f\ x)\ n)$
⟨*proof*⟩

**lemma** *tendsto-power-int'* [*tendsto-intros*]:
**fixes** $a$ :: $'a$::*real-normed-div-algebra*
**assumes** $f$: $(f \longrightarrow a)$ $F$
**and** $a$: $a \neq 0 \lor n \geq 0$
**shows** $((\lambda x.\ power\text{-}int\ (f\ x)\ n) \longrightarrow power\text{-}int\ a\ n)$ $F$

⟨*proof*⟩

**lemma** *tendsto-sgn* [*tendsto-intros*]: $(f \longrightarrow l)\ F \Longrightarrow l \neq 0 \Longrightarrow ((\lambda x.\ sgn\ (f\ x)) \longrightarrow sgn\ l)\ F$
**for** $l$ :: $'a$::*real-normed-vector*
⟨*proof*⟩

**lemma** *continuous-sgn*:
**fixes** $f$ :: $'a$::*t2-space* ⇒ $'b$::*real-normed-vector*
**assumes** *continuous F f*
**and** $f\ (Lim\ F\ (\lambda x.\ x)) \neq 0$
**shows** *continuous* $F\ (\lambda x.\ sgn\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-at-within-sgn*[*continuous-intros*]:
**fixes** $f$ :: $'a$::*t2-space* ⇒ $'b$::*real-normed-vector*
**assumes** *continuous* (*at a within s*) *f*
**and** $f\ a \neq 0$
**shows** *continuous* (*at a within s*) $(\lambda x.\ sgn\ (f\ x))$
⟨*proof*⟩

**lemma** *isCont-sgn*[*continuous-intros*]:
**fixes** $f$ :: $'a$::*t2-space* ⇒ $'b$::*real-normed-vector*
**assumes** *isCont f a*
**and** $f\ a \neq 0$
**shows** *isCont* $(\lambda x.\ sgn\ (f\ x))\ a$
⟨*proof*⟩

**lemma** *continuous-on-sgn*[*continuous-intros*]:
**fixes** $f$ :: $'a$::*topological-space* ⇒ $'b$::*real-normed-vector*
**assumes** *continuous-on s f*
**and** $\forall x \in s.\ f\ x \neq 0$
**shows** *continuous-on* $s\ (\lambda x.\ sgn\ (f\ x))$
⟨*proof*⟩

**lemma** *filterlim-at-infinity*:
**fixes** $f$ :: - ⇒ $'a$::*real-normed-vector*
**assumes** $0 \leq c$
**shows** $(LIM\ x\ F.\ f\ x :> at\text{-}infinity) \longleftrightarrow (\forall r>c.\ eventually\ (\lambda x.\ r \leq norm\ (f\ x))\ F)$
⟨*proof*⟩

**lemma** *filterlim-at-infinity-imp-norm-at-top*:
**fixes** $F$
**assumes** *filterlim f at-infinity F*
**shows** *filterlim* $(\lambda x.\ norm\ (f\ x))$ *at-top F*
⟨*proof*⟩

**lemma** *filterlim-norm-at-top-imp-at-infinity*:

**fixes** $F$
**assumes** *filterlim* $(\lambda x.\ norm\ (f\ x))$ *at-top* $F$
**shows** *filterlim f at-infinity F*
⟨*proof*⟩

**lemma** *filterlim-norm-at-top*: *filterlim norm at-top at-infinity*
⟨*proof*⟩

**lemma** *filterlim-at-infinity-conv-norm-at-top*:
*filterlim f at-infinity G* $\longleftrightarrow$ *filterlim* $(\lambda x.\ norm\ (f\ x))$ *at-top* $G$
⟨*proof*⟩

**lemma** *eventually-not-equal-at-infinity*:
*eventually* $(\lambda x.\ x \neq (a :: 'a :: \{real\text{-}normed\text{-}vector\}))$ *at-infinity*
⟨*proof*⟩

**lemma** *filterlim-int-of-nat-at-topD*:
**fixes** $F$
**assumes** *filterlim* $(\lambda x.\ f\ (int\ x))$ $F$ *at-top*
**shows** *filterlim f F at-top*
⟨*proof*⟩

**lemma** *filterlim-int-sequentially* [*tendsto-intros*]:
*filterlim int at-top sequentially*
⟨*proof*⟩

**lemma** *filterlim-real-of-int-at-top* [*tendsto-intros*]:
*filterlim real-of-int at-top at-top*
⟨*proof*⟩

**lemma** *filterlim-abs-real*: *filterlim* $(abs::real \Rightarrow real)$ *at-top at-top*
⟨*proof*⟩

**lemma** *filterlim-of-real-at-infinity* [*tendsto-intros*]:
*filterlim* $(of\text{-}real :: real \Rightarrow 'a :: real\text{-}normed\text{-}algebra\text{-}1)$ *at-infinity at-top*
⟨*proof*⟩

**lemma** *not-tendsto-and-filterlim-at-infinity*:
**fixes** $c :: 'a::real\text{-}normed\text{-}vector$
**assumes** $F \neq bot$
**and** $(f \longrightarrow c)\ F$
**and** *filterlim f at-infinity F*
**shows** *False*
⟨*proof*⟩

**lemma** *filterlim-at-infinity-imp-not-convergent*:
**assumes** *filterlim f at-infinity sequentially*
**shows** $\neg$ *convergent f*
⟨*proof*⟩

**lemma** *filterlim-at-infinity-imp-eventually-ne*:
 **assumes** *filterlim f at-infinity F*
 **shows** *eventually* $(\lambda z.\ f\ z \neq c)$ *F*
⟨*proof*⟩

**lemma** *tendsto-of-nat* [*tendsto-intros*]:
 *filterlim* (*of-nat* :: *nat* ⇒ *'a*::*real-normed-algebra-1*) *at-infinity sequentially*
⟨*proof*⟩

## 108.4 Relate *at*, *at-left* and *at-right*

This lemmas are useful for conversion between *at x* to *at-left x* and *at-right x* and also *at-right 0*.

**lemmas** *filterlim-split-at-real* = *filterlim-split-at*[**where** *'a*=*real*]

**lemma** *filtermap-nhds-shift*: *filtermap* $(\lambda x.\ x - d)$ (*nhds a*) = *nhds* $(a - d)$
 **for** *a d* :: *'a*::*real-normed-vector*
 ⟨*proof*⟩

**lemma** *filtermap-nhds-minus*: *filtermap* $(\lambda x.\ - x)$ (*nhds a*) = *nhds* $(- a)$
 **for** *a* :: *'a*::*real-normed-vector*
 ⟨*proof*⟩

**lemma** *filtermap-at-shift*: *filtermap* $(\lambda x.\ x - d)$ (*at a*) = *at* $(a - d)$
 **for** *a d* :: *'a*::*real-normed-vector*
 ⟨*proof*⟩

**lemma** *filtermap-at-right-shift*: *filtermap* $(\lambda x.\ x - d)$ (*at-right a*) = *at-right* $(a - d)$
 **for** *a d* :: *real*
 ⟨*proof*⟩

**lemma** *filterlim-shift*:
 **fixes** *d* :: *'a*::*real-normed-vector*
 **assumes** *filterlim f F* (*at a*)
 **shows** *filterlim* $(f \circ (+)\ d)$ *F* (*at* $(a - d)$)
 ⟨*proof*⟩

**lemma** *filterlim-shift-iff*:
 **fixes** *d* :: *'a*::*real-normed-vector*
 **shows** *filterlim* $(f \circ (+)\ d)$ *F* (*at* $(a - d)$) = *filterlim f F* (*at a*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *at-right-to-0*: *at-right a* = *filtermap* $(\lambda x.\ x + a)$ (*at-right 0*)
 **for** *a* :: *real*
 ⟨*proof*⟩

**lemma** *filterlim-at-right-to-0*:
*filterlim f F* (*at-right a*) $\longleftrightarrow$ *filterlim* ($\lambda x.$ *f* ($x + a$)) *F* (*at-right 0*)
**for** *a* :: *real*
⟨*proof*⟩

**lemma** *eventually-at-right-to-0*:
*eventually P* (*at-right a*) $\longleftrightarrow$ *eventually* ($\lambda x.$ *P* ($x + a$)) (*at-right 0*)
**for** *a* :: *real*
⟨*proof*⟩

**lemma** *at-to-0*: *at a* = *filtermap* ($\lambda x.$ $x + a$) (*at 0*)
**for** *a* :: *'a::real-normed-vector*
⟨*proof*⟩

**lemma** *filterlim-at-to-0*:
*filterlim f F* (*at a*) $\longleftrightarrow$ *filterlim* ($\lambda x.$ *f* ($x + a$)) *F* (*at 0*)
**for** *a* :: *'a::real-normed-vector*
⟨*proof*⟩

**lemma** *eventually-at-to-0*:
*eventually P* (*at a*) $\longleftrightarrow$ *eventually* ($\lambda x.$ *P* ($x + a$)) (*at 0*)
**for** *a* :: *'a::real-normed-vector*
⟨*proof*⟩

**lemma** *filtermap-at-minus*: *filtermap* ($\lambda x.$ $-$ *x*) (*at a*) = *at* ($-$ *a*)
**for** *a* :: *'a::real-normed-vector*
⟨*proof*⟩

**lemma** *at-left-minus*: *at-left a* = *filtermap* ($\lambda x.$ $-$ *x*) (*at-right* ($-$ *a*))
**for** *a* :: *real*
⟨*proof*⟩

**lemma** *at-right-minus*: *at-right a* = *filtermap* ($\lambda x.$ $-$ *x*) (*at-left* ($-$ *a*))
**for** *a* :: *real*
⟨*proof*⟩

**lemma** *filtermap-linear-at-within*:
**assumes** *bij f* **and** *cont*: *isCont f a* **and** *open-map*: $\bigwedge S.$ *open S* $\Longrightarrow$ *open* (*f*‘*S*)
**shows** *filtermap f* (*at a within S*) = *at* (*f a*) *within f*‘*S*
⟨*proof*⟩

**lemma** *filterlim-at-left-to-right*:
*filterlim f F* (*at-left a*) $\longleftrightarrow$ *filterlim* ($\lambda x.$ *f* ($-$ *x*)) *F* (*at-right* ($-a$))
**for** *a* :: *real*
⟨*proof*⟩

**lemma** *eventually-at-left-to-right*:
*eventually P* (*at-left a*) $\longleftrightarrow$ *eventually* ($\lambda x.$ *P* ($-$ *x*)) (*at-right* ($-a$))
**for** *a* :: *real*

⟨*proof*⟩

**lemma** *filterlim-uminus-at-top-at-bot*: *LIM x at-bot.* $-$ *x* :: *real* :> *at-top*
⟨*proof*⟩

**lemma** *filterlim-uminus-at-bot-at-top*: *LIM x at-top.* $-$ *x* :: *real* :> *at-bot*
⟨*proof*⟩

**lemma** *at-bot-mirror* :
**shows** (*at-bot*::(*'a*::{*ordered-ab-group-add*,*linorder*} *filter*)) = *filtermap uminus at-top*
⟨*proof*⟩

**lemma** *at-top-mirror* :
**shows** (*at-top*::(*'a*::{*ordered-ab-group-add*,*linorder*} *filter*)) = *filtermap uminus at-bot*
⟨*proof*⟩

**lemma** *filterlim-at-top-mirror*: (*LIM x at-top. f x* :> *F*) $\longleftrightarrow$ (*LIM x at-bot. f* ($-x$::*real*) :> *F*)
⟨*proof*⟩

**lemma** *filterlim-at-bot-mirror*: (*LIM x at-bot. f x* :> *F*) $\longleftrightarrow$ (*LIM x at-top. f* ($-x$::*real*) :> *F*)
⟨*proof*⟩

**lemma** *filterlim-uminus-at-top*: (*LIM x F. f x* :> *at-top*) $\longleftrightarrow$ (*LIM x F.* $-$ (*f x*) :: *real* :> *at-bot*)
⟨*proof*⟩

**lemma** *tendsto-at-botI-sequentially*:
**fixes** *f* :: *real* $\Rightarrow$ *'b*::*first-countable-topology*
**assumes** $*$: $\bigwedge X$. *filterlim X at-bot sequentially* $\Longrightarrow$ ($\lambda n$. *f* (*X n*)) $\longrightarrow$ *y*
**shows** (*f* $\longrightarrow$ *y*) *at-bot*
⟨*proof*⟩

**lemma** *filterlim-at-infinity-imp-filterlim-at-top*:
**assumes** *filterlim* (*f* :: *'a* $\Rightarrow$ *real*) *at-infinity F*
**assumes** *eventually* ($\lambda x$. *f x* > *0*) *F*
**shows** *filterlim f at-top F*
⟨*proof*⟩

**lemma** *filterlim-at-infinity-imp-filterlim-at-bot*:
**assumes** *filterlim* (*f* :: *'a* $\Rightarrow$ *real*) *at-infinity F*
**assumes** *eventually* ($\lambda x$. *f x* < *0*) *F*
**shows** *filterlim f at-bot F*
⟨*proof*⟩

**lemma** *filterlim-uminus-at-bot*: (*LIM x F. f x* :> *at-bot*) $\longleftrightarrow$ (*LIM x F.* $-$ (*f x*) ::

*real* :> *at-top*)
⟨*proof*⟩

**lemma** *filterlim-inverse-at-top-right*: *LIM x at-right* (*0*::*real*). *inverse x* :> *at-top*
⟨*proof*⟩

**lemma** *tendsto-inverse-0*:
**fixes** *x* :: - ⇒ ′*a*::*real-normed-div-algebra*
**shows** (*inverse* ⟶ (*0*::′*a*)) *at-infinity*
⟨*proof*⟩

**lemma** *tendsto-add-filterlim-at-infinity*:
**fixes** *c* :: ′*b*::*real-normed-vector*
**and** *F* :: ′*a filter*
**assumes** (*f* ⟶ *c*) *F*
**and** *filterlim g at-infinity F*
**shows** *filterlim* (λ*x*. *f x* + *g x*) *at-infinity F*
⟨*proof*⟩

**lemma** *tendsto-add-filterlim-at-infinity*′:
**fixes** *c* :: ′*b*::*real-normed-vector*
**and** *F* :: ′*a filter*
**assumes** *filterlim f at-infinity F*
**and** (*g* ⟶ *c*) *F*
**shows** *filterlim* (λ*x*. *f x* + *g x*) *at-infinity F*
⟨*proof*⟩

**lemma** *filterlim-inverse-at-right-top*: *LIM x at-top*. *inverse x* :> *at-right* (*0*::*real*)
⟨*proof*⟩

**lemma** *filterlim-inverse-at-top*:
(*f* ⟶ (*0* :: *real*)) *F* ⟹ *eventually* (λ*x*. *0* < *f x*) *F* ⟹ *LIM x F*. *inverse* (*f x*) :> *at-top*
⟨*proof*⟩

**lemma** *filterlim-inverse-at-bot-neg*:
*LIM x* (*at-left* (*0*::*real*)). *inverse x* :> *at-bot*
⟨*proof*⟩

**lemma** *filterlim-inverse-at-bot*:
(*f* ⟶ (*0* :: *real*)) *F* ⟹ *eventually* (λ*x*. *f x* < *0*) *F* ⟹ *LIM x F*. *inverse* (*f x*) :> *at-bot*
⟨*proof*⟩

**lemma** *at-right-to-top*: (*at-right* (*0*::*real*)) = *filtermap inverse at-top*
⟨*proof*⟩

**lemma** *eventually-at-right-to-top*:
*eventually P* (*at-right* (*0*::*real*)) ⟷ *eventually* (λ*x*. *P* (*inverse x*)) *at-top*

⟨*proof*⟩

**lemma** *filterlim-at-right-to-top*:
*filterlim f F* (*at-right* (*0*::*real*)) $\longleftrightarrow$ (*LIM x at-top*. *f* (*inverse x*) :> *F*)
⟨*proof*⟩

**lemma** *at-top-to-right*: *at-top* = *filtermap inverse* (*at-right* (*0*::*real*))
⟨*proof*⟩

**lemma** *eventually-at-top-to-right*:
*eventually P at-top* $\longleftrightarrow$ *eventually* ($\lambda x$. *P* (*inverse x*)) (*at-right* (*0*::*real*))
⟨*proof*⟩

**lemma** *filterlim-at-top-to-right*:
*filterlim f F at-top* $\longleftrightarrow$ (*LIM x* (*at-right* (*0*::*real*)). *f* (*inverse x*) :> *F*)
⟨*proof*⟩

**lemma** *filterlim-inverse-at-infinity*:
**fixes** *x* :: *-* $\Rightarrow$ *'a*::{*real-normed-div-algebra*, *division-ring*}
**shows** *filterlim inverse at-infinity* (*at* (*0*::*'a*))
⟨*proof*⟩

**lemma** *filterlim-inverse-at-iff*:
**fixes** *g* :: *'a* $\Rightarrow$ *'b*::{*real-normed-div-algebra*, *division-ring*}
**shows** (*LIM x F*. *inverse* (*g x*) :> *at 0*) $\longleftrightarrow$ (*LIM x F*. *g x* :> *at-infinity*)
⟨*proof*⟩

**lemma** *tendsto-mult-filterlim-at-infinity*:
**fixes** *c* :: *'a*::*real-normed-field*
**assumes** (*f* $\longrightarrow$ *c*) *F c* $\neq$ *0*
**assumes** *filterlim g at-infinity F*
**shows** *filterlim* ($\lambda x$. *f x* $*$ *g x*) *at-infinity F*
⟨*proof*⟩

**lemma** *filterlim-power-int-neg-at-infinity*:
**fixes** *f* :: *-* $\Rightarrow$ *'a*::{*real-normed-div-algebra*, *division-ring*}
**assumes** *n* < *0* **and** *lim*: (*f* $\longrightarrow$ *0*) *F* **and** *ev*: *eventually* ($\lambda x$. *f x* $\neq$ *0*) *F*
**shows** *filterlim* ($\lambda x$. *f x powi n*) *at-infinity F*
⟨*proof*⟩

**lemma** *tendsto-inverse-0-at-top*: *LIM x F*. *f x* :> *at-top* $\Longrightarrow$ (($\lambda x$. *inverse* (*f x*) :: *real*) $\longrightarrow$ *0*) *F*
⟨*proof*⟩

**lemma** *filterlim-inverse-at-top-iff*:
*eventually* ($\lambda x$. *0* < *f x*) *F* $\Longrightarrow$ (*LIM x F*. *inverse* (*f x*) :> *at-top*) $\longleftrightarrow$ (*f* $\longrightarrow$ (*0* :: *real*)) *F*
⟨*proof*⟩

**lemma** *filterlim-at-top-iff-inverse-0*:
  $eventually\ (\lambda x.\ 0 < f\ x)\ F \Longrightarrow (LIM\ x\ F.\ f\ x :> at\text{-}top) \longleftrightarrow ((inverse \circ f) \longrightarrow (0 :: real))\ F$
  ⟨*proof*⟩

**lemma** *real-tendsto-divide-at-top*:
  **fixes** $c::real$
  **assumes** $(f \longrightarrow c)\ F$
  **assumes** $filterlim\ g\ at\text{-}top\ F$
  **shows** $((\lambda x.\ f\ x\ /\ g\ x) \longrightarrow 0)\ F$
  ⟨*proof*⟩

**lemma** *mult-nat-left-at-top*: $c > 0 \Longrightarrow filterlim\ (\lambda x.\ c * x)\ at\text{-}top\ sequentially$
  **for** $c :: nat$
  ⟨*proof*⟩

**lemma** *mult-nat-right-at-top*: $c > 0 \Longrightarrow filterlim\ (\lambda x.\ x * c)\ at\text{-}top\ sequentially$
  **for** $c :: nat$
  ⟨*proof*⟩

**lemma** *filterlim-times-pos*:
  $LIM\ x\ F1.\ c * f\ x :> at\text{-}right\ l$
  **if** $filterlim\ f\ (at\text{-}right\ p)\ F1\ 0 < c\ l = c * p$
  **for** $c::'a::\{linordered\text{-}field,\ linorder\text{-}topology\}$
  ⟨*proof*⟩

**lemma** *filtermap-nhds-times*: $c \neq 0 \Longrightarrow filtermap\ (times\ c)\ (nhds\ a) = nhds\ (c * a)$
  **for** $a\ c :: 'a::real\text{-}normed\text{-}field$
  ⟨*proof*⟩

**lemma** *filtermap-times-pos-at-right*:
  **fixes** $c::'a::\{linordered\text{-}field,\ linorder\text{-}topology\}$
  **assumes** $c > 0$
  **shows** $filtermap\ (times\ c)\ (at\text{-}right\ p) = at\text{-}right\ (c * p)$
  ⟨*proof*⟩

**lemma** *at-to-infinity*: $(at\ (0::'a::\{real\text{-}normed\text{-}field,field\})) = filtermap\ inverse\ at\text{-}infinity$
⟨*proof*⟩

**lemma** *lim-at-infinity-0*:
  **fixes** $l :: 'a::\{real\text{-}normed\text{-}field,field\}$
  **shows** $(f \longrightarrow l)\ at\text{-}infinity \longleftrightarrow ((f \circ inverse) \longrightarrow l)\ (at\ (0::'a))$
  ⟨*proof*⟩

**lemma** *lim-zero-infinity*:
  **fixes** $l :: 'a::\{real\text{-}normed\text{-}field,field\}$
  **shows** $((\lambda x.\ f(1\ /\ x)) \longrightarrow l)\ (at\ (0::'a)) \Longrightarrow (f \longrightarrow l)\ at\text{-}infinity$
  ⟨*proof*⟩

We only show rules for multiplication and addition when the functions are either against a real value or against infinity. Further rules are easy to derive by using *filterlim ?f at-top ?F* = (*LIM x ?F*. − *?f x* :> *at-bot*).

**lemma** *filterlim-tendsto-pos-mult-at-top*:
  **assumes** *f*: (*f* ⟶ *c*) *F*
    **and** *c*: *0* < *c*
    **and** *g*: *LIM x F*. *g x* :> *at-top*
  **shows** *LIM x F*. (*f x* ∗ *g x* :: *real*) :> *at-top*
  ⟨*proof*⟩

**lemma** *filterlim-at-top-mult-at-top*:
  **assumes** *f*: *LIM x F*. *f x* :> *at-top*
    **and** *g*: *LIM x F*. *g x* :> *at-top*
  **shows** *LIM x F*. (*f x* ∗ *g x* :: *real*) :> *at-top*
  ⟨*proof*⟩

**lemma** *filterlim-at-top-mult-tendsto-pos*:
  **assumes** *f*: (*f* ⟶ *c*) *F*
    **and** *c*: *0* < *c*
    **and** *g*: *LIM x F*. *g x* :> *at-top*
  **shows** *LIM x F*. (*g x* ∗ *f x*:: *real*) :> *at-top*
  ⟨*proof*⟩

**lemma** *filterlim-tendsto-pos-mult-at-bot*:
  **fixes** *c* :: *real*
  **assumes** (*f* ⟶ *c*) *F* *0* < *c* *filterlim g at-bot F*
  **shows** *LIM x F*. *f x* ∗ *g x* :> *at-bot*
  ⟨*proof*⟩

**lemma** *filterlim-tendsto-neg-mult-at-bot*:
  **fixes** *c* :: *real*
  **assumes** *c*: (*f* ⟶ *c*) *F* *c* < *0* **and** *g*: *filterlim g at-top F*
  **shows** *LIM x F*. *f x* ∗ *g x* :> *at-bot*
  ⟨*proof*⟩

**lemma** *filterlim-cmult-at-bot-at-top*:
  **assumes** *filterlim* (*h* :: - ⇒ *real*) *at-top F c* ≠ *0* *G* = (*if c* > *0 then at-top else at-bot*)
  **shows** *filterlim* (λ*x*. *c* ∗ *h x*) *G F*
  ⟨*proof*⟩

**lemma** *filterlim-pow-at-top*:
  **fixes** *f* :: *'a* ⇒ *real*
  **assumes** *0* < *n*
    **and** *f*: *LIM x F*. *f x* :> *at-top*
  **shows** *LIM x F*. (*f x*)^*n* :: *real* :> *at-top*
  ⟨*proof*⟩

**lemma** *filterlim-pow-at-bot-even*:
**fixes** $f :: real \Rightarrow real$
**shows** $0 < n \Longrightarrow LIM\ x\ F.\ f\ x :> at\text{-}bot \Longrightarrow even\ n \Longrightarrow LIM\ x\ F.\ (f\ x)\hat{\ }n :> at\text{-}top$
⟨*proof*⟩

**lemma** *filterlim-pow-at-bot-odd*:
**fixes** $f :: real \Rightarrow real$
**shows** $0 < n \Longrightarrow LIM\ x\ F.\ f\ x :> at\text{-}bot \Longrightarrow odd\ n \Longrightarrow LIM\ x\ F.\ (f\ x)\hat{\ }n :> at\text{-}bot$
⟨*proof*⟩

**lemma** *filterlim-power-at-infinity* [*tendsto-intros*]:
**fixes** $F$ **and** $f :: 'a \Rightarrow 'b :: real\text{-}normed\text{-}div\text{-}algebra$
**assumes** *filterlim f at-infinity F* $n > 0$
**shows** *filterlim* $(\lambda x.\ f\ x\ \hat{}\ n)$ *at-infinity F*
⟨*proof*⟩

**lemma** *filterlim-tendsto-add-at-top*:
**assumes** $f$: $(f \longrightarrow c)\ F$
**and** $g$: $LIM\ x\ F.\ g\ x :> at\text{-}top$
**shows** $LIM\ x\ F.\ (f\ x + g\ x :: real) :> at\text{-}top$
⟨*proof*⟩

**lemma** *filterlim-tendsto-add-at-top-iff*:
**assumes** $f$: $(f \longrightarrow c)\ F$
**shows** $(LIM\ x\ F.\ (f\ x + g\ x :: real) :> at\text{-}top) \longleftrightarrow (LIM\ x\ F.\ g\ x :> at\text{-}top)$
⟨*proof*⟩

**lemma** *filterlim-tendsto-add-at-bot-iff*:
**fixes** $c::real$
**assumes** $f$: $(f \longrightarrow c)\ F$
**shows** $(LIM\ x\ F.\ f\ x + g\ x :> at\text{-}bot) \longleftrightarrow (LIM\ x\ F.\ g\ x :> at\text{-}bot)$
⟨*proof*⟩

**lemma** *LIM-at-top-divide*:
**fixes** $f\ g :: 'a \Rightarrow real$
**assumes** $f$: $(f \longrightarrow a)\ F\ 0 < a$
**and** $g$: $(g \longrightarrow 0)\ F$ *eventually* $(\lambda x.\ 0 < g\ x)\ F$
**shows** $LIM\ x\ F.\ f\ x\ /\ g\ x :> at\text{-}top$
⟨*proof*⟩

**lemma** *filterlim-at-top-add-at-top*:
**assumes** $f$: $LIM\ x\ F.\ f\ x :> at\text{-}top$
**and** $g$: $LIM\ x\ F.\ g\ x :> at\text{-}top$
**shows** $LIM\ x\ F.\ (f\ x + g\ x :: real) :> at\text{-}top$
⟨*proof*⟩

**lemma** *tendsto-divide-0*:

**fixes** $f$ :: - $\Rightarrow$ $'a$::{*real-normed-div-algebra*, *division-ring*}
**assumes** $f$: ($f \longrightarrow c$) $F$
**and** $g$: *LIM* $x$ $F$. $g$ $x$ :> *at-infinity*
**shows** (($\lambda x$. $f$ $x$ / $g$ $x$) $\longrightarrow$ *0*) $F$
⟨*proof*⟩

**lemma** *linear-plus-1-le-power*:
**fixes** $x$ :: *real*
**assumes** $x$: $0 \leq x$
**shows** *real* $n * x + 1 \leq (x + 1)$ ^ $n$
⟨*proof*⟩

**lemma** *filterlim-realpow-sequentially-gt1*:
**fixes** $x$ :: $'a$ :: *real-normed-div-algebra*
**assumes** $x$[*arith*]: $1 <$ *norm* $x$
**shows** *LIM* $n$ *sequentially*. $x$ ^ $n$ :> *at-infinity*
⟨*proof*⟩

**lemma** *filterlim-divide-at-infinity*:
**fixes** $f$ $g$ :: $'a \Rightarrow 'a$ :: *real-normed-field*
**assumes** *filterlim* $f$ (*nhds* $c$) $F$ *filterlim* $g$ (*at* *0*) $F$ $c \neq 0$
**shows** *filterlim* ($\lambda x$. $f$ $x$ / $g$ $x$) *at-infinity* $F$
⟨*proof*⟩

## 108.5 Floor and Ceiling

**lemma** *eventually-floor-less*:
**fixes** $f$ :: $'a \Rightarrow 'b$::{*order-topology*,*floor-ceiling*}
**assumes** $f$: ($f \longrightarrow l$) $F$
**and** $l$: $l \notin \mathbb{Z}$
**shows** $\forall_F$ $x$ *in* $F$. *of-int* (*floor* $l$) $<$ $f$ $x$
⟨*proof*⟩

**lemma** *eventually-less-ceiling*:
**fixes** $f$ :: $'a \Rightarrow 'b$::{*order-topology*,*floor-ceiling*}
**assumes** $f$: ($f \longrightarrow l$) $F$
**and** $l$: $l \notin \mathbb{Z}$
**shows** $\forall_F$ $x$ *in* $F$. $f$ $x$ $<$ *of-int* (*ceiling* $l$)
⟨*proof*⟩

**lemma** *eventually-floor-eq*:
**fixes** $f$::$'a \Rightarrow 'b$::{*order-topology*,*floor-ceiling*}
**assumes** $f$: ($f \longrightarrow l$) $F$
**and** $l$: $l \notin \mathbb{Z}$
**shows** $\forall_F$ $x$ *in* $F$. *floor* ($f$ $x$) = *floor* $l$
⟨*proof*⟩

**lemma** *eventually-ceiling-eq*:

**fixes** $f::'a \Rightarrow 'b::\{order\text{-}topology,floor\text{-}ceiling\}$
**assumes** $f$: $(f \longrightarrow l)\ F$
**and** $l$: $l \notin \mathbb{Z}$
**shows** $\forall_F\ x\ in\ F.\ ceiling\ (f\ x) = ceiling\ l$
⟨proof⟩

**lemma** *tendsto-of-int-floor*:
**fixes** $f::'a \Rightarrow 'b::\{order\text{-}topology,floor\text{-}ceiling\}$
**assumes** $(f \longrightarrow l)\ F$
**and** $l \notin \mathbb{Z}$
**shows** $((\lambda x.\ of\text{-}int\ (floor\ (f\ x)) :: 'c::\{ring\text{-}1,topological\text{-}space\}) \longrightarrow of\text{-}int\ (floor\ l))\ F$
⟨proof⟩

**lemma** *tendsto-of-int-ceiling*:
**fixes** $f::'a \Rightarrow 'b::\{order\text{-}topology,floor\text{-}ceiling\}$
**assumes** $(f \longrightarrow l)\ F$
**and** $l \notin \mathbb{Z}$
**shows** $((\lambda x.\ of\text{-}int\ (ceiling\ (f\ x)):: 'c::\{ring\text{-}1,topological\text{-}space\}) \longrightarrow of\text{-}int\ (ceiling\ l))\ F$
⟨proof⟩

**lemma** *continuous-on-of-int-floor*:
$continuous\text{-}on\ (UNIV - \mathbb{Z}::'a::\{order\text{-}topology,\ floor\text{-}ceiling\}\ set)$
$(\lambda x.\ of\text{-}int\ (floor\ x)::'b::\{ring\text{-}1,\ topological\text{-}space\})$
⟨proof⟩

**lemma** *continuous-on-of-int-ceiling*:
$continuous\text{-}on\ (UNIV - \mathbb{Z}::'a::\{order\text{-}topology,\ floor\text{-}ceiling\}\ set)$
$(\lambda x.\ of\text{-}int\ (ceiling\ x)::'b::\{ring\text{-}1,\ topological\text{-}space\})$
⟨proof⟩

## 108.6 Limits of Sequences

**lemma** [*trans*]: $X = Y \Longrightarrow Y \longrightarrow z \Longrightarrow X \longrightarrow z$
⟨proof⟩

**lemma** *LIMSEQ-iff*:
**fixes** $L :: 'a::real\text{-}normed\text{-}vector$
**shows** $(X \longrightarrow L) = (\forall r>0.\ \exists no.\ \forall n \geq no.\ norm\ (X\ n - L) < r)$
⟨proof⟩

**lemma** *LIMSEQ-I*: $(\bigwedge r.\ 0 < r \Longrightarrow \exists no.\ \forall n\geq no.\ norm\ (X\ n - L) < r) \Longrightarrow X \longrightarrow L$
**for** $L :: 'a::real\text{-}normed\text{-}vector$
⟨proof⟩

**lemma** *LIMSEQ-D*: $X \longrightarrow L \Longrightarrow 0 < r \Longrightarrow \exists no.\ \forall n\geq no.\ norm\ (X\ n - L) < r$

**for** $L$ :: $'a$::*real-normed-vector*
⟨*proof*⟩

**lemma** *LIMSEQ-linear*: $X \longrightarrow x \Longrightarrow l > 0 \Longrightarrow (\lambda\ n.\ X\ (n * l)) \longrightarrow x$
⟨*proof*⟩

Transformation of limit.

**lemma** *Lim-transform*: $(g \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ f\ x - g\ x) \longrightarrow 0)\ F \Longrightarrow (f \longrightarrow a)\ F$
**for** $a\ b$ :: $'a$::*real-normed-vector*
⟨*proof*⟩

**lemma** *Lim-transform2*: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ f\ x - g\ x) \longrightarrow 0)\ F \Longrightarrow (g \longrightarrow a)\ F$
**for** $a\ b$ :: $'a$::*real-normed-vector*
⟨*proof*⟩

**proposition** *Lim-transform-eq*: $((\lambda x.\ f\ x - g\ x) \longrightarrow 0)\ F \Longrightarrow (f \longrightarrow a)\ F \longleftrightarrow (g \longrightarrow a)\ F$
**for** $a$ :: $'a$::*real-normed-vector*
⟨*proof*⟩

**lemma** *Lim-transform-eventually*:
$[\![(f \longrightarrow l)\ F;\ eventually\ (\lambda x.\ f\ x = g\ x)\ F]\!] \Longrightarrow (g \longrightarrow l)\ F$
⟨*proof*⟩

**lemma** *Lim-transform-within*:
**assumes** $(f \longrightarrow l)\ (at\ x\ within\ S)$
**and** $0 < d$
**and** $\bigwedge x'.\ x' \in S \Longrightarrow 0 < dist\ x'\ x \Longrightarrow dist\ x'\ x < d \Longrightarrow f\ x' = g\ x'$
**shows** $(g \longrightarrow l)\ (at\ x\ within\ S)$
⟨*proof*⟩

**lemma** *filterlim-transform-within*:
**assumes** *filterlim* $g\ G\ (at\ x\ within\ S)$
**assumes** $G \le F\ 0{<}d\ (\bigwedge x'.\ x' \in S \Longrightarrow 0 < dist\ x'\ x \Longrightarrow dist\ x'\ x < d \Longrightarrow f\ x' = g\ x')$
**shows** *filterlim* $f\ F\ (at\ x\ within\ S)$
⟨*proof*⟩

Common case assuming being away from some crucial point like 0.

**lemma** *Lim-transform-away-within*:
**fixes** $a\ b$ :: $'a$::*t1-space*
**assumes** $a \ne b$
**and** $\forall x{\in}S.\ x \ne a \wedge x \ne b \longrightarrow f\ x = g\ x$
**and** $(f \longrightarrow l)\ (at\ a\ within\ S)$
**shows** $(g \longrightarrow l)\ (at\ a\ within\ S)$
⟨*proof*⟩

**lemma** *Lim-transform-away-at*:
**fixes** $a\ b :: 'a::t1\text{-}space$
**assumes** *ab*: $a \neq b$
**and** *fg*: $\forall x.\ x \neq a \wedge x \neq b \longrightarrow f\ x = g\ x$
**and** *fl*: $(f \longrightarrow l)\ (at\ a)$
**shows** $(g \longrightarrow l)\ (at\ a)$
⟨*proof*⟩

Alternatively, within an open set.

**lemma** *Lim-transform-within-open*:
**assumes** $(f \longrightarrow l)\ (at\ a\ within\ T)$
**and** *open s* **and** $a \in s$
**and** $\bigwedge x.\ x \in s \Longrightarrow x \neq a \Longrightarrow f\ x = g\ x$
**shows** $(g \longrightarrow l)\ (at\ a\ within\ T)$
⟨*proof*⟩

A congruence rule allowing us to transform limits assuming not at point.

**lemma** *Lim-cong-within*:
**assumes** $a = b$
**and** $x = y$
**and** $S = T$
**and** $\bigwedge x.\ x \neq b \Longrightarrow x \in T \Longrightarrow f\ x = g\ x$
**shows** $(f \longrightarrow x)\ (at\ a\ within\ S) \longleftrightarrow (g \longrightarrow y)\ (at\ b\ within\ T)$
⟨*proof*⟩

An unbounded sequence's inverse tends to 0.

**lemma** *LIMSEQ-inverse-zero*:
**assumes** $\bigwedge r::real.\ \exists N.\ \forall n \geq N.\ r < X\ n$
**shows** $(\lambda n.\ inverse\ (X\ n)) \longlonglongrightarrow 0$
⟨*proof*⟩

The sequence $1\ /\ n$ tends to 0 as $n$ tends to infinity.

**lemma** *LIMSEQ-inverse-real-of-nat*: $(\lambda n.\ inverse\ (real\ (Suc\ n))) \longlonglongrightarrow 0$
⟨*proof*⟩

The sequence $r + 1\ /\ n$ tends to $r$ as $n$ tends to infinity is now easily proved.

**lemma** *LIMSEQ-inverse-real-of-nat-add*: $(\lambda n.\ r + inverse\ (real\ (Suc\ n))) \longlonglongrightarrow r$
⟨*proof*⟩

**lemma** *LIMSEQ-inverse-real-of-nat-add-minus*: $(\lambda n.\ r + -inverse\ (real\ (Suc\ n))) \longlonglongrightarrow r$
⟨*proof*⟩

**lemma** *LIMSEQ-inverse-real-of-nat-add-minus-mult*: $(\lambda n.\ r * (1 + -\ inverse\ (real\ (Suc\ n)))) \longlonglongrightarrow r$
⟨*proof*⟩

**lemma** *lim-inverse-n*: $((\lambda n.\ inverse(of\text{-}nat\ n)) \longrightarrow (0::'a::real\text{-}normed\text{-}field))$ *sequentially*
⟨*proof*⟩

**lemma** *lim-inverse-n'*: $((\lambda n.\ 1\ /\ n) \longrightarrow 0)$ *sequentially*
⟨*proof*⟩

**lemma** *LIMSEQ-Suc-n-over-n*: $(\lambda n.\ of\text{-}nat\ (Suc\ n)\ /\ of\text{-}nat\ n :: 'a :: real\text{-}normed\text{-}field) \longrightarrow 1$
⟨*proof*⟩

**lemma** *LIMSEQ-n-over-Suc-n*: $(\lambda n.\ of\text{-}nat\ n\ /\ of\text{-}nat\ (Suc\ n) :: 'a :: real\text{-}normed\text{-}field) \longrightarrow 1$
⟨*proof*⟩

## 108.7 Convergence on sequences

**lemma** *convergent-cong*:
**assumes** *eventually* $(\lambda x.\ f\ x = g\ x)$ *sequentially*
**shows** *convergent f* $\longleftrightarrow$ *convergent g*
⟨*proof*⟩

**lemma** *convergent-Suc-iff*: *convergent* $(\lambda n.\ f\ (Suc\ n)) \longleftrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-ignore-initial-segment*: *convergent* $(\lambda n.\ f\ (n + m)) =$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-add*:
**fixes** $X\ Y :: nat \Rightarrow 'a::topological\text{-}monoid\text{-}add$
**assumes** *convergent* $(\lambda n.\ X\ n)$
**and** *convergent* $(\lambda n.\ Y\ n)$
**shows** *convergent* $(\lambda n.\ X\ n + Y\ n)$
⟨*proof*⟩

**lemma** *convergent-sum*:
**fixes** $X :: 'a \Rightarrow nat \Rightarrow 'b::topological\text{-}comm\text{-}monoid\text{-}add$
**shows** $(\bigwedge i.\ i \in A \Longrightarrow convergent\ (\lambda n.\ X\ i\ n)) \Longrightarrow convergent\ (\lambda n.\ \sum i{\in}A.\ X\ i\ n)$
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *convergent*:
**assumes** *convergent* $(\lambda n.\ X\ n)$
**shows** *convergent* $(\lambda n.\ f\ (X\ n))$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *convergent*:
**assumes** *convergent* $(\lambda n.\ X\ n)$

**and** *convergent* $(\lambda n.\ Y\ n)$
**shows** *convergent* $(\lambda n.\ X\ n\ {*}{*}\ Y\ n)$
⟨*proof*⟩

**lemma** *convergent-minus-iff*:
**fixes** $X$ :: *nat* ⇒ *'a::topological-group-add*
**shows** *convergent* $X \longleftrightarrow$ *convergent* $(\lambda n.\ -\ X\ n)$
⟨*proof*⟩

**lemma** *convergent-diff*:
**fixes** $X\ Y$ :: *nat* ⇒ *'a::topological-group-add*
**assumes** *convergent* $(\lambda n.\ X\ n)$
**assumes** *convergent* $(\lambda n.\ Y\ n)$
**shows** *convergent* $(\lambda n.\ X\ n\ -\ Y\ n)$
⟨*proof*⟩

**lemma** *convergent-norm*:
**assumes** *convergent f*
**shows** *convergent* $(\lambda n.$ *norm* $(f\ n))$
⟨*proof*⟩

**lemma** *convergent-of-real*:
*convergent f* $\Longrightarrow$ *convergent* $(\lambda n.$ *of-real* $(f\ n)$ :: *'a::real-normed-algebra-1*$)$
⟨*proof*⟩

**lemma** *convergent-add-const-iff*:
*convergent* $(\lambda n.\ c\ +\ f\ n$ :: *'a::topological-ab-group-add*$)$ $\longleftrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-add-const-right-iff*:
*convergent* $(\lambda n.\ f\ n\ +\ c$ :: *'a::topological-ab-group-add*$)$ $\longleftrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-diff-const-right-iff*:
*convergent* $(\lambda n.\ f\ n\ -\ c$ :: *'a::topological-ab-group-add*$)$ $\longleftrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-mult*:
**fixes** $X\ Y$ :: *nat* ⇒ *'a::topological-semigroup-mult*
**assumes** *convergent* $(\lambda n.\ X\ n)$
**and** *convergent* $(\lambda n.\ Y\ n)$
**shows** *convergent* $(\lambda n.\ X\ n\ *\ Y\ n)$
⟨*proof*⟩

**lemma** *convergent-mult-const-iff*:
**assumes** $c \neq 0$
**shows** *convergent* $(\lambda n.\ c\ *\ f\ n$ :: *'a::{field,topological-semigroup-mult}*$)$ $\longleftrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *convergent-mult-const-right-iff*:
  **fixes** $c$ :: $'a$::{*field*,*topological-semigroup-mult*}
  **assumes** $c \neq 0$
  **shows** *convergent* $(\lambda n.\ f\ n * c) \longleftrightarrow$ *convergent* $f$
  ⟨*proof*⟩

**lemma** *convergent-imp-Bseq*: *convergent* $f \Longrightarrow$ *Bseq* $f$
  ⟨*proof*⟩

A monotone sequence converges to its least upper bound.

**lemma** *LIMSEQ-incseq-SUP*:
  **fixes** $X$ :: *nat* $\Rightarrow$ $'a$::{*conditionally-complete-linorder*,*linorder-topology*}
  **assumes** $u$: *bdd-above* (*range* $X$)
    **and** $X$: *incseq* $X$
  **shows** $X \longrightarrow (SUP\ i.\ X\ i)$
  ⟨*proof*⟩

**lemma** *LIMSEQ-decseq-INF*:
  **fixes** $X$ :: *nat* $\Rightarrow$ $'a$::{*conditionally-complete-linorder*, *linorder-topology*}
  **assumes** $u$: *bdd-below* (*range* $X$)
    **and** $X$: *decseq* $X$
  **shows** $X \longrightarrow (INF\ i.\ X\ i)$
  ⟨*proof*⟩

Main monotonicity theorem.

**lemma** *Bseq-monoseq-convergent*: *Bseq* $X \Longrightarrow$ *monoseq* $X \Longrightarrow$ *convergent* $X$
  **for** $X$ :: *nat* $\Rightarrow$ *real*
  ⟨*proof*⟩

**lemma** *Bseq-mono-convergent*: *Bseq* $X \Longrightarrow (\forall\, m\ n.\ m \leq n \longrightarrow X\ m \leq X\ n) \Longrightarrow$ *convergent* $X$
  **for** $X$ :: *nat* $\Rightarrow$ *real*
  ⟨*proof*⟩

**lemma** *monoseq-imp-convergent-iff-Bseq*: *monoseq* $f \Longrightarrow$ *convergent* $f \longleftrightarrow$ *Bseq* $f$
  **for** $f$ :: *nat* $\Rightarrow$ *real*
  ⟨*proof*⟩

**lemma** *Bseq-monoseq-convergent'-inc*:
  **fixes** $f$ :: *nat* $\Rightarrow$ *real*
  **shows** *Bseq* $(\lambda n.\ f\ (n + M)) \Longrightarrow (\bigwedge m\ n.\ M \leq m \Longrightarrow m \leq n \Longrightarrow f\ m \leq f\ n) \Longrightarrow$ *convergent* $f$
  ⟨*proof*⟩

**lemma** *Bseq-monoseq-convergent'-dec*:
  **fixes** $f$ :: *nat* $\Rightarrow$ *real*
  **shows** *Bseq* $(\lambda n.\ f\ (n + M)) \Longrightarrow (\bigwedge m\ n.\ M \leq m \Longrightarrow m \leq n \Longrightarrow f\ m \geq f\ n) \Longrightarrow$ *convergent* $f$

⟨*proof*⟩

**lemma** *Cauchy-iff*: $Cauchy\ X \longleftrightarrow (\forall\, e>0.\ \exists\, M.\ \forall\, m\geq M.\ \forall\, n\geq M.\ norm\ (X\ m - X\ n) < e)$
**for** $X :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *CauchyI*: $(\bigwedge e.\ 0 < e \Longrightarrow \exists\, M.\ \forall\, m\geq M.\ \forall\, n\geq M.\ norm\ (X\ m - X\ n) < e) \Longrightarrow Cauchy\ X$
**for** $X :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *CauchyD*: $Cauchy\ X \Longrightarrow 0 < e \Longrightarrow \exists\, M.\ \forall\, m\geq M.\ \forall\, n\geq M.\ norm\ (X\ m - X\ n) < e$
**for** $X :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *incseq-convergent*:
**fixes** $X :: nat \Rightarrow real$
**assumes** *incseq X*
**and** $\forall\, i.\ X\ i \leq B$
**obtains** $L$ **where** $X \longrightarrow L$ $\forall\, i.\ X\ i \leq L$
⟨*proof*⟩

**lemma** *decseq-convergent*:
**fixes** $X :: nat \Rightarrow real$
**assumes** *decseq X*
**and** $\forall\, i.\ B \leq X\ i$
**obtains** $L$ **where** $X \longrightarrow L$ $\forall\, i.\ L \leq X\ i$
⟨*proof*⟩

**lemma** *monoseq-convergent*:
**fixes** $X :: nat \Rightarrow real$
**assumes** $X$: *monoseq X* **and** $B$: $\bigwedge i.\ |X\ i| \leq B$
**obtains** $L$ **where** $X \longrightarrow L$
⟨*proof*⟩

## 108.8 More about *filterlim* (thanks to Wenda Li)

**lemma** *filterlim-at-infinity-times*:
**fixes** $f :: 'a \Rightarrow 'b::real\text{-}normed\text{-}field$
**assumes** *filterlim f at-infinity F filterlim g at-infinity F*
**shows** *filterlim* $(\lambda x.\ f\ x * g\ x)$ *at-infinity F*
⟨*proof*⟩

**lemma** *filterlim-at-top-at-bot*[*elim*]:
**fixes** $f::'a \Rightarrow 'b::unbounded\text{-}dense\text{-}linorder$ **and** $F::'a\ filter$
**assumes** *top*:*filterlim f at-top F* **and** *bot*: *filterlim f at-bot F* **and** $F\neq bot$
**shows** *False*

⟨*proof*⟩

**lemma** *filterlim-at-top-nhds*[*elim*]:
**fixes** $f$::$'a \Rightarrow 'b$::{*unbounded-dense-linorder*,*order-topology*} **and** $F$::$'a$ *filter*
**assumes** *top*:*filterlim f at-top F* **and** *tendsto*: $(f \longrightarrow c)$ $F$ **and** $F \neq bot$
**shows** *False*
⟨*proof*⟩

**lemma** *filterlim-at-bot-nhds*[*elim*]:
**fixes** $f$::$'a \Rightarrow 'b$::{*unbounded-dense-linorder*,*order-topology*} **and** $F$::$'a$ *filter*
**assumes** *top*:*filterlim f at-bot F* **and** *tendsto*: $(f \longrightarrow c)$ $F$ **and** $F \neq bot$
**shows** *False*
⟨*proof*⟩

**lemma** *eventually-times-inverse-1*:
**fixes** $f$::$'a \Rightarrow 'b$::{*field*,*t2-space*}
**assumes** $(f \longrightarrow c)$ $F$ $c \neq 0$
**shows** $\forall_F$ *x in F. inverse* $(f\ x) * f\ x = 1$
⟨*proof*⟩

**lemma** *filterlim-at-infinity-divide-iff*:
**fixes** $f$::$'a \Rightarrow 'b$::*real-normed-field*
**assumes** $(f \longrightarrow c)$ $F$ $c \neq 0$
**shows** (*LIM x F. f x / g x :> at-infinity*) $\longleftrightarrow$ (*LIM x F. g x :> at 0*)
⟨*proof*⟩

**lemma** *filterlim-tendsto-pos-mult-at-top-iff*:
**fixes** $f$::$'a \Rightarrow$ *real*
**assumes** $(f \longrightarrow c)$ $F$ **and** $0 < c$
**shows** (*LIM x F.* $(f\ x * g\ x)$ *:> at-top*) $\longleftrightarrow$ (*LIM x F. g x :> at-top*)
⟨*proof*⟩

**lemma** *filterlim-tendsto-pos-mult-at-bot-iff*:
**fixes** $c$ :: *real*
**assumes** $(f \longrightarrow c)$ $F$ $0 < c$
**shows** (*LIM x F.* $f\ x * g\ x$ *:> at-bot*) $\longleftrightarrow$ *filterlim g at-bot F*
⟨*proof*⟩

**lemma** *filterlim-tendsto-neg-mult-at-top-iff*:
**fixes** $f$::$'a \Rightarrow$ *real*
**assumes** $(f \longrightarrow c)$ $F$ **and** $c < 0$
**shows** (*LIM x F.* $(f\ x * g\ x)$ *:> at-top*) $\longleftrightarrow$ (*LIM x F. g x :> at-bot*)
⟨*proof*⟩

**lemma** *filterlim-tendsto-neg-mult-at-bot-iff*:
**fixes** $c$ :: *real*
**assumes** $(f \longrightarrow c)$ $F$ $0 > c$
**shows** (*LIM x F.* $f\ x * g\ x$ *:> at-bot*) $\longleftrightarrow$ *filterlim g at-top F*
⟨*proof*⟩

## 108.9 Power Sequences

**lemma** *Bseq-realpow*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow Bseq\ (\lambda n.\ x \widehat{\ } n)$
**for** $x :: real$
⟨*proof*⟩

**lemma** *monoseq-realpow*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow monoseq\ (\lambda n.\ x \widehat{\ } n)$
**for** $x :: real$
⟨*proof*⟩

**lemma** *convergent-realpow*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow convergent\ (\lambda n.\ x \widehat{\ } n)$
**for** $x :: real$
⟨*proof*⟩

**lemma** *LIMSEQ-inverse-realpow-zero*: $1 < x \Longrightarrow (\lambda n.\ inverse\ (x \widehat{\ } n)) \longrightarrow 0$
**for** $x :: real$
⟨*proof*⟩

**lemma** *LIMSEQ-realpow-zero*:
**fixes** $x :: real$
**assumes** $0 \leq x\ x < 1$
**shows** $(\lambda n.\ x \widehat{\ } n) \longrightarrow 0$
⟨*proof*⟩

**lemma** *LIMSEQ-power-zero* [*tendsto-intros*]: $norm\ x < 1 \Longrightarrow (\lambda n.\ x \widehat{\ } n) \longrightarrow 0$
**for** $x :: 'a::real\text{-}normed\text{-}algebra\text{-}1$
⟨*proof*⟩

**lemma** *LIMSEQ-divide-realpow-zero*: $1 < x \Longrightarrow (\lambda n.\ a\ /\ (x \widehat{\ } n) :: real) \longrightarrow 0$
⟨*proof*⟩

**lemma**
*tendsto-power-zero*:
**fixes** $x::'a::real\text{-}normed\text{-}algebra\text{-}1$
**assumes** *filterlim f at-top F*
**assumes** $norm\ x < 1$
**shows** $((\lambda y.\ x \widehat{\ } (f\ y)) \longrightarrow 0)\ F$
⟨*proof*⟩

Limit of $c^n$ for $|c| < 1$.

**lemma** *LIMSEQ-abs-realpow-zero*: $|c| < 1 \Longrightarrow (\lambda n.\ |c| \widehat{\ } n :: real) \longrightarrow 0$
⟨*proof*⟩

**lemma** *LIMSEQ-abs-realpow-zero2*: $|c| < 1 \Longrightarrow (\lambda n.\ c \widehat{\ } n :: real) \longrightarrow 0$
⟨*proof*⟩

## 108.10 Limits of Functions

**lemma** *LIM-eq*: $f \ {-a\rightarrow}\ L = (\forall r{>}0.\ \exists s{>}0.\ \forall x.\ x \neq a \wedge \mathit{norm}\ (x - a) < s \longrightarrow \mathit{norm}\ (f\,x - L) < r)$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$ **and** $L :: {'b}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-I*:
$(\bigwedge r.\ 0 < r \Longrightarrow \exists s{>}0.\ \forall x.\ x \neq a \wedge \mathit{norm}\ (x - a) < s \longrightarrow \mathit{norm}\ (f\,x - L) < r) \Longrightarrow f \ {-a\rightarrow}\ L$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$ **and** $L :: {'b}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-D*: $f \ {-a\rightarrow}\ L \Longrightarrow 0 < r \Longrightarrow \exists s{>}0.\forall x.\ x \neq a \wedge \mathit{norm}\ (x - a) < s \longrightarrow \mathit{norm}\ (f\,x - L) < r$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$ **and** $L :: {'b}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-offset*: $f \ {-a\rightarrow}\ L \Longrightarrow (\lambda x.\ f\ (x + k))\ {-(a - k)\rightarrow}\ L$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-offset-zero*: $f \ {-a\rightarrow}\ L \Longrightarrow (\lambda h.\ f\ (a + h))\ {-0\rightarrow}\ L$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-offset-zero-cancel*: $(\lambda h.\ f\ (a + h))\ {-0\rightarrow}\ L \Longrightarrow f \ {-a\rightarrow}\ L$
**for** $a :: {'a}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-offset-zero-iff*: $\mathit{NO\text{-}MATCH}\ 0\ a \Longrightarrow f \ {-a\rightarrow}\ L \longleftrightarrow (\lambda h.\ f\ (a + h))\ {-0\rightarrow}\ L$
**for** $f :: {'a} :: \mathit{real\text{-}normed\text{-}vector} \Rightarrow \text{-}$
⟨*proof*⟩

**lemma** *tendsto-offset-zero-iff*:
**fixes** $f :: {'a} :: \mathit{real\text{-}normed\text{-}vector} \Rightarrow \text{-}$
**assumes** $\mathit{NO\text{-}MATCH}\ 0\ a\ a \in S\ \mathit{open}\ S$
**shows** $(f \longrightarrow L)\ (\mathit{at}\ a\ \mathit{within}\ S) \longleftrightarrow ((\lambda h.\ f\ (a + h)) \longrightarrow L)\ (\mathit{at}\ 0)$
⟨*proof*⟩

**lemma** *LIM-zero*: $(f \longrightarrow l)\ F \Longrightarrow ((\lambda x.\ f\,x - l) \longrightarrow 0)\ F$
**for** $f :: {'a} \Rightarrow {'b}{::}\mathit{real\text{-}normed\text{-}vector}$
⟨*proof*⟩

**lemma** *LIM-zero-cancel*:
**fixes** $f :: {'a} \Rightarrow {'b}{::}\mathit{real\text{-}normed\text{-}vector}$
**shows** $((\lambda x.\ f\,x - l) \longrightarrow 0)\ F \Longrightarrow (f \longrightarrow l)\ F$
⟨*proof*⟩

**lemma** *LIM-zero-iff*: $((\lambda x.\ f\ x - l) \longrightarrow 0)\ F = (f \longrightarrow l)\ F$
  **for** $f :: 'a \Rightarrow 'b::\mathit{real\text{-}normed\text{-}vector}$
  ⟨*proof*⟩

**lemma** *LIM-imp-LIM*:
  **fixes** $f :: 'a::\mathit{topological\text{-}space} \Rightarrow 'b::\mathit{real\text{-}normed\text{-}vector}$
  **fixes** $g :: 'a::\mathit{topological\text{-}space} \Rightarrow 'c::\mathit{real\text{-}normed\text{-}vector}$
  **assumes** $f$: $f\ -a\rightarrow\ l$
    **and** $le$: $\bigwedge x.\ x \neq a \Longrightarrow \mathit{norm}\ (g\ x - m) \leq \mathit{norm}\ (f\ x - l)$
  **shows** $g\ -a\rightarrow\ m$
  ⟨*proof*⟩

**lemma** *LIM-equal2*:
  **fixes** $f\ g :: 'a::\mathit{real\text{-}normed\text{-}vector} \Rightarrow 'b::\mathit{topological\text{-}space}$
  **assumes** $0 < R$
    **and** $\bigwedge x.\ x \neq a \Longrightarrow \mathit{norm}\ (x - a) < R \Longrightarrow f\ x = g\ x$
  **shows** $g\ -a\rightarrow\ l \Longrightarrow f\ -a\rightarrow\ l$
  ⟨*proof*⟩

**lemma** *LIM-compose2*:
  **fixes** $a :: 'a::\mathit{real\text{-}normed\text{-}vector}$
  **assumes** $f$: $f\ -a\rightarrow\ b$
    **and** $g$: $g\ -b\rightarrow\ c$
    **and** $inj$: $\exists\, d{>}0.\ \forall\, x.\ x \neq a \wedge \mathit{norm}\ (x - a) < d \longrightarrow f\ x \neq b$
  **shows** $(\lambda x.\ g\ (f\ x))\ -a\rightarrow\ c$
  ⟨*proof*⟩

**lemma** *real-LIM-sandwich-zero*:
  **fixes** $f\ g :: 'a::\mathit{topological\text{-}space} \Rightarrow \mathit{real}$
  **assumes** $f$: $f\ -a\rightarrow\ 0$
    **and** $1$: $\bigwedge x.\ x \neq a \Longrightarrow 0 \leq g\ x$
    **and** $2$: $\bigwedge x.\ x \neq a \Longrightarrow g\ x \leq f\ x$
  **shows** $g\ -a\rightarrow\ 0$
⟨*proof*⟩

## 108.11 Continuity

**lemma** *LIM-isCont-iff*: $(f\ -a\rightarrow\ f\ a) = ((\lambda h.\ f\ (a + h))\ -0\rightarrow\ f\ a)$
  **for** $f :: 'a::\mathit{real\text{-}normed\text{-}vector} \Rightarrow 'b::\mathit{topological\text{-}space}$
  ⟨*proof*⟩

**lemma** *isCont-iff*: $\mathit{isCont}\ f\ x = (\lambda h.\ f\ (x + h))\ -0\rightarrow\ f\ x$
  **for** $f :: 'a::\mathit{real\text{-}normed\text{-}vector} \Rightarrow 'b::\mathit{topological\text{-}space}$
  ⟨*proof*⟩

**lemma** *isCont-LIM-compose2*:
  **fixes** $a :: 'a::\mathit{real\text{-}normed\text{-}vector}$
  **assumes** $f$ [*unfolded isCont-def*]: $\mathit{isCont}\ f\ a$
    **and** $g$: $g\ -f\ a\rightarrow\ l$

**and** *inj*: $\exists\, d>0.\ \forall\, x.\ x \neq a \wedge norm\ (x - a) < d \longrightarrow f\, x \neq f\, a$
**shows** $(\lambda x.\ g\ (f\, x))\ {-a\rightarrow}\ l$
⟨*proof*⟩

**lemma** *isCont-norm* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ norm\ (f\, x))\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *isCont-rabs* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ |f\, x|)\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow real$
⟨*proof*⟩

**lemma** *isCont-add* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ g\ a \Longrightarrow isCont\ (\lambda x.\ f\, x + g\ x)\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow 'b::topological\text{-}monoid\text{-}add$
⟨*proof*⟩

**lemma** *isCont-minus* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ - f\ x)\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *isCont-diff* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ g\ a \Longrightarrow isCont\ (\lambda x.\ f\, x - g\ x)\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}vector$
⟨*proof*⟩

**lemma** *isCont-mult* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ g\ a \Longrightarrow isCont\ (\lambda x.\ f\, x * g\ x)\ a$
**for** $f\ g :: 'a::t2\text{-}space \Rightarrow 'b::real\text{-}normed\text{-}algebra$
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *isCont*: $isCont\ g\ a \Longrightarrow isCont\ (\lambda x.\ f\ (g\ x))\ a$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *isCont*: $isCont\ f\ a \Longrightarrow isCont\ g\ a \Longrightarrow isCont\ (\lambda x.\ f\ x ** g\ x)\ a$
⟨*proof*⟩

**lemmas** *isCont-scaleR* [*simp*] =
*bounded-bilinear.isCont* [*OF bounded-bilinear-scaleR*]

**lemmas** *isCont-of-real* [*simp*] =
*bounded-linear.isCont* [*OF bounded-linear-of-real*]

**lemma** *isCont-power* [*simp*]: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ f\, x\ \hat{}\ n)\ a$
**for** $f :: 'a::t2\text{-}space \Rightarrow 'b::\{power,real\text{-}normed\text{-}algebra\}$
⟨*proof*⟩

**lemma** *isCont-sum* [*simp*]: $\forall\, i{\in}A.\ isCont\ (f\ i)\ a \Longrightarrow isCont\ (\lambda x.\ \sum i{\in}A.\ f\ i\ x)\ a$
**for** $f :: 'a \Rightarrow 'b::t2\text{-}space \Rightarrow 'c::topological\text{-}comm\text{-}monoid\text{-}add$
⟨*proof*⟩

## 108.12 Uniform Continuity

**lemma** *uniformly-continuous-on-def*:
  **fixes** $f$ :: $'a$::*metric-space* $\Rightarrow$ $'b$::*metric-space*
  **shows** *uniformly-continuous-on s f* $\longleftrightarrow$
    $(\forall\, e>0.\ \exists\, d>0.\ \forall\, x\in s.\ \forall\, x'\in s.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e)$
  ⟨*proof*⟩

**abbreviation** *isUCont* :: $['a$::*metric-space* $\Rightarrow$ $'b$::*metric-space*$] \Rightarrow bool$
  **where** *isUCont f* $\equiv$ *uniformly-continuous-on UNIV f*

**lemma** *isUCont-def*: *isUCont f* $\longleftrightarrow$ $(\forall\, r>0.\ \exists\, s>0.\ \forall\, x\ y.\ dist\ x\ y < s \longrightarrow dist\ (f\ x)\ (f\ y) < r)$
  ⟨*proof*⟩

**lemma** *isUCont-isCont*: *isUCont f* $\Longrightarrow$ *isCont f x*
  ⟨*proof*⟩

**lemma** *uniformly-continuous-on-Cauchy*:
  **fixes** $f$ :: $'a$::*metric-space* $\Rightarrow$ $'b$::*metric-space*
  **assumes** *uniformly-continuous-on S f Cauchy X* $\bigwedge n.\ X\ n \in S$
  **shows** *Cauchy* $(\lambda n.\ f\ (X\ n))$
  ⟨*proof*⟩

**lemma** *isUCont-Cauchy*: *isUCont f* $\Longrightarrow$ *Cauchy X* $\Longrightarrow$ *Cauchy* $(\lambda n.\ f\ (X\ n))$
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *isUCont*: *isUCont f*
  ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *Cauchy*: *Cauchy X* $\Longrightarrow$ *Cauchy* $(\lambda n.\ f\ (X\ n))$
  ⟨*proof*⟩

**lemma** *LIM-less-bound*:
  **fixes** $f$ :: *real* $\Rightarrow$ *real*
  **assumes** *ev*: $b < x\ \forall\ x' \in \{\ b <..< x\}.\ 0 \le f\ x'$ **and** *isCont f x*
  **shows** $0 \le f\ x$
⟨*proof*⟩

## 108.13 Nested Intervals and Bisection – Needed for Compactness

**lemma** *nested-sequence-unique*:
  **assumes** $\forall\, n.\ f\ n \le f\ (Suc\ n)$ $\forall\, n.\ g\ (Suc\ n) \le g\ n$ $\forall\, n.\ f\ n \le g\ n$ $(\lambda n.\ f\ n - g\ n) \longrightarrow 0$
  **shows** $\exists\, l$::*real*. $((\forall\, n.\ f\ n \le l) \wedge f \longrightarrow l) \wedge ((\forall\, n.\ l \le g\ n) \wedge g \longrightarrow l)$
⟨*proof*⟩

**lemma** *Bolzano*[*consumes 1*, *case-names trans local*]:
  **fixes** $P$ :: *real* $\Rightarrow$ *real* $\Rightarrow$ *bool*

**assumes** $[arith]$: $a \le b$
**and** $trans$: $\bigwedge a\ b\ c.\ P\ a\ b \Longrightarrow P\ b\ c \Longrightarrow a \le b \Longrightarrow b \le c \Longrightarrow P\ a\ c$
**and** $local$: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow \exists\, d{>}0.\ \forall\, a\ b.\ a \le x \wedge x \le b \wedge b - a < d \longrightarrow P\ a\ b$
**shows** $P\ a\ b$
⟨*proof*⟩

**lemma** *compact-Icc*[*simp*, *intro*]: *compact* $\{a\ ..\ b{::}real\}$
⟨*proof*⟩

**lemma** *continuous-image-closed-interval*:
**fixes** $a\ b$ **and** $f$ :: *real* ⇒ *real*
**defines** $S \equiv \{a..b\}$
**assumes** $a \le b$ **and** $f$: *continuous-on* $S\ f$
**shows** $\exists\, c\ d.\ f\text{`}S = \{c..d\} \wedge c \le d$
⟨*proof*⟩

**lemma** *open-Collect-positive*:
**fixes** $f$ :: *'a::topological-space* ⇒ *real*
**assumes** $f$: *continuous-on* $s\ f$
**shows** $\exists\, A.\ open\ A \wedge A \cap s = \{x{\in}s.\ 0 < f\ x\}$
⟨*proof*⟩

**lemma** *open-Collect-less-Int*:
**fixes** $f\ g$ :: *'a::topological-space* ⇒ *real*
**assumes** $f$: *continuous-on* $s\ f$
**and** $g$: *continuous-on* $s\ g$
**shows** $\exists\, A.\ open\ A \wedge A \cap s = \{x{\in}s.\ f\ x < g\ x\}$
⟨*proof*⟩

## 108.14 Boundedness of continuous functions

By bisection, function continuous on closed interval is bounded above

**lemma** *isCont-eq-Ub*:
**fixes** $f$ :: *real* ⇒ *'a::linorder-topology*
**shows** $a \le b \Longrightarrow \forall\, x{::}real.\ a \le x \wedge x \le b \longrightarrow isCont\ f\ x \Longrightarrow$
$\exists\, M.\ (\forall\, x.\ a \le x \wedge x \le b \longrightarrow f\ x \le M) \wedge (\exists\, x.\ a \le x \wedge x \le b \wedge f\ x = M)$
⟨*proof*⟩

**lemma** *isCont-eq-Lb*:
**fixes** $f$ :: *real* ⇒ *'a::linorder-topology*
**shows** $a \le b \Longrightarrow \forall\, x.\ a \le x \wedge x \le b \longrightarrow isCont\ f\ x \Longrightarrow$
$\exists\, M.\ (\forall\, x.\ a \le x \wedge x \le b \longrightarrow M \le f\ x) \wedge (\exists\, x.\ a \le x \wedge x \le b \wedge f\ x = M)$
⟨*proof*⟩

**lemma** *isCont-bounded*:
**fixes** $f$ :: *real* ⇒ *'a::linorder-topology*
**shows** $a \le b \Longrightarrow \forall\, x.\ a \le x \wedge x \le b \longrightarrow isCont\ f\ x \Longrightarrow \exists\, M.\ \forall\, x.\ a \le x \wedge x \le$

$b \longrightarrow f\,x \le M$
$\langle proof \rangle$

**lemma** *isCont-has-Ub*:
**fixes** $f$ :: $real \Rightarrow 'a::linorder\text{-}topology$
**shows** $a \le b \Longrightarrow \forall x.\ a \le x \land x \le b \longrightarrow isCont\ f\ x \Longrightarrow$
$\exists M.\ (\forall x.\ a \le x \land x \le b \longrightarrow f\,x \le M) \land (\forall N.\ N < M \longrightarrow (\exists x.\ a \le x \land x \le b \land N < f\,x))$
$\langle proof \rangle$

**lemma** *isCont-Lb-Ub*:
**fixes** $f$ :: $real \Rightarrow real$
**assumes** $a \le b\ \forall x.\ a \le x \land x \le b \longrightarrow isCont\ f\ x$
**shows** $\exists L\ M.\ (\forall x.\ a \le x \land x \le b \longrightarrow L \le f\,x \land f\,x \le M) \land$
$(\forall y.\ L \le y \land y \le M \longrightarrow (\exists x.\ a \le x \land x \le b \land (f\,x = y)))$
$\langle proof \rangle$

Continuity of inverse function.

