

ZF

Lawrence C Paulson and others

December 17, 2025

Contents

1	Base of Zermelo-Fraenkel Set Theory	3
1.1	Signature	3
1.2	Bounded Quantifiers	3
1.3	Variations on Replacement	3
1.4	General union and intersection	4
1.5	Finite sets and binary operations	4
1.6	Axioms	5
1.7	Definite descriptions – via Replace over the set "1"	5
1.8	Ordered Pairing	6
1.9	Relations and Functions	7
1.10	ASCII syntax	8
1.11	Substitution	9
1.12	Bounded universal quantifier	9
1.13	Bounded existential quantifier	9
1.14	Rules for subsets	10
1.15	Rules for equality	11
1.16	Rules for Replace – the derived form of replacement	11
1.17	Rules for RepFun	12
1.18	Rules for Collect – forming a subset by separation	12
1.19	Rules for Unions	13
1.20	Rules for Unions of families	13
1.21	Rules for the empty set	13
1.22	Rules for Inter	14
1.23	Rules for Intersections of families	14
1.24	Rules for Powersets	15
1.25	Cantor's Theorem: There is no surjection from a set to its powerset.	15

2	Unordered Pairs	15
2.1	Unordered Pairs: constant <i>Upair</i>	15
2.2	Rules for Binary Union, Defined via <i>Upair</i>	15
2.3	Rules for Binary Intersection, Defined via <i>Upair</i>	16
2.4	Rules for Set Difference, Defined via <i>Upair</i>	16
2.5	Rules for <i>cons</i>	17
2.6	Singletons	17
2.7	Descriptions	17
2.8	Conditional Terms: <i>if-then-else</i>	18
2.9	Consequences of Foundation	19
2.10	Rules for Successor	20
2.11	Miniscoping of the Bounded Universal Quantifier	20
2.12	Miniscoping of the Bounded Existential Quantifier	21
2.13	Miniscoping of the Replacement Operator	22
2.14	Miniscoping of Unions	22
2.15	Miniscoping of Intersections	23
2.16	Other simprules	24
3	Ordered Pairs	24
3.1	Sigma: Disjoint Union of a Family of Sets	25
3.2	Projections <i>fst</i> and <i>snd</i>	26
3.3	The Eliminator, <i>split</i>	26
3.4	A version of <i>split</i> for Formulae: Result Type <i>o</i>	26
4	Basic Equalities and Inclusions	27
4.1	Bounded Quantifiers	27
4.2	Converse of a Relation	28
4.3	Finite Set Constructions Using <i>cons</i>	28
4.4	Binary Intersection	30
4.5	Binary Union	31
4.6	Set Difference	32
4.7	Big Union and Intersection	34
4.8	Unions and Intersections of Families	35
4.9	Image of a Set under a Function or Relation	41
4.10	Inverse Image of a Set under a Function or Relation	42
4.11	Powerset Operator	44
4.12	RepFun	44
4.13	Collect	45
5	Least and Greatest Fixed Points; the Knaster-Tarski Theorem	46
5.1	Monotone Operators	46
5.2	Proof of Knaster-Tarski Theorem using <i>lfp</i>	47
5.3	General Induction Rule for Least Fixedpoints	48

5.4	Proof of Knaster-Tarski Theorem using <i>gfp</i>	49
5.5	Coinduction Rules for Greatest Fixed Points	49
6	Booleans in Zermelo-Fraenkel Set Theory	50
6.1	Laws About 'not'	52
6.2	Laws About 'and'	52
6.3	Laws About 'or'	53
7	Disjoint Sums	53
7.1	Rules for the <i>Part</i> Primitive	54
7.2	Rules for Disjoint Sums	54
7.3	The Eliminator: <i>case</i>	56
7.4	More Rules for <i>Part</i> (<i>A</i> , <i>h</i>)	56
8	Functions, Function Spaces, Lambda-Abstraction	57
8.1	The Pi Operator: Dependent Function Space	57
8.2	Function Application	58
8.3	Lambda Abstraction	59
8.4	Extensionality	61
8.5	Images of Functions	61
8.6	Properties of <i>restrict</i> (<i>f</i> , <i>A</i>)	62
8.7	Unions of Functions	63
8.8	Domain and Range of a Function or Relation	64
8.9	Extensions of Functions	64
8.10	Function Updates	64
8.11	Monotonicity Theorems	65
8.11.1	Replacement in its Various Forms	65
8.11.2	Standard Products, Sums and Function Spaces	66
8.11.3	Converse, Domain, Range, Field	66
8.11.4	Images	67
9	Quine-Inspired Ordered Pairs and Disjoint Sums	67
9.1	Quine ordered pairing	69
9.1.1	QSigma: Disjoint union of a family of sets Generalizes Cartesian product	69
9.1.2	Projections: <i>qfst</i> , <i>qsnd</i>	70
9.1.3	Eliminator: <i>qsplit</i>	70
9.1.4	<i>qsplit</i> for predicates: result type <i>o</i>	70
9.1.5	<i>qconverse</i>	71
9.2	The Quine-inspired notion of disjoint sum	71
9.2.1	Eliminator – <i>qcase</i>	72
9.2.2	Monotonicity	73

10 Injections, Surjections, Bijections, Composition	73
10.1 Surjective Function Space	74
10.2 Injective Function Space	75
10.3 Bijections	75
10.4 Identity Function	76
10.5 Converse of a Function	76
10.6 Converses of Injections, Surjections, Bijections	77
10.7 Composition of Two Relations	77
10.8 Domain and Range – see Suppes, Section 3.1	78
10.9 Other Results	78
10.10 Composition Preserves Functions, Injections, and Surjections	79
10.11 Dual Properties of <i>inj</i> and <i>surj</i>	79
10.11.1 Inverses of Composition	80
10.11.2 Proving that a Function is a Bijection	80
10.11.3 Unions of Functions	80
10.11.4 Restrictions as Surjections and Bijections	81
10.11.5 Lemmas for Ramsey’s Theorem	81
11 Relations: Their General Properties and Transitive Closure	82
11.1 General properties of relations	83
11.1.1 irreflexivity	83
11.1.2 symmetry	83
11.1.3 antisymmetry	83
11.1.4 transitivity	83
11.2 Transitive closure of a relation	83
12 Well-Founded Recursion	87
12.1 Well-Founded Relations	88
12.1.1 Equivalences between <i>wf</i> and <i>wf-on</i>	88
12.1.2 Introduction Rules for <i>wf-on</i>	88
12.1.3 Well-founded Induction	89
12.2 Basic Properties of Well-Founded Relations	90
12.3 The Predicate <i>is-recfun</i>	90
12.4 Recursion: Main Existence Lemma	91
12.5 Unfolding <i>wftrec</i> (<i>r</i> , <i>a</i> , <i>H</i>)	91
12.5.1 Removal of the Premise <i>trans</i> (<i>r</i>)	91
13 Transitive Sets and Ordinals	92
13.1 Rules for Transset	93
13.1.1 Three Neat Characterisations of Transset	93
13.1.2 Consequences of Downwards Closure	93
13.1.3 Closure Properties	93
13.2 Lemmas for Ordinals	94
13.3 The Construction of Ordinals: 0, succ, Union	95

13.4	$<$ is 'less Than' for Ordinals	95
13.5	Natural Deduction Rules for Memrel	97
13.6	Transfinite Induction	98
14	Fundamental properties of the epsilon ordering ($<$ on ordinals)	98
14.0.1	Proving That $<$ is a Linear Ordering on the Ordinals	98
14.0.2	Some Rewrite Rules for $<, \leq$	99
14.1	Results about Less-Than or Equals	99
14.1.1	Transitivity Laws	100
14.1.2	Union and Intersection	100
14.2	Results about Limits	101
14.3	Limit Ordinals – General Properties	102
14.3.1	Traditional 3-Way Case Analysis on Ordinals	103
15	Special quantifiers	104
15.1	Quantifiers and union operator for ordinals	104
15.1.1	simplification of the new quantifiers	105
15.1.2	Union over ordinals	105
15.1.3	universal quantifier for ordinals	106
15.1.4	existential quantifier for ordinals	107
15.1.5	Rules for Ordinal-Indexed Unions	107
15.2	Quantification over a class	107
15.2.1	Relativized universal quantifier	108
15.2.2	Relativized existential quantifier	108
15.2.3	One-point rule for bounded quantifiers	110
15.2.4	Sets as Classes	110
16	The Natural numbers As a Least Fixed Point	111
16.1	Injectivity Properties and Induction	112
16.2	Variations on Mathematical Induction	113
16.3	quasinat: to allow a case-split rule for <i>nat-case</i>	114
16.4	Recursion on the Natural Numbers	115
17	Inductive and Coinductive Definitions	115
18	Epsilon Induction and Recursion	116
18.1	Basic Closure Properties	116
18.2	Leastness of <i>eclose</i>	117
18.3	Epsilon Recursion	118
18.4	Rank	119
18.5	Corollaries of Leastness	120

19 Partial and Total Orderings: Basic Definitions and Properties	121
19.1 Immediate Consequences of the Definitions	122
19.2 Restricting an Ordering's Domain	123
19.3 Empty and Unit Domains	124
19.3.1 Relations over the Empty Set	124
19.3.2 The Empty Relation Well-Orders the Unit Set	125
19.4 Order-Isomorphisms	125
19.5 Main results of Kunen, Chapter 1 section 6	127
19.6 Towards Kunen's Theorem 6.3: Linearity of the Similarity Relation	128
19.7 Miscellaneous Results by Krzysztof Grabczewski	129
19.8 Lemmas for the Reflexive Orders	130
20 Combining Orderings: Foundations of Ordinal Arithmetic	131
20.1 Addition of Relations – Disjoint Sum	131
20.1.1 Rewrite rules. Can be used to obtain introduction rules	131
20.1.2 Elimination Rule	132
20.1.3 Type checking	132
20.1.4 Linearity	132
20.1.5 Well-foundedness	132
20.1.6 An <i>ord-iso</i> congruence law	132
20.1.7 Associativity	133
20.2 Multiplication of Relations – Lexicographic Product	133
20.2.1 Rewrite rule. Can be used to obtain introduction rules	133
20.2.2 Type checking	133
20.2.3 Linearity	134
20.2.4 Well-foundedness	134
20.2.5 An <i>ord-iso</i> congruence law	134
20.2.6 Distributive law	135
20.2.7 Associativity	135
20.3 Inverse Image of a Relation	135
20.3.1 Rewrite rule	135
20.3.2 Type checking	135
20.3.3 Partial Ordering Properties	135
20.3.4 Linearity	136
20.3.5 Well-foundedness	136
20.4 Every well-founded relation is a subset of some inverse image of an ordinal	136
20.5 Other Results	137
20.5.1 The Empty Relation	137
20.5.2 The "measure" relation is useful with wfrec	138
20.5.3 Well-foundedness of Unions	138
20.5.4 Bijections involving Powersets	138

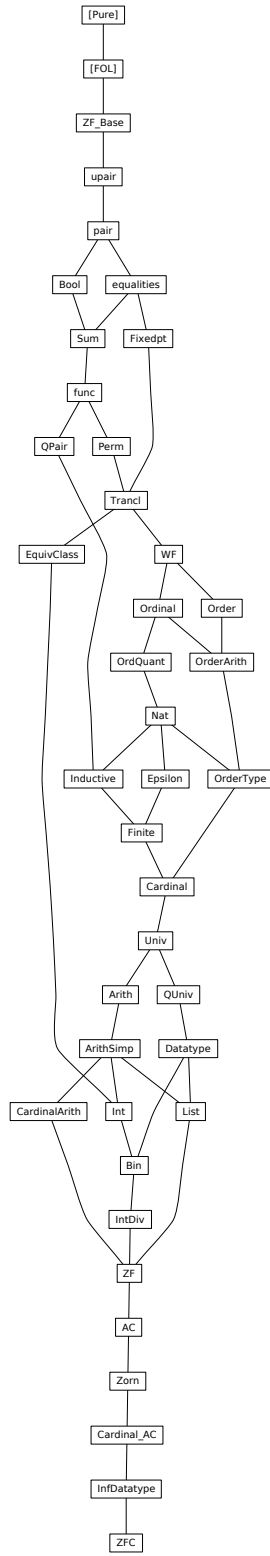
21 Order Types and Ordinal Arithmetic	139
21.1 Proofs needing the combination of Ordinal.thy and Order.thy	140
21.2 Ordermap and ordertype	140
21.2.1 Unfolding of ordermap	140
21.2.2 Showing that ordermap, ordertype yield ordinals . . .	141
21.2.3 ordermap preserves the orderings in both directions .	141
21.2.4 Isomorphisms involving ordertype	141
21.2.5 Basic equalities for ordertype	142
21.2.6 A fundamental unfolding law for ordertype.	142
21.3 Alternative definition of ordinal	142
21.4 Ordinal Addition	143
21.4.1 Order Type calculations for radd	143
21.4.2 ordify: trivial coercion to an ordinal	144
21.4.3 Basic laws for ordinal addition	144
21.4.4 Ordinal addition with successor – via associativity! .	145
21.5 Ordinal Subtraction	147
21.6 Ordinal Multiplication	147
21.6.1 A useful unfolding law	147
21.6.2 Basic laws for ordinal multiplication	148
21.6.3 Ordering/monotonicity properties of ordinal multipli- cation	149
21.7 The Relation Lt	150
22 Finite Powerset Operator and Finite Function Space	150
22.1 Finite Powerset Operator	151
22.2 Finite Function Space	152
22.3 The Contents of a Singleton Set	153
23 Cardinal Numbers Without the Axiom of Choice	153
23.1 The Schroeder-Bernstein Theorem	154
23.2 lesspoll: contributions by Krzysztof Grabczewski	156
23.3 Basic Properties of Cardinals	157
23.4 The finite cardinals	159
23.5 The first infinite cardinal: Omega, or nat	161
23.6 Towards Cardinal Arithmetic	161
23.7 Lemmas by Krzysztof Grabczewski	162
23.8 Finite and infinite sets	163
24 The Cumulative Hierarchy and a Small Universe for Recursive Types	166
24.1 Immediate Consequences of the Definition of $Vfrom(A, i)$. .	166
24.1.1 Monotonicity	167
24.1.2 A fundamental equality: $Vfrom$ does not require or- dinals!	167

24.2	Basic Closure Properties	167
24.2.1	Finite sets and ordered pairs	167
24.3	0, Successor and Limit Equations for $Vfrom$	168
24.4	$Vfrom$ applied to Limit Ordinals	168
24.4.1	Closure under Disjoint Union	169
24.5	Properties assuming $Transset(A)$	169
24.5.1	Products	170
24.5.2	Disjoint Sums, or Quine Ordered Pairs	170
24.5.3	Function Space!	170
24.6	The Set $Vset(i)$	171
24.6.1	Characterisation of the elements of $Vset(i)$	171
24.6.2	Reasoning about Sets in Terms of Their Elements' Ranks	171
24.6.3	Set Up an Environment for Simplification	172
24.6.4	Recursion over Vset Levels!	172
24.7	The Datatype Universe: $univ(A)$	172
24.7.1	The Set $univ(A)$ as a Limit	173
24.8	Closure Properties for $univ(A)$	173
24.8.1	Closure under Unordered and Ordered Pairs	173
24.8.2	The Natural Numbers	174
24.8.3	Instances for 1 and 2	174
24.8.4	Closure under Disjoint Union	174
24.9	Finite Branching Closure Properties	174
24.9.1	Closure under Finite Powerset	174
24.9.2	Closure under Finite Powers: Functions from a Natural Number	175
24.9.3	Closure under Finite Function Space	175
24.10*	For QUniv. Properties of $Vfrom$ analogous to the "take-lemma" *	176
25	A Small Universe for Lazy Recursive Types	176
25.1	Properties involving $Transset$ and Sum	177
25.2	Introduction and Elimination Rules	177
25.3	Closure Properties	177
25.4	Quine Disjoint Sum	178
25.5	Closure for Quine-Inspired Products and Sums	178
25.6	Quine Disjoint Sum	178
25.7	The Natural Numbers	179
25.8	"Take-Lemma" Rules	179
26	Datatype and CoDatatype Definitions	179

27 Arithmetic Operators and Their Definitions	180
27.1 <i>natify</i> , the Coercion to <i>nat</i>	181
27.2 Typing rules	182
27.3 Addition	183
27.4 Monotonicity of Addition	185
27.5 Multiplication	187
28 Arithmetic with simplification	188
28.1 Arithmetic simplification	188
28.1.1 Examples	188
28.2 Difference	189
28.3 Remainder	190
28.4 Division	191
28.5 Further Facts about Remainder	192
28.6 Additional theorems about \leq	192
28.7 Cancellation Laws for Common Factors in Comparisons . . .	193
28.8 More Lemmas about Remainder	194
28.8.1 More Lemmas About Difference	195
29 Lists in Zermelo-Fraenkel Set Theory	196
29.1 The function <i>zip</i>	209
30 Equivalence Relations	215
30.1 Suppes, Theorem 70: <i>r</i> is an equiv relation iff <i>converse(r)</i> <i>O</i> <i>r</i> = <i>r</i>	216
30.2 Defining Unary Operations upon Equivalence Classes	217
30.3 Defining Binary Operations upon Equivalence Classes	218
31 The Integers as Equivalence Classes Over Pairs of Natural Numbers	219
31.1 Proving that <i>intrel</i> is an equivalence relation	220
31.2 Collapsing rules: to remove <i>intify</i> from arithmetic expressions	222
31.3 <i>zminus</i> : unary negation on <i>int</i>	223
31.4 <i>znegative</i> : the test for negative integers	223
31.5 <i>nat-of</i> : Coercion of an Integer to a Natural Number	224
31.6 <i>zmagnitude</i> : magnitude of an integer, as a natural number . .	224
31.7 (<i>\$+</i>): addition on <i>int</i>	225
31.8 (<i>\$*</i>): Integer Multiplication	227
31.9 The "Less Than" Relation	229
31.10 Less Than or Equals	231
31.11 More subtraction laws (for <i>zcompare-rls</i>)	231
31.12 Monotonicity and Cancellation Results for Instantiation of the <i>CancelNumerals</i> Simprocs	232
31.13 Comparison laws	233

31.13.1	More inequality lemmas	233
31.13.2	The next several equations are permutative: watch out!	234
32	Arithmetic on Binary Integers	234
32.0.1	The Carry and Borrow Functions, <i>bin-succ</i> and <i>bin-pred</i>	237
32.0.2	<i>bin-minus</i> : Unary Negation of Binary Integers	237
32.0.3	<i>bin-add</i> : Binary Addition	237
32.0.4	<i>bin-mult</i> : Binary Multiplication	238
32.1	Computations	238
32.2	Simplification Rules for Comparison of Binary Numbers . . .	240
32.2.1	Examples	246
33	The Division Operators Div and Mod	247
33.1	Uniqueness and monotonicity of quotients and remainders . .	251
33.2	Correctness of posDivAlg, the Division Algorithm for $a \geq 0$ and $b > 0$	252
33.3	Some convenient biconditionals for products of signs	253
33.4	Correctness of negDivAlg, the division algorithm for $a < 0$ and $b > 0$	254
33.5	Existence shown by proving the division algorithm to be correct	255
33.6	division of a number by itself	258
33.7	Computation of division and remainder	259
33.8	Monotonicity in the first argument (divisor)	261
33.9	Monotonicity in the second argument (dividend)	261
33.10	More algebraic laws for zdiv and zmod	262
33.11	proving $a \text{ zdiv } (b * c) = (a \text{ zdiv } b) \text{ zdiv } c$	264
33.12	Cancellation of common factors in "zdiv"	265
33.13	Distribution of factors over "zmod"	265
34	Cardinal Arithmetic Without the Axiom of Choice	266
34.1	Cardinal addition	267
34.1.1	Cardinal addition is commutative	267
34.1.2	Cardinal addition is associative	267
34.1.3	0 is the identity for addition	268
34.1.4	Addition by another cardinal	268
34.1.5	Monotonicity of addition	268
34.1.6	Addition of finite cardinals is "ordinary" addition . . .	268
34.2	Cardinal multiplication	269
34.2.1	Cardinal multiplication is commutative	269
34.2.2	Cardinal multiplication is associative	269
34.2.3	Cardinal multiplication distributes over addition . . .	269
34.2.4	Multiplication by 0 yields 0	269
34.2.5	1 is the identity for multiplication	269
34.3	Some inequalities for multiplication	269

34.3.1	Multiplication by a non-zero cardinal	270
34.3.2	Monotonicity of multiplication	270
34.4	Multiplication of finite cardinals is "ordinary" multiplication .	270
34.5	Infinite Cardinals are Limit Ordinals	271
34.5.1	Establishing the well-ordering	271
34.5.2	Characterising initial segments of the well-ordering . .	271
34.5.3	The cardinality of initial segments	272
34.5.4	Toward's Kunen's Corollary 10.13 (1)	272
34.6	For Every Cardinal Number There Exists A Greater One . .	273
34.7	Basic Properties of Successor Cardinals	273
34.7.1	Removing elements from a finite set decreases its car- dinality	274
34.7.2	Theorems by Krzysztof Grabczewski, proofs by lcp . .	275
35	Main ZF Theory: Everything Except AC	275
35.1	Iteration of the function F	275
35.2	Transfinite Recursion	276
36	The Axiom of Choice	276
37	Zorn's Lemma	277
37.1	Mathematical Preamble	278
37.2	The Transfinite Construction	278
37.3	Some Properties of the Transfinite Construction	279
37.4	Hausdorff's Theorem: Every Set Contains a Maximal Chain .	280
37.5	Zorn's Lemma: If All Chains in S Have Upper Bounds In S , then S contains a Maximal Element	280
37.6	Zermelo's Theorem: Every Set can be Well-Ordered	281
37.7	Zorn's Lemma for Partial Orders	282
38	Cardinal Arithmetic Using AC	282
38.1	Strengthened Forms of Existing Theorems on Cardinals . . .	282
38.2	The relationship between cardinality and le-pollence	283
38.3	Other Applications of AC	283
38.4	The Main Result for Infinite-Branching Datatypes	284
39	Infinite-Branching Datatype Definitions	284



1 Base of Zermelo-Fraenkel Set Theory

```
theory ZF-Base
imports FOL
begin
```

1.1 Signature

```
declare [[eta-contract = false]]
```

```
typedecl i
instance i :: term ⟨proof⟩
```

```
axiomatization mem :: [i, i] ⇒ o (infixl ⟨∈⟩ 50) — membership relation
and zero :: i (⟨0⟩) — the empty set
and Pow :: i ⇒ i — power sets
and Inf :: i — infinite set
and Union :: i ⇒ i (⟨⟨open-block notation=⟨prefix ∪⟩⟩ ∪ -⟩ [90] 90)
and PrimReplace :: [i, [i, i] ⇒ o] ⇒ i
```

```
abbreviation not-mem :: [i, i] ⇒ o (infixl ⟨∉⟩ 50) — negated membership
relation
where x ∉ y ≡ ¬ (x ∈ y)
```

1.2 Bounded Quantifiers

```
definition Ball :: [i, i ⇒ o] ⇒ o
where Ball(A, P) ≡ ∀ x. x ∈ A ⟶ P(x)
```

```
definition Bex :: [i, i ⇒ o] ⇒ o
where Bex(A, P) ≡ ∃ x. x ∈ A ∧ P(x)
```

syntax

```
-Ball :: [pttrn, i, o] ⇒ o (⟨⟨indent=3 notation=⟨binder ∀ ∈⟩⟩ ∀ - ∈ - / -⟩ 10)
-Bex :: [pttrn, i, o] ⇒ o (⟨⟨indent=3 notation=⟨binder ∃ ∈⟩⟩ ∃ - ∈ - / -⟩ 10)
```

syntax-consts

```
-Ball ⇐ Ball and
-Bex ⇐ Bex
```

translations

```
∀ x ∈ A. P ⇐ CONST Ball(A, λx. P)
∃ x ∈ A. P ⇐ CONST Bex(A, λx. P)
```

1.3 Variations on Replacement

```
definition Replace :: [i, [i, i] ⇒ o] ⇒ i
where Replace(A, P) ≡ PrimReplace(A, λx y. (∃! z. P(x, z)) ∧ P(x, y))
```

syntax

```
-Replace :: [pttrn, pttrn, i, o] ⇒ i (⟨⟨indent=1 notation=⟨mixfix relational
replacement⟩⟩ { - / - ∈ -, - }⟩)
```

syntax-consts

-*Replace* \Rightarrow *Replace*

translations

$\{y. x \in A, Q\} \Rightarrow \text{CONST } \text{Replace}(A, \lambda x y. Q)$

definition *RepFun* :: $[i, i \Rightarrow i] \Rightarrow i$

where *RepFun*(*A*,*f*) $\equiv \{y . x \in A, y=f(x)\}$

syntax

-*RepFun* :: $[i, \text{pttrn}, i] \Rightarrow i$ ($\langle \langle \text{indent}=1 \text{ notation}=\langle \text{mixfix functional replace-ment} \rangle \rangle \{- ./ - \in -\} \rangle [51,0,51] \rangle$)

syntax-consts

-*RepFun* \Rightarrow *RepFun*

translations

$\{b. x \in A\} \Rightarrow \text{CONST } \text{RepFun}(A, \lambda x. b)$

definition *Collect* :: $[i, i \Rightarrow o] \Rightarrow i$

where *Collect*(*A*,*P*) $\equiv \{y . x \in A, x=y \wedge P(x)\}$

syntax

-*Collect* :: $[\text{pttrn}, i, o] \Rightarrow i$ ($\langle \langle \text{indent}=1 \text{ notation}=\langle \text{mixfix set comprehension} \rangle \rangle \{- \in - ./ -\} \rangle \rangle$)

syntax-consts

-*Collect* \Rightarrow *Collect*

translations

$\{x \in A. P\} \Rightarrow \text{CONST } \text{Collect}(A, \lambda x. P)$

1.4 General union and intersection

definition *Inter* :: $i \Rightarrow i$ ($\langle \langle \text{open-block notation}=\langle \text{prefix } \bigcap \rangle \bigcap - \rangle [90] 90 \rangle$)

where $\bigcap(A) \equiv \{x \in \bigcup(A) . \forall y \in A. x \in y\}$

syntax

-*UNION* :: $[\text{pttrn}, i, i] \Rightarrow i$ ($\langle \langle \text{indent}=3 \text{ notation}=\langle \text{binder } \bigcup \in \rangle \bigcup - \in - ./ - \rangle 10 \rangle$)

-*INTER* :: $[\text{pttrn}, i, i] \Rightarrow i$ ($\langle \langle \text{indent}=3 \text{ notation}=\langle \text{binder } \bigcap \in \rangle \bigcap - \in - ./ - \rangle 10 \rangle$)

syntax-consts

-*UNION* == *Union* and

-*INTER* == *Inter*

translations

$\bigcup_{x \in A} B == \text{CONST } \text{Union}(\{B. x \in A\})$

$\bigcap_{x \in A} B == \text{CONST } \text{Inter}(\{B. x \in A\})$

1.5 Finite sets and binary operations

definition *Upair* :: $[i, i] \Rightarrow i$

where *Upair*(*a*,*b*) $\equiv \{y. x \in \text{Pow}(\text{Pow}(\theta)), (x=\theta \wedge y=a) \mid (x=\text{Pow}(\theta) \wedge y=b)\}$

definition *Subset* :: $[i, i] \Rightarrow o$ (**infixl** $\langle \subseteq \rangle$ 50) — subset relation
where *subset-def*: $A \subseteq B \equiv \forall x \in A. x \in B$

definition *Diff* :: $[i, i] \Rightarrow i$ (**infixl** $\langle - \rangle$ 65) — set difference
where $A - B \equiv \{ x \in A \mid \neg(x \in B) \}$

definition *Un* :: $[i, i] \Rightarrow i$ (**infixl** $\langle \cup \rangle$ 65) — binary union
where $A \cup B \equiv \bigcup (U\text{pair}(A, B))$

definition *Int* :: $[i, i] \Rightarrow i$ (**infixl** $\langle \cap \rangle$ 70) — binary intersection
where $A \cap B \equiv \bigcap (U\text{pair}(A, B))$

definition *cons* :: $[i, i] \Rightarrow i$
where $\text{cons}(a, A) \equiv U\text{pair}(a, A) \cup A$

definition *succ* :: $i \Rightarrow i$
where $\text{succ}(i) \equiv \text{cons}(i, i)$

nonterminal *is*

syntax

$:: i \Rightarrow is$ ($\langle - \rangle$)
-Enum :: $[i, is] \Rightarrow is$ ($\langle -, / - \rangle$)
-Finset :: $is \Rightarrow i$ ($\langle (\langle \text{indent}=1 \text{ notation}=\langle \text{mixfix set enumeration} \rangle \{-\} \rangle) \rangle$)

translations

$\{x, xs\} == \text{CONST } \text{cons}(x, \{xs\})$
 $\{x\} == \text{CONST } \text{cons}(x, \emptyset)$

1.6 Axioms

axiomatization

where

extension: $A = B \longleftrightarrow A \subseteq B \wedge B \subseteq A$ **and**
Union-iff: $A \in \bigcup(C) \longleftrightarrow (\exists B \in C. A \in B)$ **and**
Pow-iff: $A \in \text{Pow}(B) \longleftrightarrow A \subseteq B$ **and**

infinity: $0 \in \text{Inf} \wedge (\forall y \in \text{Inf}. \text{succ}(y) \in \text{Inf})$ **and**

foundation: $A = \emptyset \vee (\exists x \in A. \forall y \in x. y \notin A)$ **and**

replacement: $(\forall x \in A. \forall y z. P(x, y) \wedge P(x, z) \longrightarrow y = z) \implies$
 $b \in \text{PrimReplace}(A, P) \longleftrightarrow (\exists x \in A. P(x, b))$

1.7 Definite descriptions – via Replace over the set "1"

definition *The* :: $(i \Rightarrow o) \Rightarrow i$ (**binder** $\langle \text{THE} \rangle$ 10)
where *the-def*: $\text{The}(P) \equiv \bigcup (\{y \mid x \in \{0\}, P(y)\})$

definition $If :: [o, i, i] \Rightarrow i$ ($\langle \langle notation = \langle mixfix\ if\ then\ else \rangle \rangle if\ (-) / then\ (-) / else\ (-) \rangle [10] 10 \rangle$)
where $if\text{-}def: if\ P\ then\ a\ else\ b \equiv THE\ z.\ P \wedge z=a \mid \neg P \wedge z=b$

abbreviation (*input*)
 $old\text{-}if :: [o, i, i] \Rightarrow i$ ($\langle if\ '(-,-,-)' \rangle$)
where $if(P,a,b) \equiv If(P,a,b)$

1.8 Ordered Pairing

definition $Pair :: [i, i] \Rightarrow i$
where $Pair(a,b) \equiv \{\{a,a\}, \{a,b\}\}$

definition $fst :: i \Rightarrow i$
where $fst(p) \equiv THE\ a.\ \exists b.\ p = Pair(a, b)$

definition $snd :: i \Rightarrow i$
where $snd(p) \equiv THE\ b.\ \exists a.\ p = Pair(a, b)$

definition $split :: [[i, i] \Rightarrow 'a, i] \Rightarrow 'a::\{\}$ — for pattern-matching
where $split(c) \equiv \lambda p.\ c(fst(p), snd(p))$

nonterminal *tuple-args*

syntax

$:: i \Rightarrow tuple\text{-}args\ (\langle \rightarrow \rangle)$
 $-Tuple\text{-}args :: [i, tuple\text{-}args] \Rightarrow tuple\text{-}args\ (\langle \rightarrow, / \rightarrow \rangle)$
 $-Tuple :: [i, tuple\text{-}args] \Rightarrow i$ ($\langle \langle indent = 1\ notation = \langle mixfix\ tuple\ enumeration \rangle \rangle \langle \rightarrow, / \rightarrow \rangle \rangle$)

translations

$\langle x, y, z \rangle == \langle x, \langle y, z \rangle \rangle$
 $\langle x, y \rangle == CONST\ Pair(x, y)$

nonterminal *patterns*

syntax

$-pattern :: patterns \Rightarrow pptrn$ ($\langle \langle open\text{-}block\ notation = \langle pattern\ tuple \rangle \rangle \langle \rightarrow \rangle \rangle$)
 $:: pptrn \Rightarrow patterns\ (\langle \rightarrow \rangle)$
 $-patterns :: [pptrn, patterns] \Rightarrow patterns\ (\langle \rightarrow, / \rightarrow \rangle)$

syntax-consts

$-pattern\ -patterns == split$

translations

$\lambda \langle x, y, zs \rangle. b == CONST\ split(\lambda x\ \langle y, zs \rangle. b)$
 $\lambda \langle x, y \rangle. b == CONST\ split(\lambda x\ y. b)$

definition $Sigma :: [i, i \Rightarrow i] \Rightarrow i$
where $Sigma(A,B) \equiv \bigcup_{x \in A} \bigcup_{y \in B(x)} \{\langle x, y \rangle\}$

abbreviation $cart\text{-}prod :: [i, i] \Rightarrow i$ (**infixr** $\langle \times \rangle$ 80) — Cartesian product

where $A \times B \equiv \text{Sigma}(A, \lambda\cdot. B)$

1.9 Relations and Functions

definition $\text{converse} :: i \Rightarrow i$
 where $\text{converse}(r) \equiv \{z. w \in r, \exists x y. w = \langle x, y \rangle \wedge z = \langle y, x \rangle\}$

definition $\text{domain} :: i \Rightarrow i$
 where $\text{domain}(r) \equiv \{x. w \in r, \exists y. w = \langle x, y \rangle\}$

definition $\text{range} :: i \Rightarrow i$
 where $\text{range}(r) \equiv \text{domain}(\text{converse}(r))$

definition $\text{field} :: i \Rightarrow i$
 where $\text{field}(r) \equiv \text{domain}(r) \cup \text{range}(r)$

definition $\text{relation} :: i \Rightarrow o$ — recognizes sets of pairs
 where $\text{relation}(r) \equiv \forall z \in r. \exists x y. z = \langle x, y \rangle$

definition $\text{function} :: i \Rightarrow o$ — recognizes functions; can have non-pairs
 where $\text{function}(r) \equiv \forall x y. \langle x, y \rangle \in r \longrightarrow (\forall y'. \langle x, y' \rangle \in r \longrightarrow y = y')$

definition $\text{Image} :: [i, i] \Rightarrow i$ (**infixl** $\langle \langle \rangle 90$) — image
 where $\text{image-def}: r \text{ “ } A \equiv \{y \in \text{range}(r). \exists x \in A. \langle x, y \rangle \in r\}$

definition $\text{vimage} :: [i, i] \Rightarrow i$ (**infixl** $\langle \langle \rangle 90$) — inverse image
 where $\text{vimage-def}: r \text{ -“ } A \equiv \text{converse}(r) \text{ “ } A$

definition $\text{restrict} :: [i, i] \Rightarrow i$
 where $\text{restrict}(r, A) \equiv \{z \in r. \exists x \in A. \exists y. z = \langle x, y \rangle\}$

definition $\text{Lambda} :: [i, i \Rightarrow i] \Rightarrow i$
 where $\text{lam-def}: \text{Lambda}(A, b) \equiv \{\langle x, b(x) \rangle. x \in A\}$

definition $\text{apply} :: [i, i] \Rightarrow i$ (**infixl** $\langle \langle \rangle 90$) — function application
 where $f'a \equiv \bigcup (f' \text{ “ } \{a\})$

definition $\text{Pi} :: [i, i \Rightarrow i] \Rightarrow i$
 where $\text{Pi}(A, B) \equiv \{f \in \text{Pow}(\text{Sigma}(A, B)). A \subseteq \text{domain}(f) \wedge \text{function}(f)\}$

abbreviation $\text{function-space} :: [i, i] \Rightarrow i$ (**infixr** $\langle \langle \rangle 60$) — function space
 where $A \rightarrow B \equiv \text{Pi}(A, \lambda\cdot. B)$

syntax

-PROD :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨mixfix Π ∈⟩⟩Π -∈-./ -)⟩ 10)

-SUM :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨mixfix ∑ ∈⟩⟩∑ -∈-./ -)⟩ 10)

-lam :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨mixfix λ ∈⟩⟩λ -∈-./ -)⟩ 10)

syntax-consts

-PROD == Pi and

-SUM == Sigma and

-lam == Lambda

translations

$\prod_{x \in A} B == \text{CONST } \text{Pi}(A, \lambda x. B)$

$\sum_{x \in A} B == \text{CONST } \text{Sigma}(A, \lambda x. B)$

$\lambda x \in A. f == \text{CONST } \text{Lambda}(A, \lambda x. f)$

1.10 ASCII syntax**notation (ASCII)**

cart-prod (infixr ⟨*⟩ 80) and

Int (infixl ⟨Int⟩ 70) and

Un (infixl ⟨Un⟩ 65) and

function-space (infixr ⟨->⟩ 60) and

Subset (infixl ⟨<=⟩ 50) and

mem (infixl ⟨:⟩ 50) and

not-mem (infixl ⟨¬:⟩ 50)

syntax (ASCII)

-Ball :: [pttrn, i, o] ⇒ o (⟨⟨indent=3 notation=⟨binder ALL:⟩⟩ALL -:-./ -)⟩ 10)

-Bex :: [pttrn, i, o] ⇒ o (⟨⟨indent=3 notation=⟨binder EX:⟩⟩EX -:-./ -)⟩ 10)

-Collect :: [pttrn, i, o] ⇒ i (⟨⟨indent=1 notation=⟨mixfix set comprehension⟩⟩{-: -./ -})⟩)

-Replace :: [pttrn, pttrn, i, o] ⇒ i (⟨⟨indent=1 notation=⟨mixfix relational replacement⟩⟩{-: -./ -})⟩)

-RepFun :: [i, pttrn, i] ⇒ i (⟨⟨indent=1 notation=⟨mixfix functional replacement⟩⟩{-: -./ -})⟩ [51,0,51])

-UNION :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨binder UN:⟩⟩UN -:-./ -)⟩ 10)

-INTER :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨binder INT:⟩⟩INT -:-./ -)⟩ 10)

-PROD :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨binder PROD:⟩⟩PROD -:-./ -)⟩ 10)

-SUM :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨binder SUM:⟩⟩SUM -:-./ -)⟩ 10)

-lam :: [pttrn, i, i] ⇒ i (⟨⟨indent=3 notation=⟨binder lam:⟩⟩lam -:-./ -)⟩ 10)

-*Tuple* :: [*i*, *tuple-args*] \Rightarrow *i* (\langle (\langle *indent=1 notation=* *mixfix tuple enumeration* \rangle \langle -, / - \rangle \rangle)
-*pattern* :: *patterns* \Rightarrow *pttrn* (\langle \langle - \rangle \rangle)

1.11 Substitution

lemma *subst-elim*: $\llbracket b \in A; a = b \rrbracket \Longrightarrow a \in A$
 \langle *proof* \rangle

1.12 Bounded universal quantifier

lemma *ballI* [*intro!*]: $\llbracket \bigwedge x. x \in A \Longrightarrow P(x) \rrbracket \Longrightarrow \forall x \in A. P(x)$
 \langle *proof* \rangle

lemmas *strip* = *impI allI ballI*

lemma *bspec* [*dest?*]: $\llbracket \forall x \in A. P(x); x: A \rrbracket \Longrightarrow P(x)$
 \langle *proof* \rangle

lemma *rev-ballE* [*elim*]:
 $\llbracket \forall x \in A. P(x); x \notin A \Longrightarrow Q; P(x) \Longrightarrow Q \rrbracket \Longrightarrow Q$
 \langle *proof* \rangle

lemma *ballE*: $\llbracket \forall x \in A. P(x); P(x) \Longrightarrow Q; x \notin A \Longrightarrow Q \rrbracket \Longrightarrow Q$
 \langle *proof* \rangle

lemma *rev-bspec*: $\llbracket x: A; \forall x \in A. P(x) \rrbracket \Longrightarrow P(x)$
 \langle *proof* \rangle

lemma *ball-triv* [*simp*]: $(\forall x \in A. P) \longleftrightarrow ((\exists x. x \in A) \longrightarrow P)$
 \langle *proof* \rangle

lemma *ball-cong* [*cong*]:
 $\llbracket A = A'; \bigwedge x. x \in A' \Longrightarrow P(x) \longleftrightarrow P'(x) \rrbracket \Longrightarrow (\forall x \in A. P(x)) \longleftrightarrow (\forall x \in A'. P'(x))$
 \langle *proof* \rangle

lemma *atomize-ball*:
 $(\bigwedge x. x \in A \Longrightarrow P(x)) \equiv \text{Trueprop } (\forall x \in A. P(x))$
 \langle *proof* \rangle

lemmas [*symmetric, rulify*] = *atomize-ball*
and [*symmetric, defn*] = *atomize-ball*

1.13 Bounded existential quantifier

lemma *bexI* [*intro*]: $\llbracket P(x); x: A \rrbracket \Longrightarrow \exists x \in A. P(x)$

$\langle proof \rangle$

lemma *rev-bexI*: $\llbracket x \in A; P(x) \rrbracket \Longrightarrow \exists x \in A. P(x)$
 $\langle proof \rangle$

lemma *bexCI*: $\llbracket \forall x \in A. \neg P(x) \Longrightarrow P(a); a: A \rrbracket \Longrightarrow \exists x \in A. P(x)$
 $\langle proof \rangle$

lemma *bexE* [*elim!*]: $\llbracket \exists x \in A. P(x); \bigwedge x. \llbracket x \in A; P(x) \rrbracket \Longrightarrow Q \rrbracket \Longrightarrow Q$
 $\langle proof \rangle$

lemma *bex-triv* [*simp*]: $(\exists x \in A. P) \longleftrightarrow ((\exists x. x \in A) \wedge P)$
 $\langle proof \rangle$

lemma *bex-cong* [*cong*]:
 $\llbracket A = A'; \bigwedge x. x \in A' \Longrightarrow P(x) \longleftrightarrow P'(x) \rrbracket$
 $\Longrightarrow (\exists x \in A. P(x)) \longleftrightarrow (\exists x \in A'. P'(x))$
 $\langle proof \rangle$

1.14 Rules for subsets

lemma *subsetI* [*intro!*]:
 $(\bigwedge x. x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
 $\langle proof \rangle$

lemma *subsetD* [*elim*]: $\llbracket A \subseteq B; c \in A \rrbracket \Longrightarrow c \in B$
 $\langle proof \rangle$

lemma *subsetCE* [*elim*]:
 $\llbracket A \subseteq B; c \notin A \Longrightarrow P; c \in B \Longrightarrow P \rrbracket \Longrightarrow P$
 $\langle proof \rangle$

lemma *rev-subsetD*: $\llbracket c \in A; A \subseteq B \rrbracket \Longrightarrow c \in B$
 $\langle proof \rangle$

lemma *contra-subsetD*: $\llbracket A \subseteq B; c \notin B \rrbracket \Longrightarrow c \notin A$
 $\langle proof \rangle$

lemma *rev-contra-subsetD*: $\llbracket c \notin B; A \subseteq B \rrbracket \Longrightarrow c \notin A$
 $\langle proof \rangle$

lemma *subset-refl* [*simp*]: $A \subseteq A$
 $\langle proof \rangle$

lemma *subset-trans*: $\llbracket A \subseteq B; B \subseteq C \rrbracket \implies A \subseteq C$
 $\langle proof \rangle$

lemma *subset-iff*:
 $A \subseteq B \longleftrightarrow (\forall x. x \in A \longrightarrow x \in B)$
 $\langle proof \rangle$

For calculations

declare *subsetD* [*trans*] *rev-subsetD* [*trans*] *subset-trans* [*trans*]

1.15 Rules for equality

lemma *equalityI* [*intro*]: $\llbracket A \subseteq B; B \subseteq A \rrbracket \implies A = B$
 $\langle proof \rangle$

lemma *equality-iffI*: $(\bigwedge x. x \in A \longleftrightarrow x \in B) \implies A = B$
 $\langle proof \rangle$

lemmas *equalityD1* = *extension* [*THEN iffD1*, *THEN conjunct1*]
lemmas *equalityD2* = *extension* [*THEN iffD1*, *THEN conjunct2*]

lemma *equalityE*: $\llbracket A = B; \llbracket A \subseteq B; B \subseteq A \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *equalityCE*:
 $\llbracket A = B; \llbracket c \in A; c \in B \rrbracket \implies P; \llbracket c \notin A; c \notin B \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *equality-iffD*:
 $A = B \implies (\bigwedge x. x \in A \longleftrightarrow x \in B)$
 $\langle proof \rangle$

1.16 Rules for Replace – the derived form of replacement

lemma *Replace-iff*:
 $b \in \{y. x \in A, P(x, y)\} \longleftrightarrow (\exists x \in A. P(x, b) \wedge (\forall y. P(x, y) \longrightarrow y = b))$
 $\langle proof \rangle$

lemma *ReplaceI* [*intro*]:
 $\llbracket P(x, b); x: A; \bigwedge y. P(x, y) \implies y = b \rrbracket \implies$
 $b \in \{y. x \in A, P(x, y)\}$
 $\langle proof \rangle$

lemma *ReplaceE*:

$$\begin{aligned} & \llbracket b \in \{y. x \in A, P(x,y)\}; \\ & \quad \bigwedge x. \llbracket x: A; P(x,b); \forall y. P(x,y) \longrightarrow y=b \rrbracket \implies R \\ & \rrbracket \implies R \\ & \langle proof \rangle \end{aligned}$$

lemma *ReplaceE2* [*elim!*]:

$$\begin{aligned} & \llbracket b \in \{y. x \in A, P(x,y)\}; \\ & \quad \bigwedge x. \llbracket x: A; P(x,b) \rrbracket \implies R \\ & \rrbracket \implies R \\ & \langle proof \rangle \end{aligned}$$

lemma *Replace-cong* [*cong*]:

$$\llbracket A=B; \bigwedge x y. x \in B \implies P(x,y) \longleftrightarrow Q(x,y) \rrbracket \implies \text{Replace}(A,P) = \text{Replace}(B,Q)$$

$$\langle proof \rangle$$

1.17 Rules for RepFun

lemma *RepFunI*: $a \in A \implies f(a) \in \{f(x). x \in A\}$

$$\langle proof \rangle$$

lemma *RepFun-eqI* [*intro*]: $\llbracket b=f(a); a \in A \rrbracket \implies b \in \{f(x). x \in A\}$

$$\langle proof \rangle$$

lemma *RepFunE* [*elim!*]:

$$\begin{aligned} & \llbracket b \in \{f(x). x \in A\}; \\ & \quad \bigwedge x. \llbracket x \in A; b=f(x) \rrbracket \implies P \rrbracket \implies \\ & P \\ & \langle proof \rangle \end{aligned}$$

lemma *RepFun-cong* [*cong*]:

$$\llbracket A=B; \bigwedge x. x \in B \implies f(x)=g(x) \rrbracket \implies \text{RepFun}(A,f) = \text{RepFun}(B,g)$$

$$\langle proof \rangle$$

lemma *RepFun-iff* [*simp*]: $b \in \{f(x). x \in A\} \longleftrightarrow (\exists x \in A. b=f(x))$

$$\langle proof \rangle$$

lemma *triv-RepFun* [*simp*]: $\{x. x \in A\} = A$

$$\langle proof \rangle$$

1.18 Rules for Collect – forming a subset by separation

lemma *separation* [*simp*]: $a \in \{x \in A. P(x)\} \longleftrightarrow a \in A \wedge P(a)$

$$\langle proof \rangle$$

lemma *CollectI* [*intro!*]: $\llbracket a \in A; P(a) \rrbracket \implies a \in \{x \in A. P(x)\}$

$$\langle proof \rangle$$

lemma *CollectE* [*elim!*]: $\llbracket a \in \{x \in A. P(x)\}; \llbracket a \in A; P(a) \rrbracket \implies R \rrbracket \implies R$

$\langle proof \rangle$

lemma *CollectD1*: $a \in \{x \in A. P(x)\} \implies a \in A$ **and** *CollectD2*: $a \in \{x \in A. P(x)\} \implies P(a)$
 $\langle proof \rangle$

lemma *Collect-cong* [*cong*]:
 $\llbracket A=B; \bigwedge x. x \in B \implies P(x) \longleftrightarrow Q(x) \rrbracket$
 $\implies \text{Collect}(A, \lambda x. P(x)) = \text{Collect}(B, \lambda x. Q(x))$
 $\langle proof \rangle$

1.19 Rules for Unions

declare *Union-iff* [*simp*]

lemma *UnionI* [*intro*]: $\llbracket B: C; A: B \rrbracket \implies A: \bigcup(C)$
 $\langle proof \rangle$

lemma *UnionE* [*elim!*]: $\llbracket A \in \bigcup(C); \bigwedge B. \llbracket A: B; B: C \rrbracket \implies R \rrbracket \implies R$
 $\langle proof \rangle$

1.20 Rules for Unions of families

lemma *UN-iff* [*simp*]: $b \in (\bigcup x \in A. B(x)) \longleftrightarrow (\exists x \in A. b \in B(x))$
 $\langle proof \rangle$

lemma *UN-I*: $\llbracket a: A; b: B(a) \rrbracket \implies b: (\bigcup x \in A. B(x))$
 $\langle proof \rangle$

lemma *UN-E* [*elim!*]:
 $\llbracket b \in (\bigcup x \in A. B(x)); \bigwedge x. \llbracket x: A; b: B(x) \rrbracket \implies R \rrbracket \implies R$
 $\langle proof \rangle$

lemma *UN-cong*:
 $\llbracket A=B; \bigwedge x. x \in B \implies C(x)=D(x) \rrbracket \implies (\bigcup x \in A. C(x)) = (\bigcup x \in B. D(x))$
 $\langle proof \rangle$

1.21 Rules for the empty set

lemma *not-mem-empty* [*simp*]: $a \notin 0$
 $\langle proof \rangle$

lemmas *emptyE* [*elim!*] = *not-mem-empty* [*THEN notE*]

lemma *empty-subsetI* [*simp*]: $0 \subseteq A$
 $\langle proof \rangle$

lemma *equals0I*: $\llbracket \bigwedge y. y \in A \implies \text{False} \rrbracket \implies A = 0$
 $\langle \text{proof} \rangle$

lemma *equals0D* [*dest*]: $A = 0 \implies a \notin A$
 $\langle \text{proof} \rangle$

declare *sym* [*THEN equals0D, dest*]

lemma *not-emptyI*: $a \in A \implies A \neq 0$
 $\langle \text{proof} \rangle$

lemma *not-emptyE*: $\llbracket A \neq 0; \bigwedge x. x \in A \implies R \rrbracket \implies R$
 $\langle \text{proof} \rangle$

1.22 Rules for Inter

lemma *Inter-iff*: $A \in \bigcap (C) \longleftrightarrow (\forall x \in C. A : x) \wedge C \neq 0$
 $\langle \text{proof} \rangle$

lemma *InterI* [*intro!*]:
 $\llbracket \bigwedge x. x : C \implies A : x; C \neq 0 \rrbracket \implies A \in \bigcap (C)$
 $\langle \text{proof} \rangle$

lemma *InterD* [*elim, Pure.elim*]: $\llbracket A \in \bigcap (C); B \in C \rrbracket \implies A \in B$
 $\langle \text{proof} \rangle$

lemma *InterE* [*elim*]:
 $\llbracket A \in \bigcap (C); B \notin C \implies R; A \in B \implies R \rrbracket \implies R$
 $\langle \text{proof} \rangle$

1.23 Rules for Intersections of families

lemma *INT-iff*: $b \in (\bigcap x \in A. B(x)) \longleftrightarrow (\forall x \in A. b \in B(x)) \wedge A \neq 0$
 $\langle \text{proof} \rangle$

lemma *INT-I*: $\llbracket \bigwedge x. x : A \implies b : B(x); A \neq 0 \rrbracket \implies b : (\bigcap x \in A. B(x))$
 $\langle \text{proof} \rangle$

lemma *INT-E*: $\llbracket b \in (\bigcap x \in A. B(x)); a : A \rrbracket \implies b \in B(a)$
 $\langle \text{proof} \rangle$

lemma *INT-cong*:
 $\llbracket A = B; \bigwedge x. x \in B \implies C(x) = D(x) \rrbracket \implies (\bigcap x \in A. C(x)) = (\bigcap x \in B. D(x))$
 $\langle \text{proof} \rangle$

1.24 Rules for Powersets

lemma *PowI*: $A \subseteq B \implies A \in \text{Pow}(B)$
<proof>

lemma *PowD*: $A \in \text{Pow}(B) \implies A \subseteq B$
<proof>

declare *Pow-iff* [*iff*]

lemmas *Pow-bottom* = *empty-subsetI* [*THEN PowI*] — $0 \in \text{Pow}(B)$

lemmas *Pow-top* = *subset-refl* [*THEN PowI*] — $A \in \text{Pow}(A)$

1.25 Cantor's Theorem: There is no surjection from a set to its powerset.

lemma *cantor*: $\exists S \in \text{Pow}(A). \forall x \in A. b(x) \neq S$
<proof>

end

2 Unordered Pairs

theory *upair*
imports *ZF-Base*
keywords *print-tcset* :: *diag*
begin

<ML>

2.1 Unordered Pairs: constant *Upair*

lemma *Upair-iff* [*simp*]: $c \in \text{Upair}(a,b) \longleftrightarrow (c=a \mid c=b)$
<proof>

lemma *UpairI1*: $a \in \text{Upair}(a,b)$
<proof>

lemma *UpairI2*: $b \in \text{Upair}(a,b)$
<proof>

lemma *UpairE*: $\llbracket a \in \text{Upair}(b,c); a=b \implies P; a=c \implies P \rrbracket \implies P$
<proof>

2.2 Rules for Binary Union, Defined via *Upair*

lemma *Un-iff* [*simp*]: $c \in A \cup B \longleftrightarrow (c \in A \mid c \in B)$
<proof>

lemma *UnI1*: $c \in A \implies c \in A \cup B$
 $\langle proof \rangle$

lemma *UnI2*: $c \in B \implies c \in A \cup B$
 $\langle proof \rangle$

declare *UnI1* [*elim?*] *UnI2* [*elim?*]

lemma *UnE* [*elim!*]: $\llbracket c \in A \cup B; c \in A \implies P; c \in B \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *UnE'*: $\llbracket c \in A \cup B; c \in A \implies P; \llbracket c \in B; c \notin A \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *UnCI* [*intro!*]: $(c \notin B \implies c \in A) \implies c \in A \cup B$
 $\langle proof \rangle$

2.3 Rules for Binary Intersection, Defined via *Upair*

lemma *Int-iff* [*simp*]: $c \in A \cap B \longleftrightarrow (c \in A \wedge c \in B)$
 $\langle proof \rangle$

lemma *IntI* [*intro!*]: $\llbracket c \in A; c \in B \rrbracket \implies c \in A \cap B$
 $\langle proof \rangle$

lemma *IntD1*: $c \in A \cap B \implies c \in A$
 $\langle proof \rangle$

lemma *IntD2*: $c \in A \cap B \implies c \in B$
 $\langle proof \rangle$

lemma *IntE* [*elim!*]: $\llbracket c \in A \cap B; \llbracket c \in A; c \in B \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

2.4 Rules for Set Difference, Defined via *Upair*

lemma *Diff-iff* [*simp*]: $c \in A - B \longleftrightarrow (c \in A \wedge c \notin B)$
 $\langle proof \rangle$

lemma *DiffI* [*intro!*]: $\llbracket c \in A; c \notin B \rrbracket \implies c \in A - B$
 $\langle proof \rangle$

lemma *DiffD1*: $c \in A - B \implies c \in A$
 $\langle proof \rangle$

lemma *DiffD2*: $c \in A - B \implies c \notin B$
 $\langle proof \rangle$

lemma *DiffE* [*elim!*]: $\llbracket c \in A - B; \llbracket c \in A; c \notin B \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

2.5 Rules for *cons*

lemma *cons-iff* [*simp*]: $a \in cons(b, A) \longleftrightarrow (a=b \mid a \in A)$
 $\langle proof \rangle$

lemma *consI1* [*simp, TC*]: $a \in cons(a, B)$
 $\langle proof \rangle$

lemma *consI2*: $a \in B \implies a \in cons(b, B)$
 $\langle proof \rangle$

lemma *consE* [*elim!*]: $\llbracket a \in cons(b, A); a=b \implies P; a \in A \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *consE'*:
 $\llbracket a \in cons(b, A); a=b \implies P; \llbracket a \in A; a \neq b \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *consCI* [*intro!*]: $(a \notin B \implies a=b) \implies a \in cons(b, B)$
 $\langle proof \rangle$

lemma *cons-not-0* [*simp*]: $cons(a, B) \neq 0$
 $\langle proof \rangle$

lemmas *cons-neq-0* = *cons-not-0* [*THEN notE*]

declare *cons-not-0* [*THEN not-sym, simp*]

2.6 Singletons

lemma *singleton-iff*: $a \in \{b\} \longleftrightarrow a=b$
 $\langle proof \rangle$

lemma *singletonI* [*intro!*]: $a \in \{a\}$
 $\langle proof \rangle$

lemmas *singletonE* = *singleton-iff* [*THEN iffD1, elim-format, elim!*]

2.7 Descriptions

lemma *the-equality* [*intro*]:
 $\llbracket P(a); \bigwedge x. P(x) \implies x=a \rrbracket \implies (THE x. P(x)) = a$
 $\langle proof \rangle$

lemma *the-equality2*: $\llbracket \exists !x. P(x); P(a) \rrbracket \Longrightarrow (THE\ x. P(x)) = a$
 $\langle proof \rangle$

lemma *theI*: $\exists !x. P(x) \Longrightarrow P(THE\ x. P(x))$
 $\langle proof \rangle$

lemma *the-0*: $\neg (\exists !x. P(x)) \Longrightarrow (THE\ x. P(x)) = 0$
 $\langle proof \rangle$

lemma *theI2*:
 assumes *p1*: $\neg Q(0) \Longrightarrow \exists !x. P(x)$
 and *p2*: $\bigwedge x. P(x) \Longrightarrow Q(x)$
 shows $Q(THE\ x. P(x))$
 $\langle proof \rangle$

lemma *the-eq-trivial* [*simp*]: $(THE\ x. x = a) = a$
 $\langle proof \rangle$

lemma *the-eq-trivial2* [*simp*]: $(THE\ x. a = x) = a$
 $\langle proof \rangle$

2.8 Conditional Terms: *if-then-else*

lemma *if-true* [*simp*]: $(if\ True\ then\ a\ else\ b) = a$
 $\langle proof \rangle$

lemma *if-false* [*simp*]: $(if\ False\ then\ a\ else\ b) = b$
 $\langle proof \rangle$

lemma *if-cong*:
 $\llbracket P \longleftrightarrow Q; Q \Longrightarrow a=c; \neg Q \Longrightarrow b=d \rrbracket$
 $\Longrightarrow (if\ P\ then\ a\ else\ b) = (if\ Q\ then\ c\ else\ d)$
 $\langle proof \rangle$

lemma *if-weak-cong*: $P \longleftrightarrow Q \Longrightarrow (if\ P\ then\ x\ else\ y) = (if\ Q\ then\ x\ else\ y)$
 $\langle proof \rangle$

lemma *if-P*: $P \Longrightarrow (if\ P\ then\ a\ else\ b) = a$
 $\langle proof \rangle$

lemma *if-not-P*: $\neg P \implies (\text{if } P \text{ then } a \text{ else } b) = b$
 $\langle \text{proof} \rangle$

lemma *split-if* [*split*]:
 $P(\text{if } Q \text{ then } x \text{ else } y) \longleftrightarrow ((Q \longrightarrow P(x)) \wedge (\neg Q \longrightarrow P(y)))$
 $\langle \text{proof} \rangle$

lemmas *split-if-eq1* = *split-if* [*of* $\lambda x. x = b$] **for** *b*
lemmas *split-if-eq2* = *split-if* [*of* $\lambda x. a = x$] **for** *a*

lemmas *split-if-mem1* = *split-if* [*of* $\lambda x. x \in b$] **for** *b*
lemmas *split-if-mem2* = *split-if* [*of* $\lambda x. a \in x$] **for** *a*

lemmas *split-ifs* = *split-if-eq1* *split-if-eq2* *split-if-mem1* *split-if-mem2*

lemma *if-iff*: $a: (\text{if } P \text{ then } x \text{ else } y) \longleftrightarrow P \wedge a \in x \mid \neg P \wedge a \in y$
 $\langle \text{proof} \rangle$

lemma *if-type* [*TC*]:
 $\llbracket P \implies a \in A; \neg P \implies b \in A \rrbracket \implies (\text{if } P \text{ then } a \text{ else } b): A$
 $\langle \text{proof} \rangle$

lemma *split-if-asm*: $P(\text{if } Q \text{ then } x \text{ else } y) \longleftrightarrow (\neg((Q \wedge \neg P(x)) \mid (\neg Q \wedge \neg P(y))))$
 $\langle \text{proof} \rangle$

lemmas *if-splits* = *split-if* *split-if-asm*

2.9 Consequences of Foundation

lemma *mem-asym*: $\llbracket a \in b; \neg P \implies b \in a \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *mem-irrefl*: $a \in a \implies P$
 $\langle \text{proof} \rangle$

lemma *mem-not-refl*: $a \notin a$
 $\langle \text{proof} \rangle$

lemma *mem-imp-not-eq*: $a \in A \implies a \neq A$

$\langle proof \rangle$

lemma *eq-imp-not-mem*: $a=A \implies a \notin A$
 $\langle proof \rangle$

2.10 Rules for Successor

lemma *succ-iff*: $i \in succ(j) \longleftrightarrow i=j \mid i \in j$
 $\langle proof \rangle$

lemma *succI1* [*simp*]: $i \in succ(i)$
 $\langle proof \rangle$

lemma *succI2*: $i \in j \implies i \in succ(j)$
 $\langle proof \rangle$

lemma *succE* [*elim!*]:
 $\llbracket i \in succ(j); i=j \implies P; i \in j \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *succCI* [*intro!*]: $(i \notin j \implies i=j) \implies i \in succ(j)$
 $\langle proof \rangle$

lemma *succ-not-0* [*simp*]: $succ(n) \neq 0$
 $\langle proof \rangle$

lemmas *succ-neq-0* = *succ-not-0* [*THEN notE, elim!*]

declare *succ-not-0* [*THEN not-sym, simp*]
declare *sym* [*THEN succ-neq-0, elim!*]

lemmas *succ-subsetD* = *succI1* [*THEN* [2] *subsetD*]

lemmas *succ-neq-self* = *succI1* [*THEN mem-imp-not-eq, THEN not-sym*]

lemma *succ-inject-iff* [*simp*]: $succ(m) = succ(n) \longleftrightarrow m=n$
 $\langle proof \rangle$

lemmas *succ-inject* = *succ-inject-iff* [*THEN iffD1, dest!*]

2.11 Miniscoping of the Bounded Universal Quantifier

lemma *ball-simps1*:
 $(\forall x \in A. P(x) \wedge Q) \longleftrightarrow (\forall x \in A. P(x)) \wedge (A=0 \mid Q)$
 $(\forall x \in A. P(x) \mid Q) \longleftrightarrow ((\forall x \in A. P(x)) \mid Q)$
 $(\forall x \in A. P(x) \longrightarrow Q) \longleftrightarrow ((\exists x \in A. P(x)) \longrightarrow Q)$
 $(\neg(\forall x \in A. P(x))) \longleftrightarrow (\exists x \in A. \neg P(x))$

$(\forall x \in 0. P(x)) \longleftrightarrow \text{True}$
 $(\forall x \in \text{succ}(i). P(x)) \longleftrightarrow P(i) \wedge (\forall x \in i. P(x))$
 $(\forall x \in \text{cons}(a, B). P(x)) \longleftrightarrow P(a) \wedge (\forall x \in B. P(x))$
 $(\forall x \in \text{RepFun}(A, f). P(x)) \longleftrightarrow (\forall y \in A. P(f(y)))$
 $(\forall x \in \bigcup (A). P(x)) \longleftrightarrow (\forall y \in A. \forall x \in y. P(x))$
 $\langle \text{proof} \rangle$

lemma *ball-simps2*:

$(\forall x \in A. P \wedge Q(x)) \longleftrightarrow (A=0 \mid P) \wedge (\forall x \in A. Q(x))$
 $(\forall x \in A. P \mid Q(x)) \longleftrightarrow (P \mid (\forall x \in A. Q(x)))$
 $(\forall x \in A. P \longrightarrow Q(x)) \longleftrightarrow (P \longrightarrow (\forall x \in A. Q(x)))$
 $\langle \text{proof} \rangle$

lemma *ball-simps3*:

$(\forall x \in \text{Collect}(A, Q). P(x)) \longleftrightarrow (\forall x \in A. Q(x) \longrightarrow P(x))$
 $\langle \text{proof} \rangle$

lemmas *ball-simps* [simp] = *ball-simps1 ball-simps2 ball-simps3*

lemma *ball-conj-distrib*:

$(\forall x \in A. P(x) \wedge Q(x)) \longleftrightarrow ((\forall x \in A. P(x)) \wedge (\forall x \in A. Q(x)))$
 $\langle \text{proof} \rangle$

2.12 Miniscoping of the Bounded Existential Quantifier

lemma *bex-simps1*:

$(\exists x \in A. P(x) \wedge Q) \longleftrightarrow ((\exists x \in A. P(x)) \wedge Q)$
 $(\exists x \in A. P(x) \mid Q) \longleftrightarrow (\exists x \in A. P(x)) \mid (A \neq 0 \wedge Q)$
 $(\exists x \in A. P(x) \longrightarrow Q) \longleftrightarrow ((\forall x \in A. P(x)) \longrightarrow (A \neq 0 \wedge Q))$
 $(\exists x \in 0. P(x)) \longleftrightarrow \text{False}$
 $(\exists x \in \text{succ}(i). P(x)) \longleftrightarrow P(i) \mid (\exists x \in i. P(x))$
 $(\exists x \in \text{cons}(a, B). P(x)) \longleftrightarrow P(a) \mid (\exists x \in B. P(x))$
 $(\exists x \in \text{RepFun}(A, f). P(x)) \longleftrightarrow (\exists y \in A. P(f(y)))$
 $(\exists x \in \bigcup (A). P(x)) \longleftrightarrow (\exists y \in A. \exists x \in y. P(x))$
 $(\neg(\exists x \in A. P(x))) \longleftrightarrow (\forall x \in A. \neg P(x))$
 $\langle \text{proof} \rangle$

lemma *bex-simps2*:

$(\exists x \in A. P \wedge Q(x)) \longleftrightarrow (P \wedge (\exists x \in A. Q(x)))$
 $(\exists x \in A. P \mid Q(x)) \longleftrightarrow (A \neq 0 \wedge P) \mid (\exists x \in A. Q(x))$
 $(\exists x \in A. P \longrightarrow Q(x)) \longleftrightarrow ((A=0 \mid P) \longrightarrow (\exists x \in A. Q(x)))$
 $\langle \text{proof} \rangle$

lemma *bex-simps3*:

$(\exists x \in \text{Collect}(A, Q). P(x)) \longleftrightarrow (\exists x \in A. Q(x) \wedge P(x))$
 $\langle \text{proof} \rangle$

lemmas *bex-simps* [simp] = *bex-simps1 bex-simps2 bex-simps3*

lemma *bex-disj-distrib*:

$$(\exists x \in A. P(x) \mid Q(x)) \longleftrightarrow ((\exists x \in A. P(x)) \mid (\exists x \in A. Q(x)))$$

<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>

2.13 Miniscoping of the Replacement Operator

These cover both *Replace* and *Collect*

lemma *Rep-simps* [*simp*]:

$$\begin{aligned} \{x. y \in 0, R(x,y)\} &= 0 \\ \{x \in 0. P(x)\} &= 0 \\ \{x \in A. Q\} &= (\text{if } Q \text{ then } A \text{ else } 0) \\ \text{RepFun}(0, f) &= 0 \\ \text{RepFun}(\text{succ}(i), f) &= \text{cons}(f(i), \text{RepFun}(i, f)) \\ \text{RepFun}(\text{cons}(a, B), f) &= \text{cons}(f(a), \text{RepFun}(B, f)) \end{aligned}$$

<proof>

2.14 Miniscoping of Unions

lemma *UN-simps1*:

$$\begin{aligned} (\bigcup x \in C. \text{cons}(a, B(x))) &= (\text{if } C=0 \text{ then } 0 \text{ else } \text{cons}(a, \bigcup x \in C. B(x))) \\ (\bigcup x \in C. A(x) \cup B') &= (\text{if } C=0 \text{ then } 0 \text{ else } (\bigcup x \in C. A(x)) \cup B') \\ (\bigcup x \in C. A' \cup B(x)) &= (\text{if } C=0 \text{ then } 0 \text{ else } A' \cup (\bigcup x \in C. B(x))) \\ (\bigcup x \in C. A(x) \cap B') &= ((\bigcup x \in C. A(x)) \cap B') \\ (\bigcup x \in C. A' \cap B(x)) &= (A' \cap (\bigcup x \in C. B(x))) \\ (\bigcup x \in C. A(x) - B') &= ((\bigcup x \in C. A(x)) - B') \\ (\bigcup x \in C. A' - B(x)) &= (\text{if } C=0 \text{ then } 0 \text{ else } A' - (\bigcap x \in C. B(x))) \end{aligned}$$

<proof>

lemma *UN-simps2*:

$$\begin{aligned} (\bigcup x \in \bigcup (A). B(x)) &= (\bigcup y \in A. \bigcup x \in y. B(x)) \\ (\bigcup z \in (\bigcup x \in A. B(x)). C(z)) &= (\bigcup x \in A. \bigcup z \in B(x). C(z)) \\ (\bigcup x \in \text{RepFun}(A, f). B(x)) &= (\bigcup a \in A. B(f(a))) \end{aligned}$$

$\langle \text{proof} \rangle$

lemmas *UN-simps [simp] = UN-simps1 UN-simps2*

Opposite of miniscoping: pull the operator out

lemma *UN-extend-simps1*:

$$\begin{aligned} (\bigcup x \in C. A(x)) \cup B &= (\text{if } C=0 \text{ then } B \text{ else } (\bigcup x \in C. A(x) \cup B)) \\ ((\bigcup x \in C. A(x)) \cap B) &= (\bigcup x \in C. A(x) \cap B) \\ ((\bigcup x \in C. A(x)) - B) &= (\bigcup x \in C. A(x) - B) \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *UN-extend-simps2*:

$$\begin{aligned} \text{cons}(a, \bigcup x \in C. B(x)) &= (\text{if } C=0 \text{ then } \{a\} \text{ else } (\bigcup x \in C. \text{cons}(a, B(x)))) \\ A \cup (\bigcup x \in C. B(x)) &= (\text{if } C=0 \text{ then } A \text{ else } (\bigcup x \in C. A \cup B(x))) \\ (A \cap (\bigcup x \in C. B(x))) &= (\bigcup x \in C. A \cap B(x)) \\ A - (\bigcup x \in C. B(x)) &= (\text{if } C=0 \text{ then } A \text{ else } (\bigcup x \in C. A - B(x))) \\ (\bigcup y \in A. \bigcup x \in y. B(x)) &= (\bigcup x \in \bigcup (A). B(x)) \\ (\bigcup a \in A. B(f(a))) &= (\bigcup x \in \text{RepFun}(A, f). B(x)) \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *UN-UN-extend*:

$$(\bigcup x \in A. \bigcup z \in B(x). C(z)) = (\bigcup z \in (\bigcup x \in A. B(x)). C(z))$$

$\langle \text{proof} \rangle$

lemmas *UN-extend-simps = UN-extend-simps1 UN-extend-simps2 UN-UN-extend*

2.15 Miniscoping of Intersections

lemma *INT-simps1*:

$$\begin{aligned} (\bigcap x \in C. A(x) \cap B) &= (\bigcap x \in C. A(x)) \cap B \\ (\bigcap x \in C. A(x) - B) &= (\bigcap x \in C. A(x)) - B \\ (\bigcap x \in C. A(x) \cup B) &= (\text{if } C=0 \text{ then } 0 \text{ else } (\bigcap x \in C. A(x)) \cup B) \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *INT-simps2*:

$$\begin{aligned} (\bigcap x \in C. A \cap B(x)) &= A \cap (\bigcap x \in C. B(x)) \\ (\bigcap x \in C. A - B(x)) &= (\text{if } C=0 \text{ then } 0 \text{ else } A - (\bigcup x \in C. B(x))) \\ (\bigcap x \in C. \text{cons}(a, B(x))) &= (\text{if } C=0 \text{ then } 0 \text{ else } \text{cons}(a, \bigcap x \in C. B(x))) \\ (\bigcap x \in C. A \cup B(x)) &= (\text{if } C=0 \text{ then } 0 \text{ else } A \cup (\bigcap x \in C. B(x))) \end{aligned}$$

$\langle \text{proof} \rangle$

lemmas *INT-simps [simp] = INT-simps1 INT-simps2*

Opposite of miniscoping: pull the operator out

lemma *INT-extend-simps1*:

$(\bigcap_{x \in C}. A(x)) \cap B = (\bigcap_{x \in C}. A(x) \cap B)$
 $(\bigcap_{x \in C}. A(x)) - B = (\bigcap_{x \in C}. A(x) - B)$
 $(\bigcap_{x \in C}. A(x)) \cup B = (\text{if } C=0 \text{ then } B \text{ else } (\bigcap_{x \in C}. A(x) \cup B))$
 $\langle \text{proof} \rangle$

lemma *INT-extend-simps2*:

$A \cap (\bigcap_{x \in C}. B(x)) = (\bigcap_{x \in C}. A \cap B(x))$
 $A - (\bigcup_{x \in C}. B(x)) = (\text{if } C=0 \text{ then } A \text{ else } (\bigcap_{x \in C}. A - B(x)))$
 $\text{cons}(a, \bigcap_{x \in C}. B(x)) = (\text{if } C=0 \text{ then } \{a\} \text{ else } (\bigcap_{x \in C}. \text{cons}(a, B(x))))$
 $A \cup (\bigcap_{x \in C}. B(x)) = (\text{if } C=0 \text{ then } A \text{ else } (\bigcap_{x \in C}. A \cup B(x)))$
 $\langle \text{proof} \rangle$

lemmas *INT-extend-simps* = *INT-extend-simps1* *INT-extend-simps2*

2.16 Other simprules

lemma *misc-simps* [*simp*]:

$0 \cup A = A$
 $A \cup 0 = A$
 $0 \cap A = 0$
 $A \cap 0 = 0$
 $0 - A = 0$
 $A - 0 = A$
 $\bigcup(0) = 0$
 $\bigcup(\text{cons}(b, A)) = b \cup \bigcup(A)$
 $\bigcap(\{b\}) = b$
 $\langle \text{proof} \rangle$

end

3 Ordered Pairs

theory *pair* **imports** *upair*
begin

$\langle ML \rangle$

lemma *singleton-eq-iff* [*iff*]: $\{a\} = \{b\} \longleftrightarrow a=b$
 $\langle \text{proof} \rangle$

lemma *doubleton-eq-iff*: $\{a, b\} = \{c, d\} \longleftrightarrow (a=c \wedge b=d) \mid (a=d \wedge b=c)$
 $\langle \text{proof} \rangle$

lemma *Pair-iff* [*simp*]: $\langle a, b \rangle = \langle c, d \rangle \longleftrightarrow a=c \wedge b=d$
 $\langle \text{proof} \rangle$

lemmas *Pair-inject* = *Pair-iff* [*THEN iffD1*, *THEN conjE*, *elim!*]

lemmas *Pair-inject1* = *Pair-iff* [*THEN iffD1*, *THEN conjunct1*]

lemmas *Pair-inject2* = *Pair-iff* [*THEN iffD1*, *THEN conjunct2*]

lemma *Pair-not-0*: $\langle a, b \rangle \neq 0$
 $\langle \text{proof} \rangle$

lemmas *Pair-neq-0* = *Pair-not-0* [*THEN notE*, *elim!*]

declare *sym* [*THEN Pair-neq-0*, *elim!*]

lemma *Pair-neq-fst*: $\langle a, b \rangle = a \implies P$
 $\langle \text{proof} \rangle$

lemma *Pair-neq-snd*: $\langle a, b \rangle = b \implies P$
 $\langle \text{proof} \rangle$

3.1 Sigma: Disjoint Union of a Family of Sets

Generalizes Cartesian product

lemma *Sigma-iff* [*simp*]: $\langle a, b \rangle : \text{Sigma}(A, B) \longleftrightarrow a \in A \wedge b \in B(a)$
 $\langle \text{proof} \rangle$

lemma *SigmaI* [*TC, intro!*]: $\llbracket a \in A; b \in B(a) \rrbracket \implies \langle a, b \rangle \in \text{Sigma}(A, B)$
 $\langle \text{proof} \rangle$

lemmas *SigmaD1* = *Sigma-iff* [*THEN iffD1*, *THEN conjunct1*]

lemmas *SigmaD2* = *Sigma-iff* [*THEN iffD1*, *THEN conjunct2*]

lemma *SigmaE* [*elim!*]:
 $\llbracket c \in \text{Sigma}(A, B);$
 $\bigwedge x y. \llbracket x \in A; y \in B(x); c = \langle x, y \rangle \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *SigmaE2* [*elim!*]:
 $\llbracket \langle a, b \rangle \in \text{Sigma}(A, B);$
 $\llbracket a \in A; b \in B(a) \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *Sigma-cong*:
 $\llbracket A = A'; \bigwedge x. x \in A' \implies B(x) = B'(x) \rrbracket \implies$
 $\text{Sigma}(A, B) = \text{Sigma}(A', B')$
 $\langle \text{proof} \rangle$

lemma *Sigma-empty1* [simp]: $\text{Sigma}(0, B) = 0$
 $\langle \text{proof} \rangle$

lemma *Sigma-empty2* [simp]: $A * 0 = 0$
 $\langle \text{proof} \rangle$

lemma *Sigma-empty-iff*: $A * B = 0 \longleftrightarrow A = 0 \mid B = 0$
 $\langle \text{proof} \rangle$

3.2 Projections *fst* and *snd*

lemma *fst-conv* [simp]: $\text{fst}(\langle a, b \rangle) = a$
 $\langle \text{proof} \rangle$

lemma *snd-conv* [simp]: $\text{snd}(\langle a, b \rangle) = b$
 $\langle \text{proof} \rangle$

lemma *fst-type* [TC]: $p \in \text{Sigma}(A, B) \implies \text{fst}(p) \in A$
 $\langle \text{proof} \rangle$

lemma *snd-type* [TC]: $p \in \text{Sigma}(A, B) \implies \text{snd}(p) \in B(\text{fst}(p))$
 $\langle \text{proof} \rangle$

lemma *Pair-fst-snd-eq*: $a \in \text{Sigma}(A, B) \implies \langle \text{fst}(a), \text{snd}(a) \rangle = a$
 $\langle \text{proof} \rangle$

3.3 The Eliminator, *split*

lemma *split* [simp]: $\text{split}(\lambda x y. c(x, y), \langle a, b \rangle) \equiv c(a, b)$
 $\langle \text{proof} \rangle$

lemma *split-type* [TC]:
 $\llbracket p \in \text{Sigma}(A, B);$
 $\quad \bigwedge x y. \llbracket x \in A; y \in B(x) \rrbracket \implies c(x, y): C(\langle x, y \rangle)$
 $\rrbracket \implies \text{split}(\lambda x y. c(x, y), p) \in C(p)$
 $\langle \text{proof} \rangle$

lemma *expand-split*:
 $u \in A * B \implies$
 $\quad R(\text{split}(c, u)) \longleftrightarrow (\forall x \in A. \forall y \in B. u = \langle x, y \rangle \longrightarrow R(c(x, y)))$
 $\langle \text{proof} \rangle$

3.4 A version of *split* for Formulae: Result Type *o*

lemma *splitI*: $R(a, b) \implies \text{split}(R, \langle a, b \rangle)$
 $\langle \text{proof} \rangle$

lemma *splitE*:
 $\llbracket \text{split}(R, z); z \in \text{Sigma}(A, B);$

$$\llbracket \bigwedge x y. \llbracket z = \langle x, y \rangle; R(x, y) \rrbracket \implies P \rrbracket \implies P$$

 $\langle proof \rangle$

lemma *splitD*: $split(R, \langle a, b \rangle) \implies R(a, b)$
 $\langle proof \rangle$

Complex rules for Sigma.

lemma *split-paired-Bex-Sigma* [simp]:
 $(\exists z \in Sigma(A, B). P(z)) \longleftrightarrow (\exists x \in A. \exists y \in B(x). P(\langle x, y \rangle))$
 $\langle proof \rangle$

lemma *split-paired-Ball-Sigma* [simp]:
 $(\forall z \in Sigma(A, B). P(z)) \longleftrightarrow (\forall x \in A. \forall y \in B(x). P(\langle x, y \rangle))$
 $\langle proof \rangle$

end

4 Basic Equalities and Inclusions

theory *equalities* **imports** *pair* **begin**

These cover union, intersection, converse, domain, range, etc. Philippe de Groote proved many of the inclusions.

lemma *in-mono*: $A \subseteq B \implies x \in A \longrightarrow x \in B$
 $\langle proof \rangle$

lemma *the-eq-0* [simp]: $(THE x. False) = 0$
 $\langle proof \rangle$

4.1 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*: $(\forall x \in A \cup B. P(x)) \longleftrightarrow (\forall x \in A. P(x)) \wedge (\forall x \in B. P(x))$
 $\langle proof \rangle$

lemma *bex-Un*: $(\exists x \in A \cup B. P(x)) \longleftrightarrow (\exists x \in A. P(x)) \mid (\exists x \in B. P(x))$
 $\langle proof \rangle$

lemma *ball-UN*: $(\forall z \in (\bigcup x \in A. B(x)). P(z)) \longleftrightarrow (\forall x \in A. \forall z \in B(x). P(z))$
 $\langle proof \rangle$

lemma *bex-UN*: $(\exists z \in (\bigcup x \in A. B(x)). P(z)) \longleftrightarrow (\exists x \in A. \exists z \in B(x). P(z))$
 $\langle proof \rangle$

4.2 Converse of a Relation

lemma *converse-iff* [simp]: $\langle a, b \rangle \in \text{converse}(r) \longleftrightarrow \langle b, a \rangle \in r$
 $\langle \text{proof} \rangle$

lemma *converseI* [intro!]: $\langle a, b \rangle \in r \implies \langle b, a \rangle \in \text{converse}(r)$
 $\langle \text{proof} \rangle$

lemma *converseD*: $\langle a, b \rangle \in \text{converse}(r) \implies \langle b, a \rangle \in r$
 $\langle \text{proof} \rangle$

lemma *converseE* [elim!]:

$$\begin{aligned} & \llbracket yx \in \text{converse}(r); \\ & \quad \bigwedge x y. \llbracket yx = \langle y, x \rangle; \langle x, y \rangle \in r \rrbracket \implies P \rrbracket \\ & \implies P \end{aligned}$$

 $\langle \text{proof} \rangle$

lemma *converse-converse*: $r \subseteq \text{Sigma}(A, B) \implies \text{converse}(\text{converse}(r)) = r$
 $\langle \text{proof} \rangle$

lemma *converse-type*: $r \subseteq A * B \implies \text{converse}(r) \subseteq B * A$
 $\langle \text{proof} \rangle$

lemma *converse-prod* [simp]: $\text{converse}(A * B) = B * A$
 $\langle \text{proof} \rangle$

lemma *converse-empty* [simp]: $\text{converse}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *converse-subset-iff*:
 $A \subseteq \text{Sigma}(X, Y) \implies \text{converse}(A) \subseteq \text{converse}(B) \longleftrightarrow A \subseteq B$
 $\langle \text{proof} \rangle$

