

# Computational Algebra

January 18, 2026

## Contents

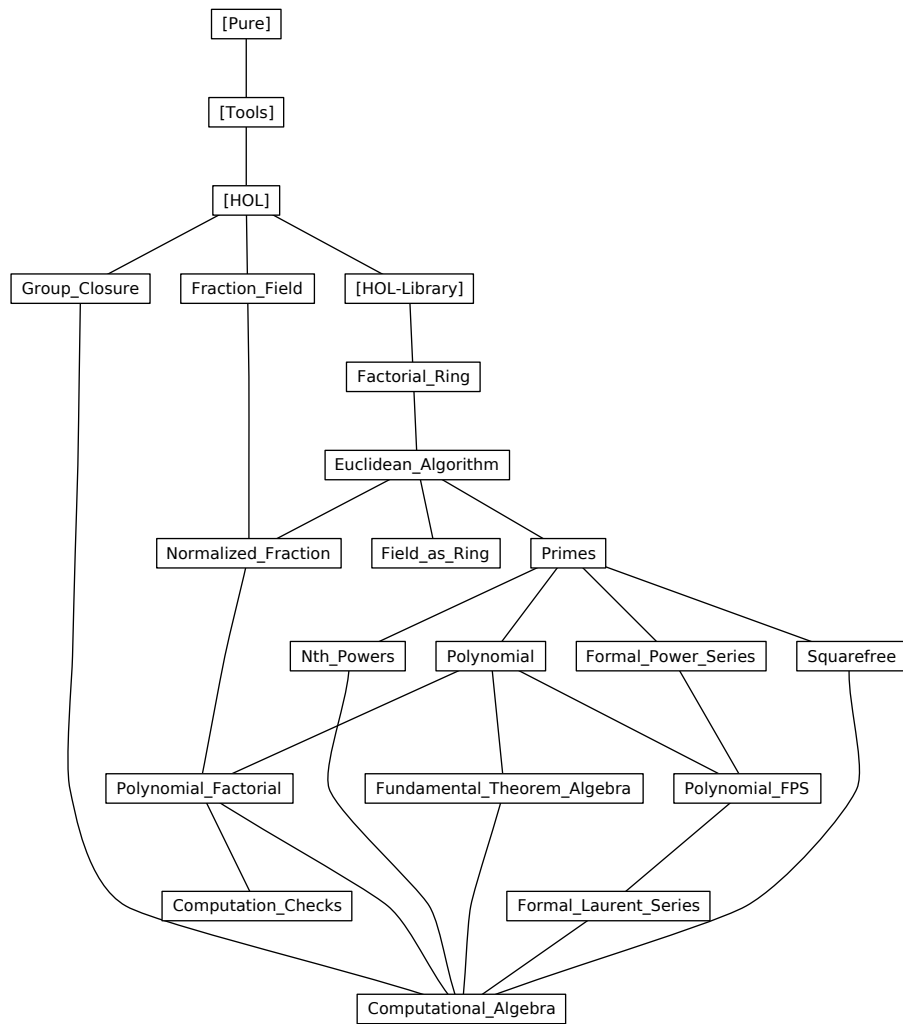
<b>1</b>	<b>Factorial (semi)rings</b>	<b>6</b>
1.1	Irreducible and prime elements . . . . .	6
1.2	Generalized primes: normalized prime elements . . . . .	10
1.3	In a semiring with GCD, each irreducible element is a prime element . . . . .	13
1.4	Factorial semirings: algebraic structures with unique prime factorizations . . . . .	14
1.5	GCD and LCM computation with unique factorizations . . . .	24
<b>2</b>	<b>Abstract euclidean algorithm in euclidean (semi)rings</b>	<b>29</b>
2.1	Generic construction of the (simple) euclidean algorithm . . . .	29
2.2	The (simple) euclidean algorithm as gcd computation . . . . .	31
2.3	The extended euclidean algorithm . . . . .	32
2.4	Typical instances . . . . .	34
<b>3</b>	<b>Primes</b>	<b>35</b>
3.1	Primes on <i>nat</i> and <i>int</i> . . . . .	35
3.2	Make prime naively executable . . . . .	37
3.3	Largest exponent of a prime factor . . . . .	38
3.4	Infinitely many primes . . . . .	39
3.5	Powers of Primes . . . . .	39
3.6	Chinese Remainder Theorem Variants . . . . .	40
3.7	Multiplicity and primality for natural numbers and integers .	41
3.8	Rings and fields with prime characteristic . . . . .	44
3.9	Finite fields . . . . .	45
3.10	The Freshman's Dream in rings of prime characteristic . . . .	46
<b>4</b>	<b>Polynomials as type over a ring structure</b>	<b>47</b>
4.1	Auxiliary: operations for lists (later) representing coefficients	47
4.2	Definition of type <i>poly</i> . . . . .	48
4.3	Degree of a polynomial . . . . .	48
4.4	The zero polynomial . . . . .	49