**lemma** *isCont-inverse-function*:
**fixes** $f\ g$ :: $real \Rightarrow real$
**assumes** $d$: $0 < d$
**and** $inj$: $\bigwedge z.\ |z-x| \le d \Longrightarrow g\ (f\ z) = z$
**and** $cont$: $\bigwedge z.\ |z-x| \le d \Longrightarrow isCont\ f\ z$
**shows** $isCont\ g\ (f\ x)$
$\langle proof \rangle$

**lemma** *isCont-inverse-function2*:
**fixes** $f\ g$ :: $real \Rightarrow real$
**shows**
$[\![a < x;\ x < b;$
$\bigwedge z.\ [\![a \le z;\ z \le b]\!] \Longrightarrow g\ (f\ z) = z;$
$\bigwedge z.\ [\![a \le z;\ z \le b]\!] \Longrightarrow isCont\ f\ z]\!] \Longrightarrow isCont\ g\ (f\ x)$
$\langle proof \rangle$

Bartle/Sherbert: Introduction to Real Analysis, Theorem 4.2.9, p. 110.

**lemma** *LIM-fun-gt-zero*: $f\ {-c\rightarrow}\ l \Longrightarrow 0 < l \Longrightarrow \exists r.\ 0 < r \land (\forall x.\ x \ne c \land |c - x| < r \longrightarrow 0 < f\,x)$
**for** $f$ :: $real \Rightarrow real$
$\langle proof \rangle$

**lemma** *LIM-fun-less-zero*: $f\ {-c\rightarrow}\ l \Longrightarrow l < 0 \Longrightarrow \exists r.\ 0 < r \land (\forall x.\ x \ne c \land |c - x| < r \longrightarrow f\,x < 0)$
**for** $f$ :: $real \Rightarrow real$
$\langle proof \rangle$

**lemma** *LIM-fun-not-zero*: $f\ {-c\rightarrow}\ l \Longrightarrow l \ne 0 \Longrightarrow \exists r.\ 0 < r \land (\forall x.\ x \ne c \land |c - x| < r \longrightarrow f\,x \ne 0)$
**for** $f$ :: $real \Rightarrow real$
$\langle proof \rangle$

**lemma** *Lim-topological*:
$(f \longrightarrow l)$ *net* $\longleftrightarrow$
*trivial-limit net* $\vee$ $(\forall S.$ *open* $S \longrightarrow l \in S \longrightarrow$ *eventually* $(\lambda x.\ f\ x \in S)$ *net*$)$
⟨*proof*⟩

**lemma** *eventually-within-Un*:
*eventually* $P$ (*at* $x$ *within* $(s \cup t)$) $\longleftrightarrow$
*eventually* $P$ (*at* $x$ *within* $s$) $\wedge$ *eventually* $P$ (*at* $x$ *within* $t$)
⟨*proof*⟩

**lemma** *Lim-within-Un*:
$(f \longrightarrow l)$ (*at* $x$ *within* $(s \cup t)$) $\longleftrightarrow$
$(f \longrightarrow l)$ (*at* $x$ *within* $s$) $\wedge$ $(f \longrightarrow l)$ (*at* $x$ *within* $t$)
⟨*proof*⟩

**end**

**theory** *Inequalities*
**imports** *Real-Vector-Spaces*
**begin**

**lemma** *Chebyshev-sum-upper*:
**fixes** $a\ b$::*nat* $\Rightarrow$ *'a*::*linordered-idom*
**assumes** $\bigwedge i\ j.\ i \le j \Longrightarrow j < n \Longrightarrow a\ i \le a\ j$
**assumes** $\bigwedge i\ j.\ i \le j \Longrightarrow j < n \Longrightarrow b\ i \ge b\ j$
**shows** *of-nat* $n * (\sum k{=}0..{<}n.\ a\ k * b\ k) \le (\sum k{=}0..{<}n.\ a\ k) * (\sum k{=}0..{<}n.$
$b\ k)$
⟨*proof*⟩

**lemma** *Chebyshev-sum-upper-nat*:
**fixes** $a\ b$ :: *nat* $\Rightarrow$ *nat*
**shows** $(\bigwedge i\ j.\ [\![\ i{\le}j;\ j{<}n\ ]\!] \Longrightarrow a\ i \le a\ j) \Longrightarrow$
$(\bigwedge i\ j.\ [\![\ i{\le}j;\ j{<}n\ ]\!] \Longrightarrow b\ i \ge b\ j) \Longrightarrow$
$n * (\sum i{=}0..{<}n.\ a\ i * b\ i) \le (\sum i{=}0..{<}n.\ a\ i) * (\sum i{=}0..{<}n.\ b\ i)$
⟨*proof*⟩

**end**

# 109 Infinite Series

**theory** *Series*
**imports** *Limits Inequalities*
**begin**

## 109.1 Definition of infinite summability

**definition** *sums* :: (*nat* $\Rightarrow$ *'a*::{*topological-space*, *comm-monoid-add*}) $\Rightarrow$ *'a* $\Rightarrow$ *bool*

(**infixr** ‹*sums*› *80*)
**where** $f$ *sums* $s \longleftrightarrow (\lambda n. \sum i<n. f\ i) \longlonglongrightarrow s$

**definition** *summable* :: $(nat \Rightarrow 'a::\{topological\text{-}space, comm\text{-}monoid\text{-}add\}) \Rightarrow bool$
**where** *summable* $f \longleftrightarrow (\exists s. f$ *sums* $s)$

**definition** *suminf* :: $(nat \Rightarrow 'a::\{topological\text{-}space, comm\text{-}monoid\text{-}add\}) \Rightarrow 'a$
(**binder** ‹$\sum$› *10*)
**where** *suminf* $f = (THE\ s. f$ *sums* $s)$

Variants of the definition

**lemma** *sums-def'*: $f$ *sums* $s \longleftrightarrow (\lambda n. \sum i = 0..n. f\ i) \longlonglongrightarrow s$
⟨*proof*⟩

**lemma** *sums-def-le*: $f$ *sums* $s \longleftrightarrow (\lambda n. \sum i \le n. f\ i) \longlonglongrightarrow s$
⟨*proof*⟩

**lemma** *bounded-imp-summable*:
**fixes** $a :: nat \Rightarrow 'a::\{conditionally\text{-}complete\text{-}linorder, linorder\text{-}topology, linordered\text{-}comm\text{-}semiring\text{-}strict\}$
**assumes** $0$: $\bigwedge n. a\ n \ge 0$ **and** *bounded*: $\bigwedge n. (\sum k \le n. a\ k) \le B$
**shows** *summable* $a$
⟨*proof*⟩

## 109.2 Infinite summability on topological monoids

**lemma** *sums-subst*[*trans*]: $f = g \Longrightarrow g$ *sums* $z \Longrightarrow f$ *sums* $z$
⟨*proof*⟩

**lemma** *sums-cong*: $(\bigwedge n. f\ n = g\ n) \Longrightarrow f$ *sums* $c \longleftrightarrow g$ *sums* $c$
⟨*proof*⟩

**lemma** *sums-summable*: $f$ *sums* $l \Longrightarrow$ *summable* $f$
⟨*proof*⟩

**lemma** *summable-iff-convergent*: *summable* $f \longleftrightarrow$ *convergent* $(\lambda n. \sum i<n. f\ i)$
⟨*proof*⟩

**lemma** *summable-iff-convergent'*: *summable* $f \longleftrightarrow$ *convergent* $(\lambda n. sum\ f\ \{..n\})$
⟨*proof*⟩

**lemma** *suminf-eq-lim*: *suminf* $f = lim\ (\lambda n. \sum i<n. f\ i)$
⟨*proof*⟩

**lemma** *sums-zero*[*simp*, *intro*]: $(\lambda n. 0)$ *sums* $0$
⟨*proof*⟩

**lemma** *summable-zero*[*simp*, *intro*]: *summable* $(\lambda n. 0)$
⟨*proof*⟩

**lemma** *sums-group*: $f$ *sums* $s \Longrightarrow 0 < k \Longrightarrow (\lambda n.\ sum\ f\ \{n * k ..< n * k + k\})$ *sums* $s$
⟨*proof*⟩

**lemma** *suminf-cong*: $(\bigwedge n.\ f\ n = g\ n) \Longrightarrow suminf\ f = suminf\ g$
⟨*proof*⟩

**lemma** *summable-cong*:
**fixes** $f\ g :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
**assumes** *eventually* $(\lambda x.\ f\ x = g\ x)$ *sequentially*
**shows** *summable* $f =$ *summable* $g$
⟨*proof*⟩

**lemma** *sums-finite*:
**assumes** [*simp*]: *finite* $N$
**and** $f$: $\bigwedge n.\ n \notin N \Longrightarrow f\ n = 0$
**shows** $f$ *sums* $(\sum n{\in}N.\ f\ n)$
⟨*proof*⟩

**corollary** *sums-0*: $(\bigwedge n.\ f\ n = 0) \Longrightarrow (f\ sums\ 0)$
⟨*proof*⟩

**lemma** *summable-finite*: *finite* $N \Longrightarrow (\bigwedge n.\ n \notin N \Longrightarrow f\ n = 0) \Longrightarrow$ *summable* $f$
⟨*proof*⟩

**lemma** *sums-If-finite-set*: *finite* $A \Longrightarrow (\lambda r.\ if\ r \in A\ then\ f\ r\ else\ 0)$ *sums* $(\sum r{\in}A.\ f\ r)$
⟨*proof*⟩

**lemma** *summable-If-finite-set*[*simp*, *intro*]: *finite* $A \Longrightarrow$ *summable* $(\lambda r.\ if\ r \in A\ then\ f\ r\ else\ 0)$
⟨*proof*⟩

**lemma** *sums-If-finite*: *finite* $\{r.\ P\ r\} \Longrightarrow (\lambda r.\ if\ P\ r\ then\ f\ r\ else\ 0)$ *sums* $(\sum r \mid P\ r.\ f\ r)$
⟨*proof*⟩

**lemma** *summable-If-finite*[*simp*, *intro*]: *finite* $\{r.\ P\ r\} \Longrightarrow$ *summable* $(\lambda r.\ if\ P\ r\ then\ f\ r\ else\ 0)$
⟨*proof*⟩

**lemma** *sums-single*: $(\lambda r.\ if\ r = i\ then\ f\ r\ else\ 0)$ *sums* $f\ i$
⟨*proof*⟩

**lemma** *summable-single*[*simp*, *intro*]: *summable* $(\lambda r.\ if\ r = i\ then\ f\ r\ else\ 0)$
⟨*proof*⟩

**context**
**fixes** $f :: nat \Rightarrow 'a::\{t2\text{-}space, comm\text{-}monoid\text{-}add\}$

**begin**

**lemma** *summable-sums*[*intro*]: $summable\ f \Longrightarrow f\ sums\ (suminf\ f)$
⟨*proof*⟩

**lemma** *summable-LIMSEQ*: $summable\ f \Longrightarrow (\lambda n.\ \sum i<n.\ f\ i) \longrightarrow suminf\ f$
⟨*proof*⟩

**lemma** *summable-LIMSEQ'*: $summable\ f \Longrightarrow (\lambda n.\ \sum i\leq n.\ f\ i) \longrightarrow suminf\ f$
⟨*proof*⟩

**lemma** *sums-unique*: $f\ sums\ s \Longrightarrow s = suminf\ f$
⟨*proof*⟩

**lemma** *sums-iff*: $f\ sums\ x \longleftrightarrow summable\ f \wedge suminf\ f = x$
⟨*proof*⟩

**lemma** *summable-sums-iff*: $summable\ f \longleftrightarrow f\ sums\ suminf\ f$
⟨*proof*⟩

**lemma** *sums-unique2*: $f\ sums\ a \Longrightarrow f\ sums\ b \Longrightarrow a = b$
**for** $a\ b :: 'a$
⟨*proof*⟩

**lemma** *sums-Uniq*: $\exists_{\leq 1} a.\ f\ sums\ a$
**for** $a\ b :: 'a$
⟨*proof*⟩

**lemma** *suminf-finite*:
**assumes** $N$: *finite* $N$
**and** $f$: $\bigwedge n.\ n \notin N \Longrightarrow f\ n = 0$
**shows** $suminf\ f = (\sum n\in N.\ f\ n)$
⟨*proof*⟩

**end**

**lemma** *suminf-zero*[*simp*]: $suminf\ (\lambda n.\ 0::'a::\{t2\text{-}space,\ comm\text{-}monoid\text{-}add\}) = 0$
⟨*proof*⟩

## 109.3 Infinite summability on ordered, topological monoids

**lemma** *sums-le*: $(\bigwedge n.\ f\ n \leq g\ n) \Longrightarrow f\ sums\ s \Longrightarrow g\ sums\ t \Longrightarrow s \leq t$
**for** $f\ g :: nat \Rightarrow 'a::\{ordered\text{-}comm\text{-}monoid\text{-}add,linorder\text{-}topology\}$
⟨*proof*⟩

**context**
**fixes** $f :: nat \Rightarrow 'a::\{ordered\text{-}comm\text{-}monoid\text{-}add,linorder\text{-}topology\}$
**begin**

**lemma** *suminf-le*: $(\bigwedge n.\ f\ n \leq g\ n) \Longrightarrow$ *summable f* $\Longrightarrow$ *summable g* $\Longrightarrow$ *suminf f* $\leq$ *suminf g*
⟨*proof*⟩

**lemma** *sum-le-suminf*:
**shows** *summable f* $\Longrightarrow$ *finite I* $\Longrightarrow$ $(\bigwedge n.\ n \in\!-\ I \Longrightarrow 0 \leq f\ n) \Longrightarrow$ *sum f I* $\leq$ *suminf f*
⟨*proof*⟩

**lemma** *suminf-nonneg*: *summable f* $\Longrightarrow$ $(\bigwedge n.\ 0 \leq f\ n) \Longrightarrow 0 \leq$ *suminf f*
⟨*proof*⟩

**lemma** *suminf-le-const*: *summable f* $\Longrightarrow$ $(\bigwedge n.$ *sum f* $\{..<n\} \leq x) \Longrightarrow$ *suminf f* $\leq x$
⟨*proof*⟩

**lemma** *suminf-eq-zero-iff*:
**assumes** *summable f* **and** *pos*: $\bigwedge n.\ 0 \leq f\ n$
**shows** *suminf f* $= 0 \longleftrightarrow (\forall n.\ f\ n = 0)$
⟨*proof*⟩

**lemma** *suminf-pos-iff*: *summable f* $\Longrightarrow$ $(\bigwedge n.\ 0 \leq f\ n) \Longrightarrow 0 <$ *suminf f* $\longleftrightarrow (\exists i.\ 0 < f\ i)$
⟨*proof*⟩

**lemma** *suminf-pos2*:
**assumes** *summable f* $\bigwedge n.\ 0 \leq f\ n\ 0 < f\ i$
**shows** $0 <$ *suminf f*
⟨*proof*⟩

**lemma** *suminf-pos*: *summable f* $\Longrightarrow$ $(\bigwedge n.\ 0 < f\ n) \Longrightarrow 0 <$ *suminf f*
⟨*proof*⟩

**end**

**context**
**fixes** $f$ :: *nat* $\Rightarrow$ *'a*::{*ordered-cancel-comm-monoid-add*,*linorder-topology*}
**begin**

**lemma** *sum-less-suminf2*:
*summable f* $\Longrightarrow$ $(\bigwedge m.\ m \geq n \Longrightarrow 0 \leq f\ m) \Longrightarrow n \leq i \Longrightarrow 0 < f\ i \Longrightarrow$ *sum f* $\{..<n\} <$ *suminf f*
⟨*proof*⟩

**lemma** *sum-less-suminf*: *summable f* $\Longrightarrow$ $(\bigwedge m.\ m \geq n \Longrightarrow 0 < f\ m) \Longrightarrow$ *sum f* $\{..<n\} <$ *suminf f*
⟨*proof*⟩

**end**

**lemma** *summableI-nonneg-bounded*:
**fixes** $f :: nat \Rightarrow 'a::\{ordered\text{-}comm\text{-}monoid\text{-}add, linorder\text{-}topology, conditionally\text{-}complete\text{-}linorder\}$
**assumes** *pos*[*simp*]: $\bigwedge n.\ 0 \leq f\ n$
**and** *le*: $\bigwedge n.\ (\sum i<n.\ f\ i) \leq x$
**shows** *summable f*
⟨*proof*⟩

**lemma** *summableI*[*intro*, *simp*]: *summable f*
**for** $f :: nat \Rightarrow 'a::\{canonically\text{-}ordered\text{-}monoid\text{-}add, linorder\text{-}topology, complete\text{-}linorder\}$
⟨*proof*⟩

**lemma** *suminf-eq-SUP-real*:
**assumes** *X*: *summable X* $\bigwedge i.\ 0 \leq X\ i$ **shows** $suminf\ X = (SUP\ i.\ \sum n<i.\ X\ n::real)$
⟨*proof*⟩

## 109.4 Infinite summability on topological monoids

**context**
**fixes** $f\ g :: nat \Rightarrow 'a::\{t2\text{-}space, topological\text{-}comm\text{-}monoid\text{-}add\}$
**begin**

**lemma** *sums-Suc*:
**assumes** $(\lambda n.\ f\ (Suc\ n))$ *sums l*
**shows** *f sums* $(l + f\ 0)$
⟨*proof*⟩

**lemma** *sums-add*: *f sums a* $\Longrightarrow$ *g sums b* $\Longrightarrow$ $(\lambda n.\ f\ n + g\ n)$ *sums* $(a + b)$
⟨*proof*⟩

**lemma** *summable-add*: *summable f* $\Longrightarrow$ *summable g* $\Longrightarrow$ *summable* $(\lambda n.\ f\ n + g\ n)$
⟨*proof*⟩

**lemma** *suminf-add*: *summable f* $\Longrightarrow$ *summable g* $\Longrightarrow$ $suminf\ f + suminf\ g = (\sum n.\ f\ n + g\ n)$
⟨*proof*⟩

**end**

**context**
**fixes** $f :: 'i \Rightarrow nat \Rightarrow 'a::\{t2\text{-}space, topological\text{-}comm\text{-}monoid\text{-}add\}$
**and** $I :: 'i\ set$
**begin**

**lemma** *sums-sum*: $(\bigwedge i.\ i \in I \Longrightarrow (f\ i)\ sums\ (x\ i)) \Longrightarrow (\lambda n.\ \sum i\in I.\ f\ i\ n)\ sums\ (\sum i\in I.\ x\ i)$
⟨*proof*⟩

**lemma** *suminf-sum*: $(\bigwedge i.\ i \in I \Longrightarrow \mathit{summable}\ (f\ i)) \Longrightarrow (\sum n.\ \sum i{\in}I.\ f\ i\ n) = (\sum i{\in}I.\ \sum n.\ f\ i\ n)$
⟨*proof*⟩

**lemma** *summable-sum*: $(\bigwedge i.\ i \in I \Longrightarrow \mathit{summable}\ (f\ i)) \Longrightarrow \mathit{summable}\ (\lambda n.\ \sum i{\in}I.\ f\ i\ n)$
⟨*proof*⟩

**end**

**lemma** *sums-If-finite-set'*:
**fixes** $f\ g :: \mathit{nat} \Rightarrow {}'a::\{\mathit{t2\text{-}space},\mathit{topological\text{-}ab\text{-}group\text{-}add}\}$
**assumes** $g\ \mathit{sums}\ S$ **and** $\mathit{finite}\ A$ **and** $S' = S + (\sum n{\in}A.\ f\ n - g\ n)$
**shows** $(\lambda n.\ \mathit{if}\ n \in A\ \mathit{then}\ f\ n\ \mathit{else}\ g\ n)\ \mathit{sums}\ S'$
⟨*proof*⟩

## 109.5 Infinite summability on real normed vector spaces

**context**
**fixes** $f :: \mathit{nat} \Rightarrow {}'a::\mathit{real\text{-}normed\text{-}vector}$
**begin**

**lemma** *sums-Suc-iff*: $(\lambda n.\ f\ (\mathit{Suc}\ n))\ \mathit{sums}\ s \longleftrightarrow f\ \mathit{sums}\ (s + f\ 0)$
⟨*proof*⟩

**lemma** *summable-Suc-iff*: $\mathit{summable}\ (\lambda n.\ f\ (\mathit{Suc}\ n)) = \mathit{summable}\ f$
⟨*proof*⟩

**lemma** *sums-Suc-imp*: $f\ 0 = 0 \Longrightarrow (\lambda n.\ f\ (\mathit{Suc}\ n))\ \mathit{sums}\ s \Longrightarrow (\lambda n.\ f\ n)\ \mathit{sums}\ s$
⟨*proof*⟩

**end**

**context**
**fixes** $f :: \mathit{nat} \Rightarrow {}'a::\mathit{real\text{-}normed\text{-}vector}$
**begin**

**lemma** *sums-diff*: $f\ \mathit{sums}\ a \Longrightarrow g\ \mathit{sums}\ b \Longrightarrow (\lambda n.\ f\ n - g\ n)\ \mathit{sums}\ (a - b)$
⟨*proof*⟩

**lemma** *summable-diff*: $\mathit{summable}\ f \Longrightarrow \mathit{summable}\ g \Longrightarrow \mathit{summable}\ (\lambda n.\ f\ n - g\ n)$
⟨*proof*⟩

**lemma** *suminf-diff*: $\mathit{summable}\ f \Longrightarrow \mathit{summable}\ g \Longrightarrow \mathit{suminf}\ f - \mathit{suminf}\ g = (\sum n.\ f\ n - g\ n)$
⟨*proof*⟩

**lemma** *sums-minus*: $f$ *sums* $a \Longrightarrow (\lambda n.\ - f\ n)$ *sums* $(- a)$
⟨*proof*⟩

**lemma** *summable-minus*: *summable* $f \Longrightarrow$ *summable* $(\lambda n.\ - f\ n)$
⟨*proof*⟩

**lemma** *suminf-minus*: *summable* $f \Longrightarrow (\sum n.\ - f\ n) = - (\sum n.\ f\ n)$
⟨*proof*⟩

**lemma** *sums-iff-shift*: $(\lambda i.\ f\ (i + n))$ *sums* $s \longleftrightarrow f$ *sums* $(s + (\sum i{<}n.\ f\ i))$
⟨*proof*⟩

**corollary** *sums-iff-shift'*: $(\lambda i.\ f\ (i + n))$ *sums* $(s - (\sum i{<}n.\ f\ i)) \longleftrightarrow f$ *sums* $s$
⟨*proof*⟩

**lemma** *sums-zero-iff-shift*:
**assumes** $\bigwedge i.\ i < n \Longrightarrow f\ i = 0$
**shows** $(\lambda i.\ f\ (i{+}n))$ *sums* $s \longleftrightarrow (\lambda i.\ f\ i)$ *sums* $s$
⟨*proof*⟩

**lemma** *summable-iff-shift* [*simp*]: *summable* $(\lambda n.\ f\ (n + k)) \longleftrightarrow$ *summable* $f$
⟨*proof*⟩

**lemma** *sums-split-initial-segment*: $f$ *sums* $s \Longrightarrow (\lambda i.\ f\ (i + n))$ *sums* $(s - (\sum i{<}n.\ f\ i))$
⟨*proof*⟩

**lemma** *summable-ignore-initial-segment*: *summable* $f \Longrightarrow$ *summable* $(\lambda n.\ f(n + k))$
⟨*proof*⟩

**lemma** *suminf-minus-initial-segment*: *summable* $f \Longrightarrow (\sum n.\ f\ (n + k)) = (\sum n.\ f\ n) - (\sum i{<}k.\ f\ i)$
⟨*proof*⟩

**lemma** *suminf-split-initial-segment*: *summable* $f \Longrightarrow$ *suminf* $f = (\sum n.\ f(n + k)) + (\sum i{<}k.\ f\ i)$
⟨*proof*⟩

**lemma** *suminf-split-head*: *summable* $f \Longrightarrow (\sum n.\ f\ (Suc\ n)) =$ *suminf* $f - f\ 0$
⟨*proof*⟩

**lemma** *suminf-exist-split*:
**fixes** $r$ :: *real*
**assumes** $0 < r$ **and** *summable* $f$
**shows** $\exists N.\ \forall n{\geq}N.\ norm\ (\sum i.\ f\ (i + n)) < r$
⟨*proof*⟩

**lemma** *summable-LIMSEQ-zero*:

**assumes** *summable f* **shows** $f \longrightarrow 0$
⟨*proof*⟩

**lemma** *summable-imp-convergent*: *summable f* $\Longrightarrow$ *convergent f*
⟨*proof*⟩

**lemma** *summable-imp-Bseq*: *summable f* $\Longrightarrow$ *Bseq f*
⟨*proof*⟩

**end**

**lemma** *summable-minus-iff*: *summable* $(\lambda n. - f\ n) \longleftrightarrow$ *summable f*
**for** $f$ :: *nat* $\Rightarrow$ *'a::real-normed-vector*
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *sums*: $(\lambda n.\ X\ n)$ *sums* $a \Longrightarrow (\lambda n.\ f\ (X\ n))$ *sums* $(f\ a)$
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *summable*: *summable* $(\lambda n.\ X\ n) \Longrightarrow$ *summable* $(\lambda n.\ f\ (X\ n))$
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *suminf*: *summable* $(\lambda n.\ X\ n) \Longrightarrow f\ (\sum n.\ X\ n) = (\sum n.\ f\ (X\ n))$
⟨*proof*⟩

**lemmas** *sums-of-real* = *bounded-linear.sums* [*OF bounded-linear-of-real*]
**lemmas** *summable-of-real* = *bounded-linear.summable* [*OF bounded-linear-of-real*]
**lemmas** *suminf-of-real* = *bounded-linear.suminf* [*OF bounded-linear-of-real*]

**lemmas** *sums-scaleR-left* = *bounded-linear.sums*[*OF bounded-linear-scaleR-left*]
**lemmas** *summable-scaleR-left* = *bounded-linear.summable*[*OF bounded-linear-scaleR-left*]
**lemmas** *suminf-scaleR-left* = *bounded-linear.suminf*[*OF bounded-linear-scaleR-left*]

**lemmas** *sums-scaleR-right* = *bounded-linear.sums*[*OF bounded-linear-scaleR-right*]
**lemmas** *summable-scaleR-right* = *bounded-linear.summable*[*OF bounded-linear-scaleR-right*]
**lemmas** *suminf-scaleR-right* = *bounded-linear.suminf*[*OF bounded-linear-scaleR-right*]

**lemma** *summable-const-iff*: *summable* $(\lambda\text{-}.\ c) \longleftrightarrow c = 0$
**for** $c$ :: *'a::real-normed-vector*
⟨*proof*⟩

## 109.6 Infinite summability on real normed algebras

**context**
**fixes** $f$ :: *nat* $\Rightarrow$ *'a::real-normed-algebra*
**begin**

**lemma** *sums-mult*: *f sums a* $\Longrightarrow (\lambda n.\ c * f\ n)$ *sums* $(c * a)$

⟨*proof*⟩

**lemma** *summable-mult*: *summable f* $\Longrightarrow$ *summable* $(\lambda n.\ c * f\ n)$
⟨*proof*⟩

**lemma** *suminf-mult*: *summable f* $\Longrightarrow$ *suminf* $(\lambda n.\ c * f\ n) = c *$ *suminf f*
⟨*proof*⟩

**lemma** *sums-mult2*: *f sums a* $\Longrightarrow$ $(\lambda n.\ f\ n * c)$ *sums* $(a * c)$
⟨*proof*⟩

**lemma** *summable-mult2*: *summable f* $\Longrightarrow$ *summable* $(\lambda n.\ f\ n * c)$
⟨*proof*⟩

**lemma** *suminf-mult2*: *summable f* $\Longrightarrow$ *suminf f* $* c = (\sum n.\ f\ n * c)$
⟨*proof*⟩

**end**

**lemma** *sums-mult-iff*:
**fixes** $f$ :: *nat* $\Rightarrow$ *'a*::{*real-normed-algebra,field*}
**assumes** $c \neq 0$
**shows** $(\lambda n.\ c * f\ n)$ *sums* $(c * d) \longleftrightarrow$ *f sums d*
⟨*proof*⟩

**lemma** *sums-mult2-iff*:
**fixes** $f$ :: *nat* $\Rightarrow$ *'a*::{*real-normed-algebra,field*}
**assumes** $c \neq 0$
**shows** $(\lambda n.\ f\ n * c)$ *sums* $(d * c) \longleftrightarrow$ *f sums d*
⟨*proof*⟩

**lemma** *sums-of-real-iff*:
$(\lambda n.$ *of-real* $(f\ n)$ :: *'a*::*real-normed-div-algebra*) *sums of-real c* $\longleftrightarrow$ *f sums c*
⟨*proof*⟩

## 109.7 Infinite summability on real normed fields

**context**
**fixes** $c$ :: *'a*::*real-normed-field*
**begin**

**lemma** *sums-divide*: *f sums a* $\Longrightarrow$ $(\lambda n.\ f\ n\ /\ c)$ *sums* $(a\ /\ c)$
⟨*proof*⟩

**lemma** *summable-divide*: *summable f* $\Longrightarrow$ *summable* $(\lambda n.\ f\ n\ /\ c)$
⟨*proof*⟩

**lemma** *suminf-divide*: *summable f* $\Longrightarrow$ *suminf* $(\lambda n.\ f\ n\ /\ c) =$ *suminf f* $/\ c$
⟨*proof*⟩

**lemma** *summable-inverse-divide*: *summable* (*inverse* $\circ$ *f*) $\Longrightarrow$ *summable* ($\lambda n.$ *c* / *f n*)
⟨*proof*⟩

**lemma** *sums-mult-D*: ($\lambda n.$ *c* $*$ *f n*) *sums a* $\Longrightarrow$ *c* $\neq$ *0* $\Longrightarrow$ *f sums* (*a*/*c*)
⟨*proof*⟩

**lemma** *summable-mult-D*: *summable* ($\lambda n.$ *c* $*$ *f n*) $\Longrightarrow$ *c* $\neq$ *0* $\Longrightarrow$ *summable f*
⟨*proof*⟩

Sum of a geometric progression.

**lemma** *geometric-sums*:
**assumes** *norm c* < *1*
**shows** ($\lambda n.$ *c*^*n*) *sums* (*1* / (*1* − *c*))
⟨*proof*⟩

**lemma** *summable-geometric*: *norm c* < *1* $\Longrightarrow$ *summable* ($\lambda n.$ *c*^*n*)
⟨*proof*⟩

**lemma** *suminf-geometric*: *norm c* < *1* $\Longrightarrow$ *suminf* ($\lambda n.$ *c*^*n*) = *1* / (*1* − *c*)
⟨*proof*⟩

**lemma** *summable-geometric-iff* [*simp*]: *summable* ($\lambda n.$ *c* ^ *n*) $\longleftrightarrow$ *norm c* < *1*
⟨*proof*⟩

**end**

Biconditional versions for constant factors

**context**
**fixes** *c* :: *'a*::*real-normed-field*
**begin**

**lemma** *summable-cmult-iff* [*simp*]: *summable* ($\lambda n.$ *c* $*$ *f n*) $\longleftrightarrow$ *c*=*0* $\vee$ *summable f*
⟨*proof*⟩

**lemma** *summable-divide-iff* [*simp*]: *summable* ($\lambda n.$ *f n* / *c*) $\longleftrightarrow$ *c*=*0* $\vee$ *summable f*
⟨*proof*⟩

**end**

**lemma** *power-half-series*: ($\lambda n.$ (*1*/*2*::*real*)^*Suc n*) *sums 1*
⟨*proof*⟩

## 109.8 Telescoping

**lemma** *telescope-sums*:

**fixes** $c$ :: $'a$::*real-normed-vector*
**assumes** $f \longrightarrow c$
**shows** $(\lambda n.\ f\ (Suc\ n) - f\ n)$ *sums* $(c - f\ 0)$
⟨*proof*⟩

**lemma** *telescope-sums'*:
**fixes** $c$ :: $'a$::*real-normed-vector*
**assumes** $f \longrightarrow c$
**shows** $(\lambda n.\ f\ n - f\ (Suc\ n))$ *sums* $(f\ 0 - c)$
⟨*proof*⟩

**lemma** *telescope-summable*:
**fixes** $c$ :: $'a$::*real-normed-vector*
**assumes** $f \longrightarrow c$
**shows** *summable* $(\lambda n.\ f\ (Suc\ n) - f\ n)$
⟨*proof*⟩

**lemma** *telescope-summable'*:
**fixes** $c$ :: $'a$::*real-normed-vector*
**assumes** $f \longrightarrow c$
**shows** *summable* $(\lambda n.\ f\ n - f\ (Suc\ n))$
⟨*proof*⟩

## 109.9 Infinite summability on Banach spaces

Cauchy-type criterion for convergence of series (c.f. Harrison).

**lemma** *summable-Cauchy*: *summable* $f \longleftrightarrow (\forall e>0.\ \exists N.\ \forall m\geq N.\ \forall n.\ norm\ (sum\ f\ \{m..<n\}) < e)$ (**is** - = *?rhs*)
**for** $f$ :: $nat \Rightarrow 'a$::*banach*
⟨*proof*⟩

**lemma** *summable-Cauchy'*:
**fixes** $f$ :: $nat \Rightarrow 'a$ :: *banach*
**assumes** *ev*: *eventually* $(\lambda m.\ \forall n\geq m.\ norm\ (sum\ f\ \{m..<n\}) \leq g\ m)$ *sequentially*
**assumes** *g0*: $g \longrightarrow 0$
**shows** *summable* $f$
⟨*proof*⟩

**context**
**fixes** $f$ :: $nat \Rightarrow 'a$::*banach*
**begin**

Absolute convergence imples normal convergence.

**lemma** *summable-norm-cancel*: *summable* $(\lambda n.\ norm\ (f\ n)) \Longrightarrow$ *summable* $f$
⟨*proof*⟩

**lemma** *summable-norm*: *summable* $(\lambda n.\ norm\ (f\ n)) \Longrightarrow norm\ (suminf\ f) \leq (\sum n.\ norm\ (f\ n))$
⟨*proof*⟩

Comparison tests.

**lemma** *summable-comparison-test*:
  **assumes** *fg*: $\exists N.\ \forall n \geq N.\ norm\ (f\ n) \leq g\ n$ **and** *g*: *summable g*
  **shows** *summable f*
⟨*proof*⟩

**lemma** *summable-comparison-test-ev*:
  $eventually\ (\lambda n.\ norm\ (f\ n) \leq g\ n)\ sequentially \Longrightarrow summable\ g \Longrightarrow summable\ f$
  ⟨*proof*⟩

A better argument order.

**lemma** *summable-comparison-test'*: $summable\ g \Longrightarrow (\bigwedge n.\ n \geq N \Longrightarrow norm\ (f\ n) \leq g\ n) \Longrightarrow summable\ f$
  ⟨*proof*⟩

## 109.10 The Ratio Test

**lemma** *summable-ratio-test*:
  **assumes** $c < 1\ \bigwedge n.\ n \geq N \Longrightarrow norm\ (f\ (Suc\ n)) \leq c * norm\ (f\ n)$
  **shows** *summable f*
⟨*proof*⟩

**end**

Application to convergence of the log function

**lemma** *norm-summable-ln-series*:
  **fixes** $z$ :: $'a$ :: {*real-normed-field*, *banach*}
  **assumes** $norm\ z < 1$
  **shows** $summable\ (\lambda n.\ norm\ (z\ \hat{}\ n\ /\ of\text{-}nat\ n))$
⟨*proof*⟩

Relations among convergence and absolute convergence for power series.

**lemma** *Abel-lemma*:
  **fixes** $a$ :: $nat \Rightarrow\ 'a$::*real-normed-vector*
  **assumes** $r$: $0 \leq r$
    **and** $r0$: $r < r0$
    **and** $M$: $\bigwedge n.\ norm\ (a\ n) * r0\hat{}n \leq M$
  **shows** $summable\ (\lambda n.\ norm\ (a\ n) * r\hat{}n)$
⟨*proof*⟩

Summability of geometric series for real algebras.

**lemma** *complete-algebra-summable-geometric*:
  **fixes** $x$ :: $'a$::{*real-normed-algebra-1*,*banach*}
  **assumes** $norm\ x < 1$
  **shows** $summable\ (\lambda n.\ x\ \hat{}\ n)$
⟨*proof*⟩

## 109.11 Cauchy Product Formula

Proof based on Analysis WebNotes: Chapter 07, Class 41 http://www.math.unl.edu/~webnotes/classes/class41/prp77.htm

**lemma** *Cauchy-product-sums*:
**fixes** $a\ b :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}algebra,banach\}$
**assumes** $a$: $summable\ (\lambda k.\ norm\ (a\ k))$
**and** $b$: $summable\ (\lambda k.\ norm\ (b\ k))$
**shows** $(\lambda k.\ \sum i{\le}k.\ a\ i * b\ (k - i))\ sums\ ((\sum k.\ a\ k) * (\sum k.\ b\ k))$
$\langle proof \rangle$

**lemma** *Cauchy-product*:
**fixes** $a\ b :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}algebra,banach\}$
**assumes** $summable\ (\lambda k.\ norm\ (a\ k))$
**and** $summable\ (\lambda k.\ norm\ (b\ k))$
**shows** $(\sum k.\ a\ k) * (\sum k.\ b\ k) = (\sum k.\ \sum i{\le}k.\ a\ i * b\ (k - i))$
$\langle proof \rangle$

**lemma** *summable-Cauchy-product*:
**fixes** $a\ b :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}algebra,banach\}$
**assumes** $summable\ (\lambda k.\ norm\ (a\ k))$
**and** $summable\ (\lambda k.\ norm\ (b\ k))$
**shows** $summable\ (\lambda k.\ \sum i{\le}k.\ a\ i * b\ (k - i))$
$\langle proof \rangle$

## 109.12 Series on *reals*

**lemma** *summable-norm-comparison-test*:
$\exists N.\ \forall n{\ge}N.\ norm\ (f\ n) \le g\ n \Longrightarrow summable\ g \Longrightarrow summable\ (\lambda n.\ norm\ (f\ n))$
$\langle proof \rangle$

**lemma** *summable-rabs-comparison-test*: $\exists N.\ \forall n{\ge}N.\ |f\ n| \le g\ n \Longrightarrow summable\ g \Longrightarrow summable\ (\lambda n.\ |f\ n|)$
**for** $f :: nat \Rightarrow real$
$\langle proof \rangle$

**lemma** *summable-rabs-cancel*: $summable\ (\lambda n.\ |f\ n|) \Longrightarrow summable\ f$
**for** $f :: nat \Rightarrow real$
$\langle proof \rangle$

**lemma** *summable-rabs*: $summable\ (\lambda n.\ |f\ n|) \Longrightarrow |suminf\ f| \le (\sum n.\ |f\ n|)$
**for** $f :: nat \Rightarrow real$
$\langle proof \rangle$

**lemma** *norm-suminf-le*:
**assumes** $\bigwedge n.\ norm\ (f\ n :: 'a :: banach) \le g\ n\ summable\ g$
**shows** $norm\ (suminf\ f) \le suminf\ g$
$\langle proof \rangle$

**lemma** *norm-sums-le*:
**assumes** *f sums F g sums G*
**assumes** $\bigwedge n.$ *norm* $(f\ n :: 'a :: banach) \leq g\ n$
**shows** *norm* $F \leq G$
⟨*proof*⟩

**lemma** *summable-zero-power* [*simp*]: *summable* $(\lambda n.\ 0\ \hat{}\ n :: 'a::\{comm\text{-}ring\text{-}1, topological\text{-}space\})$
⟨*proof*⟩

**lemma** *summable-zero-power'* [*simp*]: *summable* $(\lambda n.\ f\ n * 0\ \hat{}\ n :: 'a::\{ring\text{-}1, topological\text{-}space\})$
⟨*proof*⟩

**lemma** *summable-power-series*:
**fixes** $z$ :: *real*
**assumes** *le-1*: $\bigwedge i.\ f\ i \leq 1$
**and** *nonneg*: $\bigwedge i.\ 0 \leq f\ i$
**and** $z$: $0 \leq z\ z < 1$
**shows** *summable* $(\lambda i.\ f\ i * z\hat{}i)$
⟨*proof*⟩

**lemma** *summable-0-powser*: *summable* $(\lambda n.\ f\ n * 0\ \hat{}\ n :: 'a::real\text{-}normed\text{-}div\text{-}algebra)$
⟨*proof*⟩

**lemma** *summable-powser-split-head*:
*summable* $(\lambda n.\ f\ (Suc\ n) * z\ \hat{}\ n :: 'a::real\text{-}normed\text{-}div\text{-}algebra) =$ *summable* $(\lambda n.$
$f\ n * z\ \hat{}\ n)$
⟨*proof*⟩

**lemma** *summable-powser-ignore-initial-segment*:
**fixes** $f :: nat \Rightarrow 'a :: real\text{-}normed\text{-}div\text{-}algebra$
**shows** *summable* $(\lambda n.\ f\ (n + m) * z\ \hat{}\ n) \longleftrightarrow$ *summable* $(\lambda n.\ f\ n * z\ \hat{}\ n)$
⟨*proof*⟩

**lemma** *powser-split-head*:
**fixes** $f :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}div\text{-}algebra, banach\}$
**assumes** *summable* $(\lambda n.\ f\ n * z\ \hat{}\ n)$
**shows** *suminf* $(\lambda n.\ f\ n * z\ \hat{}\ n) = f\ 0 +$ *suminf* $(\lambda n.\ f\ (Suc\ n) * z\ \hat{}\ n) * z$
**and** *suminf* $(\lambda n.\ f\ (Suc\ n) * z\ \hat{}\ n) * z =$ *suminf* $(\lambda n.\ f\ n * z\ \hat{}\ n) - f\ 0$
**and** *summable* $(\lambda n.\ f\ (Suc\ n) * z\ \hat{}\ n)$
⟨*proof*⟩

**lemma** *summable-partial-sum-bound*:
**fixes** $f :: nat \Rightarrow 'a :: banach$
**and** $e$ :: *real*
**assumes** *summable*: *summable f*
**and** $e$: $e > 0$
**obtains** $N$ **where** $\bigwedge m\ n.\ m \geq N \Longrightarrow$ *norm* $(\sum k{=}m..n.\ f\ k) < e$
⟨*proof*⟩

**lemma** *powser-sums-if*:
$(\lambda n.\ (\mathit{if}\ n = m\ \mathit{then}\ (1 :: 'a::\{\mathit{ring\text{-}1},\mathit{topological\text{-}space}\})\ \mathit{else}\ 0) * z\hat{\ }n)\ \mathit{sums}\ z\hat{\ }m$
⟨*proof*⟩

**lemma**
**fixes** $f :: \mathit{nat} \Rightarrow \mathit{real}$
**assumes** *summable f*
**and** *inj g*
**and** *pos*: $\bigwedge x.\ 0 \le f\ x$
**shows** *summable-reindex*: $\mathit{summable}\ (f \circ g)$
**and** *suminf-reindex-mono*: $\mathit{suminf}\ (f \circ g) \le \mathit{suminf}\ f$
**and** *suminf-reindex*: $(\bigwedge x.\ x \notin \mathit{range}\ g \Longrightarrow f\ x = 0) \Longrightarrow \mathit{suminf}\ (f \circ g) = \mathit{suminf}\ f$
⟨*proof*⟩

**lemma** *sums-mono-reindex*:
**assumes** *subseq*: *strict-mono g*
**and** *zero*: $\bigwedge n.\ n \notin \mathit{range}\ g \Longrightarrow f\ n = 0$
**shows** $(\lambda n.\ f\ (g\ n))\ \mathit{sums}\ c \longleftrightarrow f\ \mathit{sums}\ c$
⟨*proof*⟩

**lemma** *summable-mono-reindex*:
**assumes** *subseq*: *strict-mono g*
**and** *zero*: $\bigwedge n.\ n \notin \mathit{range}\ g \Longrightarrow f\ n = 0$
**shows** $\mathit{summable}\ (\lambda n.\ f\ (g\ n)) \longleftrightarrow \mathit{summable}\ f$
⟨*proof*⟩

**lemma** *suminf-mono-reindex*:
**fixes** $f :: \mathit{nat} \Rightarrow 'a::\{\mathit{t2\text{-}space},\mathit{comm\text{-}monoid\text{-}add}\}$
**assumes** $\mathit{strict\text{-}mono}\ g\ \bigwedge n.\ n \notin \mathit{range}\ g \Longrightarrow f\ n = 0$
**shows** $\mathit{suminf}\ (\lambda n.\ f\ (g\ n)) = \mathit{suminf}\ f$
⟨*proof*⟩

**lemma** *summable-bounded-partials*:
**fixes** $f :: \mathit{nat} \Rightarrow 'a :: \{\mathit{real\text{-}normed\text{-}vector},\mathit{complete\text{-}space}\}$
**assumes** *bound*: $\mathit{eventually}\ (\lambda x0.\ \forall a \ge x0.\ \forall b > a.\ \mathit{norm}\ (\mathit{sum}\ f\ \{a<..b\}) \le g\ a)$ *sequentially*
**assumes** *g*: $g \longrightarrow 0$
**shows** *summable f* ⟨*proof*⟩

**end**

# 110 Differentiation

**theory** *Deriv*
**imports** *Limits*
**begin**

## 110.1 Frechet derivative

**definition** *has-derivative* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ ('*a* ⇒ '*b*) ⇒ '*a filter* ⇒ *bool* (**infix** ‹(*has'-derivative*)› *50*)
**where** (*f has-derivative f'*) *F* ⟷
*bounded-linear f'* ∧
((λ*y*. ((*f y* − *f* (*Lim F* (λ*x*. *x*))) − *f'* (*y* − *Lim F* (λ*x*. *x*))) $/_R$ *norm* (*y* − *Lim F* (λ*x*. *x*))) ⟶ *0*) *F*

Usually the filter *F* is *at x within s*. (*f has-derivative D*) (*at x within s*) means: *D* is the derivative of function *f* at point *x* within the set *s*. Where *s* is used to express left or right sided derivatives. In most cases *s* is either a variable or *UNIV*.

These are the only cases we'll care about, probably.

**lemma** *has-derivative-within*: (*f has-derivative f'*) (*at x within s*) ⟷
*bounded-linear f'* ∧ ((λ*y*. (*1* / *norm*(*y* − *x*)) $*_R$ (*f y* − (*f x* + *f'* (*y* − *x*)))) ⟶ *0*) (*at x within s*)
⟨*proof*⟩

**lemma** *has-derivative-eq-rhs*: (*f has-derivative f'*) *F* ⟹ *f'* = *g'* ⟹ (*f has-derivative g'*) *F*
⟨*proof*⟩

**definition** *has-field-derivative* :: ('*a*::*real-normed-field* ⇒ '*a*) ⇒ '*a* ⇒ '*a filter* ⇒ *bool*
(**infix** ‹(*has'-field'-derivative*)› *50*)
**where** (*f has-field-derivative D*) *F* ⟷ (*f has-derivative* (∗) *D*) *F*

**lemma** *DERIV-cong*: (*f has-field-derivative X*) *F* ⟹ *X* = *Y* ⟹ (*f has-field-derivative Y*) *F*
⟨*proof*⟩

**definition** *has-vector-derivative* :: (*real* ⇒ '*b*::*real-normed-vector*) ⇒ '*b* ⇒ *real filter* ⇒ *bool*
(**infix** ‹*has'-vector'-derivative*› *50*)
**where** (*f has-vector-derivative f'*) *net* ⟷ (*f has-derivative* (λ*x*. *x* $*_R$ *f'*)) *net*

**lemma** *has-vector-derivative-eq-rhs*:
(*f has-vector-derivative X*) *F* ⟹ *X* = *Y* ⟹ (*f has-vector-derivative Y*) *F*
⟨*proof*⟩

**named-theorems** *derivative-intros structural introduction rules for derivatives*
⟨*ML*⟩

The following syntax is only used as a legacy syntax.

**abbreviation** (*input*)
*FDERIV* :: ('*a*::*real-normed-vector* ⇒ '*b*::*real-normed-vector*) ⇒ '*a* ⇒ ('*a* ⇒ '*b*) ⇒ *bool*

(‹(‹*notation*=‹*mixfix FDERIV*››*FDERIV* (-)/ (-)/ :> (-))› [*1000*, *1000*, *60*] *60*)
**where** *FDERIV f x* :> $f' \equiv$ (*f has-derivative* $f'$) (*at x*)

**lemma** *has-derivative-bounded-linear*: (*f has-derivative* $f'$) *F* $\Longrightarrow$ *bounded-linear* $f'$
⟨*proof*⟩

**lemma** *has-derivative-linear*: (*f has-derivative* $f'$) *F* $\Longrightarrow$ *linear* $f'$
⟨*proof*⟩

**lemma** *has-derivative-ident*[*derivative-intros*, *simp*]: (($\lambda x.\ x$) *has-derivative* ($\lambda x.$ $x$)) *F*
⟨*proof*⟩

**lemma** *has-derivative-id* [*derivative-intros*, *simp*]: (*id has-derivative id*) *F*
⟨*proof*⟩

**lemma** *shift-has-derivative-id*: ((+) *d has-derivative* ($\lambda x.\ x$)) *F*
⟨*proof*⟩

**lemma** *has-derivative-const*[*derivative-intros*, *simp*]: (($\lambda x.\ c$) *has-derivative* ($\lambda x.$ *0*)) *F*
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *bounded-linear*: *bounded-linear f* ⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *has-derivative*:
(*g has-derivative* $g'$) *F* $\Longrightarrow$ (($\lambda x.\ f\ (g\ x)$) *has-derivative* ($\lambda x.\ f\ (g'\ x)$)) *F*
⟨*proof*⟩

**lemma** *has-derivative-bot* [*intro*]: *bounded-linear* $f'$ $\Longrightarrow$ (*f has-derivative* $f'$) *bot*
⟨*proof*⟩

**lemma** *has-field-derivative-bot* [*simp*, *intro*]: (*f has-field-derivative* $f'$) *bot*
⟨*proof*⟩

**lemmas** *has-derivative-scaleR-right* [*derivative-intros*] =
*bounded-linear.has-derivative* [*OF bounded-linear-scaleR-right*]

**lemmas** *has-derivative-scaleR-left* [*derivative-intros*] =
*bounded-linear.has-derivative* [*OF bounded-linear-scaleR-left*]

**lemmas** *has-derivative-mult-right* [*derivative-intros*] =
*bounded-linear.has-derivative* [*OF bounded-linear-mult-right*]

**lemmas** *has-derivative-mult-left* [*derivative-intros*] =
*bounded-linear.has-derivative* [*OF bounded-linear-mult-left*]

**lemmas** *has-derivative-of-real*[*derivative-intros*, *simp*] =
*bounded-linear.has-derivative*[*OF bounded-linear-of-real*]

**lemma** *has-derivative-add*[*simp*, *derivative-intros*]:
**assumes** $f$: $(f$ *has-derivative* $f')$ $F$
**and** $g$: $(g$ *has-derivative* $g')$ $F$
**shows** $((\lambda x.\ f\ x + g\ x)$ *has-derivative* $(\lambda x.\ f'\ x + g'\ x))$ $F$
⟨*proof*⟩

**lemma** *has-derivative-sum*[*simp*, *derivative-intros*]:
$(\bigwedge i.\ i \in I \Longrightarrow (f\ i$ *has-derivative* $f'\ i)\ F) \Longrightarrow$
$((\lambda x.\ \sum i{\in}I.\ f\ i\ x)$ *has-derivative* $(\lambda x.\ \sum i{\in}I.\ f'\ i\ x))$ $F$
⟨*proof*⟩

**lemma** *has-derivative-minus*[*simp*, *derivative-intros*]:
$(f$ *has-derivative* $f')$ $F \Longrightarrow ((\lambda x.\ -\ f\ x)$ *has-derivative* $(\lambda x.\ -\ f'\ x))$ $F$
⟨*proof*⟩

**lemma** *has-derivative-diff*[*simp*, *derivative-intros*]:
$(f$ *has-derivative* $f')$ $F \Longrightarrow (g$ *has-derivative* $g')$ $F \Longrightarrow$
$((\lambda x.\ f\ x - g\ x)$ *has-derivative* $(\lambda x.\ f'\ x - g'\ x))$ $F$
⟨*proof*⟩

**lemma** *has-derivative-at-within*:
$(f$ *has-derivative* $f')$ $(at\ x\ within\ s) \longleftrightarrow$
$(bounded\text{-}linear\ f' \wedge ((\lambda y.\ ((f\ y - f\ x) - f'\ (y - x))\ /_R\ norm\ (y - x)) \longrightarrow$
$0)$ $(at\ x\ within\ s))$
⟨*proof*⟩

**lemma** *has-derivative-iff-norm*:
$(f$ *has-derivative* $f')$ $(at\ x\ within\ s) \longleftrightarrow$
$bounded\text{-}linear\ f' \wedge ((\lambda y.\ norm\ ((f\ y - f\ x) - f'\ (y - x))\ /\ norm\ (y - x))$
$\longrightarrow 0)$ $(at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *has-derivative-at*:
$(f$ *has-derivative* $D)$ $(at\ x) \longleftrightarrow$
$(bounded\text{-}linear\ D \wedge (\lambda h.\ norm\ (f\ (x + h) - f\ x - D\ h)\ /\ norm\ h)\ {-0\rightarrow}\ 0)$
⟨*proof*⟩

**lemma** *field-has-derivative-at*:
**fixes** $x$ :: $'a$::*real-normed-field*
**shows** $(f$ *has-derivative* $(*)\ D)$ $(at\ x) \longleftrightarrow (\lambda h.\ (f\ (x + h) - f\ x)\ /\ h)\ {-0\rightarrow}\ D$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *has-derivative-iff-Ex*:
$(f$ *has-derivative* $f')$ $(at\ x) \longleftrightarrow$
$bounded\text{-}linear\ f' \wedge (\exists\, e.\ (\forall\, h.\ f\ (x{+}h) = f\ x + f'\ h + e\ h) \wedge ((\lambda h.\ norm\ (e\ h)$
$/\ norm\ h) \longrightarrow 0)\ (at\ 0))$
⟨*proof*⟩

**lemma** *has-derivative-at-within-iff-Ex*:
**assumes** $x \in S$ *open* $S$
**shows** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ S) \longleftrightarrow$
$\textit{bounded-linear}\ f' \wedge (\exists\, e.\ (\forall\, h.\ x{+}h \in S \longrightarrow f\ (x{+}h) = f\ x + f'\ h + e\ h) \wedge$
$((\lambda h.\ \textit{norm}\ (e\ h)\ /\ \textit{norm}\ h) \longrightarrow 0)\ (\textit{at}\ 0))$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *has-derivativeI*:
$\textit{bounded-linear}\ f' \Longrightarrow$
$((\lambda y.\ ((f\ y - f\ x) - f'\ (y - x))\ /_R\ \textit{norm}\ (y - x)) \longrightarrow 0)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-derivativeI-sandwich*:
**assumes** *e*: $0 < e$
**and** *bounded*: *bounded-linear* $f'$
**and** *sandwich*: $(\bigwedge y.\ y \in s \Longrightarrow y \neq x \Longrightarrow \textit{dist}\ y\ x < e \Longrightarrow$
$\textit{norm}\ ((f\ y - f\ x) - f'\ (y - x))\ /\ \textit{norm}\ (y - x) \leq H\ y)$
**and** $(H \longrightarrow 0)\ (\textit{at}\ x\ \textit{within}\ s)$
**shows** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-derivative-subset*:
$(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow t \subseteq s \Longrightarrow (f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ t)$
⟨*proof*⟩

**lemma** *has-derivative-within-singleton-iff*:
$(f\ \textit{has-derivative}\ g)\ (\textit{at}\ x\ \textit{within}\ \{x\}) \longleftrightarrow \textit{bounded-linear}\ g$
⟨*proof*⟩

### 110.1.1 Limit transformation for derivatives

**lemma** *has-derivative-transform-within*:
**assumes** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
**and** $0 < d$
**and** $x \in s$
**and** $\bigwedge x'.\ [\![x' \in s;\ \textit{dist}\ x'\ x < d]\!] \Longrightarrow f\ x' = g\ x'$
**shows** $(g\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-derivative-transform-within-open*:
**assumes** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ t)$
**and** *open* $s$
**and** $x \in s$
**and** $\bigwedge x.\ x{\in}s \Longrightarrow f\ x = g\ x$
**shows** $(g\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ t)$

⟨*proof*⟩

**lemma** *has-derivative-transform*:
**assumes** $x \in s \bigwedge x.\ x \in s \Longrightarrow g\ x = f\ x$
**assumes** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
**shows** $(g\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-derivative-transform-eventually*:
**assumes** $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
$(\forall_F\ x'\ \textit{in at}\ x\ \textit{within}\ s.\ f\ x' = g\ x')$
**assumes** $f\ x = g\ x\ x \in s$
**shows** $(g\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-field-derivative-transform-within*:
**assumes** $(f\ \textit{has-field-derivative}\ f')\ (\textit{at}\ a\ \textit{within}\ S)$
**and** $0 < d$
**and** $a \in S$
**and** $\bigwedge x.\ [\![x \in S;\ \textit{dist}\ x\ a < d]\!] \Longrightarrow f\ x = g\ x$
**shows** $(g\ \textit{has-field-derivative}\ f')\ (\textit{at}\ a\ \textit{within}\ S)$
⟨*proof*⟩

**lemma** *has-field-derivative-transform-within-open*:
**assumes** $(f\ \textit{has-field-derivative}\ f')\ (\textit{at}\ a)$
**and** $\textit{open}\ S\ a \in S$
**and** $\bigwedge x.\ x \in S \Longrightarrow f\ x = g\ x$
**shows** $(g\ \textit{has-field-derivative}\ f')\ (\textit{at}\ a)$
⟨*proof*⟩

## 110.2 Continuity

**lemma** *has-derivative-continuous*:
**assumes** $f$: $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
**shows** $\textit{continuous}\ (\textit{at}\ x\ \textit{within}\ s)\ f$
⟨*proof*⟩

## 110.3 Composition

**lemma** *tendsto-at-iff-tendsto-nhds-within*:
$f\ x = y \Longrightarrow (f \longrightarrow y)\ (\textit{at}\ x\ \textit{within}\ s) \longleftrightarrow (f \longrightarrow y)\ (\textit{inf}\ (\textit{nhds}\ x)\ (\textit{principal}\ s))$
⟨*proof*⟩

**lemma** *has-derivative-in-compose*:
**assumes** $f$: $(f\ \textit{has-derivative}\ f')\ (\textit{at}\ x\ \textit{within}\ s)$
**and** $g$: $(g\ \textit{has-derivative}\ g')\ (\textit{at}\ (f\ x)\ \textit{within}\ (f`s))$
**shows** $((\lambda x.\ g\ (f\ x))\ \textit{has-derivative}\ (\lambda x.\ g'\ (f'\ x)))\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *has-derivative-compose*:
$(f$ *has-derivative* $f')$ $(at\ x\ within\ s) \Longrightarrow (g$ *has-derivative* $g')$ $(at\ (f\ x)) \Longrightarrow$
$((\lambda x.\ g\ (f\ x))$ *has-derivative* $(\lambda x.\ g'\ (f'\ x)))$ $(at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *has-derivative-in-compose2*:
**assumes** $\bigwedge x.\ x \in t \Longrightarrow (g$ *has-derivative* $g'\ x)$ $(at\ x\ within\ t)$
**assumes** $f$ ' $s \subseteq t$ $x \in s$
**assumes** $(f$ *has-derivative* $f')$ $(at\ x\ within\ s)$
**shows** $((\lambda x.\ g\ (f\ x))$ *has-derivative* $(\lambda y.\ g'\ (f\ x)\ (f'\ y)))$ $(at\ x\ within\ s)$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *FDERIV*:
**assumes** $f$: $(f$ *has-derivative* $f')$ $(at\ x\ within\ s)$ **and** $g$: $(g$ *has-derivative* $g')$ $(at\ x\ within\ s)$
**shows** $((\lambda x.\ f\ x\ **\ g\ x)$ *has-derivative* $(\lambda h.\ f\ x\ **\ g'\ h\ +\ f'\ h\ **\ g\ x))$ $(at\ x\ within\ s)$
⟨*proof*⟩

**lemmas** *has-derivative-mult*[*simp*, *derivative-intros*] = *bounded-bilinear.FDERIV*[*OF bounded-bilinear-mult*]
**lemmas** *has-derivative-scaleR*[*simp*, *derivative-intros*] = *bounded-bilinear.FDERIV*[*OF bounded-bilinear-scaleR*]

**lemma** *has-derivative-prod*[*simp*, *derivative-intros*]:
**fixes** $f$ :: $'i \Rightarrow\ 'a$::*real-normed-vector* $\Rightarrow\ 'b$::*real-normed-field*
**shows** $(\bigwedge i.\ i \in I \Longrightarrow (f\ i$ *has-derivative* $f'\ i)$ $(at\ x\ within\ S)) \Longrightarrow$
$((\lambda x.\ \prod i{\in}I.\ f\ i\ x)$ *has-derivative* $(\lambda y.\ \sum i{\in}I.\ f'\ i\ y * (\prod j{\in}I - \{i\}.\ f\ j\ x)))$ $(at\ x\ within\ S)$
⟨*proof*⟩

**lemma** *has-derivative-power*[*simp*, *derivative-intros*]:
**fixes** $f$ :: $'a$ :: *real-normed-vector* $\Rightarrow\ 'b$ :: *real-normed-field*
**assumes** $f$: $(f$ *has-derivative* $f')$ $(at\ x\ within\ S)$
**shows** $((\lambda x.\ f\ x\hat{\ }n)$ *has-derivative* $(\lambda y.$ *of-nat* $n * f'\ y * f\ x\hat{\ }(n - 1)))$ $(at\ x\ within\ S)$
⟨*proof*⟩

**lemma** *has-derivative-inverse'*:
**fixes** $x$ :: $'a$::*real-normed-div-algebra*
**assumes** $x$: $x \neq 0$
**shows** (*inverse* *has-derivative* $(\lambda h.\ -$ (*inverse* $x * h *$ *inverse* $x)))$ $(at\ x\ within\ S)$
(**is** (- *has-derivative* ?*f*) -)
⟨*proof*⟩

**lemma** *has-derivative-inverse*[*simp*, *derivative-intros*]:
**fixes** $f$ :: - $\Rightarrow\ 'a$::*real-normed-div-algebra*
**assumes** $x$: $f\ x \neq 0$

**and** *f*: (*f has-derivative f′*) (*at x within S*)
**shows** ((λ*x*. *inverse* (*f x*)) *has-derivative* (λ*h*. − (*inverse* (*f x*) ∗ *f′ h* ∗ *inverse* (*f x*))))
(*at x within S*)
⟨*proof*⟩

**lemma** *has-derivative-divide*[*simp*, *derivative-intros*]:
**fixes** *f* :: - ⇒ ′*a*::*real-normed-div-algebra*
**assumes** *f*: (*f has-derivative f′*) (*at x within S*)
**and** *g*: (*g has-derivative g′*) (*at x within S*)
**assumes** *x*: *g x* ≠ *0*
**shows** ((λ*x*. *f x* / *g x*) *has-derivative*
(λ*h*. − *f x* ∗ (*inverse* (*g x*) ∗ *g′ h* ∗ *inverse* (*g x*)) + *f′ h* / *g x*)) (*at x within S*)
⟨*proof*⟩

**lemma** *has-derivative-power-int′*:
**fixes** *x* :: ′*a*::*real-normed-field*
**assumes** *x*: *x* ≠ *0*
**shows** ((λ*x*. *power-int x n*) *has-derivative* (λ*y*. *y* ∗ (*of-int n* ∗ *power-int x* (*n* − *1*)))) (*at x within S*)
⟨*proof*⟩

**lemma** *has-derivative-power-int*[*simp*, *derivative-intros*]:
**fixes** *f* :: - ⇒ ′*a*::*real-normed-field*
**assumes** *x*: *f x* ≠ *0*
**and** *f*: (*f has-derivative f′*) (*at x within S*)
**shows** ((λ*x*. *power-int* (*f x*) *n*) *has-derivative* (λ*h*. *f′ h* ∗ (*of-int n* ∗ *power-int* (*f x*) (*n* − *1*))))
(*at x within S*)
⟨*proof*⟩

Conventional form requires mult-AC laws. Types real and complex only.