4.3 Finite Set Constructions Using *cons*

lemma *cons-subsetI*: $\llbracket a \in C; B \subseteq C \rrbracket \implies \text{cons}(a, B) \subseteq C$
 $\langle \text{proof} \rangle$

lemma *subset-consI*: $B \subseteq \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *cons-subset-iff* [iff]: $\text{cons}(a, B) \subseteq C \longleftrightarrow a \in C \wedge B \subseteq C$
 $\langle \text{proof} \rangle$

lemmas *cons-subsetE* = *cons-subset-iff* [THEN iffD1, THEN conjE]

lemma *subset-empty-iff*: $A \subseteq 0 \longleftrightarrow A = 0$
 $\langle \text{proof} \rangle$

lemma *subset-cons-iff*: $C \subseteq \text{cons}(a, B) \longleftrightarrow C \subseteq B \mid (a \in C \wedge C - \{a\} \subseteq B)$
 $\langle \text{proof} \rangle$

lemma *cons-eq*: $\{a\} \cup B = \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *cons-commute*: $\text{cons}(a, \text{cons}(b, C)) = \text{cons}(b, \text{cons}(a, C))$
 $\langle \text{proof} \rangle$

lemma *cons-absorb*: $a: B \implies \text{cons}(a, B) = B$
 $\langle \text{proof} \rangle$

lemma *cons-Diff*: $a: B \implies \text{cons}(a, B - \{a\}) = B$
 $\langle \text{proof} \rangle$

lemma *Diff-cons-eq*: $\text{cons}(a, B) - C = (\text{if } a \in C \text{ then } B - C \text{ else } \text{cons}(a, B - C))$
 $\langle \text{proof} \rangle$

lemma *equal-singleton*: $\llbracket a: C; \bigwedge y. y \in C \implies y = b \rrbracket \implies C = \{b\}$
 $\langle \text{proof} \rangle$

lemma *[simp]*: $\text{cons}(a, \text{cons}(a, B)) = \text{cons}(a, B)$
 $\langle \text{proof} \rangle$

lemma *singleton-subsetI*: $a \in C \implies \{a\} \subseteq C$
 $\langle \text{proof} \rangle$

lemma *singleton-subsetD*: $\{a\} \subseteq C \implies a \in C$
 $\langle \text{proof} \rangle$

lemma *subset-succI*: $i \subseteq \text{succ}(i)$
 $\langle \text{proof} \rangle$

lemma *succ-subsetI*: $\llbracket i \in j; i \subseteq j \rrbracket \implies \text{succ}(i) \subseteq j$
 $\langle \text{proof} \rangle$

lemma *succ-subsetE*:
 $\llbracket \text{succ}(i) \subseteq j; \llbracket i \in j; i \subseteq j \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *succ-subset-iff*: $\text{succ}(a) \subseteq B \longleftrightarrow (a \subseteq B \wedge a \in B)$
 $\langle \text{proof} \rangle$

4.4 Binary Intersection

lemma *Int-subset-iff*: $C \subseteq A \cap B \longleftrightarrow C \subseteq A \wedge C \subseteq B$
<proof>

lemma *Int-lower1*: $A \cap B \subseteq A$
<proof>

lemma *Int-lower2*: $A \cap B \subseteq B$
<proof>

lemma *Int-greatest*: $\llbracket C \subseteq A; C \subseteq B \rrbracket \implies C \subseteq A \cap B$
<proof>

lemma *Int-cons*: $\text{cons}(a, B) \cap C \subseteq \text{cons}(a, B \cap C)$
<proof>

lemma *Int-absorb [simp]*: $A \cap A = A$
<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*

lemma *Int-absorb1*: $B \subseteq A \implies A \cap B = B$
<proof>

lemma *Int-absorb2*: $A \subseteq B \implies A \cap B = A$
<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 *subset-Int-iff*: $A \subseteq B \longleftrightarrow A \cap B = A$
<proof>

lemma *subset-Int-iff2*: $A \subseteq B \longleftrightarrow B \cap A = A$
 $\langle proof \rangle$

lemma *Int-Diff-eq*: $C \subseteq A \implies (A - B) \cap C = C - B$
 $\langle proof \rangle$

lemma *Int-cons-left*:
 $cons(a, A) \cap B = (if\ a \in B\ then\ cons(a, A \cap B)\ else\ A \cap B)$
 $\langle proof \rangle$

lemma *Int-cons-right*:
 $A \cap cons(a, B) = (if\ a \in A\ then\ cons(a, A \cap B)\ else\ A \cap B)$
 $\langle proof \rangle$

lemma *cons-Int-distrib*: $cons(x, A \cap B) = cons(x, A) \cap cons(x, B)$
 $\langle proof \rangle$

4.5 Binary Union

lemma *Un-subset-iff*: $A \cup B \subseteq C \longleftrightarrow A \subseteq C \wedge B \subseteq C$
 $\langle proof \rangle$

lemma *Un-upper1*: $A \subseteq A \cup B$
 $\langle proof \rangle$

lemma *Un-upper2*: $B \subseteq A \cup B$
 $\langle proof \rangle$

lemma *Un-least*: $\llbracket A \subseteq C; B \subseteq C \rrbracket \implies A \cup B \subseteq C$
 $\langle proof \rangle$

lemma *Un-cons*: $cons(a, B) \cup C = cons(a, B \cup C)$
 $\langle proof \rangle$

lemma *Un-absorb [simp]*: $A \cup A = A$
 $\langle proof \rangle$

lemma *Un-left-absorb*: $A \cup (A \cup B) = A \cup B$
 $\langle proof \rangle$

lemma *Un-commute*: $A \cup B = B \cup A$
 $\langle proof \rangle$

lemma *Un-left-commute*: $A \cup (B \cup C) = B \cup (A \cup C)$
 $\langle proof \rangle$

lemma *Un-assoc*: $(A \cup B) \cup C = A \cup (B \cup C)$
 $\langle proof \rangle$

lemmas $Un-ac = Un-assoc \ Un-left-absorb \ Un-commute \ Un-left-commute$

lemma $Un-absorb1: A \subseteq B \implies A \cup B = B$
 $\langle proof \rangle$

lemma $Un-absorb2: B \subseteq A \implies A \cup B = A$
 $\langle proof \rangle$

lemma $Un-Int-distrib: (A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
 $\langle proof \rangle$

lemma $subset-Un-iff: A \subseteq B \longleftrightarrow A \cup B = B$
 $\langle proof \rangle$

lemma $subset-Un-iff2: A \subseteq B \longleftrightarrow B \cup A = B$
 $\langle proof \rangle$

lemma $Un-empty [iff]: (A \cup B = 0) \longleftrightarrow (A = 0 \wedge B = 0)$
 $\langle proof \rangle$

lemma $Un-eq-Union: A \cup B = \bigcup(\{A, B\})$
 $\langle proof \rangle$

4.6 Set Difference

lemma $Diff-subset: A - B \subseteq A$
 $\langle proof \rangle$

lemma $Diff-contains: \llbracket C \subseteq A; \ C \cap B = 0 \rrbracket \implies C \subseteq A - B$
 $\langle proof \rangle$

lemma $subset-Diff-cons-iff: B \subseteq A - cons(c, C) \longleftrightarrow B \subseteq A - C \wedge c \notin B$
 $\langle proof \rangle$

lemma $Diff-cancel: A - A = 0$
 $\langle proof \rangle$

lemma $Diff-triv: A \cap B = 0 \implies A - B = A$
 $\langle proof \rangle$

lemma $empty-Diff [simp]: 0 - A = 0$
 $\langle proof \rangle$

lemma $Diff-0 [simp]: A - 0 = A$
 $\langle proof \rangle$

lemma $Diff-eq-0-iff: A - B = 0 \longleftrightarrow A \subseteq B$
 $\langle proof \rangle$

lemma *Diff-cons*: $A - \text{cons}(a, B) = A - B - \{a\}$
 $\langle \text{proof} \rangle$

lemma *Diff-cons2*: $A - \text{cons}(a, B) = A - \{a\} - B$
 $\langle \text{proof} \rangle$

lemma *Diff-disjoint*: $A \cap (B - A) = \emptyset$
 $\langle \text{proof} \rangle$

lemma *Diff-partition*: $A \subseteq B \implies A \cup (B - A) = B$
 $\langle \text{proof} \rangle$

lemma *subset-Un-Diff*: $A \subseteq B \cup (A - B)$
 $\langle \text{proof} \rangle$

lemma *double-complement*: $\llbracket A \subseteq B; B \subseteq C \rrbracket \implies B - (C - A) = A$
 $\langle \text{proof} \rangle$

lemma *double-complement-Un*: $(A \cup B) - (B - A) = A$
 $\langle \text{proof} \rangle$

lemma *Un-Int-crazy*:
 $(A \cap B) \cup (B \cap C) \cup (C \cap A) = (A \cup B) \cap (B \cup C) \cap (C \cup A)$
 $\langle \text{proof} \rangle$

lemma *Diff-Un*: $A - (B \cup C) = (A - B) \cap (A - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int*: $A - (B \cap C) = (A - B) \cup (A - C)$
 $\langle \text{proof} \rangle$

lemma *Un-Diff*: $(A \cup B) - C = (A - C) \cup (B - C)$
 $\langle \text{proof} \rangle$

lemma *Int-Diff*: $(A \cap B) - C = A \cap (B - C)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib*: $C \cap (A - B) = (C \cap A) - (C \cap B)$
 $\langle \text{proof} \rangle$

lemma *Diff-Int-distrib2*: $(A - B) \cap C = (A \cap C) - (B \cap C)$
 $\langle \text{proof} \rangle$

lemma *Un-Int-assoc-iff*: $(A \cap B) \cup C = A \cap (B \cup C) \iff C \subseteq A$
 $\langle \text{proof} \rangle$

4.7 Big Union and Intersection

lemma *Union-subset-iff*: $\bigcup(A) \subseteq C \longleftrightarrow (\forall x \in A. x \subseteq C)$
 $\langle proof \rangle$

lemma *Union-upper*: $B \in A \implies B \subseteq \bigcup(A)$
 $\langle proof \rangle$

lemma *Union-least*: $\llbracket \bigwedge x. x \in A \implies x \subseteq C \rrbracket \implies \bigcup(A) \subseteq C$
 $\langle proof \rangle$

lemma *Union-cons* [simp]: $\bigcup(\text{cons}(a, B)) = a \cup \bigcup(B)$
 $\langle proof \rangle$

lemma *Union-Un-distrib*: $\bigcup(A \cup B) = \bigcup(A) \cup \bigcup(B)$
 $\langle proof \rangle$

lemma *Union-Int-subset*: $\bigcup(A \cap B) \subseteq \bigcup(A) \cap \bigcup(B)$
 $\langle proof \rangle$

lemma *Union-disjoint*: $\bigcup(C) \cap A = 0 \longleftrightarrow (\forall B \in C. B \cap A = 0)$
 $\langle proof \rangle$

lemma *Union-empty-iff*: $\bigcup(A) = 0 \longleftrightarrow (\forall B \in A. B = 0)$
 $\langle proof \rangle$

lemma *Int-Union2*: $\bigcup(B) \cap A = (\bigcup C \in B. C \cap A)$
 $\langle proof \rangle$

lemma *Inter-subset-iff*: $A \neq 0 \implies C \subseteq \bigcap(A) \longleftrightarrow (\forall x \in A. C \subseteq x)$
 $\langle proof \rangle$

lemma *Inter-lower*: $B \in A \implies \bigcap(A) \subseteq B$
 $\langle proof \rangle$

lemma *Inter-greatest*: $\llbracket A \neq 0; \bigwedge x. x \in A \implies C \subseteq x \rrbracket \implies C \subseteq \bigcap(A)$
 $\langle proof \rangle$

lemma *INT-lower*: $x \in A \implies (\bigcap x \in A. B(x)) \subseteq B(x)$
 $\langle proof \rangle$

lemma *INT-greatest*: $\llbracket A \neq 0; \bigwedge x. x \in A \implies C \subseteq B(x) \rrbracket \implies C \subseteq (\bigcap x \in A. B(x))$
 $\langle proof \rangle$

lemma *Inter-0* [simp]: $\bigcap(0) = 0$
 $\langle proof \rangle$

lemma *Inter-Un-subset*:

$$\llbracket z \in A; z \in B \rrbracket \implies \bigcap (A) \cup \bigcap (B) \subseteq \bigcap (A \cap B)$$

<proof>

lemma *Inter-Un-distrib*:

$$\llbracket A \neq 0; B \neq 0 \rrbracket \implies \bigcap (A \cup B) = \bigcap (A) \cap \bigcap (B)$$

<proof>

lemma *Union-singleton*: $\bigcup (\{b\}) = b$

<proof>

lemma *Inter-singleton*: $\bigcap (\{b\}) = b$

<proof>

lemma *Inter-cons [simp]*:

$$\bigcap (\text{cons}(a, B)) = (\text{if } B = 0 \text{ then } a \text{ else } a \cap \bigcap (B))$$

<proof>

4.8 Unions and Intersections of Families

lemma *subset-UN-iff-eq*: $A \subseteq (\bigcup i \in I. B(i)) \longleftrightarrow A = (\bigcup i \in I. A \cap B(i))$

<proof>

lemma *UN-subset-iff*: $(\bigcup x \in A. B(x)) \subseteq C \longleftrightarrow (\forall x \in A. B(x) \subseteq C)$

<proof>

lemma *UN-upper*: $x \in A \implies B(x) \subseteq (\bigcup x \in A. B(x))$

<proof>

lemma *UN-least*: $\llbracket \bigwedge x. x \in A \implies B(x) \subseteq C \rrbracket \implies (\bigcup x \in A. B(x)) \subseteq C$

<proof>

lemma *Union-eq-UN*: $\bigcup (A) = (\bigcup x \in A. x)$

<proof>

lemma *Inter-eq-INT*: $\bigcap (A) = (\bigcap x \in A. x)$

<proof>

lemma *UN-0 [simp]*: $(\bigcup i \in 0. A(i)) = 0$

<proof>

lemma *UN-singleton*: $(\bigcup x \in A. \{x\}) = A$

<proof>

lemma *UN-Un*: $(\bigcup i \in A \cup B. C(i)) = (\bigcup i \in A. C(i)) \cup (\bigcup i \in B. C(i))$

<proof>

lemma *INT-Un*: $(\bigcap_{i \in I \cup J} A(i)) =$
 (if $I=0$ *then* $\bigcap_{j \in J} A(j)$
 else if $J=0$ *then* $\bigcap_{i \in I} A(i)$
 else $((\bigcap_{i \in I} A(i)) \cap (\bigcap_{j \in J} A(j)))$
 $\langle \text{proof} \rangle$

lemma *UN-UN-flatten*: $(\bigcup x \in (\bigcup y \in A. B(y)). C(x)) = (\bigcup y \in A. \bigcup x \in B(y). C(x))$
 $\langle \text{proof} \rangle$

lemma *Int-UN-distrib*: $B \cap (\bigcup_{i \in I} A(i)) = (\bigcup_{i \in I} B \cap A(i))$
 $\langle \text{proof} \rangle$

lemma *Un-INT-distrib*: $I \neq 0 \implies B \cup (\bigcap_{i \in I} A(i)) = (\bigcap_{i \in I} B \cup A(i))$
 $\langle \text{proof} \rangle$

lemma *Int-UN-distrib2*:
 $(\bigcup_{i \in I} A(i)) \cap (\bigcup_{j \in J} B(j)) = (\bigcup_{i \in I} \bigcup_{j \in J} A(i) \cap B(j))$
 $\langle \text{proof} \rangle$

lemma *Un-INT-distrib2*: $\llbracket I \neq 0; J \neq 0 \rrbracket \implies$
 $(\bigcap_{i \in I} A(i)) \cup (\bigcap_{j \in J} B(j)) = (\bigcap_{i \in I} \bigcap_{j \in J} A(i) \cup B(j))$
 $\langle \text{proof} \rangle$

lemma *UN-constant [simp]*: $(\bigcup y \in A. c) = (\text{if } A=0 \text{ then } 0 \text{ else } c)$
 $\langle \text{proof} \rangle$

lemma *INT-constant [simp]*: $(\bigcap y \in A. c) = (\text{if } A=0 \text{ then } 0 \text{ else } c)$
 $\langle \text{proof} \rangle$

lemma *UN-RepFun [simp]*: $(\bigcup y \in \text{RepFun}(A, f). B(y)) = (\bigcup x \in A. B(f(x)))$
 $\langle \text{proof} \rangle$

lemma *INT-RepFun [simp]*: $(\bigcap x \in \text{RepFun}(A, f). B(x)) = (\bigcap a \in A. B(f(a)))$
 $\langle \text{proof} \rangle$

lemma *INT-Union-eq*:
 $0 \notin A \implies (\bigcap x \in \bigcup(A). B(x)) = (\bigcap y \in A. \bigcap x \in y. B(x))$
 $\langle \text{proof} \rangle$

lemma *INT-UN-eq*:
 $(\forall x \in A. B(x) \neq 0) \implies (\bigcap z \in (\bigcup x \in A. B(x)). C(z)) = (\bigcap x \in A. \bigcap z \in B(x). C(z))$
 $\langle \text{proof} \rangle$

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-Int-distrib*:

$$I \neq 0 \implies (\bigcap_{i \in I}. A(i) \cap B(i)) = (\bigcap_{i \in I}. A(i)) \cap (\bigcap_{i \in I}. B(i))$$

<proof>

lemma *UN-Int-subset*:

$$(\bigcup_{z \in I \cap J}. A(z)) \subseteq (\bigcup_{z \in I}. A(z)) \cap (\bigcup_{z \in J}. A(z))$$

<proof>

lemma *Diff-UN*: $I \neq 0 \implies B - (\bigcup_{i \in I}. A(i)) = (\bigcap_{i \in I}. B - A(i))$

<proof>

lemma *Diff-INT*: $I \neq 0 \implies B - (\bigcap_{i \in I}. A(i)) = (\bigcup_{i \in I}. B - A(i))$

<proof>

lemma *Sigma-cons1*: $Sigma(cons(a, B), C) = (\{a\} * C(a)) \cup Sigma(B, C)$

<proof>

lemma *Sigma-cons2*: $A * cons(b, B) = A * \{b\} \cup A * B$

<proof>

lemma *Sigma-succ1*: $Sigma(succ(A), B) = (\{A\} * B(A)) \cup Sigma(A, B)$

<proof>

lemma *Sigma-succ2*: $A * succ(B) = A * \{B\} \cup A * B$

<proof>

lemma *SUM-UN-distrib1*:

$$(\sum x \in (\bigcup y \in A. C(y)). B(x)) = (\bigcup y \in A. \sum x \in C(y). B(x))$$

<proof>

lemma *SUM-UN-distrib2*:

$$(\sum i \in I. \bigcup j \in J. C(i, j)) = (\bigcup j \in J. \sum i \in I. C(i, j))$$

<proof>

lemma *SUM-Un-distrib1*:

$$(\sum i \in I \cup J. C(i)) = (\sum i \in I. C(i)) \cup (\sum j \in J. C(j))$$

<proof>

lemma *SUM-Un-distrib2*:

$$\langle proof \rangle \quad (\sum_{i \in I}. A(i) \cup B(i)) = (\sum_{i \in I}. A(i)) \cup (\sum_{i \in I}. B(i))$$

$$\langle proof \rangle \quad \textbf{lemma } prod-Un-distrib2: I * (A \cup B) = I * A \cup I * B$$

$$\langle proof \rangle \quad \textbf{lemma } SUM-Int-distrib1: \\ (\sum_{i \in I \cap J}. C(i)) = (\sum_{i \in I}. C(i)) \cap (\sum_{j \in J}. C(j))$$

$$\langle proof \rangle \quad \textbf{lemma } SUM-Int-distrib2: \\ (\sum_{i \in I}. A(i) \cap B(i)) = (\sum_{i \in I}. A(i)) \cap (\sum_{i \in I}. B(i))$$

$$\langle proof \rangle \quad \textbf{lemma } prod-Int-distrib2: I * (A \cap B) = I * A \cap I * B$$

$$\langle proof \rangle \quad \textbf{lemma } SUM-eq-UN: (\sum_{i \in I}. A(i)) = (\bigcup_{i \in I}. \{i\} * A(i))$$

$$\langle proof \rangle \quad \textbf{lemma } times-subset-iff: \\ (A' * B' \subseteq A * B) \longleftrightarrow (A' = 0 \mid B' = 0 \mid (A' \subseteq A) \wedge (B' \subseteq B))$$

$$\langle proof \rangle \quad \textbf{lemma } Int-Sigma-eq: \\ (\sum_{x \in A'}. B'(x)) \cap (\sum_{x \in A}. B(x)) = (\sum_{x \in A' \cap A}. B'(x) \cap B(x))$$

$$\langle proof \rangle \quad \textbf{lemma } domain-iff: a: domain(r) \longleftrightarrow (\exists y. \langle a, y \rangle \in r)$$

$$\langle proof \rangle \quad \textbf{lemma } domainI [intro]: \langle a, b \rangle \in r \implies a: domain(r)$$

$$\langle proof \rangle \quad \textbf{lemma } domainE [elim!]: \\ \llbracket a \in domain(r); \bigwedge y. \langle a, y \rangle \in r \implies P \rrbracket \implies P$$

$$\langle proof \rangle \quad \textbf{lemma } domain-subset: domain(Sigma(A, B)) \subseteq A$$

$$\langle proof \rangle \quad \textbf{lemma } domain-of-prod: b \in B \implies domain(A * B) = A$$

lemma *domain-0* [*simp*]: $\text{domain}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *domain-cons* [*simp*]: $\text{domain}(\text{cons}(\langle a, b \rangle, r)) = \text{cons}(a, \text{domain}(r))$
 $\langle \text{proof} \rangle$

lemma *domain-Un-eq* [*simp*]: $\text{domain}(A \cup B) = \text{domain}(A) \cup \text{domain}(B)$
 $\langle \text{proof} \rangle$

lemma *domain-Int-subset*: $\text{domain}(A \cap B) \subseteq \text{domain}(A) \cap \text{domain}(B)$
 $\langle \text{proof} \rangle$

lemma *domain-Diff-subset*: $\text{domain}(A) - \text{domain}(B) \subseteq \text{domain}(A - B)$
 $\langle \text{proof} \rangle$

lemma *domain-UN*: $\text{domain}(\bigcup x \in A. B(x)) = (\bigcup x \in A. \text{domain}(B(x)))$
 $\langle \text{proof} \rangle$

lemma *domain-Union*: $\text{domain}(\bigcup (A)) = (\bigcup x \in A. \text{domain}(x))$
 $\langle \text{proof} \rangle$

lemma *rangeI* [*intro*]: $\langle a, b \rangle \in r \implies b \in \text{range}(r)$
 $\langle \text{proof} \rangle$

lemma *rangeE* [*elim!*]: $\llbracket b \in \text{range}(r); \bigwedge x. \langle x, b \rangle \in r \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *range-subset*: $\text{range}(A * B) \subseteq B$
 $\langle \text{proof} \rangle$

lemma *range-of-prod*: $a \in A \implies \text{range}(A * B) = B$
 $\langle \text{proof} \rangle$

lemma *range-0* [*simp*]: $\text{range}(0) = 0$
 $\langle \text{proof} \rangle$

lemma *range-cons* [*simp*]: $\text{range}(\text{cons}(\langle a, b \rangle, r)) = \text{cons}(b, \text{range}(r))$
 $\langle \text{proof} \rangle$

lemma *range-Un-eq* [*simp*]: $\text{range}(A \cup B) = \text{range}(A) \cup \text{range}(B)$
 $\langle \text{proof} \rangle$

lemma *range-Int-subset*: $\text{range}(A \cap B) \subseteq \text{range}(A) \cap \text{range}(B)$
 $\langle \text{proof} \rangle$

lemma *range-Diff-subset*: $\text{range}(A) - \text{range}(B) \subseteq \text{range}(A - B)$

$\langle proof \rangle$

lemma *domain-converse* [simp]: $domain(converse(r)) = range(r)$
 $\langle proof \rangle$

lemma *range-converse* [simp]: $range(converse(r)) = domain(r)$
 $\langle proof \rangle$

lemma *fieldI1*: $\langle a, b \rangle \in r \implies a \in field(r)$
 $\langle proof \rangle$

lemma *fieldI2*: $\langle a, b \rangle \in r \implies b \in field(r)$
 $\langle proof \rangle$

lemma *fieldCI* [intro]:
 $(\neg \langle c, a \rangle \in r \implies \langle a, b \rangle \in r) \implies a \in field(r)$
 $\langle proof \rangle$

lemma *fieldE* [elim!]:
 $\llbracket a \in field(r);$
 $\bigwedge x. \langle a, x \rangle \in r \implies P;$
 $\bigwedge x. \langle x, a \rangle \in r \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *field-subset*: $field(A*B) \subseteq A \cup B$
 $\langle proof \rangle$

lemma *domain-subset-field*: $domain(r) \subseteq field(r)$
 $\langle proof \rangle$

lemma *range-subset-field*: $range(r) \subseteq field(r)$
 $\langle proof \rangle$

lemma *domain-times-range*: $r \subseteq Sigma(A, B) \implies r \subseteq domain(r)*range(r)$
 $\langle proof \rangle$

lemma *field-times-field*: $r \subseteq Sigma(A, B) \implies r \subseteq field(r)*field(r)$
 $\langle proof \rangle$

lemma *relation-field-times-field*: $relation(r) \implies r \subseteq field(r)*field(r)$
 $\langle proof \rangle$

lemma *field-of-prod*: $field(A*A) = A$
 $\langle proof \rangle$

lemma *field-0* [simp]: $field(0) = 0$

$\langle proof \rangle$

lemma *field-cons* [simp]: $field(cons(\langle a, b \rangle, r)) = cons(a, cons(b, field(r)))$
 $\langle proof \rangle$

lemma *field-Un-eq* [simp]: $field(A \cup B) = field(A) \cup field(B)$
 $\langle proof \rangle$

lemma *field-Int-subset*: $field(A \cap B) \subseteq field(A) \cap field(B)$
 $\langle proof \rangle$

lemma *field-Diff-subset*: $field(A) - field(B) \subseteq field(A - B)$
 $\langle proof \rangle$

lemma *field-converse* [simp]: $field(converse(r)) = field(r)$
 $\langle proof \rangle$

lemma *rel-Union*: $(\forall x \in S. \exists A B. x \subseteq A * B) \implies$
 $\bigcup(S) \subseteq domain(\bigcup(S)) * range(\bigcup(S))$
 $\langle proof \rangle$

lemma *rel-Un*: $\llbracket r \subseteq A * B; s \subseteq C * D \rrbracket \implies (r \cup s) \subseteq (A \cup C) * (B \cup D)$
 $\langle proof \rangle$

lemma *domain-Diff-eq*: $\llbracket \langle a, c \rangle \in r; c \neq b \rrbracket \implies domain(r - \{\langle a, b \rangle\}) = domain(r)$
 $\langle proof \rangle$

lemma *range-Diff-eq*: $\llbracket \langle c, b \rangle \in r; c \neq a \rrbracket \implies range(r - \{\langle a, b \rangle\}) = range(r)$
 $\langle proof \rangle$

4.9 Image of a Set under a Function or Relation

lemma *image-iff*: $b \in r''A \longleftrightarrow (\exists x \in A. \langle x, b \rangle \in r)$
 $\langle proof \rangle$

lemma *image-singleton-iff*: $b \in r''\{a\} \longleftrightarrow \langle a, b \rangle \in r$
 $\langle proof \rangle$

lemma *imageI* [intro]: $\llbracket \langle a, b \rangle \in r; a \in A \rrbracket \implies b \in r''A$
 $\langle proof \rangle$

lemma *imageE* [elim!]:
 $\llbracket b: r''A; \bigwedge x. \llbracket \langle x, b \rangle \in r; x \in A \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *image-subset*: $r \subseteq A * B \implies r''C \subseteq B$
 $\langle proof \rangle$

lemma *image-0* [simp]: $r^{-1}0 = 0$

<proof>

lemma *image-Un* [simp]: $r^{-1}(A \cup B) = (r^{-1}A) \cup (r^{-1}B)$

<proof>

lemma *image-UN*: $r^{-1}(\bigcup_{x \in A} B(x)) = \bigcup_{x \in A} r^{-1}B(x)$

<proof>

lemma *Collect-image-eq*:

$\{z \in \text{Sigma}(A, B). P(z)\}^{-1} C = (\bigcup x \in A. \{y \in B(x). x \in C \wedge P(\langle x, y \rangle)\})^{-1}$

<proof>

lemma *image-Int-subset*: $r^{-1}(A \cap B) \subseteq (r^{-1}A) \cap (r^{-1}B)$

<proof>

lemma *image-Int-square-subset*: $(r \cap A * A)^{-1}B \subseteq (r^{-1}B) \cap A$

<proof>

lemma *image-Int-square*: $B \subseteq A \implies (r \cap A * A)^{-1}B = (r^{-1}B) \cap A$

<proof>

lemma *image-0-left* [simp]: $0^{-1}A = 0$

<proof>

lemma *image-Un-left*: $(r \cup s)^{-1}A = (r^{-1}A) \cup (s^{-1}A)$

<proof>

lemma *image-Int-subset-left*: $(r \cap s)^{-1}A \subseteq (r^{-1}A) \cap (s^{-1}A)$

<proof>

4.10 Inverse Image of a Set under a Function or Relation

lemma *vimage-iff*:

$a \in r^{-1}B \iff (\exists y \in B. \langle a, y \rangle \in r)$

<proof>

lemma *vimage-singleton-iff*: $a \in r^{-1}\{b\} \iff \langle a, b \rangle \in r$

<proof>

lemma *vimageI* [intro]: $\llbracket \langle a, b \rangle \in r; b \in B \rrbracket \implies a \in r^{-1}B$

<proof>

lemma *vimageE* [elim!]:

$\llbracket a \in r^{-1}B; \bigwedge x. \llbracket \langle a, x \rangle \in r; x \in B \rrbracket \implies P \rrbracket \implies P$

<proof>

lemma *vimage-subset*: $r \subseteq A*B \implies r-''C \subseteq A$
 $\langle \text{proof} \rangle$

lemma *vimage-0* [simp]: $r-''0 = 0$
 $\langle \text{proof} \rangle$

lemma *vimage-Un* [simp]: $r-''(A \cup B) = (r-''A) \cup (r-''B)$
 $\langle \text{proof} \rangle$

lemma *vimage-Int-subset*: $r-''(A \cap B) \subseteq (r-''A) \cap (r-''B)$
 $\langle \text{proof} \rangle$

lemma *vimage-eq-UN*: $f-''B = (\bigcup y \in B. f-''\{y\})$
 $\langle \text{proof} \rangle$

lemma *function-vimage-Int*:
 $\text{function}(f) \implies f-''(A \cap B) = (f-''A) \cap (f-''B)$
 $\langle \text{proof} \rangle$

lemma *function-vimage-Diff*: $\text{function}(f) \implies f-''(A-B) = (f-''A) - (f-''B)$
 $\langle \text{proof} \rangle$

lemma *function-image-vimage*: $\text{function}(f) \implies f-''(f-''A) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *vimage-Int-square-subset*: $(r \cap A*A)-''B \subseteq (r-''B) \cap A$
 $\langle \text{proof} \rangle$

lemma *vimage-Int-square*: $B \subseteq A \implies (r \cap A*A)-''B = (r-''B) \cap A$
 $\langle \text{proof} \rangle$

lemma *vimage-0-left* [simp]: $0-''A = 0$
 $\langle \text{proof} \rangle$

lemma *vimage-Un-left*: $(r \cup s)-''A = (r-''A) \cup (s-''A)$
 $\langle \text{proof} \rangle$

lemma *vimage-Int-subset-left*: $(r \cap s)-''A \subseteq (r-''A) \cap (s-''A)$
 $\langle \text{proof} \rangle$

lemma *converse-Un* [simp]: $\text{converse}(A \cup B) = \text{converse}(A) \cup \text{converse}(B)$

$\langle \text{proof} \rangle$

lemma *converse-Int* [simp]: $\text{converse}(A \cap B) = \text{converse}(A) \cap \text{converse}(B)$
 $\langle \text{proof} \rangle$

lemma *converse-Diff* [simp]: $\text{converse}(A - B) = \text{converse}(A) - \text{converse}(B)$
 $\langle \text{proof} \rangle$

lemma *converse-UN* [simp]: $\text{converse}(\bigcup x \in A. B(x)) = (\bigcup x \in A. \text{converse}(B(x)))$
 $\langle \text{proof} \rangle$

lemma *converse-INT* [simp]:
 $\text{converse}(\bigcap x \in A. B(x)) = (\bigcap x \in A. \text{converse}(B(x)))$
 $\langle \text{proof} \rangle$

4.11 Powerset Operator

lemma *Pow-0* [simp]: $\text{Pow}(0) = \{0\}$
 $\langle \text{proof} \rangle$

lemma *Pow-insert*: $\text{Pow}(\text{cons}(a, A)) = \text{Pow}(A) \cup \{\text{cons}(a, X) \mid X \in \text{Pow}(A)\}$
 $\langle \text{proof} \rangle$

lemma *Un-Pow-subset*: $\text{Pow}(A) \cup \text{Pow}(B) \subseteq \text{Pow}(A \cup B)$
 $\langle \text{proof} \rangle$

lemma *UN-Pow-subset*: $(\bigcup x \in A. \text{Pow}(B(x))) \subseteq \text{Pow}(\bigcup x \in A. B(x))$
 $\langle \text{proof} \rangle$

lemma *subset-Pow-Union*: $A \subseteq \text{Pow}(\bigcup(A))$
 $\langle \text{proof} \rangle$

lemma *Union-Pow-eq* [simp]: $\bigcup(\text{Pow}(A)) = A$
 $\langle \text{proof} \rangle$

lemma *Union-Pow-iff*: $\bigcup(A) \in \text{Pow}(B) \longleftrightarrow A \in \text{Pow}(\text{Pow}(B))$
 $\langle \text{proof} \rangle$

lemma *Pow-Int-eq* [simp]: $\text{Pow}(A \cap B) = \text{Pow}(A) \cap \text{Pow}(B)$
 $\langle \text{proof} \rangle$

lemma *Pow-INT-eq*: $A \neq 0 \implies \text{Pow}(\bigcap x \in A. B(x)) = (\bigcap x \in A. \text{Pow}(B(x)))$
 $\langle \text{proof} \rangle$

4.12 RepFun

lemma *RepFun-subset*: $\llbracket \bigwedge x. x \in A \implies f(x) \in B \rrbracket \implies \{f(x) \mid x \in A\} \subseteq B$
 $\langle \text{proof} \rangle$

lemma *RepFun-eq-0-iff* [simp]: $\{f(x).x \in A\} = 0 \longleftrightarrow A = 0$
 $\langle \text{proof} \rangle$

lemma *RepFun-constant* [simp]: $\{c. x \in A\} = (\text{if } A = 0 \text{ then } 0 \text{ else } \{c\})$
 $\langle \text{proof} \rangle$

4.13 Collect

lemma *Collect-subset*: $\text{Collect}(A, P) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *Collect-Un*: $\text{Collect}(A \cup B, P) = \text{Collect}(A, P) \cup \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Int*: $\text{Collect}(A \cap B, P) = \text{Collect}(A, P) \cap \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Diff*: $\text{Collect}(A - B, P) = \text{Collect}(A, P) - \text{Collect}(B, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-cons*: $\{x \in \text{cons}(a, B). P(x)\} =$
 $(\text{if } P(a) \text{ then } \text{cons}(a, \{x \in B. P(x)\}) \text{ else } \{x \in B. P(x)\})$
 $\langle \text{proof} \rangle$

lemma *Int-Collect-self-eq*: $A \cap \text{Collect}(A, P) = \text{Collect}(A, P)$
 $\langle \text{proof} \rangle$

lemma *Collect-Collect-eq* [simp]:
 $\text{Collect}(\text{Collect}(A, P), Q) = \text{Collect}(A, \lambda x. P(x) \wedge Q(x))$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-Collect-eq*:
 $\text{Collect}(A, P) \cap \text{Collect}(A, Q) = \text{Collect}(A, \lambda x. P(x) \wedge Q(x))$
 $\langle \text{proof} \rangle$

lemma *Collect-Union-eq* [simp]:
 $\text{Collect}(\bigcup x \in A. B(x), P) = (\bigcup x \in A. \text{Collect}(B(x), P))$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-left*: $\{x \in A. P(x)\} \cap B = \{x \in A \cap B. P(x)\}$
 $\langle \text{proof} \rangle$

lemma *Collect-Int-right*: $A \cap \{x \in B. P(x)\} = \{x \in A \cap B. P(x)\}$
 $\langle \text{proof} \rangle$

lemma *Collect-disj-eq*: $\{x \in A. P(x) \mid Q(x)\} = \text{Collect}(A, P) \cup \text{Collect}(A, Q)$
 $\langle \text{proof} \rangle$

lemma *Collect-conj-eq*: $\{x \in A. P(x) \wedge Q(x)\} = \text{Collect}(A, P) \cap \text{Collect}(A, Q)$

$\langle proof \rangle$

lemmas *subset-SIs* = *subset-refl cons-subsetI subset-consI*
Union-least UN-least Un-least
Inter-greatest Int-greatest RepFun-subset
Un-upper1 Un-upper2 Int-lower1 Int-lower2

$\langle ML \rangle$

end

5 Least and Greatest Fixed Points; the Knaster-Tarski Theorem

theory *Fixedpt* **imports** *equalities* **begin**

definition

*bn**d-mono* :: $[i, i \Rightarrow i] \Rightarrow o$ **where**
 $bn\ d\ mono(D, h) \equiv h(D) \leq D \wedge (\forall W\ X. W \leq X \longrightarrow X \leq D \longrightarrow h(W) \subseteq h(X))$

definition

lfp :: $[i, i \Rightarrow i] \Rightarrow i$ **where**
 $lfp(D, h) \equiv \bigcap (\{X: Pow(D). h(X) \subseteq X\})$

definition

gfp :: $[i, i \Rightarrow i] \Rightarrow i$ **where**
 $gfp(D, h) \equiv \bigcup (\{X: Pow(D). X \subseteq h(X)\})$

The theorem is proved in the lattice of subsets of D , namely $Pow(D)$, with *Inter* as the greatest lower bound.

5.1 Monotone Operators

lemma *bn**d-monoI*:

$\llbracket h(D) \leq D; \bigwedge W\ X. \llbracket W \leq D; X \leq D; W \leq X \rrbracket \implies h(W) \subseteq h(X) \rrbracket \implies bn\ d\ mono(D, h)$
 $\langle proof \rangle$

lemma *bn**d-monoD1*: $bn\ d\ mono(D, h) \implies h(D) \subseteq D$
 $\langle proof \rangle$

lemma *bn**d-monoD2*: $\llbracket bn\ d\ mono(D, h); W \leq X; X \leq D \rrbracket \implies h(W) \subseteq h(X)$
 $\langle proof \rangle$

lemma *bn**d-mono-subset*:

$\llbracket \text{bnd-mono}(D, h); X \leq D \rrbracket \implies h(X) \subseteq D$
 $\langle \text{proof} \rangle$

lemma *bnd-mono-Un*:

$\llbracket \text{bnd-mono}(D, h); A \subseteq D; B \subseteq D \rrbracket \implies h(A) \cup h(B) \subseteq h(A \cup B)$
 $\langle \text{proof} \rangle$

lemma *bnd-mono-UN*:

$\llbracket \text{bnd-mono}(D, h); \forall i \in I. A(i) \subseteq D \rrbracket$
 $\implies (\bigcup_{i \in I} h(A(i))) \subseteq h(\bigcup_{i \in I} A(i))$
 $\langle \text{proof} \rangle$

lemma *bnd-mono-Int*:

$\llbracket \text{bnd-mono}(D, h); A \subseteq D; B \subseteq D \rrbracket \implies h(A \cap B) \subseteq h(A) \cap h(B)$
 $\langle \text{proof} \rangle$

5.2 Proof of Knaster-Tarski Theorem using *lfp*

lemma *lfp-lowerbound*:

$\llbracket h(A) \subseteq A; A \leq D \rrbracket \implies \text{lfp}(D, h) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *lfp-subset*: $\text{lfp}(D, h) \subseteq D$
 $\langle \text{proof} \rangle$

lemma *def-lfp-subset*: $A \equiv \text{lfp}(D, h) \implies A \subseteq D$
 $\langle \text{proof} \rangle$

lemma *lfp-greatest*:

$\llbracket h(D) \subseteq D; \bigwedge X. \llbracket h(X) \subseteq X; X \leq D \rrbracket \implies A \leq X \rrbracket \implies A \subseteq \text{lfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *lfp-lemma1*:

$\llbracket \text{bnd-mono}(D, h); h(A) \leq A; A \leq D \rrbracket \implies h(\text{lfp}(D, h)) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *lfp-lemma2*: $\text{bnd-mono}(D, h) \implies h(\text{lfp}(D, h)) \subseteq \text{lfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *lfp-lemma3*:

$\text{bnd-mono}(D, h) \implies \text{lfp}(D, h) \subseteq h(\text{lfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *lfp-unfold*: $\text{bnd-mono}(D, h) \implies \text{lfp}(D, h) = h(\text{lfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *def-lfp-unfold*:

$\llbracket A \equiv \text{lfp}(D, h); \text{ bnd-mono}(D, h) \rrbracket \implies A = h(A)$
 $\langle \text{proof} \rangle$

5.3 General Induction Rule for Least Fixedpoints

lemma *Collect-is-pre-fixedpt*:

$\llbracket \text{ bnd-mono}(D, h); \bigwedge x. x \in h(\text{Collect}(\text{lfp}(D, h), P)) \implies P(x) \rrbracket$
 $\implies h(\text{Collect}(\text{lfp}(D, h), P)) \subseteq \text{Collect}(\text{lfp}(D, h), P)$
 $\langle \text{proof} \rangle$

lemma *induct*:

$\llbracket \text{ bnd-mono}(D, h); a \in \text{lfp}(D, h);$
 $\bigwedge x. x \in h(\text{Collect}(\text{lfp}(D, h), P)) \implies P(x)$
 $\rrbracket \implies P(a)$
 $\langle \text{proof} \rangle$

lemma *def-induct*:

$\llbracket A \equiv \text{lfp}(D, h); \text{ bnd-mono}(D, h); a:A;$
 $\bigwedge x. x \in h(\text{Collect}(A, P)) \implies P(x)$
 $\rrbracket \implies P(a)$
 $\langle \text{proof} \rangle$

lemma *lfp-Int-lowerbound*:

$\llbracket h(D \cap A) \subseteq A; \text{ bnd-mono}(D, h) \rrbracket \implies \text{lfp}(D, h) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *lfp-mono*:

assumes *hmono*: $\text{ bnd-mono}(D, h)$
and *imono*: $\text{ bnd-mono}(E, i)$
and *subhi*: $\bigwedge X. X \leq D \implies h(X) \subseteq i(X)$
shows $\text{lfp}(D, h) \subseteq \text{lfp}(E, i)$
 $\langle \text{proof} \rangle$

lemma *lfp-mono2*:

$\llbracket i(D) \subseteq D; \bigwedge X. X \leq D \implies h(X) \subseteq i(X) \rrbracket \implies \text{lfp}(D, h) \subseteq \text{lfp}(D, i)$
 $\langle \text{proof} \rangle$

lemma *lfp-cong*:

$\llbracket D = D'; \bigwedge X. X \leq D' \implies h(X) = h'(X) \rrbracket \implies \text{lfp}(D, h) = \text{lfp}(D', h')$
 $\langle \text{proof} \rangle$

5.4 Proof of Knaster-Tarski Theorem using *gfp*

lemma *gfp-upperbound*: $\llbracket A \subseteq h(A); A \leq D \rrbracket \implies A \subseteq \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *gfp-subset*: $\text{gfp}(D, h) \subseteq D$
 $\langle \text{proof} \rangle$

lemma *def-gfp-subset*: $A \equiv \text{gfp}(D, h) \implies A \subseteq D$
 $\langle \text{proof} \rangle$

lemma *gfp-least*:
 $\llbracket \text{bnd-mono}(D, h); \bigwedge X. \llbracket X \subseteq h(X); X \leq D \rrbracket \implies X \leq A \rrbracket \implies$
 $\text{gfp}(D, h) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma1*:
 $\llbracket \text{bnd-mono}(D, h); A \leq h(A); A \leq D \rrbracket \implies A \subseteq h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma2*: $\text{bnd-mono}(D, h) \implies \text{gfp}(D, h) \subseteq h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *gfp-lemma3*:
 $\text{bnd-mono}(D, h) \implies h(\text{gfp}(D, h)) \subseteq \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *gfp-unfold*: $\text{bnd-mono}(D, h) \implies \text{gfp}(D, h) = h(\text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *def-gfp-unfold*:
 $\llbracket A \equiv \text{gfp}(D, h); \text{bnd-mono}(D, h) \rrbracket \implies A = h(A)$
 $\langle \text{proof} \rangle$

5.5 Coinduction Rules for Greatest Fixed Points

lemma *weak-coinduct*: $\llbracket a: X; X \subseteq h(X); X \subseteq D \rrbracket \implies a \in \text{gfp}(D, h)$
 $\langle \text{proof} \rangle$

lemma *coinduct-lemma*:
 $\llbracket X \subseteq h(X \cup \text{gfp}(D, h)); X \subseteq D; \text{bnd-mono}(D, h) \rrbracket \implies$
 $X \cup \text{gfp}(D, h) \subseteq h(X \cup \text{gfp}(D, h))$
 $\langle \text{proof} \rangle$

lemma *coinduct*:
 $\llbracket \text{bnd-mono}(D, h); a: X; X \subseteq h(X \cup \text{gfp}(D, h)); X \subseteq D \rrbracket$
 $\implies a \in \text{gfp}(D, h)$

$\langle proof \rangle$

lemma *def-coinduct*:

$$\llbracket A \equiv gfp(D, h); \text{ bnd-mono}(D, h); a: X; X \subseteq h(X \cup A); X \subseteq D \rrbracket \implies$$

$$a \in A$$

 $\langle proof \rangle$

lemma *def-Collect-coinduct*:

$$\llbracket A \equiv gfp(D, \lambda w. \text{Collect}(D, P(w))); \text{ bnd-mono}(D, \lambda w. \text{Collect}(D, P(w)));$$

$$a: X; X \subseteq D; \bigwedge z. z: X \implies P(X \cup A, z) \rrbracket \implies$$

$$a \in A$$

 $\langle proof \rangle$

lemma *gfp-mono*:

$$\llbracket \text{bnd-mono}(D, h); D \subseteq E;$$

$$\bigwedge X. X \leq D \implies h(X) \subseteq i(X) \rrbracket \implies gfp(D, h) \subseteq gfp(E, i)$$

 $\langle proof \rangle$

end

6 Booleans in Zermelo-Fraenkel Set Theory

theory *Bool* **imports** *pair* **begin**

abbreviation

one ($\langle 1 \rangle$) **where**
 $1 \equiv succ(0)$

abbreviation

two ($\langle 2 \rangle$) **where**
 $2 \equiv succ(1)$

2 is equal to bool, but is used as a number rather than a type.

definition *bool* $\equiv \{0, 1\}$

definition *cond*(*b*, *c*, *d*) $\equiv if(b=1, c, d)$

definition *not*(*b*) $\equiv cond(b, 0, 1)$

definition

and $:: [i, i] \Rightarrow i$ (**infixl** $\langle and \rangle$ 70) **where**
 $a \text{ and } b \equiv cond(a, b, 0)$

definition

or $:: [i, i] \Rightarrow i$ (**infixl** $\langle or \rangle$ 65) **where**
 $a \text{ or } b \equiv cond(a, 1, b)$

definition

$xor :: [i,i] \Rightarrow i$ (**infixl** $\langle xor \rangle$ 65) **where**
 $a xor b \equiv cond(a, not(b), b)$

lemmas $bool-defs = bool-def cond-def$

lemma $singleton-0$: $\{0\} = 1$
 $\langle proof \rangle$

lemma $bool-1I$ $[simp, TC]$: $1 \in bool$
 $\langle proof \rangle$

lemma $bool-0I$ $[simp, TC]$: $0 \in bool$
 $\langle proof \rangle$

lemma $one-not-0$: $1 \neq 0$
 $\langle proof \rangle$

lemmas $one-neq-0 = one-not-0$ $[THEN notE]$

lemma $boolE$:
 $\llbracket c: bool; \ c=1 \implies P; \ c=0 \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma $cond-1$ $[simp]$: $cond(1, c, d) = c$
 $\langle proof \rangle$

lemma $cond-0$ $[simp]$: $cond(0, c, d) = d$
 $\langle proof \rangle$

lemma $cond-type$ $[TC]$: $\llbracket b: bool; \ c: A(1); \ d: A(0) \rrbracket \implies cond(b, c, d): A(b)$
 $\langle proof \rangle$

lemma $cond-simple-type$: $\llbracket b: bool; \ c: A; \ d: A \rrbracket \implies cond(b, c, d): A$
 $\langle proof \rangle$

lemma $def-cond-1$: $\llbracket \bigwedge b. j(b) \equiv cond(b, c, d) \rrbracket \implies j(1) = c$
 $\langle proof \rangle$

lemma *def-cond-0*: $\llbracket \bigwedge b. j(b) \equiv \text{cond}(b, c, d) \rrbracket \implies j(0) = d$
 $\langle \text{proof} \rangle$

lemmas *not-1* = *not-def* [*THEN* *def-cond-1*, *simp*]
lemmas *not-0* = *not-def* [*THEN* *def-cond-0*, *simp*]

lemmas *and-1* = *and-def* [*THEN* *def-cond-1*, *simp*]
lemmas *and-0* = *and-def* [*THEN* *def-cond-0*, *simp*]

lemmas *or-1* = *or-def* [*THEN* *def-cond-1*, *simp*]
lemmas *or-0* = *or-def* [*THEN* *def-cond-0*, *simp*]

lemmas *xor-1* = *xor-def* [*THEN* *def-cond-1*, *simp*]
lemmas *xor-0* = *xor-def* [*THEN* *def-cond-0*, *simp*]

lemma *not-type* [*TC*]: $a:\text{bool} \implies \text{not}(a) \in \text{bool}$
 $\langle \text{proof} \rangle$

lemma *and-type* [*TC*]: $\llbracket a:\text{bool}; b:\text{bool} \rrbracket \implies a \text{ and } b \in \text{bool}$
 $\langle \text{proof} \rangle$

lemma *or-type* [*TC*]: $\llbracket a:\text{bool}; b:\text{bool} \rrbracket \implies a \text{ or } b \in \text{bool}$
 $\langle \text{proof} \rangle$

lemma *xor-type* [*TC*]: $\llbracket a:\text{bool}; b:\text{bool} \rrbracket \implies a \text{ xor } b \in \text{bool}$
 $\langle \text{proof} \rangle$

lemmas *bool-typechecks* = *bool-1I* *bool-0I* *cond-type* *not-type* *and-type*
or-type *xor-type*

6.1 Laws About 'not'

lemma *not-not* [*simp*]: $a:\text{bool} \implies \text{not}(\text{not}(a)) = a$
 $\langle \text{proof} \rangle$

lemma *not-and* [*simp*]: $a:\text{bool} \implies \text{not}(a \text{ and } b) = \text{not}(a) \text{ or } \text{not}(b)$
 $\langle \text{proof} \rangle$

lemma *not-or* [*simp*]: $a:\text{bool} \implies \text{not}(a \text{ or } b) = \text{not}(a) \text{ and } \text{not}(b)$
 $\langle \text{proof} \rangle$

6.2 Laws About 'and'

lemma *and-absorb* [*simp*]: $a:\text{bool} \implies a \text{ and } a = a$
 $\langle \text{proof} \rangle$

lemma *and-commute*: $\llbracket a:\text{bool}; b:\text{bool} \rrbracket \implies a \text{ and } b = b \text{ and } a$
 $\langle \text{proof} \rangle$

lemma *and-assoc*: $a:\text{bool} \implies (a \text{ and } b) \text{ and } c = a \text{ and } (b \text{ and } c)$

$\langle proof \rangle$

lemma *and-or-distrib*: $\llbracket a: bool; b: bool; c: bool \rrbracket \implies$
 $(a \text{ or } b) \text{ and } c = (a \text{ and } c) \text{ or } (b \text{ and } c)$
 $\langle proof \rangle$

6.3 Laws About 'or'

lemma *or-absorb* [*simp*]: $a: bool \implies a \text{ or } a = a$
 $\langle proof \rangle$

lemma *or-commute*: $\llbracket a: bool; b: bool \rrbracket \implies a \text{ or } b = b \text{ or } a$
 $\langle proof \rangle$

lemma *or-assoc*: $a: bool \implies (a \text{ or } b) \text{ or } c = a \text{ or } (b \text{ or } c)$
 $\langle proof \rangle$

lemma *or-and-distrib*: $\llbracket a: bool; b: bool; c: bool \rrbracket \implies$
 $(a \text{ and } b) \text{ or } c = (a \text{ or } c) \text{ and } (b \text{ or } c)$
 $\langle proof \rangle$

definition

$bool\text{-of-}o :: o \Rightarrow i$ **where**
 $bool\text{-of-}o(P) \equiv (if\ P\ then\ 1\ else\ 0)$

lemma [*simp*]: $bool\text{-of-}o(True) = 1$
 $\langle proof \rangle$

lemma [*simp*]: $bool\text{-of-}o(False) = 0$
 $\langle proof \rangle$

lemma [*simp, TC*]: $bool\text{-of-}o(P) \in bool$
 $\langle proof \rangle$

lemma [*simp*]: $(bool\text{-of-}o(P) = 1) \longleftrightarrow P$
 $\langle proof \rangle$

lemma [*simp*]: $(bool\text{-of-}o(P) = 0) \longleftrightarrow \neg P$
 $\langle proof \rangle$

end

7 Disjoint Sums

theory *Sum* **imports** *Bool equalities* **begin**

And the "Part" primitive for simultaneous recursive type definitions

definition *sum* :: $[i, i] \Rightarrow i$ (**infixr** $\langle + \rangle$ 65) **where**

$$A+B \equiv \{0\} * A \cup \{1\} * B$$

definition $Inl :: i \Rightarrow i$ **where**

$$Inl(a) \equiv \langle 0, a \rangle$$

definition $Inr :: i \Rightarrow i$ **where**

$$Inr(b) \equiv \langle 1, b \rangle$$

definition $case :: [i \Rightarrow i, i \Rightarrow i, i] \Rightarrow i$ **where**

$$case(c, d) \equiv (\lambda \langle y, z \rangle. cond(y, d(z), c(z)))$$

definition $Part :: [i, i \Rightarrow i] \Rightarrow i$ **where**

$$Part(A, h) \equiv \{x \in A. \exists z. x = h(z)\}$$

7.1 Rules for the *Part* Primitive

lemma *Part-iff*:

$$a \in Part(A, h) \longleftrightarrow a \in A \wedge (\exists y. a = h(y))$$

<proof>

lemma *Part-eqI* [*intro*]:

$$\llbracket a \in A; a = h(b) \rrbracket \Longrightarrow a \in Part(A, h)$$

<proof>

lemmas *PartI* = *refl* [*THEN* [2] *Part-eqI*]

lemma *PartE* [*elim*!]:

$$\llbracket a \in Part(A, h); \bigwedge z. \llbracket a \in A; a = h(z) \rrbracket \Longrightarrow P \rrbracket \Longrightarrow P$$

<proof>

lemma *Part-subset*: $Part(A, h) \subseteq A$

<proof>

7.2 Rules for Disjoint Sums

lemmas *sum-defs* = *sum-def Inl-def Inr-def case-def*

lemma *Sigma-bool*: $Sigma(bool, C) = C(0) + C(1)$

<proof>

lemma *InlI* [*intro!*, *simp*, *TC*]: $a \in A \Longrightarrow Inl(a) \in A+B$

<proof>

lemma *InrI* [*intro!*, *simp*, *TC*]: $b \in B \Longrightarrow Inr(b) \in A+B$

<proof>

lemma *sumE* [*elim!*]:

$\llbracket u \in A+B;$
 $\quad \bigwedge x. \llbracket x \in A; \ u=Inl(x) \rrbracket \implies P;$
 $\quad \bigwedge y. \llbracket y \in B; \ u=Inr(y) \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle proof \rangle$

lemma *Inl-iff* [*iff*]: $Inl(a)=Inl(b) \longleftrightarrow a=b$
 $\langle proof \rangle$

lemma *Inr-iff* [*iff*]: $Inr(a)=Inr(b) \longleftrightarrow a=b$
 $\langle proof \rangle$

lemma *Inl-Inr-iff* [*simp*]: $Inl(a)=Inr(b) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *Inr-Inl-iff* [*simp*]: $Inr(b)=Inl(a) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *sum-empty* [*simp*]: $0+0 = 0$
 $\langle proof \rangle$

lemmas *Inl-inject* = *Inl-iff* [*THEN iffD1*]

lemmas *Inr-inject* = *Inr-iff* [*THEN iffD1*]

lemmas *Inl-neq-Inr* = *Inl-Inr-iff* [*THEN iffD1, THEN FalseE, elim!*]

lemmas *Inr-neq-Inl* = *Inr-Inl-iff* [*THEN iffD1, THEN FalseE, elim!*]

lemma *InlD*: $Inl(a): A+B \implies a \in A$
 $\langle proof \rangle$

lemma *InrD*: $Inr(b): A+B \implies b \in B$
 $\langle proof \rangle$

lemma *sum-iff*: $u \in A+B \longleftrightarrow (\exists x. x \in A \wedge u=Inl(x)) \mid (\exists y. y \in B \wedge u=Inr(y))$
 $\langle proof \rangle$

lemma *Inl-in-sum-iff* [*simp*]: $(Inl(x) \in A+B) \longleftrightarrow (x \in A)$
 $\langle proof \rangle$

lemma *Inr-in-sum-iff* [*simp*]: $(Inr(y) \in A+B) \longleftrightarrow (y \in B)$
 $\langle proof \rangle$

lemma *sum-subset-iff*: $A+B \subseteq C+D \longleftrightarrow A \leq C \wedge B \leq D$
 $\langle proof \rangle$

lemma *sum-equal-iff*: $A+B = C+D \longleftrightarrow A=C \wedge B=D$
 $\langle proof \rangle$

lemma *sum-eq-2-times*: $A+A = 2*A$
 $\langle proof \rangle$

7.3 The Eliminator: *case*

lemma *case-Inl* [*simp*]: $case(c, d, Inl(a)) = c(a)$
 $\langle proof \rangle$

lemma *case-Inr* [*simp*]: $case(c, d, Inr(b)) = d(b)$
 $\langle proof \rangle$

lemma *case-type* [*TC*]:
 $\llbracket u \in A+B;$
 $\quad \bigwedge x. x \in A \implies c(x): C(Inl(x));$
 $\quad \bigwedge y. y \in B \implies d(y): C(Inr(y))$
 $\rrbracket \implies case(c,d,u) \in C(u)$
 $\langle proof \rangle$

lemma *expand-case*: $u \in A+B \implies$
 $R(case(c,d,u)) \longleftrightarrow$
 $((\forall x \in A. u = Inl(x) \longrightarrow R(c(x))) \wedge$
 $(\forall y \in B. u = Inr(y) \longrightarrow R(d(y))))$
 $\langle proof \rangle$

lemma *case-cong*:
 $\llbracket z \in A+B;$
 $\quad \bigwedge x. x \in A \implies c(x)=c'(x);$
 $\quad \bigwedge y. y \in B \implies d(y)=d'(y)$
 $\rrbracket \implies case(c,d,z) = case(c',d',z)$
 $\langle proof \rangle$

lemma *case-case*: $z \in A+B \implies$
 $case(c, d, case(\lambda x. Inl(c'(x)), \lambda y. Inr(d'(y)), z)) =$
 $case(\lambda x. c(c'(x)), \lambda y. d(d'(y)), z)$
 $\langle proof \rangle$

7.4 More Rules for *Part*(*A*, *h*)

lemma *Part-mono*: $A \leq B \implies Part(A,h) \leq Part(B,h)$
 $\langle proof \rangle$

lemma *Part-Collect*: $Part(Collect(A,P), h) = Collect(Part(A,h), P)$
 $\langle proof \rangle$

lemmas *Part-CollectE* =
 Part-Collect [*THEN equalityD1*, *THEN subsetD*, *THEN CollectE*]

lemma *Part-Inl*: $\text{Part}(A+B, \text{Inl}) = \{\text{Inl}(x). x \in A\}$
 $\langle \text{proof} \rangle$

lemma *Part-Inr*: $\text{Part}(A+B, \text{Inr}) = \{\text{Inr}(y). y \in B\}$
 $\langle \text{proof} \rangle$

lemma *PartD1*: $a \in \text{Part}(A, h) \implies a \in A$
 $\langle \text{proof} \rangle$

lemma *Part-id*: $\text{Part}(A, \lambda x. x) = A$
 $\langle \text{proof} \rangle$

lemma *Part-Inr2*: $\text{Part}(A+B, \lambda x. \text{Inr}(h(x))) = \{\text{Inr}(y). y \in \text{Part}(B, h)\}$
 $\langle \text{proof} \rangle$

lemma *Part-sum-equality*: $C \subseteq A+B \implies \text{Part}(C, \text{Inl}) \cup \text{Part}(C, \text{Inr}) = C$
 $\langle \text{proof} \rangle$

end

8 Functions, Function Spaces, Lambda-Abstraction

theory *func* **imports** *equalities Sum* **begin**

8.1 The Pi Operator: Dependent Function Space

lemma *subset-Sigma-imp-relation*: $r \subseteq \text{Sigma}(A, B) \implies \text{relation}(r)$
 $\langle \text{proof} \rangle$

lemma *relation-converse-converse* [*simp*]:
 $\text{relation}(r) \implies \text{converse}(\text{converse}(r)) = r$
 $\langle \text{proof} \rangle$

lemma *relation-restrict* [*simp*]: $\text{relation}(\text{restrict}(r, A))$
 $\langle \text{proof} \rangle$

lemma *Pi-iff*:
 $f \in \text{Pi}(A, B) \longleftrightarrow \text{function}(f) \wedge f \leq \text{Sigma}(A, B) \wedge A \leq \text{domain}(f)$
 $\langle \text{proof} \rangle$

lemma *Pi-iff-old*:
 $f \in \text{Pi}(A, B) \longleftrightarrow f \leq \text{Sigma}(A, B) \wedge (\forall x \in A. \exists! y. \langle x, y \rangle: f)$
 $\langle \text{proof} \rangle$

lemma *fun-is-function*: $f \in \text{Pi}(A, B) \implies \text{function}(f)$

$\langle proof \rangle$

lemma *function-imp-Pi*:

$\llbracket function(f); relation(f) \rrbracket \implies f \in domain(f) \rightarrow range(f)$
 $\langle proof \rangle$

lemma *functionI*:

$\llbracket \bigwedge x \ y \ y'. \llbracket \langle x, y \rangle : r; \langle x, y' \rangle : r \rrbracket \implies y = y' \rrbracket \implies function(r)$
 $\langle proof \rangle$

lemma *fun-is-rel*: $f \in Pi(A, B) \implies f \subseteq Sigma(A, B)$

$\langle proof \rangle$

lemma *Pi-cong*:

$\llbracket A = A'; \bigwedge x. x \in A' \implies B(x) = B'(x) \rrbracket \implies Pi(A, B) = Pi(A', B')$
 $\langle proof \rangle$

lemma *fun-weaken-type*: $\llbracket f \in A \rightarrow B; B \leq D \rrbracket \implies f \in A \rightarrow D$

$\langle proof \rangle$

8.2 Function Application

lemma *apply-equality2*: $\llbracket \langle a, b \rangle : f; \langle a, c \rangle : f; f \in Pi(A, B) \rrbracket \implies b = c$

$\langle proof \rangle$

lemma *function-apply-equality*: $\llbracket \langle a, b \rangle : f; function(f) \rrbracket \implies f'a = b$

$\langle proof \rangle$

lemma *apply-equality*: $\llbracket \langle a, b \rangle : f; f \in Pi(A, B) \rrbracket \implies f'a = b$

$\langle proof \rangle$

lemma *apply-0*: $a \notin domain(f) \implies f'a = 0$

$\langle proof \rangle$

lemma *Pi-memberD*: $\llbracket f \in Pi(A, B); c \in f \rrbracket \implies \exists x \in A. c = \langle x, f'x \rangle$

$\langle proof \rangle$

lemma *function-apply-Pair*: $\llbracket function(f); a \in domain(f) \rrbracket \implies \langle a, f'a \rangle : f$

$\langle proof \rangle$

lemma *apply-Pair*: $\llbracket f \in Pi(A, B); a \in A \rrbracket \implies \langle a, f'a \rangle : f$

$\langle proof \rangle$

lemma *apply-type* [TC]: $\llbracket f \in Pi(A,B); a \in A \rrbracket \implies f'a \in B(a)$
 $\langle proof \rangle$

lemma *apply-funtype*: $\llbracket f \in A \multimap B; a \in A \rrbracket \implies f'a \in B$
 $\langle proof \rangle$

lemma *apply-iff*: $f \in Pi(A,B) \implies \langle a,b \rangle: f \longleftrightarrow a \in A \wedge f'a = b$
 $\langle proof \rangle$

lemma *Pi-type*: $\llbracket f \in Pi(A,C); \bigwedge x. x \in A \implies f'x \in B(x) \rrbracket \implies f \in Pi(A,B)$
 $\langle proof \rangle$

lemma *Pi-Collect-iff*:
 $(f \in Pi(A, \lambda x. \{y \in B(x). P(x,y)\}))$
 $\longleftrightarrow f \in Pi(A,B) \wedge (\forall x \in A. P(x, f'x))$
 $\langle proof \rangle$

lemma *Pi-weaken-type*:
 $\llbracket f \in Pi(A,B); \bigwedge x. x \in A \implies B(x) \leq C(x) \rrbracket \implies f \in Pi(A,C)$
 $\langle proof \rangle$

lemma *domain-type*: $\llbracket \langle a,b \rangle \in f; f \in Pi(A,B) \rrbracket \implies a \in A$
 $\langle proof \rangle$

lemma *range-type*: $\llbracket \langle a,b \rangle \in f; f \in Pi(A,B) \rrbracket \implies b \in B(a)$
 $\langle proof \rangle$

lemma *Pair-mem-PiD*: $\llbracket \langle a,b \rangle: f; f \in Pi(A,B) \rrbracket \implies a \in A \wedge b \in B(a) \wedge f'a = b$
 $\langle proof \rangle$

8.3 Lambda Abstraction

lemma *lamI*: $a \in A \implies \langle a, b(a) \rangle \in (\lambda x \in A. b(x))$
 $\langle proof \rangle$

lemma *lamE*:
 $\llbracket p: (\lambda x \in A. b(x)); \bigwedge x. \llbracket x \in A; p = \langle x, b(x) \rangle \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle proof \rangle$

lemma *lamD*: $\llbracket \langle a,c \rangle: (\lambda x \in A. b(x)) \rrbracket \implies c = b(a)$
 $\langle proof \rangle$

lemma *lam-type* [TC]:

$\llbracket \bigwedge x. x \in A \implies b(x) : B(x) \rrbracket \implies (\lambda x \in A. b(x)) \in Pi(A, B)$
 $\langle proof \rangle$

lemma *lam-funtype*: $(\lambda x \in A. b(x)) \in A \multimap \{b(x). x \in A\}$
 $\langle proof \rangle$

lemma *function-lam*: *function* $(\lambda x \in A. b(x))$
 $\langle proof \rangle$

lemma *relation-lam*: *relation* $(\lambda x \in A. b(x))$
 $\langle proof \rangle$

lemma *beta-if* [simp]: $(\lambda x \in A. b(x)) \text{ ` } a = (if\ a \in A\ then\ b(a)\ else\ 0)$
 $\langle proof \rangle$

lemma *beta*: $a \in A \implies (\lambda x \in A. b(x)) \text{ ` } a = b(a)$
 $\langle proof \rangle$

lemma *lam-empty* [simp]: $(\lambda x \in 0. b(x)) = 0$
 $\langle proof \rangle$

lemma *domain-lam* [simp]: $domain(Lambda(A, b)) = A$
 $\langle proof \rangle$

lemma *lam-cong* [cong]:

$\llbracket A=A'; \bigwedge x. x \in A' \implies b(x)=b'(x) \rrbracket \implies Lambda(A, b) = Lambda(A', b')$
 $\langle proof \rangle$

lemma *lam-theI*:

$(\bigwedge x. x \in A \implies \exists! y. Q(x, y)) \implies \exists f. \forall x \in A. Q(x, f'x)$
 $\langle proof \rangle$

lemma *lam-eqE*: $\llbracket (\lambda x \in A. f(x)) = (\lambda x \in A. g(x)); a \in A \rrbracket \implies f(a)=g(a)$
 $\langle proof \rangle$

lemma *Pi-empty1* [simp]: $Pi(0, A) = \{0\}$
 $\langle proof \rangle$

lemma *singleton-fun* [simp]: $\{\langle a, b \rangle\} \in \{a\} \multimap \{b\}$
 $\langle proof \rangle$

lemma *Pi-empty2* [simp]: $(A \multimap 0) = (if\ A=0\ then\ \{0\}\ else\ 0)$
 $\langle proof \rangle$

lemma *fun-space-empty-iff* [iff]: $(A \multimap X) = 0 \longleftrightarrow X = 0 \wedge (A \neq 0)$
 <proof>

8.4 Extensionality

lemma *fun-subset*:
 $\llbracket f \in \text{Pi}(A, B); g \in \text{Pi}(C, D); A \leq C;$
 $\bigwedge x. x \in A \implies f'x = g'x \rrbracket \implies f \leq g$
 <proof>

lemma *fun-extension*:
 $\llbracket f \in \text{Pi}(A, B); g \in \text{Pi}(A, D);$
 $\bigwedge x. x \in A \implies f'x = g'x \rrbracket \implies f = g$
 <proof>

lemma *eta* [simp]: $f \in \text{Pi}(A, B) \implies (\lambda x \in A. f'x) = f$
 <proof>

lemma *fun-extension-iff*:
 $\llbracket f \in \text{Pi}(A, B); g \in \text{Pi}(A, C) \rrbracket \implies (\forall a \in A. f'a = g'a) \longleftrightarrow f = g$
 <proof>

lemma *fun-subset-eq*: $\llbracket f \in \text{Pi}(A, B); g \in \text{Pi}(A, C) \rrbracket \implies f \subseteq g \longleftrightarrow (f = g)$
 <proof>

lemma *Pi-lamE*:
 assumes major: $f \in \text{Pi}(A, B)$
 and minor: $\bigwedge b. \llbracket \forall x \in A. b(x):B(x); f = (\lambda x \in A. b(x)) \rrbracket \implies P$
 shows P
 <proof>

8.5 Images of Functions

lemma *image-lam*: $C \subseteq A \implies (\lambda x \in A. b(x)) \text{ `` } C = \{b(x). x \in C\}$
 <proof>

lemma *Repfun-function-if*:
 $\text{function}(f)$
 $\implies \{f'x. x \in C\} = (\text{if } C \subseteq \text{domain}(f) \text{ then } f''C \text{ else } \text{cons}(0, f''C))$
 <proof>

lemma *image-function*:
 $\llbracket \text{function}(f); C \subseteq \text{domain}(f) \rrbracket \implies f''C = \{f'x. x \in C\}$
 <proof>

lemma *image-fun*: $\llbracket f \in \text{Pi}(A, B); C \subseteq A \rrbracket \implies f''C = \{f'x. x \in C\}$

$\langle proof \rangle$

lemma *image-eq-UN*:

assumes $f: f \in Pi(A,B)$ $C \subseteq A$ **shows** $f''C = (\bigcup_{x \in C}. \{f'x\})$
 $\langle proof \rangle$

lemma *Pi-image-cons*:

$\llbracket f \in Pi(A,B); x \in A \rrbracket \implies f''cons(x,y) = cons(f'x, f''y)$
 $\langle proof \rangle$

8.6 Properties of $restrict(f, A)$

lemma *restrict-subset*: $restrict(f,A) \subseteq f$

$\langle proof \rangle$

lemma *function-restrictI*:

$function(f) \implies function(restrict(f,A))$
 $\langle proof \rangle$

lemma *restrict-type2*: $\llbracket f \in Pi(C,B); A \leq C \rrbracket \implies restrict(f,A) \in Pi(A,B)$

$\langle proof \rangle$

lemma *restrict*: $restrict(f,A)'a = (if\ a \in A\ then\ f'a\ else\ 0)$

$\langle proof \rangle$

lemma *restrict-empty [simp]*: $restrict(f,0) = 0$

$\langle proof \rangle$

lemma *restrict-iff*: $z \in restrict(r,A) \longleftrightarrow z \in r \wedge (\exists x \in A. \exists y. z = \langle x, y \rangle)$

$\langle proof \rangle$

lemma *restrict-restrict [simp]*:

$restrict(restrict(r,A),B) = restrict(r, A \cap B)$
 $\langle proof \rangle$

lemma *domain-restrict [simp]*: $domain(restrict(f,C)) = domain(f) \cap C$

$\langle proof \rangle$

lemma *restrict-idem*: $f \subseteq Sigma(A,B) \implies restrict(f,A) = f$

$\langle proof \rangle$

lemma *domain-restrict-idem*:

$\llbracket domain(r) \subseteq A; relation(r) \rrbracket \implies restrict(r,A) = r$
 $\langle proof \rangle$

lemma *domain-restrict-lam [simp]*: $domain(restrict(Lambda(A,f),C)) = A \cap C$

$\langle proof \rangle$

lemma *restrict-if [simp]: restrict(f,A) ‘ a = (if a ∈ A then f‘a else 0)*
 ⟨proof⟩

lemma *restrict-lam-eq:*
 $A \leq C \implies \text{restrict}(\lambda x \in C. b(x), A) = (\lambda x \in A. b(x))$
 ⟨proof⟩

lemma *fun-cons-restrict-eq:*
 $f \in \text{cons}(a, b) \rightarrow B \implies f = \text{cons}(\langle a, f \text{ ‘ } a \rangle, \text{restrict}(f, b))$
 ⟨proof⟩

8.7 Unions of Functions

lemma *function-Union:*

$$\llbracket \forall x \in S. \text{function}(x);$$

$$\forall x \in S. \forall y \in S. x \leq y \mid y \leq x \rrbracket$$

$$\implies \text{function}(\bigcup(S))$$
 ⟨proof⟩

lemma *fun-Union:*

$$\llbracket \forall f \in S. \exists C D. f \in C \rightarrow D;$$

$$\forall f \in S. \forall y \in S. f \leq y \mid y \leq f \rrbracket \implies$$

$$\bigcup(S) \in \text{domain}(\bigcup(S)) \rightarrow \text{range}(\bigcup(S))$$
 ⟨proof⟩

lemma *gen-relation-Union:*

$$(\bigwedge f. f \in F \implies \text{relation}(f)) \implies \text{relation}(\bigcup(F))$$
 ⟨proof⟩

lemmas *Un-rls = Un-subset-iff SUM-Un-distrib1 prod-Un-distrib2*
subset-trans [OF - Un-upper1]
subset-trans [OF - Un-upper2]

lemma *fun-disjoint-Un:*

$$\llbracket f \in A \rightarrow B; g \in C \rightarrow D; A \cap C = \emptyset \rrbracket$$

$$\implies (f \cup g) \in (A \cup C) \rightarrow (B \cup D)$$
 ⟨proof⟩

lemma *fun-disjoint-apply1: a ∉ domain(g) ⟹ (f ∪ g)‘a = f‘a*
 ⟨proof⟩

lemma *fun-disjoint-apply2: c ∉ domain(f) ⟹ (f ∪ g)‘c = g‘c*
 ⟨proof⟩

8.8 Domain and Range of a Function or Relation

lemma *domain-of-fun*: $f \in Pi(A,B) \implies domain(f)=A$
 $\langle proof \rangle$

lemma *apply-rangeI*: $\llbracket f \in Pi(A,B); a \in A \rrbracket \implies f'a \in range(f)$
 $\langle proof \rangle$

lemma *range-of-fun*: $f \in Pi(A,B) \implies f \in A \rightarrow range(f)$
 $\langle proof \rangle$

8.9 Extensions of Functions

lemma *fun-extend*:
 $\llbracket f \in A \rightarrow B; c \notin A \rrbracket \implies cons(\langle c, b \rangle, f) \in cons(c, A) \rightarrow cons(b, B)$
 $\langle proof \rangle$

lemma *fun-extend3*:
 $\llbracket f \in A \rightarrow B; c \notin A; b \in B \rrbracket \implies cons(\langle c, b \rangle, f) \in cons(c, A) \rightarrow B$
 $\langle proof \rangle$

lemma *extend-apply*:
 $c \notin domain(f) \implies cons(\langle c, b \rangle, f)'a = (if\ a=c\ then\ b\ else\ f'a)$
 $\langle proof \rangle$

lemma *fun-extend-apply [simp]*:
 $\llbracket f \in A \rightarrow B; c \notin A \rrbracket \implies cons(\langle c, b \rangle, f)'a = (if\ a=c\ then\ b\ else\ f'a)$
 $\langle proof \rangle$

lemmas *singleton-apply = apply-equality* [OF *singletonI singleton-fun, simp*]

lemma *cons-fun-eq*:
 $c \notin A \implies cons(c, A) \rightarrow B = (\bigcup f \in A \rightarrow B. \bigcup b \in B. \{cons(\langle c, b \rangle, f)\})$
 $\langle proof \rangle$

lemma *succ-fun-eq*: $succ(n) \rightarrow B = (\bigcup f \in n \rightarrow B. \bigcup b \in B. \{cons(\langle n, b \rangle, f)\})$
 $\langle proof \rangle$

8.10 Function Updates

definition

update $:: [i, i, i] \Rightarrow i$ **where**
 $update(f, a, b) \equiv \lambda x \in cons(a, domain(f)). if(x=a, b, f'x)$

nonterminal *updbinds* and *updbind*

syntax

-updbind $:: [i, i] \Rightarrow updbind\ (\langle (\langle indent=2\ notation=\langle infix\ update \rangle \rangle - := / -) \rangle)$
 $:: updbind \Rightarrow updbinds\ (\langle \cdot \rangle)$

$-updbinds :: [updbind, updbinds] \Rightarrow updbinds \ (\langle -, / - \rangle)$
 $-Update :: [i, updbinds] \Rightarrow i \ (\langle \langle open\text{-}block\text{ notation} = \langle mixfix\ function\ up\text{-}date \rangle \rangle - / '((-)') \rangle [900, 0] 900)$

syntax-consts

$-Update \equiv update$

translations

$-Update\ (f, -updbinds(b, bs)) == -Update\ (-Update(f, b), bs)$
 $f(x:=y) == CONST\ update(f, x, y)$

lemma *update-apply* [simp]: $f(x:=y) \text{ ' } z = (if\ z=x\ then\ y\ else\ f'z)$
 $\langle proof \rangle$

lemma *update-idem*: $\llbracket f'x = y; f \in Pi(A, B); x \in A \rrbracket \implies f(x:=y) = f$
 $\langle proof \rangle$

declare *refl* [THEN *update-idem*, *simp*]

lemma *domain-update* [simp]: $domain(f(x:=y)) = cons(x, domain(f))$
 $\langle proof \rangle$

lemma *update-type*: $\llbracket f \in Pi(A, B); x \in A; y \in B(x) \rrbracket \implies f(x:=y) \in Pi(A, B)$
 $\langle proof \rangle$

8.11 Monotonicity Theorems

8.11.1 Replacement in its Various Forms

lemma *Replace-mono*: $A \leq B \implies Replace(A, P) \subseteq Replace(B, P)$
 $\langle proof \rangle$

lemma *RepFun-mono*: $A \leq B \implies \{f(x). x \in A\} \subseteq \{f(x). x \in B\}$
 $\langle proof \rangle$

lemma *Pow-mono*: $A \leq B \implies Pow(A) \subseteq Pow(B)$
 $\langle proof \rangle$

lemma *Union-mono*: $A \leq B \implies \bigcup(A) \subseteq \bigcup(B)$
 $\langle proof \rangle$

lemma *UN-mono*:

$\llbracket A \leq C; \bigwedge x. x \in A \implies B(x) \leq D(x) \rrbracket \implies (\bigcup_{x \in A} B(x)) \subseteq (\bigcup_{x \in C} D(x))$
 $\langle proof \rangle$

lemma *Inter-anti-mono*: $\llbracket A \leq B; A \neq 0 \rrbracket \implies \bigcap(B) \subseteq \bigcap(A)$
 $\langle proof \rangle$

lemma *cons-mono*: $C \leq D \implies cons(a, C) \subseteq cons(a, D)$

$\langle proof \rangle$

lemma *Un-mono*: $\llbracket A \leq C; B \leq D \rrbracket \implies A \cup B \subseteq C \cup D$
 $\langle proof \rangle$

lemma *Int-mono*: $\llbracket A \leq C; B \leq D \rrbracket \implies A \cap B \subseteq C \cap D$
 $\langle proof \rangle$

lemma *Diff-mono*: $\llbracket A \leq C; D \leq B \rrbracket \implies A - B \subseteq C - D$
 $\langle proof \rangle$

8.11.2 Standard Products, Sums and Function Spaces

lemma *Sigma-mono* [rule-format]:
 $\llbracket A \leq C; \bigwedge x. x \in A \implies B(x) \subseteq D(x) \rrbracket \implies \text{Sigma}(A, B) \subseteq \text{Sigma}(C, D)$
 $\langle proof \rangle$

lemma *sum-mono*: $\llbracket A \leq C; B \leq D \rrbracket \implies A + B \subseteq C + D$
 $\langle proof \rangle$

lemma *Pi-mono*: $B \leq C \implies A \multimap B \subseteq A \multimap C$
 $\langle proof \rangle$

lemma *lam-mono*: $A \leq B \implies \text{Lambda}(A, c) \subseteq \text{Lambda}(B, c)$
 $\langle proof \rangle$

8.11.3 Converse, Domain, Range, Field

lemma *converse-mono*: $r \leq s \implies \text{converse}(r) \subseteq \text{converse}(s)$
 $\langle proof \rangle$

lemma *domain-mono*: $r \leq s \implies \text{domain}(r) \leq \text{domain}(s)$
 $\langle proof \rangle$

lemmas *domain-rel-subset* = *subset-trans* [OF *domain-mono domain-subset*]

lemma *range-mono*: $r \leq s \implies \text{range}(r) \leq \text{range}(s)$
 $\langle proof \rangle$

lemmas *range-rel-subset* = *subset-trans* [OF *range-mono range-subset*]

lemma *field-mono*: $r \leq s \implies \text{field}(r) \leq \text{field}(s)$
 $\langle proof \rangle$

lemma *field-rel-subset*: $r \subseteq A * A \implies \text{field}(r) \subseteq A$
 $\langle proof \rangle$

8.11.4 Images

lemma *image-pair-mono*:

$\llbracket \bigwedge x y. \langle x, y \rangle : r \implies \langle x, y \rangle : s; A \leq B \rrbracket \implies r''A \subseteq s''B$
 $\langle \text{proof} \rangle$

lemma *vimage-pair-mono*:

$\llbracket \bigwedge x y. \langle x, y \rangle : r \implies \langle x, y \rangle : s; A \leq B \rrbracket \implies r-''A \subseteq s-''B$
 $\langle \text{proof} \rangle$

lemma *image-mono*: $\llbracket r \leq s; A \leq B \rrbracket \implies r''A \subseteq s''B$

$\langle \text{proof} \rangle$

lemma *vimage-mono*: $\llbracket r \leq s; A \leq B \rrbracket \implies r-''A \subseteq s-''B$

$\langle \text{proof} \rangle$

lemma *Collect-mono*:

$\llbracket A \leq B; \bigwedge x. x \in A \implies P(x) \longrightarrow Q(x) \rrbracket \implies \text{Collect}(A, P) \subseteq \text{Collect}(B, Q)$
 $\langle \text{proof} \rangle$

lemmas *basic-monos = subset-refl imp-refl disj-mono conj-mono ex-mono*
Collect-mono Part-mono in-mono

lemma *bex-image-simp*:

$\llbracket f \in \text{Pi}(X, Y); A \subseteq X \rrbracket \implies (\exists x \in f''A. P(x)) \longleftrightarrow (\exists x \in A. P(f'x))$
 $\langle \text{proof} \rangle$

lemma *ball-image-simp*:

$\llbracket f \in \text{Pi}(X, Y); A \subseteq X \rrbracket \implies (\forall x \in f''A. P(x)) \longleftrightarrow (\forall x \in A. P(f'x))$
 $\langle \text{proof} \rangle$

end

9 Quine-Inspired Ordered Pairs and Disjoint Sums

theory *QPair* **imports** *Sum func* **begin**

For non-well-founded data structures in ZF. Does not precisely follow Quine's construction. Thanks to Thomas Forster for suggesting this approach!