4.5	List-style constructor for polynomials . . . . .	50
4.6	Quickcheck generator for polynomials . . . . .	51
4.7	List-style syntax for polynomials . . . . .	51
4.8	Representation of polynomials by lists of coefficients . . . . .	52
4.9	Fold combinator for polynomials . . . . .	55
4.10	Canonical morphism on polynomials – evaluation . . . . .	55
4.11	Monomials . . . . .	56
4.12	Leading coefficient . . . . .	57
4.13	Addition and subtraction . . . . .	57
4.14	Multiplication by a constant, polynomial multiplication and the unit polynomial . . . . .	60
4.15	Mapping polynomials . . . . .	66
4.16	Conversions . . . . .	69
4.17	Lemmas about divisibility . . . . .	70
4.18	Polynomials form an integral domain . . . . .	70
4.19	Polynomials form an ordered integral domain . . . . .	72
4.20	Synthetic division and polynomial roots . . . . .	73
	4.20.1 Synthetic division . . . . .	73
	4.20.2 Polynomial roots . . . . .	74
	4.20.3 Order of polynomial roots . . . . .	76
4.21	Additional induction rules on polynomials . . . . .	79
4.22	Composition of polynomials . . . . .	79
4.23	Closure properties of coefficients . . . . .	82
4.24	Shifting polynomials . . . . .	82
4.25	Truncating polynomials . . . . .	83
4.26	Reflecting polynomials . . . . .	83
4.27	Derivatives . . . . .	85
4.28	Algebraic numbers . . . . .	91
4.29	Algebraic integers . . . . .	94
4.30	Division of polynomials . . . . .	95
	4.30.1 Division in general . . . . .	95
	4.30.2 Pseudo-Division . . . . .	98
	4.30.3 Division in polynomials over fields . . . . .	99
	4.30.4 List-based versions for fast implementation . . . . .	104
	4.30.5 Improved Code-Equations for Polynomial (Pseudo) Di- vision . . . . .	106
4.31	Primality and irreducibility in polynomial rings . . . . .	109
4.32	Content and primitive part of a polynomial . . . . .	109
4.33	A typeclass for algebraically closed fields . . . . .	112
4.34	Polynomials and limits . . . . .	114