**lemma** *has-derivative-divide′*[*derivative-intros*]:
**fixes** *f* :: - ⇒ ′*a*::*real-normed-field*
**assumes** *f*: (*f has-derivative f′*) (*at x within S*)
**and** *g*: (*g has-derivative g′*) (*at x within S*)
**and** *x*: *g x* ≠ *0*
**shows** ((λ*x*. *f x* / *g x*) *has-derivative* (λ*h*. (*f′ h* ∗ *g x* − *f x* ∗ *g′ h*) / (*g x* ∗ *g x*))) (*at x within S*)
⟨*proof*⟩

## 110.4 Uniqueness

This can not generally shown for (*has-derivative*), as we need to approach the point from all directions. There is a proof in *Analysis* for *euclidean-space*.

**lemma** *has-derivative-at2*: (*f has-derivative f′*) (*at x*) ⟷
*bounded-linear f′* ∧ ((λ*y*. (*1* / (*norm*(*y* − *x*))) $*_R$ (*f y* − (*f x* + *f′* (*y* − *x*)))) ⟶ *0*) (*at x*)

⟨*proof*⟩

**lemma** *has-derivative-zero-unique*:
**assumes** $((\lambda x.\ 0)$ *has-derivative* $F)$ $(at\ x)$
**shows** $F = (\lambda h.\ 0)$
⟨*proof*⟩

**lemma** *has-derivative-unique*:
**assumes** $(f$ *has-derivative* $F)$ $(at\ x)$
**and** $(f$ *has-derivative* $F')$ $(at\ x)$
**shows** $F = F'$
⟨*proof*⟩

**lemma** *has-derivative-Uniq*: $\exists_{\leq 1}F.$ $(f$ *has-derivative* $F)$ $(at\ x)$
⟨*proof*⟩

## 110.5 Differentiability predicate

**definition** *differentiable* :: $('a::real\text{-}normed\text{-}vector \Rightarrow 'b::real\text{-}normed\text{-}vector) \Rightarrow 'a$ *filter* $\Rightarrow$ *bool*
(**infix** ‹*differentiable*› *50*)
**where** $f$ *differentiable* $F \longleftrightarrow (\exists D.\ (f$ *has-derivative* $D)\ F)$

**lemma** *differentiable-subset*:
$f$ *differentiable* $(at\ x\ within\ s) \Longrightarrow t \subseteq s \Longrightarrow f$ *differentiable* $(at\ x\ within\ t)$
⟨*proof*⟩

**lemmas** *differentiable-within-subset* = *differentiable-subset*

**lemma** *differentiable-ident* [*simp*, *derivative-intros*]: $(\lambda x.\ x)$ *differentiable* $F$
⟨*proof*⟩

**lemma** *differentiable-const* [*simp*, *derivative-intros*]: $(\lambda z.\ a)$ *differentiable* $F$
⟨*proof*⟩

**lemma** *differentiable-in-compose*:
$f$ *differentiable* $(at\ (g\ x)\ within\ (g`s)) \Longrightarrow g$ *differentiable* $(at\ x\ within\ s) \Longrightarrow$
$(\lambda x.\ f\ (g\ x))$ *differentiable* $(at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *differentiable-compose*:
$f$ *differentiable* $(at\ (g\ x)) \Longrightarrow g$ *differentiable* $(at\ x\ within\ s) \Longrightarrow$
$(\lambda x.\ f\ (g\ x))$ *differentiable* $(at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *differentiable-add* [*simp*, *derivative-intros*]:
$f$ *differentiable* $F \Longrightarrow g$ *differentiable* $F \Longrightarrow (\lambda x.\ f\ x + g\ x)$ *differentiable* $F$
⟨*proof*⟩

**lemma** *differentiable-sum*[*simp*, *derivative-intros*]:
**assumes** *finite s* $\forall a \in s.$ (*f a*) *differentiable net*
**shows** ($\lambda x.$ *sum* ($\lambda a.$ *f a x*) *s*) *differentiable net*
⟨*proof*⟩

**lemma** *differentiable-minus* [*simp*, *derivative-intros*]:
*f differentiable F* $\Longrightarrow$ ($\lambda x.$ $-$ *f x*) *differentiable F*
⟨*proof*⟩

**lemma** *differentiable-diff* [*simp*, *derivative-intros*]:
*f differentiable F* $\Longrightarrow$ *g differentiable F* $\Longrightarrow$ ($\lambda x.$ *f x* $-$ *g x*) *differentiable F*
⟨*proof*⟩

**lemma** *differentiable-mult* [*simp*, *derivative-intros*]:
**fixes** *f g* :: *'a::real-normed-vector* $\Rightarrow$ *'b::real-normed-algebra*
**shows** *f differentiable* (*at x within s*) $\Longrightarrow$ *g differentiable* (*at x within s*) $\Longrightarrow$
($\lambda x.$ *f x* $*$ *g x*) *differentiable* (*at x within s*)
⟨*proof*⟩

**lemma** *differentiable-cmult-left-iff* [*simp*]:
**fixes** *c::'a::real-normed-field*
**shows** ($\lambda t.$ *c* $*$ *q t*) *differentiable at t* $\longleftrightarrow$ *c* = *0* $\vee$ ($\lambda t.$ *q t*) *differentiable at t*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *differentiable-cmult-right-iff* [*simp*]:
**fixes** *c::'a::real-normed-field*
**shows** ($\lambda t.$ *q t* $*$ *c*) *differentiable at t* $\longleftrightarrow$ *c* = *0* $\vee$ ($\lambda t.$ *q t*) *differentiable at t*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *differentiable-inverse* [*simp*, *derivative-intros*]:
**fixes** *f* :: *'a::real-normed-vector* $\Rightarrow$ *'b::real-normed-field*
**shows** *f differentiable* (*at x within s*) $\Longrightarrow$ *f x* $\neq$ *0* $\Longrightarrow$
($\lambda x.$ *inverse* (*f x*)) *differentiable* (*at x within s*)
⟨*proof*⟩

**lemma** *differentiable-divide* [*simp*, *derivative-intros*]:
**fixes** *f g* :: *'a::real-normed-vector* $\Rightarrow$ *'b::real-normed-field*
**shows** *f differentiable* (*at x within s*) $\Longrightarrow$ *g differentiable* (*at x within s*) $\Longrightarrow$
*g x* $\neq$ *0* $\Longrightarrow$ ($\lambda x.$ *f x* / *g x*) *differentiable* (*at x within s*)
⟨*proof*⟩

**lemma** *differentiable-power* [*simp*, *derivative-intros*]:
**fixes** *f g* :: *'a::real-normed-vector* $\Rightarrow$ *'b::real-normed-field*
**shows** *f differentiable* (*at x within s*) $\Longrightarrow$ ($\lambda x.$ *f x* ^ *n*) *differentiable* (*at x within s*)
⟨*proof*⟩

**lemma** *differentiable-power-int* [*simp*, *derivative-intros*]:
**fixes** $f$ :: $'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-field*
**shows** $f$ *differentiable* (*at* $x$ *within* $s$) $\Longrightarrow$ $f\ x \neq 0$ $\Longrightarrow$
($\lambda x$. *power-int* ($f\ x$) $n$) *differentiable* (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *differentiable-scaleR* [*simp*, *derivative-intros*]:
$f$ *differentiable* (*at* $x$ *within* $s$) $\Longrightarrow$ $g$ *differentiable* (*at* $x$ *within* $s$) $\Longrightarrow$
($\lambda x$. $f\ x *_R g\ x$) *differentiable* (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *has-derivative-imp-has-field-derivative*:
($f$ *has-derivative* $D$) $F$ $\Longrightarrow$ ($\bigwedge x$. $x * D' = D\ x$) $\Longrightarrow$ ($f$ *has-field-derivative* $D'$) $F$
⟨*proof*⟩

**lemma** *has-field-derivative-imp-has-derivative*:
($f$ *has-field-derivative* $D$) $F$ $\Longrightarrow$ ($f$ *has-derivative* $(*)$ $D$) $F$
⟨*proof*⟩

**lemma** *DERIV-subset*:
($f$ *has-field-derivative* $f'$) (*at* $x$ *within* $s$) $\Longrightarrow$ $t \subseteq s$ $\Longrightarrow$
($f$ *has-field-derivative* $f'$) (*at* $x$ *within* $t$)
⟨*proof*⟩

**lemma** *has-field-derivative-at-within*:
($f$ *has-field-derivative* $f'$) (*at* $x$) $\Longrightarrow$ ($f$ *has-field-derivative* $f'$) (*at* $x$ *within* $s$)
⟨*proof*⟩

**abbreviation** (*input*)
*DERIV* :: ($'a$::*real-normed-field* $\Rightarrow$ $'a$) $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ $\Rightarrow$ *bool*
(‹(‹*notation*=‹*mixfix DERIV*››*DERIV* (-)/ (-)/ :> (-))› [*1000*, *1000*, *60*] *60*)
**where** *DERIV* $f\ x$ :> $D$ $\equiv$ ($f$ *has-field-derivative* $D$) (*at* $x$)

**abbreviation** *has-real-derivative* :: (*real* $\Rightarrow$ *real*) $\Rightarrow$ *real* $\Rightarrow$ *real filter* $\Rightarrow$ *bool*
(**infix** ‹(*has'-real'-derivative*)› *50*)
**where** ($f$ *has-real-derivative* $D$) $F$ $\equiv$ ($f$ *has-field-derivative* $D$) $F$

**lemma** *real-differentiable-def*:
$f$ *differentiable at* $x$ *within* $s$ $\longleftrightarrow$ ($\exists\,D$. ($f$ *has-real-derivative* $D$) (*at* $x$ *within* $s$))
⟨*proof*⟩

**lemma** *real-differentiableE* [*elim?*]:
**assumes** $f$: $f$ *differentiable* (*at* $x$ *within* $s$)
**obtains** *df* **where** ($f$ *has-real-derivative df*) (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *has-field-derivative-iff*:
($f$ *has-field-derivative* $D$) (*at* $x$ *within* $S$) $\longleftrightarrow$
(($\lambda y$. ($f\ y - f\ x$) / ($y - x$)) $\longrightarrow$ $D$) (*at* $x$ *within* $S$)

⟨*proof*⟩

**lemma** *DERIV-def*: $DERIV\ f\ x :> D \longleftrightarrow (\lambda h.\ (f\ (x + h) - f\ x)\ /\ h)\ {-0\!\to}\ D$
⟨*proof*⟩

**lemma** *has-field-derivative-unique*:
**assumes** $(f\ \textit{has-field-derivative}\ f'1)\ (at\ x\ within\ A)$
**assumes** $(f\ \textit{has-field-derivative}\ f'2)\ (at\ x\ within\ A)$
**assumes** $at\ x\ within\ A \neq bot$
**shows** $f'1 = f'2$
⟨*proof*⟩

due to Christian Pardillo Laursen, replacing a proper epsilon-delta horror

**lemma** *field-derivative-lim-unique*:
**assumes** $f$: $(f\ \textit{has-field-derivative}\ df)\ (at\ z)$
**and** $s$: $s \longrightarrow 0\ \ \bigwedge n.\ s\ n \neq 0$
**and** $a$: $(\lambda n.\ (f\ (z + s\ n) - f\ z)\ /\ s\ n) \longrightarrow a$
**shows** $df = a$
⟨*proof*⟩

**lemma** *mult-commute-abs*: $(\lambda x.\ x * c) = (*)\ c$
**for** $c$ :: $'a$::*ab-semigroup-mult*
⟨*proof*⟩

**lemma** *DERIV-compose-FDERIV*:
**fixes** $f$::*real⇒real*
**assumes** $DERIV\ f\ (g\ x) :> f'$
**assumes** $(g\ \textit{has-derivative}\ g')\ (at\ x\ within\ s)$
**shows** $((\lambda x.\ f\ (g\ x))\ \textit{has-derivative}\ (\lambda x.\ g'\ x * f'))\ (at\ x\ within\ s)$
⟨*proof*⟩

## 110.6 Vector derivative

It's for real derivatives only, and not obviously generalisable to field derivatives

**lemma** *has-real-derivative-iff-has-vector-derivative*:
$(f\ \textit{has-real-derivative}\ y)\ F \longleftrightarrow (f\ \textit{has-vector-derivative}\ y)\ F$
⟨*proof*⟩

**lemma** *has-field-derivative-subset*:
$(f\ \textit{has-field-derivative}\ y)\ (at\ x\ within\ s) \Longrightarrow t \subseteq s \Longrightarrow$
$(f\ \textit{has-field-derivative}\ y)\ (at\ x\ within\ t)$
⟨*proof*⟩

**lemma** *has-vector-derivative-const*[*simp*, *derivative-intros*]: $((\lambda x.\ c)\ \textit{has-vector-derivative}\ 0)\ net$
⟨*proof*⟩

**lemma** *has-vector-derivative-id*[*simp*, *derivative-intros*]: $((\lambda x.\ x)$ *has-vector-derivative* $1)$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-minus*[*derivative-intros*]:
$(f$ *has-vector-derivative* $f')$ *net* $\Longrightarrow$ $((\lambda x.\ -\ f\ x)$ *has-vector-derivative* $(-\ f'))$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-add*[*derivative-intros*]:
$(f$ *has-vector-derivative* $f')$ *net* $\Longrightarrow$ $(g$ *has-vector-derivative* $g')$ *net* $\Longrightarrow$
$((\lambda x.\ f\ x + g\ x)$ *has-vector-derivative* $(f' + g'))$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-sum*[*derivative-intros*]:
$(\bigwedge i.\ i \in I \Longrightarrow (f\ i$ *has-vector-derivative* $f'\ i)$ *net*$)$ $\Longrightarrow$
$((\lambda x.\ \sum i{\in}I.\ f\ i\ x)$ *has-vector-derivative* $(\sum i{\in}I.\ f'\ i))$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-diff*[*derivative-intros*]:
$(f$ *has-vector-derivative* $f')$ *net* $\Longrightarrow$ $(g$ *has-vector-derivative* $g')$ *net* $\Longrightarrow$
$((\lambda x.\ f\ x - g\ x)$ *has-vector-derivative* $(f' - g'))$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-add-const*:
$((\lambda t.\ g\ t + z)$ *has-vector-derivative* $f')$ *net* $=$ $((\lambda t.\ g\ t)$ *has-vector-derivative* $f')$ *net*
⟨*proof*⟩

**lemma** *has-vector-derivative-diff-const*:
$((\lambda t.\ g\ t - z)$ *has-vector-derivative* $f')$ *net* $=$ $((\lambda t.\ g\ t)$ *has-vector-derivative* $f')$ *net*
⟨*proof*⟩

**lemma** (**in** *bounded-linear*) *has-vector-derivative*:
**assumes** $(g$ *has-vector-derivative* $g')$ $F$
**shows** $((\lambda x.\ f\ (g\ x))$ *has-vector-derivative* $f\ g')$ $F$
⟨*proof*⟩

**lemma** (**in** *bounded-bilinear*) *has-vector-derivative*:
**assumes** $(f$ *has-vector-derivative* $f')$ (*at* $x$ *within* $s$)
**and** $(g$ *has-vector-derivative* $g')$ (*at* $x$ *within* $s$)
**shows** $((\lambda x.\ f\ x\ {*}{*}\ g\ x)$ *has-vector-derivative* $(f\ x\ {*}{*}\ g' + f'\ {*}{*}\ g\ x))$ (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *has-vector-derivative-scaleR*[*derivative-intros*]:
$(f$ *has-field-derivative* $f')$ (*at* $x$ *within* $s$) $\Longrightarrow$ $(g$ *has-vector-derivative* $g')$ (*at* $x$ *within* $s$) $\Longrightarrow$
$((\lambda x.\ f\ x *_R g\ x)$ *has-vector-derivative* $(f\ x *_R g' + f' *_R g\ x))$ (*at* $x$ *within* $s$)

⟨*proof*⟩

**lemma** *has-vector-derivative-mult*[*derivative-intros*]:
($f$ *has-vector-derivative* $f'$) (*at* $x$ *within* $s$) $\Longrightarrow$ ($g$ *has-vector-derivative* $g'$) (*at* $x$ *within* $s$) $\Longrightarrow$
(($\lambda x.\ f\ x * g\ x$) *has-vector-derivative* ($f\ x * g' + f' * g\ x$)) (*at* $x$ *within* $s$)
**for** $f\ g$ :: *real* $\Rightarrow$ *'a::real-normed-algebra*
⟨*proof*⟩

**lemma** *has-vector-derivative-of-real*[*derivative-intros*]:
($f$ *has-field-derivative* $D$) $F$ $\Longrightarrow$ (($\lambda x.$ *of-real* ($f\ x$)) *has-vector-derivative* (*of-real* $D$)) $F$
⟨*proof*⟩

**lemma** *has-vector-derivative-real-field*:
($f$ *has-field-derivative* $f'$) (*at* (*of-real* $a$)) $\Longrightarrow$ (($\lambda x.\ f$ (*of-real* $x$)) *has-vector-derivative* $f'$) (*at* $a$ *within* $s$)
⟨*proof*⟩

**lemma** *has-vector-derivative-continuous*:
($f$ *has-vector-derivative* $D$) (*at* $x$ *within* $s$) $\Longrightarrow$ *continuous* (*at* $x$ *within* $s$) $f$
⟨*proof*⟩

**lemma** *continuous-on-vector-derivative*:
($\bigwedge x.\ x \in S \Longrightarrow$ ($f$ *has-vector-derivative* $f'\ x$) (*at* $x$ *within* $S$)) $\Longrightarrow$ *continuous-on* $S\ f$
⟨*proof*⟩

**lemma** *has-vector-derivative-mult-right*[*derivative-intros*]:
**fixes** $a$ :: *'a::real-normed-algebra*
**shows** ($f$ *has-vector-derivative* $x$) $F$ $\Longrightarrow$ (($\lambda x.\ a * f\ x$) *has-vector-derivative* ($a * x$)) $F$
⟨*proof*⟩

**lemma** *has-vector-derivative-mult-left*[*derivative-intros*]:
**fixes** $a$ :: *'a::real-normed-algebra*
**shows** ($f$ *has-vector-derivative* $x$) $F$ $\Longrightarrow$ (($\lambda x.\ f\ x * a$) *has-vector-derivative* ($x * a$)) $F$
⟨*proof*⟩

**lemma** *has-vector-derivative-divide*[*derivative-intros*]:
**fixes** $a$ :: *'a::real-normed-field*
**shows** ($f$ *has-vector-derivative* $x$) $F$ $\Longrightarrow$ (($\lambda x.\ f\ x\ /\ a$) *has-vector-derivative* ($x\ /\ a$)) $F$
⟨*proof*⟩

## 110.7 Derivatives

**lemma** *DERIV-D*: *DERIV* $f\ x$ :> $D \Longrightarrow$ ($\lambda h.$ ($f$ ($x + h$) $-$ $f\ x$) / $h$) $-0\rightarrow$ $D$

⟨*proof*⟩

**lemma** *has-field-derivativeD*:
 $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ S) \Longrightarrow$
 $((\lambda y.\ (f\ y - f\ x)\ /\ (y - x)) \longrightarrow D)\ (\textit{at}\ x\ \textit{within}\ S)$
⟨*proof*⟩

**lemma** *DERIV-const* [*simp*, *derivative-intros*]: $((\lambda x.\ k)\ \textit{has-field-derivative}\ 0)\ F$
⟨*proof*⟩

**lemma** *DERIV-ident* [*simp*, *derivative-intros*]: $((\lambda x.\ x)\ \textit{has-field-derivative}\ 1)\ F$
⟨*proof*⟩

**lemma** *field-differentiable-add*[*derivative-intros*]:
 $(f\ \textit{has-field-derivative}\ f')\ F \Longrightarrow (g\ \textit{has-field-derivative}\ g')\ F \Longrightarrow$
 $((\lambda z.\ f\ z + g\ z)\ \textit{has-field-derivative}\ f' + g')\ F$
⟨*proof*⟩

**corollary** *DERIV-add*:
 $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow (g\ \textit{has-field-derivative}\ E)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
 $((\lambda x.\ f\ x + g\ x)\ \textit{has-field-derivative}\ D + E)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *field-differentiable-minus*[*derivative-intros*]:
 $(f\ \textit{has-field-derivative}\ f')\ F \Longrightarrow ((\lambda z.\ - (f\ z))\ \textit{has-field-derivative}\ -f')\ F$
⟨*proof*⟩

**corollary** *DERIV-minus*:
 $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
 $((\lambda x.\ - f\ x)\ \textit{has-field-derivative}\ -D)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *field-differentiable-diff*[*derivative-intros*]:
 $(f\ \textit{has-field-derivative}\ f')\ F \Longrightarrow$
 $(g\ \textit{has-field-derivative}\ g')\ F \Longrightarrow ((\lambda z.\ f\ z - g\ z)\ \textit{has-field-derivative}\ f' - g')\ F$
⟨*proof*⟩

**corollary** *DERIV-diff*:
 $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
 $(g\ \textit{has-field-derivative}\ E)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
 $((\lambda x.\ f\ x - g\ x)\ \textit{has-field-derivative}\ D - E)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-continuous*: $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow \textit{continuous}\ (\textit{at}\ x\ \textit{within}\ s)\ f$
⟨*proof*⟩

**corollary** *DERIV-isCont*: $\textit{DERIV}\ f\ x :> D \Longrightarrow \textit{isCont}\ f\ x$

⟨*proof*⟩

**lemma** *DERIV-atLeastAtMost-imp-continuous-on*:
**assumes** $\bigwedge x.\ [\![a \leq x;\ x \leq b]\!] \Longrightarrow \exists\, y.\ DERIV\ f\ x :> y$
**shows** *continuous-on* $\{a..b\}\ f$
⟨*proof*⟩

**lemma** *DERIV-continuous-on*:
$(\bigwedge x.\ x \in s \Longrightarrow (f\ \textit{has-field-derivative}\ (D\ x))\ (\textit{at}\ x\ \textit{within}\ s)) \Longrightarrow \textit{continuous-on}\ s\ f$
⟨*proof*⟩

**lemma** *DERIV-mult'*:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow (g\ \textit{has-field-derivative}\ E)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ f\ x * g\ x)\ \textit{has-field-derivative}\ f\ x * E + D * g\ x)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-mult*[*derivative-intros*]:
$(f\ \textit{has-field-derivative}\ Da)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow (g\ \textit{has-field-derivative}\ Db)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ f\ x * g\ x)\ \textit{has-field-derivative}\ Da * g\ x + Db * f\ x)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

Derivative of linear multiplication

**lemma** *DERIV-cmult*:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ c * f\ x)\ \textit{has-field-derivative}\ c * D)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-cmult-right*:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ f\ x * c)\ \textit{has-field-derivative}\ D * c)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-cmult-Id* [*simp*]: $((*)\ c\ \textit{has-field-derivative}\ c)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-cdivide*:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ f\ x\ /\ c)\ \textit{has-field-derivative}\ D\ /\ c)\ (\textit{at}\ x\ \textit{within}\ s)$
⟨*proof*⟩

**lemma** *DERIV-unique*: $DERIV\ f\ x :> D \Longrightarrow DERIV\ f\ x :> E \Longrightarrow D = E$
⟨*proof*⟩

**lemma** *DERIV-Uniq*: $\exists_{\leq 1} D.\ DERIV\ f\ x :> D$
⟨*proof*⟩

**lemma** *DERIV-sum*[*derivative-intros*]:
$(\bigwedge n.\ n \in S \Longrightarrow ((\lambda x.\ f\ x\ n)\ \textit{has-field-derivative}\ (f'\ n))\ F) \Longrightarrow$
$((\lambda x.\ \textit{sum}\ (f\ x)\ S)\ \textit{has-field-derivative}\ \textit{sum}\ f'\ S)\ F$
$\langle \textit{proof} \rangle$

**lemma** *DERIV-inverse'*[*derivative-intros*]:
**assumes** $(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s)$
**and** $f\ x \neq 0$
**shows** $((\lambda x.\ \textit{inverse}\ (f\ x))\ \textit{has-field-derivative}\ -\ (\textit{inverse}\ (f\ x) * D * \textit{inverse}\ (f\ x)))$
$(\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

Power of $-1$

**lemma** *DERIV-inverse*:
$x \neq 0 \Longrightarrow ((\lambda x.\ \textit{inverse}(x))\ \textit{has-field-derivative}\ -\ (\textit{inverse}\ x\ \widehat{}\ \textit{Suc}\ (\textit{Suc}\ 0)))\ (\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

Derivative of inverse

**lemma** *DERIV-inverse-fun*:
$(f\ \textit{has-field-derivative}\ d)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow f\ x \neq 0 \Longrightarrow$
$((\lambda x.\ \textit{inverse}\ (f\ x))\ \textit{has-field-derivative}\ (-\ (d * \textit{inverse}(f\ x\ \widehat{}\ \textit{Suc}\ (\textit{Suc}\ 0)))))\ (\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

Derivative of quotient

**lemma** *DERIV-divide*[*derivative-intros*]:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$(g\ \textit{has-field-derivative}\ E)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow g\ x \neq 0 \Longrightarrow$
$((\lambda x.\ f\ x\ /\ g\ x)\ \textit{has-field-derivative}\ (D * g\ x - f\ x * E)\ /\ (g\ x * g\ x))\ (\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

**lemma** *DERIV-quotient*:
$(f\ \textit{has-field-derivative}\ d)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$(g\ \textit{has-field-derivative}\ e)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow g\ x \neq 0 \Longrightarrow$
$((\lambda y.\ f\ y\ /\ g\ y)\ \textit{has-field-derivative}\ (d * g\ x - (e * f\ x))\ /\ (g\ x\ \widehat{}\ \textit{Suc}\ (\textit{Suc}\ 0)))\ (\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

**lemma** *DERIV-power-Suc*:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$
$((\lambda x.\ f\ x\ \widehat{}\ \textit{Suc}\ n)\ \textit{has-field-derivative}\ (1 + \textit{of-nat}\ n) * (D * f\ x\ \widehat{}\ n))\ (\textit{at}\ x\ \textit{within}\ s)$
$\langle \textit{proof} \rangle$

**lemma** *DERIV-power*[*derivative-intros*]:
$(f\ \textit{has-field-derivative}\ D)\ (\textit{at}\ x\ \textit{within}\ s) \Longrightarrow$

$((\lambda x.\ f\ x\ \hat{}\ n)$ *has-field-derivative of-nat* $n * (D * f\ x\ \hat{}\ (n - Suc\ 0)))$ (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *DERIV-pow*: $((\lambda x.\ x\ \hat{}\ n)$ *has-field-derivative real* $n * (x\ \hat{}\ (n - Suc\ 0)))$ (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *DERIV-power-int* [*derivative-intros*]:
**assumes** [*derivative-intros*]: ($f$ *has-field-derivative* $d$) (*at* $x$ *within* $s$)
**and** $n \geq 0 \lor f\ x \neq 0$
**shows** $((\lambda x.$ *power-int* $(f\ x)\ n)$ *has-field-derivative*
(*of-int* $n *$ *power-int* $(f\ x)\ (n - 1) * d)$) (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *DERIV-chain'*: ($f$ *has-field-derivative* $D$) (*at* $x$ *within* $s$) $\Longrightarrow$ *DERIV* $g$ $(f\ x) :> E \Longrightarrow$
$((\lambda x.\ g\ (f\ x))$ *has-field-derivative* $E * D$) (*at* $x$ *within* $s$)
⟨*proof*⟩

**corollary** *DERIV-chain2*: *DERIV* $f$ $(g\ x) :> Da \Longrightarrow$ ($g$ *has-field-derivative* $Db$) (*at* $x$ *within* $s$) $\Longrightarrow$
$((\lambda x.\ f\ (g\ x))$ *has-field-derivative* $Da * Db$) (*at* $x$ *within* $s$)
⟨*proof*⟩

Derivative of a finite product

**lemma** *has-field-derivative-prod*:
**assumes** $\bigwedge x.\ x \in A \Longrightarrow$ ($f\ x$ *has-field-derivative* $f'\ x$) (*at* $z$)
**shows** $((\lambda u.\ \prod x{\in}A.\ f\ x\ u)$ *has-field-derivative* $(\sum x{\in}A.\ f'\ x * (\prod y{\in}A-\{x\}.\ f\ y\ z)))$ (*at* $z$)
⟨*proof*⟩

**lemma** *has-field-derivative-prod'*:
**assumes** $\bigwedge x.\ x \in A \Longrightarrow f\ x\ z \neq 0$
**assumes** $\bigwedge x.\ x \in A \Longrightarrow$ ($f\ x$ *has-field-derivative* $f'\ x$) (*at* $z$)
**defines** $P \equiv (\lambda A\ u.\ \prod x{\in}A.\ f\ x\ u)$
**shows** ($P\ A$ *has-field-derivative* $(P\ A\ z * (\sum x{\in}A.\ f'\ x\ /\ f\ x\ z)))$ (*at* $z$)
⟨*proof*⟩

Standard version

**lemma** *DERIV-chain*:
*DERIV* $f$ $(g\ x) :> Da \Longrightarrow$ ($g$ *has-field-derivative* $Db$) (*at* $x$ *within* $s$) $\Longrightarrow$
($f \circ g$ *has-field-derivative* $Da * Db$) (*at* $x$ *within* $s$)
⟨*proof*⟩

**lemma** *DERIV-image-chain*:
($f$ *has-field-derivative* $Da$) (*at* $(g\ x)$ *within* $(g$ ` $s)$) $\Longrightarrow$
($g$ *has-field-derivative* $Db$) (*at* $x$ *within* $s$) $\Longrightarrow$
($f \circ g$ *has-field-derivative* $Da * Db$) (*at* $x$ *within* $s$)

⟨*proof*⟩

**lemma** *DERIV-chain-s*:
**assumes** $(\bigwedge x.\ x \in s \Longrightarrow \mathit{DERIV}\ g\ x :> g'(x))$
**and** $\mathit{DERIV}\ f\ x :> f'$
**and** $f\ x \in s$
**shows** $\mathit{DERIV}\ (\lambda x.\ g(f\ x))\ x :> f' * g'(f\ x)$
⟨*proof*⟩

**lemma** *DERIV-chain3*:
**assumes** $(\bigwedge x.\ \mathit{DERIV}\ g\ x :> g'(x))$
**and** $\mathit{DERIV}\ f\ x :> f'$
**shows** $\mathit{DERIV}\ (\lambda x.\ g(f\ x))\ x :> f' * g'(f\ x)$
⟨*proof*⟩

Alternative definition for differentiability

**lemma** *DERIV-LIM-iff*:
**fixes** $f :: 'a::\{\mathit{real\text{-}normed\text{-}vector},\mathit{inverse}\} \Rightarrow 'a$
**shows** $((\lambda h.\ (f\ (a + h) - f\ a)\ /\ h) -0\to D) = ((\lambda x.\ (f\ x - f\ a)\ /\ (x - a)) -a\to D)$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

**lemma** *has-field-derivative-cong-ev*:
**assumes** $x = y$
**and** $*$: $\mathit{eventually}\ (\lambda x.\ x \in S \longrightarrow f\ x = g\ x)\ (\mathit{nhds}\ x)$
**and** $u = v\ S = t\ x \in S$
**shows** $(f\ \mathit{has\text{-}field\text{-}derivative}\ u)\ (\mathit{at}\ x\ \mathit{within}\ S) = (g\ \mathit{has\text{-}field\text{-}derivative}\ v)\ (\mathit{at}\ y\ \mathit{within}\ t)$
⟨*proof*⟩

**lemma** *has-field-derivative-cong-eventually*:
**assumes** $\mathit{eventually}\ (\lambda x.\ f\ x = g\ x)\ (\mathit{at}\ x\ \mathit{within}\ S)\ f\ x = g\ x$
**shows** $(f\ \mathit{has\text{-}field\text{-}derivative}\ u)\ (\mathit{at}\ x\ \mathit{within}\ S) = (g\ \mathit{has\text{-}field\text{-}derivative}\ u)\ (\mathit{at}\ x\ \mathit{within}\ S)$
⟨*proof*⟩

**lemma** *DERIV-cong-ev*:
$x = y \Longrightarrow \mathit{eventually}\ (\lambda x.\ f\ x = g\ x)\ (\mathit{nhds}\ x) \Longrightarrow u = v \Longrightarrow$
$\mathit{DERIV}\ f\ x :> u \longleftrightarrow \mathit{DERIV}\ g\ y :> v$
⟨*proof*⟩

**lemma** *DERIV-mirror*: $(\mathit{DERIV}\ f\ (-\ x) :> y) \longleftrightarrow (\mathit{DERIV}\ (\lambda x.\ f\ (-\ x))\ x :> -\ y)$
**for** $f :: \mathit{real} \Rightarrow \mathit{real}$ **and** $x\ y :: \mathit{real}$
⟨*proof*⟩

**lemma** *DERIV-shift*:
$(f\ \mathit{has\text{-}field\text{-}derivative}\ y)\ (\mathit{at}\ (x + z)) = ((\lambda x.\ f\ (x + z))\ \mathit{has\text{-}field\text{-}derivative}\ y)$

$(at\ x)$
⟨proof⟩

**lemma** *DERIV-at-within-shift-lemma*:
**assumes** $(f\ has\text{-}field\text{-}derivative\ y)\ (at\ (z+x)\ within\ (+)\ z\ `\ S)$
**shows** $(f \circ (+)z\ has\text{-}field\text{-}derivative\ y)\ (at\ x\ within\ S)$
⟨proof⟩

**lemma** *DERIV-at-within-shift*:
$(f\ has\text{-}field\text{-}derivative\ y)\ (at\ (z+x)\ within\ (+)\ z\ `\ S) \longleftrightarrow$
$((\lambda x.\ f\ (z+x))\ has\text{-}field\text{-}derivative\ y)\ (at\ x\ within\ S)$ (**is** $?lhs = ?rhs$)
⟨proof⟩

**lemma** *floor-has-real-derivative*:
**fixes** $f :: real \Rightarrow 'a::\{floor\text{-}ceiling,order\text{-}topology\}$
**assumes** $isCont\ f\ x$
**and** $f\ x \notin \mathbb{Z}$
**shows** $((\lambda x.\ floor\ (f\ x))\ has\text{-}real\text{-}derivative\ 0)\ (at\ x)$
⟨proof⟩

**lemmas** *has-derivative-floor*[*derivative-intros*] =
*floor-has-real-derivative*[*THEN DERIV-compose-FDERIV*]

**lemma** *continuous-floor*:
**fixes** $x::real$
**shows** $x \notin \mathbb{Z} \Longrightarrow continuous\ (at\ x)\ (real\text{-}of\text{-}int \circ floor)$
⟨proof⟩

**lemma** *continuous-frac*:
**fixes** $x::real$
**assumes** $x \notin \mathbb{Z}$
**shows** $continuous\ (at\ x)\ frac$
⟨proof⟩

Caratheodory formulation of derivative at a point

**lemma** *CARAT-DERIV*:
$(DERIV\ f\ x :> l) \longleftrightarrow (\exists\ g.\ (\forall\ z.\ f\ z - f\ x = g\ z * (z - x)) \land isCont\ g\ x \land g\ x = l)$
(**is** $?lhs = ?rhs$)
⟨proof⟩

## 110.8 Local extrema

If $0 < f'\ x$ then $x$ is Locally Strictly Increasing At The Right.

**lemma** *has-real-derivative-pos-inc-right*:
**fixes** $f :: real \Rightarrow real$
**assumes** *der*: $(f\ has\text{-}real\text{-}derivative\ l)\ (at\ x\ within\ S)$
**and** $l$: $0 < l$
**shows** $\exists\ d > 0.\ \forall\ h > 0.\ x + h \in S \longrightarrow h < d \longrightarrow f\ x < f\ (x + h)$

⟨*proof*⟩

**lemma** *DERIV-pos-inc-right*:
**fixes** $f :: real \Rightarrow real$
**assumes** *der*: $DERIV\ f\ x :> l$
**and** $l$: $0 < l$
**shows** $\exists\, d > 0.\ \forall\, h > 0.\ h < d \longrightarrow f\ x < f\ (x + h)$
⟨*proof*⟩

**lemma** *has-real-derivative-neg-dec-left*:
**fixes** $f :: real \Rightarrow real$
**assumes** *der*: $(f\ has\text{-}real\text{-}derivative\ l)\ (at\ x\ within\ S)$
**and** $l < 0$
**shows** $\exists\, d > 0.\ \forall\, h > 0.\ x - h \in S \longrightarrow h < d \longrightarrow f\ x < f\ (x - h)$
⟨*proof*⟩

**lemma** *DERIV-neg-dec-left*:
**fixes** $f :: real \Rightarrow real$
**assumes** *der*: $DERIV\ f\ x :> l$
**and** $l$: $l < 0$
**shows** $\exists\, d > 0.\ \forall\, h > 0.\ h < d \longrightarrow f\ x < f\ (x - h)$
⟨*proof*⟩

**lemma** *has-real-derivative-pos-inc-left*:
**fixes** $f :: real \Rightarrow real$
**shows** $(f\ has\text{-}real\text{-}derivative\ l)\ (at\ x\ within\ S) \Longrightarrow 0 < l \Longrightarrow$
$\exists\, d>0.\ \forall\, h>0.\ x - h \in S \longrightarrow h < d \longrightarrow f\ (x - h) < f\ x$
⟨*proof*⟩

**lemma** *DERIV-pos-inc-left*:
**fixes** $f :: real \Rightarrow real$
**shows** $DERIV\ f\ x :> l \Longrightarrow 0 < l \Longrightarrow \exists\, d > 0.\ \forall\, h > 0.\ h < d \longrightarrow f\ (x - h)$
$< f\ x$
⟨*proof*⟩

**lemma** *has-real-derivative-neg-dec-right*:
**fixes** $f :: real \Rightarrow real$
**shows** $(f\ has\text{-}real\text{-}derivative\ l)\ (at\ x\ within\ S) \Longrightarrow l < 0 \Longrightarrow$
$\exists\, d > 0.\ \forall\, h > 0.\ x + h \in S \longrightarrow h < d \longrightarrow f\ x > f\ (x + h)$
⟨*proof*⟩

**lemma** *DERIV-neg-dec-right*:
**fixes** $f :: real \Rightarrow real$
**shows** $DERIV\ f\ x :> l \Longrightarrow l < 0 \Longrightarrow \exists\, d > 0.\ \forall\, h > 0.\ h < d \longrightarrow f\ x > f\ (x$
$+ h)$
⟨*proof*⟩

**lemma** *DERIV-local-max*:
**fixes** $f :: real \Rightarrow real$

**assumes** *der*: $DERIV\ f\ x :> l$
**and** *d*: $0 < d$
**and** *le*: $\forall y.\ |x - y| < d \longrightarrow f\ y \le f\ x$
**shows** $l = 0$
⟨*proof*⟩

Similar theorem for a local minimum

**lemma** *DERIV-local-min*:
**fixes** $f :: real \Rightarrow real$
**shows** $DERIV\ f\ x :> l \Longrightarrow 0 < d \Longrightarrow \forall y.\ |x - y| < d \longrightarrow f\ x \le f\ y \Longrightarrow l = 0$
⟨*proof*⟩

In particular, if a function is locally flat

**lemma** *DERIV-local-const*:
**fixes** $f :: real \Rightarrow real$
**shows** $DERIV\ f\ x :> l \Longrightarrow 0 < d \Longrightarrow \forall y.\ |x - y| < d \longrightarrow f\ x = f\ y \Longrightarrow l = 0$
⟨*proof*⟩

## 110.9 Rolle’s Theorem

Lemma about introducing open ball in open interval

**lemma** *lemma-interval-lt*:
**fixes** $a\ b\ x :: real$
**assumes** $a < x\ x < b$
**shows** $\exists d.\ 0 < d \wedge (\forall y.\ |x - y| < d \longrightarrow a < y \wedge y < b)$
⟨*proof*⟩

**lemma** *lemma-interval*: $a < x \Longrightarrow x < b \Longrightarrow \exists d.\ 0 < d \wedge (\forall y.\ |x - y| < d \longrightarrow a \le y \wedge y \le b)$
**for** $a\ b\ x :: real$
⟨*proof*⟩

Rolle’s Theorem. If $f$ is defined and continuous on the closed interval $[a,b]$ and differentiable on the open interval $(a,b)$, and $f\ a = f\ b$, then there exists $x0 \in (a,b)$ such that $f'\ x0 = 0$

**theorem** *Rolle-deriv*:
**fixes** $f :: real \Rightarrow real$
**assumes** $a < b$
**and** *fab*: $f\ a = f\ b$
**and** *contf*: $continuous\text{-}on\ \{a..b\}\ f$
**and** *derf*: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow (f\ has\text{-}derivative\ f'\ x)\ (at\ x)$
**shows** $\exists z.\ a < z \wedge z < b \wedge f'\ z = (\lambda v.\ 0)$
⟨*proof*⟩

**corollary** *Rolle*:
**fixes** $a\ b :: real$
**assumes** *ab*: $a < b\ f\ a = f\ b\ continuous\text{-}on\ \{a..b\}\ f$
**and** *dif* [*rule-format*]: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow f\ differentiable\ (at\ x)$

**shows** $\exists z.\ a < z \wedge z < b \wedge DERIV\ f\ z :> 0$
$\langle proof \rangle$

## 110.10 Mean Value Theorem

**theorem** *mvt*:
**fixes** $f :: real \Rightarrow real$
**assumes** $a < b$
**and** *contf*: *continuous-on* $\{a..b\}\ f$
**and** *derf*: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow (f\ has\text{-}derivative\ f'\ x)\ (at\ x)$
**obtains** $\xi$ **where** $a < \xi\ \xi < b\ f\ b - f\ a = (f'\ \xi)\ (b - a)$
$\langle proof \rangle$

**theorem** *MVT*:
**fixes** $a\ b :: real$
**assumes** *lt*: $a < b$
**and** *contf*: *continuous-on* $\{a..b\}\ f$
**and** *dif*: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow f\ differentiable\ (at\ x)$
**shows** $\exists l\ z.\ a < z \wedge z < b \wedge DERIV\ f\ z :> l \wedge f\ b - f\ a = (b - a) * l$
$\langle proof \rangle$

**corollary** *MVT2*:
**assumes** $a < b$ **and** *der*: $\bigwedge x.\ [\![a \le x;\ x \le b]\!] \Longrightarrow DERIV\ f\ x :> f'\ x$
**shows** $\exists z{::}real.\ a < z \wedge z < b \wedge (f\ b - f\ a = (b - a) * f'\ z)$
$\langle proof \rangle$

### 110.10.1 A function is constant if its derivative is 0 over an interval.

**lemma** *DERIV-isconst-end*:
**fixes** $f :: real \Rightarrow real$
**assumes** $a < b$ **and** *contf*: *continuous-on* $\{a..b\}\ f$
**and** *0*: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow DERIV\ f\ x :> 0$
**shows** $f\ b = f\ a$
$\langle proof \rangle$

**lemma** *DERIV-isconst2*:
**fixes** $f :: real \Rightarrow real$
**assumes** $a < b$ **and** *contf*: *continuous-on* $\{a..b\}\ f$ **and** *derf*: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow DERIV\ f\ x :> 0$
**and** $a \le x\ x \le b$
**shows** $f\ x = f\ a$
$\langle proof \rangle$

**lemma** *DERIV-isconst3*:
**fixes** $a\ b\ x\ y :: real$
**assumes** $a < b$
**and** $x \in \{a <..< b\}$
**and** $y \in \{a <..< b\}$
**and** *derivable*: $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow DERIV\ f\ x :> 0$

**shows** $f\ x = f\ y$
⟨*proof*⟩

**lemma** *DERIV-isconst-all*:
**fixes** $f$ :: *real* ⇒ *real*
**shows** $\forall x.\ DERIV\ f\ x :> 0 \Longrightarrow f\ x = f\ y$
⟨*proof*⟩

**lemma** *DERIV-const-ratio-const*:
**fixes** $f$ :: *real* ⇒ *real*
**assumes** $a \neq b$ **and** *df*: $\bigwedge x.\ DERIV\ f\ x :> k$
**shows** $f\ b - f\ a = (b - a) * k$
⟨*proof*⟩

**lemma** *DERIV-const-ratio-const2*:
**fixes** $f$ :: *real* ⇒ *real*
**assumes** $a \neq b$ **and** *df*: $\bigwedge x.\ DERIV\ f\ x :> k$
**shows** $(f\ b - f\ a) / (b - a) = k$
⟨*proof*⟩

**lemma** *real-average-minus-first* [*simp*]: $(a + b) / 2 - a = (b - a) / 2$
**for** $a\ b$ :: *real*
⟨*proof*⟩

**lemma** *real-average-minus-second* [*simp*]: $(b + a) / 2 - a = (b - a) / 2$
**for** $a\ b$ :: *real*
⟨*proof*⟩

Gallileo's "trick": average velocity = av. of end velocities.

**lemma** *DERIV-const-average*:
**fixes** $v$ :: *real* ⇒ *real*
**and** $a\ b$ :: *real*
**assumes** *neq*: $a \neq b$
**and** *der*: $\bigwedge x.\ DERIV\ v\ x :> k$
**shows** $v\ ((a + b) / 2) = (v\ a + v\ b) / 2$
⟨*proof*⟩

### 110.10.2 A function with positive derivative is increasing

A simple proof using the MVT, by Jeremy Avigad. And variants.

**lemma** *DERIV-pos-imp-increasing-open*:
**fixes** $a\ b$ :: *real*
**and** $f$ :: *real* ⇒ *real*
**assumes** $a < b$
**and** $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow (\exists y.\ DERIV\ f\ x :> y \wedge y > 0)$
**and** *con*: *continuous-on* $\{a..b\}\ f$
**shows** $f\ a < f\ b$
⟨*proof*⟩

**lemma** *DERIV-pos-imp-increasing*:
**fixes** $a\ b :: real$ **and** $f :: real \Rightarrow real$
**assumes** $a < b$
**and** *der*: $\bigwedge x.\ [\![a \leq x;\ x \leq b]\!] \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y > 0$
**shows** $f\ a < f\ b$
⟨*proof*⟩

**lemma** *DERIV-nonneg-imp-nondecreasing*:
**fixes** $a\ b :: real$
**and** $f :: real \Rightarrow real$
**assumes** $a \leq b$
**and** $\bigwedge x.\ [\![a \leq x;\ x \leq b]\!] \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y \geq 0$
**shows** $f\ a \leq f\ b$
⟨*proof*⟩

**lemma** *DERIV-neg-imp-decreasing-open*:
**fixes** $a\ b :: real$
**and** $f :: real \Rightarrow real$
**assumes** $a < b$
**and** $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y < 0$
**and** *con*: *continuous-on* $\{a..b\}\ f$
**shows** $f\ a > f\ b$
⟨*proof*⟩

**lemma** *DERIV-neg-imp-decreasing*:
**fixes** $a\ b :: real$ **and** $f :: real \Rightarrow real$
**assumes** $a < b$
**and** *der*: $\bigwedge x.\ [\![a \leq x;\ x \leq b]\!] \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y < 0$
**shows** $f\ a > f\ b$
⟨*proof*⟩

**lemma** *DERIV-nonpos-imp-nonincreasing*:
**fixes** $a\ b :: real$
**and** $f :: real \Rightarrow real$
**assumes** $a \leq b$
**and** $\bigwedge x.\ [\![a \leq x;\ x \leq b]\!] \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y \leq 0$
**shows** $f\ a \geq f\ b$
⟨*proof*⟩

**lemma** *DERIV-pos-imp-increasing-at-bot*:
**fixes** $f :: real \Rightarrow real$
**assumes** $\bigwedge x.\ x \leq b \Longrightarrow (\exists\, y.\ DERIV\, f\, x :> y \wedge y > 0)$
**and** *lim*: $(f \longrightarrow flim)$ *at-bot*
**shows** $flim < f\ b$
⟨*proof*⟩

**lemma** *DERIV-neg-imp-decreasing-at-top*:
**fixes** $f :: real \Rightarrow real$
**assumes** *der*: $\bigwedge x.\ x \geq b \Longrightarrow \exists\, y.\ DERIV\, f\, x :> y \wedge y < 0$

**and** *lim*: $(f \longrightarrow flim)$ *at-top*
**shows** $flim < f\ b$
⟨*proof*⟩

**proposition** *deriv-nonpos-imp-antimono*:
**assumes** *deriv*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\text{-}real\text{-}derivative\ g'\ x)\ (at\ x)$
**assumes** *nonneg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \leq 0$
**assumes** $a \leq b$
**shows** $g\ b \leq g\ a$
⟨*proof*⟩

**lemma** *DERIV-nonneg-imp-increasing-open*:
**fixes** $a\ b$ :: *real*
**and** $f$ :: *real* ⇒ *real*
**assumes** $a \leq b$
**and** $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow (\exists\, y.\ DERIV\ f\ x :> y \wedge y \geq 0)$
**and** *con*: *continuous-on* $\{a..b\}\ f$
**shows** $f\ a \leq f\ b$
⟨*proof*⟩

**lemma** *DERIV-nonpos-imp-decreasing-open*:
**fixes** $a\ b$ :: *real*
**and** $f$ :: *real* ⇒ *real*
**assumes** $a \leq b$
**and** $\bigwedge x.\ a < x \Longrightarrow x < b \Longrightarrow \exists\, y.\ DERIV\ f\ x :> y \wedge y \leq 0$
**and** *con*: *continuous-on* $\{a..b\}\ f$
**shows** $f\ a \geq f\ b$
⟨*proof*⟩

**proposition** *deriv-nonneg-imp-mono*:
**assumes** $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\text{-}real\text{-}derivative\ g'\ x)\ (at\ x)$
**assumes** $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \geq 0$
**assumes** $a \leq b$
**shows** $g\ a \leq g\ b$
⟨*proof*⟩

Derivative of inverse function

**lemma** *DERIV-inverse-function*:
**fixes** $f\ g$ :: *real* ⇒ *real*
**assumes** *der*: $DERIV\ f\ (g\ x) :> D$
**and** *neq*: $D \neq 0$
**and** *x*: $a < x\ x < b$
**and** *inj*: $\bigwedge y.\ [\![a < y;\ y < b]\!] \Longrightarrow f\ (g\ y) = y$
**and** *cont*: *isCont* $g\ x$
**shows** $DERIV\ g\ x :> inverse\ D$
⟨*proof*⟩

## 110.11 Generalized Mean Value Theorem

**theorem** *GMVT*:
 **fixes** $a\ b$ :: *real*
 **assumes** *alb*: $a < b$
 **and** *fc*: $\forall x.\ a \le x \wedge x \le b \longrightarrow isCont\ f\ x$
 **and** *fd*: $\forall x.\ a < x \wedge x < b \longrightarrow f\ differentiable\ (at\ x)$
 **and** *gc*: $\forall x.\ a \le x \wedge x \le b \longrightarrow isCont\ g\ x$
 **and** *gd*: $\forall x.\ a < x \wedge x < b \longrightarrow g\ differentiable\ (at\ x)$
 **shows** $\exists\, g'c\ f'c\ c.$
 $DERIV\ g\ c :> g'c \wedge DERIV\ f\ c :> f'c \wedge a < c \wedge c < b \wedge (f\ b - f\ a) * g'c =$
$(g\ b - g\ a) * f'c$
⟨*proof*⟩

**lemma** *GMVT'*:
 **fixes** $f\ g$ :: *real* ⇒ *real*
 **assumes** $a < b$
 **and** *isCont-f*: $\bigwedge z.\ a \le z \Longrightarrow z \le b \Longrightarrow isCont\ f\ z$
 **and** *isCont-g*: $\bigwedge z.\ a \le z \Longrightarrow z \le b \Longrightarrow isCont\ g\ z$
 **and** *DERIV-g*: $\bigwedge z.\ a < z \Longrightarrow z < b \Longrightarrow DERIV\ g\ z :> (g'\ z)$
 **and** *DERIV-f*: $\bigwedge z.\ a < z \Longrightarrow z < b \Longrightarrow DERIV\ f\ z :> (f'\ z)$
 **shows** $\exists\, c.\ a < c \wedge c < b \wedge (f\ b - f\ a) * g'\ c = (g\ b - g\ a) * f'\ c$
⟨*proof*⟩

## 110.12 L'Hopitals rule

**lemma** *isCont-If-ge*:
 **fixes** $a$ :: $'a$ :: *linorder-topology*
 **assumes** *continuous* (*at-left* $a$) $g$ **and** $f$: ($f \longrightarrow g\ a$) (*at-right* $a$)
 **shows** *isCont* ($\lambda x.$ *if* $x \le a$ *then* $g\ x$ *else* $f\ x$) $a$ (**is** *isCont ?gf* $a$)
⟨*proof*⟩

**lemma** *lhopital-right-0*:
 **fixes** *f0 g0* :: *real* ⇒ *real*
 **assumes** *f-0*: (*f0* $\longrightarrow$ *0*) (*at-right 0*)
 **and** *g-0*: (*g0* $\longrightarrow$ *0*) (*at-right 0*)
 **and** *ev*:
 *eventually* ($\lambda x.$ *g0* $x \ne 0$) (*at-right 0*)
 *eventually* ($\lambda x.\ g'\ x \ne 0$) (*at-right 0*)
 *eventually* ($\lambda x.$ *DERIV f0* $x :> f'\ x$) (*at-right 0*)
 *eventually* ($\lambda x.$ *DERIV g0* $x :> g'\ x$) (*at-right 0*)
 **and** *lim*: *filterlim* ($\lambda\ x.\ (f'\ x\ /\ g'\ x)$) $F$ (*at-right 0*)
 **shows** *filterlim* ($\lambda\ x.$ *f0* $x$ / *g0* $x$) $F$ (*at-right 0*)
⟨*proof*⟩

**lemma** *lhopital-right*:
 ($f \longrightarrow$ *0*) (*at-right* $x$) $\Longrightarrow$ ($g \longrightarrow$ *0*) (*at-right* $x$) $\Longrightarrow$
 *eventually* ($\lambda x.\ g\ x \ne 0$) (*at-right* $x$) $\Longrightarrow$
 *eventually* ($\lambda x.\ g'\ x \ne 0$) (*at-right* $x$) $\Longrightarrow$
 *eventually* ($\lambda x.$ *DERIV* $f\ x :> f'\ x$) (*at-right* $x$) $\Longrightarrow$

*eventually* ($\lambda x.$ *DERIV* $g\ x$ :> $g'\ x$) (*at-right* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ ($f'\ x$ / $g'\ x$)) $F$ (*at-right* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ $f\ x$ / $g\ x$) $F$ (*at-right* $x$)
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *lhopital-left*:
($f \longrightarrow 0$) (*at-left* $x$) $\Longrightarrow$ ($g \longrightarrow 0$) (*at-left* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ $g\ x \neq 0$) (*at-left* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ $g'\ x \neq 0$) (*at-left* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $f\ x$ :> $f'\ x$) (*at-left* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $g\ x$ :> $g'\ x$) (*at-left* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ ($f'\ x$ / $g'\ x$)) $F$ (*at-left* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ $f\ x$ / $g\ x$) $F$ (*at-left* $x$)
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *lhopital*:
($f \longrightarrow 0$) (*at* $x$) $\Longrightarrow$ ($g \longrightarrow 0$) (*at* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ $g\ x \neq 0$) (*at* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ $g'\ x \neq 0$) (*at* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $f\ x$ :> $f'\ x$) (*at* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $g\ x$ :> $g'\ x$) (*at* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ ($f'\ x$ / $g'\ x$)) $F$ (*at* $x$) $\Longrightarrow$
*filterlim* ($\lambda\ x.$ $f\ x$ / $g\ x$) $F$ (*at* $x$)
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *lhopital-right-0-at-top*:
**fixes** $f\ g$ :: *real* $\Rightarrow$ *real*
**assumes** *g-0*: *LIM* $x$ *at-right* *0*. $g\ x$ :> *at-top*
**and** *ev*:
*eventually* ($\lambda x.$ $g'\ x \neq 0$) (*at-right* *0*)
*eventually* ($\lambda x.$ *DERIV* $f\ x$ :> $f'\ x$) (*at-right* *0*)
*eventually* ($\lambda x.$ *DERIV* $g\ x$ :> $g'\ x$) (*at-right* *0*)
**and** *lim*: (($\lambda\ x.$ ($f'\ x$ / $g'\ x$)) $\longrightarrow x$) (*at-right* *0*)
**shows** (($\lambda\ x.$ $f\ x$ / $g\ x$) $\longrightarrow x$) (*at-right* *0*)
⟨*proof*⟩

**lemma** *lhopital-right-at-top*:
*LIM* $x$ *at-right* $x$. ($g$::*real* $\Rightarrow$ *real*) $x$ :> *at-top* $\Longrightarrow$
*eventually* ($\lambda x.$ $g'\ x \neq 0$) (*at-right* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $f\ x$ :> $f'\ x$) (*at-right* $x$) $\Longrightarrow$
*eventually* ($\lambda x.$ *DERIV* $g\ x$ :> $g'\ x$) (*at-right* $x$) $\Longrightarrow$
(($\lambda\ x.$ ($f'\ x$ / $g'\ x$)) $\longrightarrow y$) (*at-right* $x$) $\Longrightarrow$
(($\lambda\ x.$ $f\ x$ / $g\ x$) $\longrightarrow y$) (*at-right* $x$)
⟨*proof*⟩

**lemma** *lhopital-left-at-top*:
  *LIM x at-left x. g x :> at-top* $\Longrightarrow$
    *eventually* ($\lambda x.$ $g'$ $x \neq 0$) (*at-left x*) $\Longrightarrow$
    *eventually* ($\lambda x.$ *DERIV f x :>* $f'$ $x$) (*at-left x*) $\Longrightarrow$
    *eventually* ($\lambda x.$ *DERIV g x :>* $g'$ $x$) (*at-left x*) $\Longrightarrow$
    (($\lambda$ $x.$ ($f'$ $x$ / $g'$ $x$)) $\longrightarrow$ $y$) (*at-left x*) $\Longrightarrow$
    (($\lambda$ $x.$ $f$ $x$ / $g$ $x$) $\longrightarrow$ $y$) (*at-left x*)
  **for** $x$ :: *real*
  ⟨*proof*⟩

**lemma** *lhopital-at-top*:
  *LIM x at x.* ($g$::*real* $\Rightarrow$ *real*) $x$ :> *at-top* $\Longrightarrow$
    *eventually* ($\lambda x.$ $g'$ $x \neq 0$) (*at x*) $\Longrightarrow$
    *eventually* ($\lambda x.$ *DERIV f x :>* $f'$ $x$) (*at x*) $\Longrightarrow$
    *eventually* ($\lambda x.$ *DERIV g x :>* $g'$ $x$) (*at x*) $\Longrightarrow$
    (($\lambda$ $x.$ ($f'$ $x$ / $g'$ $x$)) $\longrightarrow$ $y$) (*at x*) $\Longrightarrow$
    (($\lambda$ $x.$ $f$ $x$ / $g$ $x$) $\longrightarrow$ $y$) (*at x*)
  ⟨*proof*⟩

**lemma** *lhospital-at-top-at-top*:
  **fixes** *f g* :: *real* $\Rightarrow$ *real*
  **assumes** *g-0*: *LIM x at-top. g x :> at-top*
    **and** $g'$: *eventually* ($\lambda x.$ $g'$ $x \neq 0$) *at-top*
    **and** *Df*: *eventually* ($\lambda x.$ *DERIV f x :>* $f'$ $x$) *at-top*
    **and** *Dg*: *eventually* ($\lambda x.$ *DERIV g x :>* $g'$ $x$) *at-top*
    **and** *lim*: (($\lambda$ $x.$ ($f'$ $x$ / $g'$ $x$)) $\longrightarrow$ $x$) *at-top*
  **shows** (($\lambda$ $x.$ $f$ $x$ / $g$ $x$) $\longrightarrow$ $x$) *at-top*
  ⟨*proof*⟩

**lemma** *lhopital-right-at-top-at-top*:
  **fixes** *f g* :: *real* $\Rightarrow$ *real*
  **assumes** *f-0*: *LIM x at-right a. f x :> at-top*
  **assumes** *g-0*: *LIM x at-right a. g x :> at-top*
    **and** *ev*:
      *eventually* ($\lambda x.$ *DERIV f x :>* $f'$ $x$) (*at-right a*)
      *eventually* ($\lambda x.$ *DERIV g x :>* $g'$ $x$) (*at-right a*)
    **and** *lim*: *filterlim* ($\lambda$ $x.$ ($f'$ $x$ / $g'$ $x$)) *at-top* (*at-right a*)
  **shows** *filterlim* ($\lambda$ $x.$ $f$ $x$ / $g$ $x$) *at-top* (*at-right a*)
⟨*proof*⟩

**lemma** *lhopital-right-at-top-at-bot*:
  **fixes** *f g* :: *real* $\Rightarrow$ *real*
  **assumes** *f-0*: *LIM x at-right a. f x :> at-top*
  **assumes** *g-0*: *LIM x at-right a. g x :> at-bot*
    **and** *ev*:
      *eventually* ($\lambda x.$ *DERIV f x :>* $f'$ $x$) (*at-right a*)
      *eventually* ($\lambda x.$ *DERIV g x :>* $g'$ $x$) (*at-right a*)
    **and** *lim*: *filterlim* ($\lambda$ $x.$ ($f'$ $x$ / $g'$ $x$)) *at-bot* (*at-right a*)
  **shows** *filterlim* ($\lambda$ $x.$ $f$ $x$ / $g$ $x$) *at-bot* (*at-right a*)

⟨*proof*⟩

**lemma** *lhopital-left-at-top-at-top*:
  **fixes** *f g* :: *real* ⇒ *real*
  **assumes** *f-0*: *LIM x at-left a. f x :> at-top*
  **assumes** *g-0*: *LIM x at-left a. g x :> at-top*
    **and** *ev*:
      *eventually* (*λx. DERIV f x :> f′ x*) (*at-left a*)
      *eventually* (*λx. DERIV g x :> g′ x*) (*at-left a*)
    **and** *lim*: *filterlim* (*λ x.* (*f′ x / g′ x*)) *at-top* (*at-left a*)
  **shows** *filterlim* (*λ x. f x / g x*) *at-top* (*at-left a*)
  ⟨*proof*⟩

**lemma** *lhopital-left-at-top-at-bot*:
  **fixes** *f g* :: *real* ⇒ *real*
  **assumes** *f-0*: *LIM x at-left a. f x :> at-top*
  **assumes** *g-0*: *LIM x at-left a. g x :> at-bot*
    **and** *ev*:
      *eventually* (*λx. DERIV f x :> f′ x*) (*at-left a*)
      *eventually* (*λx. DERIV g x :> g′ x*) (*at-left a*)
    **and** *lim*: *filterlim* (*λ x.* (*f′ x / g′ x*)) *at-bot* (*at-left a*)
  **shows** *filterlim* (*λ x. f x / g x*) *at-bot* (*at-left a*)
  ⟨*proof*⟩

**lemma** *lhopital-at-top-at-top*:
  **fixes** *f g* :: *real* ⇒ *real*
  **assumes** *f-0*: *LIM x at a. f x :> at-top*
  **assumes** *g-0*: *LIM x at a. g x :> at-top*
    **and** *ev*:
      *eventually* (*λx. DERIV f x :> f′ x*) (*at a*)
      *eventually* (*λx. DERIV g x :> g′ x*) (*at a*)
    **and** *lim*: *filterlim* (*λ x.* (*f′ x / g′ x*)) *at-top* (*at a*)
  **shows** *filterlim* (*λ x. f x / g x*) *at-top* (*at a*)
  ⟨*proof*⟩

**lemma** *lhopital-at-top-at-bot*:
  **fixes** *f g* :: *real* ⇒ *real*
  **assumes** *f-0*: *LIM x at a. f x :> at-top*
  **assumes** *g-0*: *LIM x at a. g x :> at-bot*
    **and** *ev*:
      *eventually* (*λx. DERIV f x :> f′ x*) (*at a*)
      *eventually* (*λx. DERIV g x :> g′ x*) (*at a*)
    **and** *lim*: *filterlim* (*λ x.* (*f′ x / g′ x*)) *at-bot* (*at a*)
  **shows** *filterlim* (*λ x. f x / g x*) *at-bot* (*at a*)
  ⟨*proof*⟩

**end**

# 111 Nth Roots of Real Numbers

**theory** *NthRoot*
**imports** *Deriv*
**begin**

## 111.1 Existence of Nth Root

Existence follows from the Intermediate Value Theorem

**lemma** *realpow-pos-nth*:
**fixes** $a$ :: *real*
**assumes** $n$: $0 < n$
**and** $a$: $0 < a$
**shows** $\exists r>0.\ r \,\hat{}\, n = a$
⟨*proof*⟩

**lemma** *realpow-pos-nth2*: $(0::real) < a \Longrightarrow \exists r>0.\ r \,\hat{}\, Suc\ n = a$
⟨*proof*⟩

Uniqueness of nth positive root.