W. V. Quine, On Ordered Pairs and Relations, in Selected Logic Papers, 1966.

definition

QPair :: $[i, i] \Rightarrow i \ (\langle \langle \text{indent}=1 \text{ notation}=\langle \text{mixfix Quine pair} \rangle \rangle \langle -; / - \rangle)$
where $\langle a; b \rangle \equiv a + b$

definition

$qfst :: i \Rightarrow i$ **where**
 $qfst(p) \equiv THE\ a.\ \exists\ b.\ p = \langle a; b \rangle$

definition

$qsnd :: i \Rightarrow i$ **where**
 $qsnd(p) \equiv THE\ b.\ \exists\ a.\ p = \langle a; b \rangle$

definition

$qsplit :: [[i, i] \Rightarrow 'a, i] \Rightarrow 'a::\{\}$ **where**
 $qsplit(c, p) \equiv c(qfst(p), qsnd(p))$

definition

$qconverse :: i \Rightarrow i$ **where**
 $qconverse(r) \equiv \{z.\ w \in r,\ \exists\ x\ y.\ w = \langle x; y \rangle \wedge z = \langle y; x \rangle\}$

definition

$QSigma :: [i, i \Rightarrow i] \Rightarrow i$ **where**
 $QSigma(A, B) \equiv \bigcup_{x \in A} \bigcup_{y \in B(x)} \{\langle x; y \rangle\}$

syntax

$-QSUM :: [idt, i, i] \Rightarrow i$ ($\langle \langle indent=3\ notation=\langle binder\ QSUM \rangle \rangle QSUM - \in$
 $-./\ - \rangle 10$)

syntax-consts

$-QSUM \Leftarrow QSigma$

translations

$QSUM\ x \in A.\ B \Rightarrow CONST\ QSigma(A, \lambda x.\ B)$

abbreviation

$qprod$ (**infixr** $\langle \langle * \rangle \rangle 80$) **where**
 $A \langle * \rangle B \equiv QSigma(A, \lambda -. B)$

definition

$qsum :: [i, i] \Rightarrow i$ (**infixr** $\langle \langle + \rangle \rangle 65$) **where**
 $A \langle + \rangle B \equiv (\{0\} \langle * \rangle A) \cup (\{1\} \langle * \rangle B)$

definition

$QInl :: i \Rightarrow i$ **where**
 $QInl(a) \equiv \langle 0; a \rangle$

definition

$QInr :: i \Rightarrow i$ **where**
 $QInr(b) \equiv \langle 1; b \rangle$

definition

$qcase :: [i \Rightarrow i, i \Rightarrow i, i] \Rightarrow i$ **where**
 $qcase(c, d) \equiv qsplit(\lambda y\ z.\ cond(y, d(z), c(z)))$

9.1 Quine ordered pairing

lemma *QPair-empty* [simp]: $\langle 0; 0 \rangle = 0$
 $\langle proof \rangle$

lemma *QPair-iff* [simp]: $\langle a; b \rangle = \langle c; d \rangle \longleftrightarrow a=c \wedge b=d$
 $\langle proof \rangle$

lemmas *QPair-inject* = *QPair-iff* [THEN *iffD1*, THEN *conjE*, *elim!*]

lemma *QPair-inject1*: $\langle a; b \rangle = \langle c; d \rangle \implies a=c$
 $\langle proof \rangle$

lemma *QPair-inject2*: $\langle a; b \rangle = \langle c; d \rangle \implies b=d$
 $\langle proof \rangle$

9.1.1 QSigma: Disjoint union of a family of sets Generalizes Cartesian product

lemma *QSigmaI* [intro!]: $\llbracket a \in A; b \in B(a) \rrbracket \implies \langle a; b \rangle \in QSigma(A, B)$
 $\langle proof \rangle$

lemma *QSigmaE* [elim!]:
 $\llbracket c \in QSigma(A, B);$
 $\bigwedge x y. \llbracket x \in A; y \in B(x); c = \langle x; y \rangle \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle proof \rangle$

lemma *QSigmaE2* [elim!]:
 $\llbracket \langle a; b \rangle \in QSigma(A, B); \llbracket a \in A; b \in B(a) \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *QSigmaD1*: $\langle a; b \rangle \in QSigma(A, B) \implies a \in A$
 $\langle proof \rangle$

lemma *QSigmaD2*: $\langle a; b \rangle \in QSigma(A, B) \implies b \in B(a)$
 $\langle proof \rangle$

lemma *QSigma-cong*:
 $\llbracket A=A'; \bigwedge x. x \in A' \implies B(x)=B'(x) \rrbracket \implies$
 $QSigma(A, B) = QSigma(A', B')$
 $\langle proof \rangle$

lemma *QSigma-empty1* [simp]: $QSigma(0, B) = 0$
 $\langle proof \rangle$

lemma *QSigma-empty2* [simp]: $A \langle * \rangle 0 = 0$

$\langle proof \rangle$

9.1.2 Projections: qfst, qsnd

lemma *qfst-conv* [simp]: $qfst(<a;b>) = a$
 $\langle proof \rangle$

lemma *qsnd-conv* [simp]: $qsnd(<a;b>) = b$
 $\langle proof \rangle$

lemma *qfst-type* [TC]: $p \in QSigma(A,B) \implies qfst(p) \in A$
 $\langle proof \rangle$

lemma *qsnd-type* [TC]: $p \in QSigma(A,B) \implies qsnd(p) \in B(qfst(p))$
 $\langle proof \rangle$

lemma *QPair-qfst-qsnd-eq*: $a \in QSigma(A,B) \implies <qfst(a); qsnd(a)> = a$
 $\langle proof \rangle$

9.1.3 Eliminator: qsplit

lemma *qsplit* [simp]: $qsplit(\lambda x y. c(x,y), <a;b>) \equiv c(a,b)$
 $\langle proof \rangle$

lemma *qsplit-type* [elim!]:
 $\llbracket p \in QSigma(A,B);$
 $\bigwedge x y. \llbracket x \in A; y \in B(x) \rrbracket \implies c(x,y):C(<x;y>)$
 $\rrbracket \implies qsplit(\lambda x y. c(x,y), p) \in C(p)$
 $\langle proof \rangle$

lemma *expand-qsplit*:
 $u \in A <*> B \implies R(qsplit(c,u)) \longleftrightarrow (\forall x \in A. \forall y \in B. u = <x;y> \longrightarrow R(c(x,y)))$
 $\langle proof \rangle$

9.1.4 qsplit for predicates: result type o

lemma *qsplitI*: $R(a,b) \implies qsplit(R, <a;b>)$
 $\langle proof \rangle$

lemma *qsplitE*:
 $\llbracket qsplit(R,z); z \in QSigma(A,B);$
 $\bigwedge x y. \llbracket z = <x;y>; R(x,y) \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle proof \rangle$

lemma *qsplitD*: $qsplit(R, <a;b>) \implies R(a,b)$
 $\langle proof \rangle$

9.1.5 qconverse

lemma *qconverseI* [intro!]: $\langle a;b \rangle : r \implies \langle b;a \rangle : qconverse(r)$
 $\langle proof \rangle$

lemma *qconverseD* [elim!]: $\langle a;b \rangle \in qconverse(r) \implies \langle b;a \rangle \in r$
 $\langle proof \rangle$

lemma *qconverseE* [elim!]:

$$\llbracket yx \in qconverse(r);$$

$$\bigwedge x y. \llbracket yx = \langle y;x \rangle; \langle x;y \rangle : r \rrbracket \implies P$$

$$\rrbracket \implies P$$
 $\langle proof \rangle$

lemma *qconverse-qconverse*: $r \leq QSigma(A,B) \implies qconverse(qconverse(r)) = r$
 $\langle proof \rangle$

lemma *qconverse-type*: $r \subseteq A \langle * \rangle B \implies qconverse(r) \subseteq B \langle * \rangle A$
 $\langle proof \rangle$

lemma *qconverse-prod*: $qconverse(A \langle * \rangle B) = B \langle * \rangle A$
 $\langle proof \rangle$

lemma *qconverse-empty*: $qconverse(0) = 0$
 $\langle proof \rangle$

9.2 The Quine-inspired notion of disjoint sum

lemmas *qsum-defs* = *qsum-def* *QInl-def* *QInr-def* *qcase-def*

lemma *QInlI* [intro!]: $a \in A \implies QInl(a) \in A \langle + \rangle B$
 $\langle proof \rangle$

lemma *QInrI* [intro!]: $b \in B \implies QInr(b) \in A \langle + \rangle B$
 $\langle proof \rangle$

lemma *qsumE* [elim!]:

$$\llbracket u \in A \langle + \rangle B;$$

$$\bigwedge x. \llbracket x \in A; u = QInl(x) \rrbracket \implies P;$$

$$\bigwedge y. \llbracket y \in B; u = QInr(y) \rrbracket \implies P$$

$$\rrbracket \implies P$$
 $\langle proof \rangle$

lemma *QInl-iff* [iff]: $QInl(a) = QInl(b) \longleftrightarrow a = b$
 $\langle proof \rangle$

lemma *QInr-iff* [iff]: $QInr(a) = QInr(b) \longleftrightarrow a = b$
 $\langle proof \rangle$

lemma *QInl-QInr-iff* [simp]: $QInl(a) = QInr(b) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *QInr-QInl-iff* [simp]: $QInr(b) = QInl(a) \longleftrightarrow False$
 $\langle proof \rangle$

lemma *qsum-empty* [simp]: $0 <+> 0 = 0$
 $\langle proof \rangle$

lemmas *QInl-inject* = *QInl-iff* [THEN iffD1]

lemmas *QInr-inject* = *QInr-iff* [THEN iffD1]

lemmas *QInl-neq-QInr* = *QInl-QInr-iff* [THEN iffD1, THEN FalseE, elim!]

lemmas *QInr-neq-QInl* = *QInr-QInl-iff* [THEN iffD1, THEN FalseE, elim!]

lemma *QInlD*: $QInl(a): A <+> B \implies a \in A$
 $\langle proof \rangle$

lemma *QInrD*: $QInr(b): A <+> B \implies b \in B$
 $\langle proof \rangle$

lemma *qsum-iff*:

$u \in A <+> B \longleftrightarrow (\exists x. x \in A \wedge u = QInl(x)) \mid (\exists y. y \in B \wedge u = QInr(y))$
 $\langle proof \rangle$

lemma *qsum-subset-iff*: $A <+> B \subseteq C <+> D \longleftrightarrow A \leq C \wedge B \leq D$
 $\langle proof \rangle$

lemma *qsum-equal-iff*: $A <+> B = C <+> D \longleftrightarrow A = C \wedge B = D$
 $\langle proof \rangle$

9.2.1 Eliminator – qcase

lemma *qcase-QInl* [simp]: $qcase(c, d, QInl(a)) = c(a)$
 $\langle proof \rangle$

lemma *qcase-QInr* [simp]: $qcase(c, d, QInr(b)) = d(b)$
 $\langle proof \rangle$

lemma *qcase-type*:

$\llbracket u \in A <+> B; \quad$
 $\quad \bigwedge x. x \in A \implies c(x): C(QInl(x));$
 $\quad \bigwedge y. y \in B \implies d(y): C(QInr(y))$
 $\rrbracket \implies qcase(c,d,u) \in C(u)$
 $\langle proof \rangle$

lemma *Part-QInl*: $Part(A <+> B, QInl) = \{QInl(x). x \in A\}$
 $\langle proof \rangle$

lemma *Part-QInr*: $Part(A <+> B, QInr) = \{QInr(y). y \in B\}$
 $\langle proof \rangle$

lemma *Part-QInr2*: $Part(A <+> B, \lambda x. QInr(h(x))) = \{QInr(y). y \in Part(B, h)\}$
 $\langle proof \rangle$

lemma *Part-qsum-equality*: $C \subseteq A <+> B \implies Part(C, QInl) \cup Part(C, QInr) = C$
 $\langle proof \rangle$

9.2.2 Monotonicity

lemma *QPair-mono*: $\llbracket a \leq c; \quad b \leq d \rrbracket \implies \langle a; b \rangle \subseteq \langle c; d \rangle$
 $\langle proof \rangle$

lemma *QSigma-mono* [rule-format]:
 $\llbracket A \leq C; \quad \forall x \in A. B(x) \subseteq D(x) \rrbracket \implies QSigma(A, B) \subseteq QSigma(C, D)$
 $\langle proof \rangle$

lemma *QInl-mono*: $a \leq b \implies QInl(a) \subseteq QInl(b)$
 $\langle proof \rangle$

lemma *QInr-mono*: $a \leq b \implies QInr(a) \subseteq QInr(b)$
 $\langle proof \rangle$

lemma *qsum-mono*: $\llbracket A \leq C; \quad B \leq D \rrbracket \implies A <+> B \subseteq C <+> D$
 $\langle proof \rangle$

end

10 Injections, Surjections, Bijections, Composition

theory *Perm* **imports** *func* **begin**

definition

comp $:: [i, i] \Rightarrow i \quad (\textbf{infixr } \langle O \rangle \ 60) \quad \textbf{where}$

$$r \circ s \equiv \{xz \in \text{domain}(s) * \text{range}(r) \mid \exists y. z = \langle x, y \rangle \wedge \langle x, y \rangle : s \wedge \langle y, z \rangle : r\}$$

definition

$$\begin{aligned} id &:: i \Rightarrow i \text{ where} \\ id(A) &\equiv (\lambda x \in A. x) \end{aligned}$$

definition

$$\begin{aligned} inj &:: [i, i] \Rightarrow i \text{ where} \\ inj(A, B) &\equiv \{ f \in A \multimap B \mid \forall w \in A. \forall x \in A. f'w = f'x \longrightarrow w = x \} \end{aligned}$$

definition

$$\begin{aligned} surj &:: [i, i] \Rightarrow i \text{ where} \\ surj(A, B) &\equiv \{ f \in A \multimap B \mid \forall y \in B. \exists x \in A. f'x = y \} \end{aligned}$$

definition

$$\begin{aligned} bij &:: [i, i] \Rightarrow i \text{ where} \\ bij(A, B) &\equiv inj(A, B) \cap surj(A, B) \end{aligned}$$

10.1 Surjective Function Space

lemma *surj-is-fun*: $f \in surj(A, B) \Longrightarrow f \in A \multimap B$
<proof>

lemma *fun-is-surj*: $f \in Pi(A, B) \Longrightarrow f \in surj(A, \text{range}(f))$
<proof>

lemma *surj-range*: $f \in surj(A, B) \Longrightarrow \text{range}(f) = B$
<proof>

A function with a right inverse is a surjection

lemma *f-imp-surjective*:

$$\begin{aligned} &\llbracket f \in A \multimap B; \bigwedge y. y \in B \Longrightarrow d(y) : A; \bigwedge y. y \in B \Longrightarrow f'd(y) = y \rrbracket \\ &\Longrightarrow f \in surj(A, B) \end{aligned}$$

<proof>

lemma *lam-surjective*:

$$\begin{aligned} &\llbracket \bigwedge x. x \in A \Longrightarrow c(x) : B; \\ &\quad \bigwedge y. y \in B \Longrightarrow d(y) : A; \\ &\quad \bigwedge y. y \in B \Longrightarrow c(d(y)) = y \rrbracket \\ &\Longrightarrow (\lambda x \in A. c(x)) \in surj(A, B) \end{aligned}$$

<proof>

Cantor's theorem revisited

lemma *cantor-surj*: $f \notin surj(A, \text{Pow}(A))$

$\langle proof \rangle$

10.2 Injective Function Space

lemma *inj-is-fun*: $f \in inj(A,B) \implies f \in A \multimap B$
 $\langle proof \rangle$

Good for dealing with sets of pairs, but a bit ugly in use [used in AC]

lemma *inj-equality*:
 $\llbracket \langle a,b \rangle : f; \langle c,b \rangle : f; f \in inj(A,B) \rrbracket \implies a=c$
 $\langle proof \rangle$

lemma *inj-apply-equality*: $\llbracket f \in inj(A,B); f'a=f'b; a \in A; b \in A \rrbracket \implies a=b$
 $\langle proof \rangle$

A function with a left inverse is an injection

lemma *f-imp-injective*: $\llbracket f \in A \multimap B; \forall x \in A. d(f'x)=x \rrbracket \implies f \in inj(A,B)$
 $\langle proof \rangle$

lemma *lam-injective*:
 $\llbracket \bigwedge x. x \in A \implies c(x) : B;$
 $\bigwedge x. x \in A \implies d(c(x)) = x \rrbracket$
 $\implies (\lambda x \in A. c(x)) \in inj(A,B)$
 $\langle proof \rangle$

10.3 Bijections

lemma *bij-is-inj*: $f \in bij(A,B) \implies f \in inj(A,B)$
 $\langle proof \rangle$

lemma *bij-is-surj*: $f \in bij(A,B) \implies f \in surj(A,B)$
 $\langle proof \rangle$

lemma *bij-is-fun*: $f \in bij(A,B) \implies f \in A \multimap B$
 $\langle proof \rangle$

lemma *lam-bijective*:
 $\llbracket \bigwedge x. x \in A \implies c(x) : B;$
 $\bigwedge y. y \in B \implies d(y) : A;$
 $\bigwedge x. x \in A \implies d(c(x)) = x;$
 $\bigwedge y. y \in B \implies c(d(y)) = y$
 $\rrbracket \implies (\lambda x \in A. c(x)) \in bij(A,B)$
 $\langle proof \rangle$

lemma *RepFun-bijective*: $(\forall y \in x. \exists ! y'. f(y') = f(y))$
 $\implies (\lambda z \in \{f(y). y \in x\}. THE y. f(y) = z) \in bij(\{f(y). y \in x\}, x)$
 $\langle proof \rangle$

10.4 Identity Function

lemma *idI* [*intro!*]: $a \in A \implies \langle a, a \rangle \in id(A)$
 $\langle proof \rangle$

lemma *idE* [*elim!*]: $\llbracket p \in id(A); \bigwedge x. \llbracket x \in A; p = \langle x, x \rangle \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *id-type*: $id(A) \in A \multimap A$
 $\langle proof \rangle$

lemma *id-conv* [*simp*]: $x \in A \implies id(A) 'x = x$
 $\langle proof \rangle$

lemma *id-mono*: $A \leq B \implies id(A) \subseteq id(B)$
 $\langle proof \rangle$

lemma *id-subset-inj*: $A \leq B \implies id(A): inj(A, B)$
 $\langle proof \rangle$

lemmas *id-inj* = *subset-refl* [*THEN id-subset-inj*]

lemma *id-surj*: $id(A): surj(A, A)$
 $\langle proof \rangle$

lemma *id-bij*: $id(A): bij(A, A)$
 $\langle proof \rangle$

lemma *subset-iff-id*: $A \subseteq B \longleftrightarrow id(A) \in A \multimap B$
 $\langle proof \rangle$

id as the identity relation

lemma *id-iff* [*simp*]: $\langle x, y \rangle \in id(A) \longleftrightarrow x = y \wedge y \in A$
 $\langle proof \rangle$

10.5 Converse of a Function

lemma *inj-converse-fun*: $f \in inj(A, B) \implies converse(f) \in range(f) \multimap A$
 $\langle proof \rangle$

Equations for *converse(f)*

The premises are equivalent to saying that *f* is injective...

lemma *left-inverse-lemma*:
 $\llbracket f \in A \multimap B; converse(f): C \multimap A; a \in A \rrbracket \implies converse(f) '(f'a) = a$
 $\langle proof \rangle$

lemma *left-inverse* [*simp*]: $\llbracket f \in inj(A, B); a \in A \rrbracket \implies converse(f) '(f'a) = a$
 $\langle proof \rangle$

lemma *left-inverse-eq*:

$\llbracket f \in \text{inj}(A,B); f \text{ ' } x = y; x \in A \rrbracket \implies \text{converse}(f) \text{ ' } y = x$
 $\langle \text{proof} \rangle$

lemmas *left-inverse-bij = bij-is-inj* [THEN *left-inverse*]

lemma *right-inverse-lemma*:

$\llbracket f \in A \multimap B; \text{converse}(f): C \multimap A; b \in C \rrbracket \implies f(\text{converse}(f) \text{ ' } b) = b$
 $\langle \text{proof} \rangle$

lemma *right-inverse* [*simp*]:

$\llbracket f \in \text{inj}(A,B); b \in \text{range}(f) \rrbracket \implies f(\text{converse}(f) \text{ ' } b) = b$
 $\langle \text{proof} \rangle$

lemma *right-inverse-bij*: $\llbracket f \in \text{bij}(A,B); b \in B \rrbracket \implies f(\text{converse}(f) \text{ ' } b) = b$
 $\langle \text{proof} \rangle$

10.6 Converses of Injections, Surjections, Bijections

lemma *inj-converse-inj*: $f \in \text{inj}(A,B) \implies \text{converse}(f): \text{inj}(\text{range}(f), A)$
 $\langle \text{proof} \rangle$

lemma *inj-converse-surj*: $f \in \text{inj}(A,B) \implies \text{converse}(f): \text{surj}(\text{range}(f), A)$
 $\langle \text{proof} \rangle$

Adding this as an intro! rule seems to cause looping

lemma *bij-converse-bij* [TC]: $f \in \text{bij}(A,B) \implies \text{converse}(f): \text{bij}(B,A)$
 $\langle \text{proof} \rangle$

10.7 Composition of Two Relations

The inductive definition package could derive these theorems for $r \circ s$

lemma *compI* [*intro*]: $\llbracket \langle a,b \rangle : s; \langle b,c \rangle : r \rrbracket \implies \langle a,c \rangle \in r \circ s$
 $\langle \text{proof} \rangle$

lemma *compE* [*elim!*]:

$\llbracket xz \in r \circ s;$
 $\bigwedge x y z. \llbracket xz = \langle x,z \rangle; \langle x,y \rangle : s; \langle y,z \rangle : r \rrbracket \implies P \rrbracket$
 $\implies P$
 $\langle \text{proof} \rangle$

lemma *compEpair*:

$\llbracket \langle a,c \rangle \in r \circ s;$
 $\bigwedge y. \llbracket \langle a,y \rangle : s; \langle y,c \rangle : r \rrbracket \implies P \rrbracket$
 $\implies P$
 $\langle \text{proof} \rangle$

lemma *converse-comp*: $\text{converse}(R \circ S) = \text{converse}(S) \circ \text{converse}(R)$

$\langle proof \rangle$

10.8 Domain and Range – see Suppes, Section 3.1

Boyer et al., Set Theory in First-Order Logic, JAR 2 (1986), 287-327

lemma *range-comp*: $range(r \circ s) \subseteq range(r)$

$\langle proof \rangle$

lemma *range-comp-eq*: $domain(r) \subseteq range(s) \implies range(r \circ s) = range(r)$

$\langle proof \rangle$

lemma *domain-comp*: $domain(r \circ s) \subseteq domain(s)$

$\langle proof \rangle$

lemma *domain-comp-eq*: $range(s) \subseteq domain(r) \implies domain(r \circ s) = domain(s)$

$\langle proof \rangle$

lemma *image-comp*: $(r \circ s)''A = r''(s''A)$

$\langle proof \rangle$

lemma *inj-inj-range*: $f \in inj(A, B) \implies f \in inj(A, range(f))$

$\langle proof \rangle$

lemma *inj-bij-range*: $f \in inj(A, B) \implies f \in bij(A, range(f))$

$\langle proof \rangle$

10.9 Other Results

lemma *comp-mono*: $\llbracket r' \leq r; s' \leq s \rrbracket \implies (r' \circ s') \subseteq (r \circ s)$

$\langle proof \rangle$

composition preserves relations

lemma *comp-rel*: $\llbracket s \leq A * B; r \leq B * C \rrbracket \implies (r \circ s) \subseteq A * C$

$\langle proof \rangle$

associative law for composition

lemma *comp-assoc*: $(r \circ s) \circ t = r \circ (s \circ t)$

$\langle proof \rangle$

lemma *left-comp-id*: $r \leq A * B \implies id(B) \circ r = r$

$\langle proof \rangle$

lemma *right-comp-id*: $r \leq A * B \implies r \circ id(A) = r$

$\langle proof \rangle$

10.10 Composition Preserves Functions, Injections, and Surjections

lemma *comp-function*: $\llbracket \text{function}(g); \text{function}(f) \rrbracket \implies \text{function}(f \circ g)$
 $\langle \text{proof} \rangle$

Don't think the premises can be weakened much

lemma *comp-fun*: $\llbracket g \in A \multimap B; f \in B \multimap C \rrbracket \implies (f \circ g) \in A \multimap C$
 $\langle \text{proof} \rangle$

lemma *comp-fun-apply* [*simp*]:
 $\llbracket g \in A \multimap B; a \in A \rrbracket \implies (f \circ g)'a = f'(g'a)$
 $\langle \text{proof} \rangle$

Simplifies compositions of lambda-abstractions

lemma *comp-lam*:
 $\llbracket \bigwedge x. x \in A \implies b(x): B \rrbracket$
 $\implies (\lambda y \in B. c(y)) \circ (\lambda x \in A. b(x)) = (\lambda x \in A. c(b(x)))$
 $\langle \text{proof} \rangle$

lemma *comp-inj*:
 $\llbracket g \in \text{inj}(A, B); f \in \text{inj}(B, C) \rrbracket \implies (f \circ g) \in \text{inj}(A, C)$
 $\langle \text{proof} \rangle$

lemma *comp-surj*:
 $\llbracket g \in \text{surj}(A, B); f \in \text{surj}(B, C) \rrbracket \implies (f \circ g) \in \text{surj}(A, C)$
 $\langle \text{proof} \rangle$

lemma *comp-bij*:
 $\llbracket g \in \text{bij}(A, B); f \in \text{bij}(B, C) \rrbracket \implies (f \circ g) \in \text{bij}(A, C)$
 $\langle \text{proof} \rangle$

10.11 Dual Properties of *inj* and *surj*

Useful for proofs from D Pastre. Automatic theorem proving in set theory. Artificial Intelligence, 10:1–27, 1978.

lemma *comp-mem-injD1*:
 $\llbracket (f \circ g): \text{inj}(A, C); g \in A \multimap B; f \in B \multimap C \rrbracket \implies g \in \text{inj}(A, B)$
 $\langle \text{proof} \rangle$

lemma *comp-mem-injD2*:
 $\llbracket (f \circ g): \text{inj}(A, C); g \in \text{surj}(A, B); f \in B \multimap C \rrbracket \implies f \in \text{inj}(B, C)$
 $\langle \text{proof} \rangle$

lemma *comp-mem-surjD1*:
 $\llbracket (f \circ g): \text{surj}(A, C); g \in A \multimap B; f \in B \multimap C \rrbracket \implies f \in \text{surj}(B, C)$
 $\langle \text{proof} \rangle$

lemma *comp-mem-surjD2*:

$\llbracket (f \circ g): \text{surj}(A,C); g \in A \multimap B; f \in \text{inj}(B,C) \rrbracket \implies g \in \text{surj}(A,B)$
 $\langle \text{proof} \rangle$

10.11.1 Inverses of Composition

left inverse of composition; one inclusion is $f \in A \rightarrow B \implies \text{id}(A) \subseteq \text{converse}(f) \circ f$

lemma *left-comp-inverse*: $f \in \text{inj}(A,B) \implies \text{converse}(f) \circ f = \text{id}(A)$
 $\langle \text{proof} \rangle$

right inverse of composition; one inclusion is $f \in A \rightarrow B \implies f \circ \text{converse}(f) \subseteq \text{id}(B)$

lemma *right-comp-inverse*:
 $f \in \text{surj}(A,B) \implies f \circ \text{converse}(f) = \text{id}(B)$
 $\langle \text{proof} \rangle$

10.11.2 Proving that a Function is a Bijection

lemma *comp-eq-id-iff*:
 $\llbracket f \in A \multimap B; g \in B \multimap A \rrbracket \implies f \circ g = \text{id}(B) \longleftrightarrow (\forall y \in B. f(g'y) = y)$
 $\langle \text{proof} \rangle$

lemma *fg-imp-bijective*:
 $\llbracket f \in A \multimap B; g \in B \multimap A; f \circ g = \text{id}(B); g \circ f = \text{id}(A) \rrbracket \implies f \in \text{bij}(A,B)$
 $\langle \text{proof} \rangle$

lemma *nilpotent-imp-bijective*: $\llbracket f \in A \multimap A; f \circ f = \text{id}(A) \rrbracket \implies f \in \text{bij}(A,A)$
 $\langle \text{proof} \rangle$

lemma *invertible-imp-bijective*:
 $\llbracket \text{converse}(f): B \multimap A; f \in A \multimap B \rrbracket \implies f \in \text{bij}(A,B)$
 $\langle \text{proof} \rangle$

10.11.3 Unions of Functions

See similar theorems in `func.thy`

Theorem by KG, proof by LCP

lemma *inj-disjoint-Un*:
 $\llbracket f \in \text{inj}(A,B); g \in \text{inj}(C,D); B \cap D = 0 \rrbracket$
 $\implies (\lambda a \in A \cup C. \text{if } a \in A \text{ then } f'a \text{ else } g'a) \in \text{inj}(A \cup C, B \cup D)$
 $\langle \text{proof} \rangle$

lemma *surj-disjoint-Un*:
 $\llbracket f \in \text{surj}(A,B); g \in \text{surj}(C,D); A \cap C = 0 \rrbracket$

$\implies (f \cup g) \in \text{surj}(A \cup C, B \cup D)$
 $\langle \text{proof} \rangle$

A simple, high-level proof; the version for injections follows from it, using $f \in \text{inj}(A, B) \iff f \in \text{bij}(A, \text{range}(f))$

lemma *bij-disjoint-Un*:

$\llbracket f \in \text{bij}(A, B); g \in \text{bij}(C, D); A \cap C = \emptyset; B \cap D = \emptyset \rrbracket$
 $\implies (f \cup g) \in \text{bij}(A \cup C, B \cup D)$
 $\langle \text{proof} \rangle$

10.11.4 Restrictions as Surjections and Bijections

lemma *surj-image*:

$f \in \text{Pi}(A, B) \implies f \in \text{surj}(A, f''A)$
 $\langle \text{proof} \rangle$

lemma *surj-image-eq*: $f \in \text{surj}(A, B) \implies f''A = B$
 $\langle \text{proof} \rangle$

lemma *restrict-image [simp]*: $\text{restrict}(f, A)''B = f''(A \cap B)$
 $\langle \text{proof} \rangle$

lemma *restrict-inj*:

$\llbracket f \in \text{inj}(A, B); C \leq A \rrbracket \implies \text{restrict}(f, C) \in \text{inj}(C, B)$
 $\langle \text{proof} \rangle$

lemma *restrict-surj*: $\llbracket f \in \text{Pi}(A, B); C \leq A \rrbracket \implies \text{restrict}(f, C) \in \text{surj}(C, f''C)$
 $\langle \text{proof} \rangle$

lemma *restrict-bij*:

$\llbracket f \in \text{inj}(A, B); C \leq A \rrbracket \implies \text{restrict}(f, C) \in \text{bij}(C, f''C)$
 $\langle \text{proof} \rangle$

10.11.5 Lemmas for Ramsey's Theorem

lemma *inj-weaken-type*: $\llbracket f \in \text{inj}(A, B); B \leq D \rrbracket \implies f \in \text{inj}(A, D)$
 $\langle \text{proof} \rangle$

lemma *inj-succ-restrict*:

$\llbracket f \in \text{inj}(\text{succ}(m), A) \rrbracket \implies \text{restrict}(f, m) \in \text{inj}(m, A - \{f'm\})$
 $\langle \text{proof} \rangle$

lemma *inj-extend*:

$\llbracket f \in \text{inj}(A, B); a \notin A; b \notin B \rrbracket$
 $\implies \text{cons}(\langle a, b \rangle, f) \in \text{inj}(\text{cons}(a, A), \text{cons}(b, B))$
 $\langle \text{proof} \rangle$

end

11 Relations: Their General Properties and Transitive Closure

theory *Trancl* **imports** *Fixedpt Perm* **begin**

definition

refl :: $[i,i] \Rightarrow o$ **where**
 $refl(A,r) \equiv (\forall x \in A. \langle x,x \rangle \in r)$

definition

irrefl :: $[i,i] \Rightarrow o$ **where**
 $irrefl(A,r) \equiv \forall x \in A. \langle x,x \rangle \notin r$

definition

sym :: $i \Rightarrow o$ **where**
 $sym(r) \equiv \forall x y. \langle x,y \rangle : r \longrightarrow \langle y,x \rangle : r$

definition

asym :: $i \Rightarrow o$ **where**
 $asym(r) \equiv \forall x y. \langle x,y \rangle : r \longrightarrow \neg \langle y,x \rangle : r$

definition

antisym :: $i \Rightarrow o$ **where**
 $antisym(r) \equiv \forall x y. \langle x,y \rangle : r \longrightarrow \langle y,x \rangle : r \longrightarrow x=y$

definition

trans :: $i \Rightarrow o$ **where**
 $trans(r) \equiv \forall x y z. \langle x,y \rangle : r \longrightarrow \langle y,z \rangle : r \longrightarrow \langle x,z \rangle : r$

definition

trans-on :: $[i,i] \Rightarrow o$ ($\langle \langle \langle \text{open-block notation} = \langle \text{mixfix trans-on} \rangle \rangle \text{trans}[-]'(-') \rangle \rangle$) **where**
 $trans[A](r) \equiv \forall x \in A. \forall y \in A. \forall z \in A. \langle x,y \rangle : r \longrightarrow \langle y,z \rangle : r \longrightarrow \langle x,z \rangle : r$

definition

rtrancl :: $i \Rightarrow i$ ($\langle \langle \langle \text{notation} = \langle \text{postfix } \hat{*} \rangle \rangle - \hat{*} \rangle \rangle [100] 100$) **where**
 $r\hat{*} \equiv lfp(field(r)*field(r), \lambda s. id(field(r)) \cup (r \circ s))$

definition

trancl :: $i \Rightarrow i$ ($\langle \langle \langle \text{notation} = \langle \text{postfix } \hat{+} \rangle \rangle - \hat{+} \rangle \rangle [100] 100$) **where**
 $r\hat{+} \equiv r \circ r\hat{*}$

definition

equiv :: $[i,i] \Rightarrow o$ **where**
 $equiv(A,r) \equiv r \subseteq A*A \wedge refl(A,r) \wedge sym(r) \wedge trans(r)$

11.1 General properties of relations

11.1.1 irreflexivity

lemma *irreflI*:

$\llbracket \bigwedge x. x \in A \implies \langle x, x \rangle \notin r \rrbracket \implies \text{irrefl}(A, r)$
<proof>

lemma *irreflE*: $\llbracket \text{irrefl}(A, r); x \in A \rrbracket \implies \langle x, x \rangle \notin r$
<proof>

11.1.2 symmetry

lemma *symI*:

$\llbracket \bigwedge x y. \langle x, y \rangle : r \implies \langle y, x \rangle : r \rrbracket \implies \text{sym}(r)$
<proof>

lemma *symE*: $\llbracket \text{sym}(r); \langle x, y \rangle : r \rrbracket \implies \langle y, x \rangle : r$
<proof>

11.1.3 antisymmetry

lemma *antisymI*:

$\llbracket \bigwedge x y. \llbracket \langle x, y \rangle : r; \langle y, x \rangle : r \rrbracket \implies x = y \rrbracket \implies \text{antisym}(r)$
<proof>

lemma *antisymE*: $\llbracket \text{antisym}(r); \langle x, y \rangle : r; \langle y, x \rangle : r \rrbracket \implies x = y$
<proof>

11.1.4 transitivity

lemma *transD*: $\llbracket \text{trans}(r); \langle a, b \rangle : r; \langle b, c \rangle : r \rrbracket \implies \langle a, c \rangle : r$
<proof>

lemma *trans-onD*:

$\llbracket \text{trans}[A](r); \langle a, b \rangle : r; \langle b, c \rangle : r; a \in A; b \in A; c \in A \rrbracket \implies \langle a, c \rangle : r$
<proof>

lemma *trans-imp-trans-on*: $\text{trans}(r) \implies \text{trans}[A](r)$
<proof>

lemma *trans-on-imp-trans*: $\llbracket \text{trans}[A](r); r \subseteq A * A \rrbracket \implies \text{trans}(r)$
<proof>

11.2 Transitive closure of a relation

lemma *rtrancl-bnd-mono*:

$\text{bnd-mono}(\text{field}(r) * \text{field}(r), \lambda s. \text{id}(\text{field}(r)) \cup (r \circ s))$
<proof>

lemma *rtrancl-mono*: $r \leq s \implies r^* \subseteq s^*$

$\langle proof \rangle$

lemmas *rtrancl-unfold* =
rtrancl-bnd-mono [*THEN* *rtrancl-def* [*THEN* *def-lfp-unfold*]]

lemmas *rtrancl-type* = *rtrancl-def* [*THEN* *def-lfp-subset*]

lemma *relation-rtrancl*: $relation(r^{\widehat{*}})$
 $\langle proof \rangle$

lemma *rtrancl-refl*: $\llbracket a \in field(r) \rrbracket \implies \langle a, a \rangle \in r^{\widehat{*}}$
 $\langle proof \rangle$

lemma *rtrancl-into-rtrancl*: $\llbracket \langle a, b \rangle \in r^{\widehat{*}}; \langle b, c \rangle \in r \rrbracket \implies \langle a, c \rangle \in r^{\widehat{*}}$
 $\langle proof \rangle$

lemma *r-into-rtrancl*: $\langle a, b \rangle \in r \implies \langle a, b \rangle \in r^{\widehat{*}}$
 $\langle proof \rangle$

lemma *r-subset-rtrancl*: $relation(r) \implies r \subseteq r^{\widehat{*}}$
 $\langle proof \rangle$

lemma *rtrancl-field*: $field(r^{\widehat{*}}) = field(r)$
 $\langle proof \rangle$

lemma *rtrancl-full-induct* [*case-names* *initial step*, *consumes* 1]:
 $\llbracket \langle a, b \rangle \in r^{\widehat{*}};$
 $\bigwedge x. x \in field(r) \implies P(\langle x, x \rangle);$
 $\bigwedge x y z. \llbracket P(\langle x, y \rangle); \langle x, y \rangle \in r^{\widehat{*}}; \langle y, z \rangle \in r \rrbracket \implies P(\langle x, z \rangle) \rrbracket$
 $\implies P(\langle a, b \rangle)$
 $\langle proof \rangle$

lemma *rtrancl-induct* [*case-names* *initial step*, *induct set*: *rtrancl*]:
 $\llbracket \langle a, b \rangle \in r^{\widehat{*}};$
 $P(a);$
 $\bigwedge y z. \llbracket \langle a, y \rangle \in r^{\widehat{*}}; \langle y, z \rangle \in r; P(y) \rrbracket \implies P(z)$
 $\rrbracket \implies P(b)$

$\langle proof \rangle$

lemma *trans-rtrancl*: $trans(\widehat{r^*})$
 $\langle proof \rangle$

lemmas *rtrancl-trans* = *trans-rtrancl* [*THEN transD*]

lemma *rtranclE*:
 $\llbracket \langle a, b \rangle \in \widehat{r^*}; (a=b) \implies P; \bigwedge y. \llbracket \langle a, y \rangle \in \widehat{r^*}; \langle y, b \rangle \in r \rrbracket \implies P \rrbracket$
 $\implies P$
 $\langle proof \rangle$

lemma *trans-trancl*: $trans(\widehat{r^+})$
 $\langle proof \rangle$

lemmas *trans-on-trancl* = *trans-trancl* [*THEN trans-imp-trans-on*]

lemmas *trancl-trans* = *trans-trancl* [*THEN transD*]

lemma *trancl-into-rtrancl*: $\langle a, b \rangle \in \widehat{r^+} \implies \langle a, b \rangle \in \widehat{r^*}$
 $\langle proof \rangle$

lemma *r-into-trancl*: $\langle a, b \rangle \in r \implies \langle a, b \rangle \in \widehat{r^+}$
 $\langle proof \rangle$

lemma *r-subset-trancl*: $relation(r) \implies r \subseteq \widehat{r^+}$
 $\langle proof \rangle$

lemma *rtrancl-into-trancl1*: $\llbracket \langle a, b \rangle \in \widehat{r^*}; \langle b, c \rangle \in r \rrbracket \implies \langle a, c \rangle \in \widehat{r^+}$
 $\langle proof \rangle$

lemma *rtrancl-into-trancl2*:
 $\llbracket \langle a, b \rangle \in r; \langle b, c \rangle \in \widehat{r^*} \rrbracket \implies \langle a, c \rangle \in \widehat{r^+}$
 $\langle proof \rangle$

lemma *trancI-induct* [*case-names initial step, induct set: trancI*]:

$\llbracket \langle a, b \rangle \in r^{\wedge+};$
 $\bigwedge y. \llbracket \langle a, y \rangle \in r \rrbracket \implies P(y);$
 $\bigwedge y z. \llbracket \langle a, y \rangle \in r^{\wedge+}; \langle y, z \rangle \in r; P(y) \rrbracket \implies P(z)$
 $\rrbracket \implies P(b)$
 $\langle \text{proof} \rangle$

lemma *trancIE*:

$\llbracket \langle a, b \rangle \in r^{\wedge+};$
 $\langle a, b \rangle \in r \implies P;$
 $\bigwedge y. \llbracket \langle a, y \rangle \in r^{\wedge+}; \langle y, b \rangle \in r \rrbracket \implies P$
 $\rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *trancI-type*: $r^{\wedge+} \subseteq \text{field}(r) * \text{field}(r)$
 $\langle \text{proof} \rangle$

lemma *relation-trancI*: $\text{relation}(r^{\wedge+})$
 $\langle \text{proof} \rangle$

lemma *trancI-subset-times*: $r \subseteq A * A \implies r^{\wedge+} \subseteq A * A$
 $\langle \text{proof} \rangle$

lemma *trancI-mono*: $r \leq s \implies r^{\wedge+} \subseteq s^{\wedge+}$
 $\langle \text{proof} \rangle$

lemma *trancI-eq-r*: $\llbracket \text{relation}(r); \text{trans}(r) \rrbracket \implies r^{\wedge+} = r$
 $\langle \text{proof} \rangle$

lemma *rtrancI-idemp* [*simp*]: $(r^{\wedge*})^{\wedge*} = r^{\wedge*}$
 $\langle \text{proof} \rangle$

lemma *rtrancI-subset*: $\llbracket R \subseteq S; S \subseteq R^{\wedge*} \rrbracket \implies S^{\wedge*} = R^{\wedge*}$
 $\langle \text{proof} \rangle$

lemma *rtrancI-Un-rtrancI*:

$\llbracket \text{relation}(r); \text{relation}(s) \rrbracket \implies (r^{\wedge*} \cup s^{\wedge*})^{\wedge*} = (r \cup s)^{\wedge*}$
 $\langle \text{proof} \rangle$

lemma *rtrancl-converseD*: $\langle x, y \rangle : \text{converse}(r)^\wedge * \implies \langle x, y \rangle : \text{converse}(r^\wedge *)$
 $\langle \text{proof} \rangle$

lemma *rtrancl-converseI*: $\langle x, y \rangle : \text{converse}(r^\wedge *) \implies \langle x, y \rangle : \text{converse}(r)^\wedge *$
 $\langle \text{proof} \rangle$

lemma *rtrancl-converse*: $\text{converse}(r)^\wedge * = \text{converse}(r^\wedge *)$
 $\langle \text{proof} \rangle$

lemma *trancl-converseD*: $\langle a, b \rangle : \text{converse}(r)^\wedge + \implies \langle a, b \rangle : \text{converse}(r^\wedge +)$
 $\langle \text{proof} \rangle$

lemma *trancl-converseI*: $\langle x, y \rangle : \text{converse}(r^\wedge +) \implies \langle x, y \rangle : \text{converse}(r)^\wedge +$
 $\langle \text{proof} \rangle$

lemma *trancl-converse*: $\text{converse}(r)^\wedge + = \text{converse}(r^\wedge +)$
 $\langle \text{proof} \rangle$

lemma *converse-trancl-induct* [case-names initial step, consumes 1]:
 $\llbracket \langle a, b \rangle : r^\wedge +; \bigwedge y. \langle y, b \rangle : r \implies P(y);$
 $\bigwedge y z. \llbracket \langle y, z \rangle \in r; \langle z, b \rangle \in r^\wedge +; P(z) \rrbracket \implies P(y) \rrbracket$
 $\implies P(a)$
 $\langle \text{proof} \rangle$

end

12 Well-Founded Recursion

theory *WF* imports *Trancl* begin

definition

wf :: $i \Rightarrow o$ **where**

$$wf(r) \equiv \forall Z. Z = 0 \mid (\exists x \in Z. \forall y. \langle y, x \rangle : r \longrightarrow \neg y \in Z)$$

definition

wf-on :: $[i, i] \Rightarrow o$ ($\langle \langle \text{open-block notation} = \langle \text{mixfix wf-on} \rangle \rangle wf[-]'(-) \rangle$) **where**

$$wf\text{-on}(A, r) \equiv wf(r \cap A * A)$$

definition

is-recfun :: $[i, i, [i, i] \Rightarrow i, i] \Rightarrow o$ **where**

$$is\text{-recfun}(r, a, H, f) \equiv (f = (\lambda x \in r. \text{“}\{a\}. H(x, \text{restrict}(f, r - \text{“}\{x\})))$$

definition

the-recfun :: $[i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**

$$the\text{-recfun}(r, a, H) \equiv (THE f. is\text{-recfun}(r, a, H, f))$$

definition

$wftrec :: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $wftrec(r, a, H) \equiv H(a, the-recfun(r, a, H))$

definition

$wfrec :: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $wfrec(r, a, H) \equiv wftrec(r^{\wedge}+, a, \lambda x f. H(x, restrict(f, r - \{\{x\}\})))$

definition

$wfrec-on :: [i, i, i, [i, i] \Rightarrow i] \Rightarrow i$ ($\langle open-block\ notation = \langle mixfix\ wfrec-on \rangle \rangle wfrec[-]'(-, -, -)' \rangle$)
where $wfrec[A](r, a, H) \equiv wfrec(r \cap A * A, a, H)$

12.1 Well-Founded Relations**12.1.1 Equivalences between wf and $wf-on$**

lemma $wf-imp-wf-on$: $wf(r) \Longrightarrow wf[A](r)$
 $\langle proof \rangle$

lemma $wf-on-imp-wf$: $\llbracket wf[A](r); r \subseteq A * A \rrbracket \Longrightarrow wf(r)$
 $\langle proof \rangle$

lemma $wf-on-field-imp-wf$: $wf[field(r)](r) \Longrightarrow wf(r)$
 $\langle proof \rangle$

lemma $wf-iff-wf-on-field$: $wf(r) \longleftrightarrow wf[field(r)](r)$
 $\langle proof \rangle$

lemma $wf-on-subset-A$: $\llbracket wf[A](r); B \leq A \rrbracket \Longrightarrow wf[B](r)$
 $\langle proof \rangle$

lemma $wf-on-subset-r$: $\llbracket wf[A](r); s \leq r \rrbracket \Longrightarrow wf[A](s)$
 $\langle proof \rangle$

lemma $wf-subset$: $\llbracket wf(s); r \leq s \rrbracket \Longrightarrow wf(r)$
 $\langle proof \rangle$

12.1.2 Introduction Rules for $wf-on$

If every non-empty subset of A has an r -minimal element then we have $wf[A](r)$.

lemma $wf-onI$:

assumes $prem$: $\bigwedge Z u. \llbracket Z \leq A; u \in Z; \forall x \in Z. \exists y \in Z. \langle y, x \rangle : r \rrbracket \Longrightarrow False$
shows $wf[A](r)$
 $\langle proof \rangle$

If r allows well-founded induction over A then we have $wf[A](r)$. Premise

is equivalent to $\bigwedge B. \forall x \in A. (\forall y. \langle y, x \rangle \in r \longrightarrow y \in B) \longrightarrow x \in B \implies A \subseteq B$

lemma *wf-onI2*:

assumes *prem*: $\bigwedge y B. \llbracket \forall x \in A. (\forall y \in A. \langle y, x \rangle : r \longrightarrow y \in B) \longrightarrow x \in B; \quad y \in A \rrbracket$
 $\implies y \in B$

shows $wf[A](r)$
 $\langle proof \rangle$

12.1.3 Well-founded Induction

Consider the least z in $domain(r)$ such that $P(z)$ does not hold...

lemma *wf-induct-raw*:

$\llbracket wf(r);$
 $\bigwedge x. \llbracket \forall y. \langle y, x \rangle : r \longrightarrow P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(a)$
 $\langle proof \rangle$

lemmas *wf-induct* = *wf-induct-raw* [*rule-format*, *consumes 1*, *case-names step*, *induct set: wf*]

The form of this rule is designed to match *wfI*

lemma *wf-induct2*:

$\llbracket wf(r); \quad a \in A; \quad field(r) \leq A;$
 $\bigwedge x. \llbracket x \in A; \quad \forall y. \langle y, x \rangle : r \longrightarrow P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(a)$
 $\langle proof \rangle$

lemma *field-Int-square*: $field(r \cap A * A) \subseteq A$
 $\langle proof \rangle$

lemma *wf-on-induct-raw* [*consumes 2*, *induct set: wf-on*]:

$\llbracket wf[A](r); \quad a \in A;$
 $\bigwedge x. \llbracket x \in A; \quad \forall y \in A. \langle y, x \rangle : r \longrightarrow P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(a)$
 $\langle proof \rangle$

lemma *wf-on-induct* [*consumes 2*, *case-names step*, *induct set: wf-on*]:

$wf[A](r) \implies a \in A \implies (\bigwedge x. x \in A \implies (\bigwedge y. y \in A \implies \langle y, x \rangle \in r \implies P(y)))$
 $\implies P(x) \implies P(a)$
 $\langle proof \rangle$

If r allows well-founded induction then we have $wf(r)$.

lemma *wfI*:

$\llbracket field(r) \leq A;$
 $\bigwedge y B. \llbracket \forall x \in A. (\forall y \in A. \langle y, x \rangle : r \longrightarrow y \in B) \longrightarrow x \in B; \quad y \in A \rrbracket$
 $\implies y \in B \rrbracket$
 $\implies wf(r)$
 $\langle proof \rangle$

12.2 Basic Properties of Well-Founded Relations

lemma *wf-not-refl*: $wf(r) \implies \langle a, a \rangle \notin r$
 $\langle proof \rangle$

lemma *wf-not-sym* [rule-format]: $wf(r) \implies \forall x. \langle a, x \rangle : r \longrightarrow \langle x, a \rangle \notin r$
 $\langle proof \rangle$

lemmas *wf-asy* = *wf-not-sym* [THEN swap]

lemma *wf-on-not-refl*: $\llbracket wf[A](r); a \in A \rrbracket \implies \langle a, a \rangle \notin r$
 $\langle proof \rangle$

lemma *wf-on-not-sym*:
 $\llbracket wf[A](r); a \in A \rrbracket \implies (\bigwedge b. b \in A \implies \langle a, b \rangle : r \implies \langle b, a \rangle \notin r)$
 $\langle proof \rangle$

lemma *wf-on-asy*:
 $\llbracket wf[A](r); \neg Z \implies \langle a, b \rangle \in r; \langle b, a \rangle \notin r \implies Z; \neg Z \implies a \in A; \neg Z \implies b \in A \rrbracket \implies Z$
 $\langle proof \rangle$

lemma *wf-on-chain3*:
 $\llbracket wf[A](r); \langle a, b \rangle : r; \langle b, c \rangle : r; \langle c, a \rangle : r; a \in A; b \in A; c \in A \rrbracket \implies P$
 $\langle proof \rangle$

transitive closure of a WF relation is WF provided A is downward closed

lemma *wf-on-trancl*:
 $\llbracket wf[A](r); r - \text{“} A \subseteq A \text{”} \rrbracket \implies wf[A](r^{\wedge+})$
 $\langle proof \rangle$

lemma *wf-trancl*: $wf(r) \implies wf(r^{\wedge+})$
 $\langle proof \rangle$

$r - \text{“} \{a\}$ is the set of everything under a in r

lemmas *underI* = *vimage-singleton-iff* [THEN iffD2]

lemmas *underD* = *vimage-singleton-iff* [THEN iffD1]

12.3 The Predicate *is-recfun*

lemma *is-recfun-type*: $is-recfun(r, a, H, f) \implies f \in r - \text{“} \{a\} -> range(f)$
 $\langle proof \rangle$

lemmas *is-recfun-imp-function* = *is-recfun-type* [THEN fun-is-function]

lemma *apply-recfun*:
 $\llbracket is-recfun(r, a, H, f); \langle x, a \rangle : r \rrbracket \implies f'x = H(x, restrict(f, r - \text{“} \{x\}))$

$\langle proof \rangle$

lemma *is-recfun-equal* [rule-format]:

$$\begin{aligned} & \llbracket wf(r); \text{trans}(r); \text{is-recfun}(r, a, H, f); \text{is-recfun}(r, b, H, g) \rrbracket \\ & \implies \langle x, a \rangle : r \longrightarrow \langle x, b \rangle : r \longrightarrow f'x = g'x \\ & \langle proof \rangle \end{aligned}$$

lemma *is-recfun-cut*:

$$\begin{aligned} & \llbracket wf(r); \text{trans}(r); \\ & \quad \text{is-recfun}(r, a, H, f); \text{is-recfun}(r, b, H, g); \langle b, a \rangle : r \rrbracket \\ & \implies \text{restrict}(f, r - \{\{b\}\}) = g \\ & \langle proof \rangle \end{aligned}$$

12.4 Recursion: Main Existence Lemma

lemma *is-recfun-functional*:

$$\llbracket wf(r); \text{trans}(r); \text{is-recfun}(r, a, H, f); \text{is-recfun}(r, a, H, g) \rrbracket \implies f = g$$

 $\langle proof \rangle$

lemma *the-recfun-eq*:

$$\llbracket \text{is-recfun}(r, a, H, f); wf(r); \text{trans}(r) \rrbracket \implies \text{the-recfun}(r, a, H) = f$$

 $\langle proof \rangle$

lemma *is-the-recfun*:

$$\begin{aligned} & \llbracket \text{is-recfun}(r, a, H, f); wf(r); \text{trans}(r) \rrbracket \\ & \implies \text{is-recfun}(r, a, H, \text{the-recfun}(r, a, H)) \\ & \langle proof \rangle \end{aligned}$$

lemma *unfold-the-recfun*:

$$\llbracket wf(r); \text{trans}(r) \rrbracket \implies \text{is-recfun}(r, a, H, \text{the-recfun}(r, a, H))$$

 $\langle proof \rangle$

12.5 Unfolding $wftrec(r, a, H)$

lemma *the-recfun-cut*:

$$\begin{aligned} & \llbracket wf(r); \text{trans}(r); \langle b, a \rangle : r \rrbracket \\ & \implies \text{restrict}(\text{the-recfun}(r, a, H), r - \{\{b\}\}) = \text{the-recfun}(r, b, H) \\ & \langle proof \rangle \end{aligned}$$

lemma *wftrec*:

$$\begin{aligned} & \llbracket wf(r); \text{trans}(r) \rrbracket \implies \\ & \quad wftrec(r, a, H) = H(a, \lambda x \in r - \{\{a\}\}. wftrec(r, x, H)) \\ & \langle proof \rangle \end{aligned}$$

12.5.1 Removal of the Premise $\text{trans}(r)$

lemma *wfrec*:

$$wf(r) \implies wfrec(r, a, H) = H(a, \lambda x \in r - \{\{a\}\}. wfrec(r, x, H))$$

$\langle proof \rangle$

lemma *def-wfrec*:

$$\llbracket \bigwedge x. h(x) \equiv wfrec(r, x, H); \quad wf(r) \rrbracket \implies \\ h(a) = H(a, \lambda x \in r - \{\{a\}. h(x))$$

$\langle proof \rangle$

lemma *wfrec-type*:

$$\llbracket wf(r); \quad a \in A; \quad field(r) \leq A; \\ \bigwedge x u. \llbracket x \in A; \quad u \in Pi(r - \{\{x\}, B) \rrbracket \implies H(x, u) \in B(x) \rrbracket \\ \implies wfrec(r, a, H) \in B(a)$$

$\langle proof \rangle$

lemma *wfrec-on*:

$$\llbracket wf[A](r); \quad a \in A \rrbracket \implies \\ wfrec[A](r, a, H) = H(a, \lambda x \in (r - \{\{a\}\} \cap A. wfrec[A](r, x, H))$$

$\langle proof \rangle$

Minimal-element characterization of well-foundedness

lemma *wf-eq-minimal*: $wf(r) \longleftrightarrow (\forall Q. x \in Q \longrightarrow (\exists z \in Q. \forall y. \langle y, z \rangle : r \longrightarrow y \notin Q))$

$\langle proof \rangle$

end

13 Transitive Sets and Ordinals

theory *Ordinal* **imports** *WF Bool equalities* **begin**

definition

$$Memrel \quad :: i \Rightarrow i \quad \mathbf{where} \\ Memrel(A) \quad \equiv \{ z \in A * A . \exists x y. z = \langle x, y \rangle \wedge x \in y \}$$

definition

$$Transset \quad :: i \Rightarrow o \quad \mathbf{where} \\ Transset(i) \equiv \forall x \in i. x \leq i$$

definition

$$Ord \quad :: i \Rightarrow o \quad \mathbf{where} \\ Ord(i) \quad \equiv Transset(i) \wedge (\forall x \in i. Transset(x))$$

definition

$$lt \quad :: [i, i] \Rightarrow o \quad (\mathbf{infixl} \ \langle \! \! \! \langle \! \! \! \rangle \! \! \! \rangle \ 50) \quad \mathbf{where} \\ i \langle \! \! \! \langle \! \! \! \rangle \! \! \! \rangle j \quad \equiv i \in j \wedge Ord(j)$$

definition

$$Limit \quad :: i \Rightarrow o \quad \mathbf{where}$$

$$\text{Limit}(i) \equiv \text{Ord}(i) \wedge 0 < i \wedge (\forall y. y < i \longrightarrow \text{succ}(y) < i)$$

abbreviation

le (**infixl** $\langle \leq \rangle$ 50) **where**
 $x \leq y \equiv x < \text{succ}(y)$

13.1 Rules for Transset

13.1.1 Three Neat Characterisations of Transset

lemma *Transset-iff-Pow*: $\text{Transset}(A) <-> A \leq \text{Pow}(A)$
 $\langle \text{proof} \rangle$

lemma *Transset-iff-Union-succ*: $\text{Transset}(A) <-> \bigcup (\text{succ}(A)) = A$
 $\langle \text{proof} \rangle$

lemma *Transset-iff-Union-subset*: $\text{Transset}(A) <-> \bigcup (A) \subseteq A$
 $\langle \text{proof} \rangle$

13.1.2 Consequences of Downwards Closure

lemma *Transset-doubleton-D*:
 $\llbracket \text{Transset}(C); \{a, b\} : C \rrbracket \implies a \in C \wedge b \in C$
 $\langle \text{proof} \rangle$

lemma *Transset-Pair-D*:
 $\llbracket \text{Transset}(C); \langle a, b \rangle \in C \rrbracket \implies a \in C \wedge b \in C$
 $\langle \text{proof} \rangle$

lemma *Transset-includes-domain*:
 $\llbracket \text{Transset}(C); A * B \subseteq C; b \in B \rrbracket \implies A \subseteq C$
 $\langle \text{proof} \rangle$

lemma *Transset-includes-range*:
 $\llbracket \text{Transset}(C); A * B \subseteq C; a \in A \rrbracket \implies B \subseteq C$
 $\langle \text{proof} \rangle$

13.1.3 Closure Properties

lemma *Transset-0*: $\text{Transset}(0)$
 $\langle \text{proof} \rangle$

lemma *Transset-Un*:
 $\llbracket \text{Transset}(i); \text{Transset}(j) \rrbracket \implies \text{Transset}(i \cup j)$
 $\langle \text{proof} \rangle$

lemma *Transset-Int*:
 $\llbracket \text{Transset}(i); \text{Transset}(j) \rrbracket \implies \text{Transset}(i \cap j)$
 $\langle \text{proof} \rangle$

lemma *Transset-succ*: $\text{Transset}(i) \implies \text{Transset}(\text{succ}(i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Pow*: $\text{Transset}(i) \implies \text{Transset}(\text{Pow}(i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Union*: $\text{Transset}(A) \implies \text{Transset}(\bigcup(A))$
 $\langle \text{proof} \rangle$

lemma *Transset-Union-family*:
 $\llbracket \bigwedge i. i \in A \implies \text{Transset}(i) \rrbracket \implies \text{Transset}(\bigcup(A))$
 $\langle \text{proof} \rangle$

lemma *Transset-Inter-family*:
 $\llbracket \bigwedge i. i \in A \implies \text{Transset}(i) \rrbracket \implies \text{Transset}(\bigcap(A))$
 $\langle \text{proof} \rangle$

lemma *Transset-UN*:
 $(\bigwedge x. x \in A \implies \text{Transset}(B(x))) \implies \text{Transset}(\bigcup_{x \in A} B(x))$
 $\langle \text{proof} \rangle$

lemma *Transset-INT*:
 $(\bigwedge x. x \in A \implies \text{Transset}(B(x))) \implies \text{Transset}(\bigcap_{x \in A} B(x))$
 $\langle \text{proof} \rangle$

13.2 Lemmas for Ordinals

lemma *OrdI*:
 $\llbracket \text{Transset}(i); \bigwedge x. x \in i \implies \text{Transset}(x) \rrbracket \implies \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-is-Transset*: $\text{Ord}(i) \implies \text{Transset}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-contains-Transset*:
 $\llbracket \text{Ord}(i); j \in i \rrbracket \implies \text{Transset}(j)$
 $\langle \text{proof} \rangle$

lemma *Ord-in-Ord*: $\llbracket \text{Ord}(i); j \in i \rrbracket \implies \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemma *Ord-in-Ord'*: $\llbracket j \in i; \text{Ord}(i) \rrbracket \implies \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemmas *Ord-succD* = *Ord-in-Ord* [*OF* - *succI1*]

lemma *Ord-subset-Ord*: $\llbracket \text{Ord}(i); \text{Transset}(j); j \leq i \rrbracket \implies \text{Ord}(j)$
 $\langle \text{proof} \rangle$

lemma *OrdmemD*: $\llbracket j \in i; \text{Ord}(i) \rrbracket \implies j \leq i$
 $\langle \text{proof} \rangle$

lemma *Ord-trans*: $\llbracket i \in j; j \in k; \text{Ord}(k) \rrbracket \implies i \in k$
 $\langle \text{proof} \rangle$

lemma *Ord-succ-subsetI*: $\llbracket i \in j; \text{Ord}(j) \rrbracket \implies \text{succ}(i) \subseteq j$
 $\langle \text{proof} \rangle$

13.3 The Construction of Ordinals: 0, succ, Union

lemma *Ord-0* [*iff*, *TC*]: $\text{Ord}(0)$
 $\langle \text{proof} \rangle$

lemma *Ord-succ* [*TC*]: $\text{Ord}(i) \implies \text{Ord}(\text{succ}(i))$
 $\langle \text{proof} \rangle$

lemmas *Ord-1* = *Ord-0* [*THEN Ord-succ*]

lemma *Ord-succ-iff* [*iff*]: $\text{Ord}(\text{succ}(i)) <-> \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-Un* [*intro*, *simp*, *TC*]: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{Ord}(i \cup j)$
 $\langle \text{proof} \rangle$

lemma *Ord-Int* [*TC*]: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{Ord}(i \cap j)$
 $\langle \text{proof} \rangle$

There is no set of all ordinals, for then it would contain itself

lemma *ON-class*: $\neg (\forall i. i \in X <-> \text{Ord}(i))$
 $\langle \text{proof} \rangle$

13.4 < is 'less Than' for Ordinals

lemma *ltI*: $\llbracket i \in j; \text{Ord}(j) \rrbracket \implies i < j$
 $\langle \text{proof} \rangle$

lemma *ltE*:
 $\llbracket i < j; \llbracket i \in j; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies P \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *ltD*: $i < j \implies i \in j$
 $\langle \text{proof} \rangle$

lemma *not-lt0* [*simp*]: $\neg i < 0$
 $\langle \text{proof} \rangle$

lemma *lt-Ord*: $j < i \implies \text{Ord}(j)$

$\langle \text{proof} \rangle$

lemma *lt-Ord2*: $j < i \implies \text{Ord}(i)$

$\langle \text{proof} \rangle$

lemmas *le-Ord2* = *lt-Ord2* [*THEN Ord-succD*]

lemmas *lt0E* = *not-lt0* [*THEN notE, elim!*]

lemma *lt-trans* [*trans*]: $\llbracket i < j; j < k \rrbracket \implies i < k$

$\langle \text{proof} \rangle$

lemma *lt-not-sym*: $i < j \implies \neg (j < i)$

$\langle \text{proof} \rangle$

lemmas *lt-asym* = *lt-not-sym* [*THEN swap*]

lemma *lt-irrefl* [*elim!*]: $i < i \implies P$

$\langle \text{proof} \rangle$

lemma *lt-not-refl*: $\neg i < i$

$\langle \text{proof} \rangle$

Recall that $i \leq j$ abbreviates $i < j$!

lemma *le-iff*: $i \leq j \iff i < j \mid (i = j \wedge \text{Ord}(j))$

$\langle \text{proof} \rangle$

lemma *leI*: $i < j \implies i \leq j$

$\langle \text{proof} \rangle$

lemma *le-eqI*: $\llbracket i = j; \text{Ord}(j) \rrbracket \implies i \leq j$

$\langle \text{proof} \rangle$

lemmas *le-refl* = *refl* [*THEN le-eqI*]

lemma *le-refl-iff* [*iff*]: $i \leq i \iff \text{Ord}(i)$

$\langle \text{proof} \rangle$

lemma *leCI*: $(\neg (i = j \wedge \text{Ord}(j)) \implies i < j) \implies i \leq j$

$\langle \text{proof} \rangle$

lemma *leE*:

$\llbracket i \leq j; i < j \implies P; \llbracket i = j; \text{Ord}(j) \rrbracket \implies P \rrbracket \implies P$

$\langle proof \rangle$

lemma *le-anti-sym*: $\llbracket i \leq j; j \leq i \rrbracket \implies i=j$
 $\langle proof \rangle$

lemma *le0-iff* [*simp*]: $i \leq 0 \iff i=0$
 $\langle proof \rangle$

lemmas *le0D* = *le0-iff* [*THEN iffD1, dest!*]

13.5 Natural Deduction Rules for Memrel

lemma *Memrel-iff* [*simp*]: $\langle a,b \rangle \in Memrel(A) \iff a \in b \wedge a \in A \wedge b \in A$
 $\langle proof \rangle$

lemma *MemrelI* [*intro!*]: $\llbracket a \in b; a \in A; b \in A \rrbracket \implies \langle a,b \rangle \in Memrel(A)$
 $\langle proof \rangle$

lemma *MemrelE* [*elim!*]:
 $\llbracket \langle a,b \rangle \in Memrel(A);$
 $\llbracket a \in A; b \in A; a \in b \rrbracket \implies P$
 $\implies P$
 $\langle proof \rangle$

lemma *Memrel-type*: $Memrel(A) \subseteq A * A$
 $\langle proof \rangle$

lemma *Memrel-mono*: $A \leq B \implies Memrel(A) \subseteq Memrel(B)$
 $\langle proof \rangle$

lemma *Memrel-0* [*simp*]: $Memrel(0) = 0$
 $\langle proof \rangle$

lemma *Memrel-1* [*simp*]: $Memrel(1) = 0$
 $\langle proof \rangle$

lemma *relation-Memrel*: $relation(Memrel(A))$
 $\langle proof \rangle$

lemma *wf-Memrel*: $wf(Memrel(A))$
 $\langle proof \rangle$

The premise $Ord(i)$ does not suffice.

lemma *trans-Memrel*:
 $Ord(i) \implies trans(Memrel(i))$
 $\langle proof \rangle$

However, the following premise is strong enough.

lemma *Transset-trans-Memrel*:

$\forall j \in i. \text{Transset}(j) \implies \text{trans}(\text{Memrel}(i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Memrel-iff*:

$\text{Transset}(A) \implies \langle a, b \rangle \in \text{Memrel}(A) \iff a \in b \wedge b \in A$
 $\langle \text{proof} \rangle$

13.6 Transfinite Induction

lemma *Transset-induct*:

$\llbracket i \in k; \text{Transset}(k);$
 $\bigwedge x. \llbracket x \in k; \forall y \in x. P(y) \rrbracket \implies P(x) \rrbracket$
 $\implies P(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-induct* [consumes 2]:

$i \in k \implies \text{Ord}(k) \implies (\bigwedge x. x \in k \implies (\bigwedge y. y \in x \implies P(y)) \implies P(x)) \implies P(i)$
 $\langle \text{proof} \rangle$

lemma *trans-induct* [consumes 1, case-names step]:

$\text{Ord}(i) \implies (\bigwedge x. \text{Ord}(x) \implies (\bigwedge y. y \in x \implies P(y)) \implies P(x)) \implies P(i)$
 $\langle \text{proof} \rangle$

14 Fundamental properties of the epsilon ordering ($<$ on ordinals)

14.0.1 Proving That $<$ is a Linear Ordering on the Ordinals

lemma *Ord-linear*:

$\text{Ord}(i) \implies \text{Ord}(j) \implies i \in j \mid i = j \mid j \in i$
 $\langle \text{proof} \rangle$

The trichotomy law for ordinals

lemma *Ord-linear-lt*:

assumes $o: \text{Ord}(i) \text{ Ord}(j)$
obtains $(lt) \ i < j \mid (eq) \ i = j \mid (gt) \ j < i$
 $\langle \text{proof} \rangle$

lemma *Ord-linear2*:

assumes $o: \text{Ord}(i) \text{ Ord}(j)$
obtains $(lt) \ i < j \mid (ge) \ j \leq i$
 $\langle \text{proof} \rangle$

lemma *Ord-linear-le*:

assumes $o: \text{Ord}(i) \text{ Ord}(j)$

obtains $(le) \ i \leq j \mid (ge) \ j \leq i$
 $\langle proof \rangle$

lemma *le-imp-not-lt*: $j \leq i \implies \neg i < j$
 $\langle proof \rangle$

lemma *not-lt-imp-le*: $\llbracket \neg i < j; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies j \leq i$
 $\langle proof \rangle$

14.0.2 Some Rewrite Rules for $<$, \leq

lemma *Ord-mem-iff-lt*: $\text{Ord}(j) \implies i \in j \iff i < j$
 $\langle proof \rangle$

lemma *not-lt-iff-le*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \neg i < j \iff j \leq i$
 $\langle proof \rangle$

lemma *not-le-iff-lt*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \neg i \leq j \iff j < i$
 $\langle proof \rangle$

lemma *Ord-0-le*: $\text{Ord}(i) \implies 0 \leq i$
 $\langle proof \rangle$

lemma *Ord-0-lt*: $\llbracket \text{Ord}(i); i \neq 0 \rrbracket \implies 0 < i$
 $\langle proof \rangle$

lemma *Ord-0-lt-iff*: $\text{Ord}(i) \implies i \neq 0 \iff 0 < i$
 $\langle proof \rangle$

14.1 Results about Less-Than or Equals

lemma *zero-le-succ-iff* [*iff*]: $0 \leq \text{succ}(x) \iff \text{Ord}(x)$
 $\langle proof \rangle$

lemma *subset-imp-le*: $\llbracket j \leq i; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies j \leq i$
 $\langle proof \rangle$

lemma *le-imp-subset*: $i \leq j \implies i <= j$
 $\langle proof \rangle$

lemma *le-subset-iff*: $j \leq i \iff j <= i \wedge \text{Ord}(i) \wedge \text{Ord}(j)$
 $\langle proof \rangle$

lemma *le-succ-iff*: $i \leq \text{succ}(j) \iff i \leq j \mid i = \text{succ}(j) \wedge \text{Ord}(i)$
 $\langle proof \rangle$

lemma *all-lt-imp-le*: $\llbracket \text{Ord}(i); \text{Ord}(j); \bigwedge x. x < j \implies x < i \rrbracket \implies j \leq i$
 $\langle proof \rangle$

14.1.1 Transitivity Laws

lemma *lt-trans1*: $\llbracket i \leq j; j < k \rrbracket \implies i < k$
 $\langle \text{proof} \rangle$

lemma *lt-trans2*: $\llbracket i < j; j \leq k \rrbracket \implies i < k$
 $\langle \text{proof} \rangle$

lemma *le-trans*: $\llbracket i \leq j; j \leq k \rrbracket \implies i \leq k$
 $\langle \text{proof} \rangle$

lemma *succ-leI*: $i < j \implies \text{succ}(i) \leq j$
 $\langle \text{proof} \rangle$

lemma *succ-leE*: $\text{succ}(i) \leq j \implies i < j$
 $\langle \text{proof} \rangle$

lemma *succ-le-iff* [*iff*]: $\text{succ}(i) \leq j \iff i < j$
 $\langle \text{proof} \rangle$

lemma *succ-le-imp-le*: $\text{succ}(i) \leq \text{succ}(j) \implies i \leq j$
 $\langle \text{proof} \rangle$

lemma *lt-subset-trans*: $\llbracket i \subseteq j; j < k; \text{Ord}(i) \rrbracket \implies i < k$
 $\langle \text{proof} \rangle$

lemma *lt-imp-0-lt*: $j < i \implies 0 < i$
 $\langle \text{proof} \rangle$

lemma *succ-lt-iff*: $\text{succ}(i) < j \iff i < j \wedge \text{succ}(i) \neq j$
 $\langle \text{proof} \rangle$

lemma *Ord-succ-mem-iff*: $\text{Ord}(j) \implies \text{succ}(i) \in \text{succ}(j) \iff i \in j$
 $\langle \text{proof} \rangle$

14.1.2 Union and Intersection

lemma *Un-upper1-le*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies i \leq i \cup j$
 $\langle \text{proof} \rangle$

lemma *Un-upper2-le*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies j \leq i \cup j$
 $\langle \text{proof} \rangle$

lemma *Un-least-lt*: $\llbracket i < k; j < k \rrbracket \implies i \cup j < k$
 $\langle \text{proof} \rangle$

lemma *Un-least-lt-iff*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies i \cup j < k \iff i < k \wedge j < k$
 $\langle \text{proof} \rangle$

lemma *Un-least-mem-iff*:

$$\llbracket \text{Ord}(i); \text{Ord}(j); \text{Ord}(k) \rrbracket \implies i \cup j \in k \iff i \in k \wedge j \in k$$

<proof>

lemma *Int-greatest-lt*: $\llbracket i < k; j < k \rrbracket \implies i \cap j < k$

<proof>

lemma *Ord-Un-if*:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies i \cup j = (\text{if } j < i \text{ then } i \text{ else } j)$$

<proof>

lemma *succ-Un-distrib*:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{succ}(i \cup j) = \text{succ}(i) \cup \text{succ}(j)$$

<proof>

lemma *lt-Un-iff*:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies k < i \cup j \iff k < i \mid k < j$$

<proof>

lemma *le-Un-iff*:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies k \leq i \cup j \iff k \leq i \mid k \leq j$$

<proof>

lemma *Un-upper1-lt*: $\llbracket k < i; \text{Ord}(j) \rrbracket \implies k < i \cup j$

<proof>

lemma *Un-upper2-lt*: $\llbracket k < j; \text{Ord}(i) \rrbracket \implies k < i \cup j$

<proof>

lemma *Ord-Union-succ-eq*: $\text{Ord}(i) \implies \bigcup(\text{succ}(i)) = i$

<proof>

14.2 Results about Limits

lemma *Ord-Union* [*intro,simp,TC*]: $\llbracket \bigwedge i. i \in A \implies \text{Ord}(i) \rrbracket \implies \text{Ord}(\bigcup(A))$

<proof>

lemma *Ord-UN* [*intro,simp,TC*]:

$$\llbracket \bigwedge x. x \in A \implies \text{Ord}(B(x)) \rrbracket \implies \text{Ord}(\bigcup_{x \in A} B(x))$$

<proof>

lemma *Ord-Inter* [*intro,simp,TC*]:

$$\llbracket \bigwedge i. i \in A \implies \text{Ord}(i) \rrbracket \implies \text{Ord}(\bigcap(A))$$

<proof>

lemma *Ord-INT* [*intro,simp,TC*]:

$\llbracket \bigwedge x. x \in A \implies \text{Ord}(B(x)) \rrbracket \implies \text{Ord}(\bigcap_{x \in A} B(x))$
 $\langle \text{proof} \rangle$

lemma *UN-least-le*:

$\llbracket \text{Ord}(i); \bigwedge x. x \in A \implies b(x) \leq i \rrbracket \implies (\bigcup_{x \in A} b(x)) \leq i$
 $\langle \text{proof} \rangle$

lemma *UN-succ-least-lt*:

$\llbracket j < i; \bigwedge x. x \in A \implies b(x) < j \rrbracket \implies (\bigcup_{x \in A} \text{succ}(b(x))) < i$
 $\langle \text{proof} \rangle$

lemma *UN-upper-lt*:

$\llbracket a \in A; i < b(a); \text{Ord}(\bigcup_{x \in A} b(x)) \rrbracket \implies i < (\bigcup_{x \in A} b(x))$
 $\langle \text{proof} \rangle$

lemma *UN-upper-le*:

$\llbracket a \in A; i \leq b(a); \text{Ord}(\bigcup_{x \in A} b(x)) \rrbracket \implies i \leq (\bigcup_{x \in A} b(x))$
 $\langle \text{proof} \rangle$

lemma *lt-Union-iff*: $\forall i \in A. \text{Ord}(i) \implies (j < \bigcup(A)) \iff (\exists i \in A. j < i)$
 $\langle \text{proof} \rangle$

lemma *Union-upper-le*:

$\llbracket j \in J; i \leq j; \text{Ord}(\bigcup(J)) \rrbracket \implies i \leq \bigcup J$
 $\langle \text{proof} \rangle$

lemma *le-implies-UN-le-UN*:

$\llbracket \bigwedge x. x \in A \implies c(x) \leq d(x) \rrbracket \implies (\bigcup_{x \in A} c(x)) \leq (\bigcup_{x \in A} d(x))$
 $\langle \text{proof} \rangle$

lemma *Ord-equality*: $\text{Ord}(i) \implies (\bigcup_{y \in i} \text{succ}(y)) = i$
 $\langle \text{proof} \rangle$

lemma *Ord-Union-subset*: $\text{Ord}(i) \implies \bigcup(i) \subseteq i$
 $\langle \text{proof} \rangle$

14.3 Limit Ordinals – General Properties

lemma *Limit-Union-eq*: $\text{Limit}(i) \implies \bigcup(i) = i$
 $\langle \text{proof} \rangle$

lemma *Limit-is-Ord*: $\text{Limit}(i) \implies \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Limit-has-0*: $\text{Limit}(i) \implies 0 < i$
 $\langle \text{proof} \rangle$

lemma *Limit-nonzero*: $\text{Limit}(i) \implies i \neq 0$

$\langle \text{proof} \rangle$

lemma *Limit-has-succ*: $\llbracket \text{Limit}(i); j < i \rrbracket \implies \text{succ}(j) < i$

$\langle \text{proof} \rangle$

lemma *Limit-succ-lt-iff* [simp]: $\text{Limit}(i) \implies \text{succ}(j) < i \iff (j < i)$

$\langle \text{proof} \rangle$

lemma *zero-not-Limit* [iff]: $\neg \text{Limit}(0)$

$\langle \text{proof} \rangle$

lemma *Limit-has-1*: $\text{Limit}(i) \implies 1 < i$

$\langle \text{proof} \rangle$

lemma *increasing-LimitI*: $\llbracket 0 < l; \forall x \in l. \exists y \in l. x < y \rrbracket \implies \text{Limit}(l)$

$\langle \text{proof} \rangle$

lemma *non-succ-LimitI*:

assumes $i: 0 < i$ **and** $\text{nsucc}: \bigwedge y. \text{succ}(y) \neq i$

shows $\text{Limit}(i)$

$\langle \text{proof} \rangle$

lemma *succ-LimitE* [elim!]: $\text{Limit}(\text{succ}(i)) \implies P$

$\langle \text{proof} \rangle$

lemma *not-succ-Limit* [simp]: $\neg \text{Limit}(\text{succ}(i))$

$\langle \text{proof} \rangle$

lemma *Limit-le-succD*: $\llbracket \text{Limit}(i); i \leq \text{succ}(j) \rrbracket \implies i \leq j$

$\langle \text{proof} \rangle$

14.3.1 Traditional 3-Way Case Analysis on Ordinals

lemma *Ord-cases-disj*: $\text{Ord}(i) \implies i=0 \mid (\exists j. \text{Ord}(j) \wedge i=\text{succ}(j)) \mid \text{Limit}(i)$

$\langle \text{proof} \rangle$

lemma *Ord-cases*:

assumes $i: \text{Ord}(i)$

obtains $(0) i=0 \mid (\text{succ}) j$ **where** $\text{Ord}(j) i=\text{succ}(j) \mid (\text{limit}) \text{Limit}(i)$

$\langle \text{proof} \rangle$

lemma *trans-induct3-raw*:

$\llbracket \text{Ord}(i);$

$P(0);$

$\bigwedge x. \llbracket \text{Ord}(x); P(x) \rrbracket \implies P(\text{succ}(x));$

$\bigwedge x. \llbracket \text{Limit}(x); \forall y \in x. P(y) \rrbracket \implies P(x)$

$\rrbracket \implies P(i)$

$\langle proof \rangle$

lemma *trans-induct3* [*case-names 0 succ limit, consumes 1*]:

$Ord(i) \implies P(0) \implies (\bigwedge x. Ord(x) \implies P(x) \implies P(succ(x))) \implies (\bigwedge x. Limit(x) \implies (\bigwedge y. y \in x \implies P(y)) \implies P(x)) \implies P(i)$
 $\langle proof \rangle$

A set of ordinals is either empty, contains its own union, or its union is a limit ordinal.

lemma *Union-le*: $\llbracket \bigwedge x. x \in I \implies x \leq j; Ord(j) \rrbracket \implies \bigcup(I) \leq j$
 $\langle proof \rangle$

lemma *Ord-set-cases*:

assumes $I: \forall i \in I. Ord(i)$

shows $I = 0 \vee \bigcup(I) \in I \vee (\bigcup(I) \notin I \wedge Limit(\bigcup(I)))$

$\langle proof \rangle$

If the union of a set of ordinals is a successor, then it is an element of that set.

lemma *Ord-Union-eq-succD*: $\llbracket \forall x \in X. Ord(x); \bigcup X = succ(j) \rrbracket \implies succ(j) \in X$
 $\langle proof \rangle$

lemma *Limit-Union* [*rule-format*]: $\llbracket I \neq 0; (\bigwedge i. i \in I \implies Limit(i)) \rrbracket \implies Limit(\bigcup I)$
 $\langle proof \rangle$

end

15 Special quantifiers

theory *OrdQuant* **imports** *Ordinal* **begin**

15.1 Quantifiers and union operator for ordinals

definition

$oall :: [i, i \Rightarrow o] \Rightarrow o$ **where**
 $oall(A, P) \equiv \forall x. x < A \longrightarrow P(x)$

definition

$oex :: [i, i \Rightarrow o] \Rightarrow o$ **where**
 $oex(A, P) \equiv \exists x. x < A \wedge P(x)$

definition

$OUnion :: [i, i \Rightarrow i] \Rightarrow i$ **where**
 $OUnion(i, B) \equiv \{z: \bigcup x \in i. B(x). Ord(i)\}$

syntax

$-oall \quad :: [idt, i, o] \Rightarrow o \ (\langle \langle indent=3 \ notation=\langle binder \ \forall \langle \rangle \rangle \forall \langle - \rangle / - \rangle \rangle 10)$
 $-oex \quad :: [idt, i, o] \Rightarrow o \ (\langle \langle indent=3 \ notation=\langle binder \ \exists \langle \rangle \rangle \exists \langle - \rangle / - \rangle \rangle 10)$
 $-OUNION \quad :: [idt, i, i] \Rightarrow i \ (\langle \langle indent=3 \ notation=\langle binder \ \bigcup \langle \rangle \rangle \bigcup \langle - \rangle / - \rangle \rangle 10)$
syntax-consts
 $-oall \equiv oall \text{ and}$
 $-oex \equiv oex \text{ and}$
 $-OUNION \equiv OUnion$
translations
 $\forall x < a. P \equiv CONST \ oall(a, \lambda x. P)$
 $\exists x < a. P \equiv CONST \ oex(a, \lambda x. P)$
 $\bigcup x < a. B \equiv CONST \ OUnion(a, \lambda x. B)$

15.1.1 simplification of the new quantifiers

lemma $[simp]$: $(\forall x < 0. P(x))$
 $\langle proof \rangle$

lemma $[simp]$: $\neg(\exists x < 0. P(x))$
 $\langle proof \rangle$

lemma $[simp]$: $(\forall x < succ(i). P(x)) <-> (Ord(i) \longrightarrow P(i) \wedge (\forall x < i. P(x)))$
 $\langle proof \rangle$

lemma $[simp]$: $(\exists x < succ(i). P(x)) <-> (Ord(i) \wedge (P(i) \mid (\exists x < i. P(x))))$
 $\langle proof \rangle$

15.1.2 Union over ordinals

lemma $Ord-OUN$ $[intro, simp]$:
 $\llbracket \bigwedge x. x < A \implies Ord(B(x)) \rrbracket \implies Ord(\bigcup x < A. B(x))$
 $\langle proof \rangle$

lemma $OUN-upper-lt$:
 $\llbracket a < A; \ i < b(a); \ Ord(\bigcup x < A. b(x)) \rrbracket \implies i < (\bigcup x < A. b(x))$
 $\langle proof \rangle$

lemma $OUN-upper-le$:
 $\llbracket a < A; \ i \leq b(a); \ Ord(\bigcup x < A. b(x)) \rrbracket \implies i \leq (\bigcup x < A. b(x))$
 $\langle proof \rangle$

lemma $Limit-OUN-eq$: $Limit(i) \implies (\bigcup x < i. x) = i$
 $\langle proof \rangle$

lemma $OUN-least$:
 $(\bigwedge x. x < A \implies B(x) \subseteq C) \implies (\bigcup x < A. B(x)) \subseteq C$
 $\langle proof \rangle$

lemma $OUN-least-le$:

$\llbracket \text{Ord}(i); \bigwedge x. x < A \implies b(x) \leq i \rrbracket \implies (\bigcup x < A. b(x)) \leq i$
 $\langle \text{proof} \rangle$

lemma *le-implies-OUN-le-OUN*:

$\llbracket \bigwedge x. x < A \implies c(x) \leq d(x) \rrbracket \implies (\bigcup x < A. c(x)) \leq (\bigcup x < A. d(x))$
 $\langle \text{proof} \rangle$

lemma *OUN-UN-eq*:

$(\bigwedge x. x \in A \implies \text{Ord}(B(x)))$
 $\implies (\bigcup z < (\bigcup x \in A. B(x)). C(z)) = (\bigcup x \in A. \bigcup z < B(x). C(z))$
 $\langle \text{proof} \rangle$

lemma *OUN-Union-eq*:

$(\bigwedge x. x \in X \implies \text{Ord}(x))$
 $\implies (\bigcup z < \bigcup (X). C(z)) = (\bigcup x \in X. \bigcup z < x. C(z))$
 $\langle \text{proof} \rangle$

lemma *atomize-oall* [*symmetric, rulify*]:

$(\bigwedge x. x < A \implies P(x)) \equiv \text{Trueprop } (\forall x < A. P(x))$
 $\langle \text{proof} \rangle$

15.1.3 universal quantifier for ordinals

lemma *oallI* [*intro!*]:

$\llbracket \bigwedge x. x < A \implies P(x) \rrbracket \implies \forall x < A. P(x)$
 $\langle \text{proof} \rangle$

lemma *ospec*: $\llbracket \forall x < A. P(x); x < A \rrbracket \implies P(x)$

$\langle \text{proof} \rangle$

lemma *oallE*:

$\llbracket \forall x < A. P(x); P(x) \implies Q; \neg x < A \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *rev-oallE* [*elim*]:

$\llbracket \forall x < A. P(x); \neg x < A \implies Q; P(x) \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *oall-simp* [*simp*]: $(\forall x < a. \text{True}) <-> \text{True}$

$\langle \text{proof} \rangle$

lemma *oall-cong* [*cong*]:

$\llbracket a = a'; \bigwedge x. x < a' \implies P(x) <-> P'(x) \rrbracket$
 $\implies \text{oall}(a, \lambda x. P(x)) <-> \text{oall}(a', \lambda x. P'(x))$
 $\langle \text{proof} \rangle$

15.1.4 existential quantifier for ordinals

lemma *oexI* [*intro*]:

$$\llbracket P(x); x < A \rrbracket \implies \exists x < A. P(x)$$

<proof>

lemma *oexCI*:

$$\llbracket \forall x < A. \neg P(x) \implies P(a); a < A \rrbracket \implies \exists x < A. P(x)$$

<proof>

lemma *oexE* [*elim!*]:

$$\llbracket \exists x < A. P(x); \bigwedge x. \llbracket x < A; P(x) \rrbracket \implies Q \rrbracket \implies Q$$

<proof>

lemma *oex-cong* [*cong*]:

$$\begin{aligned} &\llbracket a = a'; \bigwedge x. x < a' \implies P(x) <-> P'(x) \rrbracket \\ &\implies oex(a, \lambda x. P(x)) <-> oex(a', \lambda x. P'(x)) \end{aligned}$$

<proof>

15.1.5 Rules for Ordinal-Indexed Unions

lemma *OUN-I* [*intro*]: $\llbracket a < i; b \in B(a) \rrbracket \implies b: (\bigcup z < i. B(z))$

<proof>

lemma *OUN-E* [*elim!*]:

$$\llbracket b \in (\bigcup z < i. B(z)); \bigwedge a. \llbracket b \in B(a); a < i \rrbracket \implies R \rrbracket \implies R$$

<proof>

lemma *OUN-iff*: $b \in (\bigcup x < i. B(x)) <-> (\exists x < i. b \in B(x))$

<proof>

lemma *OUN-cong* [*cong*]:

$$\llbracket i = j; \bigwedge x. x < j \implies C(x) = D(x) \rrbracket \implies (\bigcup x < i. C(x)) = (\bigcup x < j. D(x))$$

<proof>

lemma *lt-induct*:

$$\llbracket i < k; \bigwedge x. \llbracket x < k; \forall y < x. P(y) \rrbracket \implies P(x) \rrbracket \implies P(i)$$

<proof>

15.2 Quantification over a class

definition

$$\begin{aligned} rall &:: [i \Rightarrow o, i \Rightarrow o] \Rightarrow o \text{ where} \\ rall(M, P) &\equiv \forall x. M(x) \longrightarrow P(x) \end{aligned}$$

definition

$$\begin{aligned} rex &:: [i \Rightarrow o, i \Rightarrow o] \Rightarrow o \text{ where} \\ rex(M, P) &\equiv \exists x. M(x) \wedge P(x) \end{aligned}$$

syntax

-rall $:: [pttrn, i \Rightarrow o, o] \Rightarrow o \quad (\langle \langle indent=3 \text{ notation}=\langle binder \ \forall [] \rangle \forall [-]. / - \rangle$
 10)

-rex $:: [pttrn, i \Rightarrow o, o] \Rightarrow o \quad (\langle \langle indent=3 \text{ notation}=\langle binder \ \exists [] \rangle \exists [-]. / - \rangle$
 10)

syntax-consts

-rall $\equiv rall$ and

-rex $\equiv rex$

translations

$\forall x[M]. P \equiv CONST \text{ rall}(M, \lambda x. P)$

$\exists x[M]. P \equiv CONST \text{ rex}(M, \lambda x. P)$

15.2.1 Relativized universal quantifier

lemma *rallI* [intro!]: $\llbracket \bigwedge x. M(x) \Longrightarrow P(x) \rrbracket \Longrightarrow \forall x[M]. P(x)$
 $\langle proof \rangle$

lemma *rspec*: $\llbracket \forall x[M]. P(x); M(x) \rrbracket \Longrightarrow P(x)$
 $\langle proof \rangle$

lemma *rev-rallE* [elim]:

$\llbracket \forall x[M]. P(x); \neg M(x) \rrbracket \Longrightarrow Q; P(x) \Longrightarrow Q \rrbracket \Longrightarrow Q$
 $\langle proof \rangle$

lemma *rallE*: $\llbracket \forall x[M]. P(x); P(x) \Longrightarrow Q; \neg M(x) \Longrightarrow Q \rrbracket \Longrightarrow Q$
 $\langle proof \rangle$

lemma *rall-triv* [simp]: $(\forall x[M]. P) \longleftrightarrow ((\exists x. M(x)) \longrightarrow P)$
 $\langle proof \rangle$

lemma *rall-cong* [cong]:

$(\bigwedge x. M(x) \Longrightarrow P(x) <-> P'(x)) \Longrightarrow (\forall x[M]. P(x)) <-> (\forall x[M]. P'(x))$
 $\langle proof \rangle$

15.2.2 Relativized existential quantifier

lemma *rexI* [intro]: $\llbracket P(x); M(x) \rrbracket \Longrightarrow \exists x[M]. P(x)$
 $\langle proof \rangle$

lemma *rev-rexI*: $\llbracket M(x); P(x) \rrbracket \Longrightarrow \exists x[M]. P(x)$
 $\langle proof \rangle$

lemma *rexCI*: $\llbracket \forall x[M]. \neg P(x) \Longrightarrow P(a); M(a) \rrbracket \Longrightarrow \exists x[M]. P(x)$
 $\langle proof \rangle$

lemma *rexE* [*elim!*]: $\llbracket \exists x[M]. P(x); \bigwedge x. \llbracket M(x); P(x) \rrbracket \implies Q \rrbracket \implies Q$
 $\langle \text{proof} \rangle$

lemma *rex-triv* [*simp*]: $(\exists x[M]. P) \longleftrightarrow ((\exists x. M(x)) \wedge P)$
 $\langle \text{proof} \rangle$

lemma *rex-cong* [*cong*]:
 $(\bigwedge x. M(x) \implies P(x) <-> P'(x)) \implies (\exists x[M]. P(x)) <-> (\exists x[M]. P'(x))$
 $\langle \text{proof} \rangle$

lemma *rall-is-ball* [*simp*]: $(\forall x[\lambda z. z \in A]. P(x)) <-> (\forall x \in A. P(x))$
 $\langle \text{proof} \rangle$

lemma *rex-is-bex* [*simp*]: $(\exists x[\lambda z. z \in A]. P(x)) <-> (\exists x \in A. P(x))$
 $\langle \text{proof} \rangle$

lemma *atomize-rall*: $(\bigwedge x. M(x) \implies P(x)) \equiv \text{Trueprop } (\forall x[M]. P(x))$
 $\langle \text{proof} \rangle$

declare *atomize-rall* [*symmetric, rulify*]

lemma *rall-simps1*:
 $(\forall x[M]. P(x) \wedge Q) <-> (\forall x[M]. P(x)) \wedge ((\forall x[M]. \text{False}) \mid Q)$
 $(\forall x[M]. P(x) \mid Q) <-> ((\forall x[M]. P(x)) \mid Q)$
 $(\forall x[M]. P(x) \longrightarrow Q) <-> ((\exists x[M]. P(x)) \longrightarrow Q)$
 $(\neg(\forall x[M]. P(x))) <-> (\exists x[M]. \neg P(x))$
 $\langle \text{proof} \rangle$

lemma *rall-simps2*:
 $(\forall x[M]. P \wedge Q(x)) <-> ((\forall x[M]. \text{False}) \mid P) \wedge (\forall x[M]. Q(x))$
 $(\forall x[M]. P \mid Q(x)) <-> (P \mid (\forall x[M]. Q(x)))$
 $(\forall x[M]. P \longrightarrow Q(x)) <-> (P \longrightarrow (\forall x[M]. Q(x)))$
 $\langle \text{proof} \rangle$

lemmas *rall-simps* [*simp*] = *rall-simps1 rall-simps2*

lemma *rall-conj-distrib*:
 $(\forall x[M]. P(x) \wedge Q(x)) <-> ((\forall x[M]. P(x)) \wedge (\forall x[M]. Q(x)))$
 $\langle \text{proof} \rangle$

lemma *rex-simps1*:
 $(\exists x[M]. P(x) \wedge Q) <-> ((\exists x[M]. P(x)) \wedge Q)$
 $(\exists x[M]. P(x) \mid Q) <-> (\exists x[M]. P(x)) \mid ((\exists x[M]. \text{True}) \wedge Q)$
 $(\exists x[M]. P(x) \longrightarrow Q) <-> ((\forall x[M]. P(x)) \longrightarrow ((\exists x[M]. \text{True}) \wedge Q))$
 $(\neg(\exists x[M]. P(x))) <-> (\forall x[M]. \neg P(x))$
 $\langle \text{proof} \rangle$

lemma *rex-simps2*:

$(\exists x[M]. P \wedge Q(x)) <-> (P \wedge (\exists x[M]. Q(x)))$
 $(\exists x[M]. P \mid Q(x)) <-> ((\exists x[M]. \text{True}) \wedge P) \mid (\exists x[M]. Q(x))$
 $(\exists x[M]. P \longrightarrow Q(x)) <-> (((\forall x[M]. \text{False}) \mid P) \longrightarrow (\exists x[M]. Q(x)))$
 $\langle \text{proof} \rangle$

lemmas *rex-simps* [simp] = *rex-simps1* *rex-simps2*

lemma *rex-disj-distrib*:

$(\exists x[M]. P(x) \mid Q(x)) <-> ((\exists x[M]. P(x)) \mid (\exists x[M]. Q(x)))$
 $\langle \text{proof} \rangle$

15.2.3 One-point rule for bounded quantifiers

lemma *rex-triv-one-point1* [simp]: $(\exists x[M]. x=a) <-> (M(a))$
 $\langle \text{proof} \rangle$

lemma *rex-triv-one-point2* [simp]: $(\exists x[M]. a=x) <-> (M(a))$
 $\langle \text{proof} \rangle$

lemma *rex-one-point1* [simp]: $(\exists x[M]. x=a \wedge P(x)) <-> (M(a) \wedge P(a))$
 $\langle \text{proof} \rangle$

lemma *rex-one-point2* [simp]: $(\exists x[M]. a=x \wedge P(x)) <-> (M(a) \wedge P(a))$
 $\langle \text{proof} \rangle$

lemma *rall-one-point1* [simp]: $(\forall x[M]. x=a \longrightarrow P(x)) <-> (M(a) \longrightarrow P(a))$
 $\langle \text{proof} \rangle$

lemma *rall-one-point2* [simp]: $(\forall x[M]. a=x \longrightarrow P(x)) <-> (M(a) \longrightarrow P(a))$
 $\langle \text{proof} \rangle$

15.2.4 Sets as Classes

definition

setclass :: $[i,i] \Rightarrow o$ ($\langle \langle \text{open-block notation} = \langle \text{prefix setclass} \rangle \rangle \#\#\text{-} \rangle$ [40] 40)

where

setclass(A) $\equiv \lambda x. x \in A$

lemma *setclass-iff* [simp]: *setclass*(A,x) $<-> x \in A$
 $\langle \text{proof} \rangle$

lemma *rall-setclass-is-ball* [simp]: $(\forall x[\#\#A]. P(x)) <-> (\forall x \in A. P(x))$
 $\langle \text{proof} \rangle$

lemma *rex-setclass-is-bex* [simp]: $(\exists x[\#\#A]. P(x)) <-> (\exists x \in A. P(x))$
 $\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

Setting up the one-point-rule simproc

$\langle ML \rangle$

end

16 The Natural numbers As a Least Fixed Point

theory *Nat* **imports** *OrdQuant Bool* **begin**

definition

nat :: *i* **where**
nat $\equiv \text{lfp}(\text{Inf}, \lambda X. \{0\} \cup \{\text{succ}(i). i \in X\})$

definition

quasinat :: *i* \Rightarrow *o* **where**
quasinat(*n*) $\equiv n=0 \mid (\exists m. n = \text{succ}(m))$

definition

nat-case :: [*i*, *i* \Rightarrow *i*, *i*] \Rightarrow *i* **where**
nat-case(*a*, *b*, *k*) $\equiv \text{THE } y. k=0 \wedge y=a \mid (\exists x. k=\text{succ}(x) \wedge y=b(x))$

definition

nat-rec :: [*i*, *i*, [*i*, *i*] \Rightarrow *i*] \Rightarrow *i* **where**
nat-rec(*k*, *a*, *b*) \equiv
wfrec(*Memrel*(*nat*), *k*, $\lambda n f. \text{nat-case}(a, \lambda m. b(m, f'm), n)$)

definition

Le :: *i* **where**
Le $\equiv \{\langle x, y \rangle : \text{nat} * \text{nat}. x \leq y\}$

definition

Lt :: *i* **where**
Lt $\equiv \{\langle x, y \rangle : \text{nat} * \text{nat}. x < y\}$

definition

Ge :: *i* **where**
Ge $\equiv \{\langle x, y \rangle : \text{nat} * \text{nat}. y \leq x\}$

definition

Gt :: *i* **where**
Gt $\equiv \{\langle x, y \rangle : \text{nat} * \text{nat}. y < x\}$

definition

greater-than :: *i* \Rightarrow *i* **where**
greater-than(*n*) $\equiv \{i \in \text{nat}. n < i\}$

No need for a less-than operator: a natural number is its list of predecessors!

lemma *nat-bnd-mono*: *bnd-mono*(*Inf*, $\lambda X. \{0\} \cup \{\text{succ}(i). i \in X\}$)
 $\langle \text{proof} \rangle$

lemmas *nat-unfold* = *nat-bnd-mono* [*THEN nat-def* [*THEN def-lfp-unfold*]]

lemma *nat-0I* [*iff*, *TC*]: $0 \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-succI* [*intro!*, *TC*]: $n \in \text{nat} \implies \text{succ}(n) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-1I* [*iff*, *TC*]: $1 \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-2I* [*iff*, *TC*]: $2 \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *bool-subset-nat*: $\text{bool} \subseteq \text{nat}$
 $\langle \text{proof} \rangle$

lemmas *bool-into-nat* = *bool-subset-nat* [*THEN subsetD*]

16.1 Injectivity Properties and Induction

lemma *nat-induct* [*case-names* *0 succ*, *induct set*: *nat*]:
 $\llbracket n \in \text{nat}; P(0); \bigwedge x. \llbracket x \in \text{nat}; P(x) \rrbracket \implies P(\text{succ}(x)) \rrbracket \implies P(n)$
 $\langle \text{proof} \rangle$

lemma *natE*:
assumes $n \in \text{nat}$
obtains $(0) \ n=0 \mid (\text{succ}) \ x \text{ where } x \in \text{nat} \ n=\text{succ}(x)$
 $\langle \text{proof} \rangle$

lemma *nat-into-Ord* [*simp*]: $n \in \text{nat} \implies \text{Ord}(n)$
 $\langle \text{proof} \rangle$

lemmas *nat-0-le* = *nat-into-Ord* [*THEN Ord-0-le*]

lemmas *nat-le-refl* = *nat-into-Ord* [*THEN le-refl*]

lemma *Ord-nat* [*iff*]: $\text{Ord}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *Limit-nat* [iff]: $\text{Limit}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *naturals-not-limit*: $a \in \text{nat} \implies \neg \text{Limit}(a)$
 $\langle \text{proof} \rangle$

lemma *succ-natD*: $\text{succ}(i): \text{nat} \implies i \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-succ-iff* [iff]: $\text{succ}(n): \text{nat} \longleftrightarrow n \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-le-Limit*: $\text{Limit}(i) \implies \text{nat} \leq i$
 $\langle \text{proof} \rangle$

lemmas *succ-in-naturalD* = *Ord-trans* [OF *succI1* - *nat-into-Ord*]

lemma *lt-nat-in-nat*: $\llbracket m < n; n \in \text{nat} \rrbracket \implies m \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *le-in-nat*: $\llbracket m \leq n; n \in \text{nat} \rrbracket \implies m \in \text{nat}$
 $\langle \text{proof} \rangle$

16.2 Variations on Mathematical Induction

lemmas *complete-induct* = *Ord-induct* [OF - *Ord-nat*, *case-names less*, *consumes 1*]

lemma *complete-induct-rule* [*case-names less*, *consumes 1*]:
 $i \in \text{nat} \implies (\bigwedge x. x \in \text{nat} \implies (\bigwedge y. y \in x \implies P(y)) \implies P(x)) \implies P(i)$
 $\langle \text{proof} \rangle$

lemma *nat-induct-from*:
assumes $m \leq n \ m \in \text{nat} \ n \in \text{nat}$
and $P(m)$
and $\bigwedge x. \llbracket x \in \text{nat}; m \leq x; P(x) \rrbracket \implies P(\text{succ}(x))$
shows $P(n)$
 $\langle \text{proof} \rangle$

lemma *diff-induct* [*case-names 0 0-succ succ-succ*, *consumes 2*]:
 $\llbracket m \in \text{nat}; n \in \text{nat};$
 $\bigwedge x. x \in \text{nat} \implies P(x, 0);$
 $\bigwedge y. y \in \text{nat} \implies P(0, \text{succ}(y));$
 $\bigwedge x y. \llbracket x \in \text{nat}; y \in \text{nat}; P(x, y) \rrbracket \implies P(\text{succ}(x), \text{succ}(y)) \rrbracket$
 $\implies P(m, n)$
 $\langle \text{proof} \rangle$

lemma *succ-lt-induct-lemma* [rule-format]:

$$m \in \text{nat} \implies P(m, \text{succ}(m)) \longrightarrow (\forall x \in \text{nat}. P(m, x) \longrightarrow P(m, \text{succ}(x))) \longrightarrow \\ (\forall n \in \text{nat}. m < n \longrightarrow P(m, n))$$

$\langle \text{proof} \rangle$

lemma *succ-lt-induct*:

$$\llbracket m < n; \ n \in \text{nat}; \\ P(m, \text{succ}(m)); \\ \bigwedge x. \llbracket x \in \text{nat}; \ P(m, x) \rrbracket \implies P(m, \text{succ}(x)) \rrbracket \\ \implies P(m, n)$$

$\langle \text{proof} \rangle$

16.3 quasinat: to allow a case-split rule for *nat-case*

True if the argument is zero or any successor

lemma [iff]: *quasinat*(0)

$\langle \text{proof} \rangle$

lemma [iff]: *quasinat*(*succ*(*x*))

$\langle \text{proof} \rangle$

lemma *nat-imp-quasinat*: $n \in \text{nat} \implies \text{quasinat}(n)$

$\langle \text{proof} \rangle$

lemma *non-nat-case*: $\neg \text{quasinat}(x) \implies \text{nat-case}(a, b, x) = 0$

$\langle \text{proof} \rangle$

lemma *nat-cases-disj*: $k=0 \mid (\exists y. k = \text{succ}(y)) \mid \neg \text{quasinat}(k)$

$\langle \text{proof} \rangle$

lemma *nat-cases*:

$$\llbracket k=0 \implies P; \ \bigwedge y. k = \text{succ}(y) \implies P; \ \neg \text{quasinat}(k) \implies P \rrbracket \implies P$$

$\langle \text{proof} \rangle$

lemma *nat-case-0* [simp]: $\text{nat-case}(a, b, 0) = a$

$\langle \text{proof} \rangle$

lemma *nat-case-succ* [simp]: $\text{nat-case}(a, b, \text{succ}(n)) = b(n)$

$\langle \text{proof} \rangle$

lemma *nat-case-type* [TC]:

$$\llbracket n \in \text{nat}; \ a \in C(0); \ \bigwedge m. m \in \text{nat} \implies b(m): C(\text{succ}(m)) \rrbracket \\ \implies \text{nat-case}(a, b, n) \in C(n)$$

$\langle proof \rangle$

lemma *split-nat-case*:

$P(\text{nat-case}(a,b,k)) \longleftrightarrow$
 $((k=0 \longrightarrow P(a)) \wedge (\forall x. k=\text{succ}(x) \longrightarrow P(b(x))) \wedge (\neg \text{quasinat}(k) \longrightarrow P(0)))$
 $\langle proof \rangle$

16.4 Recursion on the Natural Numbers

lemma *nat-rec-0*: $\text{nat-rec}(0,a,b) = a$

$\langle proof \rangle$

lemma *nat-rec-succ*: $m \in \text{nat} \implies \text{nat-rec}(\text{succ}(m),a,b) = b(m, \text{nat-rec}(m,a,b))$

$\langle proof \rangle$

lemma *Un-nat-type* [TC]: $\llbracket i \in \text{nat}; j \in \text{nat} \rrbracket \implies i \cup j \in \text{nat}$

$\langle proof \rangle$

lemma *Int-nat-type* [TC]: $\llbracket i \in \text{nat}; j \in \text{nat} \rrbracket \implies i \cap j \in \text{nat}$

$\langle proof \rangle$

lemma *nat-nonempty* [simp]: $\text{nat} \neq 0$

$\langle proof \rangle$

A natural number is the set of its predecessors

lemma *nat-eq-Collect-lt*: $i \in \text{nat} \implies \{j \in \text{nat}. j < i\} = i$

$\langle proof \rangle$

lemma *Le-iff* [iff]: $\langle x,y \rangle \in \text{Le} \longleftrightarrow x \leq y \wedge x \in \text{nat} \wedge y \in \text{nat}$

$\langle proof \rangle$

end

17 Inductive and Coinductive Definitions

theory *Inductive*

imports *Fixedpt QPair Nat*

keywords

inductive coinductive inductive-cases rep-datatype primrec :: thy-decl and

domains intros monos con-defs type-intros type-elim

elimination induction case-eqns recursor-eqns :: quasi-command

begin

lemma *def-swap-iff*: $a \equiv b \implies a = c \longleftrightarrow c = b$

$\langle proof \rangle$

lemma *def-trans*: $f \equiv g \implies g(a) = b \implies f(a) = b$
 $\langle proof \rangle$

lemma *refl-thin*: $\bigwedge P. a = a \implies P \implies P \langle proof \rangle$

$\langle ML \rangle$

end

18 Epsilon Induction and Recursion

theory *Epsilon* **imports** *Nat* **begin**

definition

eclose $:: i \Rightarrow i$ **where**
 $eclose(A) \equiv \bigcup_{n \in nat.} nat-rec(n, A, \lambda m r. \bigcup (r))$

definition

transrec $:: [i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $transrec(a, H) \equiv wfrec(Memrel(eclose(\{a\})), a, H)$

definition

rank $:: i \Rightarrow i$ **where**
 $rank(a) \equiv transrec(a, \lambda x f. \bigcup_{y \in x. succ(f'y))$

definition

transrec2 $:: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $transrec2(k, a, b) \equiv$
 $transrec(k,$
 $\lambda i r. if(i=0, a,$
 $if(\exists j. i=succ(j),$
 $b(THE j. i=succ(j), r'(THE j. i=succ(j))),$
 $\bigcup_{j < i. r'j)))$

definition

recursor $:: [i, [i, i] \Rightarrow i, i] \Rightarrow i$ **where**
 $recursor(a, b, k) \equiv transrec(k, \lambda n f. nat-case(a, \lambda m. b(m, f'm), n))$

definition

rec $:: [i, i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $rec(k, a, b) \equiv recursor(a, b, k)$

18.1 Basic Closure Properties

lemma *arg-subset-eclose*: $A \subseteq eclose(A)$
 $\langle proof \rangle$

lemmas *arg-into-eclose* = *arg-subset-eclose* $[THEN subsetD]$

lemma *Transset-eclose*: $\text{Transset}(\text{eclose}(A))$
 $\langle \text{proof} \rangle$

lemmas *eclose-subset* =
 $\text{Transset-eclose} \text{ [unfolded Transset-def, THEN bspec]}$

lemmas $\text{eclose}D = \text{eclose-subset} \text{ [THEN subsetD]}$

lemmas $\text{arg-in-eclose-sing} = \text{arg-subset-eclose} \text{ [THEN singleton-subsetD]}$
lemmas $\text{arg-into-eclose-sing} = \text{arg-in-eclose-sing} \text{ [THEN ecloseD]}$

lemmas *eclose-induct* =
 $\text{Transset-induct} \text{ [OF - Transset-eclose, induct set: eclose]}$

lemma *eps-induct*:
 $\llbracket \bigwedge x. \forall y \in x. P(y) \implies P(x) \rrbracket \implies P(a)$
 $\langle \text{proof} \rangle$

18.2 Leastness of *eclose*

lemma *eclose-least-lemma*:
 $\llbracket \text{Transset}(X); A \leq X; n \in \text{nat} \rrbracket \implies \text{nat-rec}(n, A, \lambda m r. \bigcup(r)) \subseteq X$
 $\langle \text{proof} \rangle$

lemma *eclose-least*:
 $\llbracket \text{Transset}(X); A \leq X \rrbracket \implies \text{eclose}(A) \subseteq X$
 $\langle \text{proof} \rangle$

lemma *eclose-induct-down* [consumes 1]:
 $\llbracket a \in \text{eclose}(b);$
 $\bigwedge y. \llbracket y \in b \rrbracket \implies P(y);$
 $\bigwedge y z. \llbracket y \in \text{eclose}(b); P(y); z \in y \rrbracket \implies P(z)$
 $\rrbracket \implies P(a)$
 $\langle \text{proof} \rangle$

lemma *Transset-eclose-eq-arg*: $\text{Transset}(X) \implies \text{eclose}(X) = X$
 $\langle \text{proof} \rangle$

A transitive set either is empty or contains the empty set.

lemma *Transset-0-lemma* [rule-format]: $\text{Transset}(A) \implies x \in A \longrightarrow 0 \in A$
 $\langle \text{proof} \rangle$

lemma *Transset-0-disj*: $\text{Transset}(A) \implies A = 0 \mid 0 \in A$

$\langle proof \rangle$

18.3 Epsilon Recursion

lemma *mem-eclose-trans*: $\llbracket A \in \text{eclose}(B); B \in \text{eclose}(C) \rrbracket \implies A \in \text{eclose}(C)$
 $\langle proof \rangle$

lemma *mem-eclose-sing-trans*:
 $\llbracket A \in \text{eclose}(\{B\}); B \in \text{eclose}(\{C\}) \rrbracket \implies A \in \text{eclose}(\{C\})$
 $\langle proof \rangle$

lemma *under-Memrel*: $\llbracket \text{Transset}(i); j \in i \rrbracket \implies \text{Memrel}(i) - \{\{j\} = j$
 $\langle proof \rangle$

lemma *lt-Memrel*: $j < i \implies \text{Memrel}(i) - \{\{j\} = j$
 $\langle proof \rangle$

lemmas *under-Memrel-eclose* = *Transset-eclose* [THEN *under-Memrel*]

lemmas *wfrec-ssubst* = *wf-Memrel* [THEN *wfrec*, THEN *ssubst*]

lemma *wfrec-eclose-eq*:
 $\llbracket k \in \text{eclose}(\{j\}); j \in \text{eclose}(\{i\}) \rrbracket \implies$
 $\text{wfrec}(\text{Memrel}(\text{eclose}(\{i\})), k, H) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{j\})), k, H)$
 $\langle proof \rangle$

lemma *wfrec-eclose-eq2*:
 $k \in i \implies \text{wfrec}(\text{Memrel}(\text{eclose}(\{i\})), k, H) = \text{wfrec}(\text{Memrel}(\text{eclose}(\{k\})), k, H)$
 $\langle proof \rangle$

lemma *transrec*: $\text{transrec}(a, H) = H(a, \lambda x \in a. \text{transrec}(x, H))$
 $\langle proof \rangle$

lemma *def-transrec*:
 $\llbracket \bigwedge x. f(x) \equiv \text{transrec}(x, H) \rrbracket \implies f(a) = H(a, \lambda x \in a. f(x))$
 $\langle proof \rangle$

lemma *transrec-type*:
 $\llbracket \bigwedge x u. \llbracket x \in \text{eclose}(\{a\}); u \in \text{Pi}(x, B) \rrbracket \implies H(x, u) \in B(x) \rrbracket$
 $\implies \text{transrec}(a, H) \in B(a)$
 $\langle proof \rangle$

lemma *eclose-sing-Ord*: $\text{Ord}(i) \implies \text{eclose}(\{i\}) \subseteq \text{succ}(i)$
 $\langle proof \rangle$

lemma *succ-subset-eclose-sing*: $\text{succ}(i) \subseteq \text{eclose}(\{i\})$

$\langle proof \rangle$

lemma *eclose-sing-Ord-eq*: $Ord(i) \implies eclose(\{i\}) = succ(i)$
 $\langle proof \rangle$

lemma *Ord-transrec-type*:
 assumes *jini*: $j \in i$
 and *ordi*: $Ord(i)$
 and *minor*: $\bigwedge x u. \llbracket x \in i; u \in Pi(x, B) \rrbracket \implies H(x, u) \in B(x)$
 shows $transrec(j, H) \in B(j)$
 $\langle proof \rangle$

18.4 Rank

lemma *rank*: $rank(a) = (\bigcup y \in a. succ(rank(y)))$
 $\langle proof \rangle$

lemma *Ord-rank [simp]*: $Ord(rank(a))$
 $\langle proof \rangle$

lemma *rank-of-Ord*: $Ord(i) \implies rank(i) = i$
 $\langle proof \rangle$

lemma *rank-lt*: $a \in b \implies rank(a) < rank(b)$
 $\langle proof \rangle$

lemma *eclose-rank-lt*: $a \in eclose(b) \implies rank(a) < rank(b)$
 $\langle proof \rangle$

lemma *rank-mono*: $a \leq b \implies rank(a) \leq rank(b)$
 $\langle proof \rangle$

lemma *rank-Pow*: $rank(Pow(a)) = succ(rank(a))$
 $\langle proof \rangle$

lemma *rank-0 [simp]*: $rank(0) = 0$
 $\langle proof \rangle$

lemma *rank-succ [simp]*: $rank(succ(x)) = succ(rank(x))$
 $\langle proof \rangle$

lemma *rank-Union*: $rank(\bigcup(A)) = (\bigcup x \in A. rank(x))$
 $\langle proof \rangle$

lemma *rank-eclose*: $rank(eclose(a)) = rank(a)$
 $\langle proof \rangle$

lemma *rank-pair1*: $rank(a) < rank(\langle a, b \rangle)$
 $\langle proof \rangle$

lemma *rank-pair2*: $\text{rank}(b) < \text{rank}(\langle a, b \rangle)$
 $\langle \text{proof} \rangle$

lemma *the-equality-if*:
 $P(a) \implies (\text{THE } x. P(x)) = (\text{if } (\exists !x. P(x)) \text{ then } a \text{ else } 0)$
 $\langle \text{proof} \rangle$

lemma *rank-apply*: $\llbracket i \in \text{domain}(f); \text{function}(f) \rrbracket \implies \text{rank}(f'i) < \text{rank}(f)$
 $\langle \text{proof} \rangle$

18.5 Corollaries of Leastness

lemma *mem-eclose-subset*: $A \in B \implies \text{eclose}(A) \leq \text{eclose}(B)$
 $\langle \text{proof} \rangle$

lemma *eclose-mono*: $A \leq B \implies \text{eclose}(A) \subseteq \text{eclose}(B)$
 $\langle \text{proof} \rangle$

lemma *eclose-idem*: $\text{eclose}(\text{eclose}(A)) = \text{eclose}(A)$
 $\langle \text{proof} \rangle$

lemma *transrec2-0* [simp]: $\text{transrec2}(0, a, b) = a$
 $\langle \text{proof} \rangle$

lemma *transrec2-succ* [simp]: $\text{transrec2}(\text{succ}(i), a, b) = b(i, \text{transrec2}(i, a, b))$
 $\langle \text{proof} \rangle$

lemma *transrec2-Limit*:
 $\text{Limit}(i) \implies \text{transrec2}(i, a, b) = (\bigcup j < i. \text{transrec2}(j, a, b))$
 $\langle \text{proof} \rangle$

lemma *def-transrec2*:
 $(\bigwedge x. f(x) \equiv \text{transrec2}(x, a, b))$
 $\implies f(0) = a \wedge$
 $f(\text{succ}(i)) = b(i, f(i)) \wedge$
 $(\text{Limit}(K) \longrightarrow f(K) = (\bigcup j < K. f(j)))$
 $\langle \text{proof} \rangle$

lemmas *recursor-lemma* = *recursor-def* [*THEN* *def-transrec*, *THEN* *trans*]

lemma *recursor-0*: $\text{recursor}(a, b, 0) = a$
 $\langle \text{proof} \rangle$

lemma *recursor-succ*: $\text{recursor}(a, b, \text{succ}(m)) = b(m, \text{recursor}(a, b, m))$
 $\langle \text{proof} \rangle$

lemma *rec-0* [*simp*]: $\text{rec}(0, a, b) = a$
 $\langle \text{proof} \rangle$

lemma *rec-succ* [*simp*]: $\text{rec}(\text{succ}(m), a, b) = b(m, \text{rec}(m, a, b))$
 $\langle \text{proof} \rangle$

lemma *rec-type*:

$$\begin{aligned} & \llbracket n \in \text{nat}; \\ & \quad a \in C(0); \\ & \quad \bigwedge m \ z. \llbracket m \in \text{nat}; \ z \in C(m) \rrbracket \implies b(m, z) \in C(\text{succ}(m)) \rrbracket \\ & \implies \text{rec}(n, a, b) \in C(n) \end{aligned}$$

 $\langle \text{proof} \rangle$

end

19 Partial and Total Orderings: Basic Definitions and Properties

theory *Order* **imports** *WF Perm* **begin**

We adopt the following convention: *ord* is used for strict orders and *order* is used for their reflexive counterparts.

definition

part-ord :: $[i, i] \Rightarrow o$ **where**
 $\text{part-ord}(A, r) \equiv \text{irrefl}(A, r) \wedge \text{trans}[A](r)$

definition

linear :: $[i, i] \Rightarrow o$ **where**
 $\text{linear}(A, r) \equiv (\forall x \in A. \forall y \in A. \langle x, y \rangle : r \mid x = y \mid \langle y, x \rangle : r)$

definition

tot-ord :: $[i, i] \Rightarrow o$ **where**
 $\text{tot-ord}(A, r) \equiv \text{part-ord}(A, r) \wedge \text{linear}(A, r)$

definition

preorder-on(*A*, *r*) $\equiv \text{refl}(A, r) \wedge \text{trans}[A](r)$

definition

$$\text{partial-order-on}(A, r) \equiv \text{preorder-on}(A, r) \wedge \text{antisym}(r)$$

abbreviation

$$\text{Preorder}(r) \equiv \text{preorder-on}(\text{field}(r), r)$$

abbreviation

$$\text{Partial-order}(r) \equiv \text{partial-order-on}(\text{field}(r), r)$$

definition

$$\begin{aligned} \text{well-ord} &:: [i, i, i] \Rightarrow o & \text{where} \\ \text{well-ord}(A, r) &\equiv \text{tot-ord}(A, r) \wedge \text{wf}[A](r) \end{aligned}$$

definition

$$\begin{aligned} \text{mono-map} &:: [i, i, i, i] \Rightarrow i & \text{where} \\ \text{mono-map}(A, r, B, s) &\equiv \\ &\{f \in A \rightarrow B. \forall x \in A. \forall y \in A. \langle x, y \rangle : r \longrightarrow \langle f'x, f'y \rangle : s\} \end{aligned}$$

definition

$$\begin{aligned} \text{ord-iso} &:: [i, i, i, i] \Rightarrow i \quad (\langle \text{notation} = \langle \text{infix ord-iso} \rangle \langle -, - \rangle \cong / \langle -, - \rangle \rangle \ 51) & \text{where} \\ \langle A, r \rangle &\cong \langle B, s \rangle \equiv \\ &\{f \in \text{bij}(A, B). \forall x \in A. \forall y \in A. \langle x, y \rangle : r \longleftrightarrow \langle f'x, f'y \rangle : s\} \end{aligned}$$

definition

$$\begin{aligned} \text{pred} &:: [i, i, i] \Rightarrow i & \text{where} \\ \text{pred}(A, x, r) &\equiv \{y \in A. \langle y, x \rangle : r\} \end{aligned}$$

definition

$$\begin{aligned} \text{ord-iso-map} &:: [i, i, i, i] \Rightarrow i & \text{where} \\ \text{ord-iso-map}(A, r, B, s) &\equiv \\ &\bigcup x \in A. \bigcup y \in B. \bigcup f \in \text{ord-iso}(\text{pred}(A, x, r), r, \text{pred}(B, y, s), s). \{\langle x, y \rangle\} \end{aligned}$$

definition

$$\begin{aligned} \text{first} &:: [i, i, i] \Rightarrow o & \text{where} \\ \text{first}(u, X, R) &\equiv u \in X \wedge (\forall v \in X. v \neq u \longrightarrow \langle u, v \rangle \in R) \end{aligned}$$

19.1 Immediate Consequences of the Definitions

lemma *part-ord-Imp-asym*:

$$\text{part-ord}(A, r) \implies \text{asym}(r \cap A * A)$$

<proof>

lemma *linearE*:

$$\begin{aligned} &\llbracket \text{linear}(A, r); \ x \in A; \ y \in A; \\ &\quad \langle x, y \rangle : r \implies P; \ x = y \implies P; \ \langle y, x \rangle : r \implies P \rrbracket \\ &\implies P \end{aligned}$$

<proof>

lemma *well-ordI*:

$\llbracket wf[A](r); linear(A,r) \rrbracket \implies well-ord(A,r)$
 $\langle proof \rangle$

lemma *well-ord-is-wf*:

$well-ord(A,r) \implies wf[A](r)$
 $\langle proof \rangle$

lemma *well-ord-is-trans-on*:

$well-ord(A,r) \implies trans[A](r)$
 $\langle proof \rangle$

lemma *well-ord-is-linear*: $well-ord(A,r) \implies linear(A,r)$

$\langle proof \rangle$

lemma *pred-iff*: $y \in pred(A,x,r) \longleftrightarrow \langle y,x \rangle : r \wedge y \in A$

$\langle proof \rangle$

lemmas *predI* = *conjI* [*THEN* *pred-iff* [*THEN* *iffD2*]]

lemma *predE*: $\llbracket y \in pred(A,x,r); \llbracket y \in A; \langle y,x \rangle : r \rrbracket \implies P \rrbracket \implies P$

$\langle proof \rangle$

lemma *pred-subset-under*: $pred(A,x,r) \subseteq r - \{x\}$

$\langle proof \rangle$

lemma *pred-subset*: $pred(A,x,r) \subseteq A$

$\langle proof \rangle$

lemma *pred-pred-eq*:

$pred(pred(A,x,r), y, r) = pred(A,x,r) \cap pred(A,y,r)$
 $\langle proof \rangle$

lemma *trans-pred-pred-eq*:

$\llbracket trans[A](r); \langle y,x \rangle : r; x \in A; y \in A \rrbracket$
 $\implies pred(pred(A,x,r), y, r) = pred(A,y,r)$
 $\langle proof \rangle$

19.2 Restricting an Ordering's Domain

lemma *part-ord-subset*:

$\llbracket part-ord(A,r); B \leq A \rrbracket \implies part-ord(B,r)$
 $\langle proof \rangle$

lemma *linear-subset*:

$\llbracket \text{linear}(A, r); B \leq A \rrbracket \implies \text{linear}(B, r)$
 $\langle \text{proof} \rangle$

lemma *tot-ord-subset*:

$\llbracket \text{tot-ord}(A, r); B \leq A \rrbracket \implies \text{tot-ord}(B, r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-subset*:

$\llbracket \text{well-ord}(A, r); B \leq A \rrbracket \implies \text{well-ord}(B, r)$
 $\langle \text{proof} \rangle$

lemma *irrefl-Int-iff*: $\text{irrefl}(A, r \cap A * A) \longleftrightarrow \text{irrefl}(A, r)$

$\langle \text{proof} \rangle$

lemma *trans-on-Int-iff*: $\text{trans}[A](r \cap A * A) \longleftrightarrow \text{trans}[A](r)$

$\langle \text{proof} \rangle$

lemma *part-ord-Int-iff*: $\text{part-ord}(A, r \cap A * A) \longleftrightarrow \text{part-ord}(A, r)$

$\langle \text{proof} \rangle$

lemma *linear-Int-iff*: $\text{linear}(A, r \cap A * A) \longleftrightarrow \text{linear}(A, r)$

$\langle \text{proof} \rangle$

lemma *tot-ord-Int-iff*: $\text{tot-ord}(A, r \cap A * A) \longleftrightarrow \text{tot-ord}(A, r)$

$\langle \text{proof} \rangle$

lemma *wf-on-Int-iff*: $\text{wf}[A](r \cap A * A) \longleftrightarrow \text{wf}[A](r)$

$\langle \text{proof} \rangle$

lemma *well-ord-Int-iff*: $\text{well-ord}(A, r \cap A * A) \longleftrightarrow \text{well-ord}(A, r)$

$\langle \text{proof} \rangle$

19.3 Empty and Unit Domains

lemma *wf-on-any-0*: $\text{wf}[A](0)$

$\langle \text{proof} \rangle$

19.3.1 Relations over the Empty Set

lemma *irrefl-0*: $\text{irrefl}(0, r)$

$\langle \text{proof} \rangle$

lemma *trans-on-0*: $\text{trans}[0](r)$

$\langle \text{proof} \rangle$

lemma *part-ord-0*: $\text{part-ord}(0, r)$

$\langle proof \rangle$

lemma *linear-0*: $linear(0, r)$
 $\langle proof \rangle$

lemma *tot-ord-0*: $tot-ord(0, r)$
 $\langle proof \rangle$

lemma *wf-on-0*: $wf[0](r)$
 $\langle proof \rangle$

lemma *well-ord-0*: $well-ord(0, r)$
 $\langle proof \rangle$

19.3.2 The Empty Relation Well-Orders the Unit Set

by Grabczewski

lemma *tot-ord-unit*: $tot-ord(\{a\}, 0)$
 $\langle proof \rangle$

lemma *well-ord-unit*: $well-ord(\{a\}, 0)$
 $\langle proof \rangle$

19.4 Order-Isomorphisms

Suppes calls them "similarities"

lemma *mono-map-is-fun*: $f \in mono-map(A, r, B, s) \implies f \in A \multimap B$
 $\langle proof \rangle$

lemma *mono-map-is-inj*:
 $\llbracket linear(A, r); wf[B](s); f \in mono-map(A, r, B, s) \rrbracket \implies f \in inj(A, B)$
 $\langle proof \rangle$

lemma *ord-isoI*:
 $\llbracket f \in bij(A, B);$
 $\bigwedge x y. \llbracket x \in A; y \in A \rrbracket \implies \langle x, y \rangle \in r \longleftrightarrow \langle f'x, f'y \rangle \in s \rrbracket$
 $\implies f \in ord-iso(A, r, B, s)$
 $\langle proof \rangle$

lemma *ord-iso-is-mono-map*:
 $f \in ord-iso(A, r, B, s) \implies f \in mono-map(A, r, B, s)$
 $\langle proof \rangle$

lemma *ord-iso-is-bij*:
 $f \in ord-iso(A, r, B, s) \implies f \in bij(A, B)$
 $\langle proof \rangle$

lemma *ord-iso-apply*:

$\llbracket f \in \text{ord-iso}(A, r, B, s); \langle x, y \rangle: r; x \in A; y \in A \rrbracket \implies \langle f'x, f'y \rangle \in s$
 $\langle \text{proof} \rangle$

lemma *ord-iso-converse*:

$\llbracket f \in \text{ord-iso}(A, r, B, s); \langle x, y \rangle: s; x \in B; y \in B \rrbracket$
 $\implies \langle \text{converse}(f)'x, \text{converse}(f)'y \rangle \in r$
 $\langle \text{proof} \rangle$

lemma *ord-iso-reft*: $\text{id}(A): \text{ord-iso}(A, r, A, r)$

$\langle \text{proof} \rangle$

lemma *ord-iso-sym*: $f \in \text{ord-iso}(A, r, B, s) \implies \text{converse}(f): \text{ord-iso}(B, s, A, r)$

$\langle \text{proof} \rangle$

lemma *mono-map-trans*:

$\llbracket g \in \text{mono-map}(A, r, B, s); f \in \text{mono-map}(B, s, C, t) \rrbracket$
 $\implies (f \circ g): \text{mono-map}(A, r, C, t)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-trans*:

$\llbracket g \in \text{ord-iso}(A, r, B, s); f \in \text{ord-iso}(B, s, C, t) \rrbracket$
 $\implies (f \circ g): \text{ord-iso}(A, r, C, t)$
 $\langle \text{proof} \rangle$

lemma *mono-ord-isoI*:

$\llbracket f \in \text{mono-map}(A, r, B, s); g \in \text{mono-map}(B, s, A, r);$
 $f \circ g = \text{id}(B); g \circ f = \text{id}(A) \rrbracket \implies f \in \text{ord-iso}(A, r, B, s)$
 $\langle \text{proof} \rangle$

lemma *well-ord-mono-ord-isoI*:

$\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s);$
 $f \in \text{mono-map}(A, r, B, s); \text{converse}(f): \text{mono-map}(B, s, A, r) \rrbracket$
 $\implies f \in \text{ord-iso}(A, r, B, s)$
 $\langle \text{proof} \rangle$

lemma *part-ord-ord-iso*:

$\llbracket \text{part-ord}(B,s); f \in \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{part-ord}(A,r)$
 $\langle \text{proof} \rangle$

lemma *linear-ord-iso*:

$\llbracket \text{linear}(B,s); f \in \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{linear}(A,r)$
 $\langle \text{proof} \rangle$

lemma *wf-on-ord-iso*:

$\llbracket \text{wf}[B](s); f \in \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{wf}[A](r)$
 $\langle \text{proof} \rangle$

lemma *well-ord-ord-iso*:

$\llbracket \text{well-ord}(B,s); f \in \text{ord-iso}(A,r,B,s) \rrbracket \implies \text{well-ord}(A,r)$
 $\langle \text{proof} \rangle$

19.5 Main results of Kunen, Chapter 1 section 6

lemma *well-ord-iso-subset-lemma*:

$\llbracket \text{well-ord}(A,r); f \in \text{ord-iso}(A,r, A',r); A' \leq A; y \in A \rrbracket$
 $\implies \neg \langle f'y, y \rangle: r$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-predE*:

$\llbracket \text{well-ord}(A,r); f \in \text{ord-iso}(A, r, \text{pred}(A,x,r), r); x \in A \rrbracket \implies P$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-pred-eq*:

$\llbracket \text{well-ord}(A,r); f \in \text{ord-iso}(\text{pred}(A,a,r), r, \text{pred}(A,c,r), r);$
 $a \in A; c \in A \rrbracket \implies a=c$
 $\langle \text{proof} \rangle$

lemma *ord-iso-image-pred*:

$\llbracket f \in \text{ord-iso}(A,r,B,s); a \in A \rrbracket \implies f `` \text{pred}(A,a,r) = \text{pred}(B, f'a, s)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-restrict-image*:

$\llbracket f \in \text{ord-iso}(A,r,B,s); C \leq A \rrbracket$
 $\implies \text{restrict}(f,C) \in \text{ord-iso}(C, r, f``C, s)$
 $\langle \text{proof} \rangle$

lemma *ord-iso-restrict-pred*:

$\llbracket f \in \text{ord-iso}(A,r,B,s); a \in A \rrbracket$
 $\implies \text{restrict}(f, \text{pred}(A,a,r)) \in \text{ord-iso}(\text{pred}(A,a,r), r, \text{pred}(B, f'a, s), s)$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-preserving*:

$\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s); \langle a, c \rangle: r; \\
f \in \text{ord-iso}(\text{pred}(A, a, r), r, \text{pred}(B, b, s), s); \\
g \in \text{ord-iso}(\text{pred}(A, c, r), r, \text{pred}(B, d, s), s); \\
a \in A; c \in A; b \in B; d \in B \rrbracket \implies \langle b, d \rangle: s$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-unique-lemma*:

$\llbracket \text{well-ord}(A, r); \\
f \in \text{ord-iso}(A, r, B, s); g \in \text{ord-iso}(A, r, B, s); y \in A \rrbracket \\
\implies \neg \langle g'y, f'y \rangle \in s$
 $\langle \text{proof} \rangle$

lemma *well-ord-iso-unique*: $\llbracket \text{well-ord}(A, r);$

$f \in \text{ord-iso}(A, r, B, s); g \in \text{ord-iso}(A, r, B, s) \rrbracket \implies f = g$
 $\langle \text{proof} \rangle$

19.6 Towards Kunen's Theorem 6.3: Linearity of the Similarity Relation

lemma *ord-iso-map-subset*: $\text{ord-iso-map}(A, r, B, s) \subseteq A * B$
 $\langle \text{proof} \rangle$

lemma *domain-ord-iso-map*: $\text{domain}(\text{ord-iso-map}(A, r, B, s)) \subseteq A$
 $\langle \text{proof} \rangle$

lemma *range-ord-iso-map*: $\text{range}(\text{ord-iso-map}(A, r, B, s)) \subseteq B$
 $\langle \text{proof} \rangle$

lemma *converse-ord-iso-map*:
 $\text{converse}(\text{ord-iso-map}(A, r, B, s)) = \text{ord-iso-map}(B, s, A, r)$
 $\langle \text{proof} \rangle$

lemma *function-ord-iso-map*:
 $\text{well-ord}(B, s) \implies \text{function}(\text{ord-iso-map}(A, r, B, s))$
 $\langle \text{proof} \rangle$

lemma *ord-iso-map-fun*: $\text{well-ord}(B, s) \implies \text{ord-iso-map}(A, r, B, s) \\
\in \text{domain}(\text{ord-iso-map}(A, r, B, s)) \rightarrow \text{range}(\text{ord-iso-map}(A, r, B, s))$
 $\langle \text{proof} \rangle$

lemma *ord-iso-map-mono-map*:
 $\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s) \rrbracket \\
\implies \text{ord-iso-map}(A, r, B, s) \\
\in \text{mono-map}(\text{domain}(\text{ord-iso-map}(A, r, B, s)), r,$

$range(ord\text{-}iso\text{-}map(A, r, B, s), s)$

$\langle proof \rangle$

lemma *ord-iso-map-ord-iso*:
 $\llbracket well\text{-}ord(A, r); well\text{-}ord(B, s) \rrbracket \implies ord\text{-}iso\text{-}map(A, r, B, s)$
 $\in ord\text{-}iso(domain(ord\text{-}iso\text{-}map(A, r, B, s)), r,$
 $range(ord\text{-}iso\text{-}map(A, r, B, s), s)$

$\langle proof \rangle$

lemma *domain-ord-iso-map-subset*:
 $\llbracket well\text{-}ord(A, r); well\text{-}ord(B, s);$
 $a \in A; a \notin domain(ord\text{-}iso\text{-}map(A, r, B, s)) \rrbracket$
 $\implies domain(ord\text{-}iso\text{-}map(A, r, B, s)) \subseteq pred(A, a, r)$

$\langle proof \rangle$

lemma *domain-ord-iso-map-cases*:
 $\llbracket well\text{-}ord(A, r); well\text{-}ord(B, s) \rrbracket$
 $\implies domain(ord\text{-}iso\text{-}map(A, r, B, s)) = A \mid$
 $(\exists x \in A. domain(ord\text{-}iso\text{-}map(A, r, B, s)) = pred(A, x, r))$

$\langle proof \rangle$

lemma *range-ord-iso-map-cases*:
 $\llbracket well\text{-}ord(A, r); well\text{-}ord(B, s) \rrbracket$
 $\implies range(ord\text{-}iso\text{-}map(A, r, B, s)) = B \mid$
 $(\exists y \in B. range(ord\text{-}iso\text{-}map(A, r, B, s)) = pred(B, y, s))$

$\langle proof \rangle$

Kunen's Theorem 6.3: Fundamental Theorem for Well-Ordered Sets

theorem *well-ord-trichotomy*:
 $\llbracket well\text{-}ord(A, r); well\text{-}ord(B, s) \rrbracket$
 $\implies ord\text{-}iso\text{-}map(A, r, B, s) \in ord\text{-}iso(A, r, B, s) \mid$
 $(\exists x \in A. ord\text{-}iso\text{-}map(A, r, B, s) \in ord\text{-}iso(pred(A, x, r), r, B, s)) \mid$
 $(\exists y \in B. ord\text{-}iso\text{-}map(A, r, B, s) \in ord\text{-}iso(A, r, pred(B, y, s), s))$

$\langle proof \rangle$

19.7 Miscellaneous Results by Krzysztof Grabczewski

lemma *irrefl-converse*: $irrefl(A, r) \implies irrefl(A, converse(r))$

$\langle proof \rangle$

lemma *trans-on-converse*: $trans[A](r) \implies trans[A](converse(r))$

$\langle proof \rangle$

lemma *part-ord-converse*: $part\text{-}ord(A, r) \implies part\text{-}ord(A, converse(r))$

$\langle proof \rangle$

lemma *linear-converse*: $linear(A, r) \implies linear(A, converse(r))$
 $\langle proof \rangle$

lemma *tot-ord-converse*: $tot-ord(A, r) \implies tot-ord(A, converse(r))$
 $\langle proof \rangle$

lemma *first-is-elem*: $first(b, B, r) \implies b \in B$
 $\langle proof \rangle$

lemma *well-ord-imp-ex1-first*:
 $\llbracket well-ord(A, r); B \leq A; B \neq 0 \rrbracket \implies (\exists ! b. first(b, B, r))$
 $\langle proof \rangle$

lemma *the-first-in*:
 $\llbracket well-ord(A, r); B \leq A; B \neq 0 \rrbracket \implies (THE\ b. first(b, B, r)) \in B$
 $\langle proof \rangle$

19.8 Lemmas for the Reflexive Orders

lemma *subset-vimage-vimage-iff*:
 $\llbracket Preorder(r); A \subseteq field(r); B \subseteq field(r) \rrbracket \implies$
 $r - " A \subseteq r - " B \longleftrightarrow (\forall a \in A. \exists b \in B. \langle a, b \rangle \in r)$
 $\langle proof \rangle$

lemma *subset-vimage1-vimage1-iff*:
 $\llbracket Preorder(r); a \in field(r); b \in field(r) \rrbracket \implies$
 $r - " \{a\} \subseteq r - " \{b\} \longleftrightarrow \langle a, b \rangle \in r$
 $\langle proof \rangle$

lemma *Refl-antisym-eq-Image1-Image1-iff*:
 $\llbracket refl(field(r), r); antisym(r); a \in field(r); b \in field(r) \rrbracket \implies$
 $r - " \{a\} = r - " \{b\} \longleftrightarrow a = b$
 $\langle proof \rangle$

lemma *Partial-order-eq-Image1-Image1-iff*:
 $\llbracket Partial-order(r); a \in field(r); b \in field(r) \rrbracket \implies$
 $r - " \{a\} = r - " \{b\} \longleftrightarrow a = b$
 $\langle proof \rangle$

lemma *Refl-antisym-eq-vimage1-vimage1-iff*:
 $\llbracket refl(field(r), r); antisym(r); a \in field(r); b \in field(r) \rrbracket \implies$
 $r - " \{a\} = r - " \{b\} \longleftrightarrow a = b$
 $\langle proof \rangle$

lemma *Partial-order-eq-vimage1-vimage1-iff*:

$\llbracket \text{Partial-order}(r); a \in \text{field}(r); b \in \text{field}(r) \rrbracket \implies$
 $r - \text{“} \{a\} = r - \text{“} \{b\} \longleftrightarrow a = b$
 $\langle \text{proof} \rangle$

end

20 Combining Orderings: Foundations of Ordinal Arithmetic

theory *OrderArith* **imports** *Order Sum Ordinal* **begin**

definition

$\text{radd} :: [i, i, i, i] \Rightarrow i$ **where**
 $\text{radd}(A, r, B, s) \equiv$
 $\{z: (A+B) * (A+B).$
 $(\exists x y. z = \langle \text{Inl}(x), \text{Inr}(y) \rangle) \mid$
 $(\exists x' x. z = \langle \text{Inl}(x'), \text{Inl}(x) \rangle \wedge \langle x', x \rangle : r) \mid$
 $(\exists y' y. z = \langle \text{Inr}(y'), \text{Inr}(y) \rangle \wedge \langle y', y \rangle : s)\}$

definition

$\text{rmult} :: [i, i, i, i] \Rightarrow i$ **where**
 $\text{rmult}(A, r, B, s) \equiv$
 $\{z: (A*B) * (A*B).$
 $\exists x' y' x y. z = \langle \langle x', y' \rangle, \langle x, y \rangle \rangle \wedge$
 $(\langle x', x \rangle : r \mid (x' = x \wedge \langle y', y \rangle : s))\}$

definition

$\text{rvimage} :: [i, i, i] \Rightarrow i$ **where**
 $\text{rvimage}(A, f, r) \equiv \{z \in A * A. \exists x y. z = \langle x, y \rangle \wedge \langle f'x, f'y \rangle : r\}$

definition

$\text{measure} :: [i, i] \Rightarrow i$ **where**
 $\text{measure}(A, f) \equiv \{\langle x, y \rangle : A * A. f(x) < f(y)\}$

20.1 Addition of Relations – Disjoint Sum

20.1.1 Rewrite rules. Can be used to obtain introduction rules

lemma *radd-Inl-Inr-iff* [iff]:

$\langle \text{Inl}(a), \text{Inr}(b) \rangle \in \text{radd}(A, r, B, s) \longleftrightarrow a \in A \wedge b \in B$
 $\langle \text{proof} \rangle$

lemma *radd-Inl-iff* [iff]:

$\langle \text{Inl}(a'), \text{Inl}(a) \rangle \in \text{radd}(A, r, B, s) \longleftrightarrow a' : A \wedge a \in A \wedge \langle a', a \rangle : r$
 $\langle \text{proof} \rangle$

lemma *radd-Inr-iff* [*iff*]:
 $\langle \text{Inr}(b'), \text{Inr}(b) \rangle \in \text{radd}(A, r, B, s) \longleftrightarrow b':B \wedge b \in B \wedge \langle b', b \rangle : s$
 $\langle \text{proof} \rangle$

lemma *radd-Inr-Inl-iff* [*simp*]:
 $\langle \text{Inr}(b), \text{Inl}(a) \rangle \in \text{radd}(A, r, B, s) \longleftrightarrow \text{False}$
 $\langle \text{proof} \rangle$

declare *radd-Inr-Inl-iff* [*THEN iffD1, dest!*]

20.1.2 Elimination Rule

lemma *raddE*:
 $\llbracket \langle p', p \rangle \in \text{radd}(A, r, B, s);$
 $\bigwedge x y. \llbracket p' = \text{Inl}(x); x \in A; p = \text{Inr}(y); y \in B \rrbracket \implies Q;$
 $\bigwedge x' x. \llbracket p' = \text{Inl}(x'); p = \text{Inl}(x); \langle x', x \rangle : r; x':A; x \in A \rrbracket \implies Q;$
 $\bigwedge y' y. \llbracket p' = \text{Inr}(y'); p = \text{Inr}(y); \langle y', y \rangle : s; y':B; y \in B \rrbracket \implies Q$
 $\rrbracket \implies Q$
 $\langle \text{proof} \rangle$

20.1.3 Type checking

lemma *radd-type*: $\text{radd}(A, r, B, s) \subseteq (A+B) * (A+B)$
 $\langle \text{proof} \rangle$

lemmas *field-radd* = *radd-type* [*THEN field-rel-subset*]

20.1.4 Linearity

lemma *linear-radd*:
 $\llbracket \text{linear}(A, r); \text{linear}(B, s) \rrbracket \implies \text{linear}(A+B, \text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

20.1.5 Well-foundedness

lemma *wf-on-radd*: $\llbracket \text{wf}[A](r); \text{wf}[B](s) \rrbracket \implies \text{wf}[A+B](\text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

lemma *wf-radd*: $\llbracket \text{wf}(r); \text{wf}(s) \rrbracket \implies \text{wf}(\text{radd}(\text{field}(r), r, \text{field}(s), s))$
 $\langle \text{proof} \rangle$

lemma *well-ord-radd*:
 $\llbracket \text{well-ord}(A, r); \text{well-ord}(B, s) \rrbracket \implies \text{well-ord}(A+B, \text{radd}(A, r, B, s))$
 $\langle \text{proof} \rangle$

20.1.6 An ord-iso congruence law

lemma *sum-bij*:
 $\llbracket f \in \text{bij}(A, C); g \in \text{bij}(B, D) \rrbracket$
 $\implies (\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z)) \in \text{bij}(A+B, C+D)$

$\langle proof \rangle$

lemma *sum-ord-iso-cong*:

$\llbracket f \in \text{ord-iso}(A, r, A', r'); g \in \text{ord-iso}(B, s, B', s') \rrbracket \implies$
 $(\lambda z \in A+B. \text{case}(\lambda x. \text{Inl}(f'x), \lambda y. \text{Inr}(g'y), z))$
 $\in \text{ord-iso}(A+B, \text{radd}(A, r, B, s), A'+B', \text{radd}(A', r', B', s'))$
 $\langle proof \rangle$

lemma *sum-disjoint-bij*: $A \cap B = 0 \implies$

$(\lambda z \in A+B. \text{case}(\lambda x. x, \lambda y. y, z)) \in \text{bij}(A+B, A \cup B)$
 $\langle proof \rangle$

20.1.7 Associativity

lemma *sum-assoc-bij*:

$(\lambda z \in (A+B)+C. \text{case}(\text{case}(\text{Inl}, \lambda y. \text{Inr}(\text{Inl}(y))), \lambda y. \text{Inr}(\text{Inr}(y)), z))$
 $\in \text{bij}((A+B)+C, A+(B+C))$
 $\langle proof \rangle$

lemma *sum-assoc-ord-iso*:

$(\lambda z \in (A+B)+C. \text{case}(\text{case}(\text{Inl}, \lambda y. \text{Inr}(\text{Inl}(y))), \lambda y. \text{Inr}(\text{Inr}(y)), z))$
 $\in \text{ord-iso}((A+B)+C, \text{radd}(A+B, \text{radd}(A, r, B, s), C, t),$
 $A+(B+C), \text{radd}(A, r, B+C, \text{radd}(B, s, C, t)))$
 $\langle proof \rangle$

20.2 Multiplication of Relations – Lexicographic Product

20.2.1 Rewrite rule. Can be used to obtain introduction rules

lemma *rmult-iff* [iff]:

$\langle \langle a', b' \rangle, \langle a, b \rangle \rangle \in \text{rmult}(A, r, B, s) \iff$
 $(\langle a', a \rangle: r \wedge a': A \wedge a \in A \wedge b': B \wedge b \in B) \mid$
 $(\langle b', b \rangle: s \wedge a' = a \wedge a \in A \wedge b': B \wedge b \in B)$

$\langle proof \rangle$

lemma *rmultE*:

$\llbracket \langle \langle a', b' \rangle, \langle a, b \rangle \rangle \in \text{rmult}(A, r, B, s);$
 $\llbracket \langle a', a \rangle: r; a': A; a \in A; b': B; b \in B \rrbracket \implies Q;$
 $\llbracket \langle b', b \rangle: s; a \in A; a' = a; b': B; b \in B \rrbracket \implies Q$
 $\rrbracket \implies Q$
 $\langle proof \rangle$

20.2.2 Type checking

lemma *rmult-type*: $\text{rmult}(A, r, B, s) \subseteq (A*B) * (A*B)$

$\langle proof \rangle$

lemmas *field-rmult* = *rmult-type* [THEN *field-rel-subset*]

20.2.3 Linearity

lemma *linear-rmult*:

$$\llbracket \text{linear}(A,r); \text{linear}(B,s) \rrbracket \implies \text{linear}(A*B, \text{rmult}(A,r,B,s))$$

<proof>

20.2.4 Well-foundedness

lemma *wf-on-rmult*: $\llbracket \text{wf}[A](r); \text{wf}[B](s) \rrbracket \implies \text{wf}[A*B](\text{rmult}(A,r,B,s))$

<proof>

lemma *wf-rmult*: $\llbracket \text{wf}(r); \text{wf}(s) \rrbracket \implies \text{wf}(\text{rmult}(\text{field}(r), r, \text{field}(s), s))$

<proof>

lemma *well-ord-rmult*:

$$\llbracket \text{well-ord}(A,r); \text{well-ord}(B,s) \rrbracket \implies \text{well-ord}(A*B, \text{rmult}(A,r,B,s))$$

<proof>

20.2.5 An ord-iso congruence law

lemma *prod-bij*:

$$\begin{aligned} & \llbracket f \in \text{bij}(A,C); g \in \text{bij}(B,D) \rrbracket \\ & \implies (\text{lam } \langle x,y \rangle : A*B. \langle f'x, g'y \rangle) \in \text{bij}(A*B, C*D) \end{aligned}$$

<proof>

lemma *prod-ord-iso-cong*:

$$\begin{aligned} & \llbracket f \in \text{ord-iso}(A,r,A',r'); g \in \text{ord-iso}(B,s,B',s') \rrbracket \\ & \implies (\text{lam } \langle x,y \rangle : A*B. \langle f'x, g'y \rangle) \\ & \quad \in \text{ord-iso}(A*B, \text{rmult}(A,r,B,s), A'*B', \text{rmult}(A',r',B',s')) \end{aligned}$$

<proof>

lemma *singleton-prod-bij*: $(\lambda z \in A. \langle x, z \rangle) \in \text{bij}(A, \{x\} * A)$

<proof>

lemma *singleton-prod-ord-iso*:

$$\begin{aligned} & \text{well-ord}(\{x\}, xr) \implies \\ & (\lambda z \in A. \langle x, z \rangle) \in \text{ord-iso}(A, r, \{x\} * A, \text{rmult}(\{x\}, xr, A, r)) \end{aligned}$$

<proof>

lemma *prod-sum-singleton-bij*:

$$\begin{aligned} & a \notin C \implies \\ & (\lambda x \in C*B + D. \text{case}(\lambda x. x, \lambda y. \langle a, y \rangle, x)) \\ & \in \text{bij}(C*B + D, C*B \cup \{a\} * D) \end{aligned}$$

<proof>

lemma *prod-sum-singleton-ord-iso*:

$$\llbracket a \in A; \text{well-ord}(A,r) \rrbracket \implies$$

$(\lambda x \in \text{pred}(A, a, r) * B + \text{pred}(B, b, s). \text{case}(\lambda x. x, \lambda y. \langle a, y \rangle, x))$
 $\in \text{ord-iso}(\text{pred}(A, a, r) * B + \text{pred}(B, b, s),$
 $\quad \text{radd}(A * B, \text{rmult}(A, r, B, s), B, s),$
 $\quad \text{pred}(A, a, r) * B \cup \{a\} * \text{pred}(B, b, s), \text{rmult}(A, r, B, s))$
 $\langle \text{proof} \rangle$

20.2.6 Distributive law

lemma *sum-prod-distrib-bij*:
 $(\text{lam } \langle x, z \rangle : (A + B) * C. \text{case}(\lambda y. \text{Inl}(\langle y, z \rangle), \lambda y. \text{Inr}(\langle y, z \rangle), x))$
 $\in \text{bij}((A + B) * C, (A * C) + (B * C))$
 $\langle \text{proof} \rangle$

lemma *sum-prod-distrib-ord-iso*:
 $(\text{lam } \langle x, z \rangle : (A + B) * C. \text{case}(\lambda y. \text{Inl}(\langle y, z \rangle), \lambda y. \text{Inr}(\langle y, z \rangle), x))$
 $\in \text{ord-iso}((A + B) * C, \text{rmult}(A + B, \text{radd}(A, r, B, s), C, t),$
 $\quad (A * C) + (B * C), \text{radd}(A * C, \text{rmult}(A, r, C, t), B * C, \text{rmult}(B, s, C, t)))$
 $\langle \text{proof} \rangle$

20.2.7 Associativity

lemma *prod-assoc-bij*:
 $(\text{lam } \langle \langle x, y \rangle, z \rangle : (A * B) * C. \langle x, \langle y, z \rangle \rangle) \in \text{bij}((A * B) * C, A * (B * C))$
 $\langle \text{proof} \rangle$

lemma *prod-assoc-ord-iso*:
 $(\text{lam } \langle \langle x, y \rangle, z \rangle : (A * B) * C. \langle x, \langle y, z \rangle \rangle)$
 $\in \text{ord-iso}((A * B) * C, \text{rmult}(A * B, \text{rmult}(A, r, B, s), C, t),$
 $\quad A * (B * C), \text{rmult}(A, r, B * C, \text{rmult}(B, s, C, t)))$
 $\langle \text{proof} \rangle$

20.3 Inverse Image of a Relation

20.3.1 Rewrite rule

lemma *rvimage-iff*: $\langle a, b \rangle \in \text{rvimage}(A, f, r) \iff \langle f'a, f'b \rangle : r \wedge a \in A \wedge b \in A$
 $\langle \text{proof} \rangle$

20.3.2 Type checking

lemma *rvimage-type*: $\text{rvimage}(A, f, r) \subseteq A * A$
 $\langle \text{proof} \rangle$

lemmas *field-rvimage* = *rvimage-type* [THEN *field-rel-subset*]

lemma *rvimage-converse*: $\text{rvimage}(A, f, \text{converse}(r)) = \text{converse}(\text{rvimage}(A, f, r))$
 $\langle \text{proof} \rangle$

20.3.3 Partial Ordering Properties

lemma *irrefl-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{irrefl}(B,r) \rrbracket \implies \text{irrefl}(A, \text{rimage}(A,f,r))$
 $\langle \text{proof} \rangle$

lemma *trans-on-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{trans}[B](r) \rrbracket \implies \text{trans}[A](\text{rimage}(A,f,r))$
 $\langle \text{proof} \rangle$

lemma *part-ord-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{part-ord}(B,r) \rrbracket \implies \text{part-ord}(A, \text{rimage}(A,f,r))$
 $\langle \text{proof} \rangle$

20.3.4 Linearity

lemma *linear-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{linear}(B,r) \rrbracket \implies \text{linear}(A, \text{rimage}(A,f,r))$
 $\langle \text{proof} \rangle$

lemma *tot-ord-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{tot-ord}(B,r) \rrbracket \implies \text{tot-ord}(A, \text{rimage}(A,f,r))$
 $\langle \text{proof} \rangle$

20.3.5 Well-foundedness

lemma *wf-rvimage* [intro!]: $\text{wf}(r) \implies \text{wf}(\text{rimage}(A,f,r))$

$\langle \text{proof} \rangle$

But note that the combination of *wf-imp-wf-on* and *wf-rvimage* gives $\text{wf}(r) \implies \text{wf}[C](\text{rimage}(A, f, r))$

lemma *wf-on-rvimage*: $\llbracket f \in A \rightarrow B; \text{wf}[B](r) \rrbracket \implies \text{wf}[A](\text{rimage}(A,f,r))$

$\langle \text{proof} \rangle$

lemma *well-ord-rvimage*:

$\llbracket f \in \text{inj}(A,B); \text{well-ord}(B,r) \rrbracket \implies \text{well-ord}(A, \text{rimage}(A,f,r))$

$\langle \text{proof} \rangle$

lemma *ord-iso-rvimage*:

$f \in \text{bij}(A,B) \implies f \in \text{ord-iso}(A, \text{rimage}(A,f,s), B, s)$

$\langle \text{proof} \rangle$

lemma *ord-iso-rvimage-eq*:

$f \in \text{ord-iso}(A,r, B,s) \implies \text{rimage}(A,f,s) = r \cap A * A$

$\langle \text{proof} \rangle$

20.4 Every well-founded relation is a subset of some inverse image of an ordinal

lemma *wf-rvimage-Ord*: $\text{Ord}(i) \implies \text{wf}(\text{rimage}(A, f, \text{Memrel}(i)))$

$\langle \text{proof} \rangle$

definition

$wfrank :: [i,i] \Rightarrow i$ **where**
 $wfrank(r,a) \equiv wfrec(r, a, \lambda x f. \bigcup y \in r - \{x\}. succ(f'y))$

definition

$wftype :: i \Rightarrow i$ **where**
 $wftype(r) \equiv \bigcup y \in range(r). succ(wfrank(r,y))$

lemma $wfrank$: $wf(r) \Longrightarrow wfrank(r,a) = (\bigcup y \in r - \{a\}. succ(wfrank(r,y)))$
 $\langle proof \rangle$

lemma Ord - $wfrank$: $wf(r) \Longrightarrow Ord(wfrank(r,a))$
 $\langle proof \rangle$

lemma $wfrank$ - lt : $\llbracket wf(r); \langle a,b \rangle \in r \rrbracket \Longrightarrow wfrank(r,a) < wfrank(r,b)$
 $\langle proof \rangle$

lemma Ord - $wftype$: $wf(r) \Longrightarrow Ord(wftype(r))$
 $\langle proof \rangle$

lemma $wftypeI$: $\llbracket wf(r); x \in field(r) \rrbracket \Longrightarrow wfrank(r,x) \in wftype(r)$
 $\langle proof \rangle$

lemma wf - imp - $subset$ - $rvimage$:

$\llbracket wf(r); r \subseteq A * A \rrbracket \Longrightarrow \exists i f. Ord(i) \wedge r \subseteq rvimage(A, f, Memrel(i))$
 $\langle proof \rangle$

theorem wf - iff - $subset$ - $rvimage$:

$relation(r) \Longrightarrow wf(r) \longleftrightarrow (\exists i f A. Ord(i) \wedge r \subseteq rvimage(A, f, Memrel(i)))$
 $\langle proof \rangle$

20.5 Other Results

lemma wf - $times$: $A \cap B = 0 \Longrightarrow wf(A * B)$
 $\langle proof \rangle$

Could also be used to prove wf - $radd$

lemma wf - Un :

$\llbracket range(r) \cap domain(s) = 0; wf(r); wf(s) \rrbracket \Longrightarrow wf(r \cup s)$
 $\langle proof \rangle$

20.5.1 The Empty Relation

lemma $wf0$: $wf(0)$
 $\langle proof \rangle$

lemma *linear0*: *linear*(0,0)
 ⟨*proof*⟩

lemma *well-ord0*: *well-ord*(0,0)
 ⟨*proof*⟩

20.5.2 The "measure" relation is useful with wfrec

lemma *measure-eq-rvimage-Memrel*:
 $measure(A,f) = rvimage(A,Lambda(A,f),Memrel(Collect(RepFun(A,f),Ord)))$
 ⟨*proof*⟩

lemma *wf-measure* [*iff*]: *wf*(*measure*(*A*,*f*))
 ⟨*proof*⟩

lemma *measure-iff* [*iff*]: $\langle x,y \rangle \in measure(A,f) \longleftrightarrow x \in A \wedge y \in A \wedge f(x) < f(y)$
 ⟨*proof*⟩

lemma *linear-measure*:
assumes *Ord**f*: $\bigwedge x. x \in A \implies Ord(f(x))$
and *inj*: $\bigwedge x y. \llbracket x \in A; y \in A; f(x) = f(y) \rrbracket \implies x=y$
shows *linear*(*A*, *measure*(*A*,*f*))
 ⟨*proof*⟩

lemma *wf-on-measure*: *wf*[*B*](*measure*(*A*,*f*))
 ⟨*proof*⟩

lemma *well-ord-measure*:
assumes *Ord**f*: $\bigwedge x. x \in A \implies Ord(f(x))$
and *inj*: $\bigwedge x y. \llbracket x \in A; y \in A; f(x) = f(y) \rrbracket \implies x=y$
shows *well-ord*(*A*, *measure*(*A*,*f*))
 ⟨*proof*⟩

lemma *measure-type*: *measure*(*A*,*f*) $\subseteq A * A$
 ⟨*proof*⟩

20.5.3 Well-foundedness of Unions

lemma *wf-on-Union*:
assumes *wfA*: *wf*[*A*](*r*)
and *wfB*: $\bigwedge a. a \in A \implies wf[B(a)](s)$
and *ok*: $\bigwedge a u v. \llbracket \langle u,v \rangle \in s; v \in B(a); a \in A \rrbracket$
 $\implies (\exists a' \in A. \langle a',a \rangle \in r \wedge u \in B(a')) \mid u \in B(a)$
shows *wf*[$\bigcup a \in A. B(a)$](*s*)
 ⟨*proof*⟩

20.5.4 Bijections involving Powersets

lemma *Pow-sum-bij*:
 $(\lambda Z \in Pow(A+B). \langle \{x \in A. Inl(x) \in Z\}, \{y \in B. Inr(y) \in Z\} \rangle)$

$\in \text{bij}(\text{Pow}(A+B), \text{Pow}(A)*\text{Pow}(B))$
 $\langle \text{proof} \rangle$

As a special case, we have $\text{bij}(\text{Pow}(A \times B), A \rightarrow \text{Pow}(B))$

lemma *Pow-Sigma-bij*:

$(\lambda r \in \text{Pow}(\text{Sigma}(A,B)). \lambda x \in A. r \text{ “ } \{x\})$
 $\in \text{bij}(\text{Pow}(\text{Sigma}(A,B)), \prod x \in A. \text{Pow}(B(x)))$
 $\langle \text{proof} \rangle$

end

21 Order Types and Ordinal Arithmetic

theory *OrderType* **imports** *OrderArith OrdQuant Nat* **begin**

The order type of a well-ordering is the least ordinal isomorphic to it. Ordinal arithmetic is traditionally defined in terms of order types, as it is here. But a definition by transfinite recursion would be much simpler!

definition

ordermap $:: [i,i] \Rightarrow i$ **where**
ordermap(*A*,*r*) $\equiv \lambda x \in A. \text{wfrec}[A](r, x, \lambda x f. f \text{ “ } \text{pred}(A,x,r))$

definition

ordertype $:: [i,i] \Rightarrow i$ **where**
ordertype(*A*,*r*) $\equiv \text{ordermap}(A,r) \text{ “ } A$

definition

Ord-alt $:: i \Rightarrow o$ **where**
Ord-alt(*X*) $\equiv \text{well-ord}(X, \text{Memrel}(X)) \wedge (\forall u \in X. u = \text{pred}(X, u, \text{Memrel}(X)))$

definition

ordify $:: i \Rightarrow i$ **where**
ordify(*x*) $\equiv \text{if } \text{Ord}(x) \text{ then } x \text{ else } 0$

definition

omult $:: [i,i] \Rightarrow i$ (**infixl** $\langle ** \rangle$ 70) **where**
 $i ** j \equiv \text{ordertype}(j*i, \text{rmult}(j, \text{Memrel}(j), i, \text{Memrel}(i)))$

definition

raw-odd $:: [i,i] \Rightarrow i$ **where**
raw-odd(*i*,*j*) $\equiv \text{ordertype}(i+j, \text{radd}(i, \text{Memrel}(i), j, \text{Memrel}(j)))$

definition

odd $:: [i,i] \Rightarrow i$ (**infixl** $\langle ++ \rangle$ 65) **where**

$$i ++ j \equiv \text{raw-odd}(\text{ordify}(i), \text{ordify}(j))$$

definition

$$\begin{array}{l} \text{odiff} \quad :: [i, i] \Rightarrow i \quad (\text{infixl } \langle -- \rangle 65) \quad \text{where} \\ i -- j \equiv \text{ordertype}(i - j, \text{Memrel}(i)) \end{array}$$

21.1 Proofs needing the combination of Ordinal.thy and Order.thy

lemma *le-well-ord-Memrel*: $j \leq i \implies \text{well-ord}(j, \text{Memrel}(i))$
 $\langle \text{proof} \rangle$

lemmas *well-ord-Memrel* = *le-reft* [THEN *le-well-ord-Memrel*]

lemma *lt-pred-Memrel*:
 $j < i \implies \text{pred}(i, j, \text{Memrel}(i)) = j$
 $\langle \text{proof} \rangle$

lemma *pred-Memrel*:
 $x \in A \implies \text{pred}(A, x, \text{Memrel}(A)) = A \cap x$
 $\langle \text{proof} \rangle$

lemma *Ord-iso-implies-eq-lemma*:
 $\llbracket j < i; f \in \text{ord-iso}(i, \text{Memrel}(i), j, \text{Memrel}(j)) \rrbracket \implies R$
 $\langle \text{proof} \rangle$

lemma *Ord-iso-implies-eq*:
 $\llbracket \text{Ord}(i); \text{Ord}(j); f \in \text{ord-iso}(i, \text{Memrel}(i), j, \text{Memrel}(j)) \rrbracket$
 $\implies i = j$
 $\langle \text{proof} \rangle$

21.2 Ordermap and ordertype

lemma *ordermap-type*:
 $\text{ordermap}(A, r) \in A -> \text{ordertype}(A, r)$
 $\langle \text{proof} \rangle$

21.2.1 Unfolding of ordermap

lemma *ordermap-eq-image*:
 $\llbracket \text{wf}[A](r); x \in A \rrbracket$
 $\implies \text{ordermap}(A, r) \text{ `` } x = \text{ordermap}(A, r) \text{ `` } \text{pred}(A, x, r)$
 $\langle \text{proof} \rangle$

lemma *ordermap-pred-unfold*:

$$\begin{aligned} & \llbracket wf[A](r); x \in A \rrbracket \\ & \implies ordermap(A,r) \text{ ' } x = \{ordermap(A,r) \text{ ' } y \mid y \in pred(A,x,r)\} \\ & \langle proof \rangle \end{aligned}$$

lemmas *ordermap-unfold* = *ordermap-pred-unfold* [*simplified pred-def*]

21.2.2 Showing that ordermap, ordertype yield ordinals

lemma *Ord-ordermap*:

$$\llbracket well-ord(A,r); x \in A \rrbracket \implies Ord(ordermap(A,r) \text{ ' } x)$$
 $\langle proof \rangle$

lemma *Ord-ordertype*:

$$well-ord(A,r) \implies Ord(ordertype(A,r))$$
 $\langle proof \rangle$

21.2.3 ordermap preserves the orderings in both directions

lemma *ordermap-mono*:

$$\begin{aligned} & \llbracket \langle w,x \rangle: r; wf[A](r); w \in A; x \in A \rrbracket \\ & \implies ordermap(A,r) \text{ ' } w \in ordermap(A,r) \text{ ' } x \\ & \langle proof \rangle \end{aligned}$$

lemma *converse-ordermap-mono*:

$$\begin{aligned} & \llbracket ordermap(A,r) \text{ ' } w \in ordermap(A,r) \text{ ' } x; well-ord(A,r); w \in A; x \in A \rrbracket \\ & \implies \langle w,x \rangle: r \\ & \langle proof \rangle \end{aligned}$$

lemma *ordermap-surj*: $ordermap(A, r) \in surj(A, ordertype(A, r))$

$\langle proof \rangle$

lemma *ordermap-bij*:

$$well-ord(A,r) \implies ordermap(A,r) \in bij(A, ordertype(A,r))$$
 $\langle proof \rangle$

21.2.4 Isomorphisms involving ordertype

lemma *ordertype-ord-iso*:

$$\begin{aligned} & well-ord(A,r) \\ & \implies ordermap(A,r) \in ord-iso(A,r, ordertype(A,r), Memrel(ordertype(A,r))) \\ & \langle proof \rangle \end{aligned}$$

lemma *ordertype-eq*:

$$\begin{aligned} & \llbracket f \in ord-iso(A,r,B,s); well-ord(B,s) \rrbracket \\ & \implies ordertype(A,r) = ordertype(B,s) \\ & \langle proof \rangle \end{aligned}$$

lemma *ordertype-eq-imp-ord-iso*:

$$\begin{aligned} & \llbracket \text{ordertype}(A,r) = \text{ordertype}(B,s); \text{well-ord}(A,r); \text{well-ord}(B,s) \rrbracket \\ & \implies \exists f. f \in \text{ord-iso}(A,r,B,s) \\ & \langle \text{proof} \rangle \end{aligned}$$

21.2.5 Basic equalities for ordertype

lemma *le-ordertype-Memrel*: $j \leq i \implies \text{ordertype}(j, \text{Memrel}(i)) = j$
 $\langle \text{proof} \rangle$

lemmas *ordertype-Memrel = le-refl* [THEN *le-ordertype-Memrel*]

lemma *ordertype-0* [simp]: $\text{ordertype}(0,r) = 0$
 $\langle \text{proof} \rangle$

lemmas *bij-ordertype-vimage = ord-iso-rvimage* [THEN *ordertype-eq*]

21.2.6 A fundamental unfolding law for ordertype.

lemma *ordermap-pred-eq-ordermap*:

$$\begin{aligned} & \llbracket \text{well-ord}(A,r); y \in A; z \in \text{pred}(A,y,r) \rrbracket \\ & \implies \text{ordermap}(\text{pred}(A,y,r), r) \text{ ` } z = \text{ordermap}(A, r) \text{ ` } z \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *ordertype-unfold*:

$$\text{ordertype}(A,r) = \{ \text{ordermap}(A,r) \text{ ` } y \mid y \in A \}$$
 $\langle \text{proof} \rangle$

Theorems by Krzysztof Grabczewski; proofs simplified by lcp

lemma *ordertype-pred-subset*: $\llbracket \text{well-ord}(A,r); x \in A \rrbracket \implies$
 $\text{ordertype}(\text{pred}(A,x,r), r) \subseteq \text{ordertype}(A,r)$
 $\langle \text{proof} \rangle$

lemma *ordertype-pred-lt*:

$$\begin{aligned} & \llbracket \text{well-ord}(A,r); x \in A \rrbracket \\ & \implies \text{ordertype}(\text{pred}(A,x,r), r) < \text{ordertype}(A,r) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *ordertype-pred-unfold*:

$$\begin{aligned} & \text{well-ord}(A,r) \\ & \implies \text{ordertype}(A,r) = \{ \text{ordertype}(\text{pred}(A,x,r), r) \mid x \in A \} \\ & \langle \text{proof} \rangle \end{aligned}$$

21.3 Alternative definition of ordinal

lemma *Ord-is-Ord-alt*: $\text{Ord}(i) \implies \text{Ord-alt}(i)$
 $\langle \text{proof} \rangle$

lemma *Ord-alt-is-Ord*:
 $Ord\text{-}alt(i) \implies Ord(i)$
 $\langle proof \rangle$

21.4 Ordinal Addition

21.4.1 Order Type calculations for radd

Addition with 0

lemma *bij-sum-0*: $(\lambda z \in A+0. case(\lambda x. x, \lambda y. y, z)) \in bij(A+0, A)$
 $\langle proof \rangle$

lemma *ordertype-sum-0-eq*:
 $well\text{-}ord(A, r) \implies ordertype(A+0, radd(A, r, 0, s)) = ordertype(A, r)$
 $\langle proof \rangle$

lemma *bij-0-sum*: $(\lambda z \in 0+A. case(\lambda x. x, \lambda y. y, z)) \in bij(0+A, A)$
 $\langle proof \rangle$

lemma *ordertype-0-sum-eq*:
 $well\text{-}ord(A, r) \implies ordertype(0+A, radd(0, s, A, r)) = ordertype(A, r)$
 $\langle proof \rangle$

Initial segments of radd. Statements by Grabczewski

lemma *pred-Inl-bij*:
 $a \in A \implies (\lambda x \in pred(A, a, r). Inl(x))$
 $\in bij(pred(A, a, r), pred(A+B, Inl(a), radd(A, r, B, s)))$
 $\langle proof \rangle$

lemma *ordertype-pred-Inl-eq*:
 $\llbracket a \in A; well\text{-}ord(A, r) \rrbracket$
 $\implies ordertype(pred(A+B, Inl(a), radd(A, r, B, s)), radd(A, r, B, s)) =$
 $ordertype(pred(A, a, r), r)$
 $\langle proof \rangle$

lemma *pred-Inr-bij*:
 $b \in B \implies$
 $id(A+pred(B, b, s))$
 $\in bij(A+pred(B, b, s), pred(A+B, Inr(b), radd(A, r, B, s)))$
 $\langle proof \rangle$

lemma *ordertype-pred-Inr-eq*:
 $\llbracket b \in B; well\text{-}ord(A, r); well\text{-}ord(B, s) \rrbracket$
 $\implies ordertype(pred(A+B, Inr(b), radd(A, r, B, s)), radd(A, r, B, s)) =$
 $ordertype(A+pred(B, b, s), radd(A, r, pred(B, b, s), s))$
 $\langle proof \rangle$

21.4.2 ordify: trivial coercion to an ordinal

lemma *Ord-ordify* [*iff*, *TC*]: $\text{Ord}(\text{ordify}(x))$
<proof>

lemma *ordify-idem* [*simp*]: $\text{ordify}(\text{ordify}(x)) = \text{ordify}(x)$
<proof>

21.4.3 Basic laws for ordinal addition

lemma *Ord-raw-oadd*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{Ord}(\text{raw-oadd}(i,j))$
<proof>

lemma *Ord-oadd* [*iff*, *TC*]: $\text{Ord}(i++j)$
<proof>

Ordinal addition with zero

lemma *raw-oadd-0*: $\text{Ord}(i) \implies \text{raw-oadd}(i,0) = i$
<proof>

lemma *oadd-0* [*simp*]: $\text{Ord}(i) \implies i++0 = i$
<proof>

lemma *raw-oadd-0-left*: $\text{Ord}(i) \implies \text{raw-oadd}(0,i) = i$
<proof>

lemma *oadd-0-left* [*simp*]: $\text{Ord}(i) \implies 0++i = i$
<proof>

lemma *oadd-eq-if-raw-oadd*:
$$i++j = (\text{if } \text{Ord}(i) \text{ then } (\text{if } \text{Ord}(j) \text{ then } \text{raw-oadd}(i,j) \text{ else } i) \\ \text{else } (\text{if } \text{Ord}(j) \text{ then } j \text{ else } 0))$$

<proof>

lemma *raw-oadd-eq-oadd*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{raw-oadd}(i,j) = i++j$
<proof>

lemma *lt-oadd1*: $k < i \implies k < i++j$
<proof>

lemma *oadd-le-self*: $\text{Ord}(i) \implies i \leq i++j$
<proof>

Various other results

lemma *id-ord-iso-Memrel*: $A \leq B \implies id(A) \in ord\text{-}iso(A, Memrel(A), A, Memrel(B))$
 $\langle proof \rangle$

lemma *subset-ord-iso-Memrel*:
 $\llbracket f \in ord\text{-}iso(A, Memrel(B), C, r); A \leq B \rrbracket \implies f \in ord\text{-}iso(A, Memrel(A), C, r)$
 $\langle proof \rangle$

lemma *restrict-ord-iso*:
 $\llbracket f \in ord\text{-}iso(i, Memrel(i), Order.pred(A, a, r), r); a \in A; j < i; trans[A](r) \rrbracket$
 $\implies restrict(f, j) \in ord\text{-}iso(j, Memrel(j), Order.pred(A, f'j, r), r)$
 $\langle proof \rangle$

lemma *restrict-ord-iso2*:
 $\llbracket f \in ord\text{-}iso(Order.pred(A, a, r), r, i, Memrel(i)); a \in A; j < i; trans[A](r) \rrbracket$
 $\implies converse(restrict(converse(f), j))$
 $\in ord\text{-}iso(Order.pred(A, converse(f)'j, r), r, j, Memrel(j))$
 $\langle proof \rangle$

lemma *ordertype-sum-Memrel*:
 $\llbracket well\text{-}ord(A, r); k < j \rrbracket$
 $\implies ordertype(A+k, radd(A, r, k, Memrel(j))) = ordertype(A+k, radd(A, r, k, Memrel(k)))$
 $\langle proof \rangle$

lemma *oadd-lt-mono2*: $k < j \implies i++k < i++j$
 $\langle proof \rangle$

lemma *oadd-lt-cancel2*: $\llbracket i++j < i++k; Ord(j) \rrbracket \implies j < k$
 $\langle proof \rangle$

lemma *oadd-lt-iff2*: $Ord(j) \implies i++j < i++k \longleftrightarrow j < k$
 $\langle proof \rangle$

lemma *oadd-inject*: $\llbracket i++j = i++k; Ord(j); Ord(k) \rrbracket \implies j = k$
 $\langle proof \rangle$

lemma *lt-oadd-disj*: $k < i++j \implies k < i \mid (\exists l \in j. k = i++l)$
 $\langle proof \rangle$

21.4.4 Ordinal addition with successor – via associativity!

lemma *oadd-assoc*: $(i++j)++k = i++(j++k)$
 $\langle proof \rangle$

lemma *oadd-unfold*: $\llbracket Ord(i); Ord(j) \rrbracket \implies i++j = i \cup (\bigcup_{k \in j} \{i++k\})$
 $\langle proof \rangle$

lemma *oadd-1*: $Ord(i) \implies i++1 = succ(i)$

<proof>

lemma *oadd-succ [simp]*: $Ord(j) \implies i++succ(j) = succ(i++j)$

<proof>

Ordinal addition with limit ordinals

lemma *oadd-UN*:

$$\begin{aligned} & \llbracket \bigwedge x. x \in A \implies Ord(j(x)); \ a \in A \rrbracket \\ & \implies i++(\bigcup_{x \in A} j(x)) = (\bigcup_{x \in A} i++j(x)) \end{aligned}$$

<proof>

lemma *oadd-Limit*: $Limit(j) \implies i++j = (\bigcup_{k \in j} i++k)$

<proof>

lemma *oadd-eq-0-iff*: $\llbracket Ord(i); Ord(j) \rrbracket \implies (i++j) = 0 \longleftrightarrow i=0 \wedge j=0$

<proof>

lemma *oadd-eq-lt-iff*: $\llbracket Ord(i); Ord(j) \rrbracket \implies 0 < (i++j) \longleftrightarrow 0 < i \mid 0 < j$

<proof>

lemma *oadd-LimitI*: $\llbracket Ord(i); Limit(j) \rrbracket \implies Limit(i++j)$

<proof>

Order/monotonicity properties of ordinal addition

lemma *oadd-le-self2*: $Ord(i) \implies i \leq j++i$

<proof>

lemma *oadd-le-mono1*: $k \leq j \implies k++i \leq j++i$

<proof>

lemma *oadd-lt-mono*: $\llbracket i' \leq i; \ j' < j \rrbracket \implies i'++j' < i++j$

<proof>

lemma *oadd-le-mono*: $\llbracket i' \leq i; \ j' \leq j \rrbracket \implies i'++j' \leq i++j$

<proof>

lemma *oadd-le-iff2*: $\llbracket Ord(j); Ord(k) \rrbracket \implies i++j \leq i++k \longleftrightarrow j \leq k$

<proof>

lemma *oadd-lt-self*: $\llbracket Ord(i); \ 0 < j \rrbracket \implies i < i++j$

<proof>

Every ordinal is exceeded by some limit ordinal.

lemma *Ord-imp-greater-Limit*: $Ord(i) \implies \exists k. i < k \wedge Limit(k)$

<proof>

lemma *Ord2-imp-greater-Limit*: $\llbracket Ord(i); Ord(j) \rrbracket \implies \exists k. i < k \wedge j < k \wedge Limit(k)$

$\langle proof \rangle$

21.5 Ordinal Subtraction

The difference is $ordertype(j - i, Memrel(j))$. It's probably simpler to define the difference recursively!

lemma *bij-sum-Diff*:

$$A \leq B \implies (\lambda y \in B. \text{if}(y \in A, \text{Inl}(y), \text{Inr}(y))) \in \text{bij}(B, A + (B - A))$$

$\langle proof \rangle$

lemma *ordertype-sum-Diff*:

$$\begin{aligned} i \leq j \implies \\ & \text{ordertype}(i + (j - i), \text{radd}(i, \text{Memrel}(j), j - i, \text{Memrel}(j))) = \\ & \text{ordertype}(j, \text{Memrel}(j)) \end{aligned}$$

$\langle proof \rangle$

lemma *Ord-odiff* [*simp*, *TC*]:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{Ord}(i - j)$$

$\langle proof \rangle$

lemma *raw-oadd-ordertype-Diff*:

$$\begin{aligned} i \leq j \\ \implies \text{raw-oadd}(i, j - i) = \text{ordertype}(i + (j - i), \text{radd}(i, \text{Memrel}(j), j - i, \text{Memrel}(j))) \end{aligned}$$

$\langle proof \rangle$

lemma *oadd-odiff-inverse*: $i \leq j \implies i ++ (j - i) = j$

$\langle proof \rangle$

lemma *odiff-oadd-inverse*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies (i ++ j) - i = j$

$\langle proof \rangle$

lemma *odiff-lt-mono2*: $\llbracket i < j; k \leq i \rrbracket \implies i - k < j - k$

$\langle proof \rangle$

21.6 Ordinal Multiplication

lemma *Ord-omult* [*simp*, *TC*]:

$$\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies \text{Ord}(i ** j)$$

$\langle proof \rangle$

21.6.1 A useful unfolding law

lemma *pred-Pair-eq*:

$$\llbracket a \in A; b \in B \rrbracket \implies \text{pred}(A * B, \langle a, b \rangle, \text{rmult}(A, r, B, s)) = \text{pred}(A, a, r) * B \cup (\{a\} * \text{pred}(B, b, s))$$

$\langle proof \rangle$

lemma *ordertype-pred-Pair-eq*:

$$\begin{aligned} & \llbracket a \in A; \ b \in B; \ \text{well-ord}(A,r); \ \text{well-ord}(B,s) \rrbracket \implies \\ & \quad \text{ordertype}(\text{pred}(A*B, \langle a,b \rangle), \text{rmult}(A,r,B,s)), \text{rmult}(A,r,B,s)) = \\ & \quad \text{ordertype}(\text{pred}(A,a,r)*B + \text{pred}(B,b,s), \\ & \quad \quad \text{radd}(A*B, \text{rmult}(A,r,B,s), B, s)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *ordertype-pred-Pair-lemma*:

$$\begin{aligned} & \llbracket i' < i; \ j' < j \rrbracket \\ & \implies \text{ordertype}(\text{pred}(i*j, \langle i',j' \rangle), \text{rmult}(i, \text{Memrel}(i), j, \text{Memrel}(j))), \\ & \quad \text{rmult}(i, \text{Memrel}(i), j, \text{Memrel}(j))) = \\ & \quad \text{raw-oadd} \ (j**i', j') \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *lt-omult*:

$$\begin{aligned} & \llbracket \text{Ord}(i); \ \text{Ord}(j); \ k < j**i \rrbracket \\ & \implies \exists j' \ i'. \ k = j**i' ++ j' \wedge j' < j \wedge i' < i \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *omult-oadd-lt*:

$$\begin{aligned} & \llbracket j' < j; \ i' < i \rrbracket \implies j**i' ++ j' < j**i \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *omult-unfold*:

$$\begin{aligned} & \llbracket \text{Ord}(i); \ \text{Ord}(j) \rrbracket \implies j**i = (\bigcup_{j' \in j}. \bigcup_{i' \in i}. \{j**i' ++ j'\}) \\ & \langle \text{proof} \rangle \end{aligned}$$

21.6.2 Basic laws for ordinal multiplication

Ordinal multiplication by zero

$$\begin{aligned} & \textbf{lemma} \ \textit{omult-0} \ [\textit{simp}]: \ i**0 = 0 \\ & \langle \text{proof} \rangle \end{aligned}$$

$$\begin{aligned} & \textbf{lemma} \ \textit{omult-0-left} \ [\textit{simp}]: \ 0**i = 0 \\ & \langle \text{proof} \rangle \end{aligned}$$

Ordinal multiplication by 1

$$\begin{aligned} & \textbf{lemma} \ \textit{omult-1} \ [\textit{simp}]: \ \text{Ord}(i) \implies i**1 = i \\ & \langle \text{proof} \rangle \end{aligned}$$

$$\begin{aligned} & \textbf{lemma} \ \textit{omult-1-left} \ [\textit{simp}]: \ \text{Ord}(i) \implies 1**i = i \\ & \langle \text{proof} \rangle \end{aligned}$$

Distributive law for ordinal multiplication and addition

lemma *oadd-omult-distrib*:

$$\begin{aligned} & \llbracket \text{Ord}(i); \ \text{Ord}(j); \ \text{Ord}(k) \rrbracket \implies i**(j++k) = (i**j)++(i**k) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *omult-succ*: $\llbracket \text{Ord}(i); \text{Ord}(j) \rrbracket \implies i**\text{succ}(j) = (i**j)++i$
 $\langle \text{proof} \rangle$

Associative law

lemma *omult-assoc*:
 $\llbracket \text{Ord}(i); \text{Ord}(j); \text{Ord}(k) \rrbracket \implies (i**j)**k = i**(j**k)$
 $\langle \text{proof} \rangle$

Ordinal multiplication with limit ordinals

lemma *omult-UN*:
 $\llbracket \text{Ord}(i); \bigwedge x. x \in A \implies \text{Ord}(j(x)) \rrbracket$
 $\implies i**(\bigcup_{x \in A} j(x)) = (\bigcup_{x \in A} i**j(x))$
 $\langle \text{proof} \rangle$

lemma *omult-Limit*: $\llbracket \text{Ord}(i); \text{Limit}(j) \rrbracket \implies i**j = (\bigcup_{k \in j} i**k)$
 $\langle \text{proof} \rangle$

21.6.3 Ordering/monotonicity properties of ordinal multiplication

lemma *lt-omult1*: $\llbracket k < i; 0 < j \rrbracket \implies k < i**j$
 $\langle \text{proof} \rangle$

lemma *omult-le-self*: $\llbracket \text{Ord}(i); 0 < j \rrbracket \implies i \leq i**j$
 $\langle \text{proof} \rangle$

lemma *omult-le-mono1*:
assumes $kj: k \leq j$ **and** $i: \text{Ord}(i)$ **shows** $k**i \leq j**i$
 $\langle \text{proof} \rangle$

lemma *omult-lt-mono2*: $\llbracket k < j; 0 < i \rrbracket \implies i**k < i**j$
 $\langle \text{proof} \rangle$

lemma *omult-le-mono2*: $\llbracket k \leq j; \text{Ord}(i) \rrbracket \implies i**k \leq i**j$
 $\langle \text{proof} \rangle$

lemma *omult-le-mono*: $\llbracket i' \leq i; j' \leq j \rrbracket \implies i'*j' \leq i**j$
 $\langle \text{proof} \rangle$

lemma *omult-lt-mono*: $\llbracket i' \leq i; j' < j; 0 < i \rrbracket \implies i'*j' < i**j$
 $\langle \text{proof} \rangle$

lemma *omult-le-self2*:
assumes $i: \text{Ord}(i)$ **and** $j: 0 < j$ **shows** $i \leq j**i$
 $\langle \text{proof} \rangle$

Further properties of ordinal multiplication

lemma *omult-inject*: $\llbracket i**j = i**k; 0 < i; \text{Ord}(j); \text{Ord}(k) \rrbracket \implies j=k$
 $\langle \text{proof} \rangle$

21.7 The Relation Lt

lemma *wf-Lt*: $wf(Lt)$

<proof>

lemma *irrefl-Lt*: $irrefl(A, Lt)$

<proof>

lemma *trans-Lt*: $trans[A](Lt)$

<proof>

lemma *part-ord-Lt*: $part-ord(A, Lt)$

<proof>

lemma *linear-Lt*: $linear(nat, Lt)$

<proof>

lemma *tot-ord-Lt*: $tot-ord(nat, Lt)$

<proof>

lemma *well-ord-Lt*: $well-ord(nat, Lt)$

<proof>

end

22 Finite Powerset Operator and Finite Function Space

theory *Finite* **imports** *Inductive Epsilon Nat* **begin**

rep-datatype

elimination *natE*

induction *nat-induct*

case-eqns *nat-case-0 nat-case-succ*

recursor-eqns *recursor-0 recursor-succ*

consts

Fin $:: i \Rightarrow i$

FiniteFun $:: [i, i] \Rightarrow i$ ($\langle \langle notation = \langle infix -||> \rangle - ||> / - \rangle [61, 60] 60 \rangle$)

inductive

domains $Fin(A) \subseteq Pow(A)$

intros

emptyI: $0 \in Fin(A)$

consI: $\llbracket a \in A; b \in Fin(A) \rrbracket \Longrightarrow cons(a, b) \in Fin(A)$

type-intros *empty-subsetI cons-subsetI PowI*

type-elim *PowD [elim-format]*

inductive
domains $FiniteFun(A,B) \subseteq Fin(A*B)$
intros
 $emptyI: 0 \in A -||> B$
 $consI: \llbracket a \in A; b \in B; h \in A -||> B; a \notin domain(h) \rrbracket$
 $\implies cons(\langle a,b \rangle, h) \in A -||> B$
type-intros $Fin.intros$

22.1 Finite Powerset Operator

lemma $Fin-mono: A \leq B \implies Fin(A) \subseteq Fin(B)$
 $\langle proof \rangle$

lemmas $FinD = Fin.dom-subset [THEN subsetD, THEN PowD]$

lemma $Fin-induct [case-names 0 cons, induct set: Fin]:$
 $\llbracket b \in Fin(A);$
 $P(0);$
 $\bigwedge x y. \llbracket x \in A; y \in Fin(A); x \notin y; P(y) \rrbracket \implies P(cons(x,y))$
 $\rrbracket \implies P(b)$
 $\langle proof \rangle$

declare $Fin.intros [simp]$

lemma $Fin-0: Fin(0) = \{0\}$
 $\langle proof \rangle$

lemma $Fin-UnI [simp]: \llbracket b \in Fin(A); c \in Fin(A) \rrbracket \implies b \cup c \in Fin(A)$
 $\langle proof \rangle$

lemma $Fin-UnionI: C \in Fin(Fin(A)) \implies \bigcup(C) \in Fin(A)$
 $\langle proof \rangle$

lemma $Fin-subset-lemma [rule-format]: b \in Fin(A) \implies \forall z. z \leq b \longrightarrow z \in Fin(A)$
 $\langle proof \rangle$

lemma $Fin-subset: \llbracket c \leq b; b \in Fin(A) \rrbracket \implies c \in Fin(A)$
 $\langle proof \rangle$

lemma *Fin-IntI1* [*intro,simp*]: $b \in \text{Fin}(A) \implies b \cap c \in \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemma *Fin-IntI2* [*intro,simp*]: $c \in \text{Fin}(A) \implies b \cap c \in \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemma *Fin-0-induct-lemma* [*rule-format*]:
 $\llbracket c \in \text{Fin}(A); b \in \text{Fin}(A); P(b);$
 $\bigwedge x y. \llbracket x \in A; y \in \text{Fin}(A); x \in y; P(y) \rrbracket \implies P(y - \{x\})$
 $\rrbracket \implies c \leq b \longrightarrow P(b - c)$
 $\langle \text{proof} \rangle$

lemma *Fin-0-induct*:
 $\llbracket b \in \text{Fin}(A);$
 $P(b);$
 $\bigwedge x y. \llbracket x \in A; y \in \text{Fin}(A); x \in y; P(y) \rrbracket \implies P(y - \{x\})$
 $\rrbracket \implies P(0)$
 $\langle \text{proof} \rangle$

lemma *nat-fun-subset-Fin*: $n \in \text{nat} \implies n \rightarrow A \subseteq \text{Fin}(\text{nat} * A)$
 $\langle \text{proof} \rangle$

22.2 Finite Function Space

lemma *FiniteFun-mono*:
 $\llbracket A \leq C; B \leq D \rrbracket \implies A - || > B \subseteq C - || > D$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-mono1*: $A \leq B \implies A - || > A \subseteq B - || > B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-is-fun*: $h \in A - || > B \implies h \in \text{domain}(h) \rightarrow B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-domain-Fin*: $h \in A - || > B \implies \text{domain}(h) \in \text{Fin}(A)$
 $\langle \text{proof} \rangle$

lemmas *FiniteFun-apply-type* = *FiniteFun-is-fun* [*THEN apply-type*]

lemma *FiniteFun-subset-lemma* [*rule-format*]:
 $b \in A - || > B \implies \forall z. z \leq b \longrightarrow z \in A - || > B$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-subset*: $\llbracket c \leq b; b \in A - || > B \rrbracket \implies c \in A - || > B$
 $\langle \text{proof} \rangle$

lemma *fun-FiniteFunI* [rule-format]: $A \in \text{Fin}(X) \implies \forall f. f \in A \multimap B \longrightarrow f \in A -||> B$
 $\langle \text{proof} \rangle$

lemma *lam-FiniteFun*: $A \in \text{Fin}(X) \implies (\lambda x \in A. b(x)) \in A -||> \{b(x). x \in A\}$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-Collect-iff*:
 $f \in \text{FiniteFun}(A, \{y \in B. P(y)\})$
 $\longleftrightarrow f \in \text{FiniteFun}(A, B) \wedge (\forall x \in \text{domain}(f). P(f'x))$
 $\langle \text{proof} \rangle$

22.3 The Contents of a Singleton Set

definition
 $\text{contents} :: i \Rightarrow i$ **where**
 $\text{contents}(X) \equiv \text{THE } x. X = \{x\}$

lemma *contents-eq* [simp]: $\text{contents}(\{x\}) = x$
 $\langle \text{proof} \rangle$

end

23 Cardinal Numbers Without the Axiom of Choice

theory *Cardinal* **imports** *OrderType Finite Nat Sum* **begin**

definition

$\text{Least} :: (i \Rightarrow o) \Rightarrow i$ (**binder** $\langle \mu \rangle$ 10) **where**
 $\text{Least}(P) \equiv \text{THE } i. \text{Ord}(i) \wedge P(i) \wedge (\forall j. j < i \longrightarrow \neg P(j))$

definition

$\text{eqpoll} :: [i, i] \Rightarrow o$ (**infixl** $\langle \approx \rangle$ 50) **where**
 $A \approx B \equiv \exists f. f \in \text{bij}(A, B)$

definition

$\text{lepoll} :: [i, i] \Rightarrow o$ (**infixl** $\langle \lesssim \rangle$ 50) **where**
 $A \lesssim B \equiv \exists f. f \in \text{inj}(A, B)$

definition

$\text{lesspoll} :: [i, i] \Rightarrow o$ (**infixl** $\langle \prec \rangle$ 50) **where**
 $A \prec B \equiv A \lesssim B \wedge \neg(A \approx B)$

definition

$\text{cardinal} :: i \Rightarrow i$ ($\langle \langle \text{open-block notation} = \langle \text{mixfix cardinal} \rangle \rangle | - \rangle$)
where $|A| \equiv (\mu i. i \approx A)$

definition

$Finite :: i \Rightarrow o$ **where**
 $Finite(A) \equiv \exists n \in nat. A \approx n$

definition

$Card :: i \Rightarrow o$ **where**
 $Card(i) \equiv (i = |i|)$

23.1 The Schroeder-Bernstein Theorem

See Davey and Priestly, page 106

lemma *decomp-bnd-mono*: $bnd\text{-}mono(X, \lambda W. X - g^{''}(Y - f^{''}W))$
 $\langle proof \rangle$

lemma *Banach-last-equation*:

$g \in Y \multimap X$
 $\implies g^{''}(Y - f^{''}lfp(X, \lambda W. X - g^{''}(Y - f^{''}W))) =$
 $X - lfp(X, \lambda W. X - g^{''}(Y - f^{''}W))$
 $\langle proof \rangle$

lemma *decomposition*:

$\llbracket f \in X \multimap Y; g \in Y \multimap X \rrbracket \implies$
 $\exists XA XB YA YB. (XA \cap XB = 0) \wedge (XA \cup XB = X) \wedge$
 $(YA \cap YB = 0) \wedge (YA \cup YB = Y) \wedge$
 $f^{''}XA = YA \wedge g^{''}YB = XB$
 $\langle proof \rangle$

lemma *schroeder-bernstein*:

$\llbracket f \in inj(X, Y); g \in inj(Y, X) \rrbracket \implies \exists h. h \in bij(X, Y)$
 $\langle proof \rangle$

lemma *bij-imp-epoll*: $f \in bij(A, B) \implies A \approx B$
 $\langle proof \rangle$

lemmas *epoll-refl* = *id-bij* [THEN *bij-imp-epoll*, *simp*]

lemma *epoll-sym*: $X \approx Y \implies Y \approx X$
 $\langle proof \rangle$

lemma *epoll-trans* [*trans*]:

$\llbracket X \approx Y; Y \approx Z \rrbracket \implies X \approx Z$
 $\langle proof \rangle$

lemma *subset-imp-lepoll*: $X \leq Y \implies X \lesssim Y$
 $\langle proof \rangle$

lemmas *lepoll-refl* = *subset-refl* [THEN *subset-imp-lepoll*, *simp*]

lemmas *le-imp-lepoll* = *le-imp-subset* [THEN *subset-imp-lepoll*]

lemma *eqpoll-imp-lepoll*: $X \approx Y \implies X \lesssim Y$
 $\langle proof \rangle$

lemma *lepoll-trans* [trans]: $\llbracket X \lesssim Y; Y \lesssim Z \rrbracket \implies X \lesssim Z$
 $\langle proof \rangle$

lemma *eq-lepoll-trans* [trans]: $\llbracket X \approx Y; Y \lesssim Z \rrbracket \implies X \lesssim Z$
 $\langle proof \rangle$

lemma *lepoll-eq-trans* [trans]: $\llbracket X \lesssim Y; Y \approx Z \rrbracket \implies X \lesssim Z$
 $\langle proof \rangle$

lemma *eqpollI*: $\llbracket X \lesssim Y; Y \lesssim X \rrbracket \implies X \approx Y$
 $\langle proof \rangle$

lemma *eqpollE*:
 $\llbracket X \approx Y; \llbracket X \lesssim Y; Y \lesssim X \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *eqpoll-iff*: $X \approx Y \longleftrightarrow X \lesssim Y \wedge Y \lesssim X$
 $\langle proof \rangle$

lemma *lepoll-0-is-0*: $A \lesssim 0 \implies A = 0$
 $\langle proof \rangle$

lemmas *empty-lepollI* = *empty-subsetI* [THEN *subset-imp-lepoll*]

lemma *lepoll-0-iff*: $A \lesssim 0 \longleftrightarrow A = 0$
 $\langle proof \rangle$

lemma *Un-lepoll-Un*:
 $\llbracket A \lesssim B; C \lesssim D; B \cap D = 0 \rrbracket \implies A \cup C \lesssim B \cup D$
 $\langle proof \rangle$

lemmas *eqpoll-0-is-0* = *eqpoll-imp-lepoll* [THEN *lepoll-0-is-0*]

lemma *eqpoll-0-iff*: $A \approx 0 \longleftrightarrow A = 0$
 $\langle proof \rangle$

lemma *eqpoll-disjoint-Un*:

$$\llbracket A \approx B; C \approx D; A \cap C = 0; B \cap D = 0 \rrbracket \\ \implies A \cup C \approx B \cup D$$

$\langle \text{proof} \rangle$

23.2 lesspoll: contributions by Krzysztof Grabczewski

lemma *lesspoll-not-refl*: $\neg (i \prec i)$

$\langle \text{proof} \rangle$

lemma *lesspoll-irrefl [elim!]*: $i \prec i \implies P$

$\langle \text{proof} \rangle$

lemma *lesspoll-imp-lepoll*: $A \prec B \implies A \lesssim B$

$\langle \text{proof} \rangle$

lemma *lepoll-well-ord*: $\llbracket A \lesssim B; \text{well-ord}(B, r) \rrbracket \implies \exists s. \text{well-ord}(A, s)$

$\langle \text{proof} \rangle$

lemma *lepoll-iff-leqpoll*: $A \lesssim B \longleftrightarrow A \prec B \mid A \approx B$

$\langle \text{proof} \rangle$

lemma *inj-not-surj-succ*:

assumes *fi*: $f \in \text{inj}(A, \text{succ}(m))$ **and** *fns*: $f \notin \text{surj}(A, \text{succ}(m))$

shows $\exists f. f \in \text{inj}(A, m)$

$\langle \text{proof} \rangle$

lemma *lesspoll-trans [trans]*:

$$\llbracket X \prec Y; Y \prec Z \rrbracket \implies X \prec Z$$

$\langle \text{proof} \rangle$

lemma *lesspoll-trans1 [trans]*:

$$\llbracket X \lesssim Y; Y \prec Z \rrbracket \implies X \prec Z$$

$\langle \text{proof} \rangle$

lemma *lesspoll-trans2 [trans]*:

$$\llbracket X \prec Y; Y \lesssim Z \rrbracket \implies X \prec Z$$

$\langle \text{proof} \rangle$

lemma *eq-lesspoll-trans [trans]*:

$$\llbracket X \approx Y; Y \prec Z \rrbracket \implies X \prec Z$$

$\langle \text{proof} \rangle$

lemma *lesspoll-eq-trans [trans]*:

$$\llbracket X \prec Y; Y \approx Z \rrbracket \implies X \prec Z$$

$\langle \text{proof} \rangle$

lemma *Least-equality*:

$\llbracket P(i); \text{Ord}(i); \bigwedge x. x < i \implies \neg P(x) \rrbracket \implies (\mu x. P(x)) = i$
 $\langle \text{proof} \rangle$

lemma *LeastI*:

assumes $P: P(i)$ **and** $i: \text{Ord}(i)$ **shows** $P(\mu x. P(x))$
 $\langle \text{proof} \rangle$

The proof is almost identical to the one above!

lemma *Least-le*:

assumes $P: P(i)$ **and** $i: \text{Ord}(i)$ **shows** $(\mu x. P(x)) \leq i$
 $\langle \text{proof} \rangle$

lemma *less-LeastE*: $\llbracket P(i); i < (\mu x. P(x)) \rrbracket \implies Q$

$\langle \text{proof} \rangle$

lemma *LeastI2*:

$\llbracket P(i); \text{Ord}(i); \bigwedge j. P(j) \implies Q(j) \rrbracket \implies Q(\mu j. P(j))$
 $\langle \text{proof} \rangle$

lemma *Least-0*:

$\llbracket \neg (\exists i. \text{Ord}(i) \wedge P(i)) \rrbracket \implies (\mu x. P(x)) = 0$
 $\langle \text{proof} \rangle$

lemma *Ord-Least* [*intro,simp,TC*]: $\text{Ord}(\mu x. P(x))$

$\langle \text{proof} \rangle$

23.3 Basic Properties of Cardinals

lemma *Least-cong*: $(\bigwedge y. P(y) \longleftrightarrow Q(y)) \implies (\mu x. P(x)) = (\mu x. Q(x))$

$\langle \text{proof} \rangle$

lemma *cardinal-cong*: $X \approx Y \implies |X| = |Y|$

$\langle \text{proof} \rangle$

lemma *well-ord-cardinal-epoll*:

assumes $r: \text{well-ord}(A,r)$ **shows** $|A| \approx A$
 $\langle \text{proof} \rangle$

lemmas *Ord-cardinal-epoll* = *well-ord-Memrel* [*THEN well-ord-cardinal-epoll*]

lemma *Ord-cardinal-idem*: $\text{Ord}(A) \implies ||A|| = |A|$
 $\langle \text{proof} \rangle$

lemma *well-ord-cardinal-eqE*:
assumes *woX*: *well-ord*(*X*,*r*) **and** *woY*: *well-ord*(*Y*,*s*) **and** *eq*: $|X| = |Y|$
shows $X \approx Y$
 $\langle \text{proof} \rangle$

lemma *well-ord-cardinal-epoll-iff*:
 $\llbracket \text{well-ord}(X,r); \text{well-ord}(Y,s) \rrbracket \implies |X| = |Y| \longleftrightarrow X \approx Y$
 $\langle \text{proof} \rangle$

lemma *Ord-cardinal-le*: $\text{Ord}(i) \implies |i| \leq i$
 $\langle \text{proof} \rangle$

lemma *Card-cardinal-eq*: $\text{Card}(K) \implies |K| = K$
 $\langle \text{proof} \rangle$

lemma *CardI*: $\llbracket \text{Ord}(i); \bigwedge j. j < i \implies \neg(j \approx i) \rrbracket \implies \text{Card}(i)$
 $\langle \text{proof} \rangle$

lemma *Card-is-Ord*: $\text{Card}(i) \implies \text{Ord}(i)$
 $\langle \text{proof} \rangle$

lemma *Card-cardinal-le*: $\text{Card}(K) \implies K \leq |K|$
 $\langle \text{proof} \rangle$

lemma *Ord-cardinal* [*simp,intro!*]: $\text{Ord}(|A|)$
 $\langle \text{proof} \rangle$

The cardinals are the initial ordinals.

lemma *Card-iff-initial*: $\text{Card}(K) \longleftrightarrow \text{Ord}(K) \wedge (\forall j. j < K \longrightarrow \neg j \approx K)$
 $\langle \text{proof} \rangle$

lemma *lt-Card-imp-lesspoll*: $\llbracket \text{Card}(a); i < a \rrbracket \implies i \prec a$
 $\langle \text{proof} \rangle$

lemma *Card-0*: $\text{Card}(0)$
 $\langle \text{proof} \rangle$

lemma *Card-Un*: $\llbracket \text{Card}(K); \text{Card}(L) \rrbracket \implies \text{Card}(K \cup L)$
 $\langle \text{proof} \rangle$

lemma *Card-cardinal* [iff]: $\text{Card}(|A|)$
 $\langle \text{proof} \rangle$

lemma *cardinal-eq-lemma*:
 assumes $i: |i| \leq j$ and $j: j \leq i$ shows $|j| = |i|$
 $\langle \text{proof} \rangle$

lemma *cardinal-mono*:
 assumes $ij: i \leq j$ shows $|i| \leq |j|$
 $\langle \text{proof} \rangle$

Since we have $|\text{succ}(\text{nat})| \leq |\text{nat}|$, the converse of *cardinal-mono* fails!

lemma *cardinal-lt-imp-lt*: $\llbracket |i| < |j|; \text{Ord}(i); \text{Ord}(j) \rrbracket \implies i < j$
 $\langle \text{proof} \rangle$

lemma *Card-lt-imp-lt*: $\llbracket |i| < K; \text{Ord}(i); \text{Card}(K) \rrbracket \implies i < K$
 $\langle \text{proof} \rangle$

lemma *Card-lt-iff*: $\llbracket \text{Ord}(i); \text{Card}(K) \rrbracket \implies (|i| < K) \longleftrightarrow (i < K)$
 $\langle \text{proof} \rangle$

lemma *Card-le-iff*: $\llbracket \text{Ord}(i); \text{Card}(K) \rrbracket \implies (K \leq |i|) \longleftrightarrow (K \leq i)$
 $\langle \text{proof} \rangle$

lemma *well-ord-lepoll-imp-cardinal-le*:
 assumes $wB: \text{well-ord}(B, r)$ and $AB: A \lesssim B$
 shows $|A| \leq |B|$
 $\langle \text{proof} \rangle$

lemma *lepoll-cardinal-le*: $\llbracket A \lesssim i; \text{Ord}(i) \rrbracket \implies |A| \leq i$
 $\langle \text{proof} \rangle$

lemma *lepoll-Ord-imp-epoll*: $\llbracket A \lesssim i; \text{Ord}(i) \rrbracket \implies |A| \approx A$
 $\langle \text{proof} \rangle$

lemma *lesspoll-imp-epoll*: $\llbracket A \prec i; \text{Ord}(i) \rrbracket \implies |A| \approx A$
 $\langle \text{proof} \rangle$

lemma *cardinal-subset-Ord*: $\llbracket A \leq i; \text{Ord}(i) \rrbracket \implies |A| \subseteq i$
 $\langle \text{proof} \rangle$

23.4 The finite cardinals

lemma *cons-lepoll-consD*:
 $\llbracket \text{cons}(u, A) \lesssim \text{cons}(v, B); u \notin A; v \notin B \rrbracket \implies A \lesssim B$

$\langle proof \rangle$

lemma *cons-epoll-consD*: $\llbracket cons(u,A) \approx cons(v,B); u \notin A; v \notin B \rrbracket \implies A \approx B$
 $\langle proof \rangle$

lemma *succ-lepoll-succD*: $succ(m) \lesssim succ(n) \implies m \lesssim n$
 $\langle proof \rangle$

lemma *nat-lepoll-imp-le*:
 $m \in nat \implies n \in nat \implies m \lesssim n \implies m \leq n$
 $\langle proof \rangle$

lemma *nat-epoll-iff*: $\llbracket m \in nat; n \in nat \rrbracket \implies m \approx n \longleftrightarrow m = n$
 $\langle proof \rangle$

lemma *nat-into-Card*:
assumes $n: n \in nat$ **shows** $Card(n)$
 $\langle proof \rangle$

lemmas *cardinal-0* = *nat-0I* [*THEN nat-into-Card, THEN Card-cardinal-eq, iff*]
lemmas *cardinal-1* = *nat-1I* [*THEN nat-into-Card, THEN Card-cardinal-eq, iff*]

lemma *succ-lepoll-natE*: $\llbracket succ(n) \lesssim n; n \in nat \rrbracket \implies P$
 $\langle proof \rangle$

lemma *nat-lepoll-imp-ex-epoll-n*:
 $\llbracket n \in nat; nat \lesssim X \rrbracket \implies \exists Y. Y \subseteq X \wedge n \approx Y$
 $\langle proof \rangle$

lemma *lepoll-succ*: $i \lesssim succ(i)$
 $\langle proof \rangle$

lemma *lepoll-imp-lesspoll-succ*:
assumes $A: A \lesssim m$ **and** $m: m \in nat$
shows $A \prec succ(m)$
 $\langle proof \rangle$

lemma *lesspoll-succ-imp-lepoll*:
 $\llbracket A \prec succ(m); m \in nat \rrbracket \implies A \lesssim m$
 $\langle proof \rangle$

lemma *lesspoll-succ-iff*: $m \in \text{nat} \implies A \prec \text{succ}(m) \longleftrightarrow A \lesssim m$
 $\langle \text{proof} \rangle$

lemma *lepoll-succ-disj*: $\llbracket A \lesssim \text{succ}(m); m \in \text{nat} \rrbracket \implies A \lesssim m \mid A \approx \text{succ}(m)$
 $\langle \text{proof} \rangle$

lemma *lesspoll-cardinal-lt*: $\llbracket A \prec i; \text{Ord}(i) \rrbracket \implies |A| < i$
 $\langle \text{proof} \rangle$

23.5 The first infinite cardinal: Omega, or nat

lemma *lt-not-lepoll*:
assumes $n: n < i$ $n \in \text{nat}$ **shows** $\neg i \lesssim n$
 $\langle \text{proof} \rangle$

A slightly weaker version of *nat-eqpoll-iff*

lemma *Ord-nat-eqpoll-iff*:
assumes $i: \text{Ord}(i)$ **and** $n: n \in \text{nat}$ **shows** $i \approx n \longleftrightarrow i = n$
 $\langle \text{proof} \rangle$

lemma *Card-nat*: $\text{Card}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *nat-le-cardinal*: $\text{nat} \leq i \implies \text{nat} \leq |i|$
 $\langle \text{proof} \rangle$

lemma *n-lesspoll-nat*: $n \in \text{nat} \implies n \prec \text{nat}$
 $\langle \text{proof} \rangle$

23.6 Towards Cardinal Arithmetic

lemma *cons-lepoll-cong*:
 $\llbracket A \lesssim B; b \notin B \rrbracket \implies \text{cons}(a, A) \lesssim \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *cons-eqpoll-cong*:
 $\llbracket A \approx B; a \notin A; b \notin B \rrbracket \implies \text{cons}(a, A) \approx \text{cons}(b, B)$
 $\langle \text{proof} \rangle$

lemma *cons-lepoll-cons-iff*:
 $\llbracket a \notin A; b \notin B \rrbracket \implies \text{cons}(a, A) \lesssim \text{cons}(b, B) \longleftrightarrow A \lesssim B$
 $\langle \text{proof} \rangle$

lemma *cons-eqpoll-cons-iff*:
 $\llbracket a \notin A; b \notin B \rrbracket \implies \text{cons}(a, A) \approx \text{cons}(b, B) \longleftrightarrow A \approx B$
 $\langle \text{proof} \rangle$

lemma *singleton-eqpoll-1*: $\{a\} \approx 1$

$\langle proof \rangle$

lemma *cardinal-singleton*: $|\{a\}| = 1$
 $\langle proof \rangle$

lemma *not-0-is-lepoll-1*: $A \neq 0 \implies 1 \lesssim A$
 $\langle proof \rangle$

lemma *succ-epoll-cong*: $A \approx B \implies succ(A) \approx succ(B)$
 $\langle proof \rangle$

lemma *sum-epoll-cong*: $\llbracket A \approx C; B \approx D \rrbracket \implies A+B \approx C+D$
 $\langle proof \rangle$

lemma *prod-epoll-cong*:
 $\llbracket A \approx C; B \approx D \rrbracket \implies A*B \approx C*D$
 $\langle proof \rangle$

lemma *inj-disjoint-epoll*:
 $\llbracket f \in inj(A,B); A \cap B = 0 \rrbracket \implies A \cup (B - range(f)) \approx B$
 $\langle proof \rangle$

23.7 Lemmas by Krzysztof Grabczewski

If A has at most $n + 1$ elements and $a \in A$ then $A - \{a\}$ has at most n .

lemma *Diff-sing-lepoll*:
 $\llbracket a \in A; A \lesssim succ(n) \rrbracket \implies A - \{a\} \lesssim n$
 $\langle proof \rangle$

If A has at least $n + 1$ elements then $A - \{a\}$ has at least n .

lemma *lepoll-Diff-sing*:
assumes A : $succ(n) \lesssim A$ **shows** $n \lesssim A - \{a\}$
 $\langle proof \rangle$

lemma *Diff-sing-epoll*: $\llbracket a \in A; A \approx succ(n) \rrbracket \implies A - \{a\} \approx n$
 $\langle proof \rangle$

lemma *lepoll-1-is-sing*: $\llbracket A \lesssim 1; a \in A \rrbracket \implies A = \{a\}$
 $\langle proof \rangle$

lemma *Un-lepoll-sum*: $A \cup B \lesssim A+B$
 $\langle proof \rangle$

lemma *well-ord-Un*:
 $\llbracket well-ord(X,R); well-ord(Y,S) \rrbracket \implies \exists T. well-ord(X \cup Y, T)$
 $\langle proof \rangle$

lemma *disj-Un-epoll-sum*: $A \cap B = 0 \implies A \cup B \approx A + B$
 $\langle \text{proof} \rangle$

23.8 Finite and infinite sets

lemma *epoll-imp-Finite-iff*: $A \approx B \implies \text{Finite}(A) \longleftrightarrow \text{Finite}(B)$
 $\langle \text{proof} \rangle$

lemma *Finite-0* [*simp*]: $\text{Finite}(0)$
 $\langle \text{proof} \rangle$

lemma *Finite-cons*: $\text{Finite}(x) \implies \text{Finite}(\text{cons}(y, x))$
 $\langle \text{proof} \rangle$

lemma *Finite-succ*: $\text{Finite}(x) \implies \text{Finite}(\text{succ}(x))$
 $\langle \text{proof} \rangle$

lemma *lepoll-nat-imp-Finite*:
assumes $A: A \lesssim n$ **and** $n: n \in \text{nat}$ **shows** $\text{Finite}(A)$
 $\langle \text{proof} \rangle$

lemma *lesspoll-nat-is-Finite*:
 $A \prec \text{nat} \implies \text{Finite}(A)$
 $\langle \text{proof} \rangle$

lemma *lepoll-Finite*:
assumes $Y: Y \lesssim X$ **and** $X: \text{Finite}(X)$ **shows** $\text{Finite}(Y)$
 $\langle \text{proof} \rangle$

lemmas *subset-Finite* = *subset-imp-lepoll* [*THEN lepoll-Finite*]

lemma *Finite-cons-iff* [*iff*]: $\text{Finite}(\text{cons}(y, x)) \longleftrightarrow \text{Finite}(x)$
 $\langle \text{proof} \rangle$

lemma *Finite-succ-iff* [*iff*]: $\text{Finite}(\text{succ}(x)) \longleftrightarrow \text{Finite}(x)$
 $\langle \text{proof} \rangle$

lemma *Finite-Int*: $\text{Finite}(A) \mid \text{Finite}(B) \implies \text{Finite}(A \cap B)$
 $\langle \text{proof} \rangle$

lemmas *Finite-Diff* = *Diff-subset* [*THEN subset-Finite*]

lemma *nat-le-infinite-Ord*:
 $\llbracket \text{Ord}(i); \neg \text{Finite}(i) \rrbracket \implies \text{nat} \leq i$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-well-ord*:

$Finite(A) \implies \exists r. well_ord(A, r)$
 $\langle proof \rangle$

lemma *succ-lepoll-imp-not-empty*: $succ(x) \lesssim y \implies y \neq 0$
 $\langle proof \rangle$

lemma *eqpoll-succ-imp-not-empty*: $x \approx succ(n) \implies x \neq 0$
 $\langle proof \rangle$

lemma *Finite-Fin-lemma* [rule-format]:
 $n \in nat \implies \forall A. (A \approx n \wedge A \subseteq X) \longrightarrow A \in Fin(X)$
 $\langle proof \rangle$

lemma *Finite-Fin*: $\llbracket Finite(A); A \subseteq X \rrbracket \implies A \in Fin(X)$
 $\langle proof \rangle$

lemma *Fin-lemma* [rule-format]: $n \in nat \implies \forall A. A \approx n \longrightarrow A \in Fin(A)$
 $\langle proof \rangle$

lemma *Finite-into-Fin*: $Finite(A) \implies A \in Fin(A)$
 $\langle proof \rangle$

lemma *Fin-into-Finite*: $A \in Fin(U) \implies Finite(A)$
 $\langle proof \rangle$

lemma *Finite-Fin-iff*: $Finite(A) \longleftrightarrow A \in Fin(A)$
 $\langle proof \rangle$

lemma *Finite-Un*: $\llbracket Finite(A); Finite(B) \rrbracket \implies Finite(A \cup B)$
 $\langle proof \rangle$

lemma *Finite-Un-iff* [simp]: $Finite(A \cup B) \longleftrightarrow (Finite(A) \wedge Finite(B))$
 $\langle proof \rangle$

The converse must hold too.

lemma *Finite-Union*: $\llbracket \forall y \in X. Finite(y); Finite(X) \rrbracket \implies Finite(\bigcup(X))$
 $\langle proof \rangle$

lemma *Finite-induct* [case-names 0 cons, induct set: Finite]:
 $\llbracket Finite(A); P(0);$
 $\quad \bigwedge x B. \llbracket Finite(B); x \notin B; P(B) \rrbracket \implies P(cons(x, B)) \rrbracket$
 $\implies P(A)$
 $\langle proof \rangle$

lemma *Diff-sing-Finite*: $Finite(A - \{a\}) \implies Finite(A)$
 $\langle proof \rangle$

lemma *Diff-Finite* [rule-format]: $Finite(B) \implies Finite(A-B) \longrightarrow Finite(A)$
 <proof>

lemma *Finite-RepFun*: $Finite(A) \implies Finite(RepFun(A,f))$
 <proof>

lemma *Finite-RepFun-iff-lemma* [rule-format]:

$$\llbracket Finite(x); \bigwedge x y. f(x)=f(y) \implies x=y \rrbracket$$

$$\implies \forall A. x = RepFun(A,f) \longrightarrow Finite(A)$$
 <proof>

I don't know why, but if the premise is expressed using meta-connectives then the simplifier cannot prove it automatically in conditional rewriting.

lemma *Finite-RepFun-iff*:

$$(\forall x y. f(x)=f(y) \longrightarrow x=y) \implies Finite(RepFun(A,f)) \longleftrightarrow Finite(A)$$
 <proof>

lemma *Finite-Pow*: $Finite(A) \implies Finite(Pow(A))$
 <proof>

lemma *Finite-Pow-imp-Finite*: $Finite(Pow(A)) \implies Finite(A)$
 <proof>

lemma *Finite-Pow-iff* [iff]: $Finite(Pow(A)) \longleftrightarrow Finite(A)$
 <proof>

lemma *Finite-cardinal-iff*:
assumes i : $Ord(i)$ **shows** $Finite(|i|) \longleftrightarrow Finite(i)$
 <proof>

lemma *nat-wf-on-converse-Memrel*: $n \in nat \implies wf[n](converse(Memrel(n)))$
 <proof>

lemma *nat-well-ord-converse-Memrel*: $n \in nat \implies well_ord(n, converse(Memrel(n)))$
 <proof>

lemma *well-ord-converse*:

$$\llbracket well_ord(A,r);$$

$$well_ord(ordertype(A,r), converse(Memrel(ordertype(A, r)))) \rrbracket$$

$$\implies well_ord(A, converse(r))$$
 <proof>

lemma *ordertype-eq-n*:
assumes r : $well_ord(A,r)$ **and** A : $A \approx n$ **and** n : $n \in nat$
shows $ordertype(A,r) = n$

$\langle proof \rangle$

lemma *Finite-well-ord-converse*:

$\llbracket Finite(A); well\text{-}ord(A,r) \rrbracket \implies well\text{-}ord(A, converse(r))$

$\langle proof \rangle$

lemma *nat-into-Finite*: $n \in nat \implies Finite(n)$

$\langle proof \rangle$

lemma *nat-not-Finite*: $\neg Finite(nat)$

$\langle proof \rangle$

end

24 The Cumulative Hierarchy and a Small Universe for Recursive Types

theory *Univ* **imports** *Epsilon Cardinal* **begin**

definition

$Vfrom :: [i,i] \Rightarrow i$ **where**
 $Vfrom(A,i) \equiv transrec(i, \lambda x f. A \cup (\bigcup y \in x. Pow(f'y)))$

abbreviation

$Vset :: i \Rightarrow i$ **where**
 $Vset(x) \equiv Vfrom(0,x)$

definition

$Vrec :: [i, [i,i] \Rightarrow i] \Rightarrow i$ **where**
 $Vrec(a,H) \equiv transrec(rank(a), \lambda x g. \lambda z \in Vset(succ(x)).$
 $H(z, \lambda w \in Vset(x). g'rank(w)'w)) \text{ ' } a$

definition

$Vrecursor :: [[i,i] \Rightarrow i, i] \Rightarrow i$ **where**
 $Vrecursor(H,a) \equiv transrec(rank(a), \lambda x g. \lambda z \in Vset(succ(x)).$
 $H(\lambda w \in Vset(x). g'rank(w)'w, z)) \text{ ' } a$

definition

$univ :: i \Rightarrow i$ **where**
 $univ(A) \equiv Vfrom(A,nat)$

24.1 Immediate Consequences of the Definition of $Vfrom(A, i)$

NOT SUITABLE FOR REWRITING – RECURSIVE!

lemma *Vfrom*: $Vfrom(A,i) = A \cup (\bigcup j \in i. Pow(Vfrom(A,j)))$

$\langle proof \rangle$

24.1.1 Monotonicity

lemma *Vfrom-mono* [rule-format]:

$A \leq B \implies \forall j. i \leq j \longrightarrow Vfrom(A, i) \subseteq Vfrom(B, j)$
 $\langle proof \rangle$

lemma *VfromI*: $\llbracket a \in Vfrom(A, j); j < i \rrbracket \implies a \in Vfrom(A, i)$
 $\langle proof \rangle$

24.1.2 A fundamental equality: Vfrom does not require ordinals!

lemma *Vfrom-rank-subset1*: $Vfrom(A, x) \subseteq Vfrom(A, rank(x))$
 $\langle proof \rangle$

lemma *Vfrom-rank-subset2*: $Vfrom(A, rank(x)) \subseteq Vfrom(A, x)$
 $\langle proof \rangle$

lemma *Vfrom-rank-eq*: $Vfrom(A, rank(x)) = Vfrom(A, x)$
 $\langle proof \rangle$

24.2 Basic Closure Properties

lemma *zero-in-Vfrom*: $y : x \implies 0 \in Vfrom(A, x)$
 $\langle proof \rangle$

lemma *i-subset-Vfrom*: $i \subseteq Vfrom(A, i)$
 $\langle proof \rangle$

lemma *A-subset-Vfrom*: $A \subseteq Vfrom(A, i)$
 $\langle proof \rangle$

lemmas *A-into-Vfrom = A-subset-Vfrom* [THEN subsetD]

lemma *subset-mem-Vfrom*: $a \subseteq Vfrom(A, i) \implies a \in Vfrom(A, succ(i))$
 $\langle proof \rangle$

24.2.1 Finite sets and ordered pairs

lemma *singleton-in-Vfrom*: $a \in Vfrom(A, i) \implies \{a\} \in Vfrom(A, succ(i))$
 $\langle proof \rangle$

lemma *doubleton-in-Vfrom*:
 $\llbracket a \in Vfrom(A, i); b \in Vfrom(A, i) \rrbracket \implies \{a, b\} \in Vfrom(A, succ(i))$
 $\langle proof \rangle$

lemma *Pair-in-Vfrom*:
 $\llbracket a \in Vfrom(A, i); b \in Vfrom(A, i) \rrbracket \implies \langle a, b \rangle \in Vfrom(A, succ(succ(i)))$
 $\langle proof \rangle$

lemma *succ-in-Vfrom*: $a \subseteq Vfrom(A, i) \implies succ(a) \in Vfrom(A, succ(succ(i)))$

$\langle proof \rangle$

24.3 0, Successor and Limit Equations for $Vfrom$

lemma *Vfrom-0*: $Vfrom(A, 0) = A$

$\langle proof \rangle$

lemma *Vfrom-succ-lemma*: $Ord(i) \implies Vfrom(A, succ(i)) = A \cup Pow(Vfrom(A, i))$

$\langle proof \rangle$

lemma *Vfrom-succ*: $Vfrom(A, succ(i)) = A \cup Pow(Vfrom(A, i))$

$\langle proof \rangle$

lemma *Vfrom-Union*: $y:X \implies Vfrom(A, \bigcup(X)) = (\bigcup y \in X. Vfrom(A, y))$

$\langle proof \rangle$

24.4 $Vfrom$ applied to Limit Ordinals

lemma *Limit-Vfrom-eq*:

$Limit(i) \implies Vfrom(A, i) = (\bigcup y \in i. Vfrom(A, y))$

$\langle proof \rangle$

lemma *Limit-VfromE*:

$\llbracket a \in Vfrom(A, i); \neg R \implies Limit(i);$
 $\bigwedge x. \llbracket x < i; a \in Vfrom(A, x) \rrbracket \implies R$

$\rrbracket \implies R$

$\langle proof \rangle$

lemma *singleton-in-VLimit*:

$\llbracket a \in Vfrom(A, i); Limit(i) \rrbracket \implies \{a\} \in Vfrom(A, i)$

$\langle proof \rangle$

lemmas *Vfrom-UnI1* =

Un-upper1 [*THEN subset-refl* [*THEN Vfrom-mono*, *THEN subsetD*]]

lemmas *Vfrom-UnI2* =

Un-upper2 [*THEN subset-refl* [*THEN Vfrom-mono*, *THEN subsetD*]]

Hard work is finding a single $j:i$ such that $a, b \leq Vfrom(A, j)$

lemma *doubleton-in-VLimit*:

$\llbracket a \in Vfrom(A, i); b \in Vfrom(A, i); Limit(i) \rrbracket \implies \{a, b\} \in Vfrom(A, i)$

$\langle proof \rangle$

lemma *Pair-in-VLimit*:

$\llbracket a \in Vfrom(A, i); b \in Vfrom(A, i); Limit(i) \rrbracket \implies \langle a, b \rangle \in Vfrom(A, i)$

Infer that a, b occur at ordinals $x, x_a < i$.

$\langle proof \rangle$

lemma *product-VLimit*: $\text{Limit}(i) \implies \text{Vfrom}(A, i) * \text{Vfrom}(A, i) \subseteq \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemmas *Sigma-subset-VLimit* =
 $\text{subset-trans } [OF \text{ Sigma-mono product-VLimit}]$

lemmas *nat-subset-VLimit* =
 $\text{subset-trans } [OF \text{ nat-le-Limit } [THEN \text{ le-imp-subset}] \text{ i-subset-Vfrom}]$

lemma *nat-into-VLimit*: $\llbracket n: \text{nat}; \text{Limit}(i) \rrbracket \implies n \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

24.4.1 Closure under Disjoint Union

lemmas *zero-in-VLimit* = $\text{Limit-has-0 } [THEN \text{ ltD}, THEN \text{ zero-in-Vfrom}]$

lemma *one-in-VLimit*: $\text{Limit}(i) \implies 1 \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Inl-in-VLimit*:
 $\llbracket a \in \text{Vfrom}(A, i); \text{Limit}(i) \rrbracket \implies \text{Inl}(a) \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Inr-in-VLimit*:
 $\llbracket b \in \text{Vfrom}(A, i); \text{Limit}(i) \rrbracket \implies \text{Inr}(b) \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *sum-VLimit*: $\text{Limit}(i) \implies \text{Vfrom}(C, i) + \text{Vfrom}(C, i) \subseteq \text{Vfrom}(C, i)$
 $\langle \text{proof} \rangle$

lemmas *sum-subset-VLimit* = $\text{subset-trans } [OF \text{ sum-mono sum-VLimit}]$

24.5 Properties assuming $\text{Transset}(A)$

lemma *Transset-Vfrom*: $\text{Transset}(A) \implies \text{Transset}(\text{Vfrom}(A, i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Vfrom-succ*:
 $\text{Transset}(A) \implies \text{Vfrom}(A, \text{succ}(i)) = \text{Pow}(\text{Vfrom}(A, i))$
 $\langle \text{proof} \rangle$

lemma *Transset-Pair-subset*: $\llbracket \langle a, b \rangle \subseteq C; \text{Transset}(C) \rrbracket \implies a: C \wedge b: C$
 $\langle \text{proof} \rangle$

lemma *Transset-Pair-subset-VLimit*:
 $\llbracket \langle a, b \rangle \subseteq \text{Vfrom}(A, i); \text{Transset}(A); \text{Limit}(i) \rrbracket$
 $\implies \langle a, b \rangle \in \text{Vfrom}(A, i)$
 $\langle \text{proof} \rangle$

lemma *Union-in-Vfrom*:

$\llbracket X \in V_{\text{from}}(A, j); \text{Transset}(A) \rrbracket \implies \bigcup (X) \in V_{\text{from}}(A, \text{succ}(j))$
 $\langle \text{proof} \rangle$

lemma *Union-in-VLimit:*

$\llbracket X \in V_{\text{from}}(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket \implies \bigcup (X) \in V_{\text{from}}(A, i)$
 $\langle \text{proof} \rangle$

General theorem for membership in $V_{\text{from}}(A, i)$ when i is a limit ordinal

lemma *in-VLimit:*

$\llbracket a \in V_{\text{from}}(A, i); b \in V_{\text{from}}(A, i); \text{Limit}(i);$
 $\bigwedge x y j. \llbracket j < i; 1:j; x \in V_{\text{from}}(A, j); y \in V_{\text{from}}(A, j) \rrbracket$
 $\implies \exists k. h(x, y) \in V_{\text{from}}(A, k) \wedge k < i \rrbracket$
 $\implies h(a, b) \in V_{\text{from}}(A, i)$

Infer that a, b occur at ordinals $x, x_a < i$.

$\langle \text{proof} \rangle$

24.5.1 Products

lemma *prod-in-Vfrom:*

$\llbracket a \in V_{\text{from}}(A, j); b \in V_{\text{from}}(A, j); \text{Transset}(A) \rrbracket$
 $\implies a * b \in V_{\text{from}}(A, \text{succ}(\text{succ}(\text{succ}(j))))$
 $\langle \text{proof} \rangle$

lemma *prod-in-VLimit:*

$\llbracket a \in V_{\text{from}}(A, i); b \in V_{\text{from}}(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket$
 $\implies a * b \in V_{\text{from}}(A, i)$
 $\langle \text{proof} \rangle$

24.5.2 Disjoint Sums, or Quine Ordered Pairs

lemma *sum-in-Vfrom:*

$\llbracket a \in V_{\text{from}}(A, j); b \in V_{\text{from}}(A, j); \text{Transset}(A); 1:j \rrbracket$
 $\implies a + b \in V_{\text{from}}(A, \text{succ}(\text{succ}(\text{succ}(j))))$
 $\langle \text{proof} \rangle$

lemma *sum-in-VLimit:*

$\llbracket a \in V_{\text{from}}(A, i); b \in V_{\text{from}}(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket$
 $\implies a + b \in V_{\text{from}}(A, i)$
 $\langle \text{proof} \rangle$

24.5.3 Function Space!

lemma *fun-in-Vfrom:*

$\llbracket a \in V_{\text{from}}(A, j); b \in V_{\text{from}}(A, j); \text{Transset}(A) \rrbracket \implies$
 $a \multimap b \in V_{\text{from}}(A, \text{succ}(\text{succ}(\text{succ}(\text{succ}(j))))$
 $\langle \text{proof} \rangle$

lemma *fun-in-VLimit:*

$\llbracket a \in V_{\text{from}}(A, i); b \in V_{\text{from}}(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket$

$\implies a \multimap b \in Vfrom(A, i)$
 $\langle proof \rangle$

lemma *Pow-in-Vfrom*:

$\llbracket a \in Vfrom(A, j); \text{Transset}(A) \rrbracket \implies Pow(a) \in Vfrom(A, succ(succ(j)))$
 $\langle proof \rangle$

lemma *Pow-in-VLimit*:

$\llbracket a \in Vfrom(A, i); \text{Limit}(i); \text{Transset}(A) \rrbracket \implies Pow(a) \in Vfrom(A, i)$
 $\langle proof \rangle$

24.6 The Set $Vset(i)$

lemma *Vset*: $Vset(i) = (\bigcup j \in i. Pow(Vset(j)))$
 $\langle proof \rangle$

lemmas *Vset-succ* = *Transset-0* [THEN *Transset-Vfrom-succ*]

lemmas *Transset-Vset* = *Transset-0* [THEN *Transset-Vfrom*]

24.6.1 Characterisation of the elements of $Vset(i)$

lemma *VsetD* [rule-format]: $Ord(i) \implies \forall b. b \in Vset(i) \longrightarrow rank(b) < i$
 $\langle proof \rangle$

lemma *VsetI-lemma* [rule-format]:

$Ord(i) \implies \forall b. rank(b) \in i \longrightarrow b \in Vset(i)$
 $\langle proof \rangle$

lemma *VsetI*: $rank(x) < i \implies x \in Vset(i)$
 $\langle proof \rangle$

Merely a lemma for the next result

lemma *Vset-Ord-rank-iff*: $Ord(i) \implies b \in Vset(i) \longleftrightarrow rank(b) < i$
 $\langle proof \rangle$

lemma *Vset-rank-iff* [simp]: $b \in Vset(a) \longleftrightarrow rank(b) < rank(a)$
 $\langle proof \rangle$

This is $rank(rank(a)) = rank(a)$

declare *Ord-rank* [THEN *rank-of-Ord*, *simp*]

lemma *rank-Vset*: $Ord(i) \implies rank(Vset(i)) = i$
 $\langle proof \rangle$

lemma *Finite-Vset*: $i \in nat \implies Finite(Vset(i))$
 $\langle proof \rangle$

24.6.2 Reasoning about Sets in Terms of Their Elements' Ranks

lemma *arg-subset-Vset-rank*: $a \subseteq Vset(rank(a))$

$\langle proof \rangle$

lemma *Int-Vset-subset*:

$\llbracket \bigwedge i. \text{Ord}(i) \implies a \cap \text{Vset}(i) \subseteq b \rrbracket \implies a \subseteq b$
 $\langle proof \rangle$

24.6.3 Set Up an Environment for Simplification

lemma *rank-Inl*: $\text{rank}(a) < \text{rank}(\text{Inl}(a))$
 $\langle proof \rangle$

lemma *rank-Inr*: $\text{rank}(a) < \text{rank}(\text{Inr}(a))$
 $\langle proof \rangle$

lemmas *rank-rls* = *rank-Inl rank-Inr rank-pair1 rank-pair2*

24.6.4 Recursion over Vset Levels!

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrec*: $\text{Vrec}(a, H) = H(a, \lambda x \in \text{Vset}(\text{rank}(a)). \text{Vrec}(x, H))$
 $\langle proof \rangle$

This form avoids giant explosions in proofs. NOTE the form of the premise!

lemma *def-Vrec*:
 $\llbracket \bigwedge x. h(x) \equiv \text{Vrec}(x, H) \rrbracket \implies$
 $h(a) = H(a, \lambda x \in \text{Vset}(\text{rank}(a)). h(x))$
 $\langle proof \rangle$

NOT SUITABLE FOR REWRITING: recursive!

lemma *Vrecursor*:
 $\text{Vrecursor}(H, a) = H(\lambda x \in \text{Vset}(\text{rank}(a)). \text{Vrecursor}(H, x), a)$
 $\langle proof \rangle$

This form avoids giant explosions in proofs. NOTE the form of the premise!

lemma *def-Vrecursor*:
 $h \equiv \text{Vrecursor}(H) \implies h(a) = H(\lambda x \in \text{Vset}(\text{rank}(a)). h(x), a)$
 $\langle proof \rangle$

24.7 The Datatype Universe: *univ*(A)

lemma *univ-mono*: $A \leq B \implies \text{univ}(A) \subseteq \text{univ}(B)$
 $\langle proof \rangle$

lemma *Transset-univ*: $\text{Transset}(A) \implies \text{Transset}(\text{univ}(A))$
 $\langle proof \rangle$

24.7.1 The Set $\text{univ}(A)$ as a Limit

lemma *univ-eq-UN*: $\text{univ}(A) = (\bigcup i \in \text{nat}. V\text{from}(A, i))$
 $\langle \text{proof} \rangle$

lemma *subset-univ-eq-Int*: $c \subseteq \text{univ}(A) \implies c = (\bigcup i \in \text{nat}. c \cap V\text{from}(A, i))$
 $\langle \text{proof} \rangle$

lemma *univ-Int-Vfrom-subset*:
 $\llbracket a \subseteq \text{univ}(X);$
 $\bigwedge i. i : \text{nat} \implies a \cap V\text{from}(X, i) \subseteq b \rrbracket$
 $\implies a \subseteq b$
 $\langle \text{proof} \rangle$

lemma *univ-Int-Vfrom-eq*:
 $\llbracket a \subseteq \text{univ}(X); \quad b \subseteq \text{univ}(X);$
 $\bigwedge i. i : \text{nat} \implies a \cap V\text{from}(X, i) = b \cap V\text{from}(X, i)$
 $\rrbracket \implies a = b$
 $\langle \text{proof} \rangle$

24.8 Closure Properties for $\text{univ}(A)$

lemma *zero-in-univ*: $0 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *zero-subset-univ*: $\{0\} \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *A-subset-univ*: $A \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *A-into-univ* = *A-subset-univ* [THEN subsetD]

24.8.1 Closure under Unordered and Ordered Pairs

lemma *singleton-in-univ*: $a : \text{univ}(A) \implies \{a\} \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *doubleton-in-univ*:
 $\llbracket a : \text{univ}(A); \quad b : \text{univ}(A) \rrbracket \implies \{a, b\} \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *Pair-in-univ*:
 $\llbracket a : \text{univ}(A); \quad b : \text{univ}(A) \rrbracket \implies \langle a, b \rangle \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *Union-in-univ*:
 $\llbracket X : \text{univ}(A); \quad \text{Transset}(A) \rrbracket \implies \bigcup (X) \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *product-univ*: $\text{univ}(A) * \text{univ}(A) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.8.2 The Natural Numbers

lemma *nat-subset-univ*: $\text{nat} \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *nat-into-univ*: $n \in \text{nat} \implies n \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.8.3 Instances for 1 and 2

lemma *one-in-univ*: $1 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

unused!

lemma *two-in-univ*: $2 \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *bool-subset-univ*: $\text{bool} \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *bool-into-univ* = *bool-subset-univ* [THEN subsetD]

24.8.4 Closure under Disjoint Union

lemma *Inl-in-univ*: $a: \text{univ}(A) \implies \text{Inl}(a) \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *Inr-in-univ*: $b: \text{univ}(A) \implies \text{Inr}(b) \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *sum-univ*: $\text{univ}(C) + \text{univ}(C) \subseteq \text{univ}(C)$
 $\langle \text{proof} \rangle$

lemmas *sum-subset-univ* = *subset-trans* [OF sum-mono sum-univ]

lemma *Sigma-subset-univ*:
 $\llbracket A \subseteq \text{univ}(D); \bigwedge x. x \in A \implies B(x) \subseteq \text{univ}(D) \rrbracket \implies \text{Sigma}(A, B) \subseteq \text{univ}(D)$
 $\langle \text{proof} \rangle$

24.9 Finite Branching Closure Properties

24.9.1 Closure under Finite Powerset

lemma *Fin-Vfrom-lemma*:
 $\llbracket b: \text{Fin}(\text{Vfrom}(A, i)); \text{Limit}(i) \rrbracket \implies \exists j. b \subseteq \text{Vfrom}(A, j) \wedge j < i$
 $\langle \text{proof} \rangle$

lemma *Fin-VLimit*: $\text{Limit}(i) \implies \text{Fin}(\text{Vfrom}(A,i)) \subseteq \text{Vfrom}(A,i)$
 $\langle \text{proof} \rangle$

lemmas *Fin-subset-VLimit* = *subset-trans* [*OF Fin-mono Fin-VLimit*]

lemma *Fin-univ*: $\text{Fin}(\text{univ}(A)) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.9.2 Closure under Finite Powers: Functions from a Natural Number

lemma *nat-fun-VLimit*:
 $\llbracket n: \text{nat}; \text{Limit}(i) \rrbracket \implies n \rightarrow \text{Vfrom}(A,i) \subseteq \text{Vfrom}(A,i)$
 $\langle \text{proof} \rangle$

lemmas *nat-fun-subset-VLimit* = *subset-trans* [*OF Pi-mono nat-fun-VLimit*]

lemma *nat-fun-univ*: $n: \text{nat} \implies n \rightarrow \text{univ}(A) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

24.9.3 Closure under Finite Function Space

General but seldom-used version; normally the domain is fixed

lemma *FiniteFun-VLimit1*:
 $\text{Limit}(i) \implies \text{Vfrom}(A,i) \multimap \text{Vfrom}(A,i) \subseteq \text{Vfrom}(A,i)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-univ1*: $\text{univ}(A) \multimap \text{univ}(A) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

Version for a fixed domain

lemma *FiniteFun-VLimit*:
 $\llbracket W \subseteq \text{Vfrom}(A,i); \text{Limit}(i) \rrbracket \implies W \multimap \text{Vfrom}(A,i) \subseteq \text{Vfrom}(A,i)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-univ*:
 $W \subseteq \text{univ}(A) \implies W \multimap \text{univ}(A) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemma *FiniteFun-in-univ*:
 $\llbracket f: W \multimap \text{univ}(A); W \subseteq \text{univ}(A) \rrbracket \implies f \in \text{univ}(A)$
 $\langle \text{proof} \rangle$

Remove \subseteq from the rule above

lemmas *FiniteFun-in-univ'* = *FiniteFun-in-univ* [*OF - subsetI*]

24.10 * For QUniv. Properties of Vfrom analogous to the "take-lemma" *

Intersecting $a*b$ with Vfrom...