<b>5</b>	<b>A formalization of formal power series</b>	<b>114</b>
5.1	The type of formal power series . . . . .	114
5.2	Subdegrees . . . . .	117
5.3	Ring structure . . . . .	120
5.4	Shifting and slicing . . . . .	128
5.5	Metrizability . . . . .	133
5.6	The topology of formal power series . . . . .	134
5.7	Division . . . . .	135
5.8	Computing reciprocals via Hensel lifting . . . . .	152
5.9	Euclidean division . . . . .	152
5.10	Formal Derivatives . . . . .	154
5.11	Powers . . . . .	157
5.12	Finite and infinite products . . . . .	159
5.13	Integration . . . . .	159
5.14	Composition . . . . .	162
5.15	Rules from Herbert Wilf's Generatingfunctionology . . . . .	162
5.15.1	Rule 1 . . . . .	162
5.15.2	Rule 2 . . . . .	162
5.15.3	Rule 3 . . . . .	163
5.15.4	Rule 5 — summation and “division” by $1 - X$ . . . . .	163
5.15.5	Rule 4 in its more general form . . . . .	163
5.16	Radicals . . . . .	166
5.17	Chain rule . . . . .	168
5.18	Compositional inverses . . . . .	168
5.19	Elementary series . . . . .	173
5.19.1	Exponential series . . . . .	173
5.19.2	Logarithmic series . . . . .	174
5.19.3	Binomial series . . . . .	175
5.19.4	Trigonometric functions . . . . .	177
5.20	Hypergeometric series . . . . .	180
<b>6</b>	<b>Converting polynomials to formal power series</b>	<b>182</b>
<b>7</b>	<b>A formalization of formal Laurent series</b>	<b>187</b>
7.1	The type of formal Laurent series . . . . .	187
7.1.1	Type definition . . . . .	187
7.1.2	Definition of basic Laurent series . . . . .	188
7.2	Subdegrees . . . . .	190
7.3	Shifting . . . . .	191
7.3.1	Shift definition . . . . .	191
7.3.2	Base factor . . . . .	193
7.4	Conversion between formal power and Laurent series . . . . .	194
7.4.1	Converting Laurent to power series . . . . .	194
7.4.2	Converting power to Laurent series . . . . .	198

7.5	Algebraic structures . . . . .	200
7.5.1	Addition . . . . .	200
7.5.2	Subtraction and negatives . . . . .	201
7.5.3	Multiplication . . . . .	203
7.5.4	Powers . . . . .	211
7.5.5	Inverses . . . . .	216
7.5.6	Division . . . . .	229
7.5.7	Units . . . . .	233
7.6	Composition . . . . .	234
7.7	Formal differentiation and integration . . . . .	238
7.7.1	Derivative . . . . .	238
7.7.2	Algebraic rules of the derivative . . . . .	240
7.7.3	Equality of derivatives . . . . .	241
7.7.4	Residues . . . . .	242
7.7.5	Integral definition and basic properties . . . . .	243
7.7.6	Algebraic rules of the integral . . . . .	245
7.7.7	Derivatives of integrals and vice versa . . . . .	246
7.8	Topology . . . . .	247
7.9	Notation . . . . .	248
<b>8</b>	<b>The fraction field of any integral domain</b>	<b>248</b>
8.1	General fractions construction . . . . .	248
8.1.1	Construction of the type of fractions . . . . .	248
8.1.2	Representation and basic operations . . . . .	249
8.1.3	The field of rational numbers . . . . .	251
8.1.4	The ordered field of fractions over an ordered idom . . . . .	251
<b>9</b>	<b>Fundamental Theorem of Algebra</b>	<b>253</b>
9.1	More lemmas about module of complex numbers . . . . .	254
9.2	Basic lemmas about polynomials . . . . .	254
9.3	Fundamental theorem of algebra . . . . .	255
9.4	Nullstellensatz, degrees and divisibility of polynomials . . . . .	256
<b>10</b>	<b><math>n</math>-th powers and roots of naturals</b>	<b>267</b>
10.1	The set of $n$ -th powers . . . . .	267
10.2	The $n$ -root of a natural number . . . . .	269
<b>11</b>	<b>Polynomials, fractions and rings</b>	<b>271</b>
11.1	Lifting elements into the field of fractions . . . . .	271
11.2	Lifting polynomial coefficients to the field of fractions . . . . .	272
11.3	Fractional content . . . . .	273
11.4	Polynomials over a field are a Euclidean ring . . . . .	274
11.5	Primality and irreducibility in polynomial rings . . . . .	275
11.6	Prime factorisation of polynomials . . . . .	276

11.7 Typeclass instances . . . . .	276
11.8 Polynomial GCD . . . . .	277
<b>12 Squarefreeness</b>	<b>278</b>
<b>13 Pieces of computational Algebra</b>	<b>283</b>
<b>14 Computation checks</b>	<b>286</b>



# 1 Factorial (semi)rings

```
theory Factorial-Ring
imports
  Main
  HOL-Library.Multiset
begin
```

```
unbundle multiset.lifting
```