**lemma** *realpow-pos-nth-unique*: $0 < n \Longrightarrow 0 < a \Longrightarrow \exists! r.\ 0 < r \wedge r \,\hat{}\, n = a$ **for** $a$ :: *real*
⟨*proof*⟩

## 111.2 Nth Root

We define roots of negative reals such that *root* $n$ $(- x) = -$ *root* $n$ $x$. This allows us to omit side conditions from many theorems.

**lemma** *inj-sgn-power*:
**assumes** $0 < n$
**shows** *inj* $(\lambda y.\ sgn\ y * |y|\hat{}n :: real)$
(**is** *inj ?f*)
⟨*proof*⟩

**lemma** *sgn-power-injE*:
$sgn\ a * |a| \,\hat{}\, n = x \Longrightarrow x = sgn\ b * |b| \,\hat{}\, n \Longrightarrow 0 < n \Longrightarrow a = b$
**for** $a$ $b$ :: *real*
⟨*proof*⟩

**definition** *root* :: *nat* ⇒ *real* ⇒ *real*
**where** $root\ n\ x = (if\ n = 0\ then\ 0\ else\ the\text{-}inv\ (\lambda y.\ sgn\ y * |y|\hat{}n)\ x)$

**lemma** *root-0* [*simp*]: $root\ 0\ x = 0$
⟨*proof*⟩

**lemma** *root-sgn-power*: $0 < n \Longrightarrow root\ n\ (sgn\ y * |y|\hat{}n) = y$
⟨*proof*⟩

**lemma** *sgn-power-root*:
**assumes** $0 < n$
**shows** $sgn\ (root\ n\ x) * |(root\ n\ x)|\hat{}n = x$
(**is** $?f\ (root\ n\ x) = x$)
⟨*proof*⟩

**lemma** *split-root*: $P\ (root\ n\ x) \longleftrightarrow (n = 0 \longrightarrow P\ 0) \wedge (0 < n \longrightarrow (\forall y.\ sgn\ y * |y|\hat{}n = x \longrightarrow P\ y))$
⟨*proof*⟩

**lemma** *real-root-zero* [*simp*]: $root\ n\ 0 = 0$
⟨*proof*⟩

**lemma** *real-root-minus*: $root\ n\ (-\ x) = -\ root\ n\ x$
⟨*proof*⟩

**lemma** *real-root-less-mono*: $0 < n \Longrightarrow x < y \Longrightarrow root\ n\ x < root\ n\ y$
⟨*proof*⟩

**lemma** *real-root-gt-zero*: $0 < n \Longrightarrow 0 < x \Longrightarrow 0 < root\ n\ x$
⟨*proof*⟩

**lemma** *real-root-ge-zero*: $0 \leq x \Longrightarrow 0 \leq root\ n\ x$
⟨*proof*⟩

**lemma** *real-root-pow-pos*: $0 < n \Longrightarrow 0 < x \Longrightarrow root\ n\ x\ \hat{}\ n = x$
⟨*proof*⟩

**lemma** *real-root-pow-pos2* [*simp*]: $0 < n \Longrightarrow 0 \leq x \Longrightarrow root\ n\ x\ \hat{}\ n = x$
⟨*proof*⟩

**lemma** *sgn-root*: $0 < n \Longrightarrow sgn\ (root\ n\ x) = sgn\ x$
⟨*proof*⟩

**lemma** *odd-real-root-pow*: $odd\ n \Longrightarrow root\ n\ x\ \hat{}\ n = x$
⟨*proof*⟩

**lemma** *real-root-power-cancel*: $0 < n \Longrightarrow 0 \leq x \Longrightarrow root\ n\ (x\ \hat{}\ n) = x$
⟨*proof*⟩

**lemma** *odd-real-root-power-cancel*: $odd\ n \Longrightarrow root\ n\ (x\ \hat{}\ n) = x$
⟨*proof*⟩

**lemma** *real-root-pos-unique*: $0 < n \Longrightarrow 0 \leq y \Longrightarrow y\ \hat{}\ n = x \Longrightarrow root\ n\ x = y$
⟨*proof*⟩

**lemma** *odd-real-root-unique*: $odd\ n \Longrightarrow y\ \hat{}\ n = x \Longrightarrow root\ n\ x = y$
⟨*proof*⟩

**lemma** *real-root-one* [*simp*]: $0 < n \Longrightarrow root\ n\ 1 = 1$
⟨*proof*⟩

Root function is strictly monotonic, hence injective.

**lemma** *real-root-le-mono*: $0 < n \Longrightarrow x \leq y \Longrightarrow root\ n\ x \leq root\ n\ y$
⟨*proof*⟩

**lemma** *real-root-less-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x < root\ n\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *real-root-le-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x \leq root\ n\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *real-root-eq-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x = root\ n\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemmas** *real-root-gt-0-iff* [*simp*] = *real-root-less-iff* [**where** *x=0*, *simplified*]
**lemmas** *real-root-lt-0-iff* [*simp*] = *real-root-less-iff* [**where** *y=0*, *simplified*]
**lemmas** *real-root-ge-0-iff* [*simp*] = *real-root-le-iff* [**where** *x=0*, *simplified*]
**lemmas** *real-root-le-0-iff* [*simp*] = *real-root-le-iff* [**where** *y=0*, *simplified*]
**lemmas** *real-root-eq-0-iff* [*simp*] = *real-root-eq-iff* [**where** *y=0*, *simplified*]

**lemma** *real-root-gt-1-iff* [*simp*]: $0 < n \Longrightarrow 1 < root\ n\ y \longleftrightarrow 1 < y$
⟨*proof*⟩

**lemma** *real-root-lt-1-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x < 1 \longleftrightarrow x < 1$
⟨*proof*⟩

**lemma** *real-root-ge-1-iff* [*simp*]: $0 < n \Longrightarrow 1 \leq root\ n\ y \longleftrightarrow 1 \leq y$
⟨*proof*⟩

**lemma** *real-root-le-1-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x \leq 1 \longleftrightarrow x \leq 1$
⟨*proof*⟩

**lemma** *real-root-eq-1-iff* [*simp*]: $0 < n \Longrightarrow root\ n\ x = 1 \longleftrightarrow x = 1$
⟨*proof*⟩

Roots of multiplication and division.

**lemma** *real-root-mult*: $root\ n\ (x * y) = root\ n\ x * root\ n\ y$
⟨*proof*⟩

**lemma** *real-root-inverse*: $root\ n\ (inverse\ x) = inverse\ (root\ n\ x)$
⟨*proof*⟩

**lemma** *real-root-divide*: $root\ n\ (x\ /\ y) = root\ n\ x\ /\ root\ n\ y$
⟨*proof*⟩

**lemma** *real-root-abs*: $0 < n \Longrightarrow root\ n\ |x| = |root\ n\ x|$

⟨proof⟩

**lemma** *root-abs-power*: $n > 0 \Longrightarrow abs\ (root\ n\ (y\ \widehat{\ }n)) = abs\ y$
⟨proof⟩

**lemma** *real-root-power*: $0 < n \Longrightarrow root\ n\ (x\ \widehat{\ }\ k) = root\ n\ x\ \widehat{\ }\ k$
⟨proof⟩

Roots of roots.

**lemma** *real-root-Suc-0* [*simp*]: $root\ (Suc\ 0)\ x = x$
⟨proof⟩

**lemma** *real-root-mult-exp*: $root\ (m * n)\ x = root\ m\ (root\ n\ x)$
⟨proof⟩

**lemma** *real-root-commute*: $root\ m\ (root\ n\ x) = root\ n\ (root\ m\ x)$
⟨proof⟩

Monotonicity in first argument.

**lemma** *real-root-strict-decreasing*:
**assumes** $0 < n\ n < N\ 1 < x$
**shows** $root\ N\ x < root\ n\ x$
⟨proof⟩

**lemma** *real-root-strict-increasing*:
**assumes** $0 < n\ n < N\ 0 < x\ x < 1$
**shows** $root\ n\ x < root\ N\ x$
⟨proof⟩

**lemma** *real-root-decreasing*: $0 < n \Longrightarrow n \leq N \Longrightarrow 1 \leq x \Longrightarrow root\ N\ x \leq root\ n\ x$
⟨proof⟩

**lemma** *real-root-increasing*: $0 < n \Longrightarrow n \leq N \Longrightarrow 0 \leq x \Longrightarrow x \leq 1 \Longrightarrow root\ n\ x \leq root\ N\ x$
⟨proof⟩

Continuity and derivatives.

**lemma** *isCont-real-root*: $isCont\ (root\ n)\ x$
⟨proof⟩

**lemma** *tendsto-real-root* [*tendsto-intros*]:
$(f \longrightarrow x)\ F \Longrightarrow ((\lambda x.\ root\ n\ (f\ x)) \longrightarrow root\ n\ x)\ F$
⟨proof⟩

**lemma** *continuous-real-root* [*continuous-intros*]:
$continuous\ F\ f \Longrightarrow continuous\ F\ (\lambda x.\ root\ n\ (f\ x))$
⟨proof⟩

**lemma** *continuous-on-real-root* [*continuous-intros*]:

*continuous-on s f* $\Longrightarrow$ *continuous-on s* ($\lambda x.$ *root n* (*f x*))
⟨*proof*⟩

**lemma** *DERIV-real-root*:
**assumes** *n*: $0 < n$
**and** *x*: $0 < x$
**shows** *DERIV* (*root n*) *x* :> *inverse* (*real n* $*$ *root n x* ^ (*n* − *Suc 0*))
⟨*proof*⟩

**lemma** *DERIV-odd-real-root*:
**assumes** *n*: *odd n*
**and** *x*: $x \neq 0$
**shows** *DERIV* (*root n*) *x* :> *inverse* (*real n* $*$ *root n x* ^ (*n* − *Suc 0*))
⟨*proof*⟩

**lemma** *DERIV-even-real-root*:
**assumes** *n*: $0 < n$
**and** *even n*
**and** *x*: $x < 0$
**shows** *DERIV* (*root n*) *x* :> *inverse* (− *real n* $*$ *root n x* ^ (*n* − *Suc 0*))
⟨*proof*⟩

**lemma** *DERIV-real-root-generic*:
**assumes** $0 < n$
**and** $x \neq 0$
**and** *even n* $\Longrightarrow 0 < x \Longrightarrow D =$ *inverse* (*real n* $*$ *root n x* ^ (*n* − *Suc 0*))
**and** *even n* $\Longrightarrow x < 0 \Longrightarrow D = -$ *inverse* (*real n* $*$ *root n x* ^ (*n* − *Suc 0*))
**and** *odd n* $\Longrightarrow D =$ *inverse* (*real n* $*$ *root n x* ^ (*n* − *Suc 0*))
**shows** *DERIV* (*root n*) *x* :> *D*
⟨*proof*⟩

**lemma** *power-tendsto-0-iff* [*simp*]:
**fixes** *f* :: *'a* ⇒ *real*
**assumes** $n > 0$
**shows** (($\lambda x.$ *f x* ^ *n*) ⟶ *0*) *F* ⟷ (*f* ⟶ *0*) *F*
⟨*proof*⟩

## 111.3 Square Root

**definition** *sqrt* :: *real* ⇒ *real*
**where** *sqrt* = *root 2*

**lemma** *pos2*: $0 < (2::nat)$
⟨*proof*⟩

**lemma** *real-sqrt-unique*: $y^2 = x \Longrightarrow 0 \leq y \Longrightarrow$ *sqrt* $x = y$
⟨*proof*⟩

**lemma** *real-sqrt-abs* [*simp*]: *sqrt* $(x^2) = |x|$

⟨proof⟩

**lemma** *real-sqrt-pow2* [*simp*]: $0 \le x \Longrightarrow (sqrt\ x)^2 = x$
⟨proof⟩

**lemma** *real-sqrt-pow2-iff* [*simp*]: $(sqrt\ x)^2 = x \longleftrightarrow 0 \le x$
⟨proof⟩

**lemma** *real-sqrt-zero* [*simp*]: $sqrt\ 0 = 0$
⟨proof⟩

**lemma** *real-sqrt-one* [*simp*]: $sqrt\ 1 = 1$
⟨proof⟩

**lemma** *real-sqrt-four* [*simp*]: $sqrt\ 4 = 2$
⟨proof⟩

**lemma** *real-sqrt-minus*: $sqrt\ (-\ x) = -\ sqrt\ x$
⟨proof⟩

**lemma** *real-sqrt-mult*: $sqrt\ (x * y) = sqrt\ x * sqrt\ y$
⟨proof⟩

**lemma** *real-sqrt-mult-self*[*simp*]: $sqrt\ a * sqrt\ a = |a|$
⟨proof⟩

**lemma** *real-sqrt-inverse*: $sqrt\ (inverse\ x) = inverse\ (sqrt\ x)$
⟨proof⟩

**lemma** *real-sqrt-divide*: $sqrt\ (x\ /\ y) = sqrt\ x\ /\ sqrt\ y$
⟨proof⟩

**lemma** *real-sqrt-power*: $sqrt\ (x\ \widehat{}\ k) = sqrt\ x\ \widehat{}\ k$
⟨proof⟩

**lemma** *real-sqrt-gt-zero*: $0 < x \Longrightarrow 0 < sqrt\ x$
⟨proof⟩

**lemma** *real-sqrt-ge-zero*: $0 \le x \Longrightarrow 0 \le sqrt\ x$
⟨proof⟩

**lemma** *real-sqrt-less-mono*: $x < y \Longrightarrow sqrt\ x < sqrt\ y$
⟨proof⟩

**lemma** *real-sqrt-le-mono*: $x \le y \Longrightarrow sqrt\ x \le sqrt\ y$
⟨proof⟩

**lemma** *real-sqrt-less-iff* [*simp*]: $sqrt\ x < sqrt\ y \longleftrightarrow x < y$
⟨proof⟩

**lemma** *real-sqrt-le-iff* [*simp*]: $sqrt\ x \leq sqrt\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *real-sqrt-eq-iff* [*simp*]: $sqrt\ x = sqrt\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *real-less-lsqrt*: $0 \leq y \Longrightarrow x < y^2 \Longrightarrow sqrt\ x < y$
⟨*proof*⟩

**lemma** *real-le-lsqrt*: $0 \leq y \Longrightarrow x \leq y^2 \Longrightarrow sqrt\ x \leq y$
⟨*proof*⟩

**lemma** *real-le-rsqrt*: $x^2 \leq y \Longrightarrow x \leq sqrt\ y$
⟨*proof*⟩

**lemma** *real-less-rsqrt*: $x^2 < y \Longrightarrow x < sqrt\ y$
⟨*proof*⟩

**lemma** *real-sqrt-power-even*:
**assumes** *even n* $x \geq 0$
**shows** $sqrt\ x\ \hat{}\ n = x\ \hat{}\ (n\ div\ 2)$
⟨*proof*⟩

**lemma** *sqrt-le-D*: $sqrt\ x \leq y \Longrightarrow x \leq y^2$
⟨*proof*⟩

**lemma** *sqrt-ge-absD*: $|x| \leq sqrt\ y \Longrightarrow x^2 \leq y$
⟨*proof*⟩

**lemma** *sqrt-even-pow2*:
**assumes** *n*: *even n*
**shows** $sqrt\ (2\ \hat{}\ n) = 2\ \hat{}\ (n\ div\ 2)$
⟨*proof*⟩

**lemmas** *real-sqrt-gt-0-iff* [*simp*] = *real-sqrt-less-iff* [**where** $x=0$, *unfolded real-sqrt-zero*]
**lemmas** *real-sqrt-lt-0-iff* [*simp*] = *real-sqrt-less-iff* [**where** $y=0$, *unfolded real-sqrt-zero*]
**lemmas** *real-sqrt-ge-0-iff* [*simp*] = *real-sqrt-le-iff* [**where** $x=0$, *unfolded real-sqrt-zero*]
**lemmas** *real-sqrt-le-0-iff* [*simp*] = *real-sqrt-le-iff* [**where** $y=0$, *unfolded real-sqrt-zero*]
**lemmas** *real-sqrt-eq-0-iff* [*simp*] = *real-sqrt-eq-iff* [**where** $y=0$, *unfolded real-sqrt-zero*]

**lemmas** *real-sqrt-gt-1-iff* [*simp*] = *real-sqrt-less-iff* [**where** $x=1$, *unfolded real-sqrt-one*]
**lemmas** *real-sqrt-lt-1-iff* [*simp*] = *real-sqrt-less-iff* [**where** $y=1$, *unfolded real-sqrt-one*]
**lemmas** *real-sqrt-ge-1-iff* [*simp*] = *real-sqrt-le-iff* [**where** $x=1$, *unfolded real-sqrt-one*]
**lemmas** *real-sqrt-le-1-iff* [*simp*] = *real-sqrt-le-iff* [**where** $y=1$, *unfolded real-sqrt-one*]
**lemmas** *real-sqrt-eq-1-iff* [*simp*] = *real-sqrt-eq-iff* [**where** $y=1$, *unfolded real-sqrt-one*]

**lemma** *sqrt-add-le-add-sqrt*:
**assumes** $0 \leq x$ $0 \leq y$

**shows** *sqrt* $(x + y) \leq$ *sqrt* $x$ + *sqrt* $y$
⟨*proof*⟩

**lemma** *isCont-real-sqrt*: *isCont sqrt x*
⟨*proof*⟩

**lemma** *tendsto-real-sqrt* [*tendsto-intros*]:
$(f \longrightarrow x)\ F \Longrightarrow ((\lambda x.\ sqrt\ (f\ x)) \longrightarrow sqrt\ x)\ F$
⟨*proof*⟩

**lemma** *continuous-real-sqrt* [*continuous-intros*]:
*continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ sqrt\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-real-sqrt* [*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ sqrt\ (f\ x))$
⟨*proof*⟩

**lemma** *DERIV-real-sqrt-generic*:
**assumes** $x \neq 0$
**and** $x > 0 \Longrightarrow D = inverse\ (sqrt\ x)\ /\ 2$
**and** $x < 0 \Longrightarrow D = -\ inverse\ (sqrt\ x)\ /\ 2$
**shows** *DERIV sqrt x* :> *D*
⟨*proof*⟩

**lemma** *DERIV-real-sqrt*: $0 < x \Longrightarrow$ *DERIV sqrt x* :> *inverse* (*sqrt x*) / *2*
⟨*proof*⟩

**declare**
*DERIV-real-sqrt-generic*[*THEN DERIV-chain2*, *derivative-intros*]
*DERIV-real-root-generic*[*THEN DERIV-chain2*, *derivative-intros*]

**lemmas** *has-derivative-real-sqrt*[*derivative-intros*] = *DERIV-real-sqrt*[*THEN DERIV-compose-FDERIV*]

**lemma** *not-real-square-gt-zero* [*simp*]: $\neg\ 0 < x * x \longleftrightarrow x = 0$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *real-sqrt-abs2* [*simp*]: *sqrt* $(x * x) = |x|$
⟨*proof*⟩

**lemma** *real-sqrt-abs'*: *sqrt* $|x| = |sqrt\ x|$
⟨*proof*⟩

**lemma** *real-inv-sqrt-pow2*: $0 < x \Longrightarrow (inverse\ (sqrt\ x))^2 = inverse\ x$
⟨*proof*⟩

**lemma** *real-sqrt-eq-zero-cancel*: $0 \leq x \Longrightarrow sqrt\ x = 0 \Longrightarrow x = 0$

⟨*proof*⟩

**lemma** *real-sqrt-ge-one*: $1 \leq x \Longrightarrow 1 \leq sqrt\ x$
⟨*proof*⟩

**lemma** *sqrt-divide-self-eq*:
**assumes** *nneg*: $0 \leq x$
**shows** $sqrt\ x\ /\ x = inverse\ (sqrt\ x)$
⟨*proof*⟩

**lemma** *real-div-sqrt*: $0 \leq x \Longrightarrow x\ /\ sqrt\ x = sqrt\ x$
⟨*proof*⟩

**lemma** *real-divide-square-eq* [*simp*]: $(r * a)\ /\ (r * r) = a\ /\ r$
**for** *a r* :: *real*
⟨*proof*⟩

**lemma** *lemma-real-divide-sqrt-less*: $0 < u \Longrightarrow u\ /\ sqrt\ 2 < u$
⟨*proof*⟩

**lemma** *four-x-squared*: $4 * x^2 = (2 * x)^2$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *sqrt-at-top*: *LIM x at-top. sqrt x* :: *real* :> *at-top*
⟨*proof*⟩

## 111.4 Square Root of Sum of Squares

**lemma** *sum-squares-bound*: $2 * x * y \leq x^2 + y^2$
**for** *x y* :: *'a*::*linordered-field*
⟨*proof*⟩

**lemma** *arith-geo-mean*:
**fixes** *u* :: *'a*::*linordered-field*
**assumes** $u^2 = x * y$ $x \geq 0$ $y \geq 0$
**shows** $u \leq (x + y)/2$
⟨*proof*⟩

**lemma** *arith-geo-mean-sqrt*:
**fixes** *x* :: *real*
**assumes** $x \geq 0$ $y \geq 0$
**shows** $sqrt\ (x * y) \leq (x + y)/2$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-mult-ge-zero* [*simp*]: $0 \leq sqrt\ ((x^2 + y^2) * (xa^2 + ya^2))$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-mult-squared-eq* [*simp*]:
$(sqrt\ ((x^2 + y^2) * (xa^2 + ya^2)))^2 = (x^2 + y^2) * (xa^2 + ya^2)$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-eq-cancel*: $sqrt\ (x^2 + y^2) = x \Longrightarrow y = 0$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-eq-cancel2*: $sqrt\ (x^2 + y^2) = y \Longrightarrow x = 0$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-ge1* [*simp*]: $x \leq sqrt\ (x^2 + y^2)$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-ge2* [*simp*]: $y \leq sqrt\ (x^2 + y^2)$
⟨*proof*⟩

**lemma** *real-sqrt-ge-abs1* [*simp*]: $|x| \leq sqrt\ (x^2 + y^2)$
⟨*proof*⟩

**lemma** *real-sqrt-ge-abs2* [*simp*]: $|y| \leq sqrt\ (x^2 + y^2)$
⟨*proof*⟩

**lemma** *le-real-sqrt-sumsq* [*simp*]: $x \leq sqrt\ (x * x + y * y)$
⟨*proof*⟩

**lemma** *sqrt-sum-squares-le-sum*:
$[\![0 \leq x;\ 0 \leq y]\!] \Longrightarrow sqrt\ (x^2 + y^2) \leq x + y$
⟨*proof*⟩

**lemma** *L2-set-mult-ineq-lemma*:
**fixes** *a b c d* :: *real*
**shows** $2 * (a * c) * (b * d) \leq a^2 * d^2 + b^2 * c^2$
⟨*proof*⟩

**lemma** *sqrt-sum-squares-le-sum-abs*: $sqrt\ (x^2 + y^2) \leq |x| + |y|$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-triangle-ineq*:
$sqrt\ ((a + c)^2 + (b + d)^2) \leq sqrt\ (a^2 + b^2) + sqrt\ (c^2 + d^2)$
⟨*proof*⟩

**lemma** *real-sqrt-sum-squares-less*: $|x| < u\ /\ sqrt\ 2 \Longrightarrow |y| < u\ /\ sqrt\ 2 \Longrightarrow sqrt\ (x^2 + y^2) < u$
⟨*proof*⟩

**lemma** *sqrt2-less-2*: $sqrt\ 2 < (2::real)$
⟨*proof*⟩

**lemma** *sqrt-sum-squares-half-less*:

$x < u/2 \Longrightarrow y < u/2 \Longrightarrow 0 \leq x \Longrightarrow 0 \leq y \Longrightarrow sqrt\ (x^2 + y^2) < u$
⟨*proof*⟩

**lemma** *LIMSEQ-root*: $(\lambda n.\ root\ n\ n) \longrightarrow 1$
⟨*proof*⟩

**lemma** *LIMSEQ-root-const*:
**assumes** $0 < c$
**shows** $(\lambda n.\ root\ n\ c) \longrightarrow 1$
⟨*proof*⟩

Legacy theorem names:

**lemmas** *real-root-pos2* = *real-root-power-cancel*
**lemmas** *real-root-pos-pos* = *real-root-gt-zero* [*THEN order-less-imp-le*]
**lemmas** *real-root-pos-pos-le* = *real-root-ge-zero*
**lemmas** *real-sqrt-eq-zero-cancel-iff* = *real-sqrt-eq-0-iff*

**end**

# 112 Power Series, Transcendental Functions etc.

**theory** *Transcendental*
**imports** *Series Deriv NthRoot*
**begin**

A theorem about the factcorial function on the reals.

**lemma** *square-fact-le-2-fact*: $fact\ n * fact\ n \leq (fact\ (2 * n) :: real)$
⟨*proof*⟩

**lemma** *fact-in-Reals*: $fact\ n \in \mathbb{R}$
⟨*proof*⟩

**lemma** *of-real-fact* [*simp*]: $of\text{-}real\ (fact\ n) = fact\ n$
⟨*proof*⟩

**lemma** *pochhammer-of-real*: $pochhammer\ (of\text{-}real\ x)\ n = of\text{-}real\ (pochhammer\ x\ n)$
⟨*proof*⟩

**lemma** *norm-fact* [*simp*]: $norm\ (fact\ n :: 'a::real\text{-}normed\text{-}algebra\text{-}1) = fact\ n$
⟨*proof*⟩

**lemma** *root-test-convergence*:
**fixes** $f :: nat \Rightarrow 'a::banach$
**assumes** $f$: $(\lambda n.\ root\ n\ (norm\ (f\ n))) \longrightarrow x$ — could be weakened to lim sup
**and** $x < 1$
**shows** *summable f*
⟨*proof*⟩

## 112.1 Properties of Power Series

**lemma** *powser-zero* [*simp*]: $(\sum n.\ f\ n * 0\ \hat{}\ n) = f\ 0$
**for** $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}algebra\text{-}1$
⟨*proof*⟩

**lemma** *powser-sums-zero*: $(\lambda n.\ a\ n * 0\hat{}n)$ *sums* $a\ 0$
**for** $a :: nat \Rightarrow 'a::real\text{-}normed\text{-}div\text{-}algebra$
⟨*proof*⟩

**lemma** *powser-sums-zero-iff* [*simp*]: $(\lambda n.\ a\ n * 0\hat{}n)$ *sums* $x \longleftrightarrow a\ 0 = x$
**for** $a :: nat \Rightarrow 'a::real\text{-}normed\text{-}div\text{-}algebra$
⟨*proof*⟩

Power series has a circle or radius of convergence: if it sums for $x$, then it sums absolutely for $z$ with $|z| < |x|$.

**lemma** *powser-insidea*:
**fixes** $x\ z :: 'a::real\text{-}normed\text{-}div\text{-}algebra$
**assumes** *1*: *summable* $(\lambda n.\ f\ n * x\hat{}n)$
**and** *2*: *norm* $z <$ *norm* $x$
**shows** *summable* $(\lambda n.\ norm\ (f\ n * z\ \hat{}\ n))$
⟨*proof*⟩

**lemma** *powser-inside*:
**fixes** $f :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}div\text{-}algebra,banach\}$
**shows**
*summable* $(\lambda n.\ f\ n * (x\hat{}n)) \Longrightarrow norm\ z < norm\ x \Longrightarrow$
*summable* $(\lambda n.\ f\ n * (z\ \hat{}\ n))$
⟨*proof*⟩

**lemma** *powser-times-n-limit-0*:
**fixes** $x :: 'a::\{real\text{-}normed\text{-}div\text{-}algebra,banach\}$
**assumes** *norm* $x < 1$
**shows** $(\lambda n.\ of\text{-}nat\ n * x\ \hat{}\ n) \longrightarrow 0$
⟨*proof*⟩

**corollary** *lim-n-over-pown*:
**fixes** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
**shows** $1 < norm\ x \Longrightarrow ((\lambda n.\ of\text{-}nat\ n\ /\ x\hat{}n) \longrightarrow 0)$ *sequentially*
⟨*proof*⟩

**lemma** *sum-split-even-odd*:
**fixes** $f :: nat \Rightarrow real$
**shows** $(\sum i<2 * n.\ if\ even\ i\ then\ f\ i\ else\ g\ i) = (\sum i<n.\ f\ (2 * i)) + (\sum i<n.\ g\ (2 * i + 1))$
⟨*proof*⟩

**lemma** *sums-if'*:
**fixes** $g :: nat \Rightarrow real$
**assumes** *g sums x*

**shows** $(\lambda\ n.\ \mathit{if\ even\ n\ then\ 0\ else\ g\ ((n - 1)\ div\ 2)})\ \mathit{sums}\ x$
⟨*proof*⟩

**lemma** *sums-if*:
**fixes** $g :: nat \Rightarrow real$
**assumes** $g\ \mathit{sums}\ x$ **and** $f\ \mathit{sums}\ y$
**shows** $(\lambda\ n.\ \mathit{if\ even\ n\ then\ f\ (n\ div\ 2)\ else\ g\ ((n - 1)\ div\ 2)})\ \mathit{sums}\ (x + y)$
⟨*proof*⟩

## 112.2 Alternating series test / Leibniz formula

**lemma** *sums-alternating-upper-lower*:
**fixes** $a :: nat \Rightarrow real$
**assumes** *mono*: $\bigwedge n.\ a\ (Suc\ n) \le a\ n$
**and** *a-pos*: $\bigwedge n.\ 0 \le a\ n$
**and** $a \longrightarrow 0$
**shows** $\exists l.\ ((\forall n.\ (\sum i<2*n.\ (-\ 1)\hat{}i*a\ i) \le l) \wedge (\lambda\ n.\ \sum i<2*n.\ (-\ 1)\hat{}i*a\ i) \longrightarrow l) \wedge$
$((\forall n.\ l \le (\sum i<2*n + 1.\ (-\ 1)\hat{}i*a\ i)) \wedge (\lambda\ n.\ \sum i<2*n + 1.\ (-\ 1)\hat{}i*a\ i) \longrightarrow l)$
(**is** $\exists l.\ ((\forall n.\ ?f\ n \le l) \wedge \text{-}) \wedge ((\forall n.\ l \le ?g\ n) \wedge \text{-})$)
⟨*proof*⟩

**lemma** *summable-Leibniz'*:
**fixes** $a :: nat \Rightarrow real$
**assumes** *a-zero*: $a \longrightarrow 0$
**and** *a-pos*: $\bigwedge n.\ 0 \le a\ n$
**and** *a-monotone*: $\bigwedge n.\ a\ (Suc\ n) \le a\ n$
**shows** *summable*: $\mathit{summable}\ (\lambda\ n.\ (-1)\hat{}n * a\ n)$
**and** $\bigwedge n.\ (\sum i<2*n.\ (-1)\hat{}i*a\ i) \le (\sum i.\ (-1)\hat{}i*a\ i)$
**and** $(\lambda n.\ \sum i<2*n.\ (-1)\hat{}i*a\ i) \longrightarrow (\sum i.\ (-1)\hat{}i*a\ i)$
**and** $\bigwedge n.\ (\sum i.\ (-1)\hat{}i*a\ i) \le (\sum i<2*n+1.\ (-1)\hat{}i*a\ i)$
**and** $(\lambda n.\ \sum i<2*n+1.\ (-1)\hat{}i*a\ i) \longrightarrow (\sum i.\ (-1)\hat{}i*a\ i)$
⟨*proof*⟩

**theorem** *summable-Leibniz*:
**fixes** $a :: nat \Rightarrow real$
**assumes** *a-zero*: $a \longrightarrow 0$
**and** *monoseq a*
**shows** $\mathit{summable}\ (\lambda\ n.\ (-1)\hat{}n * a\ n)$ (**is** *?summable*)
**and** $0 < a\ 0 \longrightarrow$
$(\forall n.\ (\sum i.\ (-\ 1)\hat{}i*a\ i) \in \{\sum i<2*n.\ (-\ 1)\hat{}i * a\ i\ ..\ \sum i<2*n+1.\ (-\ 1)\hat{}i * a\ i\})$ (**is** *?pos*)
**and** $a\ 0 < 0 \longrightarrow$
$(\forall n.\ (\sum i.\ (-\ 1)\hat{}i*a\ i) \in \{\sum i<2*n+1.\ (-\ 1)\hat{}i * a\ i\ ..\ \sum i<2*n.\ (-\ 1)\hat{}i * a\ i\})$ (**is** *?neg*)
**and** $(\lambda n.\ \sum i<2*n.\ (-\ 1)\hat{}i*a\ i) \longrightarrow (\sum i.\ (-\ 1)\hat{}i*a\ i)$ (**is** *?f*)
**and** $(\lambda n.\ \sum i<2*n+1.\ (-\ 1)\hat{}i*a\ i) \longrightarrow (\sum i.\ (-\ 1)\hat{}i*a\ i)$ (**is** *?g*)
⟨*proof*⟩

## 112.3 Term-by-Term Differentiability of Power Series

**definition** *diffs* :: $(nat \Rightarrow 'a::ring\text{-}1) \Rightarrow nat \Rightarrow 'a$
**where** $diffs\ c = (\lambda n.\ of\text{-}nat\ (Suc\ n) * c\ (Suc\ n))$

Lemma about distributing negation over it.

**lemma** *diffs-minus*: $diffs\ (\lambda n.\ -\ c\ n) = (\lambda n.\ -\ diffs\ c\ n)$
⟨*proof*⟩

**lemma** *diffs-equiv*:
**fixes** $x$ :: $'a::\{real\text{-}normed\text{-}vector,ring\text{-}1\}$
**shows** $summable\ (\lambda n.\ diffs\ c\ n * x\hat{\ }n) \Longrightarrow$
$(\lambda n.\ of\text{-}nat\ n * c\ n * x\hat{\ }(n - Suc\ 0))\ sums\ (\sum n.\ diffs\ c\ n * x\hat{\ }n)$
⟨*proof*⟩

**lemma** *lemma-termdiff1*:
**fixes** $z$ :: $'a$ :: $\{monoid\text{-}mult,comm\text{-}ring\}$
**shows** $(\sum p<m.\ (((z + h)\ \hat{\ }(m - p)) * (z\ \hat{\ }p)) - (z\ \hat{\ }m)) =$
$(\sum p<m.\ (z\ \hat{\ }p) * (((z + h)\ \hat{\ }(m - p)) - (z\ \hat{\ }(m - p))))$
⟨*proof*⟩

**lemma** *sumr-diff-mult-const2*: $sum\ f\ \{..<n\} - of\text{-}nat\ n * r = (\sum i<n.\ f\ i - r)$
**for** $r$ :: $'a::ring\text{-}1$
⟨*proof*⟩

**lemma** *lemma-termdiff2*:
**fixes** $h$ :: $'a::field$
**assumes** $h$: $h \neq 0$
**shows** $((z + h)\ \hat{\ }n - z\ \hat{\ }n)\ /\ h - of\text{-}nat\ n * z\ \hat{\ }(n - Suc\ 0) =$
$h * (\sum p< n - Suc\ 0.\ \sum q< n - Suc\ 0 - p.\ (z + h)\ \hat{\ }q * z\ \hat{\ }(n - 2 -$
$q))$
(**is** $?lhs = ?rhs$)
⟨*proof*⟩

**lemma** *real-sum-nat-ivl-bounded2*:
**fixes** $K$ :: $'a::linordered\text{-}semidom$
**assumes** $f$: $\bigwedge p::nat.\ p < n \Longrightarrow f\ p \leq K$ **and** $K$: $0 \leq K$
**shows** $sum\ f\ \{..<n-k\} \leq of\text{-}nat\ n * K$
⟨*proof*⟩

**lemma** *lemma-termdiff3*:
**fixes** $h\ z$ :: $'a::real\text{-}normed\text{-}field$
**assumes** $1$: $h \neq 0$
**and** $2$: $norm\ z \leq K$
**and** $3$: $norm\ (z + h) \leq K$
**shows** $norm\ (((z + h)\ \hat{\ }n - z\ \hat{\ }n)\ /\ h - of\text{-}nat\ n * z\ \hat{\ }(n - Suc\ 0)) \leq$
$of\text{-}nat\ n * of\text{-}nat\ (n - Suc\ 0) * K\ \hat{\ }(n - 2) * norm\ h$
⟨*proof*⟩

**lemma** *lemma-termdiff4*:
  **fixes** $f$ :: $'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
    **and** $k$ :: *real*
  **assumes** $k$: $0 < k$
    **and** *le*: $\bigwedge h.\ h \neq 0 \Longrightarrow norm\ h < k \Longrightarrow norm\ (f\ h) \leq K * norm\ h$
  **shows** $f\ -0\rightarrow\ 0$
⟨*proof*⟩

**lemma** *lemma-termdiff5*:
  **fixes** $g$ :: $'a$::*real-normed-vector* $\Rightarrow$ *nat* $\Rightarrow$ $'b$::*banach*
    **and** $k$ :: *real*
  **assumes** $k$: $0 < k$
    **and** $f$: *summable f*
    **and** *le*: $\bigwedge h\ n.\ h \neq 0 \Longrightarrow norm\ h < k \Longrightarrow norm\ (g\ h\ n) \leq f\ n * norm\ h$
  **shows** $(\lambda h.\ suminf\ (g\ h))\ -0\rightarrow\ 0$
⟨*proof*⟩

**lemma** *termdiffs-aux*:
  **fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
  **assumes** *1*: *summable* $(\lambda n.\ diffs\ (diffs\ c)\ n * K\ \hat{}\ n)$
    **and** *2*: $norm\ x < norm\ K$
  **shows** $(\lambda h.\ \sum n.\ c\ n * (((x + h)\ \hat{}\ n - x\hat{}n) / h - of\text{-}nat\ n * x\ \hat{}(n - Suc\ 0)))$
$-0\rightarrow\ 0$
⟨*proof*⟩

**lemma** *termdiffs*:
  **fixes** $K\ x$ :: $'a$::{*real-normed-field*,*banach*}
  **assumes** *1*: *summable* $(\lambda n.\ c\ n * K\ \hat{}\ n)$
    **and** *2*: *summable* $(\lambda n.\ (diffs\ c)\ n * K\ \hat{}\ n)$
    **and** *3*: *summable* $(\lambda n.\ (diffs\ (diffs\ c))\ n * K\ \hat{}\ n)$
    **and** *4*: $norm\ x < norm\ K$
  **shows** *DERIV* $(\lambda x.\ \sum n.\ c\ n * x\hat{}n)\ x :> (\sum n.\ (diffs\ c)\ n * x\hat{}n)$
  ⟨*proof*⟩

## 112.4 The Derivative of a Power Series Has the Same Radius of Convergence

**lemma** *termdiff-converges*:
  **fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
  **assumes** $K$: $norm\ x < K$
    **and** *sm*: $\bigwedge x.\ norm\ x < K \Longrightarrow summable(\lambda n.\ c\ n * x\ \hat{}\ n)$
  **shows** *summable* $(\lambda n.\ diffs\ c\ n * x\ \hat{}\ n)$
⟨*proof*⟩

**lemma** *termdiff-converges-all*:
  **fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}

**assumes** $\bigwedge x.$ *summable* $(\lambda n.\ c\ n * x\widehat{\ }n)$
**shows** *summable* $(\lambda n.\ \mathit{diffs}\ c\ n * x\widehat{\ }n)$
⟨*proof*⟩

**lemma** *termdiffs-strong*:
**fixes** $K\ x$ :: $'a$::{*real-normed-field*,*banach*}
**assumes** *sm*: *summable* $(\lambda n.\ c\ n * K\ \widehat{\ }\ n)$
**and** $K$: *norm* $x <$ *norm* $K$
**shows** *DERIV* $(\lambda x. \sum n.\ c\ n * x\widehat{\ }n)\ x :> (\sum n.\ \mathit{diffs}\ c\ n * x\widehat{\ }n)$
⟨*proof*⟩

**lemma** *termdiffs-strong-converges-everywhere*:
**fixes** $K\ x$ :: $'a$::{*real-normed-field*,*banach*}
**assumes** $\bigwedge y.$ *summable* $(\lambda n.\ c\ n * y\ \widehat{\ }\ n)$
**shows** $((\lambda x. \sum n.\ c\ n * x\widehat{\ }n)$ *has-field-derivative* $(\sum n.\ \mathit{diffs}\ c\ n * x\widehat{\ }n))$ $(at\ x)$
⟨*proof*⟩

**lemma** *termdiffs-strong'*:
**fixes** $z$ :: $'a$ :: {*real-normed-field*,*banach*}
**assumes** $\bigwedge z.$ *norm* $z < K \Longrightarrow$ *summable* $(\lambda n.\ c\ n * z\ \widehat{\ }\ n)$
**assumes** *norm* $z < K$
**shows** $((\lambda z. \sum n.\ c\ n * z\widehat{\ }n)$ *has-field-derivative* $(\sum n.\ \mathit{diffs}\ c\ n * z\widehat{\ }n))$ $(at\ z)$
⟨*proof*⟩

**lemma** *termdiffs-sums-strong*:
**fixes** $z$ :: $'a$ :: {*banach*,*real-normed-field*}
**assumes** *sums*: $\bigwedge z.$ *norm* $z < K \Longrightarrow (\lambda n.\ c\ n * z\ \widehat{\ }\ n)$ *sums* $f\ z$
**assumes** *deriv*: ($f$ *has-field-derivative* $f'$) $(at\ z)$
**assumes** *norm*: *norm* $z < K$
**shows** $(\lambda n.\ \mathit{diffs}\ c\ n * z\ \widehat{\ }\ n)$ *sums* $f'$
⟨*proof*⟩

**lemma** *isCont-powser*:
**fixes** $K\ x$ :: $'a$::{*real-normed-field*,*banach*}
**assumes** *summable* $(\lambda n.\ c\ n * K\ \widehat{\ }\ n)$
**assumes** *norm* $x <$ *norm* $K$
**shows** *isCont* $(\lambda x. \sum n.\ c\ n * x\widehat{\ }n)\ x$
⟨*proof*⟩

**lemmas** *isCont-powser'* = *isCont-o2*[*OF - isCont-powser*]

**lemma** *isCont-powser-converges-everywhere*:
**fixes** $K\ x$ :: $'a$::{*real-normed-field*,*banach*}
**assumes** $\bigwedge y.$ *summable* $(\lambda n.\ c\ n * y\ \widehat{\ }\ n)$
**shows** *isCont* $(\lambda x. \sum n.\ c\ n * x\widehat{\ }n)\ x$
⟨*proof*⟩

**lemma** *powser-limit-0*:
**fixes** $a$ :: $nat \Rightarrow\ 'a$::{*real-normed-field*,*banach*}

**assumes** *s*: $0 < s$
**and** *sm*: $\bigwedge x.\ norm\ x < s \Longrightarrow (\lambda n.\ a\ n * x \hat{}\ n)\ sums\ (f\ x)$
**shows** $(f \longrightarrow a\ 0)\ (at\ 0)$
⟨*proof*⟩

**lemma** *powser-limit-0-strong*:
**fixes** $a :: nat \Rightarrow 'a::\{real\text{-}normed\text{-}field,banach\}$
**assumes** *s*: $0 < s$
**and** *sm*: $\bigwedge x.\ x \neq 0 \Longrightarrow norm\ x < s \Longrightarrow (\lambda n.\ a\ n * x \hat{}\ n)\ sums\ (f\ x)$
**shows** $(f \longrightarrow a\ 0)\ (at\ 0)$
⟨*proof*⟩

## 112.5 Derivability of power series

**lemma** *DERIV-series'*:
**fixes** $f :: real \Rightarrow nat \Rightarrow real$
**assumes** *DERIV-f*: $\bigwedge n.\ DERIV\ (\lambda\ x.\ f\ x\ n)\ x0 :> (f'\ x0\ n)$
**and** *allf-summable*: $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow summable\ (f\ x)$
**and** *x0-in-I*: $x0 \in \{a <..< b\}$
**and** $summable\ (f'\ x0)$
**and** $summable\ L$
**and** *L-def*: $\bigwedge n\ x\ y.\ x \in \{a <..< b\} \Longrightarrow y \in \{a <..< b\} \Longrightarrow |f\ x\ n - f\ y\ n| \leq L\ n * |x - y|$
**shows** $DERIV\ (\lambda\ x.\ suminf\ (f\ x))\ x0 :> (suminf\ (f'\ x0))$
⟨*proof*⟩

**lemma** *DERIV-power-series'*:
**fixes** $f :: nat \Rightarrow real$
**assumes** *converges*: $\bigwedge x.\ x \in \{-R <..< R\} \Longrightarrow summable\ (\lambda n.\ f\ n * real\ (Suc\ n) * x\hat{}n)$
**and** *x0-in-I*: $x0 \in \{-R <..< R\}$
**and** $0 < R$
**shows** $DERIV\ (\lambda x.\ (\sum n.\ f\ n * x\hat{}(Suc\ n)))\ x0 :> (\sum n.\ f\ n * real\ (Suc\ n) * x0\hat{}n)$
(**is** $DERIV\ (\lambda x.\ suminf\ (?f\ x))\ x0 :> suminf\ (?f'\ x0)$)
⟨*proof*⟩

**lemma** *geometric-deriv-sums*:
**fixes** $z :: 'a :: \{real\text{-}normed\text{-}field,banach\}$
**assumes** $norm\ z < 1$
**shows** $(\lambda n.\ of\text{-}nat\ (Suc\ n) * z \hat{}\ n)\ sums\ (1\ /\ (1 - z)\hat{}2)$
⟨*proof*⟩

**lemma** *isCont-pochhammer* [*continuous-intros*]: $isCont\ (\lambda z.\ pochhammer\ z\ n)\ z$
**for** $z :: 'a::real\text{-}normed\text{-}field$
⟨*proof*⟩

**lemma** *continuous-on-pochhammer* [*continuous-intros*]: $continuous\text{-}on\ A\ (\lambda z.\ pochhammer\ z\ n)$

**for** $A$ :: $'a$::*real-normed-field set*
⟨*proof*⟩

**lemmas** *continuous-on-pochhammer'* [*continuous-intros*] =
*continuous-on-compose2* [*OF continuous-on-pochhammer - subset-UNIV*]

## 112.6 Exponential Function

**definition** *exp* :: $'a \Rightarrow 'a$::{*real-normed-algebra-1*,*banach*}
**where** $exp = (\lambda x. \sum n.\ x \widehat{\ } n /_R \textit{fact } n)$

**lemma** *summable-exp-generic*:
**fixes** $x$ :: $'a$::{*real-normed-algebra-1*,*banach*}
**defines** *S-def*: $S \equiv \lambda n.\ x \widehat{\ } n /_R \textit{fact } n$
**shows** *summable S*
⟨*proof*⟩

**lemma** *summable-norm-exp*: *summable* $(\lambda n.\ norm\ (x \widehat{\ } n /_R \textit{fact } n))$
**for** $x$ :: $'a$::{*real-normed-algebra-1*,*banach*}
⟨*proof*⟩

**lemma** *summable-exp*: *summable* $(\lambda n.\ inverse\ (fact\ n) * x \widehat{\ } n)$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-converges*: $(\lambda n.\ x \widehat{\ } n /_R \textit{fact } n)$ *sums exp x*
⟨*proof*⟩

**lemma** *exp-fdiffs*:
*diffs* $(\lambda n.\ inverse\ (fact\ n)) = (\lambda n.\ inverse\ (fact\ n :: 'a$::{*real-normed-field*,*banach*}))
⟨*proof*⟩

**lemma** *diffs-of-real*: *diffs* $(\lambda n.\ of\text{-}real\ (f\ n)) = (\lambda n.\ of\text{-}real\ (diffs\ f\ n))$
⟨*proof*⟩

**lemma** *DERIV-exp* [*simp*]: *DERIV exp x :> exp x*
⟨*proof*⟩

**declare** *DERIV-exp*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-exp*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemmas** *has-derivative-exp*[*derivative-intros*] = *DERIV-exp*[*THEN DERIV-compose-FDERIV*]

**lemma** *norm-exp*: $norm\ (exp\ x) \le exp\ (norm\ x)$
⟨*proof*⟩

**lemma** *isCont-exp*: *isCont exp x*
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}

⟨*proof*⟩

**lemma** *isCont-exp'* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* ($\lambda x$. *exp* (*f x*)) *a*
**for** *f* :: - $\Rightarrow$'*a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *tendsto-exp* [*tendsto-intros*]: (*f* $\longrightarrow$ *a*) *F* $\Longrightarrow$ (($\lambda x$. *exp* (*f x*)) $\longrightarrow$ *exp a*) *F*
**for** *f*:: - $\Rightarrow$'*a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *continuous-exp* [*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* ($\lambda x$. *exp* (*f x*))
**for** *f* :: - $\Rightarrow$'*a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *continuous-on-exp* [*continuous-intros*]: *continuous-on s f* $\Longrightarrow$ *continuous-on s* ($\lambda x$. *exp* (*f x*))
**for** *f* :: - $\Rightarrow$'*a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

### 112.6.1 Properties of the Exponential Function

**lemma** *exp-zero* [*simp*]: *exp 0* = *1*
⟨*proof*⟩

**lemma** *exp-series-add-commuting*:
**fixes** *x y* :: '*a*::{*real-normed-algebra-1*,*banach*}
**defines** *S-def*: $S \equiv \lambda x\ n.\ x\hat{}n\ /_R$ *fact n*
**assumes** *comm*: $x * y = y * x$
**shows** $S\ (x + y)\ n = (\sum i{\le}n.\ S\ x\ i * S\ y\ (n - i))$
⟨*proof*⟩

**lemma** *exp-add-commuting*: $x * y = y * x \Longrightarrow$ *exp* $(x + y)$ = *exp x* $*$ *exp y*
⟨*proof*⟩

**lemma** *exp-times-arg-commute*: *exp A* $*$ *A* = *A* $*$ *exp A*
⟨*proof*⟩

**lemma** *exp-add*: *exp* $(x + y)$ = *exp x* $*$ *exp y*
**for** *x y* :: '*a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-double*: *exp*($2 * z$) = *exp z* ^ *2*
⟨*proof*⟩

**lemmas** *mult-exp-exp* = *exp-add* [*symmetric*]

**lemma** *exp-of-real*: *exp* (*of-real x*) = *of-real* (*exp x*)

⟨*proof*⟩

**lemmas** *of-real-exp* = *exp-of-real*[*symmetric*]

**corollary** *exp-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow exp\ z \in \mathbb{R}$
⟨*proof*⟩

**lemma** *exp-not-eq-zero* [*simp*]: $exp\ x \neq 0$
⟨*proof*⟩

**lemma** *exp-minus-inverse*: $exp\ x * exp\ (-\ x) = 1$
⟨*proof*⟩

**lemma** *exp-minus*: $exp\ (-\ x) = inverse\ (exp\ x)$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-diff*: $exp\ (x - y) = exp\ x\ /\ exp\ y$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-of-nat-mult*: $exp\ (of\text{-}nat\ n * x) = exp\ x\ \hat{}\ n$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**corollary** *exp-of-nat2-mult*: $exp\ (x * of\text{-}nat\ n) = exp\ x\ \hat{}\ n$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-sum*: $finite\ I \Longrightarrow exp\ (sum\ f\ I) = prod\ (\lambda x.\ exp\ (f\ x))\ I$
⟨*proof*⟩

**lemma** *exp-divide-power-eq*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**assumes** $n > 0$
**shows** $exp\ (x\ /\ of\text{-}nat\ n)\ \hat{}\ n = exp\ x$
⟨*proof*⟩

**lemma** *exp-power-int*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**shows** $exp\ x\ powi\ n = exp\ (of\text{-}int\ n * x)$
⟨*proof*⟩

### 112.6.2 Properties of the Exponential Function on Reals

Comparisons of *exp x* with zero.

Proof: because every exponential can be seen as a square.

**lemma** *exp-ge-zero* [*simp*]: $0 \leq exp\ x$

**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *exp-gt-zero* [*simp*]: $0 < exp\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *not-exp-less-zero* [*simp*]: $\neg\ exp\ x < 0$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *not-exp-le-zero* [*simp*]: $\neg\ exp\ x \leq 0$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *abs-exp-cancel* [*simp*]: $|exp\ x| = exp\ x$
**for** $x$ :: *real*
⟨*proof*⟩

Strict monotonicity of exponential.

**lemma** *exp-ge-add-one-self-aux*:
**fixes** $x$ :: *real*
**assumes** $0 \leq x$
**shows** $1 + x \leq exp\ x$
⟨*proof*⟩

**lemma** *exp-gt-one*: $0 < x \Longrightarrow 1 < exp\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *exp-less-mono*:
**fixes** $x\ y$ :: *real*
**assumes** $x < y$
**shows** $exp\ x < exp\ y$
⟨*proof*⟩

**lemma** *exp-less-cancel*: $exp\ x < exp\ y \Longrightarrow x < y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *exp-less-cancel-iff* [*iff*]: $exp\ x < exp\ y \longleftrightarrow x < y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *exp-le-cancel-iff* [*iff*]: $exp\ x \leq exp\ y \longleftrightarrow x \leq y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *exp-mono*:

**fixes** *x y* :: *real*
**assumes** $x \leq y$
**shows** $exp\ x \leq exp\ y$
⟨*proof*⟩

**lemma** *exp-minus'*: $exp\ (-x) = 1\ /\ (exp\ x)$
**for** *x* :: *'a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *exp-inj-iff* [*iff*]: $exp\ x = exp\ y \longleftrightarrow x = y$
**for** *x y* :: *real*
⟨*proof*⟩

Comparisons of *exp x* with one.