This version says a, b exist one level down, in the smaller set $Vfrom(X, i)$

lemma *doubleton-in-Vfrom-D*:

$$\begin{aligned} & \llbracket \{a, b\} \in Vfrom(X, succ(i)); \text{Transset}(X) \rrbracket \\ & \implies a \in Vfrom(X, i) \wedge b \in Vfrom(X, i) \\ & \langle proof \rangle \end{aligned}$$

This weaker version says a, b exist at the same level

lemmas *Vfrom-doubleton-D = Transset-Vfrom [THEN Transset-doubleton-D]*

lemma *Pair-in-Vfrom-D*:

$$\begin{aligned} & \llbracket \langle a, b \rangle \in Vfrom(X, succ(i)); \text{Transset}(X) \rrbracket \\ & \implies a \in Vfrom(X, i) \wedge b \in Vfrom(X, i) \\ & \langle proof \rangle \end{aligned}$$

lemma *product-Int-Vfrom-subset*:

$$\begin{aligned} & \text{Transset}(X) \implies \\ & (a*b) \cap Vfrom(X, succ(i)) \subseteq (a \cap Vfrom(X, i)) * (b \cap Vfrom(X, i)) \\ & \langle proof \rangle \end{aligned}$$

$\langle ML \rangle$

end

25 A Small Universe for Lazy Recursive Types

theory *QUniv* **imports** *Univ QPair* **begin**

rep-datatype

elimination *sumE*
induction *TrueI*
case-eqns *case-Inl case-Inr*

rep-datatype

elimination *qsumE*
induction *TrueI*
case-eqns *qcase-QInl qcase-QInr*

definition

$quniv :: i \Rightarrow i$ **where**
 $quniv(A) \equiv Pow(univ(eclose(A)))$

25.1 Properties involving Transset and Sum

lemma *Transset-includes-summands*:

$\llbracket Transset(C); A+B \subseteq C \rrbracket \Longrightarrow A \subseteq C \wedge B \subseteq C$
 $\langle proof \rangle$

lemma *Transset-sum-Int-subset*:

$Transset(C) \Longrightarrow (A+B) \cap C \subseteq (A \cap C) + (B \cap C)$
 $\langle proof \rangle$

25.2 Introduction and Elimination Rules

lemma *qunivI*: $X \subseteq univ(eclose(A)) \Longrightarrow X \in quniv(A)$
 $\langle proof \rangle$

lemma *qunivD*: $X \in quniv(A) \Longrightarrow X \subseteq univ(eclose(A))$
 $\langle proof \rangle$

lemma *quniv-mono*: $A \leq B \Longrightarrow quniv(A) \subseteq quniv(B)$
 $\langle proof \rangle$

25.3 Closure Properties

lemma *univ-eclose-subset-quniv*: $univ(eclose(A)) \subseteq quniv(A)$
 $\langle proof \rangle$

lemma *univ-subset-quniv*: $univ(A) \subseteq quniv(A)$
 $\langle proof \rangle$

lemmas *univ-into-quniv* = *univ-subset-quniv* [THEN subsetD]

lemma *Pow-univ-subset-quniv*: $Pow(univ(A)) \subseteq quniv(A)$
 $\langle proof \rangle$

lemmas *univ-subset-into-quniv* =
PowI [THEN *Pow-univ-subset-quniv* [THEN subsetD]]

lemmas *zero-in-quniv* = *zero-in-univ* [THEN *univ-into-quniv*]

lemmas *one-in-quniv* = *one-in-univ* [THEN *univ-into-quniv*]

lemmas *two-in-quniv* = *two-in-univ* [THEN *univ-into-quniv*]

lemmas *A-subset-quniv* = *subset-trans* [OF *A-subset-univ univ-subset-quniv*]

lemmas *A-into-quniv* = *A-subset-quniv* [THEN subsetD]

lemma *QPair-subset-univ*:

$\llbracket a \subseteq \text{univ}(A); b \subseteq \text{univ}(A) \rrbracket \implies \langle a; b \rangle \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

25.4 Quine Disjoint Sum

lemma *QInl-subset-univ*: $a \subseteq \text{univ}(A) \implies \text{QInl}(a) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

lemmas *naturals-subset-nat* =

Ord-nat [*THEN Ord-is-Transset, unfolded Transset-def, THEN bspec*]

lemmas *naturals-subset-univ* =

subset-trans [*OF naturals-subset-nat nat-subset-univ*]

lemma *QInr-subset-univ*: $a \subseteq \text{univ}(A) \implies \text{QInr}(a) \subseteq \text{univ}(A)$
 $\langle \text{proof} \rangle$

25.5 Closure for Quine-Inspired Products and Sums

lemma *QPair-in-quniv*:

$\llbracket a: \text{quniv}(A); b: \text{quniv}(A) \rrbracket \implies \langle a; b \rangle \in \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *QSigma-quniv*: $\text{quniv}(A) <*> \text{quniv}(A) \subseteq \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *QSigma-subset-quniv* = *subset-trans* [*OF QSigma-mono QSigma-quniv*]

lemma *quniv-QPair-D*:

$\langle a; b \rangle \in \text{quniv}(A) \implies a: \text{quniv}(A) \wedge b: \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemmas *quniv-QPair-E* = *quniv-QPair-D* [*THEN conjE*]

lemma *quniv-QPair-iff*: $\langle a; b \rangle \in \text{quniv}(A) \longleftrightarrow a: \text{quniv}(A) \wedge b: \text{quniv}(A)$
 $\langle \text{proof} \rangle$

25.6 Quine Disjoint Sum

lemma *QInl-in-quniv*: $a: \text{quniv}(A) \implies \text{QInl}(a) \in \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *QInr-in-quniv*: $b: \text{quniv}(A) \implies \text{QInr}(b) \in \text{quniv}(A)$
 $\langle \text{proof} \rangle$

lemma *qsum-quniv*: $\text{quniv}(C) <+> \text{quniv}(C) \subseteq \text{quniv}(C)$

$\langle proof \rangle$

lemmas $qsum\text{-}subset\text{-}quniv = subset\text{-}trans$ [OF $qsum\text{-}mono$ $qsum\text{-}quniv$]

25.7 The Natural Numbers

lemmas $nat\text{-}subset\text{-}quniv = subset\text{-}trans$ [OF $nat\text{-}subset\text{-}univ$ $univ\text{-}subset\text{-}quniv$]

lemmas $nat\text{-}into\text{-}quniv = nat\text{-}subset\text{-}quniv$ [THEN $subsetD$]

lemmas $bool\text{-}subset\text{-}quniv = subset\text{-}trans$ [OF $bool\text{-}subset\text{-}univ$ $univ\text{-}subset\text{-}quniv$]

lemmas $bool\text{-}into\text{-}quniv = bool\text{-}subset\text{-}quniv$ [THEN $subsetD$]

lemma $QPair\text{-}Int\text{-}Vfrom\text{-}succ\text{-}subset$:

$Transset(X) \implies$
 $\langle a; b \rangle \cap Vfrom(X, succ(i)) \subseteq \langle a \cap Vfrom(X, i); b \cap Vfrom(X, i) \rangle$
 $\langle proof \rangle$

25.8 "Take-Lemma" Rules

lemma $QPair\text{-}Int\text{-}Vfrom\text{-}subset$:

$Transset(X) \implies$
 $\langle a; b \rangle \cap Vfrom(X, i) \subseteq \langle a \cap Vfrom(X, i); b \cap Vfrom(X, i) \rangle$
 $\langle proof \rangle$

lemmas $QPair\text{-}Int\text{-}Vset\text{-}subset\text{-}trans =$

$subset\text{-}trans$ [OF $Transset\text{-}0$ [THEN $QPair\text{-}Int\text{-}Vfrom\text{-}subset$] $QPair\text{-}mono$]

lemma $QPair\text{-}Int\text{-}Vset\text{-}subset\text{-}UN$:

$Ord(i) \implies \langle a; b \rangle \cap Vset(i) \subseteq (\bigcup j \in i. \langle a \cap Vset(j); b \cap Vset(j) \rangle)$
 $\langle proof \rangle$

end

26 Datatype and CoDatatype Definitions

theory $Datatype$

imports $Inductive Univ QUniv$

keywords $datatype codatatype :: thy\text{-}decl$

begin

$\langle ML \rangle$

end

27 Arithmetic Operators and Their Definitions

theory *Arith* **imports** *Univ* **begin**

Proofs about elementary arithmetic: addition, multiplication, etc.

definition

$pred :: i \Rightarrow i$ **where**
 $pred(y) \equiv nat-case(0, \lambda x. x, y)$

definition

$natify :: i \Rightarrow i$ **where**
 $natify \equiv Vrecursor(\lambda f a. \text{if } a = succ(pred(a)) \text{ then } succ(f \cdot pred(a)) \text{ else } 0)$

consts

$raw-add :: [i, i] \Rightarrow i$
 $raw-diff :: [i, i] \Rightarrow i$
 $raw-mult :: [i, i] \Rightarrow i$

primrec

$raw-add \ 0, n) = n$
 $raw-add \ (succ(m), n) = succ(raw-add(m, n))$

primrec

$raw-diff-0: \quad raw-diff(m, 0) = m$
 $raw-diff-succ: \quad raw-diff(m, succ(n)) =$
 $\quad nat-case(0, \lambda x. x, raw-diff(m, n))$

primrec

$raw-mult(0, n) = 0$
 $raw-mult(succ(m), n) = raw-add \ (n, raw-mult(m, n))$

definition

$add :: [i, i] \Rightarrow i$ **(infixl <#+> 65) where**
 $m \#+ n \equiv raw-add \ (natify(m), natify(n))$

definition

$diff :: [i, i] \Rightarrow i$ **(infixl <#-> 65) where**
 $m \#- n \equiv raw-diff \ (natify(m), natify(n))$

definition

$mult :: [i, i] \Rightarrow i$ **(infixl <#*> 70) where**
 $m \#* n \equiv raw-mult \ (natify(m), natify(n))$

definition

$raw-div :: [i, i] \Rightarrow i$ **where**
 $raw-div \ (m, n) \equiv$

$transrec(m, \lambda j f. \text{ if } j < n \mid n=0 \text{ then } 0 \text{ else } succ(f'(j\#-n)))$

definition

$raw-mod :: [i, i] \Rightarrow i$ **where**
 $raw-mod \ (m, n) \equiv$
 $transrec(m, \lambda j f. \text{ if } j < n \mid n=0 \text{ then } j \text{ else } f'(j\#-n))$

definition

$div :: [i, i] \Rightarrow i$ **(infixl $\langle div \rangle$ 70) where**
 $m \text{ div } n \equiv raw-div \ (natify(m), natify(n))$

definition

$mod :: [i, i] \Rightarrow i$ **(infixl $\langle mod \rangle$ 70) where**
 $m \text{ mod } n \equiv raw-mod \ (natify(m), natify(n))$

declare *rec-type* [simp]

nat-0-le [simp]

lemma *zero-lt-lemma*: $\llbracket 0 < k; k \in nat \rrbracket \Longrightarrow \exists j \in nat. k = succ(j)$
 $\langle proof \rangle$

lemmas *zero-lt-natE* = *zero-lt-lemma* [THEN *beE*]

27.1 *natify*, the Coercion to *nat*

lemma *pred-succ-eq* [simp]: $pred(succ(y)) = y$
 $\langle proof \rangle$

lemma *natify-succ*: $natify(succ(x)) = succ(natify(x))$
 $\langle proof \rangle$

lemma *natify-0* [simp]: $natify(0) = 0$
 $\langle proof \rangle$

lemma *natify-non-succ*: $\forall z. x \neq succ(z) \Longrightarrow natify(x) = 0$
 $\langle proof \rangle$

lemma *natify-in-nat* [iff, TC]: $natify(x) \in nat$
 $\langle proof \rangle$

lemma *natify-ident* [simp]: $n \in nat \Longrightarrow natify(n) = n$
 $\langle proof \rangle$

lemma *natify-eqE*: $\llbracket natify(x) = y; x \in nat \rrbracket \Longrightarrow x=y$
 $\langle proof \rangle$

lemma *natify-idem* [simp]: $\text{natify}(\text{natify}(x)) = \text{natify}(x)$
 $\langle \text{proof} \rangle$

lemma *add-natify1* [simp]: $\text{natify}(m) \# + n = m \# + n$
 $\langle \text{proof} \rangle$

lemma *add-natify2* [simp]: $m \# + \text{natify}(n) = m \# + n$
 $\langle \text{proof} \rangle$

lemma *mult-natify1* [simp]: $\text{natify}(m) \# * n = m \# * n$
 $\langle \text{proof} \rangle$

lemma *mult-natify2* [simp]: $m \# * \text{natify}(n) = m \# * n$
 $\langle \text{proof} \rangle$

lemma *diff-natify1* [simp]: $\text{natify}(m) \# - n = m \# - n$
 $\langle \text{proof} \rangle$

lemma *diff-natify2* [simp]: $m \# - \text{natify}(n) = m \# - n$
 $\langle \text{proof} \rangle$

lemma *mod-natify1* [simp]: $\text{natify}(m) \bmod n = m \bmod n$
 $\langle \text{proof} \rangle$

lemma *mod-natify2* [simp]: $m \bmod \text{natify}(n) = m \bmod n$
 $\langle \text{proof} \rangle$

lemma *div-natify1* [simp]: $\text{natify}(m) \text{ div } n = m \text{ div } n$
 $\langle \text{proof} \rangle$

lemma *div-natify2* [simp]: $m \text{ div } \text{natify}(n) = m \text{ div } n$
 $\langle \text{proof} \rangle$

27.2 Typing rules

lemma *raw-add-type*: $\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies \text{raw-add } (m, n) \in \text{nat}$

$\langle proof \rangle$

lemma *add-type* [*iff*, *TC*]: $m \# + n \in nat$
 $\langle proof \rangle$

lemma *raw-mult-type*: $\llbracket m \in nat; n \in nat \rrbracket \implies raw-mult(m, n) \in nat$
 $\langle proof \rangle$

lemma *mult-type* [*iff*, *TC*]: $m \# * n \in nat$
 $\langle proof \rangle$

lemma *raw-diff-type*: $\llbracket m \in nat; n \in nat \rrbracket \implies raw-diff(m, n) \in nat$
 $\langle proof \rangle$

lemma *diff-type* [*iff*, *TC*]: $m \# - n \in nat$
 $\langle proof \rangle$

lemma *diff-0-eq-0* [*simp*]: $0 \# - n = 0$
 $\langle proof \rangle$

lemma *diff-succ-succ* [*simp*]: $succ(m) \# - succ(n) = m \# - n$
 $\langle proof \rangle$

declare *raw-diff-succ* [*simp del*]

lemma *diff-0* [*simp*]: $m \# - 0 = natify(m)$
 $\langle proof \rangle$

lemma *diff-le-self*: $m \in nat \implies (m \# - n) \leq m$
 $\langle proof \rangle$

27.3 Addition

lemma *add-0-natify* [*simp*]: $0 \# + m = natify(m)$
 $\langle proof \rangle$

lemma *add-succ* [*simp*]: $succ(m) \# + n = succ(m \# + n)$
 $\langle proof \rangle$

lemma *add-0*: $m \in nat \implies 0 \# + m = m$
 $\langle proof \rangle$

lemma *add-assoc*: $(m \# + n) \# + k = m \# + (n \# + k)$
 $\langle proof \rangle$

lemma *add-0-right-natify* [*simp*]: $m \# + 0 = natify(m)$
 $\langle proof \rangle$

lemma *add-succ-right* [*simp*]: $m \# + succ(n) = succ(m \# + n)$
 $\langle proof \rangle$

lemma *add-0-right*: $m \in nat \implies m \# + 0 = m$
 $\langle proof \rangle$

lemma *add-commute*: $m \# + n = n \# + m$
 $\langle proof \rangle$

lemma *add-left-commute*: $m \# + (n \# + k) = n \# + (m \# + k)$
 $\langle proof \rangle$

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

lemma *raw-add-left-cancel*:
 $\llbracket raw-add(k, m) = raw-add(k, n); k \in nat \rrbracket \implies m = n$
 $\langle proof \rangle$

lemma *add-left-cancel-natify*: $k \# + m = k \# + n \implies natify(m) = natify(n)$
 $\langle proof \rangle$

lemma *add-left-cancel*:
 $\llbracket i = j; i \# + m = j \# + n; m \in nat; n \in nat \rrbracket \implies m = n$
 $\langle proof \rangle$

lemma *add-le-elim1-natify*: $k \# + m \leq k \# + n \implies natify(m) \leq natify(n)$
 $\langle proof \rangle$

lemma *add-le-elim1*: $\llbracket k \# + m \leq k \# + n; m \in nat; n \in nat \rrbracket \implies m \leq n$
 $\langle proof \rangle$

lemma *add-lt-elim1-natify*: $k \# + m < k \# + n \implies natify(m) < natify(n)$
 $\langle proof \rangle$

lemma *add-lt-elim1*: $\llbracket k \# + m < k \# + n; m \in nat; n \in nat \rrbracket \implies m < n$

$\langle proof \rangle$

lemma *zero-less-add*: $\llbracket n \in nat; m \in nat \rrbracket \implies 0 < m \# + n \longleftrightarrow (0 < m \mid 0 < n)$
 $\langle proof \rangle$

27.4 Monotonicity of Addition

lemma *add-lt-mono1*: $\llbracket i < j; j \in nat \rrbracket \implies i \# + k < j \# + k$
 $\langle proof \rangle$

strict, in second argument

lemma *add-lt-mono2*: $\llbracket i < j; j \in nat \rrbracket \implies k \# + i < k \# + j$
 $\langle proof \rangle$

A [clumsy] way of lifting $<$ monotonicity to \leq monotonicity

lemma *Ord-lt-mono-imp-le-mono*:
 assumes *lt-mono*: $\bigwedge i j. \llbracket i < j; j \in nat \rrbracket \implies f(i) < f(j)$
 and *ford*: $\bigwedge i. i \in k \implies \text{Ord}(f(i))$
 and *leij*: $i \leq j$
 and *jink*: $j \in k$
 shows $f(i) \leq f(j)$
 $\langle proof \rangle$

\leq monotonicity, 1st argument

lemma *add-le-mono1*: $\llbracket i \leq j; j \in nat \rrbracket \implies i \# + k \leq j \# + k$
 $\langle proof \rangle$

\leq monotonicity, both arguments

lemma *add-le-mono*: $\llbracket i \leq j; k \leq l; j \in nat; l \in nat \rrbracket \implies i \# + k \leq j \# + l$
 $\langle proof \rangle$

Combinations of less-than and less-than-or-equals

lemma *add-lt-le-mono*: $\llbracket i < j; k \leq l; j \in nat; l \in nat \rrbracket \implies i \# + k < j \# + l$
 $\langle proof \rangle$

lemma *add-le-lt-mono*: $\llbracket i \leq j; k < l; j \in nat; l \in nat \rrbracket \implies i \# + k < j \# + l$
 $\langle proof \rangle$

Less-than: in other words, strict in both arguments

lemma *add-lt-mono*: $\llbracket i < j; k < l; j \in nat; l \in nat \rrbracket \implies i \# + k < j \# + l$
 $\langle proof \rangle$

lemma *diff-add-inverse*: $(n \# + m) \# - n = \text{nativify}(m)$
 $\langle proof \rangle$

lemma *diff-add-inverse2*: $(m \# + n) \# - n = \text{nativify}(m)$

$\langle proof \rangle$

lemma *diff-cancel*: $(k \# + m) \# - (k \# + n) = m \# - n$
 $\langle proof \rangle$

lemma *diff-cancel2*: $(m \# + k) \# - (n \# + k) = m \# - n$
 $\langle proof \rangle$

lemma *diff-add-0*: $n \# - (n \# + m) = 0$
 $\langle proof \rangle$

lemma *pred-0 [simp]*: $pred(0) = 0$
 $\langle proof \rangle$

lemma *eq-succ-imp-eq-m1*: $\llbracket i = succ(j); i \in nat \rrbracket \implies j = i \# - 1 \wedge j \in nat$
 $\langle proof \rangle$

lemma *pred-Un-distrib*:
 $\llbracket i \in nat; j \in nat \rrbracket \implies pred(i \cup j) = pred(i) \cup pred(j)$
 $\langle proof \rangle$

lemma *pred-type [TC,simp]*:
 $i \in nat \implies pred(i) \in nat$
 $\langle proof \rangle$

lemma *nat-diff-pred*: $\llbracket i \in nat; j \in nat \rrbracket \implies i \# - succ(j) = pred(i \# - j)$
 $\langle proof \rangle$

lemma *diff-succ-eq-pred*: $i \# - succ(j) = pred(i \# - j)$
 $\langle proof \rangle$

lemma *nat-diff-Un-distrib*:
 $\llbracket i \in nat; j \in nat; k \in nat \rrbracket \implies (i \cup j) \# - k = (i \# - k) \cup (j \# - k)$
 $\langle proof \rangle$

lemma *diff-Un-distrib*:
 $\llbracket i \in nat; j \in nat \rrbracket \implies (i \cup j) \# - k = (i \# - k) \cup (j \# - k)$
 $\langle proof \rangle$

We actually prove $i \# - j \# - k = i \# - (j \# + k)$

lemma *diff-diff-left [simplified]*:
 $nativify(i) \# - natify(j) \# - k = natify(i) \# - (natify(j) \# + k)$
 $\langle proof \rangle$

lemma *eq-add-iff*: $(u \# + m = u \# + n) \longleftrightarrow (0 \# + m = natify(n))$
 $\langle proof \rangle$

lemma *less-add-iff*: $(u \# + m < u \# + n) \longleftrightarrow (0 \# + m < \text{natify}(n))$
 $\langle \text{proof} \rangle$

lemma *diff-add-eq*: $((u \# + m) \# - (u \# + n)) = ((0 \# + m) \# - n)$
 $\langle \text{proof} \rangle$

lemma *eq-cong2*: $u = u' \implies (t \equiv u) \equiv (t \equiv u')$
 $\langle \text{proof} \rangle$

lemma *iff-cong2*: $u \longleftrightarrow u' \implies (t \equiv u) \equiv (t \equiv u')$
 $\langle \text{proof} \rangle$

27.5 Multiplication

lemma *mult-0* [*simp*]: $0 \# * m = 0$
 $\langle \text{proof} \rangle$

lemma *mult-succ* [*simp*]: $\text{succ}(m) \# * n = n \# + (m \# * n)$
 $\langle \text{proof} \rangle$

lemma *mult-0-right* [*simp*]: $m \# * 0 = 0$
 $\langle \text{proof} \rangle$

lemma *mult-succ-right* [*simp*]: $m \# * \text{succ}(n) = m \# + (m \# * n)$
 $\langle \text{proof} \rangle$

lemma *mult-1-natify* [*simp*]: $1 \# * n = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *mult-1-right-natify* [*simp*]: $n \# * 1 = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *mult-1*: $n \in \text{nat} \implies 1 \# * n = n$
 $\langle \text{proof} \rangle$

lemma *mult-1-right*: $n \in \text{nat} \implies n \# * 1 = n$
 $\langle \text{proof} \rangle$

lemma *mult-commute*: $m \# * n = n \# * m$
 $\langle \text{proof} \rangle$

lemma *add-mult-distrib*: $(m \# + n) \# * k = (m \# * k) \# + (n \# * k)$
 $\langle \text{proof} \rangle$

lemma *add-mult-distrib-left*: $k \#* (m \# + n) = (k \#* m) \# + (k \#* n)$
 $\langle proof \rangle$

lemma *mult-assoc*: $(m \#* n) \#* k = m \#* (n \#* k)$
 $\langle proof \rangle$

lemma *mult-left-commute*: $m \#* (n \#* k) = n \#* (m \#* k)$
 $\langle proof \rangle$

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

lemma *lt-succ-eq-0-disj*:
 $\llbracket m \in nat; n \in nat \rrbracket \implies (m < succ(n)) \longleftrightarrow (m = 0 \mid (\exists j \in nat. m = succ(j) \wedge j < n))$
 $\langle proof \rangle$

lemma *less-diff-conv* [rule-format]:
 $\llbracket j \in nat; k \in nat \rrbracket \implies \forall i \in nat. (i < j \# - k) \longleftrightarrow (i \# + k < j)$
 $\langle proof \rangle$

lemmas *nat-typechecks = rec-type nat-0I nat-1I nat-succI Ord-nat*

end

28 Arithmetic with simplification

theory *ArithSimp*
imports *Arith*
begin

28.1 Arithmetic simplification

$\langle ML \rangle$

28.1.1 Examples

lemma $x \# + y = x \# + z$ $\langle proof \rangle$
lemma $y \# + x = x \# + z$ $\langle proof \rangle$
lemma $x \# + y \# + z = x \# + z$ $\langle proof \rangle$
lemma $y \# + (z \# + x) = z \# + x$ $\langle proof \rangle$
lemma $x \# + y \# + z = (z \# + y) \# + (x \# + w)$ $\langle proof \rangle$
lemma $x \#* y \# + z = (z \# + y) \# + (y \#* x \# + w)$ $\langle proof \rangle$

lemma $x \# + succ(y) = x \# + z$ $\langle proof \rangle$

lemma $x \# + \text{succ}(y) = \text{succ}(z \# + x) \langle \text{proof} \rangle$
lemma $\text{succ}(x) \# + \text{succ}(y) \# + z = \text{succ}(z \# + y) \# + \text{succ}(x \# + w) \langle \text{proof} \rangle$

lemma $(x \# + y) \# - (x \# + z) = w \langle \text{proof} \rangle$
lemma $(y \# + x) \# - (x \# + z) = dd \langle \text{proof} \rangle$
lemma $(x \# + y \# + z) \# - (x \# + z) = dd \langle \text{proof} \rangle$
lemma $(y \# + (z \# + x)) \# - (z \# + x) = dd \langle \text{proof} \rangle$
lemma $(x \# + y \# + z) \# - ((z \# + y) \# + (x \# + w)) = dd \langle \text{proof} \rangle$
lemma $(x \# * y \# + z) \# - ((z \# + y) \# + (y \# * x \# + w)) = dd \langle \text{proof} \rangle$

lemma $(x \# + \text{succ}(y)) \# - (x \# + z) = dd \langle \text{proof} \rangle$

lemma $x \# * y^2 \# + y \# * x^2 = y \# * x^2 \# + x \# * y^2 \langle \text{proof} \rangle$

lemma $(x \# + \text{succ}(y)) \# - (\text{succ}(z \# + x)) = dd \langle \text{proof} \rangle$
lemma $(\text{succ}(x) \# + \text{succ}(y) \# + z) \# - (\text{succ}(z \# + y) \# + \text{succ}(x \# + w)) = dd \langle \text{proof} \rangle$

lemma $x : \text{nat} ==> x \# + y = x \langle \text{proof} \rangle$
lemma $x : \text{nat} --> x \# + y = x \langle \text{proof} \rangle$
lemma $x : \text{nat} ==> x \# + y < x \langle \text{proof} \rangle$
lemma $x : \text{nat} ==> x < y \# + x \langle \text{proof} \rangle$
lemma $x : \text{nat} ==> x \leq \text{succ}(x) \langle \text{proof} \rangle$

lemma $x \# + y = x \langle \text{proof} \rangle$

lemma $x \# + y < x \# + z \langle \text{proof} \rangle$
lemma $y \# + x < x \# + z \langle \text{proof} \rangle$
lemma $x \# + y \# + z < x \# + z \langle \text{proof} \rangle$
lemma $y \# + z \# + x < x \# + z \langle \text{proof} \rangle$
lemma $y \# + (z \# + x) < z \# + x \langle \text{proof} \rangle$
lemma $x \# + y \# + z < (z \# + y) \# + (x \# + w) \langle \text{proof} \rangle$
lemma $x \# * y \# + z < (z \# + y) \# + (y \# * x \# + w) \langle \text{proof} \rangle$

lemma $x \# + \text{succ}(y) < x \# + z \langle \text{proof} \rangle$
lemma $x \# + \text{succ}(y) < \text{succ}(z \# + x) \langle \text{proof} \rangle$
lemma $\text{succ}(x) \# + \text{succ}(y) \# + z < \text{succ}(z \# + y) \# + \text{succ}(x \# + w) \langle \text{proof} \rangle$

lemma $x \# + \text{succ}(y) \leq \text{succ}(z \# + x) \langle \text{proof} \rangle$

28.2 Difference

lemma *diff-self-eq-0* [simp]: $m \# - m = 0 \langle \text{proof} \rangle$

lemma *add-diff-inverse*: $\llbracket n \leq m; m:\text{nat} \rrbracket \implies n \# + (m \# - n) = m$
 $\langle \text{proof} \rangle$

lemma *add-diff-inverse2*: $\llbracket n \leq m; m:\text{nat} \rrbracket \implies (m \# - n) \# + n = m$
 $\langle \text{proof} \rangle$

lemma *diff-succ*: $\llbracket n \leq m; m:\text{nat} \rrbracket \implies \text{succ}(m) \# - n = \text{succ}(m \# - n)$
 $\langle \text{proof} \rangle$

lemma *zero-less-diff* [*simp*]:
 $\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies 0 < (n \# - m) \iff m < n$
 $\langle \text{proof} \rangle$

lemma *diff-mult-distrib*: $(m \# - n) \# * k = (m \# * k) \# - (n \# * k)$
 $\langle \text{proof} \rangle$

lemma *diff-mult-distrib2*: $k \# * (m \# - n) = (k \# * m) \# - (k \# * n)$
 $\langle \text{proof} \rangle$

28.3 Remainder

lemma *div-termination*: $\llbracket 0 < n; n \leq m; m:\text{nat} \rrbracket \implies m \# - n < m$
 $\langle \text{proof} \rangle$

lemmas *div-rls* =
nat-typechecks *Ord-transrec-type* *apply-funtype*
div-termination [*THEN ltD*]
nat-into-Ord *not-lt-iff-le* [*THEN iffD1*]

lemma *raw-mod-type*: $\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies \text{raw-mod}(m, n) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *mod-type* [*TC,iff*]: $m \text{ mod } n \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-DIV*: $a \text{ div } 0 = 0$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-MOD*: $a \text{ mod } 0 = \text{natify}(a)$

$\langle proof \rangle$

lemma *raw-mod-less*: $m < n \implies \text{raw-mod } (m, n) = m$
 $\langle proof \rangle$

lemma *mod-less* [*simp*]: $\llbracket m < n; n \in \text{nat} \rrbracket \implies m \text{ mod } n = m$
 $\langle proof \rangle$

lemma *raw-mod-geq*:
 $\llbracket 0 < n; n \leq m; m : \text{nat} \rrbracket \implies \text{raw-mod } (m, n) = \text{raw-mod } (m \# -n, n)$
 $\langle proof \rangle$

lemma *mod-geq*: $\llbracket n \leq m; m : \text{nat} \rrbracket \implies m \text{ mod } n = (m \# -n) \text{ mod } n$
 $\langle proof \rangle$

28.4 Division

lemma *raw-div-type*: $\llbracket m : \text{nat}; n : \text{nat} \rrbracket \implies \text{raw-div } (m, n) \in \text{nat}$
 $\langle proof \rangle$

lemma *div-type* [*TC, iff*]: $m \text{ div } n \in \text{nat}$
 $\langle proof \rangle$

lemma *raw-div-less*: $m < n \implies \text{raw-div } (m, n) = 0$
 $\langle proof \rangle$

lemma *div-less* [*simp*]: $\llbracket m < n; n \in \text{nat} \rrbracket \implies m \text{ div } n = 0$
 $\langle proof \rangle$

lemma *raw-div-geq*: $\llbracket 0 < n; n \leq m; m : \text{nat} \rrbracket \implies \text{raw-div}(m, n) = \text{succ}(\text{raw-div}(m \# -n, n))$
 $\langle proof \rangle$

lemma *div-geq* [*simp*]:
 $\llbracket 0 < n; n \leq m; m : \text{nat} \rrbracket \implies m \text{ div } n = \text{succ } ((m \# -n) \text{ div } n)$
 $\langle proof \rangle$

declare *div-less* [*simp*] *div-geq* [*simp*]

lemma *mod-div-lemma*: $\llbracket m : \text{nat}; n : \text{nat} \rrbracket \implies (m \text{ div } n) \# * n \# + m \text{ mod } n = m$
 $\langle proof \rangle$

lemma *mod-div-equality-natify*: $(m \text{ div } n) \# * n \# + m \text{ mod } n = \text{natify}(m)$
 $\langle proof \rangle$

lemma *mod-div-equality*: $m : \text{nat} \implies (m \text{ div } n) \# * n \# + m \text{ mod } n = m$

$\langle proof \rangle$

28.5 Further Facts about Remainder

(mainly for mutilated chess board)

lemma *mod-succ-lemma*:

$\llbracket 0 < n; \ m : nat; \ n : nat \rrbracket$
 $\implies succ(m) \bmod n = (if\ succ(m \bmod n) = n\ then\ 0\ else\ succ(m \bmod n))$
 $\langle proof \rangle$

lemma *mod-succ*:

$n : nat \implies succ(m) \bmod n = (if\ succ(m \bmod n) = n\ then\ 0\ else\ succ(m \bmod n))$
 $\langle proof \rangle$

lemma *mod-less-divisor*: $\llbracket 0 < n; \ n : nat \rrbracket \implies m \bmod n < n$
 $\langle proof \rangle$

lemma *mod-1-eq* [simp]: $m \bmod 1 = 0$
 $\langle proof \rangle$

lemma *mod2-cases*: $b < 2 \implies k \bmod 2 = b \mid k \bmod 2 = (if\ b=1\ then\ 0\ else\ 1)$
 $\langle proof \rangle$

lemma *mod2-succ-succ* [simp]: $succ(succ(m)) \bmod 2 = m \bmod 2$
 $\langle proof \rangle$

lemma *mod2-add-more* [simp]: $(m\#\#+m\#+n) \bmod 2 = n \bmod 2$
 $\langle proof \rangle$

lemma *mod2-add-self* [simp]: $(m\#+m) \bmod 2 = 0$
 $\langle proof \rangle$

28.6 Additional theorems about \leq

lemma *add-le-self*: $m : nat \implies m \leq (m\#\#+n)$
 $\langle proof \rangle$

lemma *add-le-self2*: $m : nat \implies m \leq (n\#\#+m)$
 $\langle proof \rangle$

lemma *mult-le-mono1*: $\llbracket i \leq j; \ j : nat \rrbracket \implies (i\#\#+k) \leq (j\#\#+k)$
 $\langle proof \rangle$

lemma *mult-le-mono*: $\llbracket i \leq j; \ k \leq l; \ j : nat; \ l : nat \rrbracket \implies i\#\#+k \leq j\#\#+l$
 $\langle proof \rangle$

lemma *mult-lt-mono2*: $\llbracket i < j; 0 < k; j:\text{nat}; k:\text{nat} \rrbracket \implies k \#* i < k \#* j$
 $\langle \text{proof} \rangle$

lemma *mult-lt-mono1*: $\llbracket i < j; 0 < k; j:\text{nat}; k:\text{nat} \rrbracket \implies i \#* k < j \#* k$
 $\langle \text{proof} \rangle$

lemma *add-eq-0-iff* [iff]: $m \# + n = 0 \longleftrightarrow \text{natify}(m)=0 \wedge \text{natify}(n)=0$
 $\langle \text{proof} \rangle$

lemma *zero-lt-mult-iff* [iff]: $0 < m \#* n \longleftrightarrow 0 < \text{natify}(m) \wedge 0 < \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *mult-eq-1-iff* [iff]: $m \#* n = 1 \longleftrightarrow \text{natify}(m)=1 \wedge \text{natify}(n)=1$
 $\langle \text{proof} \rangle$

lemma *mult-is-zero*: $\llbracket m:\text{nat}; n:\text{nat} \rrbracket \implies (m \#* n = 0) \longleftrightarrow (m = 0 \mid n = 0)$
 $\langle \text{proof} \rangle$

lemma *mult-is-zero-natify* [iff]:
 $(m \#* n = 0) \longleftrightarrow (\text{natify}(m) = 0 \mid \text{natify}(n) = 0)$
 $\langle \text{proof} \rangle$

28.7 Cancellation Laws for Common Factors in Comparisons

lemma *mult-less-cancel-lemma*:
 $\llbracket k:\text{nat}; m:\text{nat}; n:\text{nat} \rrbracket \implies (m \#* k < n \#* k) \longleftrightarrow (0 < k \wedge m < n)$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel2* [simp]:
 $(m \#* k < n \#* k) \longleftrightarrow (0 < \text{natify}(k) \wedge \text{natify}(m) < \text{natify}(n))$
 $\langle \text{proof} \rangle$

lemma *mult-less-cancel1* [simp]:
 $(k \#* m < k \#* n) \longleftrightarrow (0 < \text{natify}(k) \wedge \text{natify}(m) < \text{natify}(n))$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel2* [simp]: $(m \#* k \leq n \#* k) \longleftrightarrow (0 < \text{natify}(k) \longrightarrow \text{natify}(m) \leq \text{natify}(n))$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel1* [simp]: $(k \#* m \leq k \#* n) \longleftrightarrow (0 < \text{natify}(k) \longrightarrow \text{natify}(m) \leq \text{natify}(n))$
 $\langle \text{proof} \rangle$

lemma *mult-le-cancel-le1*: $k \in \text{nat} \implies k \#* m \leq k \longleftrightarrow (0 < k \longrightarrow \text{natify}(m) \leq 1)$
 $\langle \text{proof} \rangle$

lemma *Ord-eq-iff-le*: $\llbracket \text{Ord}(m); \text{Ord}(n) \rrbracket \implies m=n \longleftrightarrow (m \leq n \wedge n \leq m)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel2-lemma*:
 $\llbracket k: \text{nat}; m: \text{nat}; n: \text{nat} \rrbracket \implies (m \# * k = n \# * k) \longleftrightarrow (m=n \mid k=0)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel2* [*simp*]:
 $(m \# * k = n \# * k) \longleftrightarrow (\text{natify}(m) = \text{natify}(n) \mid \text{natify}(k) = 0)$
 $\langle \text{proof} \rangle$

lemma *mult-cancel1* [*simp*]:
 $(k \# * m = k \# * n) \longleftrightarrow (\text{natify}(m) = \text{natify}(n) \mid \text{natify}(k) = 0)$
 $\langle \text{proof} \rangle$

lemma *div-cancel-raw*:
 $\llbracket 0 < n; 0 < k; k: \text{nat}; m: \text{nat}; n: \text{nat} \rrbracket \implies (k \# * m) \text{ div } (k \# * n) = m \text{ div } n$
 $\langle \text{proof} \rangle$

lemma *div-cancel*:
 $\llbracket 0 < \text{natify}(n); 0 < \text{natify}(k) \rrbracket \implies (k \# * m) \text{ div } (k \# * n) = m \text{ div } n$
 $\langle \text{proof} \rangle$

28.8 More Lemmas about Remainder

lemma *mult-mod-distrib-raw*:
 $\llbracket k: \text{nat}; m: \text{nat}; n: \text{nat} \rrbracket \implies (k \# * m) \text{ mod } (k \# * n) = k \# * (m \text{ mod } n)$
 $\langle \text{proof} \rangle$

lemma *mod-mult-distrib2*: $k \# * (m \text{ mod } n) = (k \# * m) \text{ mod } (k \# * n)$
 $\langle \text{proof} \rangle$

lemma *mult-mod-distrib*: $(m \text{ mod } n) \# * k = (m \# * k) \text{ mod } (n \# * k)$
 $\langle \text{proof} \rangle$

lemma *mod-add-self2-raw*: $n \in \text{nat} \implies (m \# + n) \text{ mod } n = m \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-add-self2* [*simp*]: $(m \# + n) \text{ mod } n = m \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-add-self1* [*simp*]: $(n \# + m) \text{ mod } n = m \text{ mod } n$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self1-raw*: $k \in \text{nat} \implies (m \# + k \# * n) \text{ mod } n = m \text{ mod } n$

$\langle \text{proof} \rangle$

lemma *mod-mult-self1* [simp]: $(m \# + k \# * n) \bmod n = m \bmod n$
 $\langle \text{proof} \rangle$

lemma *mod-mult-self2* [simp]: $(m \# + n \# * k) \bmod n = m \bmod n$
 $\langle \text{proof} \rangle$

lemma *mult-eq-self-implies-10*: $m = m \# * n \implies \text{natify}(n) = 1 \mid m = 0$
 $\langle \text{proof} \rangle$

lemma *less-imp-succ-add* [rule-format]:
 $\llbracket m < n; n: \text{nat} \rrbracket \implies \exists k \in \text{nat}. n = \text{succ}(m \# + k)$
 $\langle \text{proof} \rangle$

lemma *less-iff-succ-add*:
 $\llbracket m: \text{nat}; n: \text{nat} \rrbracket \implies (m < n) \longleftrightarrow (\exists k \in \text{nat}. n = \text{succ}(m \# + k))$
 $\langle \text{proof} \rangle$

lemma *add-lt-elim2*:
 $\llbracket a \# + d = b \# + c; a < b; b \in \text{nat}; c \in \text{nat}; d \in \text{nat} \rrbracket \implies c < d$
 $\langle \text{proof} \rangle$

lemma *add-le-elim2*:
 $\llbracket a \# + d = b \# + c; a \leq b; b \in \text{nat}; c \in \text{nat}; d \in \text{nat} \rrbracket \implies c \leq d$
 $\langle \text{proof} \rangle$

28.8.1 More Lemmas About Difference

lemma *diff-is-0-lemma*:
 $\llbracket m: \text{nat}; n: \text{nat} \rrbracket \implies m \# - n = 0 \longleftrightarrow m \leq n$
 $\langle \text{proof} \rangle$

lemma *diff-is-0-iff*: $m \# - n = 0 \longleftrightarrow \text{natify}(m) \leq \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *nat-lt-imp-diff-eq-0*:
 $\llbracket a: \text{nat}; b: \text{nat}; a < b \rrbracket \implies a \# - b = 0$
 $\langle \text{proof} \rangle$

lemma *raw-nat-diff-split*:
 $\llbracket a: \text{nat}; b: \text{nat} \rrbracket \implies$
 $(P(a \# - b)) \longleftrightarrow ((a < b \longrightarrow P(0)) \wedge (\forall d \in \text{nat}. a = b \# + d \longrightarrow P(d)))$
 $\langle \text{proof} \rangle$

lemma *nat-diff-split*:
 $(P(a \# - b)) \longleftrightarrow$
 $(\text{natify}(a) < \text{natify}(b) \longrightarrow P(0)) \wedge (\forall d \in \text{nat}. \text{natify}(a) = b \# + d \longrightarrow P(d))$

$\langle proof \rangle$

Difference and less-than

lemma *diff-lt-imp-lt*: $\llbracket (k\#-i) < (k\#-j); i \in nat; j \in nat; k \in nat \rrbracket \implies j < i$
 $\langle proof \rangle$

lemma *lt-imp-diff-lt*: $\llbracket j < i; i \leq k; k \in nat \rrbracket \implies (k\#-i) < (k\#-j)$
 $\langle proof \rangle$

lemma *diff-lt-iff-lt*: $\llbracket i \leq k; j \in nat; k \in nat \rrbracket \implies (k\#-i) < (k\#-j) \longleftrightarrow j < i$
 $\langle proof \rangle$

end

29 Lists in Zermelo-Fraenkel Set Theory

theory *List* **imports** *Datatype ArithSimp* **begin**

consts

list :: $i \Rightarrow i$

datatype

list(*A*) = *Nil* | *Cons* (*a* ∈ *A*, *l* ∈ *list*(*A*))

notation *Nil* ($\langle [] \rangle$)

syntax

-List :: $is \Rightarrow i$ ($\langle \langle \text{indent}=1 \text{ notation}=\langle \text{mixfix list enumeration} \rangle \rangle [-] \rangle$)

translations

$[x, xs] == CONST\ Cons(x, [xs])$
 $[x] == CONST\ Cons(x, [])$

consts

length :: $i \Rightarrow i$

hd :: $i \Rightarrow i$

tl :: $i \Rightarrow i$

primrec

length($[]$) = 0

length(*Cons*(*a*,*l*)) = *succ*(*length*(*l*))

primrec

hd($[]$) = 0

hd(*Cons*(*a*,*l*)) = *a*

primrec

tl($[]$) = $[]$

tl(*Cons*(*a*,*l*)) = *l*

consts

$map \quad :: [i \Rightarrow i, i] \Rightarrow i$
 $set-of-list \quad :: i \Rightarrow i$
 $app \quad :: [i, i] \Rightarrow i$ (**infixr** <@> 60)

primrec

$map(f, []) = []$
 $map(f, Cons(a, l)) = Cons(f(a), map(f, l))$

primrec

$set-of-list([]) = 0$
 $set-of-list(Cons(a, l)) = cons(a, set-of-list(l))$

primrec

$app-Nil: [] @ ys = ys$
 $app-Cons: (Cons(a, l)) @ ys = Cons(a, l @ ys)$

consts

$rev \quad :: i \Rightarrow i$
 $flat \quad :: i \Rightarrow i$
 $list-add \quad :: i \Rightarrow i$

primrec

$rev([]) = []$
 $rev(Cons(a, l)) = rev(l) @ [a]$

primrec

$flat([]) = []$
 $flat(Cons(l, ls)) = l @ flat(ls)$

primrec

$list-add([]) = 0$
 $list-add(Cons(a, l)) = a \# + list-add(l)$

consts

$drop \quad :: [i, i] \Rightarrow i$

primrec

$drop-0: \quad drop(0, l) = l$
 $drop-succ: \quad drop(succ(i), l) = tl \ (drop(i, l))$

definition

$take :: [i, i] \Rightarrow i$ **where**
 $take(n, as) \equiv list-rec(\lambda n \in nat. [],$
 $\lambda a \ l \ r. \lambda n \in nat. nat-case([], \lambda m. Cons(a, r'm), n), as)'n$

definition

$nth :: [i, i] \Rightarrow i$ **where**
 — returns the (n+1)th element of a list, or 0 if the list is too short.
 $nth(n, as) \equiv list-rec(\lambda n \in nat. 0,$
 $\lambda a \ l \ r. \lambda n \in nat. nat-case(a, \lambda m. r'm, n), as)'n$

definition

$list-update :: [i, i, i] \Rightarrow i$ **where**
 $list-update(xs, i, v) \equiv list-rec(\lambda n \in nat. Nil,$
 $\lambda u \ us \ vs. \lambda n \in nat. nat-case(Cons(v, us), \lambda m. Cons(u, vs'm), n), xs)'i$

consts

$filter :: [i \Rightarrow o, i] \Rightarrow i$
 $upt :: [i, i] \Rightarrow i$

primrec

$filter(P, Nil) = Nil$
 $filter(P, Cons(x, xs)) =$
 $(if P(x) then Cons(x, filter(P, xs)) else filter(P, xs))$

primrec

$upt(i, 0) = Nil$
 $upt(i, succ(j)) = (if i \leq j then upt(i, j)@[j] else Nil)$

definition

$min :: [i, i] \Rightarrow i$ **where**
 $min(x, y) \equiv (if x \leq y then x else y)$

definition

$max :: [i, i] \Rightarrow i$ **where**
 $max(x, y) \equiv (if x \leq y then y else x)$

declare $list.intros [simp, TC]$

inductive-cases $ConsE$: $Cons(a, l) \in list(A)$

lemma $Cons-type-iff$ $[simp]$: $Cons(a, l) \in list(A) \longleftrightarrow a \in A \wedge l \in list(A)$
 $\langle proof \rangle$

lemma $Cons-iff$: $Cons(a, l) = Cons(a', l') \longleftrightarrow a = a' \wedge l = l'$

$\langle proof \rangle$

lemma *Nil-Cons-iff*: $\neg Nil = Cons(a, l)$
 $\langle proof \rangle$

lemma *list-unfold*: $list(A) = \{0\} + (A * list(A))$
 $\langle proof \rangle$

lemma *list-mono*: $A \leq B \implies list(A) \subseteq list(B)$
 $\langle proof \rangle$

lemma *list-univ*: $list(univ(A)) \subseteq univ(A)$
 $\langle proof \rangle$

lemmas *list-subset-univ* = *subset-trans* [OF *list-mono list-univ*]

lemma *list-into-univ*: $\llbracket l \in list(A); A \subseteq univ(B) \rrbracket \implies l \in univ(B)$
 $\langle proof \rangle$

lemma *list-case-type*:
 $\llbracket l \in list(A);$
 $c \in C(Nil);$
 $\bigwedge x y. \llbracket x \in A; y \in list(A) \rrbracket \implies h(x, y) \in C(Cons(x, y))$
 $\rrbracket \implies list-case(c, h, l) \in C(l)$
 $\langle proof \rangle$

lemma *list-0-triv*: $list(0) = \{Nil\}$
 $\langle proof \rangle$

lemma *tl-type*: $l \in list(A) \implies tl(l) \in list(A)$
 $\langle proof \rangle$

lemma *drop-Nil* [*simp*]: $i \in nat \implies drop(i, Nil) = Nil$
 $\langle proof \rangle$

lemma *drop-succ-Cons* [*simp*]: $i \in nat \implies drop(succ(i), Cons(a, l)) = drop(i, l)$
 $\langle proof \rangle$

lemma *drop-type* [*simp, TC*]: $\llbracket i \in nat; l \in list(A) \rrbracket \implies drop(i, l) \in list(A)$

$\langle proof \rangle$

declare *drop-succ* [*simp del*]

lemma *list-rec-type* [*TC*]:

$\llbracket l \in list(A);$
 $c \in C(Nil);$
 $\bigwedge x y r. \llbracket x \in A; y \in list(A); r \in C(y) \rrbracket \implies h(x,y,r) \in C(Cons(x,y))$
 $\rrbracket \implies list-rec(c,h,l) \in C(l)$
 $\langle proof \rangle$

lemma *map-type* [*TC*]:

$\llbracket l \in list(A); \bigwedge x. x \in A \implies h(x) \in B \rrbracket \implies map(h,l) \in list(B)$
 $\langle proof \rangle$

lemma *map-type2* [*TC*]: $l \in list(A) \implies map(h,l) \in list(\{h(u). u \in A\})$

$\langle proof \rangle$

lemma *length-type* [*TC*]: $l \in list(A) \implies length(l) \in nat$

$\langle proof \rangle$

lemma *lt-length-in-nat*:

$\llbracket x < length(xs); xs \in list(A) \rrbracket \implies x \in nat$
 $\langle proof \rangle$

lemma *app-type* [*TC*]: $\llbracket xs: list(A); ys: list(A) \rrbracket \implies xs@ys \in list(A)$

$\langle proof \rangle$

lemma *rev-type* [*TC*]: $xs: list(A) \implies rev(xs) \in list(A)$

$\langle proof \rangle$

lemma *flat-type* [*TC*]: $ls: list(list(A)) \implies flat(ls) \in list(A)$

$\langle proof \rangle$

lemma *set-of-list-type* [TC]: $l \in \text{list}(A) \implies \text{set-of-list}(l) \in \text{Pow}(A)$
 $\langle \text{proof} \rangle$

lemma *set-of-list-append*:
 $xs: \text{list}(A) \implies \text{set-of-list}(xs @ ys) = \text{set-of-list}(xs) \cup \text{set-of-list}(ys)$
 $\langle \text{proof} \rangle$

lemma *list-add-type* [TC]: $xs: \text{list}(\text{nat}) \implies \text{list-add}(xs) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *map-ident* [simp]: $l \in \text{list}(A) \implies \text{map}(\lambda u. u, l) = l$
 $\langle \text{proof} \rangle$

lemma *map-compose*: $l \in \text{list}(A) \implies \text{map}(h, \text{map}(j, l)) = \text{map}(\lambda u. h(j(u)), l)$
 $\langle \text{proof} \rangle$

lemma *map-app-distrib*: $xs: \text{list}(A) \implies \text{map}(h, xs @ ys) = \text{map}(h, xs) @ \text{map}(h, ys)$
 $\langle \text{proof} \rangle$

lemma *map-flat*: $ls: \text{list}(\text{list}(A)) \implies \text{map}(h, \text{flat}(ls)) = \text{flat}(\text{map}(\text{map}(h), ls))$
 $\langle \text{proof} \rangle$

lemma *list-rec-map*:
 $l \in \text{list}(A) \implies$
 $\text{list-rec}(c, d, \text{map}(h, l)) =$
 $\text{list-rec}(c, \lambda x xs r. d(h(x), \text{map}(h, xs), r), l)$
 $\langle \text{proof} \rangle$

lemmas *list-CollectD* = *Collect-subset* [THEN *list-mono*, THEN *subsetD*]

lemma *map-list-Collect*: $l \in \text{list}(\{x \in A. h(x)=j(x)\}) \implies \text{map}(h, l) = \text{map}(j, l)$
 $\langle \text{proof} \rangle$

lemma *length-map* [simp]: $xs: \text{list}(A) \implies \text{length}(\text{map}(h, xs)) = \text{length}(xs)$
 $\langle \text{proof} \rangle$

lemma *length-app* [*simp*]:
 $\llbracket xs: list(A); ys: list(A) \rrbracket$
 $\implies length(xs@ys) = length(xs) \# + length(ys)$
 $\langle proof \rangle$

lemma *length-rev* [*simp*]: $xs: list(A) \implies length(rev(xs)) = length(xs)$
 $\langle proof \rangle$

lemma *length-flat*:
 $ls: list(list(A)) \implies length(flat(ls)) = list-add(map(length,ls))$
 $\langle proof \rangle$

lemma *drop-length-Cons* [*rule-format*]:
 $xs: list(A) \implies$
 $\forall x. \exists z zs. drop(length(xs), Cons(x,xs)) = Cons(z,zs)$
 $\langle proof \rangle$

lemma *drop-length* [*rule-format*]:
 $l \in list(A) \implies \forall i \in length(l). (\exists z zs. drop(i,l) = Cons(z,zs))$
 $\langle proof \rangle$

lemma *app-right-Nil* [*simp*]: $xs: list(A) \implies xs@Nil = xs$
 $\langle proof \rangle$

lemma *app-assoc*: $xs: list(A) \implies (xs@ys)@zs = xs@(ys@zs)$
 $\langle proof \rangle$

lemma *flat-app-distrib*: $ls: list(list(A)) \implies flat(ls@ms) = flat(ls)@flat(ms)$
 $\langle proof \rangle$

lemma *rev-map-distrib*: $l \in list(A) \implies rev(map(h,l)) = map(h,rev(l))$
 $\langle proof \rangle$

lemma *rev-app-distrib*:
 $\llbracket xs: list(A); ys: list(A) \rrbracket \implies rev(xs@ys) = rev(ys)@rev(xs)$
 $\langle proof \rangle$

lemma *rev-rev-ident* [*simp*]: $l \in list(A) \implies rev(rev(l)) = l$
 $\langle proof \rangle$

lemma *rev-flat*: $ls: list(list(A)) \implies rev(flat(ls)) = flat(map(rev, rev(ls)))$
 $\langle proof \rangle$

lemma *list-add-app*:
 $\llbracket xs: list(nat); \quad ys: list(nat) \rrbracket$
 $\implies list-add(xs @ ys) = list-add(ys) \# + list-add(xs)$
 $\langle proof \rangle$

lemma *list-add-rev*: $l \in list(nat) \implies list-add(rev(l)) = list-add(l)$
 $\langle proof \rangle$

lemma *list-add-flat*:
 $ls: list(list(nat)) \implies list-add(flat(ls)) = list-add(map(list-add, ls))$
 $\langle proof \rangle$

lemma *list-append-induct* [*case-names Nil snoc, consumes 1*]:
 $\llbracket l \in list(A);$
 $\quad P(Nil);$
 $\quad \bigwedge x y. \llbracket x \in A; \quad y \in list(A); \quad P(y) \rrbracket \implies P(y @ [x])$
 $\rrbracket \implies P(l)$
 $\langle proof \rangle$

lemma *list-complete-induct-lemma* [*rule-format*]:
assumes *ih*:
 $\bigwedge l. \llbracket l \in list(A);$
 $\quad \forall l' \in list(A). \text{length}(l') < \text{length}(l) \longrightarrow P(l') \rrbracket$
 $\implies P(l)$
shows $n \in nat \implies \forall l \in list(A). \text{length}(l) < n \longrightarrow P(l)$
 $\langle proof \rangle$

theorem *list-complete-induct*:
 $\llbracket l \in list(A);$
 $\quad \bigwedge l. \llbracket l \in list(A);$
 $\quad \quad \forall l' \in list(A). \text{length}(l') < \text{length}(l) \longrightarrow P(l') \rrbracket$
 $\implies P(l)$
 $\langle proof \rangle$

lemma *min-sym*: $\llbracket i \in nat; \quad j \in nat \rrbracket \implies min(i, j) = min(j, i)$

$\langle proof \rangle$

lemma *min-type* [*simp*, *TC*]: $\llbracket i \in nat; j \in nat \rrbracket \implies min(i,j):nat$
 $\langle proof \rangle$

lemma *min-0* [*simp*]: $i \in nat \implies min(0,i) = 0$
 $\langle proof \rangle$

lemma *min-02* [*simp*]: $i \in nat \implies min(i, 0) = 0$
 $\langle proof \rangle$

lemma *lt-min-iff*: $\llbracket i \in nat; j \in nat; k \in nat \rrbracket \implies i < min(j,k) \longleftrightarrow i < j \wedge i < k$
 $\langle proof \rangle$

lemma *min-succ-succ* [*simp*]:
 $\llbracket i \in nat; j \in nat \rrbracket \implies min(succ(i), succ(j)) = succ(min(i, j))$
 $\langle proof \rangle$

lemma *filter-append* [*simp*]:
 $xs:list(A) \implies filter(P, xs@ys) = filter(P, xs) @ filter(P, ys)$
 $\langle proof \rangle$

lemma *filter-type* [*simp*, *TC*]: $xs:list(A) \implies filter(P, xs):list(A)$
 $\langle proof \rangle$

lemma *length-filter*: $xs:list(A) \implies length(filter(P, xs)) \leq length(xs)$
 $\langle proof \rangle$

lemma *filter-is-subset*: $xs:list(A) \implies set-of-list(filter(P,xs)) \subseteq set-of-list(xs)$
 $\langle proof \rangle$

lemma *filter-False* [*simp*]: $xs:list(A) \implies filter(\lambda p. False, xs) = Nil$
 $\langle proof \rangle$

lemma *filter-True* [*simp*]: $xs:list(A) \implies filter(\lambda p. True, xs) = xs$
 $\langle proof \rangle$

lemma *length-is-0-iff* [*simp*]: $xs:list(A) \implies length(xs)=0 \longleftrightarrow xs=Nil$
 $\langle proof \rangle$

lemma *length-is-0-iff2* [*simp*]: $xs:list(A) \implies 0 = length(xs) \longleftrightarrow xs=Nil$
 $\langle proof \rangle$

lemma *length-tl [simp]*: $xs: \text{list}(A) \implies \text{length}(\text{tl}(xs)) = \text{length}(xs) \# - 1$
 $\langle \text{proof} \rangle$

lemma *length-greater-0-iff*: $xs: \text{list}(A) \implies 0 < \text{length}(xs) \longleftrightarrow xs \neq \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *length-succ-iff*: $xs: \text{list}(A) \implies \text{length}(xs) = \text{succ}(n) \longleftrightarrow (\exists y \text{ } ys. xs = \text{Cons}(y, ys) \wedge \text{length}(ys) = n)$
 $\langle \text{proof} \rangle$

lemma *append-is-Nil-iff [simp]*:
 $xs: \text{list}(A) \implies (xs @ ys = \text{Nil}) \longleftrightarrow (xs = \text{Nil} \wedge ys = \text{Nil})$
 $\langle \text{proof} \rangle$

lemma *append-is-Nil-iff2 [simp]*:
 $xs: \text{list}(A) \implies (\text{Nil} = xs @ ys) \longleftrightarrow (xs = \text{Nil} \wedge ys = \text{Nil})$
 $\langle \text{proof} \rangle$

lemma *append-left-is-self-iff [simp]*:
 $xs: \text{list}(A) \implies (xs @ ys = xs) \longleftrightarrow (ys = \text{Nil})$
 $\langle \text{proof} \rangle$

lemma *append-left-is-self-iff2 [simp]*:
 $xs: \text{list}(A) \implies (xs = xs @ ys) \longleftrightarrow (ys = \text{Nil})$
 $\langle \text{proof} \rangle$

lemma *append-left-is-Nil-iff [rule-format]*:
 $\llbracket xs: \text{list}(A); ys: \text{list}(A); zs: \text{list}(A) \rrbracket \implies$
 $\text{length}(ys) = \text{length}(zs) \longrightarrow (xs @ ys = zs \longleftrightarrow (xs = \text{Nil} \wedge ys = zs))$
 $\langle \text{proof} \rangle$

lemma *append-left-is-Nil-iff2 [rule-format]*:
 $\llbracket xs: \text{list}(A); ys: \text{list}(A); zs: \text{list}(A) \rrbracket \implies$
 $\text{length}(ys) = \text{length}(zs) \longrightarrow (zs = ys @ xs \longleftrightarrow (xs = \text{Nil} \wedge ys = zs))$
 $\langle \text{proof} \rangle$

lemma *append-eq-append-iff [rule-format]*:
 $xs: \text{list}(A) \implies \forall ys \in \text{list}(A).$
 $\text{length}(xs) = \text{length}(ys) \longrightarrow (xs @ us = ys @ vs \longleftrightarrow (xs = ys \wedge us = vs))$
 $\langle \text{proof} \rangle$

declare *append-eq-append-iff [simp]*

lemma *append-eq-append [rule-format]*:
 $xs: \text{list}(A) \implies$
 $\forall ys \in \text{list}(A). \forall us \in \text{list}(A). \forall vs \in \text{list}(A).$

$length(us) = length(vs) \longrightarrow (xs @ us = ys @ vs) \longrightarrow (xs = ys \wedge us = vs)$
 $\langle proof \rangle$

lemma *append-eq-append-iff2* [simp]:
 $\llbracket xs: list(A); ys: list(A); us: list(A); vs: list(A); length(us) = length(vs) \rrbracket$
 $\implies xs @ us = ys @ vs \longleftrightarrow (xs = ys \wedge us = vs)$
 $\langle proof \rangle$

lemma *append-self-iff* [simp]:
 $\llbracket xs: list(A); ys: list(A); zs: list(A) \rrbracket \implies xs @ ys = xs @ zs \longleftrightarrow ys = zs$
 $\langle proof \rangle$

lemma *append-self-iff2* [simp]:
 $\llbracket xs: list(A); ys: list(A); zs: list(A) \rrbracket \implies ys @ xs = zs @ xs \longleftrightarrow ys = zs$
 $\langle proof \rangle$

lemma *append1-eq-iff* [rule-format]:
 $xs: list(A) \implies \forall ys \in list(A). xs @ [x] = ys @ [y] \longleftrightarrow (xs = ys \wedge x = y)$
 $\langle proof \rangle$
declare *append1-eq-iff* [simp]

lemma *append-right-is-self-iff* [simp]:
 $\llbracket xs: list(A); ys: list(A) \rrbracket \implies (xs @ ys = ys) \longleftrightarrow (xs = Nil)$
 $\langle proof \rangle$

lemma *append-right-is-self-iff2* [simp]:
 $\llbracket xs: list(A); ys: list(A) \rrbracket \implies (ys = xs @ ys) \longleftrightarrow (xs = Nil)$
 $\langle proof \rangle$

lemma *hd-append* [rule-format]:
 $xs: list(A) \implies xs \neq Nil \longrightarrow hd(xs @ ys) = hd(xs)$
 $\langle proof \rangle$
declare *hd-append* [simp]

lemma *tl-append* [rule-format]:
 $xs: list(A) \implies xs \neq Nil \longrightarrow tl(xs @ ys) = tl(xs) @ ys$
 $\langle proof \rangle$
declare *tl-append* [simp]

lemma *rev-is-Nil-iff* [simp]: $xs: list(A) \implies (rev(xs) = Nil \longleftrightarrow xs = Nil)$
 $\langle proof \rangle$

lemma *Nil-is-rev-iff* [simp]: $xs: list(A) \implies (Nil = rev(xs) \longleftrightarrow xs = Nil)$
 $\langle proof \rangle$

lemma *rev-is-rev-iff* [rule-format]:
 $xs: list(A) \implies \forall ys \in list(A). rev(xs) = rev(ys) \longleftrightarrow xs = ys$

$\langle proof \rangle$

declare *rev-is-rev-iff* [*simp*]

lemma *rev-list-elim* [*rule-format*]:

$xs: list(A) \implies$

$(xs = Nil \longrightarrow P) \longrightarrow (\forall ys \in list(A). \forall y \in A. xs = ys @ [y] \longrightarrow P) \longrightarrow P$

$\langle proof \rangle$

lemma *length-drop* [*rule-format*]:

$n \in nat \implies \forall xs \in list(A). length(drop(n, xs)) = length(xs) \# - n$

$\langle proof \rangle$

declare *length-drop* [*simp*]

lemma *drop-all* [*rule-format*]:

$n \in nat \implies \forall xs \in list(A). length(xs) \leq n \longrightarrow drop(n, xs) = Nil$

$\langle proof \rangle$

declare *drop-all* [*simp*]

lemma *drop-append* [*rule-format*]:

$n \in nat \implies$

$\forall xs \in list(A). drop(n, xs @ ys) = drop(n, xs) @ drop(n \# - length(xs), ys)$

$\langle proof \rangle$

lemma *drop-drop*:

$m \in nat \implies \forall xs \in list(A). \forall n \in nat. drop(n, drop(m, xs)) = drop(n \# + m, xs)$

$\langle proof \rangle$

lemma *take-0* [*simp*]: $xs: list(A) \implies take(0, xs) = Nil$

$\langle proof \rangle$

lemma *take-succ-Cons* [*simp*]:

$n \in nat \implies take(succ(n), Cons(a, xs)) = Cons(a, take(n, xs))$

$\langle proof \rangle$

lemma *take-Nil* [*simp*]: $n \in nat \implies take(n, Nil) = Nil$

$\langle proof \rangle$

lemma *take-all* [*rule-format*]:

$n \in nat \implies \forall xs \in list(A). length(xs) \leq n \longrightarrow take(n, xs) = xs$

$\langle proof \rangle$

declare *take-all* [*simp*]

lemma *take-type* [*rule-format*]:

$xs: \text{list}(A) \implies \forall n \in \text{nat}. \text{take}(n, xs): \text{list}(A)$
 $\langle \text{proof} \rangle$

declare *take-type* [*simp*, *TC*]

lemma *take-append* [*rule-format*]:

$xs: \text{list}(A) \implies$
 $\forall ys \in \text{list}(A). \forall n \in \text{nat}. \text{take}(n, xs @ ys) =$
 $\text{take}(n, xs) @ \text{take}(n \# - \text{length}(xs), ys)$

$\langle \text{proof} \rangle$

declare *take-append* [*simp*]

lemma *take-take* [*rule-format*]:

$m \in \text{nat} \implies$
 $\forall xs \in \text{list}(A). \forall n \in \text{nat}. \text{take}(n, \text{take}(m, xs)) = \text{take}(\min(n, m), xs)$
 $\langle \text{proof} \rangle$

lemma *nth-0* [*simp*]: $\text{nth}(0, \text{Cons}(a, l)) = a$
 $\langle \text{proof} \rangle$

lemma *nth-Cons* [*simp*]: $n \in \text{nat} \implies \text{nth}(\text{succ}(n), \text{Cons}(a, l)) = \text{nth}(n, l)$
 $\langle \text{proof} \rangle$

lemma *nth-empty* [*simp*]: $\text{nth}(n, \text{Nil}) = 0$
 $\langle \text{proof} \rangle$

lemma *nth-type* [*rule-format*]:

$xs: \text{list}(A) \implies \forall n. n < \text{length}(xs) \longrightarrow \text{nth}(n, xs) \in A$
 $\langle \text{proof} \rangle$

declare *nth-type* [*simp*, *TC*]

lemma *nth-eq-0* [*rule-format*]:

$xs: \text{list}(A) \implies \forall n \in \text{nat}. \text{length}(xs) \leq n \longrightarrow \text{nth}(n, xs) = 0$
 $\langle \text{proof} \rangle$

lemma *nth-append* [*rule-format*]:

$xs: \text{list}(A) \implies$
 $\forall n \in \text{nat}. \text{nth}(n, xs @ ys) = (\text{if } n < \text{length}(xs) \text{ then } \text{nth}(n, xs)$
 $\text{else } \text{nth}(n \# - \text{length}(xs), ys))$

$\langle \text{proof} \rangle$

lemma *set-of-list-conv-nth*:

$xs: \text{list}(A)$
 $\implies \text{set-of-list}(xs) = \{x \in A. \exists i \in \text{nat}. i < \text{length}(xs) \wedge x = \text{nth}(i, xs)\}$
 $\langle \text{proof} \rangle$

lemma *nth-take-lemma* [rule-format]:

$k \in \text{nat} \implies$
 $\forall xs \in \text{list}(A). (\forall ys \in \text{list}(A). k \leq \text{length}(xs) \longrightarrow k \leq \text{length}(ys) \longrightarrow$
 $(\forall i \in \text{nat}. i < k \longrightarrow \text{nth}(i, xs) = \text{nth}(i, ys)) \longrightarrow \text{take}(k, xs) = \text{take}(k, ys))$
 $\langle \text{proof} \rangle$

lemma *nth-equalityI* [rule-format]:

$\llbracket xs:\text{list}(A); ys:\text{list}(A); \text{length}(xs) = \text{length}(ys);$
 $\forall i \in \text{nat}. i < \text{length}(xs) \longrightarrow \text{nth}(i, xs) = \text{nth}(i, ys) \rrbracket$
 $\implies xs = ys$
 $\langle \text{proof} \rangle$

lemma *take-equalityI* [rule-format]:

$\llbracket xs:\text{list}(A); ys:\text{list}(A); (\forall i \in \text{nat}. \text{take}(i, xs) = \text{take}(i, ys)) \rrbracket$
 $\implies xs = ys$
 $\langle \text{proof} \rangle$

lemma *nth-drop* [rule-format]:

$n \in \text{nat} \implies \forall i \in \text{nat}. \forall xs \in \text{list}(A). \text{nth}(i, \text{drop}(n, xs)) = \text{nth}(n \# + i, xs)$
 $\langle \text{proof} \rangle$

lemma *take-succ* [rule-format]:

$xs \in \text{list}(A)$
 $\implies \forall i. i < \text{length}(xs) \longrightarrow \text{take}(\text{succ}(i), xs) = \text{take}(i, xs) @ [\text{nth}(i, xs)]$
 $\langle \text{proof} \rangle$

lemma *take-add* [rule-format]:

$\llbracket xs \in \text{list}(A); j \in \text{nat} \rrbracket$
 $\implies \forall i \in \text{nat}. \text{take}(i \# + j, xs) = \text{take}(i, xs) @ \text{take}(j, \text{drop}(i, xs))$
 $\langle \text{proof} \rangle$

lemma *length-take*:

$l \in \text{list}(A) \implies \forall n \in \text{nat}. \text{length}(\text{take}(n, l)) = \min(n, \text{length}(l))$
 $\langle \text{proof} \rangle$

29.1 The function zip

Crafty definition to eliminate a type argument

consts

zip-aux :: $[i, i] \Rightarrow i$

primrec

$\text{zip-aux}(B, []) =$
 $(\lambda ys \in \text{list}(B). \text{list-case}([], \lambda y l. [], ys))$

$\text{zip-aux}(B, \text{Cons}(x, l)) =$
 $(\lambda ys \in \text{list}(B).$

$list\text{-}case(Nil, \lambda y\ zs.\ Cons(\langle x,y \rangle, zip\text{-}aux(B,l)\text{'}zs), ys))$

definition

$zip :: [i, i] \Rightarrow i$ **where**
 $zip(xs, ys) \equiv zip\text{-}aux(set\text{-}of\text{-}list(ys),xs)\text{'}ys$

lemma *list-on-set-of-list*: $xs \in list(A) \implies xs \in list(set\text{-}of\text{-}list(xs))$
 $\langle proof \rangle$

lemma *zip-Nil* [simp]: $ys: list(A) \implies zip(Nil, ys) = Nil$
 $\langle proof \rangle$

lemma *zip-Nil2* [simp]: $xs: list(A) \implies zip(xs, Nil) = Nil$
 $\langle proof \rangle$

lemma *zip-aux-unique* [rule-format]:
 $\llbracket B \leq C; xs \in list(A) \rrbracket$
 $\implies \forall ys \in list(B). zip\text{-}aux(C,xs) \text{' } ys = zip\text{-}aux(B,xs) \text{' } ys$
 $\langle proof \rangle$

lemma *zip-Cons-Cons* [simp]:
 $\llbracket xs: list(A); ys: list(B); x \in A; y \in B \rrbracket \implies$
 $zip(Cons(x,xs), Cons(y, ys)) = Cons(\langle x,y \rangle, zip(xs, ys))$
 $\langle proof \rangle$

lemma *zip-type* [rule-format]:
 $xs: list(A) \implies \forall ys \in list(B). zip(xs, ys): list(A*B)$
 $\langle proof \rangle$

declare *zip-type* [simp, TC]

lemma *length-zip* [rule-format]:
 $xs: list(A) \implies \forall ys \in list(B). length(zip(xs,ys)) =$
 $min(length(xs), length(ys))$
 $\langle proof \rangle$

declare *length-zip* [simp]

lemma *zip-append1* [rule-format]:
 $\llbracket ys: list(A); zs: list(B) \rrbracket \implies$
 $\forall xs \in list(A). zip(xs @ ys, zs) =$
 $zip(xs, take(length(xs), zs)) @ zip(ys, drop(length(xs), zs))$
 $\langle proof \rangle$

lemma *zip-append2* [rule-format]:
 $\llbracket xs: list(A); zs: list(B) \rrbracket \implies \forall ys \in list(B). zip(xs, ys @ zs) =$
 $zip(take(length(ys), xs), ys) @ zip(drop(length(ys), xs), zs)$

$\langle proof \rangle$

lemma *zip-append* [simp]:

$\llbracket length(xs) = length(us); length(ys) = length(vs);$
 $xs:list(A); us:list(B); ys:list(A); vs:list(B) \rrbracket$
 $\implies zip(xs@us, us@vs) = zip(xs, us) @ zip(ys, vs)$
 $\langle proof \rangle$

lemma *zip-rev* [rule-format]:

$ys:list(B) \implies \forall xs \in list(A).$
 $length(xs) = length(ys) \longrightarrow zip(rev(xs), rev(ys)) = rev(zip(xs, ys))$
 $\langle proof \rangle$

declare *zip-rev* [simp]

lemma *nth-zip* [rule-format]:

$ys:list(B) \implies \forall i \in nat. \forall xs \in list(A).$
 $i < length(xs) \longrightarrow i < length(ys) \longrightarrow$
 $nth(i, zip(xs, ys)) = \langle nth(i, xs), nth(i, ys) \rangle$
 $\langle proof \rangle$

declare *nth-zip* [simp]

lemma *set-of-list-zip* [rule-format]:

$\llbracket xs:list(A); ys:list(B); i \in nat \rrbracket$
 $\implies set-of-list(zip(xs, ys)) =$
 $\{ \langle x, y \rangle : A*B. \exists i \in nat. i < \min(length(xs), length(ys))$
 $\wedge x = nth(i, xs) \wedge y = nth(i, ys) \}$
 $\langle proof \rangle$

lemma *list-update-Nil* [simp]: $i \in nat \implies list-update(Nil, i, v) = Nil$

$\langle proof \rangle$

lemma *list-update-Cons-0* [simp]: $list-update(Cons(x, xs), 0, v) = Cons(v, xs)$

$\langle proof \rangle$

lemma *list-update-Cons-succ* [simp]:

$n \in nat \implies$
 $list-update(Cons(x, xs), succ(n), v) = Cons(x, list-update(xs, n, v))$
 $\langle proof \rangle$

lemma *list-update-type* [rule-format]:

$\llbracket xs:list(A); v \in A \rrbracket \implies \forall n \in nat. list-update(xs, n, v):list(A)$
 $\langle proof \rangle$

declare *list-update-type* [simp, TC]

lemma *length-list-update* [rule-format]:

$xs:list(A) \implies \forall i \in nat. length(list-update(xs, i, v)) = length(xs)$

$\langle proof \rangle$

declare *length-list-update* [simp]

lemma *nth-list-update* [rule-format]:

$$\llbracket xs: \text{list}(A) \rrbracket \implies \forall i \in \text{nat}. \forall j \in \text{nat}. i < \text{length}(xs) \longrightarrow \\ \text{nth}(j, \text{list-update}(xs, i, x)) = (\text{if } i=j \text{ then } x \text{ else } \text{nth}(j, xs))$$

$\langle proof \rangle$

lemma *nth-list-update-eq* [simp]:

$$\llbracket i < \text{length}(xs); xs: \text{list}(A) \rrbracket \implies \text{nth}(i, \text{list-update}(xs, i, x)) = x$$

$\langle proof \rangle$

lemma *nth-list-update-neq* [rule-format]:

$$xs: \text{list}(A) \implies$$

$$\forall i \in \text{nat}. \forall j \in \text{nat}. i \neq j \longrightarrow \text{nth}(j, \text{list-update}(xs, i, x)) = \text{nth}(j, xs)$$

$\langle proof \rangle$

declare *nth-list-update-neq* [simp]

lemma *list-update-overwrite* [rule-format]:

$$xs: \text{list}(A) \implies \forall i \in \text{nat}. i < \text{length}(xs)$$

$$\longrightarrow \text{list-update}(\text{list-update}(xs, i, x), i, y) = \text{list-update}(xs, i, y)$$

$\langle proof \rangle$

declare *list-update-overwrite* [simp]

lemma *list-update-same-conv* [rule-format]:

$$xs: \text{list}(A) \implies$$

$$\forall i \in \text{nat}. i < \text{length}(xs) \longrightarrow$$

$$(\text{list-update}(xs, i, x) = xs) \longleftrightarrow (\text{nth}(i, xs) = x)$$

$\langle proof \rangle$

lemma *update-zip* [rule-format]:

$$ys: \text{list}(B) \implies$$

$$\forall i \in \text{nat}. \forall xy \in A * B. \forall xs \in \text{list}(A).$$

$$\text{length}(xs) = \text{length}(ys) \longrightarrow$$

$$\text{list-update}(\text{zip}(xs, ys), i, xy) = \text{zip}(\text{list-update}(xs, i, \text{fst}(xy)), \\ \text{list-update}(ys, i, \text{snd}(xy)))$$

$\langle proof \rangle$

lemma *set-update-subset-cons* [rule-format]:

$$xs: \text{list}(A) \implies$$

$$\forall i \in \text{nat}. \text{set-of-list}(\text{list-update}(xs, i, x)) \subseteq \text{cons}(x, \text{set-of-list}(xs))$$

$\langle proof \rangle$

lemma *set-of-list-update-subsetI*:

$$\llbracket \text{set-of-list}(xs) \subseteq A; xs: \text{list}(A); x \in A; i \in \text{nat} \rrbracket$$

$$\implies \text{set-of-list}(\text{list-update}(xs, i, x)) \subseteq A$$

$\langle proof \rangle$

lemma *upt-rec*:

$j \in \text{nat} \implies \text{upt}(i, j) = (\text{if } i < j \text{ then } \text{Cons}(i, \text{upt}(\text{succ}(i), j)) \text{ else } \text{Nil})$
 $\langle \text{proof} \rangle$

lemma *upt-conv-Nil* [simp]: $\llbracket j \leq i; j \in \text{nat} \rrbracket \implies \text{upt}(i, j) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *upt-succ-append*:

$\llbracket i \leq j; j \in \text{nat} \rrbracket \implies \text{upt}(i, \text{succ}(j)) = \text{upt}(i, j) @ [j]$
 $\langle \text{proof} \rangle$

lemma *upt-conv-Cons*:

$\llbracket i < j; j \in \text{nat} \rrbracket \implies \text{upt}(i, j) = \text{Cons}(i, \text{upt}(\text{succ}(i), j))$
 $\langle \text{proof} \rangle$

lemma *upt-type* [simp, TC]: $j \in \text{nat} \implies \text{upt}(i, j) : \text{list}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *upt-add-eq-append*:

$\llbracket i \leq j; j \in \text{nat}; k \in \text{nat} \rrbracket \implies \text{upt}(i, j \# + k) = \text{upt}(i, j) @ \text{upt}(j, j \# + k)$
 $\langle \text{proof} \rangle$

lemma *length-upt* [simp]: $\llbracket i \in \text{nat}; j \in \text{nat} \rrbracket \implies \text{length}(\text{upt}(i, j)) = j \# - i$
 $\langle \text{proof} \rangle$

lemma *nth-upt* [simp]:

$\llbracket i \in \text{nat}; j \in \text{nat}; k \in \text{nat}; i \# + k < j \rrbracket \implies \text{nth}(k, \text{upt}(i, j)) = i \# + k$
 $\langle \text{proof} \rangle$

lemma *take-upt* [rule-format]:

$\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies$
 $\forall i \in \text{nat}. i \# + m \leq n \longrightarrow \text{take}(m, \text{upt}(i, n)) = \text{upt}(i, i \# + m)$
 $\langle \text{proof} \rangle$

declare *take-upt* [simp]

lemma *map-succ-upt*:

$\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies \text{map}(\text{succ}, \text{upt}(m, n)) = \text{upt}(\text{succ}(m), \text{succ}(n))$
 $\langle \text{proof} \rangle$

lemma *nth-map* [rule-format]:

$xs : \text{list}(A) \implies$
 $\forall n \in \text{nat}. n < \text{length}(xs) \longrightarrow \text{nth}(n, \text{map}(f, xs)) = f(\text{nth}(n, xs))$
 $\langle \text{proof} \rangle$

declare *nth-map* [simp]

lemma *nth-map-upt* [rule-format]:

$\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies$
 $\forall i \in \text{nat}. i < n \#- m \longrightarrow \text{nth}(i, \text{map}(f, \text{upt}(m, n))) = f(m \#+ i)$
 $\langle \text{proof} \rangle$

definition

sublist :: $[i, i] \Rightarrow i$ **where**
sublist(*xs*, *A*) \equiv
 $\text{map}(\text{fst}, (\text{filter}(\lambda p. \text{snd}(p): A, \text{zip}(\text{xs}, \text{upt}(0, \text{length}(\text{xs}))))))$

lemma *sublist-0* [simp]: $\text{xs}:\text{list}(A) \implies \text{sublist}(\text{xs}, 0) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *sublist-Nil* [simp]: $\text{sublist}(\text{Nil}, A) = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma *sublist-shift-lemma*:

$\llbracket \text{xs}:\text{list}(B); i \in \text{nat} \rrbracket \implies$
 $\text{map}(\text{fst}, \text{filter}(\lambda p. \text{snd}(p): A, \text{zip}(\text{xs}, \text{upt}(i, i \#+ \text{length}(\text{xs})))))) =$
 $\text{map}(\text{fst}, \text{filter}(\lambda p. \text{snd}(p): \text{nat} \wedge \text{snd}(p) \#+ i \in A, \text{zip}(\text{xs}, \text{upt}(0, \text{length}(\text{xs}))))))$
 $\langle \text{proof} \rangle$

lemma *sublist-type* [simp, TC]:

$\text{xs}:\text{list}(B) \implies \text{sublist}(\text{xs}, A):\text{list}(B)$
 $\langle \text{proof} \rangle$

lemma *upt-add-eq-append2*:

$\llbracket i \in \text{nat}; j \in \text{nat} \rrbracket \implies \text{upt}(0, i \#+ j) = \text{upt}(0, i) @ \text{upt}(i, i \#+ j)$
 $\langle \text{proof} \rangle$

lemma *sublist-append*:

$\llbracket \text{xs}:\text{list}(B); \text{ys}:\text{list}(B) \rrbracket \implies$
 $\text{sublist}(\text{xs} @ \text{ys}, A) = \text{sublist}(\text{xs}, A) @ \text{sublist}(\text{ys}, \{j \in \text{nat}. j \#+ \text{length}(\text{xs}): A\})$
 $\langle \text{proof} \rangle$

lemma *sublist-Cons*:

$\llbracket \text{xs}:\text{list}(B); x \in B \rrbracket \implies$
 $\text{sublist}(\text{Cons}(x, \text{xs}), A) =$
 $(\text{if } 0 \in A \text{ then } [x] \text{ else } []) @ \text{sublist}(\text{xs}, \{j \in \text{nat}. \text{succ}(j) \in A\})$
 $\langle \text{proof} \rangle$

lemma *sublist-singleton* [simp]:

$\text{sublist}([x], A) = (\text{if } 0 \in A \text{ then } [x] \text{ else } [])$
 $\langle \text{proof} \rangle$

lemma *sublist-upt-eq-take* [rule-format]:

$xs : list(A) \implies \forall n \in nat. sublist(xs, n) = take(n, xs)$
 $\langle proof \rangle$
declare *sublist-upt-eq-take* [simp]

lemma *sublist-Int-eq*:
 $xs \in list(B) \implies sublist(xs, A \cap nat) = sublist(xs, A)$
 $\langle proof \rangle$

Repetition of a List Element

consts *repeat* :: $[i, i] \Rightarrow i$
primrec
 $repeat(a, 0) = []$

$repeat(a, succ(n)) = Cons(a, repeat(a, n))$

lemma *length-repeat*: $n \in nat \implies length(repeat(a, n)) = n$
 $\langle proof \rangle$

lemma *repeat-succ-app*: $n \in nat \implies repeat(a, succ(n)) = repeat(a, n) @ [a]$
 $\langle proof \rangle$

lemma *repeat-type* [TC]: $\llbracket a \in A; n \in nat \rrbracket \implies repeat(a, n) \in list(A)$
 $\langle proof \rangle$

end

30 Equivalence Relations

theory *EquivClass* **imports** *Transl Perm* **begin**

definition
 $quotient :: [i, i] \Rightarrow i \quad (\text{infixl } \langle ' / ' \rangle 90) \quad \text{where}$
 $A / r \equiv \{ r^{-1} \{ x \} \mid x \in A \}$

definition
 $congruent :: [i, i \Rightarrow i] \Rightarrow o \quad \text{where}$
 $congruent(r, b) \equiv \forall y z. \langle y, z \rangle : r \longrightarrow b(y) = b(z)$

definition
 $congruent2 :: [i, i, [i, i] \Rightarrow i] \Rightarrow o \quad \text{where}$
 $congruent2(r1, r2, b) \equiv \forall y1 z1 y2 z2.$
 $\langle y1, z1 \rangle : r1 \longrightarrow \langle y2, z2 \rangle : r2 \longrightarrow b(y1, y2) = b(z1, z2)$

abbreviation
 $RESPECTS :: [i \Rightarrow i, i] \Rightarrow o \quad (\text{infixr } \langle respects \rangle 80) \quad \text{where}$
 $f respects r \equiv congruent(r, f)$

abbreviation
 $RESPECTS2 :: [i \Rightarrow i \Rightarrow i, i] \Rightarrow o \quad (\text{infixr } \langle respects2 \rangle 80) \quad \text{where}$

$f \text{ respects2 } r \equiv \text{congruent2}(r, r, f)$

— Abbreviation for the common case where the relations are identical

30.1 Suppes, Theorem 70: r is an equiv relation iff $\text{converse}(r)$ $O \ r = r$

lemma *sym-trans-comp-subset*:

$\llbracket \text{sym}(r); \text{trans}(r) \rrbracket \implies \text{converse}(r) \ O \ r \subseteq r$
 $\langle \text{proof} \rangle$

lemma *refl-comp-subset*:

$\llbracket \text{refl}(A, r); r \subseteq A * A \rrbracket \implies r \subseteq \text{converse}(r) \ O \ r$
 $\langle \text{proof} \rangle$

lemma *equiv-comp-eq*:

$\text{equiv}(A, r) \implies \text{converse}(r) \ O \ r = r$
 $\langle \text{proof} \rangle$

lemma *comp-equivI*:

$\llbracket \text{converse}(r) \ O \ r = r; \text{domain}(r) = A \rrbracket \implies \text{equiv}(A, r)$
 $\langle \text{proof} \rangle$

lemma *equiv-class-subset*:

$\llbracket \text{sym}(r); \text{trans}(r); \langle a, b \rangle: r \rrbracket \implies r''\{a\} \subseteq r''\{b\}$
 $\langle \text{proof} \rangle$

lemma *equiv-class-eq*:

$\llbracket \text{equiv}(A, r); \langle a, b \rangle: r \rrbracket \implies r''\{a\} = r''\{b\}$
 $\langle \text{proof} \rangle$

lemma *equiv-class-self*:

$\llbracket \text{equiv}(A, r); a \in A \rrbracket \implies a \in r''\{a\}$
 $\langle \text{proof} \rangle$

lemma *subset-equiv-class*:

$\llbracket \text{equiv}(A, r); r''\{b\} \subseteq r''\{a\}; b \in A \rrbracket \implies \langle a, b \rangle: r$
 $\langle \text{proof} \rangle$

lemma *eq-equiv-class*: $\llbracket r''\{a\} = r''\{b\}; \text{equiv}(A, r); b \in A \rrbracket \implies \langle a, b \rangle: r$
 $\langle \text{proof} \rangle$

lemma *equiv-class-nondisjoint*:

$\llbracket \text{equiv}(A, r); x: (r''\{a\} \cap r''\{b\}) \rrbracket \implies \langle a, b \rangle: r$

$\langle proof \rangle$

lemma *equiv-type*: $equiv(A, r) \implies r \subseteq A * A$
 $\langle proof \rangle$

lemma *equiv-class-eq-iff*:
 $equiv(A, r) \implies \langle x, y \rangle: r \iff r''\{x\} = r''\{y\} \wedge x \in A \wedge y \in A$
 $\langle proof \rangle$

lemma *eq-equiv-class-iff*:
 $\llbracket equiv(A, r); x \in A; y \in A \rrbracket \implies r''\{x\} = r''\{y\} \iff \langle x, y \rangle: r$
 $\langle proof \rangle$

lemma *quotientI* $[TC]$: $x \in A \implies r''\{x\}: A//r$
 $\langle proof \rangle$

lemma *quotientE*:
 $\llbracket X \in A//r; \bigwedge x. \llbracket X = r''\{x\}; x \in A \rrbracket \implies P \rrbracket \implies P$
 $\langle proof \rangle$

lemma *Union-quotient*:
 $equiv(A, r) \implies \bigcup (A//r) = A$
 $\langle proof \rangle$

lemma *quotient-disj*:
 $\llbracket equiv(A, r); X \in A//r; Y \in A//r \rrbracket \implies X=Y \mid (X \cap Y \subseteq \emptyset)$
 $\langle proof \rangle$

30.2 Defining Unary Operations upon Equivalence Classes

lemma *UN-equiv-class*:
 $\llbracket equiv(A, r); b \text{ respects } r; a \in A \rrbracket \implies (\bigcup_{x \in r''\{a\}}. b(x)) = b(a)$
 $\langle proof \rangle$

lemma *UN-equiv-class-type*:
 $\llbracket equiv(A, r); b \text{ respects } r; X \in A//r; \bigwedge x. x \in A \implies b(x) \in B \rrbracket$
 $\implies (\bigcup_{x \in X}. b(x)) \in B$
 $\langle proof \rangle$

lemma *UN-equiv-class-inject*:
 $\llbracket equiv(A, r); b \text{ respects } r;$
 $(\bigcup_{x \in X}. b(x)) = (\bigcup_{y \in Y}. b(y)); X \in A//r; Y \in A//r;$
 $\bigwedge x y. \llbracket x \in A; y \in A; b(x) = b(y) \rrbracket \implies \langle x, y \rangle: r \rrbracket$

$\implies X=Y$
 $\langle \text{proof} \rangle$

30.3 Defining Binary Operations upon Equivalence Classes

lemma *congruent2-implies-congruent*:

$\llbracket \text{equiv}(A, r1); \text{congruent2}(r1, r2, b); a \in A \rrbracket \implies \text{congruent}(r2, b(a))$
 $\langle \text{proof} \rangle$

lemma *congruent2-implies-congruent-UN*:

$\llbracket \text{equiv}(A1, r1); \text{equiv}(A2, r2); \text{congruent2}(r1, r2, b); a \in A2 \rrbracket \implies$
 $\text{congruent}(r1, \lambda x1. \bigcup x2 \in r2. \{a\}. b(x1, x2))$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class2*:

$\llbracket \text{equiv}(A1, r1); \text{equiv}(A2, r2); \text{congruent2}(r1, r2, b); a1: A1; a2: A2 \rrbracket$
 $\implies (\bigcup x1 \in r1. \{a1\}. \bigcup x2 \in r2. \{a2\}. b(x1, x2)) = b(a1, a2)$
 $\langle \text{proof} \rangle$

lemma *UN-equiv-class-type2*:

$\llbracket \text{equiv}(A, r); b \text{ respects2 } r;$
 $X1: A//r; X2: A//r;$
 $\bigwedge x1 \ x2. \llbracket x1: A; x2: A \rrbracket \implies b(x1, x2) \in B$
 $\rrbracket \implies (\bigcup x1 \in X1. \bigcup x2 \in X2. b(x1, x2)) \in B$
 $\langle \text{proof} \rangle$

lemma *congruent2I*:

$\llbracket \text{equiv}(A1, r1); \text{equiv}(A2, r2);$
 $\bigwedge y \ z \ w. \llbracket w \in A2; \langle y, z \rangle \in r1 \rrbracket \implies b(y, w) = b(z, w);$
 $\bigwedge y \ z \ w. \llbracket w \in A1; \langle y, z \rangle \in r2 \rrbracket \implies b(w, y) = b(w, z)$
 $\rrbracket \implies \text{congruent2}(r1, r2, b)$
 $\langle \text{proof} \rangle$

lemma *congruent2-commuteI*:

assumes *equivA*: $\text{equiv}(A, r)$
and *commute*: $\bigwedge y \ z. \llbracket y \in A; z \in A \rrbracket \implies b(y, z) = b(z, y)$
and *cong*: $\bigwedge y \ z \ w. \llbracket w \in A; \langle y, z \rangle: r \rrbracket \implies b(w, y) = b(w, z)$
shows *b respects2 r*
 $\langle \text{proof} \rangle$

lemma *congruent-commuteI*:

$\llbracket \text{equiv}(A, r); Z \in A//r;$
 $\bigwedge w. \llbracket w \in A \rrbracket \implies \text{congruent}(r, \lambda z. b(w, z));$
 $\bigwedge x \ y. \llbracket x \in A; y \in A \rrbracket \implies b(y, x) = b(x, y)$
 $\rrbracket \implies \text{congruent}(r, \lambda w. \bigcup z \in Z. b(w, z))$

$\langle proof \rangle$

end

31 The Integers as Equivalence Classes Over Pairs of Natural Numbers

theory *Int* **imports** *EquivClass ArithSimp* **begin**

definition

intrel :: *i* **where**
 $intrel \equiv \{p \in (nat * nat) * (nat * nat). \\ \exists x1\ y1\ x2\ y2. p = \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle \wedge x1 \# + y2 = x2 \# + y1\}$

definition

int :: *i* **where**
 $int \equiv (nat * nat) // intrel$

definition

int-of :: *i* \Rightarrow *i* — coercion from nat to int ($\langle \langle open-block notation = \langle prefix \$\# \rangle \rangle \$\#$
 $- \rangle [80] 80$)
where $\$ \# m \equiv intrel \text{ `` } \{ \langle natify(m), 0 \rangle \}$

definition

intify :: *i* \Rightarrow *i* — coercion from ANYTHING to int **where**
 $intify(m) \equiv \text{if } m \in int \text{ then } m \text{ else } \$ \# 0$

definition

raw-zminus :: *i* \Rightarrow *i* **where**
 $raw-zminus(z) \equiv \bigcup \langle x, y \rangle \in z. intrel \text{ `` } \{ \langle y, x \rangle \}$

definition

zminus :: *i* \Rightarrow *i* ($\langle \langle open-block notation = \langle prefix \$- \rangle \rangle \$- - \rangle [80] 80$)
where $\$ - z \equiv raw-zminus (intify(z))$

definition

znegative :: *i* \Rightarrow *o* **where**
 $znegative(z) \equiv \exists x\ y. x < y \wedge y \in nat \wedge \langle x, y \rangle \in z$

definition

iszero :: *i* \Rightarrow *o* **where**
 $iszero(z) \equiv z = \$ \# 0$

definition

raw-nat-of :: *i* \Rightarrow *i* **where**
 $raw-nat-of(z) \equiv natify (\bigcup \langle x, y \rangle \in z. x \# - y)$

definition

nat-of :: $i \Rightarrow i$ **where**
nat-of(z) \equiv *raw-nat-of* (*intify*(z))

definition

zmagnitude :: $i \Rightarrow i$ **where**
 — could be replaced by an absolute value function from int to int?
zmagnitude(z) \equiv
 THE m . $m \in \text{nat} \wedge ((\neg \text{znegative}(z) \wedge z = \$\# m) \mid$
 $(\text{znegative}(z) \wedge \$- z = \$\# m))$

definition

raw-zmult :: $[i, i] \Rightarrow i$ **where**
 $\text{raw-zmult}(z1, z2) \equiv$
 $\bigcup p1 \in z1. \bigcup p2 \in z2. \text{split}(\lambda x1 \ y1. \text{split}(\lambda x2 \ y2.$
 $\text{intrel}''\{\langle x1 \# * x2 \ \# + \ y1 \# * y2, x1 \# * y2 \ \# + \ y1 \# * x2 \rangle\}, p2), p1)$

definition

zmult :: $[i, i] \Rightarrow i$ (**infixl** $\langle \$* \rangle$ 70) **where**
 $z1 \ \$* \ z2 \equiv \text{raw-zmult} (\text{intify}(z1), \text{intify}(z2))$

definition

raw-zadd :: $[i, i] \Rightarrow i$ **where**
 $\text{raw-zadd}(z1, z2) \equiv$
 $\bigcup z1 \in z1. \bigcup z2 \in z2. \text{let } \langle x1, y1 \rangle = z1; \langle x2, y2 \rangle = z2$
 $\text{in intrel}''\{\langle x1 \# + x2, y1 \# + y2 \rangle\}$

definition

zadd :: $[i, i] \Rightarrow i$ (**infixl** $\langle \$+ \rangle$ 65) **where**
 $z1 \ \$+ \ z2 \equiv \text{raw-zadd} (\text{intify}(z1), \text{intify}(z2))$

definition

zdiff :: $[i, i] \Rightarrow i$ (**infixl** $\langle \$- \rangle$ 65) **where**
 $z1 \ \$- \ z2 \equiv z1 \ \$+ \ \text{zminus}(z2)$

definition

zless :: $[i, i] \Rightarrow o$ (**infixl** $\langle \$< \rangle$ 50) **where**
 $z1 \ \$< \ z2 \equiv \text{znegative}(z1 \ \$- \ z2)$

definition

zle :: $[i, i] \Rightarrow o$ (**infixl** $\langle \$\leq \rangle$ 50) **where**
 $z1 \ \$\leq \ z2 \equiv z1 \ \$< \ z2 \mid \text{intify}(z1) = \text{intify}(z2)$

declare *quotientE* [*elim!*]

31.1 Proving that *intrel* is an equivalence relation

lemma *intrel-iff* [*simp*]:

$\langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle >: \text{intrel} \longleftrightarrow$
 $x1 \in \text{nat} \wedge y1 \in \text{nat} \wedge x2 \in \text{nat} \wedge y2 \in \text{nat} \wedge x1 \# + y2 = x2 \# + y1$
 $\langle \text{proof} \rangle$

lemma *intrelI* [*intro!*]:
 $\llbracket x1 \# + y2 = x2 \# + y1; x1 \in \text{nat}; y1 \in \text{nat}; x2 \in \text{nat}; y2 \in \text{nat} \rrbracket$
 $\implies \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle >: \text{intrel}$
 $\langle \text{proof} \rangle$

lemma *intrelE* [*elim!*]:
 $\llbracket p \in \text{intrel};$
 $\bigwedge x1 \ y1 \ x2 \ y2. \llbracket p = \langle \langle x1, y1 \rangle, \langle x2, y2 \rangle \rangle >; x1 \# + y2 = x2 \# + y1;$
 $x1 \in \text{nat}; y1 \in \text{nat}; x2 \in \text{nat}; y2 \in \text{nat} \rrbracket \implies Q \rrbracket$
 $\implies Q$
 $\langle \text{proof} \rangle$

lemma *int-trans-lemma*:
 $\llbracket x1 \# + y2 = x2 \# + y1; x2 \# + y3 = x3 \# + y2 \rrbracket \implies x1 \# + y3 = x3 \# + y1$
 $\langle \text{proof} \rangle$

lemma *equiv-intrel*: *equiv*(*nat***nat*, *intrel*)
 $\langle \text{proof} \rangle$

lemma *image-intrel-int*: $\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies \text{intrel} \text{ `` } \{ \langle m, n \rangle \} \in \text{int}$
 $\langle \text{proof} \rangle$

declare *equiv-intrel* [*THEN eq-equiv-class-iff*, *simp*]
declare *conj-cong* [*cong*]

lemmas *eq-intrelD* = *eq-equiv-class* [*OF* - *equiv-intrel*]

lemma *int-of-type* [*simp*, *TC*]: $\$ \# m \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *int-of-eq* [*iff*]: $(\$ \# m = \$ \# n) \longleftrightarrow \text{nativify}(m) = \text{nativify}(n)$
 $\langle \text{proof} \rangle$

lemma *int-of-inject*: $\llbracket \$ \# m = \$ \# n; m \in \text{nat}; n \in \text{nat} \rrbracket \implies m = n$
 $\langle \text{proof} \rangle$

lemma *intify-in-int* [*iff*, *TC*]: *intify*(*x*) \in *int*
 $\langle \text{proof} \rangle$

lemma *intify-ident* [simp]: $n \in \text{int} \implies \text{intify}(n) = n$
 $\langle \text{proof} \rangle$

31.2 Collapsing rules: to remove *intify* from arithmetic expressions

lemma *intify-idem* [simp]: $\text{intify}(\text{intify}(x)) = \text{intify}(x)$
 $\langle \text{proof} \rangle$

lemma *int-of-natify* [simp]: $\$ \# (\text{natify}(m)) = \$ \# m$
 $\langle \text{proof} \rangle$

lemma *zminus-intify* [simp]: $\$ - (\text{intify}(m)) = \$ - m$
 $\langle \text{proof} \rangle$

lemma *zadd-intify1* [simp]: $\text{intify}(x) \$ + y = x \$ + y$
 $\langle \text{proof} \rangle$

lemma *zadd-intify2* [simp]: $x \$ + \text{intify}(y) = x \$ + y$
 $\langle \text{proof} \rangle$

lemma *zdiff-intify1* [simp]: $\text{intify}(x) \$ - y = x \$ - y$
 $\langle \text{proof} \rangle$

lemma *zdiff-intify2* [simp]: $x \$ - \text{intify}(y) = x \$ - y$
 $\langle \text{proof} \rangle$

lemma *zmult-intify1* [simp]: $\text{intify}(x) \$ * y = x \$ * y$
 $\langle \text{proof} \rangle$

lemma *zmult-intify2* [simp]: $x \$ * \text{intify}(y) = x \$ * y$
 $\langle \text{proof} \rangle$

lemma *zless-intify1* [simp]: $\text{intify}(x) \$ < y \longleftrightarrow x \$ < y$
 $\langle \text{proof} \rangle$

lemma *zless-intify2* [simp]: $x \$ < \text{intify}(y) \longleftrightarrow x \$ < y$
 $\langle \text{proof} \rangle$

lemma *zle-intify1* [simp]: $\text{intify}(x) \$ \leq y \longleftrightarrow x \$ \leq y$
 $\langle \text{proof} \rangle$

lemma *zle-intify2* [*simp*]: $x \leq \text{intify}(y) \longleftrightarrow x \leq y$
 ⟨*proof*⟩

31.3 *zminus*: unary negation on *int*

lemma *zminus-congruent*: $(\lambda \langle x, y \rangle. \text{intrel} \{ \langle y, x \rangle \})$ respects *intrel*
 ⟨*proof*⟩

lemma *raw-zminus-type*: $z \in \text{int} \implies \text{raw-zminus}(z) \in \text{int}$
 ⟨*proof*⟩

lemma *zminus-type* [*TC, iff*]: $\$-z \in \text{int}$
 ⟨*proof*⟩

lemma *raw-zminus-inject*:
 $\llbracket \text{raw-zminus}(z) = \text{raw-zminus}(w); z \in \text{int}; w \in \text{int} \rrbracket \implies z = w$
 ⟨*proof*⟩

lemma *zminus-inject-intify* [*dest!*]: $\$-z = \$-w \implies \text{intify}(z) = \text{intify}(w)$
 ⟨*proof*⟩

lemma *zminus-inject*: $\llbracket \$-z = \$-w; z \in \text{int}; w \in \text{int} \rrbracket \implies z = w$
 ⟨*proof*⟩

lemma *raw-zminus*:
 $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{raw-zminus}(\text{intrel} \{ \langle x, y \rangle \}) = \text{intrel} \{ \langle y, x \rangle \}$
 ⟨*proof*⟩

lemma *zminus*:
 $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \$- (\text{intrel} \{ \langle x, y \rangle \}) = \text{intrel} \{ \langle y, x \rangle \}$
 ⟨*proof*⟩

lemma *raw-zminus-zminus*: $z \in \text{int} \implies \text{raw-zminus} (\text{raw-zminus}(z)) = z$
 ⟨*proof*⟩

lemma *zminus-zminus-intify* [*simp*]: $\$- (\$- z) = \text{intify}(z)$
 ⟨*proof*⟩

lemma *zminus-int0* [*simp*]: $\$- (\$ \# 0) = \$ \# 0$
 ⟨*proof*⟩

lemma *zminus-zminus*: $z \in \text{int} \implies \$- (\$- z) = z$
 ⟨*proof*⟩

31.4 *znegative*: the test for negative integers

lemma *znegative*: $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{znegative}(\text{intrel} \{ \langle x, y \rangle \}) \longleftrightarrow x < y$
 ⟨*proof*⟩

lemma *not-znegative-int-of [iff]*: $\neg \text{znegative}(\$ \# n)$
 $\langle \text{proof} \rangle$

lemma *znegative-zminus-int-of [simp]*: $\text{znegative}(\$ - \$ \# \text{succ}(n))$
 $\langle \text{proof} \rangle$

lemma *not-znegative-imp-zero*: $\neg \text{znegative}(\$ - \$ \# n) \implies \text{natify}(n) = 0$
 $\langle \text{proof} \rangle$

31.5 *nat-of*: Coercion of an Integer to a Natural Number

lemma *nat-of-intify [simp]*: $\text{nat-of}(\text{intify}(z)) = \text{nat-of}(z)$
 $\langle \text{proof} \rangle$

lemma *nat-of-congruent*: $(\lambda x. (\lambda \langle x, y \rangle. x \# - y)(x))$ respects *intrel*
 $\langle \text{proof} \rangle$

lemma *raw-nat-of*:
 $\llbracket x \in \text{nat}; y \in \text{nat} \rrbracket \implies \text{raw-nat-of}(\text{intrel} \{ \langle x, y \rangle \}) = x \# - y$
 $\langle \text{proof} \rangle$

lemma *raw-nat-of-int-of*: $\text{raw-nat-of}(\$ \# n) = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *nat-of-int-of [simp]*: $\text{nat-of}(\$ \# n) = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *raw-nat-of-type*: $\text{raw-nat-of}(z) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *nat-of-type [iff, TC]*: $\text{nat-of}(z) \in \text{nat}$
 $\langle \text{proof} \rangle$

31.6 *zmagnitude*: magnitude of an integer, as a natural number

lemma *zmagnitude-int-of [simp]*: $\text{zmagnitude}(\$ \# n) = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *natify-int-of-eq*: $\text{natify}(x) = n \implies \$ \# x = \$ \# n$
 $\langle \text{proof} \rangle$

lemma *zmagnitude-zminus-int-of [simp]*: $\text{zmagnitude}(\$ - \$ \# n) = \text{natify}(n)$
 $\langle \text{proof} \rangle$

lemma *zmagnitude-type [iff, TC]*: $\text{zmagnitude}(z) \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *not-zneg-int-of*:

$\llbracket z \in \text{int}; \neg \text{znegative}(z) \rrbracket \implies \exists n \in \text{nat}. z = \$\# n$
 $\langle \text{proof} \rangle$

lemma *not-zneg-mag [simp]*:

$\llbracket z \in \text{int}; \neg \text{znegative}(z) \rrbracket \implies \$\# (\text{zmagnitude}(z)) = z$
 $\langle \text{proof} \rangle$

lemma *zneg-int-of*:

$\llbracket \text{znegative}(z); z \in \text{int} \rrbracket \implies \exists n \in \text{nat}. z = \$- (\$ \# \text{succ}(n))$
 $\langle \text{proof} \rangle$

lemma *zneg-mag [simp]*:

$\llbracket \text{znegative}(z); z \in \text{int} \rrbracket \implies \$\# (\text{zmagnitude}(z)) = \$- z$
 $\langle \text{proof} \rangle$

lemma *int-cases*: $z \in \text{int} \implies \exists n \in \text{nat}. z = \$\# n \mid z = \$- (\$ \# \text{succ}(n))$
 $\langle \text{proof} \rangle$

lemma *not-zneg-raw-nat-of*:

$\llbracket \neg \text{znegative}(z); z \in \text{int} \rrbracket \implies \$\# (\text{raw-nat-of}(z)) = z$
 $\langle \text{proof} \rangle$

lemma *not-zneg-nat-of-intify*:

$\neg \text{znegative}(\text{intify}(z)) \implies \$\# (\text{nat-of}(z)) = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *not-zneg-nat-of*: $\llbracket \neg \text{znegative}(z); z \in \text{int} \rrbracket \implies \$\# (\text{nat-of}(z)) = z$
 $\langle \text{proof} \rangle$

lemma *zneg-nat-of [simp]*: $\text{znegative}(\text{intify}(z)) \implies \text{nat-of}(z) = 0$
 $\langle \text{proof} \rangle$

31.7 (\$+): addition on int

Congruence Property for Addition

lemma *zadd-congruent2*:

$(\lambda z1\ z2. \text{let } \langle x1, y1 \rangle = z1; \langle x2, y2 \rangle = z2$
 $\text{in } \text{intrel} \{ \langle x1 \# + x2, y1 \# + y2 \rangle \})$
 $\text{respects2 } \text{intrel}$
 $\langle \text{proof} \rangle$

lemma *raw-zadd-type*: $\llbracket z \in \text{int}; w \in \text{int} \rrbracket \implies \text{raw-zadd}(z, w) \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *zadd-type [iff, TC]*: $z \$+ w \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *raw-zadd*:

$$\begin{aligned} & \llbracket x1 \in \text{nat}; y1 \in \text{nat}; x2 \in \text{nat}; y2 \in \text{nat} \rrbracket \\ & \implies \text{raw-zadd} (\text{intrel} \{ \langle x1, y1 \rangle \}, \text{intrel} \{ \langle x2, y2 \rangle \}) = \\ & \quad \text{intrel} \{ \langle x1 \# + x2, y1 \# + y2 \rangle \} \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zadd*:

$$\begin{aligned} & \llbracket x1 \in \text{nat}; y1 \in \text{nat}; x2 \in \text{nat}; y2 \in \text{nat} \rrbracket \\ & \implies (\text{intrel} \{ \langle x1, y1 \rangle \}) \$+ (\text{intrel} \{ \langle x2, y2 \rangle \}) = \\ & \quad \text{intrel} \{ \langle x1 \# + x2, y1 \# + y2 \rangle \} \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *raw-zadd-int0*: $z \in \text{int} \implies \text{raw-zadd} (\$ \# 0, z) = z$
 $\langle \text{proof} \rangle$

lemma *zadd-int0-intify [simp]*: $\$ \# 0 \$+ z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zadd-int0*: $z \in \text{int} \implies \$ \# 0 \$+ z = z$
 $\langle \text{proof} \rangle$

lemma *raw-zminus-zadd-distrib*:

$$\begin{aligned} & \llbracket z \in \text{int}; w \in \text{int} \rrbracket \implies \$- \text{raw-zadd}(z, w) = \text{raw-zadd}(\$- z, \$- w) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zminus-zadd-distrib [simp]*: $\$- (z \$+ w) = \$- z \$+ \$- w$
 $\langle \text{proof} \rangle$

lemma *raw-zadd-commute*:

$$\begin{aligned} & \llbracket z \in \text{int}; w \in \text{int} \rrbracket \implies \text{raw-zadd}(z, w) = \text{raw-zadd}(w, z) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zadd-commute*: $z \$+ w = w \$+ z$
 $\langle \text{proof} \rangle$

lemma *raw-zadd-assoc*:

$$\begin{aligned} & \llbracket z1: \text{int}; z2: \text{int}; z3: \text{int} \rrbracket \\ & \implies \text{raw-zadd} (\text{raw-zadd}(z1, z2), z3) = \text{raw-zadd}(z1, \text{raw-zadd}(z2, z3)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zadd-assoc*: $(z1 \$+ z2) \$+ z3 = z1 \$+ (z2 \$+ z3)$
 $\langle \text{proof} \rangle$

lemma *zadd-left-commute*: $z1 \$+ (z2 \$+ z3) = z2 \$+ (z1 \$+ z3)$
 $\langle \text{proof} \rangle$

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

lemma *int-of-add*: $\$ \# (m \# + n) = (\$ \# m) \$ + (\$ \# n)$
 $\langle proof \rangle$

lemma *int-succ-int-1*: $\$ \# succ(m) = \$ \# 1 \$ + (\$ \# m)$
 $\langle proof \rangle$

lemma *int-of-diff*:
 $\llbracket m \in nat; n \leq m \rrbracket \implies \$ \# (m \# - n) = (\$ \# m) \$ - (\$ \# n)$
 $\langle proof \rangle$

lemma *raw-zadd-zminus-inverse*: $z \in int \implies raw-zadd(z, \$ - z) = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-zminus-inverse [simp]*: $z \$ + (\$ - z) = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-zminus-inverse2 [simp]*: $(\$ - z) \$ + z = \$ \# 0$
 $\langle proof \rangle$

lemma *zadd-int0-right-intify [simp]*: $z \$ + \$ \# 0 = intify(z)$
 $\langle proof \rangle$

lemma *zadd-int0-right*: $z \in int \implies z \$ + \$ \# 0 = z$
 $\langle proof \rangle$

31.8 $(\$ \#)$: Integer Multiplication

Congruence property for multiplication

lemma *zmult-congruent2*:
 $(\lambda p1 p2. split(\lambda x1 y1. split(\lambda x2 y2. intrel' \{ \langle x1 \# * x2 \# + y1 \# * y2, x1 \# * y2 \# + y1 \# * x2 \rangle \}, p2), p1))$
respects2 intrel
 $\langle proof \rangle$

lemma *raw-zmult-type*: $\llbracket z \in int; w \in int \rrbracket \implies raw-zmult(z, w) \in int$
 $\langle proof \rangle$

lemma *zmult-type [iff, TC]*: $z \$ * w \in int$
 $\langle proof \rangle$

lemma *raw-zmult*:
 $\llbracket x1 \in nat; y1 \in nat; x2 \in nat; y2 \in nat \rrbracket$
 $\implies raw-zmult(intrel' \{ \langle x1, y1 \rangle \}, intrel' \{ \langle x2, y2 \rangle \}) =$
 $intrel' \{ \langle x1 \# * x2 \# + y1 \# * y2, x1 \# * y2 \# + y1 \# * x2 \rangle \}$
 $\langle proof \rangle$

lemma *zmult*:

$$\begin{aligned} & \llbracket x1 \in \text{nat}; y1 \in \text{nat}; x2 \in \text{nat}; y2 \in \text{nat} \rrbracket \\ & \implies (\text{intrel} \langle \{x1, y1\} \rangle) \$* (\text{intrel} \langle \{x2, y2\} \rangle) = \\ & \quad \text{intrel} \langle \{x1 \#* x2 \# + y1 \#* y2, x1 \#* y2 \# + y1 \#* x2\} \rangle \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *raw-zmult-int0*: $z \in \text{int} \implies \text{raw-zmult} (\$ \# 0, z) = \$ \# 0$
 $\langle \text{proof} \rangle$

lemma *zmult-int0 [simp]*: $\$ \# 0 \$* z = \$ \# 0$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-int1*: $z \in \text{int} \implies \text{raw-zmult} (\$ \# 1, z) = z$
 $\langle \text{proof} \rangle$

lemma *zmult-int1-intify [simp]*: $\$ \# 1 \$* z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-int1*: $z \in \text{int} \implies \$ \# 1 \$* z = z$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-commute*:

$$\llbracket z \in \text{int}; w \in \text{int} \rrbracket \implies \text{raw-zmult}(z, w) = \text{raw-zmult}(w, z)$$
 $\langle \text{proof} \rangle$

lemma *zmult-commute*: $z \$* w = w \$* z$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-zminus*:

$$\llbracket z \in \text{int}; w \in \text{int} \rrbracket \implies \text{raw-zmult}(\$ - z, w) = \$ - \text{raw-zmult}(z, w)$$
 $\langle \text{proof} \rangle$

lemma *zmult-zminus [simp]*: $(\$ - z) \$* w = \$ - (z \$* w)$
 $\langle \text{proof} \rangle$

lemma *zmult-zminus-right [simp]*: $w \$* (\$ - z) = \$ - (w \$* z)$
 $\langle \text{proof} \rangle$

lemma *raw-zmult-assoc*:

$$\begin{aligned} & \llbracket z1: \text{int}; z2: \text{int}; z3: \text{int} \rrbracket \\ & \implies \text{raw-zmult} (\text{raw-zmult}(z1, z2), z3) = \text{raw-zmult}(z1, \text{raw-zmult}(z2, z3)) \\ & \langle \text{proof} \rangle \end{aligned}$$

lemma *zmult-assoc*: $(z1 \$* z2) \$* z3 = z1 \$* (z2 \$* z3)$
 $\langle \text{proof} \rangle$

lemma *zmult-left-commute*: $z1 \$* (z2 \$* z3) = z2 \$* (z1 \$* z3)$
 $\langle \text{proof} \rangle$

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

lemma *raw-zadd-zmult-distrib:*

$\llbracket z1: int; z2: int; w \in int \rrbracket$
 $\implies raw\text{-}zmult(raw\text{-}zadd(z1, z2), w) =$
 $raw\text{-}zadd (raw\text{-}zmult(z1, w), raw\text{-}zmult(z2, w))$
 $\langle proof \rangle$

lemma *zadd-zmult-distrib:* $(z1 \$+ z2) \$* w = (z1 \$* w) \$+ (z2 \$* w)$
 $\langle proof \rangle$

lemma *zadd-zmult-distrib2:* $w \$* (z1 \$+ z2) = (w \$* z1) \$+ (w \$* z2)$
 $\langle proof \rangle$

lemmas *int-typechecks =*
int-of-type zminus-type zmagnitude-type zadd-type zmult-type

lemma *zdiff-type [iff, TC]:* $z \$- w \in int$
 $\langle proof \rangle$

lemma *zminus-zdiff-eq [simp]:* $\$- (z \$- y) = y \$- z$
 $\langle proof \rangle$

lemma *zdiff-zmult-distrib:* $(z1 \$- z2) \$* w = (z1 \$* w) \$- (z2 \$* w)$
 $\langle proof \rangle$

lemma *zdiff-zmult-distrib2:* $w \$* (z1 \$- z2) = (w \$* z1) \$- (w \$* z2)$
 $\langle proof \rangle$

lemma *zadd-zdiff-eq:* $x \$+ (y \$- z) = (x \$+ y) \$- z$
 $\langle proof \rangle$

lemma *zdiff-zadd-eq:* $(x \$- y) \$+ z = (x \$+ z) \$- y$
 $\langle proof \rangle$

31.9 The "Less Than" Relation

lemma *zless-linear-lemma:*

$\llbracket z \in int; w \in int \rrbracket \implies z \$< w \mid z = w \mid w \$< z$
 $\langle proof \rangle$

lemma *zless-linear:* $z \$< w \mid intify(z) = intify(w) \mid w \$< z$
 $\langle proof \rangle$

lemma *zless-not-refl [iff]:* $\neg (z \$< z)$

$\langle proof \rangle$

lemma *neq-iff-zless*: $\llbracket x \in int; y \in int \rrbracket \implies (x \neq y) \longleftrightarrow (x \$< y \mid y \$< x)$
 $\langle proof \rangle$

lemma *zless-imp-intify-neq*: $w \$< z \implies intify(w) \neq intify(z)$
 $\langle proof \rangle$

lemma *zless-imp-succ-zadd-lemma*:
 $\llbracket w \$< z; w \in int; z \in int \rrbracket \implies (\exists n \in nat. z = w \$+ \$\#(succ(n)))$
 $\langle proof \rangle$

lemma *zless-imp-succ-zadd*:
 $w \$< z \implies (\exists n \in nat. w \$+ \$\#(succ(n)) = intify(z))$
 $\langle proof \rangle$

lemma *zless-succ-zadd-lemma*:
 $w \in int \implies w \$< w \$+ \$\# succ(n)$
 $\langle proof \rangle$

lemma *zless-succ-zadd*: $w \$< w \$+ \$\# succ(n)$
 $\langle proof \rangle$

lemma *zless-iff-succ-zadd*:
 $w \$< z \longleftrightarrow (\exists n \in nat. w \$+ \$\#(succ(n)) = intify(z))$
 $\langle proof \rangle$

lemma *zless-int-of [simp]*: $\llbracket m \in nat; n \in nat \rrbracket \implies (\$ \# m \$< \$ \# n) \longleftrightarrow (m < n)$
 $\langle proof \rangle$

lemma *zless-trans-lemma*:
 $\llbracket x \$< y; y \$< z; x \in int; y \in int; z \in int \rrbracket \implies x \$< z$
 $\langle proof \rangle$

lemma *zless-trans [trans]*: $\llbracket x \$< y; y \$< z \rrbracket \implies x \$< z$
 $\langle proof \rangle$

lemma *zless-not-sym*: $z \$< w \implies \neg (w \$< z)$
 $\langle proof \rangle$

lemmas *zless-asm* = *zless-not-sym* [THEN swap]

lemma *zless-imp-zle*: $z \$< w \implies z \$\leq w$
 $\langle proof \rangle$

lemma *zle-linear*: $z \$\leq w \mid w \$\leq z$
 $\langle proof \rangle$

31.10 Less Than or Equals

lemma *zle-refl*: $z \leq z$
 $\langle proof \rangle$

lemma *zle-eq-refl*: $x=y \implies x \leq y$
 $\langle proof \rangle$

lemma *zle-anti-sym-intify*: $\llbracket x \leq y; y \leq x \rrbracket \implies \text{intify}(x) = \text{intify}(y)$
 $\langle proof \rangle$

lemma *zle-anti-sym*: $\llbracket x \leq y; y \leq x; x \in \text{int}; y \in \text{int} \rrbracket \implies x=y$
 $\langle proof \rangle$

lemma *zle-trans-lemma*:
 $\llbracket x \in \text{int}; y \in \text{int}; z \in \text{int}; x \leq y; y \leq z \rrbracket \implies x \leq z$
 $\langle proof \rangle$

lemma *zle-trans* [trans]: $\llbracket x \leq y; y \leq z \rrbracket \implies x \leq z$
 $\langle proof \rangle$

lemma *zle-zless-trans* [trans]: $\llbracket i \leq j; j < k \rrbracket \implies i < k$
 $\langle proof \rangle$

lemma *zless-zle-trans* [trans]: $\llbracket i < j; j \leq k \rrbracket \implies i < k$
 $\langle proof \rangle$

lemma *not-zless-iff-zle*: $\neg (z < w) \longleftrightarrow (w \leq z)$
 $\langle proof \rangle$

lemma *not-zle-iff-zless*: $\neg (z \leq w) \longleftrightarrow (w < z)$
 $\langle proof \rangle$

31.11 More subtraction laws (for *zcompare-rls*)

lemma *zdifff-zdifff-eq*: $(x \$- y) \$- z = x \$- (y \$+ z)$
 $\langle proof \rangle$

lemma *zdifff-zdifff-eq2*: $x \$- (y \$- z) = (x \$+ z) \$- y$
 $\langle proof \rangle$

lemma *zdifff-zless-iff*: $(x \$- y < z) \longleftrightarrow (x < z \$+ y)$
 $\langle proof \rangle$

lemma *zless-zdifff-iff*: $(x < z \$- y) \longleftrightarrow (x \$+ y < z)$
 $\langle proof \rangle$

lemma *zdifff-eq-iff*: $\llbracket x \in \text{int}; z \in \text{int} \rrbracket \implies (x \$- y = z) \longleftrightarrow (x = z \$+ y)$
 $\langle proof \rangle$

lemma *eq-zdiff-iff*: $\llbracket x \in \text{int}; z \in \text{int} \rrbracket \implies (x = z\$-y) \longleftrightarrow (x \$+ y = z)$
 $\langle \text{proof} \rangle$

lemma *zdiff-zle-iff-lemma*:
 $\llbracket x \in \text{int}; z \in \text{int} \rrbracket \implies (x\$-y \$\leq z) \longleftrightarrow (x \$\leq z \$+ y)$
 $\langle \text{proof} \rangle$

lemma *zdiff-zle-iff*: $(x\$-y \$\leq z) \longleftrightarrow (x \$\leq z \$+ y)$
 $\langle \text{proof} \rangle$

lemma *zle-zdiff-iff-lemma*:
 $\llbracket x \in \text{int}; z \in \text{int} \rrbracket \implies (x \$\leq z\$-y) \longleftrightarrow (x \$+ y \$\leq z)$
 $\langle \text{proof} \rangle$

lemma *zle-zdiff-iff*: $(x \$\leq z\$-y) \longleftrightarrow (x \$+ y \$\leq z)$
 $\langle \text{proof} \rangle$

This list of rewrites simplifies (in)equalities by bringing subtractions to the top and then moving negative terms to the other side. Use with *zadd-ac*

lemmas *zcompare-rls* =
zdiff-def [*symmetric*]
zadd-zdiff-eq *zdiff-zadd-eq* *zdiff-zdiff-eq* *zdiff-zdiff-eq2*
zdiff-zless-iff *zless-zdiff-iff* *zdiff-zle-iff* *zle-zdiff-iff*
zdiff-eq-iff *eq-zdiff-iff*

31.12 Monotonicity and Cancellation Results for Instantiation of the CancelNumerals Simprocs

lemma *zadd-left-cancel*:
 $\llbracket w \in \text{int}; w': \text{int} \rrbracket \implies (z \$+ w' = z \$+ w) \longleftrightarrow (w' = w)$
 $\langle \text{proof} \rangle$

lemma *zadd-left-cancel-intify* [*simp*]:
 $(z \$+ w' = z \$+ w) \longleftrightarrow \text{intify}(w') = \text{intify}(w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel*:
 $\llbracket w \in \text{int}; w': \text{int} \rrbracket \implies (w' \$+ z = w \$+ z) \longleftrightarrow (w' = w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-intify* [*simp*]:
 $(w' \$+ z = w \$+ z) \longleftrightarrow \text{intify}(w') = \text{intify}(w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-zless* [*simp*]: $(w' \$+ z \$< w \$+ z) \longleftrightarrow (w' \$< w)$
 $\langle \text{proof} \rangle$

lemma *zadd-left-cancel-zless* [*simp*]: $(z \$+ w' \$< z \$+ w) \longleftrightarrow (w' \$< w)$
 $\langle \text{proof} \rangle$

lemma *zadd-right-cancel-zle* [*simp*]: $(w' \$+ z \$\leq w \$+ z) \longleftrightarrow w' \$\leq w$
 $\langle proof \rangle$

lemma *zadd-left-cancel-zle* [*simp*]: $(z \$+ w' \$\leq z \$+ w) \longleftrightarrow w' \$\leq w$
 $\langle proof \rangle$

lemmas *zadd-zless-mono1* = *zadd-right-cancel-zless* [*THEN iffD2*]

lemmas *zadd-zless-mono2* = *zadd-left-cancel-zless* [*THEN iffD2*]

lemmas *zadd-zle-mono1* = *zadd-right-cancel-zle* [*THEN iffD2*]

lemmas *zadd-zle-mono2* = *zadd-left-cancel-zle* [*THEN iffD2*]

lemma *zadd-zle-mono*: $\llbracket w' \$\leq w; z' \$\leq z \rrbracket \Longrightarrow w' \$+ z' \$\leq w \$+ z$
 $\langle proof \rangle$

lemma *zadd-zless-mono*: $\llbracket w' \$< w; z' \$\leq z \rrbracket \Longrightarrow w' \$+ z' \$< w \$+ z$
 $\langle proof \rangle$

31.13 Comparison laws

lemma *zminus-zless-zminus* [*simp*]: $(\$- x \$< \$- y) \longleftrightarrow (y \$< x)$
 $\langle proof \rangle$

lemma *zminus-zle-zminus* [*simp*]: $(\$- x \$\leq \$- y) \longleftrightarrow (y \$\leq x)$
 $\langle proof \rangle$

31.13.1 More inequality lemmas

lemma *equation-zminus*: $\llbracket x \in int; y \in int \rrbracket \Longrightarrow (x = \$- y) \longleftrightarrow (y = \$- x)$
 $\langle proof \rangle$

lemma *zminus-equation*: $\llbracket x \in int; y \in int \rrbracket \Longrightarrow (\$- x = y) \longleftrightarrow (\$- y = x)$
 $\langle proof \rangle$

lemma *equation-zminus-intify*: $(intify(x) = \$- y) \longleftrightarrow (intify(y) = \$- x)$
 $\langle proof \rangle$

lemma *zminus-equation-intify*: $(\$- x = intify(y)) \longleftrightarrow (\$- y = intify(x))$
 $\langle proof \rangle$

31.13.2 The next several equations are permutative: watch out!

lemma *zless-zminus*: $(x \text{ \$< \$- } y) \longleftrightarrow (y \text{ \$< \$- } x)$
 $\langle \text{proof} \rangle$

lemma *zminus-zless*: $(\text{\$- } x \text{ \$< } y) \longleftrightarrow (\text{\$- } y \text{ \$< } x)$
 $\langle \text{proof} \rangle$

lemma *zle-zminus*: $(x \text{ \$}\leq \text{\$- } y) \longleftrightarrow (y \text{ \$}\leq \text{\$- } x)$
 $\langle \text{proof} \rangle$

lemma *zminus-zle*: $(\text{\$- } x \text{ \$}\leq y) \longleftrightarrow (\text{\$- } y \text{ \$}\leq x)$
 $\langle \text{proof} \rangle$

end

32 Arithmetic on Binary Integers

theory *Bin*
imports *Int Datatype*
begin

consts *bin* :: *i*
datatype
bin = *Pls*
| *Min*
| *Bit* (*w* ∈ *bin*, *b* ∈ *bool*) (**infixl** $\langle \text{BIT} \rangle$ 90)

consts
integ-of :: $i \Rightarrow i$
NCons :: $[i, i] \Rightarrow i$
bin-succ :: $i \Rightarrow i$
bin-pred :: $i \Rightarrow i$
bin-minus :: $i \Rightarrow i$
bin-adder :: $i \Rightarrow i$
bin-mult :: $[i, i] \Rightarrow i$

primrec
integ-of-Pls: *integ-of* (*Pls*) = $\text{\$}\# 0$
integ-of-Min: *integ-of* (*Min*) = $\text{\$-}(\text{\$}\# 1)$
integ-of-BIT: *integ-of* (*w BIT b*) = $\text{\$}\# b \text{\$+ } \text{integ-of}(w) \text{\$+ } \text{integ-of}(w)$

primrec
NCons-Pls: *NCons* (*Pls*, *b*) = *cond*(*b*, *Pls BIT b*, *Pls*)
NCons-Min: *NCons* (*Min*, *b*) = *cond*(*b*, *Min*, *Min BIT b*)
NCons-BIT: *NCons* (*w BIT c*, *b*) = *w BIT c BIT b*

primrec

bin-succ-Pls: $\text{bin-succ } (Pls) = Pls \text{ BIT } 1$

bin-succ-Min: $\text{bin-succ } (Min) = Pls$

bin-succ-BIT: $\text{bin-succ } (w \text{ BIT } b) = \text{cond}(b, \text{bin-succ}(w) \text{ BIT } 0, NCons(w, 1))$

primrec

bin-pred-Pls: $\text{bin-pred } (Pls) = Min$

bin-pred-Min: $\text{bin-pred } (Min) = Min \text{ BIT } 0$

bin-pred-BIT: $\text{bin-pred } (w \text{ BIT } b) = \text{cond}(b, NCons(w, 0), \text{bin-pred}(w) \text{ BIT } 1)$

primrec

bin-minus-Pls:

$\text{bin-minus } (Pls) = Pls$

bin-minus-Min:

$\text{bin-minus } (Min) = Pls \text{ BIT } 1$

bin-minus-BIT:

$\text{bin-minus } (w \text{ BIT } b) = \text{cond}(b, \text{bin-pred}(NCons(\text{bin-minus}(w), 0)), \text{bin-minus}(w) \text{ BIT } 0)$

primrec

bin-adder-Pls:

$\text{bin-adder } (Pls) = (\lambda w \in bin. w)$

bin-adder-Min:

$\text{bin-adder } (Min) = (\lambda w \in bin. \text{bin-pred}(w))$

bin-adder-BIT:

$\text{bin-adder } (v \text{ BIT } x) =$
 $(\lambda w \in bin.$
 $\text{bin-case } (v \text{ BIT } x, \text{bin-pred}(v \text{ BIT } x),$
 $\lambda w y. NCons(\text{bin-adder } (v) \text{ ' cond}(x \text{ and } y, \text{bin-succ}(w), w),$
 $x \text{ xor } y),$
 $w))$

definition

$\text{bin-add} :: [i, i] \Rightarrow i$ **where**

$\text{bin-add}(v, w) \equiv \text{bin-adder}(v) \text{ ' } w$

primrec

bin-mult-Pls:

$\text{bin-mult } (Pls, w) = Pls$

bin-mult-Min:

$\text{bin-mult } (Min, w) = \text{bin-minus}(w)$

bin-mult-BIT:

$\text{bin-mult } (v \text{ BIT } b, w) = \text{cond}(b, \text{bin-add}(NCons(\text{bin-mult}(v, w), 0), w),$
 $NCons(\text{bin-mult}(v, w), 0))$

syntax

$-Int0 :: i \ (\langle \#()0 \rangle)$
 $-Int1 :: i \ (\langle \#()1 \rangle)$
 $-Int2 :: i \ (\langle \#()2 \rangle)$
 $-Neg-Int1 :: i \ (\langle \#-()1 \rangle)$
 $-Neg-Int2 :: i \ (\langle \#-()2 \rangle)$

translations

$\#0 \Rightarrow CONST \text{ integ-of}(CONST \text{ Pls})$
 $\#1 \Rightarrow CONST \text{ integ-of}(CONST \text{ Pls BIT } 1)$
 $\#2 \Rightarrow CONST \text{ integ-of}(CONST \text{ Pls BIT } 1 \text{ BIT } 0)$
 $\#-1 \Rightarrow CONST \text{ integ-of}(CONST \text{ Min})$
 $\#-2 \Rightarrow CONST \text{ integ-of}(CONST \text{ Min BIT } 0)$

syntax

$-Int :: \text{num-token} \Rightarrow i \ (\langle \langle \text{open-block notation} = \langle \text{literal number} \rangle \# - \rangle \rangle 1000)$
 $-Neg-Int :: \text{num-token} \Rightarrow i \ (\langle \langle \text{open-block notation} = \langle \text{literal number} \rangle \# - - \rangle \rangle 1000)$

syntax-consts

$-Int0 \ -Int1 \ -Int2 \ -Int \ -Neg-Int1 \ -Neg-Int2 \ -Neg-Int \Rightarrow \text{integ-of}$

$\langle ML \rangle$

declare $\text{bin.intros} \ [\text{simp}, TC]$

lemma $NCons\text{-}Pls\text{-}0$: $NCons(Pls, 0) = Pls$
 $\langle \text{proof} \rangle$

lemma $NCons\text{-}Pls\text{-}1$: $NCons(Pls, 1) = Pls \text{ BIT } 1$
 $\langle \text{proof} \rangle$

lemma $NCons\text{-}Min\text{-}0$: $NCons(Min, 0) = Min \text{ BIT } 0$
 $\langle \text{proof} \rangle$

lemma $NCons\text{-}Min\text{-}1$: $NCons(Min, 1) = Min$
 $\langle \text{proof} \rangle$

lemma $NCons\text{-}BIT$: $NCons(w \text{ BIT } x, b) = w \text{ BIT } x \text{ BIT } b$
 $\langle \text{proof} \rangle$

lemmas $NCons\text{-}simps \ [\text{simp}] =$
 $NCons\text{-}Pls\text{-}0 \ NCons\text{-}Pls\text{-}1 \ NCons\text{-}Min\text{-}0 \ NCons\text{-}Min\text{-}1 \ NCons\text{-}BIT$

lemma $\text{integ-of-type} \ [TC]$: $w \in \text{bin} \Longrightarrow \text{integ-of}(w) \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *NCons-type* [TC]: $\llbracket w \in \text{bin}; b \in \text{bool} \rrbracket \implies \text{NCons}(w, b) \in \text{bin}$
 $\langle \text{proof} \rangle$

lemma *bin-succ-type* [TC]: $w \in \text{bin} \implies \text{bin-succ}(w) \in \text{bin}$
 $\langle \text{proof} \rangle$

lemma *bin-pred-type* [TC]: $w \in \text{bin} \implies \text{bin-pred}(w) \in \text{bin}$
 $\langle \text{proof} \rangle$

lemma *bin-minus-type* [TC]: $w \in \text{bin} \implies \text{bin-minus}(w) \in \text{bin}$
 $\langle \text{proof} \rangle$

lemma *bin-add-type* [rule-format]:
 $v \in \text{bin} \implies \forall w \in \text{bin}. \text{bin-add}(v, w) \in \text{bin}$
 $\langle \text{proof} \rangle$

declare *bin-add-type* [TC]

lemma *bin-mult-type* [TC]: $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket \implies \text{bin-mult}(v, w) \in \text{bin}$
 $\langle \text{proof} \rangle$

32.0.1 The Carry and Borrow Functions, *bin-succ* and *bin-pred*

lemma *integ-of-NCons* [simp]:
 $\llbracket w \in \text{bin}; b \in \text{bool} \rrbracket \implies \text{integ-of}(\text{NCons}(w, b)) = \text{integ-of}(w \text{ BIT } b)$
 $\langle \text{proof} \rangle$

lemma *integ-of-succ* [simp]:
 $w \in \text{bin} \implies \text{integ-of}(\text{bin-succ}(w)) = \$\#1 \ \$+ \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

lemma *integ-of-pred* [simp]:
 $w \in \text{bin} \implies \text{integ-of}(\text{bin-pred}(w)) = \$- \ (\$ \#1) \ \$+ \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

32.0.2 *bin-minus*: Unary Negation of Binary Integers

lemma *integ-of-minus*: $w \in \text{bin} \implies \text{integ-of}(\text{bin-minus}(w)) = \$- \text{integ-of}(w)$
 $\langle \text{proof} \rangle$

32.0.3 *bin-add*: Binary Addition

lemma *bin-add-Pls* [simp]: $w \in \text{bin} \implies \text{bin-add}(\text{Pls}, w) = w$
 $\langle \text{proof} \rangle$

lemma *bin-add-Pls-right*: $w \in \text{bin} \implies \text{bin-add}(w, \text{Pls}) = w$
 $\langle \text{proof} \rangle$

lemma *bin-add-Min* [simp]: $w \in \text{bin} \implies \text{bin-add}(\text{Min}, w) = \text{bin-pred}(w)$

$\langle proof \rangle$

lemma *bin-add-Min-right*: $w \in bin \implies bin-add(w, Min) = bin-pred(w)$
 $\langle proof \rangle$

lemma *bin-add-BIT-Pls* [simp]: $bin-add(v BIT x, Pls) = v BIT x$
 $\langle proof \rangle$

lemma *bin-add-BIT-Min* [simp]: $bin-add(v BIT x, Min) = bin-pred(v BIT x)$
 $\langle proof \rangle$

lemma *bin-add-BIT-BIT* [simp]:
 $\llbracket w \in bin; y \in bool \rrbracket$
 $\implies bin-add(v BIT x, w BIT y) =$
 $NCons(bin-add(v, cond(x \text{ and } y, bin-succ(w), w)), x \text{ xor } y)$
 $\langle proof \rangle$

lemma *integ-of-add* [rule-format]:
 $v \in bin \implies$
 $\forall w \in bin. integ-of(bin-add(v, w)) = integ-of(v) \$+ integ-of(w)$
 $\langle proof \rangle$

lemma *diff-integ-of-eq*:
 $\llbracket v \in bin; w \in bin \rrbracket$
 $\implies integ-of(v) \$- integ-of(w) = integ-of(bin-add(v, bin-minus(w)))$
 $\langle proof \rangle$

32.0.4 *bin-mult*: Binary Multiplication

lemma *integ-of-mult*:
 $\llbracket v \in bin; w \in bin \rrbracket$
 $\implies integ-of(bin-mult(v, w)) = integ-of(v) \$* integ-of(w)$
 $\langle proof \rangle$

32.1 Computations

lemma *bin-succ-1*: $bin-succ(w BIT 1) = bin-succ(w) BIT 0$
 $\langle proof \rangle$

lemma *bin-succ-0*: $bin-succ(w BIT 0) = NCons(w, 1)$
 $\langle proof \rangle$

lemma *bin-pred-1*: $bin-pred(w BIT 1) = NCons(w, 0)$
 $\langle proof \rangle$

lemma *bin-pred-0*: $bin-pred(w BIT 0) = bin-pred(w) BIT 1$
 $\langle proof \rangle$

lemma *bin-minus-1*: $\text{bin-minus}(w \text{ BIT } 1) = \text{bin-pred}(\text{NCons}(\text{bin-minus}(w), 0))$
 $\langle \text{proof} \rangle$

lemma *bin-minus-0*: $\text{bin-minus}(w \text{ BIT } 0) = \text{bin-minus}(w) \text{ BIT } 0$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-11*: $w \in \text{bin} \implies \text{bin-add}(v \text{ BIT } 1, w \text{ BIT } 1) =$
 $\text{NCons}(\text{bin-add}(v, \text{bin-succ}(w)), 0)$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-10*: $w \in \text{bin} \implies \text{bin-add}(v \text{ BIT } 1, w \text{ BIT } 0) =$
 $\text{NCons}(\text{bin-add}(v, w), 1)$
 $\langle \text{proof} \rangle$

lemma *bin-add-BIT-0*: $\llbracket w \in \text{bin}; y \in \text{bool} \rrbracket$
 $\implies \text{bin-add}(v \text{ BIT } 0, w \text{ BIT } y) = \text{NCons}(\text{bin-add}(v, w), y)$
 $\langle \text{proof} \rangle$

lemma *bin-mult-1*: $\text{bin-mult}(v \text{ BIT } 1, w) = \text{bin-add}(\text{NCons}(\text{bin-mult}(v, w), 0), w)$
 $\langle \text{proof} \rangle$

lemma *bin-mult-0*: $\text{bin-mult}(v \text{ BIT } 0, w) = \text{NCons}(\text{bin-mult}(v, w), 0)$
 $\langle \text{proof} \rangle$

lemma *int-of-0*: $\$ \# 0 = \# 0$
 $\langle \text{proof} \rangle$

lemma *int-of-succ*: $\$ \# \text{succ}(n) = \# 1 \$ + \$ \# n$
 $\langle \text{proof} \rangle$

lemma *zminus-0* [*simp*]: $\$ - \# 0 = \# 0$
 $\langle \text{proof} \rangle$

lemma *zadd-0-intify* [*simp*]: $\# 0 \$ + z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zadd-0-right-intify* [*simp*]: $z \$ + \# 0 = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-1-intify* [*simp*]: $\# 1 \$ * z = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-1-right-intify* [simp]: $z \$* \#1 = \text{intify}(z)$
 $\langle \text{proof} \rangle$

lemma *zmult-0* [simp]: $\#0 \$* z = \#0$
 $\langle \text{proof} \rangle$

lemma *zmult-0-right* [simp]: $z \$* \#0 = \#0$
 $\langle \text{proof} \rangle$

lemma *zmult-minus1* [simp]: $\#-1 \$* z = \$-z$
 $\langle \text{proof} \rangle$

lemma *zmult-minus1-right* [simp]: $z \$* \#-1 = \$-z$
 $\langle \text{proof} \rangle$

32.2 Simplification Rules for Comparison of Binary Numbers

Thanks to Norbert Voelker

lemma *eq-integ-of-eq*:
 $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies ((\text{integ-of}(v)) = \text{integ-of}(w)) \longleftrightarrow$
 $\text{iszero } (\text{integ-of } (\text{bin-add } (v, \text{bin-minus}(w))))$
 $\langle \text{proof} \rangle$

lemma *iszero-integ-of-Pls*: $\text{iszero } (\text{integ-of}(Pls))$
 $\langle \text{proof} \rangle$

lemma *nonzero-integ-of-Min*: $\neg \text{iszero } (\text{integ-of}(Min))$
 $\langle \text{proof} \rangle$

lemma *iszero-integ-of-BIT*:
 $\llbracket w \in \text{bin}; x \in \text{bool} \rrbracket$
 $\implies \text{iszero } (\text{integ-of } (w \text{ BIT } x)) \longleftrightarrow (x=0 \wedge \text{iszero } (\text{integ-of}(w)))$
 $\langle \text{proof} \rangle$

lemma *iszero-integ-of-0*:
 $w \in \text{bin} \implies \text{iszero } (\text{integ-of } (w \text{ BIT } 0)) \longleftrightarrow \text{iszero } (\text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *iszero-integ-of-1*: $w \in \text{bin} \implies \neg \text{iszero } (\text{integ-of } (w \text{ BIT } 1))$
 $\langle \text{proof} \rangle$

lemma *less-integ-of-eq-neg*:

$\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies \text{integ-of}(v) \$< \text{integ-of}(w)$
 $\iff \text{znegative}(\text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w))))$
 $\langle \text{proof} \rangle$

lemma *not-neg-integ-of-Pls*: $\neg \text{znegative}(\text{integ-of}(\text{Pls}))$

$\langle \text{proof} \rangle$

lemma *neg-integ-of-Min*: $\text{znegative}(\text{integ-of}(\text{Min}))$

$\langle \text{proof} \rangle$

lemma *neg-integ-of-BIT*:

$\llbracket w \in \text{bin}; x \in \text{bool} \rrbracket$
 $\implies \text{znegative}(\text{integ-of}(w \text{ BIT } x)) \iff \text{znegative}(\text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *le-integ-of-eq-not-less*:

$(\text{integ-of}(x) \$\leq (\text{integ-of}(w))) \iff \neg (\text{integ-of}(w) \$< (\text{integ-of}(x)))$
 $\langle \text{proof} \rangle$

declare *bin-succ-BIT* [*simp del*]

bin-pred-BIT [*simp del*]
bin-minus-BIT [*simp del*]
NCons-Pls [*simp del*]
NCons-Min [*simp del*]
bin-adder-BIT [*simp del*]
bin-mult-BIT [*simp del*]

declare *integ-of-Pls* [*simp del*] *integ-of-Min* [*simp del*] *integ-of-BIT* [*simp del*]

lemmas *bin-arith-extra-simps* =

integ-of-add [*symmetric*]
integ-of-minus [*symmetric*]
integ-of-mult [*symmetric*]
bin-succ-1 bin-succ-0
bin-pred-1 bin-pred-0
bin-minus-1 bin-minus-0
bin-add-Pls-right bin-add-Min-right
bin-add-BIT-0 bin-add-BIT-10 bin-add-BIT-11
diff-integ-of-eq
bin-mult-1 bin-mult-0 NCons-simps

lemmas *bin-arith-simps* =
 bin-pred-Pls bin-pred-Min
 bin-succ-Pls bin-succ-Min
 bin-add-Pls bin-add-Min
 bin-minus-Pls bin-minus-Min
 bin-mult-Pls bin-mult-Min
 bin-arith-extra-simps

lemmas *bin-rel-simps* =
 eq-integ-of-eq iszero-integ-of-Pls nonzero-integ-of-Min
 iszero-integ-of-0 iszero-integ-of-1
 less-integ-of-eq-neg
 not-neg-integ-of-Pls neg-integ-of-Min neg-integ-of-BIT
 le-integ-of-eq-not-less

declare *bin-arith-simps* [*simp*]
declare *bin-rel-simps* [*simp*]

lemma *add-integ-of-left* [*simp*]:
 $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies \text{integ-of}(v) \$+ (\text{integ-of}(w) \$+ z) = (\text{integ-of}(\text{bin-add}(v,w)) \$+ z)$
 $\langle \text{proof} \rangle$

lemma *mult-integ-of-left* [*simp*]:
 $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies \text{integ-of}(v) \$* (\text{integ-of}(w) \$* z) = (\text{integ-of}(\text{bin-mult}(v,w)) \$* z)$
 $\langle \text{proof} \rangle$

lemma *add-integ-of-diff1* [*simp*]:
 $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies \text{integ-of}(v) \$+ (\text{integ-of}(w) \$- c) = \text{integ-of}(\text{bin-add}(v,w)) \$- (c)$
 $\langle \text{proof} \rangle$

lemma *add-integ-of-diff2* [*simp*]:
 $\llbracket v \in \text{bin}; w \in \text{bin} \rrbracket$
 $\implies \text{integ-of}(v) \$+ (c \$- \text{integ-of}(w)) =$
 $\text{integ-of}(\text{bin-add}(v, \text{bin-minus}(w))) \$+ (c)$
 $\langle \text{proof} \rangle$

declare *int-of-0* [*simp*] *int-of-succ* [*simp*]

lemma *zdiff0 [simp]*: $\#0 \ \$- \ x = \$-x$
 $\langle proof \rangle$

lemma *zdiff0-right [simp]*: $x \ \$- \ \#0 = \text{intify}(x)$
 $\langle proof \rangle$

lemma *zdiff-self [simp]*: $x \ \$- \ x = \#0$
 $\langle proof \rangle$

lemma *znegative-iff-zless-0*: $k \in \text{int} \implies \text{znegative}(k) \longleftrightarrow k \ \$< \ \#0$
 $\langle proof \rangle$

lemma *zero-zless-imp-znegative-zminus*: $\llbracket \#0 \ \$< \ k; k \in \text{int} \rrbracket \implies \text{znegative}(\$-k)$
 $\langle proof \rangle$

lemma *zero-zle-int-of [simp]*: $\#0 \ \$\leq \ \$\# \ n$
 $\langle proof \rangle$

lemma *nat-of-0 [simp]*: $\text{nat-of}(\#0) = 0$
 $\langle proof \rangle$

lemma *nat-le-int0-lemma*: $\llbracket z \ \$\leq \ \$\#0; z \in \text{int} \rrbracket \implies \text{nat-of}(z) = 0$
 $\langle proof \rangle$

lemma *nat-le-int0*: $z \ \$\leq \ \$\#0 \implies \text{nat-of}(z) = 0$
 $\langle proof \rangle$

lemma *int-of-eq-0-imp-natify-eq-0*: $\$ \# \ n = \#0 \implies \text{natify}(n) = 0$
 $\langle proof \rangle$

lemma *nat-of-zminus-int-of*: $\text{nat-of}(\$- \$\# \ n) = 0$
 $\langle proof \rangle$

lemma *int-of-nat-of*: $\#0 \ \$\leq \ z \implies \$ \# \ \text{nat-of}(z) = \text{intify}(z)$
 $\langle proof \rangle$

declare *int-of-nat-of [simp]* *nat-of-zminus-int-of [simp]*

lemma *int-of-nat-of-if*: $\$ \# \ \text{nat-of}(z) = (\text{if } \#0 \ \$\leq \ z \text{ then } \text{intify}(z) \text{ else } \#0)$
 $\langle proof \rangle$

lemma *zless-nat-iff-int-zless*: $\llbracket m \in \text{nat}; z \in \text{int} \rrbracket \implies (m < \text{nat-of}(z)) \longleftrightarrow (\$ \# \ m \ \$< \ z)$
 $\langle proof \rangle$

lemma *zless-nat-conj-lemma*: $\$ \# 0 \ \$ < z \implies (\text{nat-of}(w) < \text{nat-of}(z)) \longleftrightarrow (w \ \$ < z)$
 $\langle \text{proof} \rangle$

lemma *zless-nat-conj*: $(\text{nat-of}(w) < \text{nat-of}(z)) \longleftrightarrow (\$ \# 0 \ \$ < z \wedge w \ \$ < z)$
 $\langle \text{proof} \rangle$

lemma *integ-of-minus-reorient* [*simp*]:
 $(\text{integ-of}(w) = \$ - x) \longleftrightarrow (\$ - x = \text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *integ-of-add-reorient* [*simp*]:
 $(\text{integ-of}(w) = x \ \$ + y) \longleftrightarrow (x \ \$ + y = \text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *integ-of-diff-reorient* [*simp*]:
 $(\text{integ-of}(w) = x \ \$ - y) \longleftrightarrow (x \ \$ - y = \text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemma *integ-of-mult-reorient* [*simp*]:
 $(\text{integ-of}(w) = x \ \$ * y) \longleftrightarrow (x \ \$ * y = \text{integ-of}(w))$
 $\langle \text{proof} \rangle$

lemmas [*simp*] =
zminus-equation [**where** $y = \text{integ-of}(w)$]
equation-zminus [**where** $x = \text{integ-of}(w)$]
for w

lemmas [*iff*] =
zminus-zless [**where** $y = \text{integ-of}(w)$]
zless-zminus [**where** $x = \text{integ-of}(w)$]
for w

lemmas [*iff*] =
zminus-zle [**where** $y = \text{integ-of}(w)$]
zle-zminus [**where** $x = \text{integ-of}(w)$]
for w

lemmas [*simp*] =
Let-def [**where** $s = \text{integ-of}(w)$] **for** w

lemma *zless-iff-zdiff-zless-0*: $(x \text{ \$< } y) \longleftrightarrow (x\$-y \text{ \$< } \#0)$
 ⟨proof⟩

lemma *eq-iff-zdiff-eq-0*: $\llbracket x \in \text{int}; y \in \text{int} \rrbracket \implies (x = y) \longleftrightarrow (x\$-y = \#0)$
 ⟨proof⟩

lemma *zle-iff-zdiff-zle-0*: $(x \text{ \$≤ } y) \longleftrightarrow (x\$-y \text{ \$≤ } \#0)$
 ⟨proof⟩

lemma *left-zadd-zmult-distrib*: $i\$*u \text{ \$+ } (j\$*u \text{ \$+ } k) = (i\$+j)\$*u \text{ \$+ } k$
 ⟨proof⟩

lemma *eq-add-iff1*: $(i\$*u \text{ \$+ } m = j\$*u \text{ \$+ } n) \longleftrightarrow ((i\$-j)\$*u \text{ \$+ } m = \text{intify}(n))$
 ⟨proof⟩

lemma *eq-add-iff2*: $(i\$*u \text{ \$+ } m = j\$*u \text{ \$+ } n) \longleftrightarrow (\text{intify}(m) = (j\$-i)\$*u \text{ \$+ } n)$
 ⟨proof⟩

context fixes $n :: i$
begin

lemmas *rel-iff-rel-0-rls* =
zless-iff-zdiff-zless-0 [where $y = u \text{ \$+ } v$]
eq-iff-zdiff-eq-0 [where $y = u \text{ \$+ } v$]
zle-iff-zdiff-zle-0 [where $y = u \text{ \$+ } v$]
zless-iff-zdiff-zless-0 [where $y = n$]
eq-iff-zdiff-eq-0 [where $y = n$]
zle-iff-zdiff-zle-0 [where $y = n$]
for $u \ v$

lemma *less-add-iff1*: $(i\$*u \text{ \$+ } m \text{ \$< } j\$*u \text{ \$+ } n) \longleftrightarrow ((i\$-j)\$*u \text{ \$+ } m \text{ \$< } n)$
 ⟨proof⟩

lemma *less-add-iff2*: $(i\$*u \text{ \$+ } m \text{ \$< } j\$*u \text{ \$+ } n) \longleftrightarrow (m \text{ \$< } (j\$-i)\$*u \text{ \$+ } n)$
 ⟨proof⟩

end

lemma *le-add-iff1*: $(i\$*u \text{ \$+ } m \text{ \$≤ } j\$*u \text{ \$+ } n) \longleftrightarrow ((i\$-j)\$*u \text{ \$+ } m \text{ \$≤ } n)$
 ⟨proof⟩

lemma *le-add-iff2*: $(i * u + m \leq j * u + n) \longleftrightarrow (m \leq (j - i) * u + n)$
 $\langle proof \rangle$

$\langle ML \rangle$

32.2.1 Examples

combine-numerals-prod (products of separate literals)

lemma $\#5 * x * \#3 = y \langle proof \rangle$

schematic-goal $y2 + ?x42 = y + y2 \langle proof \rangle$

lemma $oo : int \implies l + (l + \#2) + oo = oo \langle proof \rangle$

lemma $\#9 * x + y = x * \#23 + z \langle proof \rangle$

lemma $y + x = x + z \langle proof \rangle$

lemma $x : int \implies x + y + z = x + z \langle proof \rangle$

lemma $x : int \implies y + (z + x) = z + x \langle proof \rangle$

lemma $z : int \implies x + y + z = (z + y) + (x + w) \langle proof \rangle$

lemma $z : int \implies x * y + z = (z + y) + (y * x + w) \langle proof \rangle$

lemma $\#-3 * x + y \leq x * \#2 + z \langle proof \rangle$

lemma $y + x \leq x + z \langle proof \rangle$

lemma $x + y + z \leq x + z \langle proof \rangle$

lemma $y + (z + x) < z + x \langle proof \rangle$

lemma $x + y + z < (z + y) + (x + w) \langle proof \rangle$

lemma $x * y + z < (z + y) + (y * x + w) \langle proof \rangle$

lemma $l + \#2 + \#2 + \#2 + (l + \#2) + (oo + \#2) = uu \langle proof \rangle$

lemma $u : int \implies \#2 * u = u \langle proof \rangle$

lemma $(i + j + \#12 + k) - \#15 = y \langle proof \rangle$

lemma $(i + j + \#12 + k) - \#5 = y \langle proof \rangle$

lemma $y - b < b \langle proof \rangle$

lemma $y - (\#3 * b + c) < b - \#2 * c \langle proof \rangle$

lemma $(\#2 * x - (u * v) + y) - v * \#3 * u = w \langle proof \rangle$

lemma $(\#2 * x * u * v + (u * v) * \#4 + y) - v * u * \#4 = w \langle proof \rangle$

lemma $(\#2 * x * u * v + (u * v) * \#4 + y) - v * u = w \langle proof \rangle$

lemma $u * v - (x * u * v + (u * v) * \#4 + y) = w \langle proof \rangle$

lemma $(i + j + \#12 + k) = u + \#15 + y \langle proof \rangle$

lemma $(i + j * \#2 + \#12 + k) = j + \#5 + y \langle proof \rangle$

lemma $\#2 * y + \#3 * z + \#6 * w + \#2 * y + \#3 * z + \#2 * u = \#2 * y' + \#3 * z' + \#6 * w' + \#2 * y' + \#3 * z' + u + vv$

$\langle proof \rangle$

lemma $a + -(b+c) + b = d \langle proof \rangle$

lemma $a + -(b+c) - b = d \langle proof \rangle$

negative numerals

lemma $(i + j + \#-2 + k) - (u + \#5 + y) = zz \langle proof \rangle$

lemma $(i + j + \#-3 + k) < u + \#5 + y \langle proof \rangle$

lemma $(i + j + \#3 + k) < u + \#-6 + y \langle proof \rangle$

lemma $(i + j + \#-12 + k) - \#15 = y \langle proof \rangle$

lemma $(i + j + \#12 + k) - \#-15 = y \langle proof \rangle$

lemma $(i + j + \#-12 + k) - \#-15 = y \langle proof \rangle$

Multiplying separated numerals

lemma $\#6 * (\#x * \#2) = uu \langle proof \rangle$

lemma $\#4 * (\#x * \#x) * (\#2 * \#x) = uu \langle proof \rangle$

end

33 The Division Operators Div and Mod

theory *IntDiv*

imports *Bin OrderArith*

begin

definition

$quorem :: [i,i] \Rightarrow o$ **where**

$quorem \equiv \lambda \langle a,b \rangle \langle q,r \rangle.$

$a = b * q + r \wedge$

$(\#0 < b \wedge \#0 \leq r \wedge r < b \mid \neg(\#0 < b) \wedge b < r \wedge r \leq \#0)$

definition

$adjust :: [i,i] \Rightarrow i$ **where**

$adjust(b) \equiv \lambda \langle q,r \rangle. \text{ if } \#0 \leq r - b \text{ then } \langle \#2 * q + \#1, r - b \rangle$

$\text{ else } \langle \#2 * q, r \rangle$

definition

$posDivAlg :: i \Rightarrow i$ **where**

$posDivAlg(ab) \equiv$

$wfrec(measure(int*int, \lambda \langle a,b \rangle. \text{ nat-of } (a - b + \#1)),$

$ab,$

$\lambda \langle a,b \rangle f. \text{ if } (a < b \mid b \leq \#0) \text{ then } \langle \#0, a \rangle$

$\text{ else } adjust(b, f \text{ ` } <a, \#2 * b >))$

definition

$negDivAlg :: i \Rightarrow i$ **where**

$negDivAlg(ab) \equiv$
 $wfrec(measure(int*int, \lambda\langle a,b \rangle. nat-of (\$- a \$- b)),$
 $ab,$
 $\lambda\langle a,b \rangle f. \text{ if } (\#0 \$\leq a\$+b \mid b\$ \leq \#0) \text{ then } <\#-1, a\$+b>$
 $\text{ else } adjust(b, f \text{ ` } <a, \#2\$*b>))$

definition

$negateSnd :: i \Rightarrow i$ **where**

$negateSnd \equiv \lambda\langle q,r \rangle. <q, \$-r>$

definition

$divAlg :: i \Rightarrow i$ **where**

$divAlg \equiv$
 $\lambda\langle a,b \rangle. \text{ if } \#0 \$\leq a \text{ then}$
 $\text{ if } \#0 \$\leq b \text{ then } posDivAlg (\langle a,b \rangle)$
 $\text{ else if } a=\#0 \text{ then } <\#0, \#0>$
 $\text{ else } negateSnd (negDivAlg (<\$-a, \$-b>))$
 else
 $\text{ if } \#0\$<b \text{ then } negDivAlg (\langle a,b \rangle)$
 $\text{ else } negateSnd (posDivAlg (<\$-a, \$-b>))$

definition

$zdiv :: [i,i] \Rightarrow i$ **(infixl <zdiv> 70) where**
 $a \text{ zdiv } b \equiv fst (divAlg (<intify(a), intify(b)>))$

definition

$zmod :: [i,i] \Rightarrow i$ **(infixl <zmod> 70) where**
 $a \text{ zmod } b \equiv snd (divAlg (<intify(a), intify(b)>))$

lemma $zpos-add-zpos-imp-zpos$: $\llbracket \#0 \$< x; \#0 \$< y \rrbracket \implies \#0 \$< x \$+ y$
 $\langle proof \rangle$

lemma $zpos-add-zpos-imp-zpos$: $\llbracket \#0 \$\leq x; \#0 \$\leq y \rrbracket \implies \#0 \$\leq x \$+ y$
 $\langle proof \rangle$

lemma $zneg-add-zneg-imp-zneg$: $\llbracket x \$< \#0; y \$< \#0 \rrbracket \implies x \$+ y \$< \#0$
 $\langle proof \rangle$

lemma *zneg-or-0-add-zneg-or-0-imp-zneg-or-0*:

$\llbracket x \leq \#0; y \leq \#0 \rrbracket \implies x + y \leq \#0$
 $\langle proof \rangle$

lemma *zero-lt-zmagnitude*: $\llbracket \#0 < k; k \in int \rrbracket \implies 0 < zmagnitude(k)$

$\langle proof \rangle$

lemma *zless-add-succ-iff*:

$(w < z + \# succ(m)) \longleftrightarrow (w < z + \#m \mid intify(w) = z + \#m)$
 $\langle proof \rangle$

lemma *zadd-succ-lemma*:

$z \in int \implies (w + \# succ(m) \leq z) \longleftrightarrow (w + \#m < z)$
 $\langle proof \rangle$

lemma *zadd-succ-zle-iff*: $(w + \# succ(m) \leq z) \longleftrightarrow (w + \#m < z)$

$\langle proof \rangle$

lemma *zless-add1-iff-zle*: $(w < z + \#1) \longleftrightarrow (w \leq z)$

$\langle proof \rangle$

lemma *add1-zle-iff*: $(w + \#1 \leq z) \longleftrightarrow (w < z)$

$\langle proof \rangle$

lemma *add1-left-zle-iff*: $(\#1 + w \leq z) \longleftrightarrow (w < z)$

$\langle proof \rangle$

lemma *zmult-mono-lemma*: $k \in nat \implies i \leq j \implies i * \#k \leq j * \#k$

$\langle proof \rangle$

lemma *zmult-zle-mono1*: $\llbracket i \leq j; \#0 \leq k \rrbracket \implies i * k \leq j * k$

$\langle proof \rangle$

lemma *zmult-zle-mono1-neg*: $\llbracket i \leq j; k \leq \#0 \rrbracket \implies j * k \leq i * k$

$\langle proof \rangle$

lemma *zmult-zle-mono2*: $\llbracket i \leq j; \#0 \leq k \rrbracket \implies k * i \leq k * j$

$\langle proof \rangle$

lemma *zmult-zle-mono2-neg*: $\llbracket i \leq j; k \leq \#0 \rrbracket \implies k * j \leq k * i$
 $\langle proof \rangle$

lemma *zmult-zle-mono*:
 $\llbracket i \leq j; k \leq l; \#0 \leq j; \#0 \leq k \rrbracket \implies i * k \leq j * l$
 $\langle proof \rangle$

lemma *zmult-zless-mono2-lemma* [rule-format]:
 $\llbracket i < j; k \in \text{nat} \rrbracket \implies 0 < k \longrightarrow \#k * i < \#k * j$
 $\langle proof \rangle$

lemma *zmult-zless-mono2*: $\llbracket i < j; \#0 < k \rrbracket \implies k * i < k * j$
 $\langle proof \rangle$

lemma *zmult-zless-mono1*: $\llbracket i < j; \#0 < k \rrbracket \implies i * k < j * k$
 $\langle proof \rangle$

lemma *zmult-zless-mono*:
 $\llbracket i < j; k < l; \#0 < j; \#0 < k \rrbracket \implies i * k < j * l$
 $\langle proof \rangle$

lemma *zmult-zless-mono1-neg*: $\llbracket i < j; k < \#0 \rrbracket \implies j * k < i * k$
 $\langle proof \rangle$

lemma *zmult-zless-mono2-neg*: $\llbracket i < j; k < \#0 \rrbracket \implies k * j < k * i$
 $\langle proof \rangle$

lemma *zmult-eq-lemma*:
 $\llbracket m \in \text{int}; n \in \text{int} \rrbracket \implies (m = \#0 \mid n = \#0) \longleftrightarrow (m * n = \#0)$
 $\langle proof \rangle$

lemma *zmult-eq-0-iff* [iff]: $(m * n = \#0) \longleftrightarrow (\text{intify}(m) = \#0 \mid \text{intify}(n) = \#0)$
 $\langle proof \rangle$

lemma *zmult-zless-lemma*:
 $\llbracket k \in \text{int}; m \in \text{int}; n \in \text{int} \rrbracket$
 $\implies (m * k < n * k) \longleftrightarrow ((\#0 < k \wedge m < n) \mid (k < \#0 \wedge n < m))$
 $\langle proof \rangle$

lemma *zmult-zless-cancel2*:

$$(m\$*k \$< n\$*k) \longleftrightarrow ((\#0 \$< k \wedge m\$<n) \mid (k \$< \#0 \wedge n\$<m))$$