## 1.1 Irreducible and prime elements

```
context comm-semiring-1
begin
```

```
definition irreducible :: 'a  $\Rightarrow$  bool where
  irreducible p  $\longleftrightarrow$  p  $\neq$  0  $\wedge$   $\neg$ p dvd 1  $\wedge$  ( $\forall$  a b. p = a * b  $\longrightarrow$  a dvd 1  $\vee$  b dvd 1)
```

```
lemma not-irreducible-zero [simp]:  $\neg$ irreducible 0
  <proof>
```

```
lemma irreducible-not-unit: irreducible p  $\implies$   $\neg$ p dvd 1
  <proof>
```

```
lemma not-irreducible-one [simp]:  $\neg$ irreducible 1
  <proof>
```

```
lemma irreducibleI:
  p  $\neq$  0  $\implies$   $\neg$ p dvd 1  $\implies$  ( $\wedge$  a b. p = a * b  $\implies$  a dvd 1  $\vee$  b dvd 1)  $\implies$  irreducible
p
  <proof>
```

```
lemma irreducibleD: irreducible p  $\implies$  p = a * b  $\implies$  a dvd 1  $\vee$  b dvd 1
  <proof>
```

```
lemma irreducible-mono:
  assumes irr: irreducible b and a dvd b  $\neg$ a dvd 1
  shows irreducible a
  <proof>
```

```
lemma irreducible-multD:
  assumes l: irreducible (a*b)
  shows a dvd 1  $\wedge$  irreducible b  $\vee$  b dvd 1  $\wedge$  irreducible a
  <proof>
```

```
lemma irreducible-power-iff [simp]:
  irreducible (p  $\wedge$  n)  $\longleftrightarrow$  irreducible p  $\wedge$  n = 1
  <proof>
```

**definition** *prime-elem* :: 'a  $\Rightarrow$  bool **where**  
 $\text{prime-elem } p \longleftrightarrow p \neq 0 \wedge \neg p \text{ dvd } 1 \wedge (\forall a \ b. \ p \text{ dvd } (a * b) \longrightarrow p \text{ dvd } a \vee p \text{ dvd } b)$

**lemma** *not-prime-elem-zero* [simp]:  $\neg \text{prime-elem } 0$   
 ⟨proof⟩

**lemma** *prime-elem-not-unit*:  $\text{prime-elem } p \Longrightarrow \neg p \text{ dvd } 1$   
 ⟨proof⟩

**lemma** *prime-elemI*:  
 $p \neq 0 \Longrightarrow \neg p \text{ dvd } 1 \Longrightarrow (\bigwedge a \ b. \ p \text{ dvd } (a * b) \Longrightarrow p \text{ dvd } a \vee p \text{ dvd } b) \Longrightarrow \text{prime-elem } p$   
 ⟨proof⟩

**lemma** *prime-elem-dvd-multD*:  
 $\text{prime-elem } p \Longrightarrow p \text{ dvd } (a * b) \Longrightarrow p \text{ dvd } a \vee p \text{ dvd } b$   
 ⟨proof⟩

**lemma** *prime-elem-dvd-mult-iff*:  
 $\text{prime-elem } p \Longrightarrow p \text{ dvd } (a * b) \longleftrightarrow p \text{ dvd } a \vee p \text{ dvd } b$   
 ⟨proof⟩

**lemma** *not-prime-elem-one* [simp]:  
 $\neg \text{prime-elem } 1$   
 ⟨proof⟩

**lemma** *prime-elem-not-zeroI*:  
**assumes** *prime-elem* *p*  
**shows**  $p \neq 0$   
 ⟨proof⟩

**lemma** *prime-elem-dvd-power*:  
 $\text{prime-elem } p \Longrightarrow p \text{ dvd } x^n \Longrightarrow p \text{ dvd } x$   
 ⟨proof⟩

**lemma** *prime-elem-dvd-power-iff*:  
 $\text{prime-elem } p \Longrightarrow n > 0 \Longrightarrow p \text{ dvd } x^n \longleftrightarrow p \text{ dvd } x$   
 ⟨proof⟩