**lemma** *one-less-exp-iff* [*simp*]: $1 < exp\ x \longleftrightarrow 0 < x$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *exp-less-one-iff* [*simp*]: $exp\ x < 1 \longleftrightarrow x < 0$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *one-le-exp-iff* [*simp*]: $1 \leq exp\ x \longleftrightarrow 0 \leq x$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *exp-le-one-iff* [*simp*]: $exp\ x \leq 1 \longleftrightarrow x \leq 0$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *exp-eq-one-iff* [*simp*]: $exp\ x = 1 \longleftrightarrow x = 0$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *lemma-exp-total*: $1 \leq y \Longrightarrow \exists x.\ 0 \leq x \land x \leq y - 1 \land exp\ x = y$
**for** *y* :: *real*
⟨*proof*⟩

**lemma** *exp-total*: $0 < y \Longrightarrow \exists x.\ exp\ x = y$
**for** *y* :: *real*
⟨*proof*⟩

## 112.7 Natural Logarithm

**class** *ln* = *real-normed-algebra-1* + *banach* +
**fixes** *ln* :: $'a \Rightarrow 'a$
**assumes** *ln-one* [*simp*]: $ln\ 1 = 0$

**definition** *powr* :: $'a \Rightarrow 'a \Rightarrow 'a$::*ln* (**infixr** ‹*powr*› *80*)

— exponentation via ln and exp
**where** $x$ *powr* $a \equiv$ *if* $x = 0$ *then* $0$ *else* *exp* $(a * ln\ x)$

**lemma** *powr-0* [*simp*]: $0$ *powr* $z = 0$
⟨*proof*⟩

We totalise *ln* over all reals exactly as done in Mathlib

**instantiation** *real* :: *ln*
**begin**

**definition** *raw-ln-real* :: *real* ⇒ *real*
**where** *raw-ln-real* $x \equiv$ (*THE* $u$. *exp* $u = x$)

**definition** *ln-real* :: *real* ⇒ *real*
**where** *ln-real* $\equiv \lambda x$. *if* $x=0$ *then* $0$ *else* *raw-ln-real* $|x|$

**instance**
⟨*proof*⟩

**end**

**lemma** *powr-eq-0-iff* [*simp*]: $w$ *powr* $z = 0 \longleftrightarrow w = 0$
⟨*proof*⟩

**lemma** *raw-ln-exp* [*simp*]: *raw-ln-real* (*exp* $x$) $= x$
⟨*proof*⟩

**lemma** *exp-raw-ln* [*simp*]: $0 < x \Longrightarrow$ *exp* (*raw-ln-real* $x$) $= x$
⟨*proof*⟩

**lemma** *raw-ln-unique*: *exp* $y = x \Longrightarrow$ *raw-ln-real* $x = y$
⟨*proof*⟩

**lemma** *abs-raw-ln*: $x \neq 0 \Longrightarrow$ *raw-ln-real*$|x| =$ *ln* $x$
⟨*proof*⟩

**lemma** *ln-0* [*simp*]: *ln* ($0$::*real*) $= 0$
⟨*proof*⟩

**lemma** *ln-minus*: *ln* $(-x) =$ *ln* $x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-exp* [*simp*]: *ln* (*exp* $x$) $= x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *exp-ln-abs*:
**fixes** $x$::*real*

**shows** $x \neq 0 \Longrightarrow exp\ (ln\ x) = |x|$
⟨*proof*⟩

**lemma** *exp-ln* [*simp*]: $0 < x \Longrightarrow exp\ (ln\ x) = x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *exp-ln-iff* [*simp*]: $exp\ (ln\ x) = x \longleftrightarrow 0 < x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-unique*: $exp\ y = x \Longrightarrow ln\ x = y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-unique'*: $exp\ y = |x| \Longrightarrow ln\ x = y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *raw-ln-mult*: $x>0 \Longrightarrow y>0 \Longrightarrow raw\text{-}ln\text{-}real\ (x * y) = raw\text{-}ln\text{-}real\ x + raw\text{-}ln\text{-}real\ y$
⟨*proof*⟩

**lemma** *ln-mult*: $ln\ (x * y) = (if\ x\neq 0 \wedge y\neq 0\ then\ ln\ x + ln\ y\ else\ 0)$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-mult-pos*: $x>0 \Longrightarrow y>0 \Longrightarrow ln\ (x * y) = ln\ x + ln\ y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-prod*: $finite\ I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow f\ i \neq 0) \Longrightarrow ln\ (prod\ f\ I) = sum\ (\lambda x.\ ln(f\ x))\ I$
**for** $f$ :: $'a \Rightarrow real$
⟨*proof*⟩

**lemma** *ln-inverse*: $ln\ (inverse\ x) = -\ ln\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-div*: $ln\ (x/y) = (if\ x\neq 0 \wedge y\neq 0\ then\ ln\ x - ln\ y\ else\ 0)$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-divide-pos*: $x>0 \Longrightarrow y>0 \Longrightarrow ln\ (x/y) = ln\ x - ln\ y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-realpow*: $ln\ (x\widehat{\ }n) = real\ n * ln\ x$

⟨*proof*⟩

**lemma** *ln-less-cancel-iff* [*simp*]: $0 < x \Longrightarrow 0 < y \Longrightarrow \ln x < \ln y \longleftrightarrow x < y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-le-cancel-iff* [*simp*]: $0 < x \Longrightarrow 0 < y \Longrightarrow \ln x \leq \ln y \longleftrightarrow x \leq y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-mono*: $\bigwedge x$::*real*. $[\![x \leq y;\ 0 < x]\!] \Longrightarrow \ln x \leq \ln y$
⟨*proof*⟩

**lemma** *ln-strict-mono*: $\bigwedge x$::*real*. $[\![x < y;\ 0 < x]\!] \Longrightarrow \ln x < \ln y$
⟨*proof*⟩

**lemma** *ln-inj-iff* [*simp*]: $0 < x \Longrightarrow 0 < y \Longrightarrow \ln x = \ln y \longleftrightarrow x = y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-add-one-self-le-self*: $0 \leq x \Longrightarrow \ln (1 + x) \leq x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-less-self* [*simp*]: $0 < x \Longrightarrow \ln x < x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-ge-iff*: $\bigwedge x$::*real*. $0 < x \Longrightarrow y \leq \ln x \longleftrightarrow \exp y \leq x$
⟨*proof*⟩

**lemma** *ln-ge-zero* [*simp*]: $1 \leq x \Longrightarrow 0 \leq \ln x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-ge-zero-imp-ge-one*: $0 \leq \ln x \Longrightarrow 0 < x \Longrightarrow 1 \leq x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-ge-zero-iff* [*simp*]: $0 < x \Longrightarrow 0 \leq \ln x \longleftrightarrow 1 \leq x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-less-zero-iff* [*simp*]: $0 < x \Longrightarrow \ln x < 0 \longleftrightarrow x < 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-le-zero-iff* [*simp*]: $0 < x \Longrightarrow \ln x \leq 0 \longleftrightarrow x \leq 1$
**for** $x$ :: *real*

⟨*proof*⟩

**lemma** *ln-gt-zero*: $1 < x \Longrightarrow 0 < \ln x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-gt-zero-imp-gt-one*: $0 < \ln x \Longrightarrow 0 < x \Longrightarrow 1 < x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-gt-zero-iff* [*simp*]: $0 < x \Longrightarrow 0 < \ln x \longleftrightarrow 1 < x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-eq-zero-iff* [*simp*]: $0 < x \Longrightarrow \ln x = 0 \longleftrightarrow x = 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-less-zero*: $0 < x \Longrightarrow x < 1 \Longrightarrow \ln x < 0$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-eq-one-iff* [*simp*]:
$a$ *powr* $x = 1 \longleftrightarrow x = 0$ **if** $a > 1$ **for** $a$ $x$ :: *real*
⟨*proof*⟩

A consequence of our "totalising" of ln

**lemma** *uminus-powr-eq*: $(-a)$ *powr* $x = a$ *powr* $x$ **for** $x$::*real*
⟨*proof*⟩

**lemma** *isCont-ln-pos*:
**fixes** $x$ :: *real*
**assumes** $x > 0$
**shows** *isCont ln x*
⟨*proof*⟩

**lemma** *isCont-ln*:
**fixes** $x$ :: *real*
**assumes** $x \neq 0$
**shows** *isCont ln x*
⟨*proof*⟩

**lemma** *tendsto-ln* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow a \neq 0 \Longrightarrow ((\lambda x.\ \ln\ (f\ x)) \longrightarrow \ln\ a)\ F$
**for** $a$ :: *real*
⟨*proof*⟩

**lemma** *continuous-ln*:
*continuous* $F\ f \Longrightarrow f\ (Lim\ F\ (\lambda x.\ x)) \neq 0 \Longrightarrow$ *continuous* $F\ (\lambda x.\ \ln\ (f\ x$ :: *real*$))$

⟨*proof*⟩

**lemma** *isCont-ln′* [*continuous-intros*]:
*continuous* (*at x*) *f* $\Longrightarrow$ *f x* $\neq$ *0* $\Longrightarrow$ *continuous* (*at x*) ($\lambda x$. *ln* (*f x* :: *real*))
⟨*proof*⟩

**lemma** *continuous-within-ln* [*continuous-intros*]:
*continuous* (*at x within s*) *f* $\Longrightarrow$ *f x* $\neq$ *0* $\Longrightarrow$ *continuous* (*at x within s*) ($\lambda x$. *ln* (*f x* :: *real*))
⟨*proof*⟩

**lemma** *continuous-on-ln* [*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ ($\forall$ *x*$\in$*s*. *f x* $\neq$ *0*) $\Longrightarrow$ *continuous-on s* ($\lambda x$. *ln* (*f x* :: *real*))
⟨*proof*⟩

**lemma** *DERIV-ln*: *0* < *x* $\Longrightarrow$ *DERIV ln x* :> *inverse x*
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *DERIV-ln-divide*: *0* < *x* $\Longrightarrow$ *DERIV ln x* :> *1*/*x*
**for** *x* :: *real*
⟨*proof*⟩

**declare** *DERIV-ln-divide*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-ln-divide*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemmas** *has-derivative-ln*[*derivative-intros*] = *DERIV-ln*[*THEN DERIV-compose-FDERIV*]

**lemma** *ln-series*:
**assumes** *0* < *x* **and** *x* < *2*
**shows** *ln x* = ($\sum$ *n*. (−*1*)^*n* ∗ (*1* / *real* (*n* + *1*)) ∗ (*x* − *1*)^(*Suc n*))
(**is** *ln x* = *suminf* (*?f* (*x* − *1*)))
⟨*proof*⟩

**lemma** *exp-first-terms*:
**fixes** *x* :: ′*a*::{*real-normed-algebra-1*,*banach*}
**shows** *exp x* = ($\sum$ *n*<*k*. *inverse*(*fact n*) $*_R$ (*x* ^ *n*)) + ($\sum$ *n*. *inverse*(*fact* (*n* + *k*)) $*_R$ (*x* ^ (*n* + *k*)))
⟨*proof*⟩

**lemma** *exp-first-term*: *exp x* = *1* + ($\sum$ *n*. *inverse* (*fact* (*Suc n*)) $*_R$ (*x* ^ *Suc n*))
**for** *x* :: ′*a*::{*real-normed-algebra-1*,*banach*}
⟨*proof*⟩

**lemma** *exp-first-two-terms*: *exp x* = *1* + *x* + ($\sum$ *n*. *inverse* (*fact* (*n* + *2*)) $*_R$ (*x* ^ (*n* + *2*)))
**for** *x* :: ′*a*::{*real-normed-algebra-1*,*banach*}
⟨*proof*⟩

**lemma** *exp-bound*:
  **fixes** $x$ :: *real*
  **assumes** $a$: $0 \leq x$
    **and** $b$: $x \leq 1$
  **shows** $exp\ x \leq 1 + x + x^2$
⟨*proof*⟩

**corollary** *exp-half-le2*: $exp(1/2) \leq (2::real)$
  ⟨*proof*⟩

**corollary** *exp-le*: $exp\ 1 \leq (3::real)$
  ⟨*proof*⟩

**lemma** *exp-bound-half*: $norm\ z \leq 1/2 \Longrightarrow norm\ (exp\ z) \leq 2$
  ⟨*proof*⟩

**lemma** *exp-bound-lemma*:
  **assumes** $norm\ z \leq 1/2$
  **shows** $norm\ (exp\ z) \leq 1 + 2 * norm\ z$
⟨*proof*⟩

**lemma** *real-exp-bound-lemma*: $0 \leq x \Longrightarrow x \leq 1/2 \Longrightarrow exp\ x \leq 1 + 2 * x$
  **for** $x$ :: *real*
  ⟨*proof*⟩

**lemma** *ln-one-minus-pos-upper-bound*:
  **fixes** $x$ :: *real*
  **assumes** $a$: $0 \leq x$ **and** $b$: $x < 1$
  **shows** $ln\ (1 - x) \leq - x$
⟨*proof*⟩

**lemma** *exp-ge-add-one-self* [*simp*]: $1 + x \leq exp\ x$
  **for** $x$ :: *real*
⟨*proof*⟩

**lemma** *exp-gt-self*: $x < exp\ (x::real)$
  ⟨*proof*⟩

**lemma** *ln-one-plus-pos-lower-bound*:
  **fixes** $x$ :: *real*
  **assumes** $a$: $0 \leq x$ **and** $b$: $x \leq 1$
  **shows** $x - x^2 \leq ln\ (1 + x)$
⟨*proof*⟩

**lemma** *ln-one-minus-pos-lower-bound*:
  **fixes** $x$ :: *real*
  **assumes** $a$: $0 \leq x$ **and** $b$: $x \leq 1/2$
  **shows** $- x - 2 * x^2 \leq ln\ (1 - x)$

⟨proof⟩

**lemma** *ln-add-one-self-le-self2*:
  **fixes** $x$ :: *real*
  **shows** $-1 < x \Longrightarrow \ln (1 + x) \leq x$
  ⟨proof⟩

**lemma** *abs-ln-one-plus-x-minus-x-bound-nonneg*:
  **fixes** $x$ :: *real*
  **assumes** $x$: $0 \leq x$ **and** $x1$: $x \leq 1$
  **shows** $|\ln (1 + x) - x| \leq x^2$
⟨proof⟩

**lemma** *abs-ln-one-plus-x-minus-x-bound-nonpos*:
  **fixes** $x$ :: *real*
  **assumes** $a$: $-(1/2) \leq x$ **and** $b$: $x \leq 0$
  **shows** $|\ln (1 + x) - x| \leq 2 * x^2$
⟨proof⟩

**lemma** *abs-ln-one-plus-x-minus-x-bound*:
  **fixes** $x$ :: *real*
  **assumes** $|x| \leq 1/2$
  **shows** $|\ln (1 + x) - x| \leq 2 * x^2$
⟨proof⟩

**lemma** *ln-x-over-x-mono*:
  **fixes** $x$ :: *real*
  **assumes** $x$: $\exp 1 \leq x$ $x \leq y$
  **shows** $\ln y / y \leq \ln x / x$
⟨proof⟩

**lemma** *ln-le-minus-one*: $0 < x \Longrightarrow \ln x \leq x - 1$
  **for** $x$ :: *real*
⟨proof⟩

**corollary** *ln-diff-le*: $0 < x \Longrightarrow 0 < y \Longrightarrow \ln x - \ln y \leq (x - y) / y$
  **for** $x$ :: *real*
⟨proof⟩

**lemma** *ln-add1-ge*:
  **fixes** $t$::*real*
  **shows** $t \geq 0 \Longrightarrow \ln (t+1) \geq t / (1+t)$
⟨proof⟩

**lemma** *ln-eq-minus-one*:
  **fixes** $x$ :: *real*
  **assumes** $0 < x$ $\ln x = x - 1$
  **shows** $x = 1$
⟨proof⟩

**corollary** *ln-diff-less*: $0 < x \Longrightarrow 0 < y \Longrightarrow x \neq y \Longrightarrow ln\ x - ln\ y < (x - y) / y$
**for** $x :: real$
⟨*proof*⟩

**lemma** *ln-add1-gt*:
**fixes** $t::real$
**shows** $t>0 \Longrightarrow ln\ (t+1) > t / (1+t)$
⟨*proof*⟩

**lemma** *ln-add-one-self-less-self*:
**fixes** $x :: real$
**assumes** $x > 0$
**shows** $ln\ (1 + x) < x$
⟨*proof*⟩

**lemma** *ln-x-over-x-tendsto-0*: $((\lambda x::real.\ ln\ x / x) \longrightarrow 0)\ at\text{-}top$
⟨*proof*⟩

**corollary** *exp-1-gt-powr*:
**assumes** $x > (0::real)$
**shows** $exp\ 1 > (1 + 1/x)\ powr\ x$
⟨*proof*⟩

**lemma** *exp-ge-one-plus-x-over-n-power-n*:
**assumes** $x \geq -\ real\ n\ n > 0$
**shows** $(1 + x / of\text{-}nat\ n)\ \hat{}\ n \leq exp\ x$
⟨*proof*⟩

**lemma** *exp-ge-one-minus-x-over-n-power-n*:
**assumes** $x \leq real\ n\ n > 0$
**shows** $(1 - x / of\text{-}nat\ n)\ \hat{}\ n \leq exp\ (-x)$
⟨*proof*⟩

**lemma** *exp-at-bot*: $(exp \longrightarrow (0::real))\ at\text{-}bot$
⟨*proof*⟩

**lemma** *exp-at-top*: $LIM\ x\ at\text{-}top.\ exp\ x :: real :> at\text{-}top$
⟨*proof*⟩

**lemma** *lim-exp-minus-1*: $((\lambda z::'a.\ (exp(z) - 1) / z) \longrightarrow 1)\ (at\ 0)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *ln-at-0*: $LIM\ x\ at\text{-}right\ 0.\ ln\ (x::real) :> at\text{-}bot$
⟨*proof*⟩

**lemma** *ln-at-top*: $LIM\ x\ at\text{-}top.\ ln\ (x::real) :> at\text{-}top$
⟨*proof*⟩

**lemma** *filtermap-ln-at-top*: *filtermap* (*ln*::*real* ⇒ *real*) *at-top* = *at-top*
⟨*proof*⟩

**lemma** *filtermap-exp-at-top*: *filtermap* (*exp*::*real* ⇒ *real*) *at-top* = *at-top*
⟨*proof*⟩

**lemma** *filtermap-ln-at-right*: *filtermap ln* (*at-right* (*0*::*real*)) = *at-bot*
⟨*proof*⟩

**lemma** *tendsto-power-div-exp-0*: ((λ*x*. *x* ^ *k* / *exp x*) ⟶ (*0*::*real*)) *at-top*
⟨*proof*⟩

### 112.7.1 A couple of simple bounds

**lemma** *exp-plus-inverse-exp*:
**fixes** *x*::*real*
**shows** *2* ≤ *exp x* + *inverse* (*exp x*)
⟨*proof*⟩

**lemma** *real-le-x-sinh*:
**fixes** *x*::*real*
**assumes** *0* ≤ *x*
**shows** *x* ≤ (*exp x* − *inverse*(*exp x*)) / *2*
⟨*proof*⟩

**lemma** *real-le-abs-sinh*:
**fixes** *x*::*real*
**shows** *abs x* ≤ *abs*((*exp x* − *inverse*(*exp x*)) / *2*)
⟨*proof*⟩

## 112.8 The general logarithm

**definition** *log* :: *real* ⇒ *real* ⇒ *real*
— logarithm of $x$ to base $a$
**where** *log a x* = *ln x* / *ln a*

**lemma** *log-exp* [*simp*]: *log b* (*exp x*) = *x* / *ln b*
⟨*proof*⟩

**lemma** *tendsto-log* [*tendsto-intros*]:
(*f* ⟶ *a*) *F* ⟹ (*g* ⟶ *b*) *F* ⟹ *0* < *a* ⟹ *a* ≠ *1* ⟹ *b*≠*0* ⟹
((λ*x*. *log* (*f x*) (*g x*)) ⟶ *log a b*) *F*
⟨*proof*⟩

**lemma** *continuous-log*:
**assumes** *continuous F f*
**and** *continuous F g*
**and** *f* (*Lim F* (λ*x*. *x*)) > *0*
**and** *f* (*Lim F* (λ*x*. *x*)) ≠ *1*

**and** *g* (*Lim F* (*λx. x*)) ≠ *0*
**shows** *continuous F* (*λx. log* (*f x*) (*g x*))
⟨*proof*⟩

**lemma** *continuous-at-within-log*[*continuous-intros*]:
**assumes** *continuous* (*at a within s*) *f*
**and** *continuous* (*at a within s*) *g*
**and** *0* < *f a*
**and** *f a* ≠ *1*
**and** *g a* ≠ *0*
**shows** *continuous* (*at a within s*) (*λx. log* (*f x*) (*g x*))
⟨*proof*⟩

**lemma** *continuous-on-log*[*continuous-intros*]:
**assumes** *continuous-on S f continuous-on S g*
**and** ∀ *x*∈*S*. *0* < *f x* ∀ *x*∈*S*. *f x* ≠ *1* ∀ *x*∈*S*. *g x* ≠ *0*
**shows** *continuous-on S* (*λx. log* (*f x*) (*g x*))
⟨*proof*⟩

**lemma** *exp-powr-real*:
**fixes** *x*::*real* **shows** *exp x powr y* = *exp* (*x*∗*y*)
⟨*proof*⟩

**lemma** *powr-one-eq-one* [*simp*]: *1 powr a* = *1*
⟨*proof*⟩

**lemma** *powr-zero-eq-one* [*simp*]: *x powr 0* = (*if x* = *0 then 0 else 1*)
⟨*proof*⟩

**lemma** *powr-eq-one-iff-gen*[*simp*]: *a powr x* = *1* ⟷ *x* = *0* **if** *a* > *0 a* ≠ *1* **for** *a x* :: *real*
⟨*proof*⟩

**lemma** *powr-one-gt-zero-iff* [*simp*]: *x powr 1* = *x* ⟷ *0* ≤ *x*
**for** *x* :: *real*
⟨*proof*⟩
**declare** *powr-one-gt-zero-iff* [*THEN iffD2*, *simp*]

**lemma** *powr-diff*:
**fixes** *w*:: *'a*::{*ln*,*real-normed-field*}
**shows** *w powr* (*z1* − *z2*) = *w powr z1* / *w powr z2*
⟨*proof*⟩

**lemma** *powr-mult*: (*x* ∗ *y*) *powr a* = (*x powr a*) ∗ (*y powr a*)
**for** *a x y* :: *real*
⟨*proof*⟩

**lemma** *prod-powr-distrib*:
**fixes** *x* :: *'a* ⇒ *real*

**shows** $(prod\ x\ I)\ powr\ r = (\prod i{\in}I.\ x\ i\ powr\ r)$
⟨*proof*⟩

**lemma** *powr-ge-zero* [*simp*]: $0 \leq x\ powr\ y$
**for** $x\ y$ :: *real*
⟨*proof*⟩

**lemma** *powr-non-neg*[*simp*]: $\neg\ a\ powr\ x < 0$ **for** $a\ x$::*real*
⟨*proof*⟩

**lemma** *inverse-powr*: $\bigwedge y$::*real*. *inverse* $y$ *powr* $a$ = *inverse* ($y$ *powr* $a$)
⟨*proof*⟩

**lemma** *powr-divide*: $(x\ /\ y)\ powr\ a = (x\ powr\ a)\ /\ (y\ powr\ a)$
**for** $a\ b\ x$ :: *real*
⟨*proof*⟩

**lemma** *powr-add*: $x\ powr\ (a + b) = (x\ powr\ a) * (x\ powr\ b)$
**for** $a\ b\ x$ :: *'a*::{*ln*,*real-normed-field*}
⟨*proof*⟩

**lemma** *powr-mult-base*: $0 \leq x \Longrightarrow x * x\ powr\ y = x\ powr\ (1 + y)$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-mult-base'*: $abs\ x * x\ powr\ y = x\ powr\ (1 + y)$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-powr*: $(x\ powr\ a)\ powr\ b = x\ powr\ (a * b)$
**for** $a\ b\ x$ :: *real*
⟨*proof*⟩

**lemma** *powr-power*:
**fixes** $z$:: *'a*::{*real-normed-field*,*ln*}
**shows** $z \neq 0 \Longrightarrow (z\ powr\ u)\ \hat{}\ n = z\ powr\ (of\text{-}nat\ n * u)$
⟨*proof*⟩

**lemma** *powr-powr-swap*: $(x\ powr\ a)\ powr\ b = (x\ powr\ b)\ powr\ a$
**for** $a\ b\ x$ :: *real*
⟨*proof*⟩

**lemma** *powr-minus*: $x\ powr\ (-\ a) = inverse\ (x\ powr\ a)$
**for** $a\ x$ :: *'a*::{*ln*,*real-normed-field*}
⟨*proof*⟩

**lemma** *powr-minus-divide*: $x\ powr\ (-\ a) = 1/(x\ powr\ a)$
**for** $a\ x$ :: *'a*::{*ln*,*real-normed-field*}
⟨*proof*⟩

**lemma** *powr-sum*:
  **assumes** $x \neq 0$
  **shows** $x$ *powr sum* $f\ A = (\prod y{\in}A.\ x$ *powr* $f\ y)$
⟨*proof*⟩

**lemma** *divide-powr-uminus*: $a\ /\ b$ *powr* $c = a * b$ *powr* $(-\ c)$
  **for** $a\ b\ c$ :: *real*
  ⟨*proof*⟩

**lemma** *powr-less-mono*: $a < b \Longrightarrow 1 < x \Longrightarrow x$ *powr* $a < x$ *powr* $b$
  **for** $a\ b\ x$ :: *real*
  ⟨*proof*⟩

**lemma** *powr-less-cancel*: $x$ *powr* $a < x$ *powr* $b \Longrightarrow 1 < x \Longrightarrow a < b$
  **for** $a\ b\ x$ :: *real*
  ⟨*proof*⟩

**lemma** *powr-less-cancel-iff* [*simp*]: $1 < x \Longrightarrow x$ *powr* $a < x$ *powr* $b \longleftrightarrow a < b$
  **for** $a\ b\ x$ :: *real*
  ⟨*proof*⟩

**lemma** *powr-le-cancel-iff* [*simp*]: $1 < x \Longrightarrow x$ *powr* $a \leq x$ *powr* $b \longleftrightarrow a \leq b$
  **for** $a\ b\ x$ :: *real*
  ⟨*proof*⟩

**lemma** *powr-realpow*: $0 < x \Longrightarrow x$ *powr* (*real* $n$) $= x\hat{}n$
  ⟨*proof*⟩

**lemma** *powr-realpow'*: $(z$ :: *real*$) \geq 0 \Longrightarrow n \neq 0 \Longrightarrow z$ *powr of-nat* $n = z\ \hat{}\ n$
  ⟨*proof*⟩

**lemma** *powr-real-of-int'*:
  **assumes** $x \geq 0\ x \neq 0 \lor n > 0$
  **shows** $x$ *powr real-of-int* $n$ = *power-int* $x\ n$
  ⟨*proof*⟩

**lemma** *exp-minus-ge*:
  **fixes** $x$::*real* **shows** $1 - x \leq exp\ (-x)$
  ⟨*proof*⟩

**lemma** *exp-minus-greater*:
  **fixes** $x$::*real* **shows** $1 - x < exp\ (-x) \longleftrightarrow x \neq 0$
  ⟨*proof*⟩

**lemma** *log-ln*: $ln\ x = log\ (exp\ 1)\ x$
  ⟨*proof*⟩

**lemma** *DERIV-log*:

**assumes** $x > 0$
**shows** $DERIV\ (\lambda y.\ log\ b\ y)\ x :> 1\ /\ (ln\ b * x)$
⟨proof⟩

**lemmas** *DERIV-log*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-log*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemma** *powr-log-cancel* [*simp*]: $0 < a \Longrightarrow a \neq 1 \Longrightarrow 0 < x \Longrightarrow a\ powr\ (log\ a\ x) = x$
⟨proof⟩

**lemma** *log-powr-cancel* [*simp*]: $0 < a \Longrightarrow a \neq 1 \Longrightarrow log\ a\ (a\ powr\ x) = x$
⟨proof⟩

**lemma** *powr-eq-iff*: $[\![y>0;\ a>1]\!] \Longrightarrow a\ powr\ x = y \longleftrightarrow log\ a\ y = x$
⟨proof⟩

**lemma** *log-mult*:
$log\ a\ (x * y) = (if\ x\neq 0 \wedge y\neq 0\ then\ log\ a\ x + log\ a\ y\ else\ 0)$
⟨proof⟩

**lemma** *log-mult-pos*:
$x>0 \Longrightarrow y>0 \Longrightarrow log\ a\ (x * y) = log\ a\ x + log\ a\ y$
⟨proof⟩

**lemma** *log-eq-div-ln-mult-log*:
$0 < b \Longrightarrow b \neq 1 \Longrightarrow 0 < x \Longrightarrow log\ a\ x = (ln\ b/ln\ a) * log\ b\ x$
⟨proof⟩

Base 10 logarithms

**lemma** *log-base-10-eq1*: $0 < x \Longrightarrow log\ 10\ x = (ln\ (exp\ 1)\ /\ ln\ 10) * ln\ x$
⟨proof⟩

**lemma** *log-base-10-eq2*: $0 < x \Longrightarrow log\ 10\ x = (log\ 10\ (exp\ 1)) * ln\ x$
⟨proof⟩

**lemma** *log-one* [*simp*]: $log\ a\ 1 = 0$
⟨proof⟩

**lemma** *log-eq-one* [*simp*]: $0 < a \Longrightarrow a \neq 1 \Longrightarrow log\ a\ a = 1$
⟨proof⟩

**lemma** *log-inverse*: $log\ a\ (inverse\ x) = -\ log\ a\ x$
⟨proof⟩

**lemma** *log-recip*: $log\ a\ (1/x) = -\ log\ a\ x$
⟨proof⟩

**lemma** *log-divide*:
  $log\ a\ (x\ /\ y) = (if\ x{\neq}0 \wedge y{\neq}0\ then\ log\ a\ x - log\ a\ y\ else\ 0)$
  ⟨*proof*⟩

**lemma** *log-divide-pos*:
  $x{>}0 \Longrightarrow y{>}0 \Longrightarrow log\ a\ (x\ /\ y) = log\ a\ x - log\ a\ y$
  ⟨*proof*⟩

**lemma** *powr-gt-zero* [*simp*]: $0 < x\ powr\ a \longleftrightarrow x \neq 0$
  **for** $a\ x :: real$
  ⟨*proof*⟩

**lemma** *powr-nonneg-iff* [*simp*]: $a\ powr\ x \leq 0 \longleftrightarrow a = 0$
  **for** $a\ x{::}real$
  ⟨*proof*⟩

**lemma** *log-add-eq-powr*: $0 < b \Longrightarrow b \neq 1 \Longrightarrow x{\neq}0 \Longrightarrow log\ b\ x + y = log\ b\ (x * b\ powr\ y)$
  **and** *add-log-eq-powr*: $0 < b \Longrightarrow b \neq 1 \Longrightarrow x{\neq}0 \Longrightarrow y + log\ b\ x = log\ b\ (b\ powr\ y * x)$
  **and** *log-minus-eq-powr*: $0 < b \Longrightarrow b \neq 1 \Longrightarrow x{\neq}0 \Longrightarrow log\ b\ x - y = log\ b\ (x * b\ powr\ -y)$
  ⟨*proof*⟩

**lemma** *minus-log-eq-powr*: $0 < b \Longrightarrow b \neq 1 \Longrightarrow x{\neq}0 \Longrightarrow y - log\ b\ x = log\ b\ (b\ powr\ y\ /\ x)$
  ⟨*proof*⟩

**lemma** *log-less-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow log\ a\ x < log\ a\ y \longleftrightarrow x < y$
  ⟨*proof*⟩

**lemma** *log-inj*:
  **assumes** $1 < b$
  **shows** $inj\text{-}on\ (log\ b)\ \{0 <..\}$
⟨*proof*⟩

**lemma** *log-le-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow log\ a\ x \leq log\ a\ y \longleftrightarrow x \leq y$
  ⟨*proof*⟩

**lemma** *log-mono*: $1 < a \Longrightarrow 0 < x \Longrightarrow x \leq y \Longrightarrow log\ a\ x \leq log\ a\ y$
  ⟨*proof*⟩

**lemma** *log-less*: $1 < a \Longrightarrow 0 < x \Longrightarrow x < y \Longrightarrow log\ a\ x < log\ a\ y$
  ⟨*proof*⟩

**lemma** *zero-less-log-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 0 < log\ a\ x \longleftrightarrow 1 < x$

⟨*proof*⟩

**lemma** *zero-le-log-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 0 \leq \log a\ x \longleftrightarrow 1 \leq x$
⟨*proof*⟩

**lemma** *log-less-zero-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow \log a\ x < 0 \longleftrightarrow x < 1$
⟨*proof*⟩

**lemma** *log-le-zero-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow \log a\ x \leq 0 \longleftrightarrow x \leq 1$
⟨*proof*⟩

**lemma** *one-less-log-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 1 < \log a\ x \longleftrightarrow a < x$
⟨*proof*⟩

**lemma** *one-le-log-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow 1 \leq \log a\ x \longleftrightarrow a \leq x$
⟨*proof*⟩

**lemma** *log-less-one-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow \log a\ x < 1 \longleftrightarrow x < a$
⟨*proof*⟩

**lemma** *log-le-one-cancel-iff* [*simp*]: $1 < a \Longrightarrow 0 < x \Longrightarrow \log a\ x \leq 1 \longleftrightarrow x \leq a$
⟨*proof*⟩

**lemma** *le-log-iff*:
**fixes** $b\ x\ y$ :: *real*
**assumes** $1 < b\ x > 0$
**shows** $y \leq \log b\ x \longleftrightarrow b\ powr\ y \leq x$
⟨*proof*⟩

**lemma** *less-log-iff*:
**assumes** $1 < b\ x > 0$
**shows** $y < \log b\ x \longleftrightarrow b\ powr\ y < x$
⟨*proof*⟩

**lemma**
**assumes** $1 < b\ x > 0$
**shows** *log-less-iff*: $\log b\ x < y \longleftrightarrow x < b\ powr\ y$
**and** *log-le-iff*: $\log b\ x \leq y \longleftrightarrow x \leq b\ powr\ y$
⟨*proof*⟩

**lemmas** *powr-le-iff* = *le-log-iff* [*symmetric*]
**and** *powr-less-iff* = *less-log-iff* [*symmetric*]
**and** *less-powr-iff* = *log-less-iff* [*symmetric*]
**and** *le-powr-iff* = *log-le-iff* [*symmetric*]

**lemma** *le-log-of-power*:
**assumes** $b\ \widehat{}\ n \leq m\ 1 < b$
**shows** $n \leq \log b\ m$

⟨proof⟩

**lemma** *le-log2-of-power*: $2 \mathbin{\hat{}} n \leq m \Longrightarrow n \leq \log 2\ m$ **for** $m\ n :: nat$
⟨proof⟩

**lemma** *log-of-power-le*: $\llbracket\ m \leq b \mathbin{\hat{}} n;\ b > 1;\ m > 0\ \rrbracket \Longrightarrow \log b\ (real\ m) \leq n$
⟨proof⟩

**lemma** *log2-of-power-le*: $\llbracket\ m \leq 2 \mathbin{\hat{}} n;\ m > 0\ \rrbracket \Longrightarrow \log 2\ m \leq n$ **for** $m\ n :: nat$
⟨proof⟩

**lemma** *log-of-power-less*: $\llbracket\ m < b \mathbin{\hat{}} n;\ b > 1;\ m > 0\ \rrbracket \Longrightarrow \log b\ (real\ m) < n$
⟨proof⟩

**lemma** *log2-of-power-less*: $\llbracket\ m < 2 \mathbin{\hat{}} n;\ m > 0\ \rrbracket \Longrightarrow \log 2\ m < n$ **for** $m\ n :: nat$
⟨proof⟩

**lemma** *less-log-of-power*:
  **assumes** $b \mathbin{\hat{}} n < m\ 1 < b$
  **shows** $n < \log b\ m$
⟨proof⟩

**lemma** *less-log2-of-power*: $2 \mathbin{\hat{}} n < m \Longrightarrow n < \log 2\ m$ **for** $m\ n :: nat$
  ⟨proof⟩

**lemma** *gr-one-powr*[*simp*]:
  **fixes** $x\ y :: real$ **shows** $\llbracket\ x > 1;\ y > 0\ \rrbracket \Longrightarrow 1 < x\ powr\ y$
  ⟨proof⟩

**lemma** *log-pow-cancel* [*simp*]:
  $a > 0 \Longrightarrow a \neq 1 \Longrightarrow \log a\ (a \mathbin{\hat{}} b) = b$
  ⟨proof⟩

**lemma** *floor-log-eq-powr-iff*: $x > 0 \Longrightarrow b > 1 \Longrightarrow \lfloor \log b\ x \rfloor = k \longleftrightarrow b\ powr\ k \leq x \wedge x < b\ powr\ (k + 1)$
  ⟨proof⟩

**lemma** *floor-log-nat-eq-powr-iff*:
  **fixes** $b\ n\ k :: nat$
  **shows** $\llbracket\ b \geq 2;\ k > 0\ \rrbracket \Longrightarrow floor\ (\log b\ (real\ k)) = n \longleftrightarrow b \hat{} n \leq k \wedge k < b \hat{} (n+1)$
⟨proof⟩

**lemma** *floor-log-nat-eq-if*:
  **fixes** $b\ n\ k :: nat$
  **assumes** $b \hat{} n \leq k\ k < b \hat{} (n+1)\ b \geq 2$
  **shows** $floor\ (\log b\ (real\ k)) = n$
⟨proof⟩

**lemma** *ceiling-log-eq-powr-iff*:
$[\![ x > 0; b > 1 ]\!] \Longrightarrow \lceil log\ b\ x \rceil = int\ k + 1 \longleftrightarrow b\ powr\ k < x \wedge x \leq b\ powr\ (k + 1)$
$\langle proof \rangle$

**lemma** *ceiling-log-nat-eq-powr-iff*:
**fixes** $b\ n\ k :: nat$
**shows** $[\![ b \geq 2; k > 0 ]\!] \Longrightarrow \lceil log\ b\ (real\ k) \rceil = int\ n + 1 \longleftrightarrow (b\hat{}n < k \wedge k \leq b\hat{}(n+1))$
$\langle proof \rangle$

**lemma** *ceiling-log-nat-eq-if*:
**fixes** $b\ n\ k :: nat$
**assumes** $b\hat{}n < k\ k \leq b\hat{}(n+1)\ b \geq 2$
**shows** $\lceil log\ (real\ b)\ (real\ k) \rceil = int\ n + 1$
$\langle proof \rangle$

**lemma** *floor-log2-div2*:
**fixes** $n :: nat$
**assumes** $n \geq 2$
**shows** $\lfloor log\ 2\ (real\ n) \rfloor = \lfloor log\ 2\ (n\ div\ 2) \rfloor + 1$
$\langle proof \rangle$

**lemma** *ceiling-log2-div2*:
**assumes** $n \geq 2$
**shows** $\lceil log\ 2\ (real\ n) \rceil = \lceil log\ 2\ ((n-1)\ div\ 2 + 1) \rceil + 1$
$\langle proof \rangle$

**lemma** *powr-real-of-int*:
$x > 0 \Longrightarrow x\ powr\ real\text{-}of\text{-}int\ n = (if\ n \geq 0\ then\ x\ \hat{}\ nat\ n\ else\ inverse\ (x\ \hat{}\ nat\ (-\ n)))$
$\langle proof \rangle$

**lemma** *powr-numeral* [*simp*]: $0 \leq x \Longrightarrow x\ powr\ (numeral\ n :: real) = x\ \hat{}(numeral\ n)$
$\langle proof \rangle$

**lemma** *powr-int*:
**assumes** $x > 0$
**shows** $x\ powr\ i = (if\ i \geq 0\ then\ x\ \hat{}\ nat\ i\ else\ 1/x\ \hat{}\ nat\ (-i))$
$\langle proof \rangle$

**lemma** *power-of-nat-log-ge*: $b > 1 \Longrightarrow b\ \hat{}\ nat\ \lceil log\ b\ x \rceil \geq x$
$\langle proof \rangle$

**lemma** *power-of-nat-log-le*:
**assumes** $b > 1\ x \geq 1$
**shows** $b\ \hat{}\ nat\ \lfloor log\ b\ x \rfloor \leq x$
$\langle proof \rangle$

**definition** *powr-real* :: *real* ⇒ *real* ⇒ *real*
**where** [*code-abbrev*, *simp*]: *powr-real* = *Transcendental.powr*

**lemma** *compute-powr-real* [*code*]:
*powr-real b i* =
(*if* $b \leq 0$ *then Code.abort* (*STR* ''*powr-real with nonpositive base*'') ($\lambda$-. *powr-real b i*)
*else if* $\lfloor i \rfloor = i$ *then* (*if* $0 \leq i$ *then* $b$ ^ *nat* $\lfloor i \rfloor$ *else* $1 / b$ ^ *nat* $\lfloor - i \rfloor$)
*else Code.abort* (*STR* ''*powr-real with non−integer exponent*'') ($\lambda$-. *powr-real b i*))
**for** *b i* :: *real*
⟨*proof*⟩

**lemma** *powr-one*: $0 \leq x \Longrightarrow x$ *powr* $1 = x$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *powr-one'* [*simp*]: $x$ *powr* $1 = |x|$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *powr-neg-one*: $0 < x \Longrightarrow x$ *powr* $-1 = 1/x$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *powr-neg-one'* [*simp*]: $x$ *powr* $-1 = 1/|x|$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *powr-neg-numeral*: $0 < x \Longrightarrow x$ *powr* $-$ *numeral n* $= 1/x$ ^ *numeral n*
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *root-powr-inverse*: $0 < n \Longrightarrow 0 \leq x \Longrightarrow$ *root n x* $= x$ *powr* $(1/n)$
⟨*proof*⟩

**lemma** *powr-inverse-root*: $0 < n \Longrightarrow x$ *powr* $(1/n) =$ |*root n x*|
⟨*proof*⟩

**lemma** *ln-powr* [*simp*]: *ln* ($x$ *powr* $y$) $= y * $ *ln* $x$
**for** *x* :: *real*
⟨*proof*⟩

**lemma** *ln-root*: $n > 0 \Longrightarrow$ *ln* (*root n b*) = *ln b* / *n*
⟨*proof*⟩

**lemma** *ln-sqrt*: $0 \leq x \Longrightarrow$ *ln* (*sqrt x*) = *ln x* / *2*
⟨*proof*⟩

**lemma** *log-root*: $n > 0 \Longrightarrow a \geq 0 \Longrightarrow log\ b\ (root\ n\ a) = \ log\ b\ a\ /\ n$
⟨*proof*⟩

**lemma** *log-powr*: $log\ b\ (x\ powr\ y) = y * log\ b\ x$
⟨*proof*⟩

**lemma** *log-nat-power*: $0 \leq x \Longrightarrow log\ b\ (x\hat{}n) = real\ n * log\ b\ x$
⟨*proof*⟩

**lemma** *log-of-power-eq*:
**assumes** $m = b\ \hat{}\ n\ b > 1$
**shows** $n = log\ b\ (real\ m)$
⟨*proof*⟩

**lemma** *log2-of-power-eq*: $m = 2\ \hat{}\ n \Longrightarrow n = log\ 2\ m$ **for** $m\ n :: nat$
⟨*proof*⟩

**lemma** *log-base-change*: $0 < a \Longrightarrow a \neq 1 \Longrightarrow log\ b\ x = log\ a\ x\ /\ log\ a\ b$
⟨*proof*⟩

**lemma** *log-base-pow*: $0 < a \Longrightarrow log\ (a\ \hat{}\ n)\ x = log\ a\ x\ /\ n$
⟨*proof*⟩

**lemma** *log-base-powr*: $a \neq 0 \Longrightarrow log\ (a\ powr\ b)\ x = log\ a\ x\ /\ b$
⟨*proof*⟩

**lemma** *log-base-root*: $n > 0 \Longrightarrow log\ (root\ n\ b)\ x = n * (log\ b\ x)$
⟨*proof*⟩

**lemma** *ln-bound*: $0 < x \Longrightarrow ln\ x \leq x$ **for** $x :: real$
⟨*proof*⟩

**lemma** *powr-less-one*:
**fixes** $x::real$
**assumes** $1 < x\ y < 0$
**shows** $x\ powr\ y < 1$
⟨*proof*⟩

**lemma** *powr-le-one-le*: $\bigwedge x\ y::real.\ 0 < x \Longrightarrow x \leq 1 \Longrightarrow 1 \leq y \Longrightarrow x\ powr\ y \leq x$
⟨*proof*⟩

**lemma** *powr-mono*:
**fixes** $x :: real$
**assumes** $a \leq b$ **and** $1 \leq x$ **shows** $x\ powr\ a \leq x\ powr\ b$
⟨*proof*⟩

**lemma** *ge-one-powr-ge-zero*: $1 \leq x \Longrightarrow 0 \leq a \Longrightarrow 1 \leq x\ powr\ a$

**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-less-mono2*: $0 < a \Longrightarrow 0 \leq x \Longrightarrow x < y \Longrightarrow x$ *powr* $a < y$ *powr* $a$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-less-mono2-neg*: $a < 0 \Longrightarrow 0 < x \Longrightarrow x < y \Longrightarrow y$ *powr* $a < x$ *powr* $a$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-mono2*: $x$ *powr* $a \leq y$ *powr* $a$ **if** $0 \leq a$ $0 \leq x$ $x \leq y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-less-cancel2*: $0 < a \Longrightarrow 0 < x \Longrightarrow 0 < y \Longrightarrow x$ *powr* $a < y$ *powr* $a \Longrightarrow x < y$
**for** $a$ $x$ $y$ ::*real*
⟨*proof*⟩

**lemma** *powr01-less-one*:
**fixes** $x$::*real*
**assumes** $0 < x$ $x < 1$
**shows** $x$ *powr* $a < 1 \longleftrightarrow a>0$
⟨*proof*⟩

**lemma** *powr-le1*: $0 \leq a \Longrightarrow |x| \leq 1 \Longrightarrow x$ *powr* $a \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-mono2'*:
**fixes** $a$ $x$ $y$ :: *real*
**assumes** $a \leq 0$ $x > 0$ $x \leq y$
**shows** $x$ *powr* $a \geq y$ *powr* $a$
⟨*proof*⟩

**lemma** *powr-mono'*: $a \leq (b$::*real*$) \Longrightarrow x \geq 0 \Longrightarrow x \leq 1 \Longrightarrow x$ *powr* $b \leq x$ *powr* $a$
⟨*proof*⟩

**lemma** *powr-mono-both*:
**fixes** $x$ :: *real*
**assumes** $0 \leq a$ $a \leq b$ $1 \leq x$ $x \leq y$
**shows** $x$ *powr* $a \leq y$ *powr* $b$
⟨*proof*⟩

**lemma** *powr-mono-both'*:
**fixes** $x$ :: *real*
**assumes** $a \geq b$ $b \geq 0$ $0 < x$ $x \leq y$ $y \leq 1$

**shows** $x$ *powr* $a \leq y$ *powr* $b$
⟨*proof*⟩

**lemma** *powr-less-mono'*:
**assumes** $(x::real) > 0$ $x < 1$ $a < b$
**shows** $x$ *powr* $b < x$ *powr* $a$
⟨*proof*⟩

**lemma** *powr-inj*: $0 < a \Longrightarrow a \neq 1 \Longrightarrow a$ *powr* $x = a$ *powr* $y \longleftrightarrow x = y$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *powr-half-sqrt*: $0 \leq x \Longrightarrow x$ *powr* $(1/2) =$ *sqrt* $x$
⟨*proof*⟩

**lemma** *powr-half-sqrt-powr*: $0 \leq x \Longrightarrow x$ *powr* $(a/2) =$ *sqrt*($x$ *powr* $a$)
⟨*proof*⟩

**lemma** *square-powr-half* [*simp*]:
**fixes** $x$::*real* **shows** $x^2$ *powr* $(1/2) = |x|$
⟨*proof*⟩

**lemma** *ln-powr-bound*: $1 \leq x \Longrightarrow 0 < a \Longrightarrow$ *ln* $x \leq (x$ *powr* $a)$ / $a$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *ln-powr-bound2*:
**fixes** $x$ :: *real*
**assumes** $1 < x$ **and** $0 < a$
**shows** (*ln* $x$) *powr* $a \leq (a$ *powr* $a) * x$
⟨*proof*⟩

**lemma** *tendsto-powr*:
**fixes** $a$ $b$ :: *real*
**assumes** $f$: $(f \longrightarrow a)$ $F$
**and** $g$: $(g \longrightarrow b)$ $F$
**and** $a$: $a \neq 0$
**shows** $((\lambda x.\ f\ x$ *powr* $g\ x) \longrightarrow a$ *powr* $b)$ $F$
⟨*proof*⟩

**lemma** *tendsto-powr'*[*tendsto-intros*]:
**fixes** $a$ :: *real*
**assumes** $f$: $(f \longrightarrow a)$ $F$
**and** $g$: $(g \longrightarrow b)$ $F$
**and** $a$: $a \neq 0 \lor (b > 0 \land$ *eventually* $(\lambda x.\ f\ x \geq 0)$ $F)$
**shows** $((\lambda x.\ f\ x$ *powr* $g\ x) \longrightarrow a$ *powr* $b)$ $F$
⟨*proof*⟩

**lemma** *continuous-powr*:

**assumes** *continuous F f*
**and** *continuous F g*
**and** *f* (*Lim F* ($\lambda x.$ *x*)) $\neq$ *0*
**shows** *continuous F* ($\lambda x.$ (*f x*) *powr* (*g x* :: *real*))
⟨*proof*⟩

**lemma** *continuous-at-within-powr*[*continuous-intros*]:
**fixes** *f g* :: - $\Rightarrow$ *real*
**assumes** *continuous* (*at a within s*) *f*
**and** *continuous* (*at a within s*) *g*
**and** *f a* $\neq$ *0*
**shows** *continuous* (*at a within s*) ($\lambda x.$ (*f x*) *powr* (*g x*))
⟨*proof*⟩

**lemma** *continuous-on-powr*[*continuous-intros*]:
**fixes** *f g* :: - $\Rightarrow$ *real*
**assumes** *continuous-on s f continuous-on s g* **and** $\forall x \in s.$ *f x* $\neq$ *0*
**shows** *continuous-on s* ($\lambda x.$ (*f x*) *powr* (*g x*))
⟨*proof*⟩

**lemma** *tendsto-powr2*:
**fixes** *a* :: *real*
**assumes** *f*: (*f* $\longrightarrow$ *a*) *F*
**and** *g*: (*g* $\longrightarrow$ *b*) *F*
**and** $\forall_F$ *x in F*. *0* $\leq$ *f x*
**and** *b*: *0* < *b*
**shows** (($\lambda x.$ *f x powr g x*) $\longrightarrow$ *a powr b*) *F*
⟨*proof*⟩

**lemma** *has-derivative-powr*[*derivative-intros*]:
**assumes** *g*[*derivative-intros*]: (*g has-derivative g'*) (*at x within X*)
**and** *f*[*derivative-intros*]:(*f has-derivative f'*) (*at x within X*)
**assumes** *pos*: *0* < *g x* **and** $x \in X$
**shows** (($\lambda x.$ *g x powr f x*::*real*) *has-derivative* ($\lambda h.$ (*g x powr f x*) $*$ (*f' h* $*$ *ln* (*g x*) + *g' h* $*$ *f x* / *g x*))) (*at x within X*)
⟨*proof*⟩

**lemma** *has-derivative-const-powr* [*derivative-intros*]:
**fixes** *a*::*real*
**assumes** $\bigwedge x.$ (*f has-derivative f'*) (*at x*)
**shows** (($\lambda x.$ *a powr* (*f x*)) *has-derivative* ($\lambda y.$ *f' y* $*$ *ln a* $*$ *a powr* (*f x*))) (*at x*)
⟨*proof*⟩

**lemma** *has-real-derivative-const-powr* [*derivative-intros*]:
**fixes** *a*::*real*
**assumes** $\bigwedge x.$ (*f has-real-derivative f' x*) (*at x*)
**shows** (($\lambda x.$ *a powr* (*f x*)) *has-real-derivative* (*f' x* $*$ *ln a* $*$ *a powr* (*f x*))) (*at x*)
⟨*proof*⟩

**lemma** *DERIV-powr*:
**fixes** $r$ :: *real*
**assumes** $g$: *DERIV* $g\ x$ :> $m$
**and** *pos*: $g\ x > 0$
**and** $f$: *DERIV* $f\ x$ :> $r$
**shows** *DERIV* $(\lambda x.\ g\ x$ *powr* $f\ x)\ x$ :> $(g\ x$ *powr* $f\ x) * (r * ln\ (g\ x) + m * f\ x\ /\ g\ x)$
⟨*proof*⟩

**lemma** *DERIV-fun-powr*:
**fixes** $r$ :: *real*
**assumes** $g$: *DERIV* $g\ x$ :> $m$
**and** *pos*: $g\ x > 0$
**shows** *DERIV* $(\lambda x.\ (g\ x)$ *powr* $r)\ x$ :> $r * (g\ x)$ *powr* $(r - $ *of-nat* $1) * m$
⟨*proof*⟩

**lemma** *has-real-derivative-powr*:
**assumes** $z > 0$
**shows** $((\lambda z.\ z$ *powr* $r)$ *has-real-derivative* $r * z$ *powr* $(r - 1))$ $(at\ z)$
⟨*proof*⟩

**declare** *has-real-derivative-powr*[*THEN DERIV-chain2*, *derivative-intros*]

A more general version, by Johannes Hölzl

**lemma** *has-real-derivative-powr'*:
**fixes** $f\ g$ :: *real* ⇒ *real*
**assumes** ($f$ *has-real-derivative* $f'$) $(at\ x)$
**assumes** ($g$ *has-real-derivative* $g'$) $(at\ x)$
**assumes** $f\ x > 0$
**defines** $h \equiv \lambda x.\ f\ x$ *powr* $g\ x * (g' * ln\ (f\ x) + f' * g\ x\ /\ f\ x)$
**shows** $((\lambda x.\ f\ x$ *powr* $g\ x)$ *has-real-derivative* $h\ x)$ $(at\ x)$
⟨*proof*⟩

**lemma** *tendsto-zero-powrI*:
**assumes** $(f \longrightarrow (0::real))\ F$ $(g \longrightarrow b)\ F$ $\forall_F\ x$ *in* $F.\ 0 \leq f\ x$ $0 < b$
**shows** $((\lambda x.\ f\ x$ *powr* $g\ x) \longrightarrow 0)\ F$
⟨*proof*⟩

**lemma** *continuous-on-powr'*:
**fixes** $f\ g$ :: - ⇒ *real*
**assumes** *continuous-on* $s\ f$ *continuous-on* $s\ g$
**and** $\forall x \in s.\ f\ x \geq 0 \wedge (f\ x = 0 \longrightarrow g\ x > 0)$
**shows** *continuous-on* $s$ $(\lambda x.\ (f\ x)$ *powr* $(g\ x))$
⟨*proof*⟩

**lemma** *tendsto-neg-powr*:
**assumes** $s < 0$
**and** $f$: *LIM* $x\ F.\ f\ x$ :> *at-top*
**shows** $((\lambda x.\ f\ x$ *powr* $s) \longrightarrow (0::real))\ F$

⟨*proof*⟩

**lemma** *tendsto-exp-limit-at-right*: $((\lambda y.\ (1 + x * y)\ powr\ (1\ /\ y)) \longrightarrow exp\ x)$ $(at\text{-}right\ 0)$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *tendsto-exp-limit-at-top*: $((\lambda y.\ (1 + x\ /\ y)\ powr\ y) \longrightarrow exp\ x)\ at\text{-}top$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *tendsto-exp-limit-sequentially*: $(\lambda n.\ (1 + x\ /\ n)\ \widehat{}\ n) \longrightarrow exp\ x$
**for** $x$ :: *real*
⟨*proof*⟩

## 112.9 Sine and Cosine

**definition** *sin-coeff* :: *nat* ⇒ *real*
**where** $sin\text{-}coeff = (\lambda n.\ if\ even\ n\ then\ 0\ else\ (-\ 1)\ \widehat{}\ ((n - Suc\ 0)\ div\ 2)\ /\ (fact\ n))$

**definition** *cos-coeff* :: *nat* ⇒ *real*
**where** $cos\text{-}coeff = (\lambda n.\ if\ even\ n\ then\ ((-\ 1)\ \widehat{}\ (n\ div\ 2))\ /\ (fact\ n)\ else\ 0)$

**definition** *sin* :: $'a \Rightarrow 'a::\{real\text{-}normed\text{-}algebra\text{-}1,banach\}$
**where** $sin = (\lambda x.\ \sum n.\ sin\text{-}coeff\ n *_R x\widehat{}n)$

**definition** *cos* :: $'a \Rightarrow 'a::\{real\text{-}normed\text{-}algebra\text{-}1,banach\}$
**where** $cos = (\lambda x.\ \sum n.\ cos\text{-}coeff\ n *_R x\widehat{}n)$

**lemma** *sin-coeff-0* [*simp*]: *sin-coeff 0 = 0*
⟨*proof*⟩

**lemma** *cos-coeff-0* [*simp*]: *cos-coeff 0 = 1*
⟨*proof*⟩

**lemma** *sin-coeff-Suc*: *sin-coeff (Suc n) = cos-coeff n / real (Suc n)*
⟨*proof*⟩

**lemma** *cos-coeff-Suc*: *cos-coeff (Suc n) = − sin-coeff n / real (Suc n)*
⟨*proof*⟩

**lemma** *summable-norm-sin*: $summable\ (\lambda n.\ norm\ (sin\text{-}coeff\ n *_R x\widehat{}n))$
**for** $x$ :: $'a::\{real\text{-}normed\text{-}algebra\text{-}1,banach\}$
⟨*proof*⟩

**lemma** *summable-norm-cos*: $summable\ (\lambda n.\ norm\ (cos\text{-}coeff\ n *_R x\widehat{}n))$
**for** $x$ :: $'a::\{real\text{-}normed\text{-}algebra\text{-}1,banach\}$
⟨*proof*⟩

**lemma** *sin-converges*: $(\lambda n.\ sin\text{-}coeff\ n *_R x\hat{\ }n)$ *sums sin x*
⟨*proof*⟩

**lemma** *cos-converges*: $(\lambda n.\ cos\text{-}coeff\ n *_R x\hat{\ }n)$ *sums cos x*
⟨*proof*⟩

**lemma** *sin-of-real*: *sin* (*of-real x*) = *of-real* (*sin x*)
**for** *x* :: *real*
⟨*proof*⟩

**corollary** *sin-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow sin\ z \in \mathbb{R}$
⟨*proof*⟩

**lemma** *cos-of-real*: *cos* (*of-real x*) = *of-real* (*cos x*)
**for** *x* :: *real*
⟨*proof*⟩

**corollary** *cos-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow cos\ z \in \mathbb{R}$
⟨*proof*⟩

**lemma** *diffs-sin-coeff*: *diffs sin-coeff* = *cos-coeff*
⟨*proof*⟩

**lemma** *diffs-cos-coeff*: *diffs cos-coeff* = ($\lambda n.$ − *sin-coeff n*)
⟨*proof*⟩

**lemma** *sin-int-times-real*: *sin* (*of-int m* ∗ *of-real x*) = *of-real* (*sin* (*of-int m* ∗ *x*))
⟨*proof*⟩

**lemma** *cos-int-times-real*: *cos* (*of-int m* ∗ *of-real x*) = *of-real* (*cos* (*of-int m* ∗ *x*))
⟨*proof*⟩

Now at last we can get the derivatives of exp, sin and cos.

**lemma** *DERIV-sin* [*simp*]: *DERIV sin x* :> *cos x*
**for** *x* :: *'a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**declare** *DERIV-sin*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-sin*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemmas** *has-derivative-sin*[*derivative-intros*] = *DERIV-sin*[*THEN DERIV-compose-FDERIV*]

**lemma** *DERIV-cos* [*simp*]: *DERIV cos x* :> − *sin x*
**for** *x* :: *'a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**declare** *DERIV-cos*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-cos*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemmas** *has-derivative-cos*[*derivative-intros*] = *DERIV-cos*[*THEN DERIV-compose-FDERIV*]

**lemma** *isCont-sin*: *isCont sin x*
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *continuous-on-sin-real*: *continuous-on* $\{a..b\}$ *sin* **for** $a::real$
⟨*proof*⟩

**lemma** *isCont-cos*: *isCont cos x*
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *continuous-on-cos-real*: *continuous-on* $\{a..b\}$ *cos* **for** $a::real$
⟨*proof*⟩

**context**
**fixes** $f :: 'a::t2\text{-}space \Rightarrow 'b::\{real\text{-}normed\text{-}field,banach\}$
**begin**

**lemma** *isCont-sin'* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* $(\lambda x.\ sin\ (f\ x))\ a$
⟨*proof*⟩

**lemma** *isCont-cos'* [*simp*]: *isCont f a* $\Longrightarrow$ *isCont* $(\lambda x.\ cos\ (f\ x))\ a$
⟨*proof*⟩

**lemma** *tendsto-sin* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ sin\ (f\ x)) \longrightarrow sin\ a)\ F$
⟨*proof*⟩

**lemma** *tendsto-cos* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow ((\lambda x.\ cos\ (f\ x)) \longrightarrow cos\ a)\ F$
⟨*proof*⟩

**lemma** *continuous-sin* [*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ sin\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-sin* [*continuous-intros*]: *continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ sin\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-cos* [*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ cos\ (f\ x))$

⟨*proof*⟩

**lemma** *continuous-on-cos* [*continuous-intros*]: *continuous-on* $s\,f \Longrightarrow$ *continuous-on* $s$ ($\lambda x.$ *cos* ($f$ $x$))
⟨*proof*⟩

**end**

**lemma** *continuous-within-sin*: *continuous* (*at* $z$ *within* $s$) *sin*
**for** $z$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *continuous-within-cos*: *continuous* (*at* $z$ *within* $s$) *cos*
**for** $z$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

## 112.10 Properties of Sine and Cosine

**lemma** *sin-zero* [*simp*]: *sin 0 = 0*
⟨*proof*⟩

**lemma** *cos-zero* [*simp*]: *cos 0 = 1*
⟨*proof*⟩

**lemma** *DERIV-fun-sin*: *DERIV* $g$ $x$ :> $m \Longrightarrow$ *DERIV* ($\lambda x.$ *sin* ($g$ $x$)) $x$ :> *cos* ($g$ $x$) $*$ $m$
⟨*proof*⟩

**lemma** *DERIV-fun-cos*: *DERIV* $g$ $x$ :> $m \Longrightarrow$ *DERIV* ($\lambda x.$ *cos*($g$ $x$)) $x$ :> $-$ *sin* ($g$ $x$) $*$ $m$
⟨*proof*⟩

## 112.11 Deriving the Addition Formulas

The product of two cosine series.

**lemma** *cos-x-cos-y*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**shows**
($\lambda p. \sum n \le p.$
*if even* $p \wedge$ *even* $n$
*then* ((−*1*) ^ (*p div 2*) ∗ (*p choose n*) / (*fact p*)) $*_R$ ($x$^$n$) ∗ $y$^($p-n$) *else 0*)
*sums* (*cos* $x$ ∗ *cos* $y$)
⟨*proof*⟩

The product of two sine series.