<proof>

lemma *zmult-zless-cancel1*:

$$(k\$*m \$< k\$*n) \longleftrightarrow ((\#0 \$< k \wedge m\$<n) \mid (k \$< \#0 \wedge n\$<m))$$

<proof>

lemma *zmult-zle-cancel2*:

$$(m\$*k \$\leq n\$*k) \longleftrightarrow ((\#0 \$< k \longrightarrow m\$ \leq n) \wedge (k \$< \#0 \longrightarrow n\$ \leq m))$$

<proof>

lemma *zmult-zle-cancel1*:

$$(k\$*m \$\leq k\$*n) \longleftrightarrow ((\#0 \$< k \longrightarrow m\$ \leq n) \wedge (k \$< \#0 \longrightarrow n\$ \leq m))$$

<proof>

lemma *int-eq-iff-zle*: $\llbracket m \in \text{int}; n \in \text{int} \rrbracket \implies m=n \longleftrightarrow (m \$\leq n \wedge n \$\leq m)$

<proof>

lemma *zmult-cancel2-lemma*:

$$\llbracket k \in \text{int}; m \in \text{int}; n \in \text{int} \rrbracket \implies (m\$*k = n\$*k) \longleftrightarrow (k=\#0 \mid m=n)$$

<proof>

lemma *zmult-cancel2 [simp]*:

$$(m\$*k = n\$*k) \longleftrightarrow (\text{intify}(k) = \#0 \mid \text{intify}(m) = \text{intify}(n))$$

<proof>

lemma *zmult-cancel1 [simp]*:

$$(k\$*m = k\$*n) \longleftrightarrow (\text{intify}(k) = \#0 \mid \text{intify}(m) = \text{intify}(n))$$

<proof>

33.1 Uniqueness and monotonicity of quotients and remainders

lemma *unique-quotient-lemma*:

$$\llbracket b\$*q' \$+ r' \$\leq b\$*q \$+ r; \#0 \$\leq r'; \#0 \$< b; r \$< b \rrbracket$$

$$\implies q' \$\leq q$$

<proof>

lemma *unique-quotient-lemma-neg*:

$$\llbracket b\$*q' \$+ r' \$\leq b\$*q \$+ r; r \$\leq \#0; b \$< \#0; b \$< r \rrbracket$$

$$\implies q \$\leq q'$$

<proof>

lemma *unique-quotient*:

$$\llbracket \text{quorem}(\langle a, b \rangle, \langle q, r \rangle); \text{quorem}(\langle a, b \rangle, \langle q', r' \rangle); b \in \text{int}; b \neq \#0; q \in \text{int}; q' \in \text{int} \rrbracket \implies q = q'$$

$\langle proof \rangle$

lemma *unique-remainder*:

$\llbracket quorem(\langle a, b \rangle, \langle q, r \rangle); quorem(\langle a, b \rangle, \langle q', r' \rangle); b \in int; b \neq \#0;$
 $q \in int; q' \in int;$
 $r \in int; r' \in int \rrbracket \implies r = r'$

$\langle proof \rangle$

33.2 Correctness of posDivAlg, the Division Algorithm for $a \geq 0$ and $b > 0$

lemma *adjust-eq [simp]*:

$adjust(b, \langle q, r \rangle) = (let\ diff = r \$ - b\ in$
 $if\ \#0 \$ \leq diff\ then\ \langle \#2 \$ * q \$ + \#1, diff \rangle$
 $else\ \langle \#2 \$ * q, r \rangle)$

$\langle proof \rangle$

lemma *posDivAlg-termination*:

$\llbracket \#0 \$ < b; \neg a \$ < b \rrbracket$
 $\implies nat-of(a \$ - \#2 \$ * b \$ + \#1) < nat-of(a \$ - b \$ + \#1)$

$\langle proof \rangle$

lemmas *posDivAlg-unfold = def-wfrec [OF posDivAlg-def wf-measure]*

lemma *posDivAlg-eqn*:

$\llbracket \#0 \$ < b; a \in int; b \in int \rrbracket \implies$
 $posDivAlg(\langle a, b \rangle) =$
 $(if\ a \$ < b\ then\ \langle \#0, a \rangle\ else\ adjust(b, posDivAlg(\langle a, \#2 \$ * b \rangle)))$

$\langle proof \rangle$

lemma *posDivAlg-induct-lemma [rule-format]*:

assumes *prem*:

$\bigwedge a\ b. \llbracket a \in int; b \in int;$
 $\neg(a \$ < b \mid b \$ \leq \#0) \longrightarrow P(\langle a, \#2 \$ * b \rangle) \rrbracket \implies P(\langle a, b \rangle)$

shows $\langle u, v \rangle \in int * int \implies P(\langle u, v \rangle)$

$\langle proof \rangle$

lemma *posDivAlg-induct [consumes 2]*:

assumes *u-int*: $u \in int$

and *v-int*: $v \in int$

and *ih*: $\bigwedge a\ b. \llbracket a \in int; b \in int;$

$\neg(a \$ < b \mid b \$ \leq \#0) \longrightarrow P(a, \#2 \$ * b) \rrbracket \implies P(a, b)$

shows $P(u, v)$

$\langle proof \rangle$

lemma *intify-eq-0-iff-zle*: $intify(m) = \#0 \longleftrightarrow (m \$ \leq \#0 \wedge \#0 \$ \leq m)$

$\langle proof \rangle$

33.3 Some convenient biconditionals for products of signs

lemma *zmult-pos*: $\llbracket \#0 \$< i; \#0 \$< j \rrbracket \implies \#0 \$< i \$* j$
 $\langle proof \rangle$

lemma *zmult-neg*: $\llbracket i \$< \#0; j \$< \#0 \rrbracket \implies \#0 \$< i \$* j$
 $\langle proof \rangle$

lemma *zmult-pos-neg*: $\llbracket \#0 \$< i; j \$< \#0 \rrbracket \implies i \$* j \$< \#0$
 $\langle proof \rangle$

lemma *int-0-less-lemma*:
 $\llbracket x \in int; y \in int \rrbracket$
 $\implies (\#0 \$< x \$* y) \longleftrightarrow (\#0 \$< x \wedge \#0 \$< y \mid x \$< \#0 \wedge y \$< \#0)$
 $\langle proof \rangle$

lemma *int-0-less-mult-iff*:
 $(\#0 \$< x \$* y) \longleftrightarrow (\#0 \$< x \wedge \#0 \$< y \mid x \$< \#0 \wedge y \$< \#0)$
 $\langle proof \rangle$

lemma *int-0-le-lemma*:
 $\llbracket x \in int; y \in int \rrbracket$
 $\implies (\#0 \$\leq x \$* y) \longleftrightarrow (\#0 \$\leq x \wedge \#0 \$\leq y \mid x \$\leq \#0 \wedge y \$\leq \#0)$
 $\langle proof \rangle$

lemma *int-0-le-mult-iff*:
 $(\#0 \$\leq x \$* y) \longleftrightarrow ((\#0 \$\leq x \wedge \#0 \$\leq y) \mid (x \$\leq \#0 \wedge y \$\leq \#0))$
 $\langle proof \rangle$

lemma *zmult-less-0-iff*:
 $(x \$* y \$< \#0) \longleftrightarrow (\#0 \$< x \wedge y \$< \#0 \mid x \$< \#0 \wedge \#0 \$< y)$
 $\langle proof \rangle$

lemma *zmult-le-0-iff*:
 $(x \$* y \$\leq \#0) \longleftrightarrow (\#0 \$\leq x \wedge y \$\leq \#0 \mid x \$\leq \#0 \wedge \#0 \$\leq y)$
 $\langle proof \rangle$

lemma *posDivAlg-type* [*rule-format*]:
 $\llbracket a \in int; b \in int \rrbracket \implies posDivAlg(\langle a, b \rangle) \in int * int$
 $\langle proof \rangle$

lemma *posDivAlg-correct* [rule-format]:

$$\llbracket a \in \text{int}; b \in \text{int} \rrbracket \implies \#0 \leq a \longrightarrow \#0 < b \longrightarrow \text{quorem}(\langle a, b \rangle, \text{posDivAlg}(\langle a, b \rangle))$$

$$\langle \text{proof} \rangle$$

33.4 Correctness of negDivAlg, the division algorithm for $a < 0$ and $b > 0$

lemma *negDivAlg-termination*:

$$\llbracket \#0 < b; a \ \$+ \ b \ \$< \ #0 \rrbracket \implies \text{nat-of}(\$- \ a \ \$- \ \#2 \ \$* \ b) < \text{nat-of}(\$- \ a \ \$- \ b)$$

$$\langle \text{proof} \rangle$$

lemmas *negDivAlg-unfold* = *def-wfrec* [OF *negDivAlg-def* *wf-measure*]

lemma *negDivAlg-eqn*:

$$\llbracket \#0 < b; a \in \text{int}; b \in \text{int} \rrbracket \implies$$

$$\text{negDivAlg}(\langle a, b \rangle) =$$

$$(\text{if } \#0 \leq a\$+b \text{ then } <\#-1, a\$+b>$$

$$\text{else } \text{adjust}(b, \text{negDivAlg}(<a, \#2\$*b>)))$$

$$\langle \text{proof} \rangle$$

lemma *negDivAlg-induct-lemma* [rule-format]:
assumes *prem*:

$$\bigwedge a \ b. \llbracket a \in \text{int}; b \in \text{int};$$

$$\neg (\#0 \leq a \ \$+ \ b \mid b \ \$\leq \ #0) \longrightarrow P(<a, \#2 \ \$* \ b>)\rrbracket$$

$$\implies P(\langle a, b \rangle)$$
shows $\langle u, v \rangle \in \text{int} * \text{int} \implies P(\langle u, v \rangle)$

$$\langle \text{proof} \rangle$$

lemma *negDivAlg-induct* [consumes 2]:
assumes *u-int*: $u \in \text{int}$
and *v-int*: $v \in \text{int}$
and *ih*: $\bigwedge a \ b. \llbracket a \in \text{int}; b \in \text{int};$

$$\neg (\#0 \leq a \ \$+ \ b \mid b \ \$\leq \ #0) \longrightarrow P(a, \#2 \ \$* \ b)\rrbracket$$

$$\implies P(a, b)$$
shows $P(u, v)$

$$\langle \text{proof} \rangle$$

lemma *negDivAlg-type*:

$$\llbracket a \in \text{int}; b \in \text{int} \rrbracket \implies \text{negDivAlg}(\langle a, b \rangle) \in \text{int} * \text{int}$$

$$\langle \text{proof} \rangle$$

lemma *negDivAlg-correct* [rule-format]:

$$\llbracket a \in \text{int}; b \in \text{int} \rrbracket$$

$\implies a \text{ \$} < \#0 \longrightarrow \#0 \text{ \$} < b \longrightarrow \text{quorem } (\langle a, b \rangle, \text{negDivAlg}(\langle a, b \rangle))$
 $\langle \text{proof} \rangle$

33.5 Existence shown by proving the division algorithm to be correct

lemma *quorem-0*: $\llbracket b \neq \#0; \ b \in \text{int} \rrbracket \implies \text{quorem } (<\#0, b>, <\#0, \#0>)$
 $\langle \text{proof} \rangle$

lemma *posDivAlg-zero-divisor*: $\text{posDivAlg}(<a, \#0>) = <\#0, a>$
 $\langle \text{proof} \rangle$

lemma *posDivAlg-0* [simp]: $\text{posDivAlg } (<\#0, b>) = <\#0, \#0>$
 $\langle \text{proof} \rangle$

lemma *linear-arith-lemma*: $\neg (\#0 \text{ \$} \leq \#-1 \text{ \$} + b) \implies (b \text{ \$} \leq \#0)$
 $\langle \text{proof} \rangle$

lemma *negDivAlg-minus1* [simp]: $\text{negDivAlg } (<\#-1, b>) = <\#-1, b \text{ \$} - \#1>$
 $\langle \text{proof} \rangle$

lemma *negateSnd-eq* [simp]: $\text{negateSnd } (\langle q, r \rangle) = \langle q, \text{\$} - r \rangle$
 $\langle \text{proof} \rangle$

lemma *negateSnd-type*: $qr \in \text{int} * \text{int} \implies \text{negateSnd } (qr) \in \text{int} * \text{int}$
 $\langle \text{proof} \rangle$

lemma *quorem-neg*:
 $\llbracket \text{quorem } (<\text{\$} - a, \text{\$} - b>, qr); \ a \in \text{int}; \ b \in \text{int}; \ qr \in \text{int} * \text{int} \rrbracket$
 $\implies \text{quorem } (\langle a, b \rangle, \text{negateSnd}(qr))$
 $\langle \text{proof} \rangle$

lemma *divAlg-correct*:
 $\llbracket b \neq \#0; \ a \in \text{int}; \ b \in \text{int} \rrbracket \implies \text{quorem } (\langle a, b \rangle, \text{divAlg}(\langle a, b \rangle))$
 $\langle \text{proof} \rangle$

lemma *divAlg-type*: $\llbracket a \in \text{int}; \ b \in \text{int} \rrbracket \implies \text{divAlg}(\langle a, b \rangle) \in \text{int} * \text{int}$
 $\langle \text{proof} \rangle$

lemma *zdiv-intify1* [simp]: $\text{intify}(x) \text{ zdiv } y = x \text{ zdiv } y$
 $\langle \text{proof} \rangle$

lemma *zdiv-intify2* [simp]: $x \text{ zdiv } \text{intify}(y) = x \text{ zdiv } y$
 $\langle \text{proof} \rangle$

lemma *zdiv-type* [*iff*, *TC*]: $z \text{ zdiv } w \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *zmod-intify1* [*simp*]: $\text{intify}(x) \text{ zmod } y = x \text{ zmod } y$
 $\langle \text{proof} \rangle$

lemma *zmod-intify2* [*simp*]: $x \text{ zmod } \text{intify}(y) = x \text{ zmod } y$
 $\langle \text{proof} \rangle$

lemma *zmod-type* [*iff*, *TC*]: $z \text{ zmod } w \in \text{int}$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-ZDIV*: $a \text{ zdiv } \#0 = \#0$
 $\langle \text{proof} \rangle$

lemma *DIVISION-BY-ZERO-ZMOD*: $a \text{ zmod } \#0 = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *raw-zmod-zdiv-equality*:
 $\llbracket a \in \text{int}; b \in \text{int} \rrbracket \implies a = b \$* (a \text{ zdiv } b) \$+ (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zdiv-equality*: $\text{intify}(a) = b \$* (a \text{ zdiv } b) \$+ (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *pos-mod*: $\#0 \$< b \implies \#0 \$\leq a \text{ zmod } b \wedge a \text{ zmod } b \$< b$
 $\langle \text{proof} \rangle$

lemmas *pos-mod-sign* = *pos-mod* [*THEN conjunct1*]
and *pos-mod-bound* = *pos-mod* [*THEN conjunct2*]

lemma *neg-mod*: $b \$< \#0 \implies a \text{ zmod } b \$\leq \#0 \wedge b \$< a \text{ zmod } b$
 $\langle \text{proof} \rangle$

lemmas *neg-mod-sign* = *neg-mod* [*THEN conjunct1*]
and *neg-mod-bound* = *neg-mod* [*THEN conjunct2*]

lemma *quorem-div-mod*:
 $\llbracket b \neq \#0; a \in \text{int}; b \in \text{int} \rrbracket$

$\implies \text{quorem}(\langle a, b \rangle, \langle a \text{ zdiv } b, a \text{ zmod } b \rangle)$
 $\langle \text{proof} \rangle$

lemma *quorem-div*:

$\llbracket \text{quorem}(\langle a, b \rangle, \langle q, r \rangle); b \neq \#0; a \in \text{int}; b \in \text{int}; q \in \text{int} \rrbracket$
 $\implies a \text{ zdiv } b = q$
 $\langle \text{proof} \rangle$

lemma *quorem-mod*:

$\llbracket \text{quorem}(\langle a, b \rangle, \langle q, r \rangle); b \neq \#0; a \in \text{int}; b \in \text{int}; q \in \text{int}; r \in \text{int} \rrbracket$
 $\implies a \text{ zmod } b = r$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-pos-trivial-raw*:

$\llbracket a \in \text{int}; b \in \text{int}; \#0 \leq a; a < b \rrbracket \implies a \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-pos-trivial*: $\llbracket \#0 \leq a; a < b \rrbracket \implies a \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-neg-trivial-raw*:

$\llbracket a \in \text{int}; b \in \text{int}; a \leq \#0; b < a \rrbracket \implies a \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-neg-trivial*: $\llbracket a \leq \#0; b < a \rrbracket \implies a \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zadd-le-0-lemma*: $\llbracket a + b \leq \#0; \#0 < a; \#0 < b \rrbracket \implies \text{False}$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-neg-trivial-raw*:

$\llbracket a \in \text{int}; b \in \text{int}; \#0 < a; a + b \leq \#0 \rrbracket \implies a \text{ zdiv } b = \#-1$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-neg-trivial*: $\llbracket \#0 < a; a + b \leq \#0 \rrbracket \implies a \text{ zdiv } b = \#-1$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-pos-trivial-raw*:

$\llbracket a \in \text{int}; b \in \text{int}; \#0 \leq a; a < b \rrbracket \implies a \text{ zmod } b = a$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-pos-trivial*: $\llbracket \#0 \leq a; a < b \rrbracket \implies a \text{ zmod } b = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zmod-neg-neg-trivial-raw*:

$\llbracket a \in \text{int}; b \in \text{int}; a \leq \#0; b < a \rrbracket \implies a \text{ zmod } b = a$
 $\langle \text{proof} \rangle$

lemma *zmod-neg-neg-trivial*: $\llbracket a \leq \#0; b < a \rrbracket \implies a \text{ zmod } b = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-neg-trivial-raw*:
 $\llbracket a \in \text{int}; b \in \text{int}; \#0 < a; a\$+b \leq \#0 \rrbracket \implies a \text{ zmod } b = a\$+b$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-neg-trivial*: $\llbracket \#0 < a; a\$+b \leq \#0 \rrbracket \implies a \text{ zmod } b = a\$+b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zminus-zminus-raw*:
 $\llbracket a \in \text{int}; b \in \text{int} \rrbracket \implies (\$-a) \text{ zdiv } (\$-b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zminus-zminus [simp]*: $(\$-a) \text{ zdiv } (\$-b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus-zminus-raw*:
 $\llbracket a \in \text{int}; b \in \text{int} \rrbracket \implies (\$-a) \text{ zmod } (\$-b) = \$- (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zminus-zminus [simp]*: $(\$-a) \text{ zmod } (\$-b) = \$- (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

33.6 division of a number by itself

lemma *self-quotient-aux1*: $\llbracket \#0 < a; a = r \$+ a\$*q; r < a \rrbracket \implies \#1 \leq q$
 $\langle \text{proof} \rangle$

lemma *self-quotient-aux2*: $\llbracket \#0 < a; a = r \$+ a\$*q; \#0 \leq r \rrbracket \implies q \leq \#1$
 $\langle \text{proof} \rangle$

lemma *self-quotient*:
 $\llbracket \text{quorem}(\langle a, a \rangle, \langle q, r \rangle); a \in \text{int}; q \in \text{int}; a \neq \#0 \rrbracket \implies q = \#1$
 $\langle \text{proof} \rangle$

lemma *self-remainder*:
 $\llbracket \text{quorem}(\langle a, a \rangle, \langle q, r \rangle); a \in \text{int}; q \in \text{int}; r \in \text{int}; a \neq \#0 \rrbracket \implies r = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-self-raw*: $\llbracket a \neq \#0; a \in \text{int} \rrbracket \implies a \text{ zdiv } a = \#1$
 $\langle \text{proof} \rangle$

lemma *zdiv-self [simp]*: $\text{intify}(a) \neq \#0 \implies a \text{ zdiv } a = \#1$
 $\langle \text{proof} \rangle$

lemma *zmod-self-raw*: $a \in \text{int} \implies a \text{ zmod } a = \#0$
 $\langle \text{proof} \rangle$

lemma *zmod-self [simp]*: $a \text{ zmod } a = \#0$
 $\langle \text{proof} \rangle$

33.7 Computation of division and remainder

lemma *zdiv-zero [simp]*: $\#0 \text{ zdiv } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-eq-minus1*: $\#0 \text{ \$< } b \implies \#-1 \text{ zdiv } b = \#-1$
 $\langle \text{proof} \rangle$

lemma *zmod-zero [simp]*: $\#0 \text{ zmod } b = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-minus1*: $\#0 \text{ \$< } b \implies \#-1 \text{ zdiv } b = \#-1$
 $\langle \text{proof} \rangle$

lemma *zmod-minus1*: $\#0 \text{ \$< } b \implies \#-1 \text{ zmod } b = b \text{ \$- } \#1$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-pos*: $\llbracket \#0 \text{ \$< } a; \#0 \text{ \$}\leq b \rrbracket$
 $\implies a \text{ zdiv } b = \text{fst } (\text{posDivAlg}(<\text{intify}(a), \text{intify}(b)>))$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-pos*:
 $\llbracket \#0 \text{ \$< } a; \#0 \text{ \$}\leq b \rrbracket$
 $\implies a \text{ zmod } b = \text{snd } (\text{posDivAlg}(<\text{intify}(a), \text{intify}(b)>))$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-pos*:
 $\llbracket a \text{ \$< } \#0; \#0 \text{ \$< } b \rrbracket$
 $\implies a \text{ zdiv } b = \text{fst } (\text{negDivAlg}(<\text{intify}(a), \text{intify}(b)>))$
 $\langle \text{proof} \rangle$

lemma *zmod-neg-pos*:

$\llbracket a \text{ \$} < \#0; \#0 \text{ \$} < b \rrbracket$
 $\implies a \text{ zmod } b = \text{snd } (\text{negDivAlg}(<\text{intify}(a), \text{intify}(b)>))$
 $\langle \text{proof} \rangle$

lemma *zdiv-pos-neg*:
 $\llbracket \#0 \text{ \$} < a; b \text{ \$} < \#0 \rrbracket$
 $\implies a \text{ zdiv } b = \text{fst } (\text{negateSnd}(\text{negDivAlg } (<\$-a, \$-b>)))$
 $\langle \text{proof} \rangle$

lemma *zmod-pos-neg*:
 $\llbracket \#0 \text{ \$} < a; b \text{ \$} < \#0 \rrbracket$
 $\implies a \text{ zmod } b = \text{snd } (\text{negateSnd}(\text{negDivAlg } (<\$-a, \$-b>)))$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-neg*:
 $\llbracket a \text{ \$} < \#0; b \text{ \$} \leq \#0 \rrbracket$
 $\implies a \text{ zdiv } b = \text{fst } (\text{negateSnd}(\text{posDivAlg } (<\$-a, \$-b>)))$
 $\langle \text{proof} \rangle$

lemma *zmod-neg-neg*:
 $\llbracket a \text{ \$} < \#0; b \text{ \$} \leq \#0 \rrbracket$
 $\implies a \text{ zmod } b = \text{snd } (\text{negateSnd}(\text{posDivAlg } (<\$-a, \$-b>)))$
 $\langle \text{proof} \rangle$

declare *zdiv-pos-pos* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zdiv-neg-pos* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zdiv-pos-neg* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zdiv-neg-neg* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zmod-pos-pos* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zmod-neg-pos* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zmod-pos-neg* [of integ-of (v) integ-of (w), simp] **for** v w
declare *zmod-neg-neg* [of integ-of (v) integ-of (w), simp] **for** v w
declare *posDivAlg-eqn* [of concl: integ-of (v) integ-of (w), simp] **for** v w
declare *negDivAlg-eqn* [of concl: integ-of (v) integ-of (w), simp] **for** v w

lemma *zmod-1* [simp]: $a \text{ zmod } \#1 = \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-1* [simp]: $a \text{ zdiv } \#1 = \text{intify}(a)$
 $\langle \text{proof} \rangle$

lemma *zmod-minus1-right* [simp]: $a \text{ zmod } \#-1 = \#0$

$\langle proof \rangle$

lemma *zdiv-minus1-right-raw*: $a \in int \implies a \text{ zdiv } \#-1 = \$-a$
 $\langle proof \rangle$

lemma *zdiv-minus1-right*: $a \text{ zdiv } \#-1 = \$-a$
 $\langle proof \rangle$

declare *zdiv-minus1-right* [*simp*]

33.8 Monotonicity in the first argument (divisor)

lemma *zdiv-mono1*: $\llbracket a \$\leq a'; \#0 \$< b \rrbracket \implies a \text{ zdiv } b \$\leq a' \text{ zdiv } b$
 $\langle proof \rangle$

lemma *zdiv-mono1-neg*: $\llbracket a \$\leq a'; b \$< \#0 \rrbracket \implies a' \text{ zdiv } b \$\leq a \text{ zdiv } b$
 $\langle proof \rangle$

33.9 Monotonicity in the second argument (dividend)

lemma *q-pos-lemma*:

$\llbracket \#0 \$\leq b' \$* q' \$+ r'; r' \$< b'; \#0 \$< b' \rrbracket \implies \#0 \$\leq q'$
 $\langle proof \rangle$

lemma *zdiv-mono2-lemma*:

$\llbracket b \$* q \$+ r = b' \$* q' \$+ r'; \#0 \$\leq b' \$* q' \$+ r';$
 $r' \$< b'; \#0 \$\leq r; \#0 \$< b'; b' \$\leq b \rrbracket$
 $\implies q \$\leq q'$
 $\langle proof \rangle$

lemma *zdiv-mono2-raw*:

$\llbracket \#0 \$\leq a; \#0 \$< b'; b' \$\leq b; a \in int \rrbracket$
 $\implies a \text{ zdiv } b \$\leq a \text{ zdiv } b'$
 $\langle proof \rangle$

lemma *zdiv-mono2*:

$\llbracket \#0 \$\leq a; \#0 \$< b'; b' \$\leq b \rrbracket$
 $\implies a \text{ zdiv } b \$\leq a \text{ zdiv } b'$
 $\langle proof \rangle$

lemma *q-neg-lemma*:

$\llbracket b' \$* q' \$+ r' \$< \#0; \#0 \$\leq r'; \#0 \$< b' \rrbracket \implies q' \$< \#0$
 $\langle proof \rangle$

lemma *zdiv-mono2-neg-lemma*:

$\llbracket b \$* q \$+ r = b' \$* q' \$+ r'; b' \$* q' \$+ r' \$< \#0;$
 $r \$< b; \#0 \$\leq r'; \#0 \$< b'; b' \$\leq b \rrbracket$
 $\implies q' \$\leq q$

$\langle proof \rangle$

lemma *zdiv-mono2-neg-raw*:

$$\llbracket a \ \$ \neq 0; \ #0 \ \$ \leq b'; \ b' \ \$ \leq b; \ a \in int \rrbracket \\ \implies a \ zdiv \ b' \ \$ \leq a \ zdiv \ b$$

$\langle proof \rangle$

lemma *zdiv-mono2-neg*: $\llbracket a \ \$ \neq 0; \ #0 \ \$ \leq b'; \ b' \ \$ \leq b \rrbracket$

$$\implies a \ zdiv \ b' \ \$ \leq a \ zdiv \ b$$

$\langle proof \rangle$

33.10 More algebraic laws for zdiv and zmod

lemma *zmult1-lemma*:

$$\llbracket quorem(\langle b, c \rangle, \langle q, r \rangle); \ c \in int; \ c \neq \#0 \rrbracket \\ \implies quorem(\langle a\$*b, c \rangle, \langle a\$*q \ \$ + (a\$*r) \ zdiv \ c, (a\$*r) \ zmod \ c \rangle)$$

$\langle proof \rangle$

lemma *zdiv-zmult1-eq-raw*:

$$\llbracket b \in int; \ c \in int \rrbracket \\ \implies (a\$*b) \ zdiv \ c = a\$*(b \ zdiv \ c) \ \$ + a\$*(b \ zmod \ c) \ zdiv \ c$$

$\langle proof \rangle$

lemma *zdiv-zmult1-eq*: $(a\$*b) \ zdiv \ c = a\$*(b \ zdiv \ c) \ \$ + a\$*(b \ zmod \ c) \ zdiv \ c$

$\langle proof \rangle$

lemma *zmod-zmult1-eq-raw*:

$$\llbracket b \in int; \ c \in int \rrbracket \implies (a\$*b) \ zmod \ c = a\$*(b \ zmod \ c) \ zmod \ c$$

$\langle proof \rangle$

lemma *zmod-zmult1-eq*: $(a\$*b) \ zmod \ c = a\$*(b \ zmod \ c) \ zmod \ c$

$\langle proof \rangle$

lemma *zmod-zmult1-eq'*: $(a\$*b) \ zmod \ c = ((a \ zmod \ c) \ \$* \ b) \ zmod \ c$

$\langle proof \rangle$

lemma *zmod-zmult-distrib*: $(a\$*b) \ zmod \ c = ((a \ zmod \ c) \ \$* \ (b \ zmod \ c)) \ zmod \ c$

$\langle proof \rangle$

lemma *zdiv-zmult-self1* [simp]: $intify(b) \neq \#0 \implies (a\$*b) \ zdiv \ b = intify(a)$

$\langle proof \rangle$

lemma *zdiv-zmult-self2* [simp]: $intify(b) \neq \#0 \implies (b\$*a) \ zdiv \ b = intify(a)$

$\langle proof \rangle$

lemma *zmod-zmult-self1* [simp]: $(a\$*b) \ zmod \ b = \#0$

$\langle proof \rangle$

lemma *zmod-zmult-self2* [simp]: $(b\$*a) \ zmod \ b = \#0$

$\langle proof \rangle$

lemma *zadd1-lemma*:

$\llbracket quorem(\langle a, c \rangle, \langle aq, ar \rangle); quorem(\langle b, c \rangle, \langle bq, br \rangle);$
 $c \in int; c \neq \#0 \rrbracket$
 $\implies quorem(\langle a\$+b, c \rangle, \langle aq \$+ bq \$+ (ar\$+br) zdiv c, (ar\$+br) zmod c \rangle)$
 $\langle proof \rangle$

lemma *zdiv-zadd1-eq-raw*:

$\llbracket a \in int; b \in int; c \in int \rrbracket \implies$
 $(a\$+b) zdiv c = a zdiv c \$+ b zdiv c \$+ ((a zmod c \$+ b zmod c) zdiv c)$
 $\langle proof \rangle$

lemma *zdiv-zadd1-eq*:

$(a\$+b) zdiv c = a zdiv c \$+ b zdiv c \$+ ((a zmod c \$+ b zmod c) zdiv c)$
 $\langle proof \rangle$

lemma *zmod-zadd1-eq-raw*:

$\llbracket a \in int; b \in int; c \in int \rrbracket$
 $\implies (a\$+b) zmod c = (a zmod c \$+ b zmod c) zmod c$
 $\langle proof \rangle$

lemma *zmod-zadd1-eq*: $(a\$+b) zmod c = (a zmod c \$+ b zmod c) zmod c$

$\langle proof \rangle$

lemma *zmod-div-trivial-raw*:

$\llbracket a \in int; b \in int \rrbracket \implies (a zmod b) zdiv b = \#0$
 $\langle proof \rangle$

lemma *zmod-div-trivial [simp]*: $(a zmod b) zdiv b = \#0$

$\langle proof \rangle$

lemma *zmod-mod-trivial-raw*:

$\llbracket a \in int; b \in int \rrbracket \implies (a zmod b) zmod b = a zmod b$
 $\langle proof \rangle$

lemma *zmod-mod-trivial [simp]*: $(a zmod b) zmod b = a zmod b$

$\langle proof \rangle$

lemma *zmod-zadd-left-eq*: $(a\$+b) zmod c = ((a zmod c) \$+ b) zmod c$

$\langle proof \rangle$

lemma *zmod-zadd-right-eq*: $(a\$+b) zmod c = (a \$+ (b zmod c)) zmod c$

$\langle proof \rangle$

lemma *zdiv-zadd-self1* [simp]:

$\text{intify}(a) \neq \#0 \implies (a\$+b) \text{ zdiv } a = b \text{ zdiv } a \$+ \#1$
 $\langle \text{proof} \rangle$

lemma *zdiv-zadd-self2* [simp]:

$\text{intify}(a) \neq \#0 \implies (b\$+a) \text{ zdiv } a = b \text{ zdiv } a \$+ \#1$
 $\langle \text{proof} \rangle$

lemma *zmod-zadd-self1* [simp]: $(a\$+b) \text{ zmod } a = b \text{ zmod } a$

$\langle \text{proof} \rangle$

lemma *zmod-zadd-self2* [simp]: $(b\$+a) \text{ zmod } a = b \text{ zmod } a$

$\langle \text{proof} \rangle$

33.11 proving a $\text{zdiv} (b*c) = (a \text{ zdiv } b) \text{ zdiv } c$

lemma *zdiv-zmult2-aux1*:

$\llbracket \#0 \$< c; b \$< r; r \$\leq \#0 \rrbracket \implies b\$*c \$< b\$*(q \text{ zmod } c) \$+ r$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux2*:

$\llbracket \#0 \$< c; b \$< r; r \$\leq \#0 \rrbracket \implies b \$* (q \text{ zmod } c) \$+ r \$\leq \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux3*:

$\llbracket \#0 \$< c; \#0 \$\leq r; r \$< b \rrbracket \implies \#0 \$\leq b \$* (q \text{ zmod } c) \$+ r$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-aux4*:

$\llbracket \#0 \$< c; \#0 \$\leq r; r \$< b \rrbracket \implies b \$* (q \text{ zmod } c) \$+ r \$< b \$* c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-lemma*:

$\llbracket \text{quorem} (\langle a, b \rangle, \langle q, r \rangle); a \in \text{int}; b \in \text{int}; b \neq \#0; \#0 \$< c \rrbracket$
 $\implies \text{quorem} (\langle a, b\$*c \rangle, \langle q \text{ zdiv } c, b\$*(q \text{ zmod } c) \$+ r \rangle)$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-eq-raw*:

$\llbracket \#0 \$< c; a \in \text{int}; b \in \text{int} \rrbracket \implies a \text{ zdiv } (b\$*c) = (a \text{ zdiv } b) \text{ zdiv } c$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult2-eq*: $\#0 \$< c \implies a \text{ zdiv } (b\$*c) = (a \text{ zdiv } b) \text{ zdiv } c$

$\langle \text{proof} \rangle$

lemma *zmod-zmult2-eq-raw*:

$\llbracket \#0 \$< c; a \in \text{int}; b \in \text{int} \rrbracket$
 $\implies a \text{ zmod } (b\$*c) = b\$*(a \text{ zdiv } b \text{ zmod } c) \$+ a \text{ zmod } b$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult2-eq*:

$\#0 \ \$< c \implies a \text{ zmod } (b\$*c) = b\$*(a \text{ zdiv } b \text{ zmod } c) \$+ a \text{ zmod } b$
 $\langle \text{proof} \rangle$

33.12 Cancellation of common factors in "zdiv"

lemma *zdiv-zmult-zmult1-aux1*:

$\llbracket \#0 \ \$< b; \text{intify}(c) \neq \#0 \rrbracket \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-aux2*:

$\llbracket b \ \$< \#0; \text{intify}(c) \neq \#0 \rrbracket \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1-raw*:

$\llbracket \text{intify}(c) \neq \#0; b \in \text{int} \rrbracket \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult1*: $\text{intify}(c) \neq \#0 \implies (c\$*a) \text{ zdiv } (c\$*b) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

lemma *zdiv-zmult-zmult2*: $\text{intify}(c) \neq \#0 \implies (a\$*c) \text{ zdiv } (b\$*c) = a \text{ zdiv } b$
 $\langle \text{proof} \rangle$

33.13 Distribution of factors over "zmod"

lemma *zmod-zmult-zmult1-aux1*:

$\llbracket \#0 \ \$< b; \text{intify}(c) \neq \#0 \rrbracket$
 $\implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1-aux2*:

$\llbracket b \ \$< \#0; \text{intify}(c) \neq \#0 \rrbracket$
 $\implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1-raw*:

$\llbracket b \in \text{int}; c \in \text{int} \rrbracket \implies (c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult1*: $(c\$*a) \text{ zmod } (c\$*b) = c \$* (a \text{ zmod } b)$
 $\langle \text{proof} \rangle$

lemma *zmod-zmult-zmult2*: $(a\$*c) \text{ zmod } (b\$*c) = (a \text{ zmod } b) \$* c$
 $\langle \text{proof} \rangle$

lemma *zdiv-neg-pos-less0*: $\llbracket a \text{ \$} < \#0; \#0 \text{ \$} < b \rrbracket \implies a \text{ zdiv } b \text{ \$} < \#0$
 $\langle \text{proof} \rangle$

lemma *zdiv-nonneg-neg-le0*: $\llbracket \#0 \text{ \$} \leq a; b \text{ \$} < \#0 \rrbracket \implies a \text{ zdiv } b \text{ \$} \leq \#0$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-nonneg-iff*: $\#0 \text{ \$} < b \implies (\#0 \text{ \$} \leq a \text{ zdiv } b) \longleftrightarrow (\#0 \text{ \$} \leq a)$
 $\langle \text{proof} \rangle$

lemma *neg-imp-zdiv-nonneg-iff*: $b \text{ \$} < \#0 \implies (\#0 \text{ \$} \leq a \text{ zdiv } b) \longleftrightarrow (a \text{ \$} \leq \#0)$
 $\langle \text{proof} \rangle$

lemma *pos-imp-zdiv-neg-iff*: $\#0 \text{ \$} < b \implies (a \text{ zdiv } b \text{ \$} < \#0) \longleftrightarrow (a \text{ \$} < \#0)$
 $\langle \text{proof} \rangle$

lemma *neg-imp-zdiv-neg-iff*: $b \text{ \$} < \#0 \implies (a \text{ zdiv } b \text{ \$} < \#0) \longleftrightarrow (\#0 \text{ \$} < a)$
 $\langle \text{proof} \rangle$

end

34 Cardinal Arithmetic Without the Axiom of Choice

theory *CardinalArith* **imports** *Cardinal OrderArith ArithSimp Finite* **begin**

definition

InfCard $:: i \Rightarrow o$ **where**
InfCard(*i*) $\equiv \text{Card}(i) \wedge \text{nat} \leq i$

definition

cmult $:: [i, i] \Rightarrow i$ (**infixl** $\langle \otimes \rangle$ 70) **where**
 $i \otimes j \equiv |i * j|$

definition

cadd $:: [i, i] \Rightarrow i$ (**infixl** $\langle \oplus \rangle$ 65) **where**
 $i \oplus j \equiv |i + j|$

definition

csquare-rel $:: i \Rightarrow i$ **where**
csquare-rel(*K*) \equiv
 $\text{rvmage}(K * K,$
 $\text{lam } \langle x, y \rangle : K * K. < x \cup y, x, y >,$
 $\text{rmult}(K, \text{Memrel}(K), K * K, \text{rmult}(K, \text{Memrel}(K), K, \text{Memrel}(K))))$

definition

jump-cardinal $:: i \Rightarrow i$ **where**

— This definition is more complex than Kunen's but it more easily proved to be a cardinal

$$\text{jump-cardinal}(K) \equiv \bigcup X \in \text{Pow}(K). \{z. r \in \text{Pow}(K * K), \text{well-ord}(X, r) \wedge z = \text{ordertype}(X, r)\}$$

definition

$\text{csucc} :: i \Rightarrow i$ **where**

— needed because $\text{jump-cardinal}(K)$ might not be the successor of K

$$\text{csucc}(K) \equiv \mu L. \text{Card}(L) \wedge K < L$$

lemma *Card-Union* [*simp,intro,TC*]:

assumes $A: \bigwedge x. x \in A \Rightarrow \text{Card}(x)$ **shows** $\text{Card}(\bigcup(A))$
 $\langle \text{proof} \rangle$

lemma *Card-UN*: $(\bigwedge x. x \in A \Rightarrow \text{Card}(K(x))) \Rightarrow \text{Card}(\bigcup_{x \in A} K(x))$

$\langle \text{proof} \rangle$

lemma *Card-OUN* [*simp,intro,TC*]:

$(\bigwedge x. x \in A \Rightarrow \text{Card}(K(x))) \Rightarrow \text{Card}(\bigcup_{x < A} K(x))$
 $\langle \text{proof} \rangle$

lemma *in-Card-imp-lesspoll*: $\llbracket \text{Card}(K); b \in K \rrbracket \Rightarrow b \prec K$

$\langle \text{proof} \rangle$

34.1 Cardinal addition

Note: Could omit proving the algebraic laws for cardinal addition and multiplication. On finite cardinals these operations coincide with addition and multiplication of natural numbers; on infinite cardinals they coincide with union (maximum). Either way we get most laws for free.

34.1.1 Cardinal addition is commutative

lemma *sum-commute-epoll*: $A + B \approx B + A$

$\langle \text{proof} \rangle$

lemma *cadd-commute*: $i \oplus j = j \oplus i$

$\langle \text{proof} \rangle$

34.1.2 Cardinal addition is associative

lemma *sum-assoc-epoll*: $(A + B) + C \approx A + (B + C)$

$\langle \text{proof} \rangle$

Unconditional version requires AC

lemma *well-ord-cadd-assoc*:

assumes $i: \text{well-ord}(i, ri)$ **and** $j: \text{well-ord}(j, rj)$ **and** $k: \text{well-ord}(k, rk)$

shows $(i \oplus j) \oplus k = i \oplus (j \oplus k)$
 $\langle proof \rangle$

34.1.3 0 is the identity for addition

lemma *sum-0-eqpoll*: $0 + A \approx A$
 $\langle proof \rangle$

lemma *cadd-0 [simp]*: $Card(K) \implies 0 \oplus K = K$
 $\langle proof \rangle$

34.1.4 Addition by another cardinal

lemma *sum-lepoll-self*: $A \lesssim A + B$
 $\langle proof \rangle$

lemma *cadd-le-self*:
assumes K : $Card(K)$ **and** L : $Ord(L)$ **shows** $K \leq (K \oplus L)$
 $\langle proof \rangle$

34.1.5 Monotonicity of addition

lemma *sum-lepoll-mono*:
 $\llbracket A \lesssim C; B \lesssim D \rrbracket \implies A + B \lesssim C + D$
 $\langle proof \rangle$

lemma *cadd-le-mono*:
 $\llbracket K' \leq K; L' \leq L \rrbracket \implies (K' \oplus L') \leq (K \oplus L)$
 $\langle proof \rangle$

34.1.6 Addition of finite cardinals is "ordinary" addition

lemma *sum-succ-eqpoll*: $succ(A) + B \approx succ(A + B)$
 $\langle proof \rangle$

lemma *cadd-succ-lemma*:
assumes $Ord(m)$ $Ord(n)$ **shows** $succ(m) \oplus n = |succ(m \oplus n)|$
 $\langle proof \rangle$

lemma *nat-cadd-eq-add*:
assumes m : $m \in nat$ **and** $[simp]$: $n \in nat$ **shows** $m \oplus n = m \# + n$
 $\langle proof \rangle$

34.2 Cardinal multiplication

34.2.1 Cardinal multiplication is commutative

lemma *prod-commute-epoll*: $A*B \approx B*A$
<proof>

lemma *cmult-commute*: $i \otimes j = j \otimes i$
<proof>

34.2.2 Cardinal multiplication is associative

lemma *prod-assoc-epoll*: $(A*B)*C \approx A*(B*C)$
<proof>

Unconditional version requires AC

lemma *well-ord-cmult-assoc*:
 assumes i : *well-ord*(i,ri) **and** j : *well-ord*(j,rj) **and** k : *well-ord*(k,rk)
 shows $(i \otimes j) \otimes k = i \otimes (j \otimes k)$
<proof>

34.2.3 Cardinal multiplication distributes over addition

lemma *sum-prod-distrib-epoll*: $(A+B)*C \approx (A*C)+(B*C)$
<proof>

lemma *well-ord-cadd-cmult-distrib*:
 assumes i : *well-ord*(i,ri) **and** j : *well-ord*(j,rj) **and** k : *well-ord*(k,rk)
 shows $(i \oplus j) \otimes k = (i \otimes k) \oplus (j \otimes k)$
<proof>

34.2.4 Multiplication by 0 yields 0

lemma *prod-0-epoll*: $0*A \approx 0$
<proof>

lemma *cmult-0 [simp]*: $0 \otimes i = 0$
<proof>

34.2.5 1 is the identity for multiplication

lemma *prod-singleton-epoll*: $\{x\}*A \approx A$
<proof>

lemma *cmult-1 [simp]*: $\text{Card}(K) \implies 1 \otimes K = K$
<proof>

34.3 Some inequalities for multiplication

lemma *prod-square-lepoll*: $A \lesssim A*A$
<proof>

lemma *cmult-square-le*: $\text{Card}(K) \implies K \leq K \otimes K$
 $\langle \text{proof} \rangle$

34.3.1 Multiplication by a non-zero cardinal

lemma *prod-lepoll-self*: $b \in B \implies A \lesssim A * B$
 $\langle \text{proof} \rangle$

lemma *cmult-le-self*:
 $\llbracket \text{Card}(K); \text{Ord}(L); 0 < L \rrbracket \implies K \leq (K \otimes L)$
 $\langle \text{proof} \rangle$

34.3.2 Monotonicity of multiplication

lemma *prod-lepoll-mono*:
 $\llbracket A \lesssim C; B \lesssim D \rrbracket \implies A * B \lesssim C * D$
 $\langle \text{proof} \rangle$

lemma *cmult-le-mono*:
 $\llbracket K' \leq K; L' \leq L \rrbracket \implies (K' \otimes L') \leq (K \otimes L)$
 $\langle \text{proof} \rangle$

34.4 Multiplication of finite cardinals is "ordinary" multiplication

lemma *prod-succ-epoll*: $\text{succ}(A) * B \approx B + A * B$
 $\langle \text{proof} \rangle$

lemma *cmult-succ-lemma*:
 $\llbracket \text{Ord}(m); \text{Ord}(n) \rrbracket \implies \text{succ}(m) \otimes n = n \oplus (m \otimes n)$
 $\langle \text{proof} \rangle$

lemma *nat-cmult-eq-mult*: $\llbracket m \in \text{nat}; n \in \text{nat} \rrbracket \implies m \otimes n = m \# * n$
 $\langle \text{proof} \rangle$

lemma *cmult-2*: $\text{Card}(n) \implies 2 \otimes n = n \oplus n$
 $\langle \text{proof} \rangle$

lemma *sum-lepoll-prod*:
assumes $C: 2 \lesssim C$ **shows** $B + B \lesssim C * B$
 $\langle \text{proof} \rangle$

lemma *lepoll-imp-sum-lepoll-prod*: $\llbracket A \lesssim B; 2 \lesssim A \rrbracket \implies A + B \lesssim A * B$
 $\langle \text{proof} \rangle$

34.5 Infinite Cardinals are Limit Ordinals

lemma *nat-cons-lepoll*: $\text{nat} \lesssim A \implies \text{cons}(u, A) \lesssim A$
 $\langle \text{proof} \rangle$

lemma *nat-cons-epoll*: $\text{nat} \lesssim A \implies \text{cons}(u, A) \approx A$
 $\langle \text{proof} \rangle$

lemma *nat-succ-epoll*: $\text{nat} \subseteq A \implies \text{succ}(A) \approx A$
 $\langle \text{proof} \rangle$

lemma *InfCard-nat*: $\text{InfCard}(\text{nat})$
 $\langle \text{proof} \rangle$

lemma *InfCard-is-Card*: $\text{InfCard}(K) \implies \text{Card}(K)$
 $\langle \text{proof} \rangle$

lemma *InfCard-Un*:
 $\llbracket \text{InfCard}(K); \text{Card}(L) \rrbracket \implies \text{InfCard}(K \cup L)$
 $\langle \text{proof} \rangle$

lemma *InfCard-is-Limit*: $\text{InfCard}(K) \implies \text{Limit}(K)$
 $\langle \text{proof} \rangle$

lemma *ordermap-epoll-pred*:
 $\llbracket \text{well-ord}(A, r); x \in A \rrbracket \implies \text{ordermap}(A, r) \cdot x \approx \text{Order.pred}(A, x, r)$
 $\langle \text{proof} \rangle$

34.5.1 Establishing the well-ordering

lemma *well-ord-csquare*:
assumes K : $\text{Ord}(K)$ **shows** $\text{well-ord}(K * K, \text{csquare-rel}(K))$
 $\langle \text{proof} \rangle$

34.5.2 Characterising initial segments of the well-ordering

lemma *csquareD*:
 $\llbracket \langle x, y \rangle, \langle z, z \rangle \in \text{csquare-rel}(K); x < K; y < K; z < K \rrbracket \implies x \leq z \wedge y \leq z$
 $\langle \text{proof} \rangle$

lemma *pred-csquare-subset*:
 $z < K \implies \text{Order.pred}(K * K, \langle z, z \rangle, \text{csquare-rel}(K)) \subseteq \text{succ}(z) * \text{succ}(z)$
 $\langle \text{proof} \rangle$

lemma *csquare-ltI*:

$\llbracket x < z; y < z; z < K \rrbracket \implies \langle \langle x, y \rangle, \langle z, z \rangle \rangle \in \text{csquare-rel}(K)$
 $\langle \text{proof} \rangle$

lemma *csquare-or-eqI*:

$\llbracket x \leq z; y \leq z; z < K \rrbracket \implies \langle \langle x, y \rangle, \langle z, z \rangle \rangle \in \text{csquare-rel}(K) \mid x = z \wedge y = z$
 $\langle \text{proof} \rangle$

34.5.3 The cardinality of initial segments

lemma *ordermap-z-lt*:

$\llbracket \text{Limit}(K); x < K; y < K; z = \text{succ}(x \cup y) \rrbracket \implies$
 $\text{ordermap}(K * K, \text{csquare-rel}(K)) \restriction \langle x, y \rangle <$
 $\text{ordermap}(K * K, \text{csquare-rel}(K)) \restriction \langle z, z \rangle$
 $\langle \text{proof} \rangle$

Kunen: "each $\langle x, y \rangle \in K \times K$ has no more than $z \times z$ predecessors..." (page 29)

lemma *ordermap-csquare-le*:

assumes $K: \text{Limit}(K)$ **and** $x: x < K$ **and** $y: y < K$
defines $z \equiv \text{succ}(x \cup y)$
shows $|\text{ordermap}(K \times K, \text{csquare-rel}(K)) \restriction \langle x, y \rangle| \leq |\text{succ}(z)| \otimes |\text{succ}(z)|$
 $\langle \text{proof} \rangle$

Kunen: "... so the order type is $\leq K$ "

lemma *ordertype-csquare-le*:

assumes $IK: \text{InfCard}(K)$ **and** $\text{eq}: \bigwedge y. y \in K \implies \text{InfCard}(y) \implies y \otimes y = y$
shows $\text{ordertype}(K * K, \text{csquare-rel}(K)) \leq K$
 $\langle \text{proof} \rangle$

lemma *InfCard-csquare-eq*:

assumes $IK: \text{InfCard}(K)$ **shows** $K \otimes K = K$
 $\langle \text{proof} \rangle$

lemma *well-ord-InfCard-square-eq*:

assumes $r: \text{well-ord}(A, r)$ **and** $I: \text{InfCard}(|A|)$ **shows** $A \times A \approx A$
 $\langle \text{proof} \rangle$

lemma *InfCard-square-eqpoll*: $\text{InfCard}(K) \implies K \times K \approx K$

$\langle \text{proof} \rangle$

lemma *Inf-Card-is-InfCard*: $\llbracket \text{Card}(i); \neg \text{Finite}(i) \rrbracket \implies \text{InfCard}(i)$

$\langle \text{proof} \rangle$

34.5.4 Toward's Kunen's Corollary 10.13 (1)

lemma *InfCard-le-cmult-eq*: $\llbracket \text{InfCard}(K); L \leq K; 0 < L \rrbracket \implies K \otimes L = K$

$\langle proof \rangle$

lemma *InfCard-cmult-eq*: $\llbracket InfCard(K); InfCard(L) \rrbracket \implies K \otimes L = K \cup L$
 $\langle proof \rangle$

lemma *InfCard-cdouble-eq*: $InfCard(K) \implies K \oplus K = K$
 $\langle proof \rangle$

lemma *InfCard-le-cadd-eq*: $\llbracket InfCard(K); L \leq K \rrbracket \implies K \oplus L = K$
 $\langle proof \rangle$

lemma *InfCard-cadd-eq*: $\llbracket InfCard(K); InfCard(L) \rrbracket \implies K \oplus L = K \cup L$
 $\langle proof \rangle$

34.6 For Every Cardinal Number There Exists A Greater One

This result is Kunen's Theorem 10.16, which would be trivial using AC

lemma *Ord-jump-cardinal*: $Ord(jump-cardinal(K))$
 $\langle proof \rangle$

lemma *jump-cardinal-iff*:
 $i \in jump-cardinal(K) \longleftrightarrow$
 $(\exists r \ X. \ r \subseteq K * K \wedge X \subseteq K \wedge well-ord(X, r) \wedge i = ordertype(X, r))$
 $\langle proof \rangle$

lemma *K-lt-jump-cardinal*: $Ord(K) \implies K < jump-cardinal(K)$
 $\langle proof \rangle$

lemma *Card-jump-cardinal-lemma*:
 $\llbracket well-ord(X, r); r \subseteq K * K; X \subseteq K;$
 $f \in bij(ordertype(X, r), jump-cardinal(K)) \rrbracket$
 $\implies jump-cardinal(K) \in jump-cardinal(K)$
 $\langle proof \rangle$

lemma *Card-jump-cardinal*: $Card(jump-cardinal(K))$
 $\langle proof \rangle$

34.7 Basic Properties of Successor Cardinals

lemma *csucc-basic*: $Ord(K) \implies Card(csucc(K)) \wedge K < csucc(K)$
 $\langle proof \rangle$

lemmas *Card-csucc* = *csucc-basic* [*THEN conjunct1*]

lemmas *lt-csucc* = *csucc-basic* [*THEN conjunct2*]

lemma *Ord-0-lt-csucc*: $\text{Ord}(K) \implies 0 < \text{csucc}(K)$
 $\langle \text{proof} \rangle$

lemma *csucc-le*: $\llbracket \text{Card}(L); K < L \rrbracket \implies \text{csucc}(K) \leq L$
 $\langle \text{proof} \rangle$

lemma *lt-csucc-iff*: $\llbracket \text{Ord}(i); \text{Card}(K) \rrbracket \implies i < \text{csucc}(K) \longleftrightarrow |i| \leq K$
 $\langle \text{proof} \rangle$

lemma *Card-lt-csucc-iff*:
 $\llbracket \text{Card}(K'); \text{Card}(K) \rrbracket \implies K' < \text{csucc}(K) \longleftrightarrow K' \leq K$
 $\langle \text{proof} \rangle$

lemma *InfCard-csucc*: $\text{InfCard}(K) \implies \text{InfCard}(\text{csucc}(K))$
 $\langle \text{proof} \rangle$

34.7.1 Removing elements from a finite set decreases its cardinality

lemma *Finite-imp-cardinal-cons* [*simp*]:
assumes *FA*: *Finite*(*A*) **and** *a*: $a \notin A$ **shows** $|\text{cons}(a, A)| = \text{succ}(|A|)$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-succ-cardinal-Diff*:
 $\llbracket \text{Finite}(A); a \in A \rrbracket \implies \text{succ}(|A - \{a\}|) = |A|$
 $\langle \text{proof} \rangle$

lemma *Finite-imp-cardinal-Diff*: $\llbracket \text{Finite}(A); a \in A \rrbracket \implies |A - \{a\}| < |A|$
 $\langle \text{proof} \rangle$

lemma *Finite-cardinal-in-nat* [*simp*]: $\text{Finite}(A) \implies |A| \in \text{nat}$
 $\langle \text{proof} \rangle$

lemma *card-Un-Int*:
 $\llbracket \text{Finite}(A); \text{Finite}(B) \rrbracket \implies |A| \# + |B| = |A \cup B| \# + |A \cap B|$
 $\langle \text{proof} \rangle$

lemma *card-Un-disjoint*:
 $\llbracket \text{Finite}(A); \text{Finite}(B); A \cap B = 0 \rrbracket \implies |A \cup B| = |A| \# + |B|$
 $\langle \text{proof} \rangle$

lemma *card-partition*:
assumes *FC*: *Finite*(*C*)
shows
 $\text{Finite} (\bigcup C) \implies$

$(\forall c \in C. |c| = k) \implies$
 $(\forall c1 \in C. \forall c2 \in C. c1 \neq c2 \longrightarrow c1 \cap c2 = 0) \implies$
 $k \#* |C| = |\bigcup C|$
 $\langle proof \rangle$

34.7.2 Theorems by Krzysztof Grabczewski, proofs by lcp

lemmas *nat-implies-well-ord* = *nat-into-Ord* [*THEN well-ord-Memrel*]

lemma *nat-sum-ecpoll-sum*:

assumes *m*: $m \in \text{nat}$ **and** *n*: $n \in \text{nat}$ **shows** $m + n \approx m \# + n$
 $\langle proof \rangle$

lemma *Ord-subset-natD* [*rule-format*]: $\text{Ord}(i) \implies i \subseteq \text{nat} \implies i \in \text{nat} \mid i = \text{nat}$
 $\langle proof \rangle$

lemma *Ord-nat-subset-into-Card*: $\llbracket \text{Ord}(i); i \subseteq \text{nat} \rrbracket \implies \text{Card}(i)$
 $\langle proof \rangle$

end

35 Main ZF Theory: Everything Except AC

theory *ZF* **imports** *List IntDiv CardinalArith* **begin**

35.1 Iteration of the function *F*

consts *iterates* :: $[i \Rightarrow i, i, i] \Rightarrow i$ ($\langle \langle \text{notation} = \langle \text{mixfix iterates} \rangle \rangle \cdot \hat{\sim} '(-) \rangle$) [*60,1000,1000*]
60)

primrec

$F^{\hat{\sim} 0} (x) = x$
 $F^{\hat{\sim} (\text{succ}(n))} (x) = F(F^{\hat{\sim} n} (x))$

definition

iterates-omega :: $[i \Rightarrow i, i] \Rightarrow i$ ($\langle \langle \text{notation} = \langle \text{mixfix iterates-omega} \rangle \rangle \cdot \hat{\sim} \omega '(-) \rangle$)
 $[60,1000]$ *60*) **where**
 $F^{\hat{\sim} \omega} (x) \equiv \bigcup n \in \text{nat}. F^{\hat{\sim} n} (x)$

lemma *iterates-triv*:

$\llbracket n \in \text{nat}; F(x) = x \rrbracket \implies F^{\hat{\sim} n} (x) = x$
 $\langle proof \rangle$

lemma *iterates-type* [*TC*]:

$\llbracket n \in \text{nat}; a \in A; \bigwedge x. x \in A \implies F(x) \in A \rrbracket$
 $\implies F^{\hat{\sim} n} (a) \in A$
 $\langle proof \rangle$

lemma *iterates-omega-triv*:

$F(x) = x \implies F^\omega(x) = x$
 $\langle \text{proof} \rangle$

lemma *Ord-iterates* [simp]:
 $\llbracket n \in \text{nat}; \bigwedge i. \text{Ord}(i) \implies \text{Ord}(F(i)); \text{Ord}(x) \rrbracket$
 $\implies \text{Ord}(F^\omega(x))$
 $\langle \text{proof} \rangle$

lemma *iterates-commute*: $n \in \text{nat} \implies F(F^\omega(x)) = F^\omega(F(x))$
 $\langle \text{proof} \rangle$

35.2 Transfinite Recursion

Transfinite recursion for definitions based on the three cases of ordinals

definition

$\text{transrec3} :: [i, i, [i, i] \Rightarrow i, [i, i] \Rightarrow i] \Rightarrow i$ **where**
 $\text{transrec3}(k, a, b, c) \equiv$
 $\text{transrec}(k, \lambda x r.$
 $\quad \text{if } x=0 \text{ then } a$
 $\quad \text{else if } \text{Limit}(x) \text{ then } c(x, \lambda y \in x. r'y)$
 $\quad \text{else } b(\text{Arith.pred}(x), r \text{ ` Arith.pred}(x)))$

lemma *transrec3-0* [simp]: $\text{transrec3}(0, a, b, c) = a$
 $\langle \text{proof} \rangle$

lemma *transrec3-succ* [simp]:
 $\text{transrec3}(\text{succ}(i), a, b, c) = b(i, \text{transrec3}(i, a, b, c))$
 $\langle \text{proof} \rangle$

lemma *transrec3-Limit*:
 $\text{Limit}(i) \implies$
 $\text{transrec3}(i, a, b, c) = c(i, \lambda j \in i. \text{transrec3}(j, a, b, c))$
 $\langle \text{proof} \rangle$

$\langle \text{ML} \rangle$

end

36 The Axiom of Choice

theory *AC* **imports** *ZF* **begin**

This definition comes from Halmos (1960), page 59.

axiomatization where

$AC: \llbracket a \in A; \bigwedge x. x \in A \implies (\exists y. y \in B(x)) \rrbracket \implies \exists z. z \in \text{Pi}(A, B)$

lemma *AC-Pi*: $\llbracket \bigwedge x. x \in A \implies (\exists y. y \in B(x)) \rrbracket \implies \exists z. z \in Pi(A,B)$
 $\langle proof \rangle$

lemma *AC-ball-Pi*: $\forall x \in A. \exists y. y \in B(x) \implies \exists y. y \in Pi(A,B)$
 $\langle proof \rangle$

lemma *AC-Pi-Pow*: $\exists f. f \in (\prod X \in Pow(C) - \{0\}. X)$
 $\langle proof \rangle$

lemma *AC-func*:
 $\llbracket \bigwedge x. x \in A \implies (\exists y. y \in x) \rrbracket \implies \exists f \in A \multimap \bigcup (A). \forall x \in A. f'x \in x$
 $\langle proof \rangle$

lemma *non-empty-family*: $\llbracket 0 \notin A; x \in A \rrbracket \implies \exists y. y \in x$
 $\langle proof \rangle$

lemma *AC-func0*: $0 \notin A \implies \exists f \in A \multimap \bigcup (A). \forall x \in A. f'x \in x$
 $\langle proof \rangle$

lemma *AC-func-Pow*: $\exists f \in (Pow(C) - \{0\}) \multimap C. \forall x \in Pow(C) - \{0\}. f'x \in x$
 $\langle proof \rangle$

lemma *AC-Pi0*: $0 \notin A \implies \exists f. f \in (\prod x \in A. x)$
 $\langle proof \rangle$

end

37 Zorn's Lemma

theory *Zorn* **imports** *OrderArith AC Inductive* **begin**

Based upon the unpublished article “Towards the Mechanization of the Proofs of Some Classical Theorems of Set Theory,” by Abrial and Laffitte.

definition

Subset-rel :: $i \Rightarrow i$ **where**
 $Subset-rel(A) \equiv \{z \in A * A . \exists x y. z = \langle x, y \rangle \wedge x <= y \wedge x \neq y\}$

definition

chain :: $i \Rightarrow i$ **where**
 $chain(A) \equiv \{F \in Pow(A). \forall X \in F. \forall Y \in F. X <= Y \mid Y <= X\}$

definition

super :: $[i, i] \Rightarrow i$ **where**
 $super(A, c) \equiv \{d \in chain(A). c <= d \wedge c \neq d\}$

definition

maxchain :: $i \Rightarrow i$ **where**
 $maxchain(A) \equiv \{c \in chain(A). super(A, c) = 0\}$

definition

increasing :: $i \Rightarrow i$ **where**

$$\text{increasing}(A) \equiv \{f \in \text{Pow}(A) \rightarrow \text{Pow}(A). \forall x. x \leq A \rightarrow x \leq f'x\}$$

Lemma for the inductive definition below

lemma *Union-in-Pow*: $Y \in \text{Pow}(\text{Pow}(A)) \Rightarrow \bigcup(Y) \in \text{Pow}(A)$
 ⟨proof⟩

We could make the inductive definition conditional on $\text{next} \in \text{increasing}(S)$ but instead we make this a side-condition of an introduction rule. Thus the induction rule lets us assume that condition! Many inductive proofs are therefore unconditional.

consts

TFin :: $[i, i] \Rightarrow i$

inductive

domains $\text{TFin}(S, \text{next}) \subseteq \text{Pow}(S)$

intros

nextI: $\llbracket x \in \text{TFin}(S, \text{next}); \text{next} \in \text{increasing}(S) \rrbracket$
 $\Rightarrow \text{next}'x \in \text{TFin}(S, \text{next})$

Pow-UnionI: $Y \in \text{Pow}(\text{TFin}(S, \text{next})) \Rightarrow \bigcup(Y) \in \text{TFin}(S, \text{next})$

monos *Pow-mono*

con-defs *increasing-def*

type-intros *CollectD1 [THEN apply-funtype] Union-in-Pow*

37.1 Mathematical Preamble

lemma *Union-lemma0*: $(\forall x \in C. x \leq A \mid B \leq x) \Rightarrow \bigcup(C) \leq A \mid B \leq \bigcup(C)$
 ⟨proof⟩

lemma *Inter-lemma0*:

$\llbracket c \in C; \forall x \in C. A \leq x \mid x \leq B \rrbracket \Rightarrow A \subseteq \bigcap(C) \mid \bigcap(C) \subseteq B$
 ⟨proof⟩

37.2 The Transfinite Construction

lemma *increasingD1*: $f \in \text{increasing}(A) \Rightarrow f \in \text{Pow}(A) \rightarrow \text{Pow}(A)$
 ⟨proof⟩

lemma *increasingD2*: $\llbracket f \in \text{increasing}(A); x \leq A \rrbracket \Rightarrow x \subseteq f'x$
 ⟨proof⟩

lemmas *TFin-UnionI = PowI [THEN TFin.Pow-UnionI]*

lemmas *TFin-is-subset = TFin.dom-subset [THEN subsetD, THEN PowD]*

Structural induction on $TFin(S, next)$

lemma *TFin-induct*:

$\llbracket n \in TFin(S, next);$
 $\bigwedge x. \llbracket x \in TFin(S, next); P(x); next \in increasing(S) \rrbracket \implies P(next'x);$
 $\bigwedge Y. \llbracket Y \subseteq TFin(S, next); \forall y \in Y. P(y) \rrbracket \implies P(\bigcup(Y))$
 $\rrbracket \implies P(n)$
 $\langle proof \rangle$

37.3 Some Properties of the Transfinite Construction

lemmas *increasing-trans* = *subset-trans* [*OF* - *increasingD2*,
OF - - *TFin-is-subset*]

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:

$\llbracket n \in TFin(S, next); m \in TFin(S, next);$
 $\forall x \in TFin(S, next). x \leq m \longrightarrow x = m \mid next'x \leq m \rrbracket$
 $\implies n \leq m \mid next'm \leq n$
 $\langle proof \rangle$

Lemma 2 of section 3.2. Interesting in its own right! Requires $next \in increasing(S)$ in the second induction step.

lemma *TFin-linear-lemma2*:

$\llbracket m \in TFin(S, next); next \in increasing(S) \rrbracket$
 $\implies \forall n \in TFin(S, next). n \leq m \longrightarrow n = m \mid next'n \subseteq m$
 $\langle proof \rangle$

a more convenient form for Lemma 2

lemma *TFin-subsetD*:

$\llbracket n \leq m; m \in TFin(S, next); n \in TFin(S, next); next \in increasing(S) \rrbracket$
 $\implies n = m \mid next'n \subseteq m$
 $\langle proof \rangle$

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*:

$\llbracket m \in TFin(S, next); n \in TFin(S, next); next \in increasing(S) \rrbracket$
 $\implies n \subseteq m \mid m \leq n$
 $\langle proof \rangle$

Lemma 3 of section 3.3

lemma *equal-next-upper*:

$\llbracket n \in TFin(S, next); m \in TFin(S, next); m = next'm \rrbracket \implies n \subseteq m$
 $\langle proof \rangle$

Property 3.3 of section 3.3

lemma *equal-next-Union*:

$\llbracket m \in TFin(S, next); next \in increasing(S) \rrbracket$
 $\implies m = next'm \leftrightarrow m = \bigcup(TFin(S, next))$
 $\langle proof \rangle$

37.4 Hausdorff's Theorem: Every Set Contains a Maximal Chain

NOTE: We assume the partial ordering is \subseteq , the subset relation!

* Defining the "next" operation for Hausdorff's Theorem *

lemma *chain-subset-Pow*: $chain(A) \subseteq Pow(A)$
 $\langle proof \rangle$

lemma *super-subset-chain*: $super(A,c) \subseteq chain(A)$
 $\langle proof \rangle$

lemma *maxchain-subset-chain*: $maxchain(A) \subseteq chain(A)$
 $\langle proof \rangle$

lemma *choice-super*:

$\llbracket ch \in (\prod X \in Pow(chain(S)) - \{0\}. X); X \in chain(S); X \notin maxchain(S) \rrbracket$
 $\implies ch \text{ ' } super(S,X) \in super(S,X)$
 $\langle proof \rangle$

lemma *choice-not-equals*:

$\llbracket ch \in (\prod X \in Pow(chain(S)) - \{0\}. X); X \in chain(S); X \notin maxchain(S) \rrbracket$
 $\implies ch \text{ ' } super(S,X) \neq X$
 $\langle proof \rangle$

This justifies Definition 4.4

lemma *Hausdorff-next-exists*:

$ch \in (\prod X \in Pow(chain(S)) - \{0\}. X) \implies$
 $\exists next \in increasing(S). \forall X \in Pow(S).$
 $next \text{ ' } X = if(X \in chain(S) - maxchain(S), ch \text{ ' } super(S,X), X)$
 $\langle proof \rangle$

Lemma 4

lemma *TFin-chain-lemma4*:

$\llbracket c \in TFin(S,next);$
 $ch \in (\prod X \in Pow(chain(S)) - \{0\}. X);$
 $next \in increasing(S);$
 $\forall X \in Pow(S). next \text{ ' } X =$
 $if(X \in chain(S) - maxchain(S), ch \text{ ' } super(S,X), X) \rrbracket$
 $\implies c \in chain(S)$
 $\langle proof \rangle$

theorem *Hausdorff*: $\exists c. c \in maxchain(S)$
 $\langle proof \rangle$

37.5 Zorn's Lemma: If All Chains in S Have Upper Bounds In S, then S contains a Maximal Element

Used in the proof of Zorn's Lemma

lemma *chain-extend*:

$\llbracket c \in \text{chain}(A); z \in A; \forall x \in c. x \leq z \rrbracket \implies \text{cons}(z, c) \in \text{chain}(A)$
 $\langle \text{proof} \rangle$

lemma *Zorn*: $\forall c \in \text{chain}(S). \bigcup(c) \in S \implies \exists y \in S. \forall z \in S. y \leq z \longrightarrow y = z$
 $\langle \text{proof} \rangle$

Alternative version of Zorn's Lemma

theorem *Zorn2*:

$\forall c \in \text{chain}(S). \exists y \in S. \forall x \in c. x \subseteq y \implies \exists y \in S. \forall z \in S. y \leq z \longrightarrow y = z$
 $\langle \text{proof} \rangle$

37.6 Zermelo's Theorem: Every Set can be Well-Ordered

Lemma 5

lemma *TFin-well-lemma5*:

$\llbracket n \in \text{TFin}(S, \text{next}); Z \subseteq \text{TFin}(S, \text{next}); z \in Z; \neg \bigcap(Z) \in Z \rrbracket$
 $\implies \forall m \in Z. n \subseteq m$
 $\langle \text{proof} \rangle$

Well-ordering of $\text{TFin}(S, \text{next})$

lemma *well-ord-TFin-lemma*: $\llbracket Z \subseteq \text{TFin}(S, \text{next}); z \in Z \rrbracket \implies \bigcap(Z) \in Z$
 $\langle \text{proof} \rangle$

This theorem just packages the previous result

lemma *well-ord-TFin*:

$\text{next} \in \text{increasing}(S)$
 $\implies \text{well-ord}(\text{TFin}(S, \text{next}), \text{Subset-rel}(\text{TFin}(S, \text{next})))$
 $\langle \text{proof} \rangle$

* Defining the "next" operation for Zermelo's Theorem *

lemma *choice-Diff*:

$\llbracket \text{ch} \in (\prod X \in \text{Pow}(S) - \{0\}. X); X \subseteq S; X \neq S \rrbracket \implies \text{ch}'(S - X) \in S - X$
 $\langle \text{proof} \rangle$

This justifies Definition 6.1

lemma *Zermelo-next-exists*:

$\text{ch} \in (\prod X \in \text{Pow}(S) - \{0\}. X) \implies$
 $\exists \text{next} \in \text{increasing}(S). \forall X \in \text{Pow}(S).$
 $\text{next}'X = (\text{if } X = S \text{ then } S \text{ else } \text{cons}(\text{ch}'(S - X), X))$
 $\langle \text{proof} \rangle$

The construction of the injection

lemma *choice-imp-injection*:

$\llbracket \text{ch} \in (\prod X \in \text{Pow}(S) - \{0\}. X);$
 $\text{next} \in \text{increasing}(S);$
 $\forall X \in \text{Pow}(S). \text{next}'X = \text{if}(X = S, S, \text{cons}(\text{ch}'(S - X), X)) \rrbracket$

$$\implies (\lambda x \in S. \bigcup (\{y \in TFin(S, next). x \notin y\}))$$

$$\in inj(S, TFin(S, next) - \{S\})$$

$$\langle proof \rangle$$

The wellordering theorem

theorem *AC-well-ord*: $\exists r. well\text{-}ord(S, r)$
 $\langle proof \rangle$

37.7 Zorn's Lemma for Partial Orders

Reimported from HOL by Clemens Ballarin.

definition *Chain* :: $i \Rightarrow i$ **where**
 $Chain(r) = \{A \in Pow(field(r)). \forall a \in A. \forall b \in A. \langle a, b \rangle \in r \mid \langle b, a \rangle \in r\}$

lemma *mono-Chain*:
 $r \subseteq s \implies Chain(r) \subseteq Chain(s)$
 $\langle proof \rangle$

theorem *Zorn-po*:
assumes *po*: *Partial-order*(r)
and *u*: $\forall C \in Chain(r). \exists u \in field(r). \forall a \in C. \langle a, u \rangle \in r$
shows $\exists m \in field(r). \forall a \in field(r). \langle m, a \rangle \in r \longrightarrow a = m$
 $\langle proof \rangle$

end

38 Cardinal Arithmetic Using AC

theory *Cardinal-AC* **imports** *CardinalArith* *Zorn* **begin**

38.1 Strengthened Forms of Existing Theorems on Cardinals

lemma *cardinal-eqpoll*: $|A| \approx A$
 $\langle proof \rangle$

The theorem $||A|| = |A|$

lemmas *cardinal-idem* = *cardinal-eqpoll* [*THEN* *cardinal-cong*, *simp*]

lemma *cardinal-eqE*: $|X| = |Y| \implies X \approx Y$
 $\langle proof \rangle$

lemma *cardinal-eqpoll-iff*: $|X| = |Y| \longleftrightarrow X \approx Y$
 $\langle proof \rangle$

lemma *cardinal-disjoint-Un*:

$$[|A|=|B|; |C|=|D|; A \cap C = 0; B \cap D = 0]$$

$$\implies |A \cup C| = |B \cup D|$$
 $\langle proof \rangle$

lemma *lepoll-imp-cardinal-le*: $A \lesssim B \implies |A| \leq |B|$
 $\langle \text{proof} \rangle$

lemma *cadd-assoc*: $(i \oplus j) \oplus k = i \oplus (j \oplus k)$
 $\langle \text{proof} \rangle$

lemma *cmult-assoc*: $(i \otimes j) \otimes k = i \otimes (j \otimes k)$
 $\langle \text{proof} \rangle$

lemma *cadd-cmult-distrib*: $(i \oplus j) \otimes k = (i \otimes k) \oplus (j \otimes k)$
 $\langle \text{proof} \rangle$

lemma *InfCard-square-eq*: $\text{InfCard}(|A|) \implies A * A \approx A$
 $\langle \text{proof} \rangle$

38.2 The relationship between cardinality and le-pollence

lemma *Card-le-imp-lepoll*:
assumes $|A| \leq |B|$ **shows** $A \lesssim B$
 $\langle \text{proof} \rangle$

lemma *le-Card-iff*: $\text{Card}(K) \implies |A| \leq K \longleftrightarrow A \lesssim K$
 $\langle \text{proof} \rangle$

lemma *cardinal-0-iff-0* [simp]: $|A| = 0 \longleftrightarrow A = 0$
 $\langle \text{proof} \rangle$

lemma *cardinal-lt-iff-lesspoll*:
assumes $i: \text{Ord}(i)$ **shows** $i < |A| \longleftrightarrow i \prec A$
 $\langle \text{proof} \rangle$

lemma *cardinal-le-imp-lepoll*: $i \leq |A| \implies i \lesssim A$
 $\langle \text{proof} \rangle$

38.3 Other Applications of AC

lemma *surj-implies-inj*:
assumes $f: f \in \text{surj}(X, Y)$ **shows** $\exists g. g \in \text{inj}(Y, X)$
 $\langle \text{proof} \rangle$

Kunen's Lemma 10.20

lemma *surj-implies-cardinal-le*:
assumes $f: f \in \text{surj}(X, Y)$ **shows** $|Y| \leq |X|$
 $\langle \text{proof} \rangle$

Kunen's Lemma 10.21

lemma *cardinal-UN-le*:
assumes $K: \text{InfCard}(K)$

shows $(\bigwedge i. i \in K \implies |X(i)| \leq K) \implies |\bigcup i \in K. X(i)| \leq K$
 $\langle proof \rangle$

The same again, using *csucc*

lemma *cardinal-UN-lt-csucc*:

$$\llbracket InfCard(K); \bigwedge i. i \in K \implies |X(i)| < csucc(K) \rrbracket$$

$$\implies |\bigcup i \in K. X(i)| < csucc(K)$$
 $\langle proof \rangle$

The same again, for a union of ordinals. In use, $j(i)$ is a bit like $\text{rank}(i)$, the least ordinal j such that $i:V_{\text{from}(A,j)}$.

lemma *cardinal-UN-Ord-lt-csucc*:

$$\llbracket InfCard(K); \bigwedge i. i \in K \implies j(i) < csucc(K) \rrbracket$$

$$\implies (\bigcup i \in K. j(i)) < csucc(K)$$
 $\langle proof \rangle$

38.4 The Main Result for Infinite-Branching Datatypes

As above, but the index set need not be a cardinal. Work backwards along the injection from W into K , given that $W \neq \emptyset$.

lemma *inj-UN-subset*:
assumes $f: f \in \text{inj}(A, B)$ **and** $a: a \in A$
shows $(\bigcup x \in A. C(x)) \subseteq (\bigcup y \in B. C(\text{if } y \in \text{range}(f) \text{ then } \text{converse}(f) 'y \text{ else } a))$
 $\langle proof \rangle$

theorem *le-UN-Ord-lt-csucc*:
assumes $IK: InfCard(K)$ **and** $WK: |W| \leq K$ **and** $j: \bigwedge w. w \in W \implies j(w) < csucc(K)$
shows $(\bigcup w \in W. j(w)) < csucc(K)$
 $\langle proof \rangle$

end

39 Infinite-Branching Datatype Definitions

theory *InfDatatype* **imports** *Datatype Univ Finite Cardinal-AC* **begin**

lemmas *fun-Limit-VfromE* =
 $Limit-VfromE \ [OF \ apply-funtype \ InfCard-csucc \ [THEN \ InfCard-is-Limit]]$

lemma *fun-Vcsucc-lemma*:
assumes $f: f \in D \rightarrow V_{\text{from}}(A, csucc(K))$ **and** $DK: |D| \leq K$ **and** $ICK: InfCard(K)$
shows $\exists j. f \in D \rightarrow V_{\text{from}}(A, j) \wedge j < csucc(K)$
 $\langle proof \rangle$

lemma *subset-Vcsucc*:

$$\begin{aligned} & \llbracket D \subseteq V_{\text{from}}(A, \text{csucc}(K)); |D| \leq K; \text{InfCard}(K) \rrbracket \\ & \implies \exists j. D \subseteq V_{\text{from}}(A, j) \wedge j < \text{csucc}(K) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *fun-Vcsucc*:

$$\begin{aligned} & \llbracket |D| \leq K; \text{InfCard}(K); D \subseteq V_{\text{from}}(A, \text{csucc}(K)) \rrbracket \implies \\ & D \rightarrow V_{\text{from}}(A, \text{csucc}(K)) \subseteq V_{\text{from}}(A, \text{csucc}(K)) \end{aligned}$$
 $\langle \text{proof} \rangle$

lemma *fun-in-Vcsucc*:

$$\begin{aligned} & \llbracket f: D \rightarrow V_{\text{from}}(A, \text{csucc}(K)); |D| \leq K; \text{InfCard}(K); \\ & D \subseteq V_{\text{from}}(A, \text{csucc}(K)) \rrbracket \\ & \implies f: V_{\text{from}}(A, \text{csucc}(K)) \end{aligned}$$
 $\langle \text{proof} \rangle$

Remove \subseteq from the rule above

lemmas *fun-in-Vcsucc' = fun-in-Vcsucc* [OF - - - subsetI]

lemma *Card-fun-Vcsucc*:

$$\text{InfCard}(K) \implies K \rightarrow V_{\text{from}}(A, \text{csucc}(K)) \subseteq V_{\text{from}}(A, \text{csucc}(K))$$
 $\langle \text{proof} \rangle$

lemma *Card-fun-in-Vcsucc*:

$$\llbracket f: K \rightarrow V_{\text{from}}(A, \text{csucc}(K)); \text{InfCard}(K) \rrbracket \implies f: V_{\text{from}}(A, \text{csucc}(K))$$
 $\langle \text{proof} \rangle$

lemma *Limit-csucc*: $\text{InfCard}(K) \implies \text{Limit}(\text{csucc}(K))$

$\langle \text{proof} \rangle$

lemmas *Pair-in-Vcsucc = Pair-in-VLimit* [OF - - Limit-csucc]

lemmas *Inl-in-Vcsucc = Inl-in-VLimit* [OF - Limit-csucc]

lemmas *Inr-in-Vcsucc = Inr-in-VLimit* [OF - Limit-csucc]

lemmas *zero-in-Vcsucc = Limit-csucc* [THEN zero-in-VLimit]

lemmas *nat-into-Vcsucc = nat-into-VLimit* [OF - Limit-csucc]

lemmas *InfCard-nat-Un-cardinal = InfCard-Un* [OF InfCard-nat Card-cardinal]

lemmas *le-nat-Un-cardinal =*

Un-upper2-le [OF Ord-nat Card-cardinal [THEN Card-is-Ord]]

lemmas *UN-upper-cardinal = UN-upper* [THEN subset-imp-lepoll, THEN lepoll-imp-cardinal-le]

```

lemmas Data-Arg-intros =
  SigmaI InlI InrI
  Pair-in-univ Inl-in-univ Inr-in-univ
  zero-in-univ A-into-univ nat-into-univ UnCI

```

```

lemmas inf-datatype-intros =
  InfCard-nat InfCard-nat-Un-cardinal
  Pair-in-Vcsucc Inl-in-Vcsucc Inr-in-Vcsucc
  zero-in-Vcsucc A-into-Vfrom nat-into-Vcsucc
  Card-fun-in-Vcsucc fun-in-Vcsucc' UN-I

```

```

end
theory ZFC imports ZF InfDatatype
begin

end

```