**lemma** *sin-x-sin-y*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**shows**

$(\lambda p.\ \sum n{\le}p.$
$\quad$ *if even p* $\wedge$ *odd n*
$\quad$ *then* $-$ $((-1)\ \hat{}\ (p\ div\ 2) * (p\ choose\ n)\ /\ (fact\ p)) *_R (x\hat{}n) * y\hat{}(p-n)$
$\quad$ *else* $0)$
*sums* $(sin\ x * sin\ y)$
⟨*proof*⟩

**lemma** *sums-cos-x-plus-y*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**shows**
$(\lambda p.\ \sum n{\le}p.$
$\quad$ *if even p*
$\quad$ *then* $((-1)\ \hat{}\ (p\ div\ 2) * (p\ choose\ n)\ /\ (fact\ p)) *_R (x\hat{}n) * y\hat{}(p-n)$
$\quad$ *else* $0)$
*sums cos* $(x + y)$
⟨*proof*⟩

**theorem** *cos-add*:
**fixes** $x$ :: $'a$::{*real-normed-field*,*banach*}
**shows** $cos\ (x + y) = cos\ x * cos\ y - sin\ x * sin\ y$
⟨*proof*⟩

**lemma** *sin-minus-converges*: $(\lambda n.\ -\ (sin\text{-}coeff\ n *_R (-x)\hat{}n))$ *sums sin x*
⟨*proof*⟩

**lemma** *sin-minus* [*simp*]: $sin\ (-\ x) = -\ sin\ x$
**for** $x$ :: $'a$::{*real-normed-algebra-1*,*banach*}
⟨*proof*⟩

**lemma** *cos-minus-converges*: $(\lambda n.\ (cos\text{-}coeff\ n *_R (-x)\hat{}n))$ *sums cos x*
⟨*proof*⟩

**lemma** *cos-minus* [*simp*]: $cos\ (-x) = cos\ x$
**for** $x$ :: $'a$::{*real-normed-algebra-1*,*banach*}
⟨*proof*⟩

**lemma** *cos-abs-real* [*simp*]: $cos\ |x :: real| = cos\ x$
⟨*proof*⟩

**lemma** *sin-cos-squared-add* [*simp*]: $(sin\ x)^2 + (cos\ x)^2 = 1$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-cos-squared-add2* [*simp*]: $(cos\ x)^2 + (sin\ x)^2 = 1$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-cos-squared-add3* [*simp*]: $cos\ x * cos\ x + sin\ x * sin\ x = 1$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}

⟨*proof*⟩

**lemma** *sin-squared-eq*: $(sin\ x)^2 = 1 - (cos\ x)^2$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *cos-squared-eq*: $(cos\ x)^2 = 1 - (sin\ x)^2$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *abs-sin-le-one* [*simp*]: $|sin\ x| \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *sin-ge-minus-one* [*simp*]: $-\ 1 \leq sin\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *sin-le-one* [*simp*]: $sin\ x \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *abs-cos-le-one* [*simp*]: $|cos\ x| \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *cos-ge-minus-one* [*simp*]: $-\ 1 \leq cos\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *cos-le-one* [*simp*]: $cos\ x \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *cos-diff*: $cos\ (x - y) = cos\ x * cos\ y + sin\ x * sin\ y$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *cos-double*: $cos(2{*}x) = (cos\ x)^2 - (sin\ x)^2$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-cos-le1*: $|sin\ x * sin\ y + cos\ x * cos\ y| \leq 1$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *DERIV-fun-pow*: $DERIV\ g\ x :> m \Longrightarrow DERIV\ (\lambda x.\ (g\ x)\ \hat{}\ n)\ x :> real$
$n * (g\ x)\ \hat{}(n - 1) * m$
⟨*proof*⟩

**lemma** *DERIV-fun-exp*: $DERIV\ g\ x :> m \Longrightarrow DERIV\ (\lambda x.\ exp\ (g\ x))\ x :> exp\ (g\ x) * m$
⟨*proof*⟩

## 112.12 The Constant Pi

**definition** $pi :: real$
**where** $pi = 2 * (THE\ x.\ 0 \leq x \wedge x \leq 2 \wedge cos\ x = 0)$

Show that there's a least positive $x$ with $cos\ x = 0$; hence define pi.

**lemma** *sin-paired*: $(\lambda n.\ (-\ 1)\ \hat{}\ n\ /\ (fact\ (2 * n + 1)) * x\ \hat{}\ (2 * n + 1))\ sums\ sin\ x$
**for** $x :: real$
⟨*proof*⟩

**lemma** *sin-gt-zero-02*:
**fixes** $x :: real$
**assumes** $0 < x$ **and** $x < 2$
**shows** $0 < sin\ x$
⟨*proof*⟩

**lemma** *cos-double-less-one*: $0 < x \Longrightarrow x < 2 \Longrightarrow cos\ (2 * x) < 1$
**for** $x :: real$
⟨*proof*⟩

**lemma** *cos-paired*: $(\lambda n.\ (-\ 1)\ \hat{}\ n\ /\ (fact\ (2 * n)) * x\ \hat{}\ (2 * n))\ sums\ cos\ x$
**for** $x :: real$
⟨*proof*⟩

**lemma** *sum-pos-lt-pair*:
**fixes** $f :: nat \Rightarrow real$
**assumes** *f*: *summable f* **and** *fplus*: $\bigwedge d.\ 0 < f\ (k + (Suc(Suc\ 0) * d)) + f\ (k + ((Suc\ (Suc\ 0) * d) + 1))$
**shows** $sum\ f\ \{..<k\} < suminf\ f$
⟨*proof*⟩

**lemma** *cos-two-less-zero* [*simp*]: $cos\ 2 < (0::real)$
⟨*proof*⟩

**lemmas** *cos-two-neq-zero* [*simp*] = *cos-two-less-zero* [*THEN less-imp-neq*]
**lemmas** *cos-two-le-zero* [*simp*] = *cos-two-less-zero* [*THEN order-less-imp-le*]

**lemma** *cos-is-zero*: $\exists!x::real.\ 0 \leq x \wedge x \leq 2 \wedge cos\ x = 0$
⟨*proof*⟩

**lemma** *pi-half*: $pi/2 = (THE\ x.\ 0 \leq x \wedge x \leq 2 \wedge cos\ x = 0)$
⟨*proof*⟩

**lemma** *cos-pi-half* [*simp*]: *cos* (*pi*/*2*) = *0*
⟨*proof*⟩

**lemma** *cos-of-real-pi-half* [*simp*]: *cos* ((*of-real pi*/*2*) :: *'a*) = *0*
**if** *SORT-CONSTRAINT*(*'a*::{*real-field*,*banach*,*real-normed-algebra-1*})
⟨*proof*⟩

**lemma** *pi-half-gt-zero* [*simp*]: *0* < *pi*/*2*
⟨*proof*⟩

**lemmas** *pi-half-neq-zero* [*simp*] = *pi-half-gt-zero* [*THEN less-imp-neq*, *symmetric*]
**lemmas** *pi-half-ge-zero* [*simp*] = *pi-half-gt-zero* [*THEN order-less-imp-le*]

**lemma** *pi-half-less-two* [*simp*]: *pi*/*2* < *2*
⟨*proof*⟩

**lemmas** *pi-half-neq-two* [*simp*] = *pi-half-less-two* [*THEN less-imp-neq*]
**lemmas** *pi-half-le-two* [*simp*] = *pi-half-less-two* [*THEN order-less-imp-le*]

**lemma** *pi-gt-zero* [*simp*]: *0* < *pi*
⟨*proof*⟩

**lemma** *pi-ge-zero* [*simp*]: *0* ≤ *pi*
⟨*proof*⟩

**lemma** *pi-neq-zero* [*simp*]: *pi* ≠ *0*
⟨*proof*⟩

**lemma** *pi-not-less-zero* [*simp*]: ¬ *pi* < *0*
⟨*proof*⟩

**lemma** *minus-pi-half-less-zero*: −(*pi*/*2*) < *0*
⟨*proof*⟩

**lemma** *m2pi-less-pi*: − (*2*∗*pi*) < *pi*
⟨*proof*⟩

**lemma** *sin-pi-half* [*simp*]: *sin*(*pi*/*2*) = *1*
⟨*proof*⟩

**lemma** *sin-of-real-pi-half* [*simp*]: *sin* ((*of-real pi*/*2*) :: *'a*) = *1*
**if** *SORT-CONSTRAINT*(*'a*::{*real-field*,*banach*,*real-normed-algebra-1*})
⟨*proof*⟩

**lemma** *sin-cos-eq*: *sin x* = *cos* (*of-real pi*/*2* − *x*)
**for** *x* :: *'a*::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *minus-sin-cos-eq*: − *sin x* = *cos* (*x* + *of-real pi*/*2*)

**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *cos-sin-eq*: $cos\ x = sin\ (of\text{-}real\ pi/2 - x)$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-add*: $sin\ (x + y) = sin\ x * cos\ y + cos\ x * sin\ y$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-diff*: $sin\ (x - y) = sin\ x * cos\ y - cos\ x * sin\ y$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *sin-double*: $sin(2 * x) = 2 * sin\ x * cos\ x$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *cos-of-real-pi* [*simp*]: $cos\ (of\text{-}real\ pi) = -1$
⟨*proof*⟩

**lemma** *sin-of-real-pi* [*simp*]: $sin\ (of\text{-}real\ pi) = 0$
⟨*proof*⟩

**lemma** *cos-pi* [*simp*]: $cos\ pi = -1$
⟨*proof*⟩

**lemma** *sin-pi* [*simp*]: $sin\ pi = 0$
⟨*proof*⟩

**lemma** *sin-periodic-pi* [*simp*]: $sin\ (x + pi) = -\ sin\ x$
⟨*proof*⟩

**lemma** *sin-periodic-pi2* [*simp*]: $sin\ (pi + x) = -\ sin\ x$
⟨*proof*⟩

**lemma** *cos-periodic-pi* [*simp*]: $cos\ (x + pi) = -\ cos\ x$
⟨*proof*⟩

**lemma** *cos-periodic-pi2* [*simp*]: $cos\ (pi + x) = -\ cos\ x$
⟨*proof*⟩

**lemma** *sin-periodic* [*simp*]: $sin\ (x + 2 * pi) = sin\ x$
⟨*proof*⟩

**lemma** *cos-periodic* [*simp*]: $cos\ (x + 2 * pi) = cos\ x$
⟨*proof*⟩

**lemma** *cos-npi* [*simp*]: *cos* (*real n* ∗ *pi*) = (− *1*) ^ *n*
⟨*proof*⟩

**lemma** *cos-npi2* [*simp*]: *cos* (*pi* ∗ *real n*) = (− *1*) ^ *n*
⟨*proof*⟩

**lemma** *sin-npi* [*simp*]: *sin* (*real n* ∗ *pi*) = *0*
**for** *n* :: *nat*
⟨*proof*⟩

**lemma** *sin-npi2* [*simp*]: *sin* (*pi* ∗ *real n*) = *0*
**for** *n* :: *nat*
⟨*proof*⟩

**lemma** *sin-npi-numeral* [*simp*]: *sin*(*Num.numeral n* ∗ *pi*) = *0*
⟨*proof*⟩

**lemma** *sin-npi2-numeral* [*simp*]: *sin* (*pi* ∗ *Num.numeral n*) = *0*
⟨*proof*⟩

**lemma** *sin-npi-complex'* [*simp*]: *sin* (*of-nat n* ∗ *of-real pi*) = *0*
⟨*proof*⟩

**lemma** *cos-npi-numeral* [*simp*]: *cos* (*Num.numeral n* ∗ *pi*) = (− *1*) ^*Num.numeral n*
⟨*proof*⟩

**lemma** *cos-npi2-numeral* [*simp*]: *cos* (*pi* ∗ *Num.numeral n*) = (− *1*) ^*Num.numeral n*
⟨*proof*⟩

**lemma** *cos-npi-complex'* [*simp*]: *cos* (*of-nat n* ∗ *of-real pi*) = (−*1*) ^ *n* **for** *n*
⟨*proof*⟩

**lemma** *cos-two-pi* [*simp*]: *cos* (*2* ∗ *pi*) = *1*
⟨*proof*⟩

**lemma** *sin-two-pi* [*simp*]: *sin* (*2* ∗ *pi*) = *0*
⟨*proof*⟩

**context**
**fixes** *w* :: *'a*::{*real-normed-field*,*banach*}

**begin**

**lemma** *sin-times-sin*: *sin w* ∗ *sin z* = (*cos* (*w* − *z*) − *cos* (*w* + *z*)) / *2*
⟨*proof*⟩

**lemma** *sin-times-cos*: *sin w* ∗ *cos z* = (*sin* (*w* + *z*) + *sin* (*w* − *z*)) / *2*

⟨proof⟩

**lemma** *cos-times-sin*: $cos\ w * sin\ z = (sin\ (w + z) - sin\ (w - z)) / 2$
⟨proof⟩

**lemma** *cos-times-cos*: $cos\ w * cos\ z = (cos\ (w - z) + cos\ (w + z)) / 2$
⟨proof⟩

**lemma** *cos-double-cos*: $cos\ (2 * w) = 2 * cos\ w\ \hat{}\ 2 - 1$
⟨proof⟩

**lemma** *cos-double-sin*: $cos\ (2 * w) = 1 - 2 * sin\ w\ \hat{}\ 2$
⟨proof⟩

**end**

**lemma** *sin-plus-sin*: $sin\ w + sin\ z = 2 * sin\ ((w + z) / 2) * cos\ ((w - z) / 2)$
**for** $w$ :: $'a$::{*real-normed-field*,*banach*}
⟨proof⟩

**lemma** *sin-diff-sin*: $sin\ w - sin\ z = 2 * sin\ ((w - z) / 2) * cos\ ((w + z) / 2)$
**for** $w$ :: $'a$::{*real-normed-field*,*banach*}
⟨proof⟩

**lemma** *cos-plus-cos*: $cos\ w + cos\ z = 2 * cos\ ((w + z) / 2) * cos\ ((w - z) / 2)$
**for** $w$ :: $'a$::{*real-normed-field*,*banach*,*field*}
⟨proof⟩

**lemma** *cos-diff-cos*: $cos\ w - cos\ z = 2 * sin\ ((w + z) / 2) * sin\ ((z - w) / 2)$
**for** $w$ :: $'a$::{*real-normed-field*,*banach*,*field*}
⟨proof⟩

**lemma** *sin-pi-minus* [*simp*]: $sin\ (pi - x) = sin\ x$
⟨proof⟩

**lemma** *cos-pi-minus* [*simp*]: $cos\ (pi - x) = - (cos\ x)$
⟨proof⟩

**lemma** *sin-minus-pi* [*simp*]: $sin\ (x - pi) = - (sin\ x)$
⟨proof⟩

**lemma** *cos-minus-pi* [*simp*]: $cos\ (x - pi) = - (cos\ x)$
⟨proof⟩

**lemma** *sin-2pi-minus* [*simp*]: $sin\ (2 * pi - x) = - (sin\ x)$
⟨proof⟩

**lemma** *cos-2pi-minus* [*simp*]: $cos\ (2 * pi - x) = cos\ x$
⟨proof⟩

**lemma** *sin-gt-zero2*: $0 < x \Longrightarrow x < pi/2 \Longrightarrow 0 < sin\ x$
⟨*proof*⟩

**lemma** *sin-less-zero*:
**assumes** $- pi/2 < x$ **and** $x < 0$
**shows** $sin\ x < 0$
⟨*proof*⟩

**lemma** *pi-less-4*: $pi < 4$
⟨*proof*⟩

**lemma** *cos-gt-zero*: $0 < x \Longrightarrow x < pi/2 \Longrightarrow 0 < cos\ x$
⟨*proof*⟩

**lemma** *cos-gt-zero-pi*: $-(pi/2) < x \Longrightarrow x < pi/2 \Longrightarrow 0 < cos\ x$
⟨*proof*⟩

**lemma** *cos-ge-zero*: $-(pi/2) \leq x \Longrightarrow x \leq pi/2 \Longrightarrow 0 \leq cos\ x$
⟨*proof*⟩

**lemma** *sin-gt-zero*: $0 < x \Longrightarrow x < pi \Longrightarrow 0 < sin\ x$
⟨*proof*⟩

**lemma** *sin-lt-zero*: $pi < x \Longrightarrow x < 2 * pi \Longrightarrow sin\ x < 0$
⟨*proof*⟩

**lemma** *pi-ge-two*: $2 \leq pi$
⟨*proof*⟩

**lemma** *sin-ge-zero*: $0 \leq x \Longrightarrow x \leq pi \Longrightarrow 0 \leq sin\ x$
⟨*proof*⟩

**lemma** *sin-le-zero*: $pi \leq x \Longrightarrow x < 2 * pi \Longrightarrow sin\ x \leq 0$
⟨*proof*⟩

**lemma** *sin-pi-divide-n-ge-0* [*simp*]:
**assumes** $n \neq 0$
**shows** $0 \leq sin\ (pi/real\ n)$
⟨*proof*⟩

**lemma** *sin-pi-divide-n-gt-0*:
**assumes** $2 \leq n$
**shows** $0 < sin\ (pi/real\ n)$
⟨*proof*⟩

Proof resembles that of *cos-is-zero* but with *pi* for the upper bound

**lemma** *cos-total*:
**assumes** $y$: $-1 \leq y\ y \leq 1$

**shows** $\exists!x.\ 0 \le x \wedge x \le pi \wedge cos\ x = y$
$\langle proof \rangle$

**lemma** *sin-total*:
**assumes** $y$: $-1 \le y\ y \le 1$
**shows** $\exists!x.\ -(pi/2) \le x \wedge x \le pi/2 \wedge sin\ x = y$
$\langle proof \rangle$

**lemma** *cos-zero-lemma*:
**assumes** $0 \le x\ cos\ x = 0$
**shows** $\exists\, n.\ odd\ n \wedge x = of\text{-}nat\ n * (pi/2)$
$\langle proof \rangle$

**lemma** *sin-zero-lemma*:
**assumes** $0 \le x\ sin\ x = 0$
**shows** $\exists\, n{::}nat.\ even\ n \wedge x = real\ n * (pi/2)$
$\langle proof \rangle$

**lemma** *cos-zero-iff*:
$cos\ x = 0 \longleftrightarrow ((\exists\, n.\ odd\ n \wedge x = real\ n * (pi/2)) \vee (\exists\, n.\ odd\ n \wedge x = -(real\ n * (pi/2))))$
(**is** $?lhs = ?rhs$)
$\langle proof \rangle$

**lemma** *sin-zero-iff*:
$sin\ x = 0 \longleftrightarrow ((\exists\, n.\ even\ n \wedge x = real\ n * (pi/2)) \vee (\exists\, n.\ even\ n \wedge x = -(real\ n * (pi/2))))$
(**is** $?lhs = ?rhs$)
$\langle proof \rangle$

**lemma** *sin-zero-pi-iff*:
**fixes** $x{::}real$
**assumes** $|x| < pi$
**shows** $sin\ x = 0 \longleftrightarrow x = 0$
$\langle proof \rangle$

**lemma** *cos-zero-iff-int*: $cos\ x = 0 \longleftrightarrow (\exists\, i.\ odd\ i \wedge x = of\text{-}int\ i * (pi/2))$
$\langle proof \rangle$

**lemma** *sin-zero-iff-int*: $sin\ x = 0 \longleftrightarrow (\exists\, i.\ even\ i \wedge x = of\text{-}int\ i * (pi/2))$ (**is** $?lhs = ?rhs$)
$\langle proof \rangle$

**lemma** *sin-zero-iff-int2*: $sin\ x = 0 \longleftrightarrow (\exists\, i{::}int.\ x = of\text{-}int\ i * pi)$
$\langle proof \rangle$

**lemma** *cos-zero-iff-int2*:
**fixes** $x{::}real$
**shows** $cos\ x = 0 \longleftrightarrow (\exists\, n{::}int.\ x = n * pi + pi/2)$

⟨*proof*⟩

**lemma** *sin-npi-int* [*simp*]: $sin\ (pi * of\text{-}int\ n) = 0$
⟨*proof*⟩

**lemma** *cos-monotone-0-pi*:
**assumes** $0 \leq y$ **and** $y < x$ **and** $x \leq pi$
**shows** $cos\ x < cos\ y$
⟨*proof*⟩

**lemma** *cos-monotone-0-pi-le*:
**assumes** $0 \leq y$ **and** $y \leq x$ **and** $x \leq pi$
**shows** $cos\ x \leq cos\ y$
⟨*proof*⟩

**lemma** *cos-monotone-minus-pi-0*:
**assumes** $-\ pi \leq y$ **and** $y < x$ **and** $x \leq 0$
**shows** $cos\ y < cos\ x$
⟨*proof*⟩

**lemma** *cos-monotone-minus-pi-0'*:
**assumes** $-\ pi \leq y$ **and** $y \leq x$ **and** $x \leq 0$
**shows** $cos\ y \leq cos\ x$
⟨*proof*⟩

**lemma** *sin-monotone-2pi*:
**assumes** $-\ (pi/2) \leq y$ **and** $y < x$ **and** $x \leq pi/2$
**shows** $sin\ y < sin\ x$
⟨*proof*⟩

**lemma** *sin-monotone-2pi-le*:
**assumes** $-\ (pi/2) \leq y$ **and** $y \leq x$ **and** $x \leq pi/2$
**shows** $sin\ y \leq sin\ x$
⟨*proof*⟩

**lemma** *sin-x-le-x*:
**fixes** $x$ :: *real*
**assumes** $x \geq 0$
**shows** $sin\ x \leq x$
⟨*proof*⟩

**lemma** *sin-x-ge-neg-x*:
**fixes** $x$ :: *real*
**assumes** $x$: $x \geq 0$
**shows** $sin\ x \geq -\ x$
⟨*proof*⟩

**lemma** *abs-sin-x-le-abs-x*: $|sin\ x| \leq |x|$
**for** $x$ :: *real*

⟨*proof*⟩

## 112.13 More Corollaries about Sine and Cosine

**lemma** *sin-cos-npi* [*simp*]: $sin\ (real\ (Suc\ (2 * n)) * pi/2) = (-1)\ \hat{}\ n$
⟨*proof*⟩

**lemma** *cos-2npi* [*simp*]: $cos\ (2 * real\ n * pi) = 1$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *cos-3over2-pi* [*simp*]: $cos\ (3/2*pi) = 0$
⟨*proof*⟩

**lemma** *sin-2npi* [*simp*]: $sin\ (2 * real\ n * pi) = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *sin-3over2-pi* [*simp*]: $sin\ (3/2*pi) = -\ 1$
⟨*proof*⟩

**lemma** *cos-pi-eq-zero* [*simp*]: $cos\ (pi * real\ (Suc\ (2 * m))\ /\ 2) = 0$
⟨*proof*⟩

**lemma** *DERIV-cos-add* [*simp*]: $DERIV\ (\lambda x.\ cos\ (x + k))\ xa :> -\ sin\ (xa + k)$
⟨*proof*⟩

**lemma** *sin-zero-norm-cos-one*:
**fixes** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
**assumes** $sin\ x = 0$
**shows** $norm\ (cos\ x) = 1$
⟨*proof*⟩

**lemma** *sin-zero-abs-cos-one*: $sin\ x = 0 \Longrightarrow |cos\ x| = (1::real)$
⟨*proof*⟩

**lemma** *cos-one-sin-zero*:
**fixes** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
**assumes** $cos\ x = 1$
**shows** $sin\ x = 0$
⟨*proof*⟩

**lemma** *sin-times-pi-eq-0*: $sin\ (x * pi) = 0 \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *cos-one-2pi*: $cos\ x = 1 \longleftrightarrow (\exists\, n::nat.\ x = n * 2 * pi) \lor (\exists\, n::nat.\ x = -(n * 2 * pi))$
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

**lemma** *cos-one-2pi-int*: $cos\ x = 1 \longleftrightarrow (\exists\, n{::}int.\ x = n * 2 * pi)$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *cos-npi-int* [*simp*]:
**fixes** $n{::}int$ **shows** $cos\ (pi * of\text{-}int\ n) = (if\ even\ n\ then\ 1\ else\ -1)$
⟨*proof*⟩

**lemma** *sin-cos-sqrt*: $0 \le sin\ x \Longrightarrow sin\ x = sqrt\ (1 - (cos(x)\ \hat{}\ 2))$
⟨*proof*⟩

**lemma** *sin-eq-0-pi*: $-\ pi < x \Longrightarrow x < pi \Longrightarrow sin\ x = 0 \Longrightarrow x = 0$
⟨*proof*⟩

**lemma** *cos-treble-cos*: $cos\ (3 * x) = 4 * cos\ x\ \hat{}\ 3 - 3 * cos\ x$
**for** $x :: 'a{::}\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *cos-45*: $cos\ (pi/4) = sqrt\ 2\ /\ 2$
⟨*proof*⟩

**lemma** *cos-30*: $cos\ (pi/6) = sqrt\ 3/2$
⟨*proof*⟩

**lemma** *sin-45*: $sin\ (pi/4) = sqrt\ 2\ /\ 2$
⟨*proof*⟩

**lemma** *sin-60*: $sin\ (pi/3) = sqrt\ 3/2$
⟨*proof*⟩

**lemma** *cos-60*: $cos\ (pi/3) = 1/2$
⟨*proof*⟩

**lemma** *sin-30*: $sin\ (pi/6) = 1/2$
⟨*proof*⟩

**lemma** *cos-120*: $cos\ (2 * pi/3) = -1/2$
**and** *sin-120*: $sin\ (2 * pi/3) = sqrt\ 3\ /\ 2$
⟨*proof*⟩

**lemma** *cos-120'*: $cos\ (pi * 2\ /\ 3) = -1/2$
⟨*proof*⟩

**lemma** *sin-120'*: $sin\ (pi * 2\ /\ 3) = sqrt\ 3\ /\ 2$
⟨*proof*⟩

**lemma** *cos-integer-2pi*: $n \in \mathbb{Z} \Longrightarrow cos(2 * pi * n) = 1$
⟨*proof*⟩

**lemma** *sin-integer-2pi*: $n \in \mathbb{Z} \Longrightarrow sin(2 * pi * n) = 0$
⟨*proof*⟩

**lemma** *cos-int-2pin* [*simp*]: $cos\ ((2 * pi) * of\text{-}int\ n) = 1$
⟨*proof*⟩

**lemma** *sin-int-2pin* [*simp*]: $sin\ ((2 * pi) * of\text{-}int\ n) = 0$
⟨*proof*⟩

**lemma** *sin-cos-eq-iff*: $sin\ y = sin\ x \wedge cos\ y = cos\ x \longleftrightarrow (\exists\, n::int.\ y = x + 2 * pi * n)$ (**is** $?L=?R$)
⟨*proof*⟩

**lemma** *sincos-principal-value*: $\exists\, y.\ (-\ pi < y \wedge y \leq pi) \wedge (sin\ y = sin\ x \wedge cos\ y = cos\ x)$
⟨*proof*⟩

## 112.14 Tangent

**definition** $tan :: 'a \Rightarrow 'a::\{real\text{-}normed\text{-}field,banach\}$
**where** $tan = (\lambda x.\ sin\ x\ /\ cos\ x)$

**lemma** *tan-of-real*: $of\text{-}real\ (tan\ x) = (tan\ (of\text{-}real\ x) :: 'a::\{real\text{-}normed\text{-}field,banach\})$
⟨*proof*⟩

**lemma** *tan-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow tan\ z \in \mathbb{R}$
**for** $z :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *tan-zero* [*simp*]: $tan\ 0 = 0$
⟨*proof*⟩

**lemma** *tan-pi* [*simp*]: $tan\ pi = 0$
⟨*proof*⟩

**lemma** *tan-npi* [*simp*]: $tan\ (real\ n * pi) = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *tan-pi-half* [*simp*]: $tan\ (pi\ /\ 2) = 0$
⟨*proof*⟩

**lemma** *tan-minus* [*simp*]: $tan\ (-\ x) = -\ tan\ x$
⟨*proof*⟩

**lemma** *tan-periodic* [*simp*]: $tan\ (x + 2 * pi) = tan\ x$
⟨*proof*⟩

**lemma** *lemma-tan-add1*: $cos\ x \neq 0 \Longrightarrow cos\ y \neq 0 \Longrightarrow 1 - tan\ x * tan\ y = cos$

$(x + y)/(cos\ x * cos\ y)$
⟨proof⟩

**lemma** *add-tan-eq*: $cos\ x \neq 0 \Longrightarrow cos\ y \neq 0 \Longrightarrow tan\ x + tan\ y = sin(x + y)/(cos\ x * cos\ y)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨proof⟩

**lemma** *tan-eq-0-cos-sin*: $tan\ x = 0 \longleftrightarrow cos\ x = 0 \lor sin\ x = 0$
⟨proof⟩

Note: half of these zeros would normally be regarded as undefined cases.

**lemma** *tan-eq-0-Ex*:
**assumes** $tan\ x = 0$
**obtains** $k::int$ **where** $x = (k/2) * pi$
⟨proof⟩

**lemma** *tan-add*:
$cos\ x \neq 0 \Longrightarrow cos\ y \neq 0 \Longrightarrow cos\ (x + y) \neq 0 \Longrightarrow tan\ (x + y) = (tan\ x + tan\ y)/(1 - tan\ x * tan\ y)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨proof⟩

**lemma** *tan-double*: $cos\ x \neq 0 \Longrightarrow cos\ (2 * x) \neq 0 \Longrightarrow tan\ (2 * x) = (2 * tan\ x) / (1 - (tan\ x)^2)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨proof⟩

**lemma** *tan-gt-zero*: $0 < x \Longrightarrow x < pi/2 \Longrightarrow 0 < tan\ x$
⟨proof⟩

**lemma** *tan-less-zero*:
**assumes** $- pi/2 < x$ **and** $x < 0$
**shows** $tan\ x < 0$
⟨proof⟩

**lemma** *tan-half*: $tan\ x = sin\ (2 * x) / (cos\ (2 * x) + 1)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach,field\}$
⟨proof⟩

**lemma** *tan-30*: $tan\ (pi/6) = 1 / sqrt\ 3$
⟨proof⟩

**lemma** *tan-45*: $tan\ (pi/4) = 1$
⟨proof⟩

**lemma** *tan-60*: $tan\ (pi/3) = sqrt\ 3$
⟨proof⟩

**lemma** *DERIV-tan* [*simp*]: $cos\ x \neq 0 \Longrightarrow DERIV\ tan\ x :> inverse\ ((cos\ x)^2)$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**declare** *DERIV-tan*[*THEN DERIV-chain2*, *derivative-intros*]
**and** *DERIV-tan*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]

**lemmas** *has-derivative-tan*[*derivative-intros*] = *DERIV-tan*[*THEN DERIV-compose-FDERIV*]

**lemma** *isCont-tan*: $cos\ x \neq 0 \Longrightarrow isCont\ tan\ x$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *isCont-tan'* [*simp*,*continuous-intros*]:
**fixes** $a$ :: $'a$::{*real-normed-field*,*banach*} **and** $f$ :: $'a \Rightarrow 'a$
**shows** $isCont\ f\ a \Longrightarrow cos\ (f\ a) \neq 0 \Longrightarrow isCont\ (\lambda x.\ tan\ (f\ x))\ a$
⟨*proof*⟩

**lemma** *tendsto-tan* [*tendsto-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** $(f \longrightarrow a)\ F \Longrightarrow cos\ a \neq 0 \Longrightarrow ((\lambda x.\ tan\ (f\ x)) \longrightarrow tan\ a)\ F$
⟨*proof*⟩

**lemma** *continuous-tan*:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** $continuous\ F\ f \Longrightarrow cos\ (f\ (Lim\ F\ (\lambda x.\ x))) \neq 0 \Longrightarrow continuous\ F\ (\lambda x.\ tan\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-tan* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** $continuous\text{-}on\ s\ f \Longrightarrow (\forall x \in s.\ cos\ (f\ x) \neq 0) \Longrightarrow continuous\text{-}on\ s\ (\lambda x.\ tan\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-within-tan* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** $continuous\ (at\ x\ within\ s)\ f \Longrightarrow$
$cos\ (f\ x) \neq 0 \Longrightarrow continuous\ (at\ x\ within\ s)\ (\lambda x.\ tan\ (f\ x))$
⟨*proof*⟩

**lemma** *LIM-cos-div-sin*: $(\lambda x.\ cos(x)/sin(x))\ -pi/2\rightarrow\ 0$
⟨*proof*⟩

**lemma** *lemma-tan-total*:
**assumes** $0 < y$ **shows** $\exists x.\ 0 < x \wedge x < pi/2 \wedge y < tan\ x$
⟨*proof*⟩

**lemma** *tan-total-pos*:
**assumes** $0 \leq y$ **shows** $\exists x.\ 0 \leq x \wedge x < pi/2 \wedge tan\ x = y$
⟨*proof*⟩

**lemma** *lemma-tan-total1*: $\exists x.\ -(pi/2) < x \wedge x < (pi/2) \wedge tan\ x = y$
⟨*proof*⟩

**proposition** *tan-total*: $\exists!\ x.\ -(pi/2) < x \wedge x < (pi/2) \wedge tan\ x = y$
⟨*proof*⟩

**lemma** *tan-monotone*:
**assumes** $-\ (pi/2) < y$ **and** $y < x$ **and** $x < pi/2$
**shows** $tan\ y < tan\ x$
⟨*proof*⟩

**lemma** *tan-monotone'*:
**assumes** $-\ (pi/2) < y$
**and** $y < pi/2$
**and** $-\ (pi/2) < x$
**and** $x < pi/2$
**shows** $y < x \longleftrightarrow tan\ y < tan\ x$
⟨*proof*⟩

**lemma** *tan-inverse*: $1\ /\ (tan\ y) = tan\ (pi/2\ -\ y)$
⟨*proof*⟩

**lemma** *tan-periodic-pi*[*simp*]: $tan\ (x\ +\ pi) = tan\ x$
⟨*proof*⟩

**lemma** *tan-periodic-nat*[*simp*]: $tan\ (x\ +\ real\ n\ *\ pi) = tan\ x$
⟨*proof*⟩

**lemma** *tan-periodic-int*[*simp*]: $tan\ (x\ +\ of\text{-}int\ i\ *\ pi) = tan\ x$
⟨*proof*⟩

**lemma** *tan-periodic-n*[*simp*]: $tan\ (x\ +\ numeral\ n\ *\ pi) = tan\ x$
⟨*proof*⟩

**lemma** *tan-minus-45* [*simp*]: $tan\ (-(pi/4)) = -1$
⟨*proof*⟩

**lemma** *tan-diff*:
$cos\ x \neq 0 \Longrightarrow cos\ y \neq 0 \Longrightarrow cos\ (x\ -\ y) \neq 0 \Longrightarrow tan\ (x\ -\ y) = (tan\ x\ -\ tan\ y)/(1\ +\ tan\ x\ *\ tan\ y)$
**for** $x$ :: $'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *tan-pos-pi2-le*: $0 \leq x \Longrightarrow x < pi/2 \Longrightarrow 0 \leq tan\ x$
⟨*proof*⟩

**lemma** *cos-tan*: $|x| < pi/2 \Longrightarrow cos\ x = 1\ /\ sqrt\ (1 + tan\ x\ \hat{}\ 2)$
⟨*proof*⟩

**lemma** *cos-tan-half*: $cos\ x \neq 0 \Longrightarrow\ cos\ (2*x) = (1 - (tan\ x)\hat{}2)\ /\ (1 + (tan\ x)\hat{}2)$
⟨*proof*⟩

**lemma** *sin-tan*: $|x| < pi/2 \Longrightarrow sin\ x = tan\ x\ /\ sqrt\ (1 + tan\ x\ \hat{}\ 2)$
⟨*proof*⟩

**lemma** *sin-tan-half*: $sin\ (2*x) = 2 * tan\ x\ /\ (1 + (tan\ x)\hat{}2)$
⟨*proof*⟩

**lemma** *tan-mono-le*: $-(pi/2) < x \Longrightarrow x \leq y \Longrightarrow y < pi/2 \Longrightarrow tan\ x \leq tan\ y$
⟨*proof*⟩

**lemma** *tan-mono-lt-eq*:
$-(pi/2) < x \Longrightarrow x < pi/2 \Longrightarrow -(pi/2) < y \Longrightarrow y < pi/2 \Longrightarrow tan\ x < tan\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *tan-mono-le-eq*:
$-(pi/2) < x \Longrightarrow x < pi/2 \Longrightarrow -(pi/2) < y \Longrightarrow y < pi/2 \Longrightarrow tan\ x \leq tan\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *tan-bound-pi2*: $|x| < pi/4 \Longrightarrow |tan\ x| < 1$
⟨*proof*⟩

**lemma** *tan-cot*: $tan(pi/2 - x) = inverse(tan\ x)$
⟨*proof*⟩

## 112.15 Cotangent

**definition** $cot :: 'a \Rightarrow 'a::\{real\text{-}normed\text{-}field,banach\}$
**where** $cot = (\lambda x.\ cos\ x\ /\ sin\ x)$

**lemma** *cot-of-real*: $of\text{-}real\ (cot\ x) = (cot\ (of\text{-}real\ x) :: 'a::\{real\text{-}normed\text{-}field,banach\})$
⟨*proof*⟩

**lemma** *cot-in-Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow cot\ z \in \mathbb{R}$
**for** $z :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *cot-zero* [*simp*]: $cot\ 0 = 0$
⟨*proof*⟩

**lemma** *cot-pi* [*simp*]: $cot\ pi = 0$

⟨*proof*⟩

**lemma** *cot-npi* [*simp*]: $cot\ (real\ n * pi) = 0$
**for** $n :: nat$
⟨*proof*⟩

**lemma** *cot-minus* [*simp*]: $cot\ (-\ x) = -\ cot\ x$
⟨*proof*⟩

**lemma** *cot-periodic* [*simp*]: $cot\ (x + 2 * pi) = cot\ x$
⟨*proof*⟩

**lemma** *cot-altdef*: $cot\ x = inverse\ (tan\ x)$
⟨*proof*⟩

**lemma** *tan-altdef*: $tan\ x = inverse\ (cot\ x)$
⟨*proof*⟩

**lemma** *tan-cot'*: $tan\ (pi/2 - x) = cot\ x$
⟨*proof*⟩

**lemma** *cot-gt-zero*: $0 < x \Longrightarrow x < pi/2 \Longrightarrow 0 < cot\ x$
⟨*proof*⟩

**lemma** *cot-less-zero*:
**assumes** *lb*: $-\ pi/2 < x$ **and** $x < 0$
**shows** $cot\ x < 0$
⟨*proof*⟩

**lemma** *DERIV-cot* [*simp*]: $sin\ x \neq 0 \Longrightarrow DERIV\ cot\ x :> -inverse\ ((sin\ x)^2)$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *isCont-cot*: $sin\ x \neq 0 \Longrightarrow isCont\ cot\ x$
**for** $x :: 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *isCont-cot'* [*simp*,*continuous-intros*]:
$isCont\ f\ a \Longrightarrow sin\ (f\ a) \neq 0 \Longrightarrow isCont\ (\lambda x.\ cot\ (f\ x))\ a$
**for** $a :: 'a::\{real\text{-}normed\text{-}field,banach\}$ **and** $f :: 'a \Rightarrow 'a$
⟨*proof*⟩

**lemma** *tendsto-cot* [*tendsto-intros*]: $(f \longrightarrow a)\ F \Longrightarrow sin\ a \neq 0 \Longrightarrow ((\lambda x.\ cot\ (f\ x)) \longrightarrow cot\ a)\ F$
**for** $f :: 'a \Rightarrow 'a::\{real\text{-}normed\text{-}field,banach\}$
⟨*proof*⟩

**lemma** *continuous-cot*:
$continuous\ F\ f \Longrightarrow sin\ (f\ (Lim\ F\ (\lambda x.\ x))) \neq 0 \Longrightarrow continuous\ F\ (\lambda x.\ cot\ (f\ x))$

**for** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
⟨*proof*⟩

**lemma** *continuous-on-cot* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** *continuous-on s f* $\Longrightarrow$ ($\forall x \in s$. *sin* ($f\,x$) $\neq$ *0*) $\Longrightarrow$ *continuous-on s* ($\lambda x$. *cot* ($f\,x$))
⟨*proof*⟩

**lemma** *continuous-within-cot* [*continuous-intros*]:
**fixes** $f$ :: $'a \Rightarrow 'a$::{*real-normed-field*,*banach*}
**shows** *continuous* (*at x within s*) *f* $\Longrightarrow$ *sin* ($f\,x$) $\neq$ *0* $\Longrightarrow$ *continuous* (*at x within s*) ($\lambda x$. *cot* ($f\,x$))
⟨*proof*⟩

## 112.16 Inverse Trigonometric Functions

**definition** *arcsin* :: *real* $\Rightarrow$ *real*
**where** *arcsin y* = (*THE x*. $-(pi/2) \leq x \wedge x \leq pi/2 \wedge sin\ x = y$)

**definition** *arccos* :: *real* $\Rightarrow$ *real*
**where** *arccos y* = (*THE x*. $0 \leq x \wedge x \leq pi \wedge cos\ x = y$)

**definition** *arctan* :: *real* $\Rightarrow$ *real*
**where** *arctan y* = (*THE x*. $-(pi/2) < x \wedge x < pi/2 \wedge tan\ x = y$)

**lemma** *arcsin*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow -\ (pi/2) \leq arcsin\ y \wedge arcsin\ y \leq pi/2 \wedge sin\ (arcsin\ y) = y$
⟨*proof*⟩

**lemma** *arcsin-pi*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow -\ (pi/2) \leq arcsin\ y \wedge arcsin\ y \leq pi \wedge sin\ (arcsin\ y) = y$
⟨*proof*⟩

**lemma** *sin-arcsin* [*simp*]: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow sin\ (arcsin\ y) = y$
⟨*proof*⟩

**lemma** *arcsin-bounded*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow -\ (pi/2) \leq arcsin\ y \wedge arcsin\ y \leq pi/2$
⟨*proof*⟩

**lemma** *arcsin-lbound*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow -\ (pi/2) \leq arcsin\ y$
⟨*proof*⟩

**lemma** *arcsin-ubound*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow arcsin\ y \leq pi/2$
⟨*proof*⟩

**lemma** *arcsin-lt-bounded*:
**assumes** $-\ 1 < y\ y < 1$

**shows** $-(pi/2) < arcsin\ y \wedge arcsin\ y < pi/2$
⟨*proof*⟩

**lemma** *arcsin-sin*: $-(pi/2) \leq x \Longrightarrow x \leq pi/2 \Longrightarrow arcsin\ (sin\ x) = x$
⟨*proof*⟩

**lemma** *arcsin-unique*:
**assumes** $-pi/2 \leq x$ **and** $x \leq pi/2$ **and** $sin\ x = y$ **shows** $arcsin\ y = x$
⟨*proof*⟩

**lemma** *arcsin-0* [*simp*]: $arcsin\ 0 = 0$
⟨*proof*⟩

**lemma** *arcsin-1* [*simp*]: $arcsin\ 1 = pi/2$
⟨*proof*⟩

**lemma** *arcsin-minus-1* [*simp*]: $arcsin\ (-\ 1) = -\ (pi/2)$
⟨*proof*⟩

**lemma** *arcsin-minus*: $-\ 1 \leq x \Longrightarrow x \leq 1 \Longrightarrow arcsin\ (-\ x) = -\ arcsin\ x$
⟨*proof*⟩

**lemma** *arcsin-one-half* [*simp*]: $arcsin\ (1/2) = pi\ /\ 6$
**and** *arcsin-minus-one-half* [*simp*]: $arcsin\ (-(1/2)) = -pi\ /\ 6$
⟨*proof*⟩

**lemma** *arcsin-one-over-sqrt-2*: $arcsin\ (1\ /\ sqrt\ 2) = pi\ /\ 4$
⟨*proof*⟩

**lemma** *arcsin-eq-iff*: $|x| \leq 1 \Longrightarrow |y| \leq 1 \Longrightarrow arcsin\ x = arcsin\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *cos-arcsin-nonzero*: $-\ 1 < x \Longrightarrow x < 1 \Longrightarrow cos\ (arcsin\ x) \neq 0$
⟨*proof*⟩

**lemma** *arccos*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow 0 \leq arccos\ y \wedge arccos\ y \leq pi \wedge cos\ (arccos\ y) = y$
⟨*proof*⟩

**lemma** *cos-arccos* [*simp*]: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow cos\ (arccos\ y) = y$
⟨*proof*⟩

**lemma** *arccos-bounded*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow 0 \leq arccos\ y \wedge arccos\ y \leq pi$
⟨*proof*⟩

**lemma** *arccos-lbound*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow 0 \leq arccos\ y$
⟨*proof*⟩

**lemma** *arccos-ubound*: $-\ 1 \leq y \Longrightarrow y \leq 1 \Longrightarrow arccos\ y \leq pi$

⟨*proof*⟩

**lemma** *arccos-lt-bounded*:
**assumes** $-\ 1 < y\ y < 1$
**shows** $0 < arccos\ y \wedge arccos\ y < pi$
⟨*proof*⟩

**lemma** *arccos-cos*: $0 \leq x \Longrightarrow x \leq pi \Longrightarrow arccos\ (cos\ x) = x$
⟨*proof*⟩

**lemma** *arccos-cos2*: $x \leq 0 \Longrightarrow -\ pi \leq x \Longrightarrow arccos\ (cos\ x) = -x$
⟨*proof*⟩

**lemma** *arccos-unique*:
**assumes** $0 \leq x$ **and** $x \leq pi$ **and** $cos\ x = y$ **shows** $arccos\ y = x$
⟨*proof*⟩

**lemma** *cos-arcsin*:
**assumes** $-\ 1 \leq x\ x \leq 1$
**shows** $cos\ (arcsin\ x) = sqrt\ (1\ -\ x^2)$
⟨*proof*⟩

**lemma** *sin-arccos*:
**assumes** $-\ 1 \leq x\ x \leq 1$
**shows** $sin\ (arccos\ x) = sqrt\ (1\ -\ x^2)$
⟨*proof*⟩

**lemma** *arccos-0* [*simp*]: $arccos\ 0 = pi/2$
⟨*proof*⟩

**lemma** *arccos-1* [*simp*]: $arccos\ 1 = 0$
⟨*proof*⟩

**lemma** *arccos-minus-1* [*simp*]: $arccos\ (-\ 1) = pi$
⟨*proof*⟩

**lemma** *arccos-minus*: $-1 \leq x \Longrightarrow x \leq 1 \Longrightarrow arccos\ (-\ x) = pi\ -\ arccos\ x$
⟨*proof*⟩

**lemma** *arccos-one-half* [*simp*]: $arccos\ (1/2) = pi\ /\ 3$
**and** *arccos-minus-one-half* [*simp*]: $arccos\ (-(1/2)) = 2 * pi\ /\ 3$
⟨*proof*⟩

**lemma** *arccos-one-over-sqrt-2*: $arccos\ (1\ /\ sqrt\ 2) = pi\ /\ 4$
⟨*proof*⟩

**corollary** *arccos-minus-abs*:
**assumes** $|x| \leq 1$
**shows** $arccos\ (-\ x) = pi\ -\ arccos\ x$

⟨*proof*⟩

**lemma** *sin-arccos-nonzero*: $- 1 < x \Longrightarrow x < 1 \Longrightarrow sin\ (arccos\ x) \neq 0$
⟨*proof*⟩

**lemma** *arctan*: $- (pi/2) < arctan\ y \wedge arctan\ y < pi/2 \wedge tan\ (arctan\ y) = y$
⟨*proof*⟩

**lemma** *tan-arctan*: $tan\ (arctan\ y) = y$
⟨*proof*⟩

**lemma** *arctan-bounded*: $- (pi/2) < arctan\ y \wedge arctan\ y < pi/2$
⟨*proof*⟩

**lemma** *arctan-lbound*: $- (pi/2) < arctan\ y$
⟨*proof*⟩

**lemma** *arctan-ubound*: $arctan\ y < pi/2$
⟨*proof*⟩

**lemma** *arctan-unique*:
**assumes** $-(pi/2) < x$
**and** $x < pi/2$
**and** $tan\ x = y$
**shows** $arctan\ y = x$
⟨*proof*⟩

**lemma** *arctan-tan*: $-(pi/2) < x \Longrightarrow x < pi/2 \Longrightarrow arctan\ (tan\ x) = x$
⟨*proof*⟩

**lemma** *arctan-zero-zero* [*simp*]: $arctan\ 0 = 0$
⟨*proof*⟩

**lemma** *arctan-minus*: $arctan\ (- x) = - arctan\ x$
⟨*proof*⟩

**lemma** *cos-arctan-not-zero* [*simp*]: $cos\ (arctan\ x) \neq 0$
⟨*proof*⟩

**lemma** *tan-eq-arctan-Ex*:
**shows** $tan\ x = y \longleftrightarrow (\exists k::int.\ x = arctan\ y + k*pi \vee (x = pi/2 + k*pi \wedge y=0))$
⟨*proof*⟩

**lemma** *arctan-tan-eq-abs-pi*:
**assumes** $cos\ \vartheta \neq 0$
**obtains** $k$ **where** $arctan\ (tan\ \vartheta) = \vartheta - of\text{-}int\ k * pi$
⟨*proof*⟩

**lemma** *tan-eq*:
**assumes** $tan\ x = tan\ y$ $tan\ x \neq 0$
**obtains** $k::int$ **where** $x = y + k * pi$
⟨*proof*⟩

**lemma** *cos-arctan*: $cos\ (arctan\ x) = 1\ /\ sqrt\ (1 + x^2)$
⟨*proof*⟩

**lemma** *sin-arctan*: $sin\ (arctan\ x) = x\ /\ sqrt\ (1 + x^2)$
⟨*proof*⟩

**lemma** *tan-sec*: $cos\ x \neq 0 \Longrightarrow 1 + (tan\ x)^2 = (inverse\ (cos\ x))^2$
**for** $x$ :: $'a$::{*real-normed-field,banach,field*}
⟨*proof*⟩

**lemma** *arctan-less-iff*: $arctan\ x < arctan\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *arctan-le-iff*: $arctan\ x \leq arctan\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *arctan-eq-iff*: $arctan\ x = arctan\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *zero-less-arctan-iff* [*simp*]: $0 < arctan\ x \longleftrightarrow 0 < x$
⟨*proof*⟩

**lemma** *arctan-less-zero-iff* [*simp*]: $arctan\ x < 0 \longleftrightarrow x < 0$
⟨*proof*⟩

**lemma** *zero-le-arctan-iff* [*simp*]: $0 \leq arctan\ x \longleftrightarrow 0 \leq x$
⟨*proof*⟩

**lemma** *arctan-le-zero-iff* [*simp*]: $arctan\ x \leq 0 \longleftrightarrow x \leq 0$
⟨*proof*⟩

**lemma** *arctan-eq-zero-iff* [*simp*]: $arctan\ x = 0 \longleftrightarrow x = 0$
⟨*proof*⟩

**lemma** *continuous-on-arcsin'*: *continuous-on* $\{-1\ ..\ 1\}$ *arcsin*
⟨*proof*⟩

**lemma** *continuous-on-arcsin* [*continuous-intros*]:
*continuous-on* $s\ f \Longrightarrow (\forall x \in s.\ -1 \leq f\ x \wedge f\ x \leq 1) \Longrightarrow$ *continuous-on* $s$ $(\lambda x.$ *arcsin* $(f\ x))$
⟨*proof*⟩

**lemma** *isCont-arcsin*: $-1 < x \Longrightarrow x < 1 \Longrightarrow$ *isCont arcsin* $x$
⟨*proof*⟩

**lemma** *continuous-on-arccos'*: *continuous-on* $\{-1 .. 1\}$ *arccos*
⟨*proof*⟩

**lemma** *continuous-on-arccos* [*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ $(\forall x \in s.\ -1 \le f\ x \wedge f\ x \le 1)$ $\Longrightarrow$ *continuous-on s* $(\lambda x.$ *arccos* $(f\ x))$
⟨*proof*⟩

**lemma** *isCont-arccos*: $-1 < x \Longrightarrow x < 1 \Longrightarrow$ *isCont arccos x*
⟨*proof*⟩

**lemma** *isCont-arctan*: *isCont arctan x*
⟨*proof*⟩

**lemma** *tendsto-arctan* [*tendsto-intros*]: $(f \longrightarrow x)\ F \Longrightarrow ((\lambda x.\ arctan\ (f\ x)) \longrightarrow arctan\ x)\ F$
⟨*proof*⟩

**lemma** *continuous-arctan* [*continuous-intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ arctan\ (f\ x))$
⟨*proof*⟩

**lemma** *continuous-on-arctan* [*continuous-intros*]:
*continuous-on s f* $\Longrightarrow$ *continuous-on s* $(\lambda x.\ arctan\ (f\ x))$
⟨*proof*⟩

**lemma** *DERIV-arcsin*:
**assumes** $-1 < x\ x < 1$
**shows** *DERIV arcsin x* :> *inverse* (*sqrt* $(1 - x^2)$)
⟨*proof*⟩

**lemma** *DERIV-arccos*:
**assumes** $-1 < x\ x < 1$
**shows** *DERIV arccos x* :> *inverse* ($-$ *sqrt* $(1 - x^2)$)
⟨*proof*⟩

**lemma** *DERIV-arctan*: *DERIV arctan x* :> *inverse* $(1 + x^2)$
⟨*proof*⟩

**declare**
*DERIV-arcsin*[*THEN DERIV-chain2*, *derivative-intros*]
*DERIV-arcsin*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]
*DERIV-arccos*[*THEN DERIV-chain2*, *derivative-intros*]
*DERIV-arccos*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *derivative-intros*]
*DERIV-arctan*[*THEN DERIV-chain2*, *derivative-intros*]
*DERIV-arctan*[*THEN DERIV-chain2*, *unfolded has-field-derivative-def*, *deriva-*

*tive-intros*]

**lemmas** *has-derivative-arctan*[*derivative-intros*] = *DERIV-arctan*[*THEN DERIV-compose-FDERIV*]
**and** *has-derivative-arccos*[*derivative-intros*] = *DERIV-arccos*[*THEN DERIV-compose-FDERIV*]
**and** *has-derivative-arcsin*[*derivative-intros*] = *DERIV-arcsin*[*THEN DERIV-compose-FDERIV*]

**lemma** *filterlim-tan-at-right*: *filterlim tan at-bot* (*at-right* (− (*pi*/*2*)))
⟨*proof*⟩

**lemma** *filterlim-tan-at-left*: *filterlim tan at-top* (*at-left* (*pi*/*2*))
⟨*proof*⟩

**lemma** *tendsto-arctan-at-top*: (*arctan* ⟶ (*pi*/*2*)) *at-top*
⟨*proof*⟩

**lemma** *tendsto-arctan-at-bot*: (*arctan* ⟶ − (*pi*/*2*)) *at-bot*
⟨*proof*⟩

**lemma** *sin-multiple-reduce*:
*sin* (*x* ∗ *numeral n* :: *'a* :: {*real-normed-field*, *banach*}) =
*sin x* ∗ *cos* (*x* ∗ *of-nat* (*pred-numeral n*)) + *cos x* ∗ *sin* (*x* ∗ *of-nat* (*pred-numeral n*))
⟨*proof*⟩

**lemma** *cos-multiple-reduce*:
*cos* (*x* ∗ *numeral n* :: *'a* :: {*real-normed-field*, *banach*}) =
*cos* (*x* ∗ *of-nat* (*pred-numeral n*)) ∗ *cos x* − *sin* (*x* ∗ *of-nat* (*pred-numeral n*)) ∗ *sin x*
⟨*proof*⟩

**lemma** *arccos-eq-pi-iff*: $x \in \{-1..1\} \Longrightarrow arccos\ x = pi \longleftrightarrow x = -1$
⟨*proof*⟩

**lemma** *arccos-eq-0-iff*: $x \in \{-1..1\} \Longrightarrow arccos\ x = 0 \longleftrightarrow x = 1$
⟨*proof*⟩

## 112.17 Prove Totality of the Trigonometric Functions

**lemma** *cos-arccos-abs*: $|y| \le 1 \Longrightarrow cos\ (arccos\ y) = y$
⟨*proof*⟩

**lemma** *sin-arccos-abs*: $|y| \le 1 \Longrightarrow sin\ (arccos\ y) = sqrt\ (1 - y^2)$
⟨*proof*⟩

**lemma** *sin-mono-less-eq*:
$-(pi/2) \le x \Longrightarrow x \le pi/2 \Longrightarrow -(pi/2) \le y \Longrightarrow y \le pi/2 \Longrightarrow sin\ x < sin\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *sin-mono-le-eq*:
$-(pi/2) \leq x \Longrightarrow x \leq pi/2 \Longrightarrow -(pi/2) \leq y \Longrightarrow y \leq pi/2 \Longrightarrow sin\ x \leq sin\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *sin-inj-pi*:
$-(pi/2) \leq x \Longrightarrow x \leq pi/2 \Longrightarrow -(pi/2) \leq y \Longrightarrow y \leq pi/2 \Longrightarrow sin\ x = sin\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *arcsin-le-iff*:
**assumes** $x \geq -1$ $x \leq 1$ $y \geq -pi/2$ $y \leq pi/2$
**shows** $arcsin\ x \leq y \longleftrightarrow x \leq sin\ y$
⟨*proof*⟩

**lemma** *le-arcsin-iff*:
**assumes** $x \geq -1$ $x \leq 1$ $y \geq -pi/2$ $y \leq pi/2$
**shows** $arcsin\ x \geq y \longleftrightarrow x \geq sin\ y$
⟨*proof*⟩

**lemma** *cos-mono-less-eq*: $0 \leq x \Longrightarrow x \leq pi \Longrightarrow 0 \leq y \Longrightarrow y \leq pi \Longrightarrow cos\ x < cos\ y \longleftrightarrow y < x$
⟨*proof*⟩

**lemma** *cos-mono-le-eq*: $0 \leq x \Longrightarrow x \leq pi \Longrightarrow 0 \leq y \Longrightarrow y \leq pi \Longrightarrow cos\ x \leq cos\ y \longleftrightarrow y \leq x$
⟨*proof*⟩

**lemma** *cos-inj-pi*: $0 \leq x \Longrightarrow x \leq pi \Longrightarrow 0 \leq y \Longrightarrow y \leq pi \Longrightarrow cos\ x = cos\ y \Longrightarrow x = y$
⟨*proof*⟩

**lemma** *arccos-le-pi2*: $[\![0 \leq y;\ y \leq 1]\!] \Longrightarrow arccos\ y \leq pi/2$
⟨*proof*⟩

**lemma** *sincos-total-pi-half*:
**assumes** $0 \leq x$ $0 \leq y$ $x^2 + y^2 = 1$
**shows** $\exists t.\ 0 \leq t \wedge t \leq pi/2 \wedge x = cos\ t \wedge y = sin\ t$
⟨*proof*⟩

**lemma** *sincos-total-pi*:
**assumes** $0 \leq y$ $x^2 + y^2 = 1$
**shows** $\exists t.\ 0 \leq t \wedge t \leq pi \wedge x = cos\ t \wedge y = sin\ t$
⟨*proof*⟩

**lemma** *sincos-total-2pi-le*:
**assumes** $x^2 + y^2 = 1$
**shows** $\exists t.\ 0 \leq t \wedge t \leq 2 * pi \wedge x = cos\ t \wedge y = sin\ t$
⟨*proof*⟩

**lemma** *sincos-total-2pi*:
**assumes** $x^2 + y^2 = 1$
**obtains** $t$ **where** $0 \leq t$ $t < 2*pi$ $x = cos\ t$ $y = sin\ t$
⟨*proof*⟩

**lemma** *arcsin-less-mono*: $|x| \leq 1 \Longrightarrow |y| \leq 1 \Longrightarrow arcsin\ x < arcsin\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *arcsin-le-mono*: $|x| \leq 1 \Longrightarrow |y| \leq 1 \Longrightarrow arcsin\ x \leq arcsin\ y \longleftrightarrow x \leq y$
⟨*proof*⟩

**lemma** *arcsin-less-arcsin*: $-1 \leq x \Longrightarrow x < y \Longrightarrow y \leq 1 \Longrightarrow arcsin\ x < arcsin\ y$
⟨*proof*⟩

**lemma** *arcsin-le-arcsin*: $-1 \leq x \Longrightarrow x \leq y \Longrightarrow y \leq 1 \Longrightarrow arcsin\ x \leq arcsin\ y$
⟨*proof*⟩

**lemma** *arcsin-nonneg*: $x \in \{0..1\} \Longrightarrow arcsin\ x \geq 0$
⟨*proof*⟩

**lemma** *arccos-less-mono*: $|x| \leq 1 \Longrightarrow |y| \leq 1 \Longrightarrow arccos\ x < arccos\ y \longleftrightarrow y < x$
⟨*proof*⟩

**lemma** *arccos-le-mono*: $|x| \leq 1 \Longrightarrow |y| \leq 1 \Longrightarrow arccos\ x \leq arccos\ y \longleftrightarrow y \leq x$
⟨*proof*⟩

**lemma** *arccos-less-arccos*: $-1 \leq x \Longrightarrow x < y \Longrightarrow y \leq 1 \Longrightarrow arccos\ y < arccos\ x$
⟨*proof*⟩

**lemma** *arccos-le-arccos*: $-1 \leq x \Longrightarrow x \leq y \Longrightarrow y \leq 1 \Longrightarrow arccos\ y \leq arccos\ x$
⟨*proof*⟩

**lemma** *arccos-eq-iff*: $|x| \leq 1 \wedge |y| \leq 1 \Longrightarrow arccos\ x = arccos\ y \longleftrightarrow x = y$
⟨*proof*⟩

**lemma** *arccos-cos-eq-abs*:
**assumes** $|\vartheta| \leq pi$
**shows** $arccos\ (cos\ \vartheta) = |\vartheta|$
⟨*proof*⟩

**lemma** *arccos-cos-eq-abs-2pi*:
**obtains** $k$ **where** $arccos\ (cos\ \vartheta) = |\vartheta - of\text{-}int\ k * (2 * pi)|$
⟨*proof*⟩

**lemma** *arccos-arctan*:
**assumes** $-1 < x$ $x < 1$
**shows** $arccos\ x = pi/2 - arctan(x\ /\ sqrt(1 - x^2))$

⟨*proof*⟩

**lemma** *arcsin-plus-arccos*:
**assumes** $-1 \leq x$ $x \leq 1$
**shows** $arcsin\ x + arccos\ x = pi/2$
⟨*proof*⟩

**lemma** *arcsin-arccos-eq*: $-1 \leq x \Longrightarrow x \leq 1 \Longrightarrow arcsin\ x = pi/2 - arccos\ x$
⟨*proof*⟩

**lemma** *arccos-arcsin-eq*: $-1 \leq x \Longrightarrow x \leq 1 \Longrightarrow arccos\ x = pi/2 - arcsin\ x$
⟨*proof*⟩

**lemma** *arcsin-arctan*: $-1 < x \Longrightarrow x < 1 \Longrightarrow arcsin\ x = arctan(x\ /\ sqrt(1 - x^2))$
⟨*proof*⟩

**lemma** *arcsin-arccos-sqrt-pos*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow arcsin\ x = arccos(sqrt(1 - x^2))$
⟨*proof*⟩

**lemma** *arcsin-arccos-sqrt-neg*: $-1 \leq x \Longrightarrow x \leq 0 \Longrightarrow arcsin\ x = -arccos(sqrt(1 - x^2))$
⟨*proof*⟩

**lemma** *arccos-arcsin-sqrt-pos*: $0 \leq x \Longrightarrow x \leq 1 \Longrightarrow arccos\ x = arcsin(sqrt(1 - x^2))$
⟨*proof*⟩

**lemma** *arccos-arcsin-sqrt-neg*: $-1 \leq x \Longrightarrow x \leq 0 \Longrightarrow arccos\ x = pi - arcsin(sqrt(1 - x^2))$
⟨*proof*⟩

**lemma** *cos-limit-1*:
**assumes** $(\lambda j.\ cos\ (\vartheta\ j)) \longrightarrow 1$
**shows** $\exists k.\ (\lambda j.\ \vartheta\ j - of\text{-}int\ (k\ j) * (2 * pi)) \longrightarrow 0$
⟨*proof*⟩

**lemma** *cos-diff-limit-1*:
**assumes** $(\lambda j.\ cos\ (\vartheta\ j - \Theta)) \longrightarrow 1$
**obtains** $k$ **where** $(\lambda j.\ \vartheta\ j - of\text{-}int\ (k\ j) * (2 * pi)) \longrightarrow \Theta$
⟨*proof*⟩

## 112.18 Machin's formula

**lemma** *arctan-one*: $arctan\ 1 = pi/4$
⟨*proof*⟩

**lemma** *tan-total-pi4*:
**assumes** $|x| < 1$

**shows** $\exists\, z.\ -\ (pi/4) < z \wedge z < pi/4 \wedge tan\ z = x$
⟨*proof*⟩

**lemma** *arctan-add*:
**assumes** $|x| \leq 1$ $|y| < 1$
**shows** $arctan\ x + arctan\ y = arctan\ ((x + y)\ /\ (1 - x * y))$
⟨*proof*⟩

**lemma** *arctan-double*: $|x| < 1 \Longrightarrow 2 * arctan\ x = arctan\ ((2 * x)\ /\ (1 - x^2))$
⟨*proof*⟩

**theorem** *machin*: $pi/4 = 4 * arctan\ (1\ /\ 5) - arctan\ (1/239)$
⟨*proof*⟩

**lemma** *machin-Euler*: $5 * arctan\ (1\ /\ 7) + 2 * arctan\ (3\ /\ 79) = pi/4$
⟨*proof*⟩

## 112.19 Introducing the inverse tangent power series

**lemma** *monoseq-arctan-series*:
**fixes** $x :: real$
**assumes** $|x| \leq 1$
**shows** $monoseq\ (\lambda n.\ 1\ /\ real\ (n * 2 + 1) * x\hat{}(n * 2 + 1))$
(**is** $monoseq\ ?a$)
⟨*proof*⟩

**lemma** *zeroseq-arctan-series*:
**fixes** $x :: real$
**assumes** $|x| \leq 1$
**shows** $(\lambda n.\ 1\ /\ real\ (n * 2 + 1) * x\hat{}(n * 2 + 1)) \longlongrightarrow 0$
(**is** $?a \longlongrightarrow 0$)
⟨*proof*⟩

**lemma** *summable-arctan-series*:
**fixes** $n :: nat$
**assumes** $|x| \leq 1$
**shows** $summable\ (\lambda\ k.\ (-1)\hat{}k * (1\ /\ real\ (k{*}2{+}1) * x\ \hat{}\ (k{*}2{+}1)))$
(**is** $summable\ (?c\ x)$)
⟨*proof*⟩

**lemma** *DERIV-arctan-series*:
**assumes** $|x| < 1$
**shows** $DERIV\ (\lambda x'.\ \sum k.\ (-1)\hat{}k * (1\ /\ real\ (k * 2 + 1) * x'\ \hat{}\ (k * 2 + 1)))$
$x :>$
$(\sum k.\ (-1)\hat{}k * x\hat{}(k * 2))$
(**is** $DERIV\ ?arctan\ \text{-} :> ?Int$)
⟨*proof*⟩

**lemma** *arctan-series*:

**assumes** $|x| \leq 1$
**shows** *arctan* $x = (\sum k.\ (-1)\hat{~}k * (1\ /\ real\ (k * 2 + 1) * x\ \hat{~}\ (k * 2 + 1)))$
(**is** - = *suminf* $(\lambda\ n.\ ?c\ x\ n)$)
⟨*proof*⟩

**lemma** *arctan-half*: $arctan\ x = 2 * arctan\ (x\ /\ (1 + sqrt(1 + x^2)))$
**for** $x$ :: *real*
⟨*proof*⟩

**lemma** *arctan-monotone*: $x < y \Longrightarrow arctan\ x < arctan\ y$
⟨*proof*⟩

**lemma** *arctan-monotone'*: $x \leq y \Longrightarrow arctan\ x \leq arctan\ y$
⟨*proof*⟩

**lemma** *arctan-inverse*:
**assumes** $x \neq 0$
**shows** $arctan\ (1/x) = sgn\ x * pi/2 - arctan\ x$
⟨*proof*⟩

**theorem** *pi-series*: $pi/4 = (\sum k.\ (-1)\hat{~}k * 1\ /\ real\ (k * 2 + 1))$
(**is** - = *?SUM*)
⟨*proof*⟩

## 112.20 Existence of Polar Coordinates

**lemma** *cos-x-y-le-one*: $|x\ /\ sqrt\ (x^2 + y^2)| \leq 1$
⟨*proof*⟩

**lemma** *polar-Ex*: $\exists\, r$::*real*. $\exists\, a.\ x = r * cos\ a \wedge y = r * sin\ a$
⟨*proof*⟩

## 112.21 Basics about polynomial functions: products, extremal behaviour and root counts

**lemma** *polynomial-product-nat*:
**fixes** $x$ :: *nat*
**assumes** $m$: $\bigwedge i.\ i > m \Longrightarrow int\ (a\ i) = 0$
**and** $n$: $\bigwedge j.\ j > n \Longrightarrow int\ (b\ j) = 0$
**shows** $(\sum i{\leq}m.\ (a\ i) * x\ \hat{~}\ i) * (\sum j{\leq}n.\ (b\ j) * x\ \hat{~}\ j) =$
$(\sum r{\leq}m + n.\ (\sum k{\leq}r.\ (a\ k) * (b\ (r - k))) * x\ \hat{~}\ r)$
⟨*proof*⟩

**lemma** *polyfun-diff*:
**fixes** $x$ :: *'a::idom*
**assumes** $1 \leq n$
**shows** $(\sum i{\leq}n.\ a\ i * x\hat{~}i) - (\sum i{\leq}n.\ a\ i * y\hat{~}i) =$
$(x - y) * (\sum j{<}n.\ (\sum i{=}Suc\ j..n.\ a\ i * y\hat{~}(i - j - 1)) * x\hat{~}j)$
⟨*proof*⟩

**lemma** *polyfun-diff-alt*:
**fixes** $x :: 'a::idom$
**assumes** $1 \leq n$
**shows** $(\sum i{\leq}n.\ a\ i * x\hat{\ }i) - (\sum i{\leq}n.\ a\ i * y\hat{\ }i) =$
$(x - y) * ((\sum j{<}n.\ \sum k{<}n{-}j.\ a(j + k + 1) * y\hat{\ }k * x\hat{\ }j))$
$\langle proof \rangle$

**lemma** *polyfun-linear-factor*:
**fixes** $a :: 'a::idom$
**shows** $\exists\, b.\ \forall\, z.\ (\sum i{\leq}n.\ c(i) * z\hat{\ }i) = (z - a) * (\sum i{<}n.\ b(i) * z\hat{\ }i) + (\sum i{\leq}n.$
$c(i) * a\hat{\ }i)$
$\langle proof \rangle$

**lemma** *polyfun-linear-factor-root*:
**fixes** $a :: 'a::idom$
**assumes** $(\sum i{\leq}n.\ c(i) * a\hat{\ }i) = 0$
**obtains** $b$ **where** $\bigwedge z.\ (\sum i{\leq}n.\ c\ i * z\hat{\ }i) = (z - a) * (\sum i{<}n.\ b\ i * z\hat{\ }i)$
$\langle proof \rangle$

**lemma** *isCont-polynom*: $isCont\ (\lambda w.\ \sum i{\leq}n.\ c\ i * w\hat{\ }i)\ a$
**for** $c :: nat \Rightarrow 'a::real\text{-}normed\text{-}div\text{-}algebra$
$\langle proof \rangle$

**lemma** *zero-polynom-imp-zero-coeffs*:
**fixes** $c :: nat \Rightarrow 'a::\{ab\text{-}semigroup\text{-}mult,real\text{-}normed\text{-}div\text{-}algebra\}$
**assumes** $\bigwedge w.\ (\sum i{\leq}n.\ c\ i * w\hat{\ }i) = 0\ \ k \leq n$
**shows** $c\ k = 0$
$\langle proof \rangle$

**lemma** *polyfun-rootbound*:
**fixes** $c :: nat \Rightarrow 'a::\{idom,real\text{-}normed\text{-}div\text{-}algebra\}$
**assumes** $c\ k \neq 0\ k{\leq}n$
**shows** $finite\ \{z.\ (\sum i{\leq}n.\ c(i) * z\hat{\ }i) = 0\} \wedge card\ \{z.\ (\sum i{\leq}n.\ c(i) * z\hat{\ }i) = 0\}$
$\leq n$
$\langle proof \rangle$

**lemma**
**fixes** $c :: nat \Rightarrow 'a::\{idom,real\text{-}normed\text{-}div\text{-}algebra\}$
**assumes** $c\ k \neq 0\ k{\leq}n$
**shows** *polyfun-roots-finite*: $finite\ \{z.\ (\sum i{\leq}n.\ c(i) * z\hat{\ }i) = 0\}$
**and** *polyfun-roots-card*: $card\ \{z.\ (\sum i{\leq}n.\ c(i) * z\hat{\ }i) = 0\} \leq n$
$\langle proof \rangle$

**lemma** *polyfun-finite-roots*:
**fixes** $c :: nat \Rightarrow 'a::\{idom,real\text{-}normed\text{-}div\text{-}algebra\}$
**shows** $finite\ \{x.\ (\sum i{\leq}n.\ c\ i * x\hat{\ }i) = 0\} \longleftrightarrow (\exists\, i{\leq}n.\ c\ i \neq 0)$

(**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *polyfun-eq-0*: $(\forall x.\ (\sum i{\le}n.\ c\ i * x\hat{}i) = 0) \longleftrightarrow (\forall i{\le}n.\ c\ i = 0)$
**for** $c$ :: *nat* ⇒ *'a*::{*idom*,*real-normed-div-algebra*}

⟨*proof*⟩

**lemma** *polyfun-eq-coeffs*: $(\forall x.\ (\sum i{\le}n.\ c\ i * x\hat{}i) = (\sum i{\le}n.\ d\ i * x\hat{}i)) \longleftrightarrow (\forall i{\le}n.\ c\ i = d\ i)$
**for** $c$ :: *nat* ⇒ *'a*::{*idom*,*real-normed-div-algebra*}
⟨*proof*⟩

**lemma** *polyfun-eq-const*:
**fixes** $c$ :: *nat* ⇒ *'a*::{*idom*,*real-normed-div-algebra*}
**shows** $(\forall x.\ (\sum i{\le}n.\ c\ i * x\hat{}i) = k) \longleftrightarrow c\ 0 = k \wedge (\forall i \in \{1..n\}.\ c\ i = 0)$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

**lemma** *root-polyfun*:
**fixes** $z$ :: *'a*::*idom*
**assumes** $1 \le n$
**shows** $z\hat{}n = a \longleftrightarrow (\sum i{\le}n.\ (\mathit{if}\ i = 0\ \mathit{then}\ -a\ \mathit{else\ if}\ i{=}n\ \mathit{then}\ 1\ \mathit{else}\ 0) * z\hat{}i) = 0$
⟨*proof*⟩

**lemma**
**assumes** *SORT-CONSTRAINT*(*'a*::{*idom*,*real-normed-div-algebra*})
**and** $1 \le n$
**shows** *finite-roots-unity*: *finite* {*z*::*'a*. $z\hat{}n = 1$}
**and** *card-roots-unity*: *card* {*z*::*'a*. $z\hat{}n = 1$} $\le n$
⟨*proof*⟩

## 112.22 Hyperbolic functions

**definition** *sinh* :: *'a* :: {*banach*, *real-normed-algebra-1*} ⇒ *'a* **where**
$\mathit{sinh}\ x = (\mathit{exp}\ x - \mathit{exp}\ (-x))\ /_R\ 2$

**definition** *cosh* :: *'a* :: {*banach*, *real-normed-algebra-1*} ⇒ *'a* **where**
$\mathit{cosh}\ x = (\mathit{exp}\ x + \mathit{exp}\ (-x))\ /_R\ 2$

**definition** *tanh* :: *'a* :: {*banach*, *real-normed-field*} ⇒ *'a* **where**
$\mathit{tanh}\ x = \mathit{sinh}\ x\ /\ \mathit{cosh}\ x$

**definition** *arsinh* :: *'a* :: {*banach*, *real-normed-algebra-1*, *ln*} ⇒ *'a* **where**
$\mathit{arsinh}\ x = \mathit{ln}\ (x + (x\hat{}2 + 1)\ \mathit{powr}\ \mathit{of\text{-}real}\ (1/2))$

**definition** *arcosh* :: *'a* :: {*banach*, *real-normed-algebra-1*, *ln*} ⇒ *'a* **where**
$\mathit{arcosh}\ x = \mathit{ln}\ (x + (x\hat{}2 - 1)\ \mathit{powr}\ \mathit{of\text{-}real}\ (1/2))$

**definition** *artanh* :: $'a$ :: {*banach*, *real-normed-field*, *ln*} $\Rightarrow$ $'a$ **where**
*artanh x* = *ln* ((*1* + *x*) / (*1* − *x*)) / *2*

**lemma** *arsinh-0* [*simp*]: *arsinh 0* = *0*
⟨*proof*⟩

**lemma** *arcosh-1* [*simp*]: *arcosh 1* = *0*
⟨*proof*⟩

**lemma** *artanh-0* [*simp*]: *artanh 0* = *0*
⟨*proof*⟩

**lemma** *tanh-altdef*:
*tanh x* = (*exp x* − *exp* (−*x*)) / (*exp x* + *exp* (−*x*))
⟨*proof*⟩

**lemma** *tanh-real-altdef*: *tanh* (*x*::*real*) = (*1* − *exp* (− *2* ∗ *x*)) / (*1* + *exp* (− *2* ∗ *x*))
⟨*proof*⟩

**lemma** *sinh-converges*: ($\lambda n$. *if even n then 0 else x* ^ *n* $/_R$ *fact n*) *sums sinh x*
⟨*proof*⟩

**lemma** *cosh-converges*: ($\lambda n$. *if even n then x* ^ *n* $/_R$ *fact n else 0*) *sums cosh x*
⟨*proof*⟩

**lemma** *sinh-0* [*simp*]: *sinh 0* = *0*
⟨*proof*⟩

**lemma** *cosh-0* [*simp*]: *cosh 0* = *1*
⟨*proof*⟩

**lemma** *tanh-0* [*simp*]: *tanh 0* = *0*
⟨*proof*⟩

**lemma** *sinh-minus* [*simp*]: *sinh* (− *x*) = −*sinh x*
⟨*proof*⟩

**lemma** *cosh-minus* [*simp*]: *cosh* (− *x*) = *cosh x*
⟨*proof*⟩

**lemma** *tanh-minus* [*simp*]: *tanh* (−*x*) = −*tanh x*
⟨*proof*⟩

**lemma** *sinh-ln-real*: *x* > *0* $\Longrightarrow$ *sinh* (*ln x* :: *real*) = (*x* − *inverse x*) / *2*
⟨*proof*⟩

**lemma** *cosh-ln-real*: $x > 0 \Longrightarrow cosh\ (ln\ x :: real) = (x + inverse\ x)\ /\ 2$
⟨*proof*⟩

**lemma** *tanh-ln-real*:
$tanh\ (ln\ x :: real) = (x\ \hat{}\ 2 - 1)\ /\ (x\ \hat{}\ 2 + 1)$ **if** $x > 0$
⟨*proof*⟩

**lemma** *has-field-derivative-scaleR-right* [*derivative-intros*]:
$(f\ has\text{-}field\text{-}derivative\ D)\ F \Longrightarrow ((\lambda x.\ c *_R f\ x)\ has\text{-}field\text{-}derivative\ (c *_R D))\ F$
⟨*proof*⟩

**lemma** *has-field-derivative-sinh* [*THEN DERIV-chain2*, *derivative-intros*]:
$(sinh\ has\text{-}field\text{-}derivative\ cosh\ x)\ (at\ (x :: 'a :: \{banach,\ real\text{-}normed\text{-}field\}))$
⟨*proof*⟩

**lemma** *has-field-derivative-cosh* [*THEN DERIV-chain2*, *derivative-intros*]:
$(cosh\ has\text{-}field\text{-}derivative\ sinh\ x)\ (at\ (x :: 'a :: \{banach,\ real\text{-}normed\text{-}field\}))$
⟨*proof*⟩

**lemma** *has-field-derivative-tanh* [*THEN DERIV-chain2*, *derivative-intros*]:
$cosh\ x \neq 0 \Longrightarrow (tanh\ has\text{-}field\text{-}derivative\ 1 - tanh\ x\ \hat{}\ 2)$
$(at\ (x :: 'a :: \{banach,\ real\text{-}normed\text{-}field\}))$
⟨*proof*⟩

**lemma** *has-derivative-sinh* [*derivative-intros*]:
**fixes** $g :: 'a \Rightarrow ('a :: \{banach,\ real\text{-}normed\text{-}field\})$
**assumes** $(g\ has\text{-}derivative\ (\lambda x.\ Db * x))\ (at\ x\ within\ s)$
**shows** $((\lambda x.\ sinh\ (g\ x))\ has\text{-}derivative\ (\lambda y.\ (cosh\ (g\ x) * Db) * y))\ (at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *has-derivative-cosh* [*derivative-intros*]:
**fixes** $g :: 'a \Rightarrow ('a :: \{banach,\ real\text{-}normed\text{-}field\})$
**assumes** $(g\ has\text{-}derivative\ (\lambda y.\ Db * y))\ (at\ x\ within\ s)$
**shows** $((\lambda x.\ cosh\ (g\ x))\ has\text{-}derivative\ (\lambda y.\ (sinh\ (g\ x) * Db) * y))\ (at\ x\ within\ s)$
⟨*proof*⟩

**lemma** *sinh-plus-cosh*: $sinh\ x + cosh\ x = exp\ x$
⟨*proof*⟩

**lemma** *cosh-plus-sinh*: $cosh\ x + sinh\ x = exp\ x$
⟨*proof*⟩

**lemma** *cosh-minus-sinh*: $cosh\ x - sinh\ x = exp\ (-x)$
⟨*proof*⟩

**lemma** *sinh-minus-cosh*: $sinh\ x - cosh\ x = -exp\ (-x)$
⟨*proof*⟩

**context**
**fixes** $x$ :: $'a$ :: $\{$*real-normed-field*, *banach*$\}$
**begin**

**lemma** *sinh-zero-iff*: $sinh\ x = 0 \longleftrightarrow exp\ x \in \{1, -1\}$
⟨*proof*⟩

**lemma** *cosh-zero-iff*: $cosh\ x = 0 \longleftrightarrow exp\ x\ \hat{}\ 2 = -1$
⟨*proof*⟩

**lemma** *cosh-square-eq*: $cosh\ x\ \hat{}\ 2 = sinh\ x\ \hat{}\ 2 + 1$
⟨*proof*⟩

**lemma** *sinh-square-eq*: $sinh\ x\ \hat{}\ 2 = cosh\ x\ \hat{}\ 2 - 1$
⟨*proof*⟩

**lemma** *hyperbolic-pythagoras*: $cosh\ x\ \hat{}\ 2 - sinh\ x\ \hat{}\ 2 = 1$
⟨*proof*⟩

**lemma** *sinh-add*: $sinh\ (x + y) = sinh\ x * cosh\ y + cosh\ x * sinh\ y$
⟨*proof*⟩

**lemma** *sinh-diff*: $sinh\ (x - y) = sinh\ x * cosh\ y - cosh\ x * sinh\ y$
⟨*proof*⟩

**lemma** *cosh-add*: $cosh\ (x + y) = cosh\ x * cosh\ y + sinh\ x * sinh\ y$
⟨*proof*⟩

**lemma** *cosh-diff*: $cosh\ (x - y) = cosh\ x * cosh\ y - sinh\ x * sinh\ y$
⟨*proof*⟩

**lemma** *tanh-add*:
$tanh\ (x + y) = (tanh\ x + tanh\ y)\ /\ (1 + tanh\ x * tanh\ y)$
**if** $cosh\ x \neq 0$ $cosh\ y \neq 0$
⟨*proof*⟩

**lemma** *sinh-double*: $sinh\ (2 * x) = 2 * sinh\ x * cosh\ x$
⟨*proof*⟩

**lemma** *cosh-double*: $cosh\ (2 * x) = cosh\ x\ \hat{}\ 2 + sinh\ x\ \hat{}\ 2$
⟨*proof*⟩

**end**

**lemma** *sinh-field-def*: $sinh\ z = (exp\ z - exp\ (-z))\ /\ (2$ :: $'a$ :: $\{$*banach*, *real-normed-field*$\})$
⟨*proof*⟩

**lemma** *cosh-field-def*: $cosh\ z = (exp\ z + exp\ (-z))\ /\ (2 :: 'a :: \{banach, real\text{-}normed\text{-}field\})$
⟨*proof*⟩

### 112.22.1 More specific properties of the real functions

**lemma** *plus-inverse-ge-2*:
**fixes** $x :: real$
**assumes** $x > 0$
**shows** $x + inverse\ x \geq 2$
⟨*proof*⟩

**lemma** *sinh-real-nonneg-iff* [*simp*]: $sinh\ (x :: real) \geq 0 \longleftrightarrow x \geq 0$
⟨*proof*⟩

**lemma** *sinh-real-pos-iff* [*simp*]: $sinh\ (x :: real) > 0 \longleftrightarrow x > 0$
⟨*proof*⟩

**lemma** *sinh-real-nonpos-iff* [*simp*]: $sinh\ (x :: real) \leq 0 \longleftrightarrow x \leq 0$
⟨*proof*⟩

**lemma** *sinh-real-neg-iff* [*simp*]: $sinh\ (x :: real) < 0 \longleftrightarrow x < 0$
⟨*proof*⟩

**lemma** *cosh-real-ge-1*: $cosh\ (x :: real) \geq 1$
⟨*proof*⟩

**lemma** *cosh-real-pos* [*simp*]: $cosh\ (x :: real) > 0$
⟨*proof*⟩

**lemma** *cosh-real-nonneg*[*simp*]: $cosh\ (x :: real) \geq 0$
⟨*proof*⟩

**lemma** *cosh-real-nonzero* [*simp*]: $cosh\ (x :: real) \neq 0$
⟨*proof*⟩

**lemma** *arsinh-real-def*: $arsinh\ (x::real) = ln\ (x + sqrt\ (x\hat{}2 + 1))$
⟨*proof*⟩

**lemma** *arcosh-real-def*: $x \geq 1 \Longrightarrow arcosh\ (x::real) = ln\ (x + sqrt\ (x\hat{}2 - 1))$
⟨*proof*⟩

**lemma** *arsinh-real-aux*: $0 < x + sqrt\ (x\ \hat{}\ 2 + 1 :: real)$
⟨*proof*⟩

**lemma** *arsinh-minus-real* [*simp*]: $arsinh\ (-x::real) = -arsinh\ x$
⟨*proof*⟩

**lemma** *artanh-minus-real* [*simp*]:
**assumes** $abs\ x < 1$

**shows** $artanh\ (-x::real) = -artanh\ x$
⟨*proof*⟩

**lemma** *sinh-less-cosh-real*: $sinh\ (x :: real) < cosh\ x$
⟨*proof*⟩

**lemma** *sinh-le-cosh-real*: $sinh\ (x :: real) \leq cosh\ x$
⟨*proof*⟩

**lemma** *tanh-real-lt-1*: $tanh\ (x :: real) < 1$
⟨*proof*⟩

**lemma** *tanh-real-gt-neg1*: $tanh\ (x :: real) > -1$
⟨*proof*⟩

**lemma** *tanh-real-bounds*: $tanh\ (x :: real) \in \{-1<..<1\}$
⟨*proof*⟩

**context**
**fixes** $x :: real$
**begin**

**lemma** *arsinh-sinh-real*: $arsinh\ (sinh\ x) = x$
⟨*proof*⟩

**lemma** *arcosh-cosh-real*: $x \geq 0 \Longrightarrow arcosh\ (cosh\ x) = x$
⟨*proof*⟩

**lemma** *artanh-tanh-real*: $artanh\ (tanh\ x) = x$
⟨*proof*⟩

**lemma** *sinh-real-zero-iff* [*simp*]: $sinh\ x = 0 \longleftrightarrow x = 0$
⟨*proof*⟩

**lemma** *cosh-real-one-iff* [*simp*]: $cosh\ x = 1 \longleftrightarrow x = 0$
⟨*proof*⟩

**lemma** *tanh-real-nonneg-iff* [*simp*]: $tanh\ x \geq 0 \longleftrightarrow x \geq 0$
⟨*proof*⟩

**lemma** *tanh-real-pos-iff* [*simp*]: $tanh\ x > 0 \longleftrightarrow x > 0$
⟨*proof*⟩

**lemma** *tanh-real-nonpos-iff* [*simp*]: $tanh\ x \leq 0 \longleftrightarrow x \leq 0$
⟨*proof*⟩

**lemma** *tanh-real-neg-iff* [*simp*]: $tanh\ x < 0 \longleftrightarrow x < 0$
⟨*proof*⟩

**lemma** *tanh-real-zero-iff* [*simp*]: $tanh\ x = 0 \longleftrightarrow x = 0$
⟨*proof*⟩

**end**

**lemma** *sinh-real-strict-mono*: $strict\text{-}mono\ (sinh :: real \Rightarrow real)$
⟨*proof*⟩

**lemma** *cosh-real-strict-mono*:
**assumes** $0 \leq x$ **and** $x < (y::real)$
**shows** $cosh\ x < cosh\ y$
⟨*proof*⟩

**lemma** *tanh-real-strict-mono*: $strict\text{-}mono\ (tanh :: real \Rightarrow real)$
⟨*proof*⟩

**lemma** *sinh-real-abs* [*simp*]: $sinh\ (abs\ x :: real) = abs\ (sinh\ x)$
⟨*proof*⟩

**lemma** *cosh-real-abs* [*simp*]: $cosh\ (abs\ x :: real) = cosh\ x$
⟨*proof*⟩

**lemma** *tanh-real-abs* [*simp*]: $tanh\ (abs\ x :: real) = abs\ (tanh\ x)$
⟨*proof*⟩

**lemma** *sinh-real-eq-iff* [*simp*]: $sinh\ x = sinh\ y \longleftrightarrow x = (y :: real)$
⟨*proof*⟩

**lemma** *tanh-real-eq-iff* [*simp*]: $tanh\ x = tanh\ y \longleftrightarrow x = (y :: real)$
⟨*proof*⟩

**lemma** *cosh-real-eq-iff* [*simp*]: $cosh\ x = cosh\ y \longleftrightarrow abs\ x = abs\ (y :: real)$
⟨*proof*⟩

**lemma** *sinh-real-le-iff* [*simp*]: $sinh\ x \leq sinh\ y \longleftrightarrow x \leq (y::real)$
⟨*proof*⟩

**lemma** *cosh-real-nonneg-le-iff*: $x \geq 0 \Longrightarrow y \geq 0 \Longrightarrow cosh\ x \leq cosh\ y \longleftrightarrow x \leq (y::real)$
⟨*proof*⟩

**lemma** *cosh-real-nonpos-le-iff*: $x \leq 0 \Longrightarrow y \leq 0 \Longrightarrow cosh\ x \leq cosh\ y \longleftrightarrow x \geq (y::real)$
⟨*proof*⟩

**lemma** *tanh-real-le-iff* [*simp*]: $tanh\ x \leq tanh\ y \longleftrightarrow x \leq (y::real)$
⟨*proof*⟩

**lemma** *sinh-real-less-iff* [*simp*]: *sinh x* < *sinh y* ⟷ *x* < (*y*::*real*)
⟨*proof*⟩

**lemma** *cosh-real-nonneg-less-iff*: *x* ≥ *0* ⟹ *y* ≥ *0* ⟹ *cosh x* < *cosh y* ⟷ *x* < (*y*::*real*)
⟨*proof*⟩

**lemma** *cosh-real-nonpos-less-iff*: *x* ≤ *0* ⟹ *y* ≤ *0* ⟹ *cosh x* < *cosh y* ⟷ *x* > (*y*::*real*)
⟨*proof*⟩

**lemma** *tanh-real-less-iff* [*simp*]: *tanh x* < *tanh y* ⟷ *x* < (*y*::*real*)
⟨*proof*⟩

### 112.22.2 Limits

**lemma** *sinh-real-at-top*: *filterlim* (*sinh* :: *real* ⇒ *real*) *at-top at-top*
⟨*proof*⟩

**lemma** *sinh-real-at-bot*: *filterlim* (*sinh* :: *real* ⇒ *real*) *at-bot at-bot*
⟨*proof*⟩

**lemma** *cosh-real-at-top*: *filterlim* (*cosh* :: *real* ⇒ *real*) *at-top at-top*
⟨*proof*⟩

**lemma** *cosh-real-at-bot*: *filterlim* (*cosh* :: *real* ⇒ *real*) *at-top at-bot*
⟨*proof*⟩

**lemma** *tanh-real-at-top*: (*tanh* ⟶ (*1*::*real*)) *at-top*
⟨*proof*⟩

**lemma** *tanh-real-at-bot*: (*tanh* ⟶ (−*1*::*real*)) *at-bot*
⟨*proof*⟩

### 112.22.3 Properties of the inverse hyperbolic functions

**lemma** *isCont-sinh*: *isCont sinh* (*x* :: ′*a* :: {*real-normed-field*, *banach*})
⟨*proof*⟩

**lemma** *isCont-cosh*: *isCont cosh* (*x* :: ′*a* :: {*real-normed-field*, *banach*})
⟨*proof*⟩

**lemma** *isCont-tanh*: *cosh x* ≠ *0* ⟹ *isCont tanh* (*x* :: ′*a* :: {*real-normed-field*, *banach*})
⟨*proof*⟩

**lemma** *continuous-on-sinh* [*continuous-intros*]:
**fixes** *f* :: - ⇒′*a*::{*real-normed-field*,*banach*}
**assumes** *continuous-on A f*
**shows** *continuous-on A* (λ*x*. *sinh* (*f x*))

⟨*proof*⟩

**lemma** *continuous-on-cosh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous-on A f*
**shows** *continuous-on A* ($\lambda x$. *cosh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-sinh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous F f*
**shows** *continuous F* ($\lambda x$. *sinh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-cosh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous F f*
**shows** *continuous F* ($\lambda x$. *cosh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-on-tanh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous-on A f* $\bigwedge x.\ x \in A \Longrightarrow$ *cosh* ($f$ $x$) $\neq$ *0*
**shows** *continuous-on A* ($\lambda x$. *tanh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-at-within-tanh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous* (*at x within A*) *f cosh* ($f$ $x$) $\neq$ *0*
**shows** *continuous* (*at x within A*) ($\lambda x$. *tanh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-tanh* [*continuous-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**assumes** *continuous F f cosh* ($f$ (*Lim F* ($\lambda x$. $x$))) $\neq$ *0*
**shows** *continuous F* ($\lambda x$. *tanh* ($f$ $x$))
⟨*proof*⟩

**lemma** *tendsto-sinh* [*tendsto-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**shows** ($f$ $\longrightarrow$ $a$) $F$ $\Longrightarrow$ (($\lambda x$. *sinh* ($f$ $x$)) $\longrightarrow$ *sinh* $a$) $F$
⟨*proof*⟩

**lemma** *tendsto-cosh* [*tendsto-intros*]:
**fixes** $f$ :: - ⇒'$a$::{*real-normed-field*,*banach*}
**shows** ($f$ $\longrightarrow$ $a$) $F$ $\Longrightarrow$ (($\lambda x$. *cosh* ($f$ $x$)) $\longrightarrow$ *cosh* $a$) $F$
⟨*proof*⟩

**lemma** *tendsto-tanh* [*tendsto-intros*]:

**fixes** $f$ :: - $\Rightarrow$'$a$::{$real$-$normed$-$field$,$banach$}
**shows** ($f$ $\longrightarrow$ $a$) $F$ $\Longrightarrow$ $cosh$ $a \neq 0$ $\Longrightarrow$ (($\lambda x$. $tanh$ ($f$ $x$)) $\longrightarrow$ $tanh$ $a$) $F$
⟨*proof*⟩

**lemma** *arsinh-real-has-field-derivative* [*derivative-intros*]:
**fixes** $x$ :: *real*
**shows** (*arsinh has-field-derivative* ($1$ / (*sqrt* ($x$ ^ $2$ + $1$)))) (*at* $x$ *within* $A$)
⟨*proof*⟩

**lemma** *arcosh-real-has-field-derivative* [*derivative-intros*]:
**fixes** $x$ :: *real*
**assumes** $x > 1$
**shows** (*arcosh has-field-derivative* ($1$ / (*sqrt* ($x$ ^ $2$ − $1$)))) (*at* $x$ *within* $A$)
⟨*proof*⟩

**lemma** *artanh-real-has-field-derivative* [*derivative-intros*]:
(*artanh has-field-derivative* ($1$ / ($1$ − $x$ ^ $2$))) (*at* $x$ *within* $A$) **if**
$|x| < 1$ **for** $x$ :: *real*
⟨*proof*⟩

**lemma** *cosh-double-cosh*: *cosh* ($2$ ∗ $x$ :: '$a$ :: {*banach*, *real-normed-field*}) = $2$ ∗ (*cosh* $x$)$^2$ − $1$
⟨*proof*⟩

**lemma** *sinh-multiple-reduce*:
*sinh* ($x$ ∗ *numeral* $n$ :: '$a$ :: {*real-normed-field*, *banach*}) =
*sinh* $x$ ∗ *cosh* ($x$ ∗ *of-nat* (*pred-numeral* $n$)) + *cosh* $x$ ∗ *sinh* ($x$ ∗ *of-nat* (*pred-numeral* $n$))
⟨*proof*⟩

**lemma** *cosh-multiple-reduce*:
*cosh* ($x$ ∗ *numeral* $n$ :: '$a$ :: {*real-normed-field*, *banach*}) =
*cosh* ($x$ ∗ *of-nat* (*pred-numeral* $n$)) ∗ *cosh* $x$ + *sinh* ($x$ ∗ *of-nat* (*pred-numeral* $n$)) ∗ *sinh* $x$
⟨*proof*⟩

**lemma** *cosh-arcosh-real* [*simp*]:
**assumes** $x \geq$ ($1$ :: *real*)
**shows** *cosh* (*arcosh* $x$) = $x$
⟨*proof*⟩

**lemma** *arcosh-eq-0-iff-real* [*simp*]: $x \geq 1 \Longrightarrow$ *arcosh* $x = 0 \longleftrightarrow x =$ ($1$ :: *real*)
⟨*proof*⟩

**lemma** *arcosh-nonneg-real* [*simp*]:
**assumes** $x \geq 1$
**shows** *arcosh* ($x$ :: *real*) $\geq 0$
⟨*proof*⟩

**lemma** *arcosh-real-strict-mono*:
**fixes** $x\ y :: real$
**assumes** $1 \leq x\ x < y$
**shows** $arcosh\ x < arcosh\ y$
⟨*proof*⟩

**lemma** *arcosh-less-iff-real* [*simp*]:
**fixes** $x\ y :: real$
**assumes** $1 \leq x\ 1 \leq y$
**shows** $arcosh\ x < arcosh\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *arcosh-real-gt-1-iff* [*simp*]: $x \geq 1 \Longrightarrow arcosh\ x > 0 \longleftrightarrow x \neq (1 :: real)$
⟨*proof*⟩

**lemma** *sinh-arcosh-real*: $x \geq 1 \Longrightarrow sinh\ (arcosh\ x) = sqrt\ (x^2 - 1)$
⟨*proof*⟩

**lemma** *sinh-arsinh-real* [*simp*]: $sinh\ (arsinh\ x :: real) = x$
⟨*proof*⟩

**lemma** *arsinh-real-strict-mono*:
**fixes** $x\ y :: real$
**assumes** $x < y$
**shows** $arsinh\ x < arsinh\ y$
⟨*proof*⟩

**lemma** *arsinh-less-iff-real* [*simp*]:
**fixes** $x\ y :: real$
**shows** $arsinh\ x < arsinh\ y \longleftrightarrow x < y$
⟨*proof*⟩

**lemma** *arsinh-real-eq-0-iff* [*simp*]: $arsinh\ x = 0 \longleftrightarrow x = (0 :: real)$
⟨*proof*⟩

**lemma** *arsinh-real-pos-iff* [*simp*]: $arsinh\ x > 0 \longleftrightarrow x > (0 :: real)$
⟨*proof*⟩

**lemma** *arsinh-real-neg-iff* [*simp*]: $arsinh\ x < 0 \longleftrightarrow x < (0 :: real)$
⟨*proof*⟩

**lemma** *cosh-arsinh-real*: $cosh\ (arsinh\ x) = sqrt\ (x^2 + 1)$
⟨*proof*⟩

**lemma** *continuous-on-arsinh* [*continuous-intros*]: $continuous\text{-}on\ A\ (arsinh :: real \Rightarrow real)$
⟨*proof*⟩

**lemma** *continuous-on-arcosh* [*continuous-intros*]:
**assumes** $A \subseteq \{1..\}$
**shows** *continuous-on* $A$ (*arcosh* :: *real* $\Rightarrow$ *real*)
⟨*proof*⟩

**lemma** *continuous-on-artanh* [*continuous-intros*]:
**assumes** $A \subseteq \{-1<..<1\}$
**shows** *continuous-on* $A$ (*artanh* :: *real* $\Rightarrow$ *real*)
⟨*proof*⟩

**lemma** *continuous-on-arsinh'* [*continuous-intros*]:
**fixes** $f$ :: *real* $\Rightarrow$ *real*
**assumes** *continuous-on* $A$ $f$
**shows** *continuous-on* $A$ ($\lambda x.$ *arsinh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-on-arcosh'* [*continuous-intros*]:
**fixes** $f$ :: *real* $\Rightarrow$ *real*
**assumes** *continuous-on* $A$ $f$ $\bigwedge x.\ x \in A \Longrightarrow f\ x \geq 1$
**shows** *continuous-on* $A$ ($\lambda x.$ *arcosh* ($f$ $x$))
⟨*proof*⟩

**lemma** *continuous-on-artanh'* [*continuous-intros*]:
**fixes** $f$ :: *real* $\Rightarrow$ *real*
**assumes** *continuous-on* $A$ $f$ $\bigwedge x.\ x \in A \Longrightarrow f\ x \in \{-1<..<1\}$
**shows** *continuous-on* $A$ ($\lambda x.$ *artanh* ($f$ $x$))
⟨*proof*⟩

**lemma** *isCont-arsinh* [*continuous-intros*]: *isCont arsinh* ($x$ :: *real*)
⟨*proof*⟩

**lemma** *isCont-arcosh* [*continuous-intros*]:
**assumes** $x > 1$
**shows** *isCont arcosh* ($x$ :: *real*)
⟨*proof*⟩

**lemma** *isCont-artanh* [*continuous-intros*]:
**assumes** $x > -1$ $x < 1$
**shows** *isCont artanh* ($x$ :: *real*)
⟨*proof*⟩

**lemma** *tendsto-arsinh* [*tendsto-intros*]: ($f \longrightarrow a$) $F \Longrightarrow$ (($\lambda x.$ *arsinh* ($f$ $x$)) $\longrightarrow$ *arsinh* $a$) $F$
**for** $f$ :: - $\Rightarrow$ *real*
⟨*proof*⟩

**lemma** *tendsto-arcosh-strong* [*tendsto-intros*]:
**fixes** $f$ :: - $\Rightarrow$ *real*

**assumes** $(f \longrightarrow a)$ $F$ $a \geq 1$ *eventually* $(\lambda x.\ f\ x \geq 1)$ $F$
**shows** $((\lambda x.\ arcosh\ (f\ x)) \longrightarrow arcosh\ a)$ $F$
⟨*proof*⟩

**lemma** *tendsto-arcosh*:
**fixes** $f$ :: - ⇒ *real*
**assumes** $(f \longrightarrow a)$ $F$ $a > 1$
**shows** $((\lambda x.\ arcosh\ (f\ x)) \longrightarrow arcosh\ a)$ $F$
⟨*proof*⟩

**lemma** *tendsto-arcosh-at-left-1*: $(arcosh \longrightarrow 0)$ $(at\text{-}right\ (1::real))$
⟨*proof*⟩

**lemma** *tendsto-artanh* [*tendsto-intros*]:
**fixes** $f$ :: $'a \Rightarrow real$
**assumes** $(f \longrightarrow a)$ $F$ $a > -1$ $a < 1$
**shows** $((\lambda x.\ artanh\ (f\ x)) \longrightarrow artanh\ a)$ $F$
⟨*proof*⟩

**lemma** *continuous-arsinh* [*continuous-intros*]:
*continuous* $F$ $f \Longrightarrow$ *continuous* $F$ $(\lambda x.\ arsinh\ (f\ x :: real))$
⟨*proof*⟩

**lemma** *continuous-arcosh-strong* [*continuous-intros*]:
**assumes** *continuous* $F$ $f$ *eventually* $(\lambda x.\ f\ x \geq 1)$ $F$
**shows** *continuous* $F$ $(\lambda x.\ arcosh\ (f\ x :: real))$
⟨*proof*⟩

**lemma** *continuous-arcosh*:
*continuous* $F$ $f \Longrightarrow f\ (Lim\ F\ (\lambda x.\ x)) > 1 \Longrightarrow$ *continuous* $F$ $(\lambda x.\ arcosh\ (f\ x :: real))$
⟨*proof*⟩

**lemma** *continuous-artanh* [*continuous-intros*]:
*continuous* $F$ $f \Longrightarrow f\ (Lim\ F\ (\lambda x.\ x)) \in \{-1<..<1\} \Longrightarrow$ *continuous* $F$ $(\lambda x.\ artanh\ (f\ x :: real))$
⟨*proof*⟩

**lemma** *arsinh-real-at-top*:
*filterlim* $(arsinh :: real \Rightarrow real)$ *at-top at-top*
⟨*proof*⟩

**lemma** *arsinh-real-at-bot*:
*filterlim* $(arsinh :: real \Rightarrow real)$ *at-bot at-bot*
⟨*proof*⟩

**lemma** *arcosh-real-at-top*:
*filterlim* $(arcosh :: real \Rightarrow real)$ *at-top at-top*

⟨*proof*⟩

**lemma** *artanh-real-at-left-1*:
*filterlim* (*artanh* :: *real* ⇒ *real*) *at-top* (*at-left 1*)
⟨*proof*⟩

**lemma** *artanh-real-at-right-1*:
*filterlim* (*artanh* :: *real* ⇒ *real*) *at-bot* (*at-right* (−*1*))
⟨*proof*⟩

## 112.23 Simprocs for root and power literals

**lemma** *numeral-powr-numeral-real* [*simp*]:
*numeral m powr numeral n* = (*numeral m* ^ *numeral n* :: *real*)
⟨*proof*⟩

**context**
**begin**

**private lemma** *sqrt-numeral-simproc-aux*:
**assumes** *m* ∗ *m* ≡ *n*
**shows** *sqrt* (*numeral n* :: *real*) ≡ *numeral m*
⟨*proof*⟩ **lemma** *root-numeral-simproc-aux*:
**assumes** *Num.pow m n* ≡ *x*
**shows** *root* (*numeral n*) (*numeral x* :: *real*) ≡ *numeral m*
⟨*proof*⟩ **lemma** *powr-numeral-simproc-aux*:
**assumes** *Num.pow y n* = *x*
**shows** *numeral x powr* (*m* / *numeral n* :: *real*) ≡ *numeral y powr m*
⟨*proof*⟩ **lemma** *numeral-powr-inverse-eq*:
*numeral x powr* (*inverse* (*numeral n*)) = *numeral x powr* (*1* / *numeral n* :: *real*)
⟨*proof*⟩

⟨*ML*⟩

**end**

⟨*ML*⟩

**lemma** *root 100 1267650600228229401496703205376* = *2*
⟨*proof*⟩

**lemma** *sqrt 196* = *14*
⟨*proof*⟩

**lemma** *256 powr* (*7* / *4* :: *real*) = *16384*
⟨*proof*⟩

**lemma** $27$ *powr* (*inverse* $3$) $=$ ($3$::*real*)
⟨*proof*⟩

**end**

# 113 Complex Numbers: Rectangular and Polar Representations

**theory** *Complex*
**imports** *Transcendental Real-Vector-Spaces*
**begin**

We use the **codatatype** command to define the type of complex numbers. This allows us to use **primcorec** to define complex functions by defining their real and imaginary result separately.

**codatatype** *complex* = *Complex* (*Re*: *real*) (*Im*: *real*)

**lemma** *complex-surj*: *Complex* (*Re z*) (*Im z*) = *z*
⟨*proof*⟩

**lemma** *complex-eqI* [*intro?*]: *Re x* = *Re y* $\Longrightarrow$ *Im x* = *Im y* $\Longrightarrow$ *x* = *y*
⟨*proof*⟩

**lemma** *complex-eq-iff*: *x* = *y* $\longleftrightarrow$ *Re x* = *Re y* $\wedge$ *Im x* = *Im y*
⟨*proof*⟩

## 113.1 Addition and Subtraction

**instantiation** *complex* :: *ab-group-add*
**begin**

**primcorec** *zero-complex*
**where**
*Re* $0$ = $0$
| *Im* $0$ = $0$

**primcorec** *plus-complex*
**where**
*Re* (*x* + *y*) = *Re x* + *Re y*
| *Im* (*x* + *y*) = *Im x* + *Im y*

**primcorec** *uminus-complex*
**where**
*Re* (− *x*) = − *Re x*
| *Im* (− *x*) = − *Im x*

**primcorec** *minus-complex*
**where**

  $Re\ (x - y) = Re\ x - Re\ y$
| $Im\ (x - y) = Im\ x - Im\ y$

**instance**
⟨*proof*⟩

**end**

## 113.2 Multiplication and Division

**instantiation** *complex* :: *field*
**begin**

**primcorec** *one-complex*
**where**
  $Re\ 1 = 1$
| $Im\ 1 = 0$

**primcorec** *times-complex*
**where**
  $Re\ (x * y) = Re\ x * Re\ y - Im\ x * Im\ y$
| $Im\ (x * y) = Re\ x * Im\ y + Im\ x * Re\ y$

**primcorec** *inverse-complex*
**where**
  $Re\ (inverse\ x) = Re\ x\ /\ ((Re\ x)^2 + (Im\ x)^2)$
| $Im\ (inverse\ x) = -\ Im\ x\ /\ ((Re\ x)^2 + (Im\ x)^2)$

**definition** $x\ div\ y = x * inverse\ y$ **for** $x\ y$ :: *complex*

**instance**
⟨*proof*⟩

**end**

**lemma** *Re-divide*: $Re\ (x\ /\ y) = (Re\ x * Re\ y + Im\ x * Im\ y)\ /\ ((Re\ y)^2 + (Im\ y)^2)$
⟨*proof*⟩

**lemma** *Im-divide*: $Im\ (x\ /\ y) = (Im\ x * Re\ y - Re\ x * Im\ y)\ /\ ((Re\ y)^2 + (Im\ y)^2)$
⟨*proof*⟩

**lemma** *Complex-divide*:
$(x\ /\ y) = Complex\ ((Re\ x * Re\ y + Im\ x * Im\ y)\ /\ ((Re\ y)^2 + (Im\ y)^2))$
$((Im\ x * Re\ y - Re\ x * Im\ y)\ /\ ((Re\ y)^2 + (Im\ y)^2))$
⟨*proof*⟩

**lemma** *Re-power2*: $Re\ (x\ \hat{}\ 2) = (Re\ x)\hat{}2 - (Im\ x)\hat{}2$

⟨proof⟩

**lemma** *Im-power2*: $Im\ (x\ \hat{}\ 2) = 2 * Re\ x * Im\ x$
⟨proof⟩

**lemma** *Re-power-real* [*simp*]: $Im\ x = 0 \Longrightarrow Re\ (x\ \hat{}\ n) = Re\ x\ \hat{}\ n$
⟨proof⟩

**lemma** *Im-power-real* [*simp*]: $Im\ x = 0 \Longrightarrow Im\ (x\ \hat{}\ n) = 0$
⟨proof⟩

## 113.3 Scalar Multiplication

**instantiation** *complex* :: *real-field*
**begin**

**primcorec** *scaleR-complex*
**where**
$Re\ (scaleR\ r\ x) = r * Re\ x$
$|\ Im\ (scaleR\ r\ x) = r * Im\ x$

**instance**
⟨proof⟩

**end**

## 113.4 Numerals, Arithmetic, and Embedding from R

**declare** [[*coercion of-real* :: *real* ⇒ *complex*]]
**declare** [[*coercion of-rat* :: *rat* ⇒ *complex*]]
**declare** [[*coercion of-int* :: *int* ⇒ *complex*]]
**declare** [[*coercion of-nat* :: *nat* ⇒ *complex*]]

**abbreviation** *complex-of-nat*::*nat* ⇒ *complex*
**where** *complex-of-nat* ≡ *of-nat*

**abbreviation** *complex-of-int*::*int* ⇒ *complex*
**where** *complex-of-int* ≡ *of-int*

**abbreviation** *complex-of-rat*::*rat* ⇒ *complex*
**where** *complex-of-rat* ≡ *of-rat*

**abbreviation** *complex-of-real* :: *real* ⇒ *complex*
**where** *complex-of-real* ≡ *of-real*

**lemma** *complex-Re-of-nat* [*simp*]: $Re\ (of\text{-}nat\ n) = of\text{-}nat\ n$
⟨proof⟩

**lemma** *complex-Im-of-nat* [*simp*]: $Im\ (of\text{-}nat\ n) = 0$
⟨proof⟩

**lemma** *complex-Re-of-int* [*simp*]: *Re* (*of-int* *z*) = *of-int* *z*
⟨*proof*⟩

**lemma** *complex-Im-of-int* [*simp*]: *Im* (*of-int* *z*) = *0*
⟨*proof*⟩

**lemma** *complex-Re-numeral* [*simp*]: *Re* (*numeral* *v*) = *numeral* *v*
⟨*proof*⟩

**lemma** *complex-Im-numeral* [*simp*]: *Im* (*numeral* *v*) = *0*
⟨*proof*⟩

**lemma** *Re-complex-of-real* [*simp*]: *Re* (*complex-of-real* *z*) = *z*
⟨*proof*⟩

**lemma** *Im-complex-of-real* [*simp*]: *Im* (*complex-of-real* *z*) = *0*
⟨*proof*⟩

**lemma** *Re-divide-numeral* [*simp*]: *Re* (*z* / *numeral* *w*) = *Re* *z* / *numeral* *w*
⟨*proof*⟩

**lemma** *Im-divide-numeral* [*simp*]: *Im* (*z* / *numeral* *w*) = *Im* *z* / *numeral* *w*
⟨*proof*⟩

**lemma** *Re-divide-of-nat* [*simp*]: *Re* (*z* / *of-nat* *n*) = *Re* *z* / *of-nat* *n*
⟨*proof*⟩

**lemma** *Im-divide-of-nat* [*simp*]: *Im* (*z* / *of-nat* *n*) = *Im* *z* / *of-nat* *n*
⟨*proof*⟩

**lemma** *Re-inverse* [*simp*]: $r \in \mathbb{R} \Longrightarrow$ *Re* (*inverse* *r*) = *inverse* (*Re* *r*)
⟨*proof*⟩

**lemma** *Im-inverse* [*simp*]: $r \in \mathbb{R} \Longrightarrow$ *Im* (*inverse* *r*) = *0*
⟨*proof*⟩

**lemma** *of-real-Re* [*simp*]: $z \in \mathbb{R} \Longrightarrow$ *of-real* (*Re* *z*) = *z*
⟨*proof*⟩

**lemma** *complex-Re-fact* [*simp*]: *Re* (*fact* *n*) = *fact* *n*
⟨*proof*⟩

**lemma** *surj-Re*: *surj* *Re*
⟨*proof*⟩

**lemma** *surj-Im*: *surj* *Im*
⟨*proof*⟩

**lemma** *complex-Im-fact* [*simp*]: *Im* (*fact n*) = *0*
⟨*proof*⟩

**lemma** *Re-prod-Reals*: ($\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{R}) \Longrightarrow Re\ (prod\ f\ A) = prod\ (\lambda x.\ Re\ (f\ x))\ A$
⟨*proof*⟩

## 113.5 The Complex Number *i*

**primcorec** *imaginary-unit* :: *complex* (‹i›)
**where**
*Re* i = *0*
| *Im* i = *1*

**lemma** *Complex-eq*: *Complex a b* = *a* + i ∗ *b*
⟨*proof*⟩

**lemma** *complex-eq*: *a* = *Re a* + i ∗ *Im a*
⟨*proof*⟩

**lemma** *fun-complex-eq*: *f* = (λ*x*. *Re* (*f x*) + i ∗ *Im* (*f x*))
⟨*proof*⟩

**lemma** *i-squared* [*simp*]: i ∗ i = −*1*
⟨*proof*⟩

**lemma** *power2-i* [*simp*]: $\mathrm{i}^2 = -1$
⟨*proof*⟩

**lemma** *inverse-i* [*simp*]: *inverse* i = − i
⟨*proof*⟩

**lemma** *divide-i* [*simp*]: *x* / i = − i ∗ *x*
⟨*proof*⟩

**lemma** *complex-i-mult-minus* [*simp*]: i ∗ (i ∗ *x*) = − *x*
⟨*proof*⟩

**lemma** *complex-i-not-zero* [*simp*]: i ≠ *0*
⟨*proof*⟩

**lemma** *complex-i-not-one* [*simp*]: i ≠ *1*
⟨*proof*⟩

**lemma** *complex-i-not-numeral* [*simp*]: i ≠ *numeral w*
⟨*proof*⟩

**lemma** *complex-i-not-neg-numeral* [*simp*]: i ≠ − *numeral w*
⟨*proof*⟩

**lemma** *complex-split-polar*: $\exists\, r\ a.\ z = \textit{complex-of-real}\ r * (\cos a + \mathrm{i} * \sin a)$
⟨*proof*⟩

**lemma** *i-even-power* [*simp*]: $\mathrm{i} \,\hat{}\, (n * 2) = (-1) \,\hat{}\, n$
⟨*proof*⟩

**lemma** *i-even-power′* [*simp*]: $\textit{even}\ n \Longrightarrow \mathrm{i} \,\hat{}\, n = (-1) \,\hat{}\, (n\ \textit{div}\ 2)$
⟨*proof*⟩

**lemma** *Re-i-times* [*simp*]: $\textit{Re}\ (\mathrm{i} * z) = -\ \textit{Im}\ z$
⟨*proof*⟩

**lemma** *Im-i-times* [*simp*]: $\textit{Im}\ (\mathrm{i} * z) = \textit{Re}\ z$
⟨*proof*⟩

**lemma** *i-times-eq-iff*: $\mathrm{i} * w = z \longleftrightarrow w = -\ (\mathrm{i} * z)$
⟨*proof*⟩

**lemma** *divide-numeral-i* [*simp*]: $z\ /\ (\textit{numeral}\ n * \mathrm{i}) = -\ (\mathrm{i} * z)\ /\ \textit{numeral}\ n$
⟨*proof*⟩

**lemma** *imaginary-eq-real-iff* [*simp*]:
**assumes** $y \in \textit{Reals}\ x \in \textit{Reals}$
**shows** $\mathrm{i} * y = x \longleftrightarrow x{=}0 \wedge y{=}0$
⟨*proof*⟩

**lemma** *real-eq-imaginary-iff* [*simp*]:
**assumes** $y \in \textit{Reals}\ x \in \textit{Reals}$
**shows** $x = \mathrm{i} * y \ \longleftrightarrow x{=}0 \wedge y{=}0$
⟨*proof*⟩

## 113.6 Vector Norm

**instantiation** *complex* :: *real-normed-field*
**begin**

**definition** $\textit{norm}\ z = \textit{sqrt}\ ((\textit{Re}\ z)^2 + (\textit{Im}\ z)^2)$

**abbreviation** *cmod* :: *complex* ⇒ *real*
**where** $\textit{cmod} \equiv \textit{norm}$

**definition** *complex-sgn-def*: $\textit{sgn}\ x = x\ /_R\ \textit{cmod}\ x$

**definition** *dist-complex-def*: $\textit{dist}\ x\ y = \textit{cmod}\ (x - y)$

**definition** *uniformity-complex-def* [*code del*]:
$(\textit{uniformity} :: (\textit{complex} \times \textit{complex})\ \textit{filter}) = (\textit{INF}\ e{\in}\{0\ {<}..\}.\ \textit{principal}\ \{(x, y).\ \textit{dist}\ x\ y < e\})$

**definition** *open-complex-def* [*code del*]:
$open\ (U :: complex\ set) \longleftrightarrow (\forall x{\in}U.\ eventually\ (\lambda(x', y).\ x' = x \longrightarrow y \in U)\ uniformity)$

**instance**
⟨*proof*⟩

**end**

**declare** *uniformity-Abort*[**where** *'a = complex, code*]

**lemma** *Re-divide'*: $Re\ (x\ /\ y) = (Re\ x * Re\ y + Im\ x * Im\ y)\ /\ (norm\ y)^2$
⟨*proof*⟩

**lemma** *Im-divide'*: $Im\ (x\ /\ y) = (Im\ x * Re\ y - Re\ x * Im\ y)\ /\ (norm\ y)^2$
⟨*proof*⟩

**lemma** *norm-ii* [*simp*]: $norm\ \mathrm{i} = 1$
⟨*proof*⟩

**lemma** *cmod-unit-one*: $cmod\ (cos\ a + \mathrm{i} * sin\ a) = 1$
⟨*proof*⟩

**lemma** *cmod-complex-polar*: $cmod\ (r * (cos\ a + \mathrm{i} * sin\ a)) = |r|$
⟨*proof*⟩

**lemma** *complex-Re-le-cmod*: $Re\ x \leq cmod\ x$
⟨*proof*⟩

**lemma** *complex-mod-minus-le-complex-mod*: $-\ cmod\ x \leq cmod\ x$
⟨*proof*⟩

**lemma** *complex-mod-triangle-ineq2*: $cmod\ (b + a) - cmod\ b \leq cmod\ a$
⟨*proof*⟩

**lemma** *abs-Re-le-cmod*: $|Re\ x| \leq cmod\ x$
⟨*proof*⟩

**lemma** *abs-Im-le-cmod*: $|Im\ x| \leq cmod\ x$
⟨*proof*⟩

**lemma** *cmod-le*: $cmod\ z \leq |Re\ z| + |Im\ z|$
⟨*proof*⟩

**lemma** *cmod-eq-Re*: $Im\ z = 0 \Longrightarrow cmod\ z = |Re\ z|$
⟨*proof*⟩

**lemma** *cmod-eq-Im*: $Re\ z = 0 \Longrightarrow cmod\ z = |Im\ z|$

⟨*proof*⟩

**lemma** *cmod-power2*: $(cmod\ z)^2 = (Re\ z)^2 + (Im\ z)^2$
⟨*proof*⟩

**lemma** *cmod-plus-Re-le-0-iff*: $cmod\ z + Re\ z \leq 0 \longleftrightarrow Re\ z = -\ cmod\ z$
⟨*proof*⟩

**lemma** *cmod-Re-le-iff*: $Im\ x = Im\ y \Longrightarrow cmod\ x \leq cmod\ y \longleftrightarrow |Re\ x| \leq |Re\ y|$
⟨*proof*⟩

**lemma** *cmod-Im-le-iff*: $Re\ x = Re\ y \Longrightarrow cmod\ x \leq cmod\ y \longleftrightarrow |Im\ x| \leq |Im\ y|$
⟨*proof*⟩

**lemma** *Im-eq-0*: $|Re\ z| = cmod\ z \Longrightarrow Im\ z = 0$
⟨*proof*⟩

**lemma** *abs-sqrt-wlog*: $(\bigwedge x.\ x \geq 0 \Longrightarrow P\ x\ (x^2)) \Longrightarrow P\ |x|\ (x^2)$
**for** $x$::$'a$::*linordered-idom*
⟨*proof*⟩

**lemma** *complex-abs-le-norm*: $|Re\ z| + |Im\ z| \leq sqrt\ 2 * norm\ z$
⟨*proof*⟩

**lemma** *complex-unit-circle*: $z \neq 0 \Longrightarrow (Re\ z\ /\ cmod\ z)^2 + (Im\ z\ /\ cmod\ z)^2 = 1$
⟨*proof*⟩

Properties of complex signum.

**lemma** *sgn-eq*: $sgn\ z = z\ /\ complex\text{-}of\text{-}real\ (cmod\ z)$
⟨*proof*⟩

**lemma** *Re-sgn* [*simp*]: $Re(sgn\ z) = Re(z)/cmod\ z$
⟨*proof*⟩

**lemma** *Im-sgn* [*simp*]: $Im(sgn\ z) = Im(z)/cmod\ z$
⟨*proof*⟩

## 113.7 Absolute value

**instantiation** *complex* :: *field-abs-sgn*
**begin**

**definition** *abs-complex* :: *complex* ⇒ *complex*
**where** *abs-complex* = *of-real* ∘ *norm*

**instance**
⟨*proof*⟩
**end**

## 113.8 Completeness of the Complexes

**lemma** *bounded-linear-Re*: *bounded-linear Re*
⟨*proof*⟩

**lemma** *bounded-linear-Im*: *bounded-linear Im*
⟨*proof*⟩

**lemmas** *Cauchy-Re* = *bounded-linear.Cauchy* [*OF bounded-linear-Re*]
**lemmas** *Cauchy-Im* = *bounded-linear.Cauchy* [*OF bounded-linear-Im*]
**lemmas** *tendsto-Re* [*tendsto-intros*] = *bounded-linear.tendsto* [*OF bounded-linear-Re*]
**lemmas** *tendsto-Im* [*tendsto-intros*] = *bounded-linear.tendsto* [*OF bounded-linear-Im*]
**lemmas** *isCont-Re* [*simp*] = *bounded-linear.isCont* [*OF bounded-linear-Re*]
**lemmas** *isCont-Im* [*simp*] = *bounded-linear.isCont* [*OF bounded-linear-Im*]
**lemmas** *continuous-Re* [*simp*] = *bounded-linear.continuous* [*OF bounded-linear-Re*]
**lemmas** *continuous-Im* [*simp*] = *bounded-linear.continuous* [*OF bounded-linear-Im*]
**lemmas** *continuous-on-Re* [*continuous-intros*] = *bounded-linear.continuous-on*[*OF bounded-linear-Re*]
**lemmas** *continuous-on-Im* [*continuous-intros*] = *bounded-linear.continuous-on*[*OF bounded-linear-Im*]
**lemmas** *has-derivative-Re* [*derivative-intros*] = *bounded-linear.has-derivative*[*OF bounded-linear-Re*]
**lemmas** *has-derivative-Im* [*derivative-intros*] = *bounded-linear.has-derivative*[*OF bounded-linear-Im*]
**lemmas** *sums-Re* = *bounded-linear.sums* [*OF bounded-linear-Re*]
**lemmas** *sums-Im* = *bounded-linear.sums* [*OF bounded-linear-Im*]
**lemmas** *Re-suminf* = *bounded-linear.suminf*[*OF bounded-linear-Re*]
**lemmas** *Im-suminf* = *bounded-linear.suminf*[*OF bounded-linear-Im*]

**lemma** *continuous-on-Complex* [*continuous-intros*]:
*continuous-on A f* ⟹ *continuous-on A g* ⟹ *continuous-on A* (λ*x*. *Complex* (*f x*) (*g x*))
⟨*proof*⟩

**lemma** *tendsto-Complex* [*tendsto-intros*]:
(*f* ⟶ *a*) *F* ⟹ (*g* ⟶ *b*) *F* ⟹ ((λ*x*. *Complex* (*f x*) (*g x*)) ⟶ *Complex a b*) *F*
⟨*proof*⟩

**lemma** *tendsto-complex-iff*:
(*f* ⟶ *x*) *F* ⟷ (((λ*x*. *Re* (*f x*)) ⟶ *Re x*) *F* ∧ ((λ*x*. *Im* (*f x*)) ⟶ *Im x*) *F*)
⟨*proof*⟩

**lemma** *continuous-complex-iff*:
*continuous F f* ⟷ *continuous F* (λ*x*. *Re* (*f x*)) ∧ *continuous F* (λ*x*. *Im* (*f x*))
⟨*proof*⟩

**lemma** *continuous-on-of-real-o-iff* [*simp*]:
*continuous-on S* (λ*x*. *complex-of-real* (*g x*)) = *continuous-on S g*

⟨proof⟩

**lemma** *continuous-on-of-real-id* [*simp*]:
*continuous-on S* (*of-real* :: *real* ⇒ *'a*::*real-normed-algebra-1*)
⟨proof⟩

**lemma** *has-vector-derivative-complex-iff*: (*f has-vector-derivative x*) *F* ⟷
((λ*x*. *Re* (*f x*)) *has-field-derivative* (*Re x*)) *F* ∧
((λ*x*. *Im* (*f x*)) *has-field-derivative* (*Im x*)) *F*
⟨proof⟩

**lemma** *has-field-derivative-Re*[*derivative-intros*]:
(*f has-vector-derivative D*) *F* ⟹ ((λ*x*. *Re* (*f x*)) *has-field-derivative* (*Re D*)) *F*
⟨proof⟩

**lemma** *has-field-derivative-Im*[*derivative-intros*]:
(*f has-vector-derivative D*) *F* ⟹ ((λ*x*. *Im* (*f x*)) *has-field-derivative* (*Im D*)) *F*
⟨proof⟩

**instance** *complex* :: *banach*
⟨proof⟩

**declare** *DERIV-power*[**where** *'a*=*complex*, *unfolded of-nat-def*[*symmetric*], *derivative-intros*]

## 113.9 Complex Conjugation

**primcorec** *cnj* :: *complex* ⇒ *complex*
**where**
*Re* (*cnj z*) = *Re z*
| *Im* (*cnj z*) = − *Im z*

**lemma** *complex-cnj-cancel-iff* [*simp*]: *cnj x* = *cnj y* ⟷ *x* = *y*
⟨proof⟩

**lemma** *complex-cnj-cnj* [*simp*]: *cnj* (*cnj z*) = *z*
⟨proof⟩

**lemma** *in-image-cnj-iff*: *z* ∈ *cnj* ' *A* ⟷ *cnj z* ∈ *A*
⟨proof⟩

**lemma** *image-cnj-conv-vimage-cnj*: *cnj* ' *A* = *cnj* −' *A*
⟨proof⟩

**lemma** *complex-cnj-zero* [*simp*]: *cnj 0* = *0*
⟨proof⟩

**lemma** *complex-cnj-zero-iff* [*iff*]: *cnj z* = *0* ⟷ *z* = *0*
⟨proof⟩

**lemma** *complex-cnj-one-iff* [*simp*]: $cnj\ z = 1 \longleftrightarrow z = 1$
⟨*proof*⟩

**lemma** *complex-cnj-add* [*simp*]: $cnj\ (x + y) = cnj\ x + cnj\ y$
⟨*proof*⟩

**lemma** *cnj-sum* [*simp*]: $cnj\ (sum\ f\ s) = (\sum x{\in}s.\ cnj\ (f\ x))$
⟨*proof*⟩

**lemma** *complex-cnj-diff* [*simp*]: $cnj\ (x - y) = cnj\ x - cnj\ y$
⟨*proof*⟩

**lemma** *complex-cnj-minus* [*simp*]: $cnj\ (-\ x) = -\ cnj\ x$
⟨*proof*⟩

**lemma** *complex-cnj-one* [*simp*]: $cnj\ 1 = 1$
⟨*proof*⟩

**lemma** *complex-cnj-mult* [*simp*]: $cnj\ (x * y) = cnj\ x * cnj\ y$
⟨*proof*⟩

**lemma** *cnj-prod* [*simp*]: $cnj\ (prod\ f\ s) = (\prod x{\in}s.\ cnj\ (f\ x))$
⟨*proof*⟩

**lemma** *complex-cnj-inverse* [*simp*]: $cnj\ (inverse\ x) = inverse\ (cnj\ x)$
⟨*proof*⟩

**lemma** *complex-cnj-divide* [*simp*]: $cnj\ (x\ /\ y) = cnj\ x\ /\ cnj\ y$
⟨*proof*⟩

**lemma** *complex-cnj-power* [*simp*]: $cnj\ (x\ \widehat{}\ n) = cnj\ x\ \widehat{}\ n$
⟨*proof*⟩

**lemma** *complex-cnj-of-nat* [*simp*]: $cnj\ (of\text{-}nat\ n) = of\text{-}nat\ n$
⟨*proof*⟩

**lemma** *complex-cnj-of-int* [*simp*]: $cnj\ (of\text{-}int\ z) = of\text{-}int\ z$
⟨*proof*⟩

**lemma** *complex-cnj-numeral* [*simp*]: $cnj\ (numeral\ w) = numeral\ w$
⟨*proof*⟩

**lemma** *complex-cnj-neg-numeral* [*simp*]: $cnj\ (-\ numeral\ w) = -\ numeral\ w$
⟨*proof*⟩

**lemma** *complex-cnj-scaleR* [*simp*]: $cnj\ (scaleR\ r\ x) = scaleR\ r\ (cnj\ x)$
⟨*proof*⟩

**lemma** *complex-mod-cnj* [*simp*]: $cmod\ (cnj\ z) = cmod\ z$
⟨*proof*⟩

**lemma** *complex-cnj-complex-of-real* [*simp*]: $cnj\ (of\text{-}real\ x) = of\text{-}real\ x$
⟨*proof*⟩

**lemma** *complex-cnj-i* [*simp*]: $cnj\ \mathrm{i} = -\ \mathrm{i}$
⟨*proof*⟩

**lemma** *complex-add-cnj*: $z + cnj\ z = complex\text{-}of\text{-}real\ (2 * Re\ z)$
⟨*proof*⟩

**lemma** *complex-diff-cnj*: $z - cnj\ z = complex\text{-}of\text{-}real\ (2 * Im\ z) * \mathrm{i}$
⟨*proof*⟩

**lemma** *Ints-cnj* [*intro*]: $x \in \mathbb{Z} \Longrightarrow cnj\ x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *cnj-in-Ints-iff* [*simp*]: $cnj\ x \in \mathbb{Z} \longleftrightarrow x \in \mathbb{Z}$
⟨*proof*⟩

**lemma** *complex-mult-cnj*: $z * cnj\ z = complex\text{-}of\text{-}real\ ((Re\ z)^2 + (Im\ z)^2)$
⟨*proof*⟩

**lemma** *cnj-add-mult-eq-Re*: $z * cnj\ w + cnj\ z * w = 2 * Re\ (z * cnj\ w)$
⟨*proof*⟩

**lemma** *complex-mod-mult-cnj*: $cmod\ (z * cnj\ z) = (cmod\ z)^2$
⟨*proof*⟩

**lemma** *complex-mod-sqrt-Re-mult-cnj*: $cmod\ z = sqrt\ (Re\ (z * cnj\ z))$
⟨*proof*⟩

**lemma** *complex-In-mult-cnj-zero* [*simp*]: $Im\ (z * cnj\ z) = 0$
⟨*proof*⟩

**lemma** *complex-cnj-fact* [*simp*]: $cnj\ (fact\ n) = fact\ n$
⟨*proof*⟩

**lemma** *complex-cnj-pochhammer* [*simp*]: $cnj\ (pochhammer\ z\ n) = pochhammer\ (cnj\ z)\ n$
⟨*proof*⟩

**lemma** *bounded-linear-cnj*: *bounded-linear cnj*
⟨*proof*⟩

**lemma** *linear-cnj*: *linear cnj*
⟨*proof*⟩

**lemmas** *tendsto-cnj* [*tendsto-intros*] = *bounded-linear.tendsto* [*OF bounded-linear-cnj*]
**and** *isCont-cnj* [*simp*] = *bounded-linear.isCont* [*OF bounded-linear-cnj*]
**and** *continuous-cnj* [*simp*, *continuous-intros*] = *bounded-linear.continuous* [*OF bounded-linear-cnj*]
**and** *continuous-on-cnj* [*simp*, *continuous-intros*] = *bounded-linear.continuous-on* [*OF bounded-linear-cnj*]
**and** *has-derivative-cnj* [*simp*, *derivative-intros*] = *bounded-linear.has-derivative* [*OF bounded-linear-cnj*]

**lemma** *lim-cnj*: $((\lambda x.\ cnj(f\ x)) \longrightarrow cnj\ l)\ F \longleftrightarrow (f \longrightarrow l)\ F$
⟨*proof*⟩

**lemma** *sums-cnj*: $((\lambda x.\ cnj(f\ x))\ sums\ cnj\ l) \longleftrightarrow (f\ sums\ l)$
⟨*proof*⟩

**lemma** *differentiable-cnj-iff*:
$(\lambda z.\ cnj\ (f\ z))$ *differentiable at* $x$ *within* $A \longleftrightarrow f$ *differentiable at* $x$ *within* $A$
⟨*proof*⟩

**lemma** *has-vector-derivative-cnj* [*derivative-intros*]:
**assumes** $(f$ *has-vector-derivative* $f')$ $(at\ z\ within\ A)$
**shows** $((\lambda z.\ cnj\ (f\ z))$ *has-vector-derivative* $cnj\ f')$ $(at\ z\ within\ A)$
⟨*proof*⟩

**lemma** *has-field-derivative-cnj-cnj*:
**assumes** $(f$ *has-field-derivative* $F)$ $(at\ (cnj\ z))$
**shows** $((cnj \circ f \circ cnj)$ *has-field-derivative* $cnj\ F)$ $(at\ z)$
⟨*proof*⟩

## 113.10 Basic Lemmas

**lemma** *complex-of-real-code*[*code-unfold*]: $of\text{-}real = (\lambda x.\ Complex\ x\ 0)$
⟨*proof*⟩

**lemma** *complex-eq-0*: $z=0 \longleftrightarrow (Re\ z)^2 + (Im\ z)^2 = 0$
⟨*proof*⟩

**lemma** *complex-neq-0*: $z\neq 0 \longleftrightarrow (Re\ z)^2 + (Im\ z)^2 > 0$
⟨*proof*⟩

**lemma** *complex-norm-square*: $of\text{-}real\ ((norm\ z)^2) = z * cnj\ z$
⟨*proof*⟩

**lemma** *complex-div-cnj*: $a\ /\ b = (a * cnj\ b)\ /\ (norm\ b)^2$
⟨*proof*⟩

**lemma** *Re-complex-div-eq-0*: $Re\ (a\ /\ b) = 0 \longleftrightarrow Re\ (a * cnj\ b) = 0$
⟨*proof*⟩

**lemma** *Im-complex-div-eq-0*: $Im\ (a\ /\ b) = 0 \longleftrightarrow Im\ (a * cnj\ b) = 0$
⟨*proof*⟩

**lemma** *complex-div-gt-0*: $(Re\ (a\ /\ b) > 0 \longleftrightarrow Re\ (a * cnj\ b) > 0) \wedge (Im\ (a\ /\ b) > 0 \longleftrightarrow Im\ (a * cnj\ b) > 0)$
⟨*proof*⟩

**lemma** *Re-complex-div-gt-0*: $Re\ (a\ /\ b) > 0 \longleftrightarrow Re\ (a * cnj\ b) > 0$
**and** *Im-complex-div-gt-0*: $Im\ (a\ /\ b) > 0 \longleftrightarrow Im\ (a * cnj\ b) > 0$
⟨*proof*⟩

**lemma** *Re-complex-div-ge-0*: $Re\ (a\ /\ b) \geq 0 \longleftrightarrow Re\ (a * cnj\ b) \geq 0$
⟨*proof*⟩

**lemma** *Im-complex-div-ge-0*: $Im\ (a\ /\ b) \geq 0 \longleftrightarrow Im\ (a * cnj\ b) \geq 0$
⟨*proof*⟩

**lemma** *Re-complex-div-lt-0*: $Re\ (a\ /\ b) < 0 \longleftrightarrow Re\ (a * cnj\ b) < 0$
⟨*proof*⟩

**lemma** *Im-complex-div-lt-0*: $Im\ (a\ /\ b) < 0 \longleftrightarrow Im\ (a * cnj\ b) < 0$
⟨*proof*⟩

**lemma** *Re-complex-div-le-0*: $Re\ (a\ /\ b) \leq 0 \longleftrightarrow Re\ (a * cnj\ b) \leq 0$
⟨*proof*⟩

**lemma** *Im-complex-div-le-0*: $Im\ (a\ /\ b) \leq 0 \longleftrightarrow Im\ (a * cnj\ b) \leq 0$
⟨*proof*⟩

**lemma** *Re-divide-of-real* [*simp*]: $Re\ (z\ /\ of\text{-}real\ r) = Re\ z\ /\ r$
⟨*proof*⟩

**lemma** *Im-divide-of-real* [*simp*]: $Im\ (z\ /\ of\text{-}real\ r) = Im\ z\ /\ r$
⟨*proof*⟩

**lemma** *Re-divide-Reals* [*simp*]: $r \in \mathbb{R} \Longrightarrow Re\ (z\ /\ r) = Re\ z\ /\ Re\ r$
⟨*proof*⟩

**lemma** *Im-divide-Reals* [*simp*]: $r \in \mathbb{R} \Longrightarrow Im\ (z\ /\ r) = Im\ z\ /\ Re\ r$
⟨*proof*⟩

**lemma** *Re-sum*[*simp*]: $Re\ (sum\ f\ s) = (\sum x{\in}s.\ Re\ (f\ x))$
⟨*proof*⟩

**lemma** *Im-sum*[*simp*]: $Im\ (sum\ f\ s) = (\sum x{\in}s.\ Im(f\ x))$
⟨*proof*⟩

**lemma** *Rats-complex-of-real-iff* [*iff*]: $complex\text{-}of\text{-}real\ x \in \mathbb{Q} \longleftrightarrow x \in \mathbb{Q}$
⟨*proof*⟩

**lemma** *sum-Re-le-cmod*: $(\sum i\in I.\ Re\ (z\ i)) \leq cmod\ (\sum i\in I.\ z\ i)$
⟨*proof*⟩

**lemma** *sum-Im-le-cmod*: $(\sum i\in I.\ Im\ (z\ i)) \leq cmod\ (\sum i\in I.\ z\ i)$
⟨*proof*⟩

**lemma** *sums-complex-iff*: $f\ sums\ x \longleftrightarrow ((\lambda x.\ Re\ (f\ x))\ sums\ Re\ x) \wedge ((\lambda x.\ Im\ (f\ x))\ sums\ Im\ x)$
⟨*proof*⟩

**lemma** *summable-complex-iff*: $summable\ f \longleftrightarrow summable\ (\lambda x.\ Re\ (f\ x)) \wedge summable\ (\lambda x.\ Im\ (f\ x))$
⟨*proof*⟩

**lemma** *summable-complex-of-real* [*simp*]: $summable\ (\lambda n.\ complex\text{-}of\text{-}real\ (f\ n)) \longleftrightarrow summable\ f$
⟨*proof*⟩

**lemma** *summable-Re*: $summable\ f \Longrightarrow summable\ (\lambda x.\ Re\ (f\ x))$
⟨*proof*⟩

**lemma** *summable-Im*: $summable\ f \Longrightarrow summable\ (\lambda x.\ Im\ (f\ x))$
⟨*proof*⟩

**lemma** *complex-is-Nat-iff*: $z \in \mathbb{N} \longleftrightarrow Im\ z = 0 \wedge (\exists\, i.\ Re\ z = of\text{-}nat\ i)$
⟨*proof*⟩

**lemma** *complex-is-Int-iff*: $z \in \mathbb{Z} \longleftrightarrow Im\ z = 0 \wedge (\exists\, i.\ Re\ z = of\text{-}int\ i)$
⟨*proof*⟩

**lemma** *complex-is-Real-iff*: $z \in \mathbb{R} \longleftrightarrow Im\ z = 0$
⟨*proof*⟩

**lemma** *sgn-complex-iff*: $sgn\ x = sgn\ (Re\ x) \longleftrightarrow x \in \mathbb{R}$
⟨*proof*⟩

**lemma** *Reals-cnj-iff*: $z \in \mathbb{R} \longleftrightarrow cnj\ z = z$
⟨*proof*⟩

**lemma** *in-Reals-norm*: $z \in \mathbb{R} \Longrightarrow norm\ z = |Re\ z|$
⟨*proof*⟩

**lemma** *Re-Reals-divide*: $r \in \mathbb{R} \Longrightarrow Re\ (r\ /\ z) = Re\ r * Re\ z\ /\ (norm\ z)^2$
⟨*proof*⟩

**lemma** *Im-Reals-divide*: $r \in \mathbb{R} \Longrightarrow Im\ (r\ /\ z) = -Re\ r * Im\ z\ /\ (norm\ z)^2$
⟨*proof*⟩

**lemma** *series-comparison-complex*:
  **fixes** $f$:: *nat* $\Rightarrow$ $'a$::*banach*
  **assumes** *sg*: *summable g*
    **and** $\bigwedge n.\ g\ n \in \mathbb{R}$ $\bigwedge n.\ Re\ (g\ n) \geq 0$
    **and** *fg*: $\bigwedge n.\ n \geq N \Longrightarrow norm(f\ n) \leq norm(g\ n)$
  **shows** *summable f*
⟨*proof*⟩

## 113.11 Polar Form for Complex Numbers

**lemma** *complex-unimodular-polar*:
  **assumes** *norm z = 1*
  **obtains** *t* **where** $0 \leq t$ $t < 2 * pi$ $z = Complex\ (cos\ t)\ (sin\ t)$
  ⟨*proof*⟩

### 113.11.1 $\cos\theta + i \sin\theta$

**primcorec** *cis* :: *real* $\Rightarrow$ *complex*
  **where**
    $Re\ (cis\ a) = cos\ a$
  | $Im\ (cis\ a) = sin\ a$

**lemma** *cis-zero* [*simp*]: $cis\ 0 = 1$
  ⟨*proof*⟩

**lemma** *norm-cis* [*simp*]: $norm\ (cis\ a) = 1$
  ⟨*proof*⟩

**lemma** *sgn-cis* [*simp*]: $sgn\ (cis\ a) = cis\ a$
  ⟨*proof*⟩

**lemma** *cis-2pi* [*simp*]: $cis\ (2 * pi) = 1$
  ⟨*proof*⟩

**lemma** *cis-neq-zero* [*simp*]: $cis\ a \neq 0$
  ⟨*proof*⟩

**lemma** *cis-cnj*: $cnj\ (cis\ t) = cis\ (-t)$
  ⟨*proof*⟩

**lemma** *cis-mult*: $cis\ a * cis\ b = cis\ (a + b)$
  ⟨*proof*⟩

**lemma** *DeMoivre*: $(cis\ a)\ \hat{}\ n = cis\ (real\ n * a)$
  ⟨*proof*⟩

**lemma** *cis-inverse* [*simp*]: $inverse\ (cis\ a) = cis\ (-\ a)$
  ⟨*proof*⟩

**lemma** *cis-divide*: $cis\ a\ /\ cis\ b = cis\ (a - b)$

⟨*proof*⟩

**lemma** *cis-power-int*: *cis x powi n = cis (of-int n ∗ x)*
⟨*proof*⟩

**lemma** *complex-cnj-power-int* [*simp*]: *cnj (x powi n) = cnj x powi n*
⟨*proof*⟩

**lemma** *divide-conv-cnj*: *norm z = 1* $\Longrightarrow$ *x / z = x ∗ cnj z*
⟨*proof*⟩

**lemma** *i-not-in-Reals* [*simp*, *intro*]: i $\notin \mathbb{R}$
⟨*proof*⟩

**lemma** *cos-n-Re-cis-pow-n*: *cos (real n ∗ a) = Re (cis a ^ n)*
⟨*proof*⟩

**lemma** *sin-n-Im-cis-pow-n*: *sin (real n ∗ a) = Im (cis a ^ n)*
⟨*proof*⟩

**lemma** *cis-pi* [*simp*]: *cis pi = −1*
⟨*proof*⟩

**lemma** *cis-pi-half*[*simp*]: *cis (pi / 2) =* i
⟨*proof*⟩

**lemma** *cis-minus-pi-half*[*simp*]: *cis (−(pi / 2)) =* −i
⟨*proof*⟩

**lemma** *cis-multiple-2pi*[*simp*]: *n* $\in \mathbb{Z} \Longrightarrow$ *cis (2 ∗ pi ∗ n) = 1*
⟨*proof*⟩

**lemma** *minus-cis*: *−cis x = cis (x + pi)*
⟨*proof*⟩

**lemma** *minus-cis'*: *−cis x = cis (x − pi)*
⟨*proof*⟩

### 113.11.2 $r(\cos\theta + i\sin\theta)$

**definition** *rcis* :: *real* ⇒ *real* ⇒ *complex*
**where** *rcis r a = complex-of-real r ∗ cis a*

**lemma** *Re-rcis* [*simp*]: *Re(rcis r a) = r ∗ cos a*
⟨*proof*⟩

**lemma** *Im-rcis* [*simp*]: *Im(rcis r a) = r ∗ sin a*
⟨*proof*⟩

**lemma** *rcis-Ex*: $\exists\, r\ a.\ z = rcis\ r\ a$
⟨*proof*⟩

**lemma** *complex-mod-rcis* [*simp*]: $cmod\ (rcis\ r\ a) = |r|$
⟨*proof*⟩

**lemma** *cis-rcis-eq*: $cis\ a = rcis\ 1\ a$
⟨*proof*⟩

**lemma** *rcis-mult*: $rcis\ r1\ a * rcis\ r2\ b = rcis\ (r1 * r2)\ (a + b)$
⟨*proof*⟩

**lemma** *rcis-zero-mod* [*simp*]: $rcis\ 0\ a = 0$
⟨*proof*⟩

**lemma** *rcis-zero-arg* [*simp*]: $rcis\ r\ 0 = complex\text{-}of\text{-}real\ r$
⟨*proof*⟩

**lemma** *rcis-eq-zero-iff* [*simp*]: $rcis\ r\ a = 0 \longleftrightarrow r = 0$
⟨*proof*⟩

**lemma** *DeMoivre2*: $(rcis\ r\ a)\ \hat{}\ n = rcis\ (r\ \hat{}\ n)\ (real\ n * a)$
⟨*proof*⟩

**lemma** *rcis-inverse*: $inverse(rcis\ r\ a) = rcis\ (1\ /\ r)\ (-\ a)$
⟨*proof*⟩

**lemma** *rcis-divide*: $rcis\ r1\ a\ /\ rcis\ r2\ b = rcis\ (r1\ /\ r2)\ (a - b)$
⟨*proof*⟩

### 113.11.3 Complex exponential

**lemma** *exp-Reals-eq*:
**assumes** $z \in \mathbb{R}$
**shows** $exp\ z = of\text{-}real\ (exp\ (Re\ z))$
⟨*proof*⟩

**lemma** *cis-conv-exp*: $cis\ b = exp\ (\mathrm{i} * b)$
⟨*proof*⟩

**lemma** *exp-eq-polar*: $exp\ z = exp\ (Re\ z) * cis\ (Im\ z)$
⟨*proof*⟩

**lemma** *Re-exp*: $Re\ (exp\ z) = exp\ (Re\ z) * cos\ (Im\ z)$
⟨*proof*⟩

**lemma** *Im-exp*: $Im\ (exp\ z) = exp\ (Re\ z) * sin\ (Im\ z)$
⟨*proof*⟩

**lemma** *norm-cos-sin* [*simp*]: *norm* (*Complex* (*cos t*) (*sin t*)) = *1*
⟨*proof*⟩

**lemma** *norm-exp-eq-Re* [*simp*]: *norm* (*exp z*) = *exp* (*Re z*)
⟨*proof*⟩

**lemma** *complex-exp-exists*: $\exists$ *a r*. *z* = *complex-of-real r* $*$ *exp a*
⟨*proof*⟩

**lemma** *exp-pi-i* [*simp*]: *exp* (*of-real pi* $*$ i) = $-$*1*
⟨*proof*⟩

**lemma** *exp-pi-i'* [*simp*]: *exp* (i $*$ *of-real pi*) = $-$*1*
⟨*proof*⟩

**lemma** *exp-two-pi-i* [*simp*]: *exp* (*2* $*$ *of-real pi* $*$ i) = *1*
⟨*proof*⟩

**lemma** *exp-two-pi-i'* [*simp*]: *exp* (i $*$ (*of-real pi* $*$ *2*)) = *1*
⟨*proof*⟩

**lemma** *continuous-on-cis* [*continuous-intros*]:
*continuous-on A f* $\Longrightarrow$ *continuous-on A* ($\lambda$*x*. *cis* (*f x*))
⟨*proof*⟩

**lemma** *tendsto-exp-0-Re-at-bot*: (*exp* $\longrightarrow$ *0*) (*filtercomap Re at-bot*)
⟨*proof*⟩

**lemma** *filterlim-exp-at-infinity-Re-at-top*: *filterlim exp at-infinity* (*filtercomap Re at-top*)
⟨*proof*⟩

**lemma** *tendsto-cis* [*tendsto-intros*]:
**assumes** (*f* $\longrightarrow$ *x*) *F*
**shows** (($\lambda$*u*. *cis* (*f u*)) $\longrightarrow$ *cis x*) *F*
⟨*proof*⟩

**lemma** *tendsto-rcis* [*tendsto-intros*]:
**assumes** (*f* $\longrightarrow$ *r*) *F* (*g* $\longrightarrow$ *x*) *F*
**shows** (($\lambda$*u*. *rcis* (*f u*) (*g u*)) $\longrightarrow$ *rcis r x*) *F*
⟨*proof*⟩

**lemma** *continuous-on-rcis* [*continuous-intros*]:
*continuous-on A f* $\Longrightarrow$ *continuous-on A g* $\Longrightarrow$ *continuous-on A* ($\lambda$*x*. *rcis* (*f x*) (*g x*))
⟨*proof*⟩

**lemma** *has-derivative-cis* [*derivative-intros*]:
**assumes** (*f has-derivative d*) (*at x within A*)

**shows** $((\lambda x.\ cis\ (f\ x))\ has\text{-}derivative\ (\lambda t.\ d\ t *_R (i * cis\ (f\ x))))\ (at\ x\ within\ A)$
⟨*proof*⟩

### 113.11.4 Complex argument

**definition** $Arg :: complex \Rightarrow real$
**where** $Arg\ z = (if\ z = 0\ then\ 0\ else\ (SOME\ a.\ sgn\ z = cis\ a \wedge -\ pi < a \wedge a \leq pi))$

**lemma** *Arg-zero*: $Arg\ 0 = 0$
⟨*proof*⟩

**lemma** *cis-Arg-unique*:
**assumes** $sgn\ z = cis\ x$ **and** $-pi < x$ **and** $x \leq pi$
**shows** $Arg\ z = x$
⟨*proof*⟩

**lemma** *Arg-correct*:
**assumes** $z \neq 0$
**shows** $sgn\ z = cis\ (Arg\ z) \wedge -pi < Arg\ z \wedge Arg\ z \leq pi$
⟨*proof*⟩

**lemma** *Arg-bounded*: $-\ pi < Arg\ z \wedge Arg\ z \leq pi$
⟨*proof*⟩

**lemma** *cis-Arg*: $z \neq 0 \Longrightarrow cis\ (Arg\ z) = sgn\ z$
⟨*proof*⟩

**lemma** *rcis-cmod-Arg*: $rcis\ (cmod\ z)\ (Arg\ z) = z$
⟨*proof*⟩

**lemma** *rcis-cnj*:
**shows** $cnj\ a = rcis\ (cmod\ a)\ (-\ Arg\ a)$
⟨*proof*⟩

**lemma** *cos-Arg-i-mult-zero* [*simp*]: $y \neq 0 \Longrightarrow Re\ y = 0 \Longrightarrow cos\ (Arg\ y) = 0$
⟨*proof*⟩

**lemma** *Arg-ii* [*simp*]: $Arg\ i = pi/2$
⟨*proof*⟩

**lemma** *Arg-minus-ii* [*simp*]: $Arg\ (-i) = -pi/2$
⟨*proof*⟩

**lemma** *cos-Arg*: $z \neq 0 \Longrightarrow cos\ (Arg\ z) = Re\ z\ /\ norm\ z$
⟨*proof*⟩

**lemma** *sin-Arg*: $z \neq 0 \Longrightarrow sin\ (Arg\ z) = Im\ z\ /\ norm\ z$
⟨*proof*⟩

## 113.12 Complex n-th roots

**lemma** *bij-betw-roots-unity*:
**assumes** $n > 0$
**shows** $\textit{bij-betw}\ (\lambda k.\ \textit{cis}\ (2 * \textit{pi} * \textit{real}\ k\ /\ \textit{real}\ n))\ \{..<n\}\ \{z.\ z\ \hat{}\ n = 1\}$
(**is** *bij-betw ?f - -*)
⟨*proof*⟩

**lemma** *card-roots-unity-eq*:
**assumes** $n > 0$
**shows** $\textit{card}\ \{z{::}\textit{complex}.\ z\ \hat{}\ n = 1\} = n$
⟨*proof*⟩

**lemma** *bij-betw-nth-root-unity*:
**fixes** $c :: \textit{complex}$ **and** $n :: \textit{nat}$
**assumes** $c$: $c \neq 0$ **and** $n$: $n > 0$
**defines** $c' \equiv \textit{root}\ n\ (\textit{norm}\ c) * \textit{cis}\ (\textit{Arg}\ c\ /\ n)$
**shows** $\textit{bij-betw}\ (\lambda z.\ c' * z)\ \{z.\ z\ \hat{}\ n = 1\}\ \{z.\ z\ \hat{}\ n = c\}$
⟨*proof*⟩

**lemma** *finite-nth-roots* [*intro*]:
**assumes** $n > 0$
**shows** $\textit{finite}\ \{z{::}\textit{complex}.\ z\ \hat{}\ n = c\}$
⟨*proof*⟩

**lemma** *card-nth-roots*:
**assumes** $c \neq 0\ n > 0$
**shows** $\textit{card}\ \{z{::}\textit{complex}.\ z\ \hat{}\ n = c\} = n$
⟨*proof*⟩

**lemma** *sum-roots-unity*:
**assumes** $n > 1$
**shows** $\sum\{z{::}\textit{complex}.\ z\ \hat{}\ n = 1\} = 0$
⟨*proof*⟩

**lemma** *sum-nth-roots*:
**assumes** $n > 1$
**shows** $\sum\{z{::}\textit{complex}.\ z\ \hat{}\ n = c\} = 0$
⟨*proof*⟩

## 113.13 Square root of complex numbers

**primcorec** $\textit{csqrt} :: \textit{complex} \Rightarrow \textit{complex}$
**where**
$\textit{Re}\ (\textit{csqrt}\ z) = \textit{sqrt}\ ((\textit{cmod}\ z + \textit{Re}\ z)\ /\ 2)$
$|\ \textit{Im}\ (\textit{csqrt}\ z) = (\textit{if}\ \textit{Im}\ z = 0\ \textit{then}\ 1\ \textit{else}\ \textit{sgn}\ (\textit{Im}\ z)) * \textit{sqrt}\ ((\textit{cmod}\ z - \textit{Re}\ z)\ /\ 2)$

**lemma** *csqrt-of-real-nonneg* [*simp*]: $\textit{Im}\ x = 0 \Longrightarrow \textit{Re}\ x \geq 0 \Longrightarrow \textit{csqrt}\ x = \textit{sqrt}\ (\textit{Re}\ x)$

⟨proof⟩

**lemma** *csqrt-of-real-nonpos* [*simp*]: $Im\ x = 0 \Longrightarrow Re\ x \leq 0 \Longrightarrow csqrt\ x = \mathrm{i} * sqrt\ |Re\ x|$
⟨proof⟩

**lemma** *of-real-sqrt*: $x \geq 0 \Longrightarrow of\text{-}real\ (sqrt\ x) = csqrt\ (of\text{-}real\ x)$
⟨proof⟩

**lemma** *csqrt-0* [*simp*]: $csqrt\ 0 = 0$
⟨proof⟩

**lemma** *csqrt-1* [*simp*]: $csqrt\ 1 = 1$
⟨proof⟩

**lemma** *csqrt-ii* [*simp*]: $csqrt\ \mathrm{i} = (1 + \mathrm{i})\ /\ sqrt\ 2$
⟨proof⟩

**lemma** *power2-csqrt*[*simp*,*algebra*]: $(csqrt\ z)^2 = z$
⟨proof⟩

**lemma** *csqrt-power-even*:
**assumes** *even n*
**shows** $csqrt\ z\ \hat{}\ n = z\ \hat{}\ (n\ div\ 2)$
⟨proof⟩

**lemma** *norm-csqrt* [*simp*]: $norm\ (csqrt\ z) = sqrt\ (norm\ z)$
⟨proof⟩

**lemma** *csqrt-eq-0* [*simp*]: $csqrt\ z = 0 \longleftrightarrow z = 0$
⟨proof⟩

**lemma** *csqrt-eq-1* [*simp*]: $csqrt\ z = 1 \longleftrightarrow z = 1$
⟨proof⟩

**lemma** *csqrt-principal*: $0 < Re\ (csqrt\ z) \vee Re\ (csqrt\ z) = 0 \wedge 0 \leq Im\ (csqrt\ z)$
⟨proof⟩

**lemma** *Re-csqrt*: $0 \leq Re\ (csqrt\ z)$
⟨proof⟩

**lemma** *csqrt-square*:
**assumes** $0 < Re\ b \vee (Re\ b = 0 \wedge 0 \leq Im\ b)$
**shows** $csqrt\ (b\hat{}2) = b$
⟨proof⟩

**lemma** *csqrt-unique*: $w^2 = z \Longrightarrow 0 < Re\ w \vee Re\ w = 0 \wedge 0 \leq Im\ w \Longrightarrow csqrt\ z = w$
⟨proof⟩

**lemma** *csqrt-minus* [*simp*]:
**assumes** $Im\ x < 0 \lor (Im\ x = 0 \land 0 \leq Re\ x)$
**shows** $csqrt\ (-x) = \mathrm{i} * csqrt\ x$
$\langle proof \rangle$

**lemma** *csqrt-neq-neg-real*:
**assumes** $Im\ x = 0\ Re\ x < 0$
**shows** $csqrt\ z \neq x$
$\langle proof \rangle$

**lemma** *csqrt-of-real*: $x \geq 0 \Longrightarrow csqrt\ (of\text{-}real\ x) = of\text{-}real\ (sqrt\ x)$
$\langle proof \rangle$

**lemma** *csqrt-of-real'*: $csqrt\ (of\text{-}real\ x) = of\text{-}real\ (sqrt\ |x|) * (if\ x \geq 0\ then\ 1\ else\ \mathrm{i})$
$\langle proof \rangle$

**lemma** *csqrt-minus-Reals*:
**assumes** $x \in \mathbb{R}$
**shows** $csqrt\ (-\ x) = sgn\ (Re\ x) * \mathrm{i} * csqrt\ x$
$\langle proof \rangle$

**lemmas** $cmod\text{-}def = norm\text{-}complex\text{-}def$

**lemma** *Complex-simps*:
**shows** *Complex-eq-0*: $Complex\ a\ b = 0 \longleftrightarrow a = 0 \land b = 0$
**and** *complex-add*: $Complex\ a\ b + Complex\ c\ d = Complex\ (a + c)\ (b + d)$
**and** *complex-minus*: $-\ (Complex\ a\ b) = Complex\ (-\ a)\ (-\ b)$
**and** *complex-diff*: $Complex\ a\ b - Complex\ c\ d = Complex\ (a - c)\ (b - d)$
**and** *Complex-eq-1*: $Complex\ a\ b = 1 \longleftrightarrow a = 1 \land b = 0$
**and** *Complex-eq-neg-1*: $Complex\ a\ b = -\ 1 \longleftrightarrow a = -\ 1 \land b = 0$
**and** *complex-mult*: $Complex\ a\ b * Complex\ c\ d = Complex\ (a * c - b * d)\ (a * d + b * c)$
**and** *complex-inverse*: $inverse\ (Complex\ a\ b) = Complex\ (a\ /\ (a^2 + b^2))\ (-\ b\ /\ (a^2 + b^2))$
**and** *Complex-eq-numeral*: $Complex\ a\ b = numeral\ w \longleftrightarrow a = numeral\ w \land b = 0$
**and** *Complex-eq-neg-numeral*: $Complex\ a\ b = -\ numeral\ w \longleftrightarrow a = -\ numeral\ w \land b = 0$
**and** *complex-scaleR*: $scaleR\ r\ (Complex\ a\ b) = Complex\ (r * a)\ (r * b)$
**and** *Complex-eq-i*: $Complex\ x\ y = \mathrm{i} \longleftrightarrow x = 0 \land y = 1$
**and** *i-mult-Complex*: $\mathrm{i} * Complex\ a\ b = Complex\ (-\ b)\ a$
**and** *Complex-mult-i*: $Complex\ a\ b * \mathrm{i} = Complex\ (-\ b)\ a$
**and** *i-complex-of-real*: $\mathrm{i} * complex\text{-}of\text{-}real\ r = Complex\ 0\ r$
**and** *complex-of-real-i*: $complex\text{-}of\text{-}real\ r * \mathrm{i} = Complex\ 0\ r$
**and** *Complex-add-complex-of-real*: $Complex\ x\ y + complex\text{-}of\text{-}real\ r = Complex\ (x+r)\ y$
**and** *complex-of-real-add-Complex*: $complex\text{-}of\text{-}real\ r + Complex\ x\ y = Complex\ (r+x)\ y$

**and** *Complex-mult-complex-of-real*: *Complex x y* $*$ *complex-of-real r* = *Complex* $(x{*}r)$ $(y{*}r)$
**and** *complex-of-real-mult-Complex*: *complex-of-real r* $*$ *Complex x y* = *Complex* $(r{*}x)$ $(r{*}y)$
**and** *complex-eq-cancel-iff2*: (*Complex x y* = *complex-of-real xa*) = ($x = xa \wedge y = 0$)
**and** *complex-cnj*: *cnj* (*Complex a b*) = *Complex a* (− *b*)
**and** *Complex-sum'*: *sum* ($\lambda x$. *Complex* (*f x*) *0*) *s* = *Complex* (*sum f s*) *0*
**and** *Complex-sum*: *Complex* (*sum f s*) *0* = *sum* ($\lambda x$. *Complex* (*f x*) *0*) *s*
**and** *complex-of-real-def*: *complex-of-real r* = *Complex r 0*
**and** *complex-norm*: *cmod* (*Complex x y*) = *sqrt* ($x^2 + y^2$)
⟨*proof*⟩

**lemma** *Complex-in-Reals*: *Complex x 0* $\in \mathbb{R}$
⟨*proof*⟩

**lemma** *Complex-divide-complex-of-real*: *Complex x y* / *of-real r* = *Complex* $(x/r)$ $(y/r)$
⟨*proof*⟩

**lemma** *cmod-neg-real*: *cmod* (*Complex* $(-x)$ *y*) = *cmod* (*Complex x y*)
⟨*proof*⟩

Express a complex number as a linear combination of two others, not collinear with the origin

**lemma** *complex-axes*:
**assumes** *Im* $(y/x) \neq 0$
**obtains** *a b* **where** $z$ = *of-real a* $*$ *x* + *of-real b* $*$ *y*
⟨*proof*⟩

**end**

# 114 MacLaurin and Taylor Series

**theory** *MacLaurin*
**imports** *Transcendental*
**begin**

## 114.1 Maclaurin's Theorem with Lagrange Form of Remainder

This is a very long, messy proof even now that it's been broken down into lemmas.

**lemma** *Maclaurin-lemma*:
$0 < h \Longrightarrow$
$\exists B$::*real*. $f\ h = (\sum m{<}n.\ (j\ m\ /\ (\mathit{fact}\ m)) * (h\hat{\ }m)) + (B * ((h\hat{\ }n)\ /(\mathit{fact}\ n)))$
⟨*proof*⟩

**lemma** *eq-diff-eq'*: $x = y - z \longleftrightarrow y = x + z$
  **for** $x\ y\ z$ :: *real*
  ⟨*proof*⟩

**lemma** *fact-diff-Suc*: $n < Suc\ m \Longrightarrow fact\ (Suc\ m - n) = (Suc\ m - n) * fact\ (m - n)$
  ⟨*proof*⟩

**lemma** *Maclaurin-lemma2*:
  **fixes** *B*
  **assumes** *DERIV*: $\forall m\ t.\ m < n \wedge 0 \le t \wedge t \le h \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
    **and** *INIT*: $n = Suc\ k$
  **defines** $difg \equiv$
    $(\lambda m\ t{::}real.\ diff\ m\ t -$
      $((\sum p<n - m.\ diff\ (m + p)\ 0\ /\ fact\ p * t\ \hat{}\ p) + B * (t\ \hat{}\ (n - m)\ /\ fact\ (n - m))))$
    (**is** $difg \equiv (\lambda m\ t.\ diff\ m\ t - ?difg\ m\ t)$)
  **shows** $\forall m\ t.\ m < n \wedge 0 \le t \wedge t \le h \longrightarrow DERIV\ (difg\ m)\ t :> difg\ (Suc\ m)\ t$
⟨*proof*⟩

**lemma** *Maclaurin*:
  **assumes** *h*: $0 < h$
    **and** *n*: $0 < n$
    **and** *diff-0*: $diff\ 0 = f$
    **and** *diff-Suc*: $\forall m\ t.\ m < n \wedge 0 \le t \wedge t \le h \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
  **shows**
    $\exists t{::}real.\ 0 < t \wedge t < h \wedge$
      $f\ h = sum\ (\lambda m.\ (diff\ m\ 0\ /\ fact\ m) * h\ \hat{}\ m)\ \{..<n\} + (diff\ n\ t\ /\ fact\ n) * h\ \hat{}\ n$
⟨*proof*⟩

**lemma** *Maclaurin2*:
  **fixes** $n$ :: *nat*
    **and** $h$ :: *real*
  **assumes** *INIT1*: $0 < h$
    **and** *INIT2*: $diff\ 0 = f$
    **and** *DERIV*: $\forall m\ t.\ m < n \wedge 0 \le t \wedge t \le h \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
  **shows** $\exists t.\ 0 < t \wedge t \le h \wedge f\ h = (\sum m<n.\ diff\ m\ 0\ /\ (fact\ m) * h\ \hat{}\ m) + diff\ n\ t\ /\ fact\ n * h\ \hat{}\ n$
⟨*proof*⟩

**lemma** *Maclaurin-minus*:
  **fixes** $n$ :: *nat* **and** $h$ :: *real*
  **assumes** $h < 0\ 0 < n\ diff\ 0 = f$
    **and** *DERIV*: $\forall m\ t.\ m < n \wedge h \le t \wedge t \le 0 \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$

**shows** $\exists\, t.\ h < t \wedge t < 0 \wedge f\ h = (\sum m{<}n.\ \mathit{diff}\ m\ 0\ /\ \mathit{fact}\ m * h \ \hat{}\ m) + \mathit{diff}\ n\ t\ /\ \mathit{fact}\ n * h \ \hat{}\ n$
$\langle proof \rangle$

## 114.2 More Convenient "Bidirectional" Version.

**lemma** *Maclaurin-bi-le*:
**fixes** $n :: nat$ **and** $x :: real$
**assumes** $\mathit{diff}\ 0 = f$
**and** $DERIV : \forall\, m\ t.\ m < n \wedge |t| \leq |x| \longrightarrow DERIV\ (\mathit{diff}\ m)\ t :> \mathit{diff}\ (Suc\ m)\ t$
**shows** $\exists\, t.\ |t| \leq |x| \wedge f\ x = (\sum m{<}n.\ \mathit{diff}\ m\ 0\ /\ (\mathit{fact}\ m) * x \ \hat{}\ m) + \mathit{diff}\ n\ t\ /\ (\mathit{fact}\ n) * x \ \hat{}\ n$
(**is** $\exists\, t.\ \text{-} \wedge f\ x = ?f\ x\ t$)
$\langle proof \rangle$

**lemma** *Maclaurin-all-lt*:
**fixes** $x :: real$
**assumes** $INIT1$: $\mathit{diff}\ 0 = f$
**and** $INIT2$: $0 < n$
**and** $INIT3$: $x \neq 0$
**and** $DERIV$: $\forall\, m\ x.\ DERIV\ (\mathit{diff}\ m)\ x :> \mathit{diff}(Suc\ m)\ x$
**shows** $\exists\, t.\ 0 < |t| \wedge |t| < |x| \wedge f\ x =$
$(\sum m{<}n.\ (\mathit{diff}\ m\ 0\ /\ \mathit{fact}\ m) * x \ \hat{}\ m) + (\mathit{diff}\ n\ t\ /\ \mathit{fact}\ n) * x \ \hat{}\ n$
(**is** $\exists\, t.\ \text{-} \wedge \text{-} \wedge f\ x = ?f\ x\ t$)
$\langle proof \rangle$

**lemma** *Maclaurin-zero*: $x = 0 \Longrightarrow n \neq 0 \Longrightarrow (\sum m{<}n.\ (\mathit{diff}\ m\ 0\ /\ \mathit{fact}\ m) * x \ \hat{}\ m) = \mathit{diff}\ 0\ 0$
**for** $x :: real$ **and** $n :: nat$
$\langle proof \rangle$

**lemma** *Maclaurin-all-le*:
**fixes** $x :: real$ **and** $n :: nat$
**assumes** $INIT$: $\mathit{diff}\ 0 = f$
**and** $DERIV$: $\forall\, m\ x.\ DERIV\ (\mathit{diff}\ m)\ x :> \mathit{diff}\ (Suc\ m)\ x$
**shows** $\exists\, t.\ |t| \leq |x| \wedge f\ x = (\sum m{<}n.\ (\mathit{diff}\ m\ 0\ /\ \mathit{fact}\ m) * x \ \hat{}\ m) + (\mathit{diff}\ n\ t\ /\ \mathit{fact}\ n) * x \ \hat{}\ n$
(**is** $\exists\, t.\ \text{-} \wedge f\ x = ?f\ x\ t$)
$\langle proof \rangle$

**lemma** *Maclaurin-all-le-objl*:
$\mathit{diff}\ 0 = f \wedge (\forall\, m\ x.\ DERIV\ (\mathit{diff}\ m)\ x :> \mathit{diff}\ (Suc\ m)\ x) \longrightarrow$
$(\exists\, t{::}real.\ |t| \leq |x| \wedge f\ x = (\sum m{<}n.\ (\mathit{diff}\ m\ 0\ /\ \mathit{fact}\ m) * x \ \hat{}\ m) + (\mathit{diff}\ n\ t\ /\ \mathit{fact}\ n) * x \ \hat{}\ n)$
**for** $x :: real$ **and** $n :: nat$
$\langle proof \rangle$

## 114.3 Version for Exponential Function

**lemma** *Maclaurin-exp-lt*:
**fixes** $x$ :: *real* **and** $n$ :: *nat*
**shows**
$x \neq 0 \Longrightarrow n > 0 \Longrightarrow$
$(\exists\, t.\ 0 < |t| \wedge |t| < |x| \wedge exp\ x = (\sum m{<}n.\ (x \mathbin{\char`\^} m) / fact\ m) + (exp\ t / fact\ n) * x \mathbin{\char`\^} n)$
⟨*proof*⟩

**lemma** *Maclaurin-exp-le*:
**fixes** $x$ :: *real* **and** $n$ :: *nat*
**shows** $\exists\, t.\ |t| \leq |x| \wedge exp\ x = (\sum m{<}n.\ (x \mathbin{\char`\^} m) / fact\ m) + (exp\ t / fact\ n) * x \mathbin{\char`\^} n$
⟨*proof*⟩

**corollary** *exp-lower-Taylor-quadratic*: $0 \leq x \Longrightarrow 1 + x + x^2 / 2 \leq exp\ x$
**for** $x$ :: *real*
⟨*proof*⟩

**corollary** *ln-2-less-1*: $ln\ 2 < (1{::}real)$
⟨*proof*⟩

## 114.4 Version for Sine Function

**lemma** *mod-exhaust-less-4*: $m\ mod\ 4 = 0 \vee m\ mod\ 4 = 1 \vee m\ mod\ 4 = 2 \vee m\ mod\ 4 = 3$
**for** $m$ :: *nat*
⟨*proof*⟩

It is unclear why so many variant results are needed.

**lemma** *sin-expansion-lemma*: $sin\ (x + real\ (Suc\ m) * pi / 2) = cos\ (x + real\ m * pi / 2)$
⟨*proof*⟩

**lemma** *Maclaurin-sin-expansion2*:
$\exists\, t.\ |t| \leq |x| \wedge$
$sin\ x = (\sum m{<}n.\ sin\text{-}coeff\ m * x \mathbin{\char`\^} m) + (sin\ (t + 1/2 * real\ n * pi) / fact\ n) * x \mathbin{\char`\^} n$
⟨*proof*⟩

**lemma** *Maclaurin-sin-expansion*:
$\exists\, t.\ sin\ x = (\sum m{<}n.\ sin\text{-}coeff\ m * x \mathbin{\char`\^} m) + (sin\ (t + 1/2 * real\ n * pi) / fact\ n) * x \mathbin{\char`\^} n$
⟨*proof*⟩

**lemma** *Maclaurin-sin-expansion3*:
**assumes** $n > 0$ $x > 0$
**shows** $\exists\, t.\ 0 < t \wedge t < x \wedge$

$sin\ x = (\sum m<n.\ sin\text{-}coeff\ m * x\ \hat{}\ m) + (sin\ (t + 1/2 * real\ n * pi) / fact\ n) * x\ \hat{}\ n$
⟨proof⟩

**lemma** *Maclaurin-sin-expansion4*:
**assumes** $0 < x$
**shows** $\exists\, t.\ 0 < t \wedge t \leq x \wedge sin\ x = (\sum m<n.\ sin\text{-}coeff\ m * x\ \hat{}\ m) + (sin\ (t + 1/2 * real\ n * pi) / fact\ n) * x\ \hat{}\ n$
⟨proof⟩

## 114.5 Maclaurin Expansion for Cosine Function

**lemma** *sumr-cos-zero-one* [*simp*]: $(\sum m<Suc\ n.\ cos\text{-}coeff\ m * 0\ \hat{}\ m) = 1$
⟨proof⟩

**lemma** *cos-expansion-lemma*: $cos\ (x + real\ (Suc\ m) * pi / 2) = -\ sin\ (x + real\ m * pi / 2)$
⟨proof⟩

**lemma** *Maclaurin-cos-expansion*:
$\exists\, t{::}real.\ |t| \leq |x| \wedge$
$cos\ x = (\sum m<n.\ cos\text{-}coeff\ m * x\ \hat{}\ m) + (cos(t + 1/2 * real\ n * pi) / fact\ n) * x\ \hat{}\ n$
⟨proof⟩

**lemma** *Maclaurin-cos-expansion2*:
**assumes** $x > 0\ n > 0$
**shows** $\exists\, t.\ 0 < t \wedge t < x \wedge$
$cos\ x = (\sum m<n.\ cos\text{-}coeff\ m * x\ \hat{}\ m) + (cos\ (t + 1/2 * real\ n * pi) / fact\ n) * x\ \hat{}\ n$
⟨proof⟩

**lemma** *Maclaurin-minus-cos-expansion*:
**assumes** $n > 0\ x < 0$
**shows** $\exists\, t.\ x < t \wedge t < 0 \wedge$
$cos\ x = (\sum m<n.\ cos\text{-}coeff\ m * x\ \hat{}\ m) + ((cos\ (t + 1/2 * real\ n * pi) / fact\ n) * x\ \hat{}\ n)$
⟨proof⟩

**lemma** *sin-bound-lemma*: $x = y \Longrightarrow |u| \leq v \Longrightarrow |(x + u) - y| \leq v$
**for** $x\ y\ u\ v :: real$
⟨proof⟩

**lemma** *Maclaurin-sin-bound*: $|sin\ x - (\sum m<n.\ sin\text{-}coeff\ m * x\ \hat{}\ m)| \leq inverse\ (fact\ n) * |x|\ \hat{}\ n$

⟨*proof*⟩

## 115 Taylor series

We use MacLaurin and the translation of the expansion point *c* to *0* to prove Taylor's theorem.

**lemma** *Taylor-up*:
**assumes** *INIT*: $n > 0$ *diff 0* $= f$
**and** *DERIV*: $\forall m\ t.\ m < n \wedge a \le t \wedge t \le b \longrightarrow DERIV\ (diff\ m)\ t :> (diff\ (Suc\ m)\ t)$
**and** *INTERV*: $a \le c\ c < b$
**shows** $\exists t::real.\ c < t \wedge t < b \wedge$
$f\ b = (\sum m{<}n.\ (diff\ m\ c\ /\ fact\ m) * (b - c)\hat{}m) + (diff\ n\ t\ /\ fact\ n) * (b - c)\hat{}n$
⟨*proof*⟩

**lemma** *Taylor-down*:
**fixes** $a$ :: *real* **and** $n$ :: *nat*
**assumes** *INIT*: $n > 0$ *diff 0* $= f$
**and** *DERIV*: $(\forall m\ t.\ m < n \wedge a \le t \wedge t \le b \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t)$
**and** *INTERV*: $a < c\ c \le b$
**shows** $\exists t.\ a < t \wedge t < c \wedge$
$f\ a = (\sum m{<}n.\ (diff\ m\ c\ /\ fact\ m) * (a - c)\hat{}m) + (diff\ n\ t\ /\ fact\ n) * (a - c)\hat{}n$
⟨*proof*⟩

**theorem** *Taylor*:
**fixes** $a$ :: *real* **and** $n$ :: *nat*
**assumes** *INIT*: $n > 0$ *diff 0* $= f$
**and** *DERIV*: $\forall m\ t.\ m < n \wedge a \le t \wedge t \le b \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
**and** *INTERV*: $a \le c\ \ c \le b\ a \le x\ x \le b\ x \ne c$
**shows** $\exists t.$
$(if\ x < c\ then\ x < t \wedge t < c\ else\ c < t \wedge t < x) \wedge$
$f\ x = (\sum m{<}n.\ (diff\ m\ c\ /\ fact\ m) * (x - c)\hat{}m) + (diff\ n\ t\ /\ fact\ n) * (x - c)\hat{}n$
⟨*proof*⟩

**end**

## 116 More facts about binomial coefficients

These facts could have been proven before, but having real numbers makes the proofs a lot easier. Thanks to Alexander Maletzky among others.

**theory** *Binomial-Plus*

**imports** *Real*
**begin**

## 116.1 More facts about binomial coefficients

These facts could have been proven before, but having real numbers makes the proofs a lot easier.

**lemma** *central-binomial-odd*:
$odd\ n \Longrightarrow n\ choose\ (Suc\ (n\ div\ 2)) = n\ choose\ (n\ div\ 2)$
⟨*proof*⟩

**lemma** *binomial-less-binomial-Suc*:
**assumes** $k$: $k < n\ div\ 2$
**shows** $n\ choose\ k < n\ choose\ (Suc\ k)$
⟨*proof*⟩

**lemma** *binomial-strict-mono*:
**assumes** $k < k'\ 2{*}k' \le n$
**shows** $n\ choose\ k < n\ choose\ k'$
⟨*proof*⟩

**lemma** *binomial-mono*:
**assumes** $k \le k'\ 2{*}k' \le n$
**shows** $n\ choose\ k \le n\ choose\ k'$
⟨*proof*⟩

**lemma** *binomial-strict-antimono*:
**assumes** $k < k'\ 2 * k \ge n\ k' \le n$
**shows** $n\ choose\ k > n\ choose\ k'$
⟨*proof*⟩

**lemma** *binomial-antimono*:
**assumes** $k \le k'\ k \ge n\ div\ 2\ k' \le n$
**shows** $n\ choose\ k \ge n\ choose\ k'$
⟨*proof*⟩

**lemma** *binomial-maximum*: $n\ choose\ k \le n\ choose\ (n\ div\ 2)$
⟨*proof*⟩

**lemma** *binomial-maximum'*: $(2{*}n)\ choose\ k \le (2{*}n)\ choose\ n$
⟨*proof*⟩

**lemma** *central-binomial-lower-bound*:
**assumes** $n > 0$
**shows** $4\hat{}n\ /\ (2{*}real\ n) \le real\ ((2{*}n)\ choose\ n)$
⟨*proof*⟩

**lemma** *upper-le-binomial*:
**assumes** $0 < k$ **and** $k < n$

**shows** $n \le n$ *choose* $k$
⟨*proof*⟩

## 116.2 Results about binomials and integers, thanks to Alexander Maletzky

Restore original sort constraints: semidom rather than field of char 0

⟨*ML*⟩

**lemma** *gbinomial-eq-0-int*:
**assumes** $n < k$
**shows** $(int\ n)\ gchoose\ k = 0$
⟨*proof*⟩

**corollary** *gbinomial-eq-0*: $0 \le a \Longrightarrow a < int\ k \Longrightarrow a\ gchoose\ k = 0$
⟨*proof*⟩

**lemma** *gbinomial-mono*:
**fixes** $k$::*nat* **and** $a$::*real*
**assumes** *of-nat* $k \le a$ $a \le b$ **shows** $a\ gchoose\ k \le b\ gchoose\ k$
⟨*proof*⟩

**lemma** *int-binomial*: $int\ (n\ choose\ k) = (int\ n)\ gchoose\ k$
⟨*proof*⟩

**lemma** *falling-fact-pochhammer*: $prod\ (\lambda i.\ a - int\ i)\ \{0..<k\} = (-\ 1)\ \hat{}\ k * pochhammer\ (-\ a)\ k$
⟨*proof*⟩

**lemma** *falling-fact-pochhammer'*: $prod\ (\lambda i.\ a - int\ i)\ \{0..<k\} = pochhammer\ (a - int\ k + 1)\ k$
⟨*proof*⟩

**lemma** *gbinomial-int-pochhammer*: $(a{::}int)\ gchoose\ k = (-\ 1)\ \hat{}\ k * pochhammer\ (-\ a)\ k\ div\ fact\ k$
⟨*proof*⟩

**lemma** *gbinomial-int-pochhammer'*: $a\ gchoose\ k = pochhammer\ (a - int\ k + 1)\ k\ div\ fact\ k$
⟨*proof*⟩

**lemma** *fact-dvd-pochhammer*: $fact\ k\ dvd\ pochhammer\ (a{::}int)\ k$
⟨*proof*⟩

**lemma** *gbinomial-int-negated-upper*: $(a\ gchoose\ k) = (-1)\ \hat{}\ k * ((int\ k - a - 1)\ gchoose\ k)$
⟨*proof*⟩

**lemma** *gbinomial-int-mult-fact*: $fact\ k * (a\ gchoose\ k) = (\prod i = 0..<k.\ a - int\ i)$

⟨*proof*⟩

**corollary** *gbinomial-int-mult-fact'*: $(a\ gchoose\ k) * fact\ k = (\prod i = 0..<k.\ a - int\ i)$
⟨*proof*⟩

**lemma** *gbinomial-int-binomial*:
$a\ gchoose\ k = (if\ 0 \le a\ then\ int\ ((nat\ a)\ choose\ k)\ else\ (-1::int)\hat{\ }k * int\ ((k + (nat\ (-\ a)) - 1)\ choose\ k))$
⟨*proof*⟩

**corollary** *gbinomial-nneg*: $0 \le a \Longrightarrow a\ gchoose\ k = int\ ((nat\ a)\ choose\ k)$
⟨*proof*⟩

**corollary** *gbinomial-neg*: $a < 0 \Longrightarrow a\ gchoose\ k = (-1::int)\hat{\ }k * int\ ((k + (nat\ (-\ a)) - 1)\ choose\ k)$
⟨*proof*⟩

**lemma** *of-int-gbinomial*: $of\text{-}int\ (a\ gchoose\ k) = (of\text{-}int\ a :: 'a::field\text{-}char\text{-}0)\ gchoose\ k$
⟨*proof*⟩

**lemma** *uminus-one-gbinomial* [*simp*]: $(-\ 1::int)\ gchoose\ k = (-\ 1)\ \hat{}\ k$
⟨*proof*⟩

**lemma** *gbinomial-int-Suc-Suc*: $(x + 1::int)\ gchoose\ (Suc\ k) = (x\ gchoose\ k) + (x\ gchoose\ (Suc\ k))$
⟨*proof*⟩

**corollary** *plus-Suc-gbinomial*:
$(x + (1 + int\ k))\ gchoose\ (Suc\ k) = ((x + int\ k)\ gchoose\ k) + ((x + int\ k)\ gchoose\ (Suc\ k))$
(**is** *?l* = *?r*)
⟨*proof*⟩

**lemma** *gbinomial-int-n-n* [*simp*]: $(int\ n)\ gchoose\ n = 1$
⟨*proof*⟩

**lemma** *gbinomial-int-Suc-n* [*simp*]: $(1 + int\ n)\ gchoose\ n = 1 + int\ n$
⟨*proof*⟩

**lemma** *zbinomial-eq-0-iff* [*simp*]: $a\ gchoose\ k = 0 \longleftrightarrow (0 \le a \wedge a < int\ k)$
⟨*proof*⟩

## 116.3 Sums

**lemma** *gchoose-rising-sum-nat*: $(\sum j \le n.\ int\ j + int\ k\ gchoose\ k) = (int\ n + int\ k + 1)\ gchoose\ (Suc\ k)$
⟨*proof*⟩

**lemma** *gchoose-rising-sum*:
**assumes** $0 \leq n$ — Necessary condition.
**shows** $(\sum j{=}0..n.\ j + int\ k\ gchoose\ k) = (n + int\ k + 1)\ gchoose\ (Suc\ k)$
$\langle proof \rangle$

**end**

# 117 Comprehensive Complex Theory

**theory** *Complex-Main*
**imports**
*Complex*
*MacLaurin*
*Binomial-Plus*
**begin**

**end**

# References

[1] H. Davenport. *The Higher Arithmetic.* Cambridge University Press, 1992.

[2] M. Gordon. HOL: A machine oriented formulation of higher-order logic. Technical Report 68, University of Cambridge, Computer Laboratory, 1985.

[3] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* Addison–Wesley, Boston, MA, USA, 2nd edition, 1994.

[4] M. Holz, K. Steffens, and E. Weitz. *Introduction to Cardinal Arithmetic.* Birkhäuser, 1999.

[5] D. Leijen. Division and modulus for computer scientists. 2001.

[6] C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications*, LNCS 664, pages 328–345. Springer, 1993.