**lemma** *prime-elem-imp-nonzero* [simp]:  
**ASSUMPTION**  $(\text{prime-elem } x) \Longrightarrow x \neq 0$   
 ⟨proof⟩

**lemma** *prime-elem-imp-not-one* [simp]:  
**ASSUMPTION**  $(\text{prime-elem } x) \Longrightarrow x \neq 1$   
 ⟨proof⟩

**end**

**lemma** (in *normalization-semidom*) *irreducible-cong*:  
 assumes *normalize a = normalize b*  
 shows *irreducible a  $\longleftrightarrow$  irreducible b*  
 <proof>

**lemma** (in *normalization-semidom*) *associatedE1*:  
 assumes *normalize a = normalize b*  
 obtains *u where is-unit u a = u \* b*  
 <proof>

**lemma** (in *normalization-semidom*) *associatedE2*:  
 assumes *normalize a = normalize b*  
 obtains *u where is-unit u b = u \* a*  
 <proof>

**lemma** (in *normalization-semidom*) *normalize-power-normalize*:  
*normalize (normalize x  $\wedge$  n) = normalize (x  $\wedge$  n)*  
 <proof>

**context** *algebraic-semidom*  
**begin**

**lemma** *prime-elem-imp-irreducible*:  
 assumes *prime-elem p*  
 shows *irreducible p*  
 <proof>

**lemma** (in *algebraic-semidom*) *unit-imp-no-irreducible-divisors*:  
 assumes *is-unit x irreducible p*  
 shows  $\neg p \text{ dvd } x$   
 <proof>

**lemma** *unit-imp-no-prime-divisors*:  
 assumes *is-unit x prime-elem p*  
 shows  $\neg p \text{ dvd } x$   
 <proof>

**lemma** *prime-elem-mono*:  
 assumes *prime-elem p  $\neg q \text{ dvd } 1 \text{ } q \text{ dvd } p$*   
 shows *prime-elem q*  
 <proof>

**lemma** *irreducibleD'*:  
 assumes *irreducible a b dvd a*  
 shows *a dvd b  $\vee$  is-unit b*



*<proof>*

**lemma** *irreducibleI'*:

assumes  $a \neq 0 \neg is-unit\ a \wedge b.\ b\ dvd\ a \implies a\ dvd\ b \vee is-unit\ b$   
shows *irreducible*  $a$

*<proof>*

**lemma** *irreducible-altdef*:

*irreducible*  $x \iff x \neq 0 \wedge \neg is-unit\ x \wedge (\forall b.\ b\ dvd\ x \longrightarrow x\ dvd\ b \vee is-unit\ b)$

*<proof>*

**lemma** *prime-elem-multD*:

assumes *prime-elem*  $(a * b)$   
shows  $is-unit\ a \vee is-unit\ b$

*<proof>*

**lemma** *prime-elemD2*:

assumes *prime-elem*  $p$  and  $a\ dvd\ p$  and  $\neg is-unit\ a$   
shows  $p\ dvd\ a$

*<proof>*

**lemma** *prime-elem-dvd-prod-msetE*:

assumes *prime-elem*  $p$   
assumes *dvd*:  $p\ dvd\ prod-mset\ A$   
obtains  $a$  where  $a \in \# A$  and  $p\ dvd\ a$

*<proof>*

**context**

**begin**

**lemma** *prime-elem-powerD*:

assumes *prime-elem*  $(p \wedge n)$   
shows *prime-elem*  $p \wedge n = 1$

*<proof>*

**lemma** *prime-elem-power-iff*:

*prime-elem*  $(p \wedge n) \iff prime-elem\ p \wedge n = 1$

*<proof>*

**end**

**lemma** *irreducible-mult-unit-left*:

$is-unit\ a \implies irreducible\ (a * p) \iff irreducible\ p$

*<proof>*

**lemma** *prime-elem-mult-unit-left*:

$is-unit\ a \implies prime-elem\ (a * p) \iff prime-elem\ p$

*<proof>*

**lemma** *prime-elem-dvd-cases*:  
 assumes  $pk: p*k \text{ dvd } m*n$  **and**  $p: \text{prime-elem } p$   
 shows  $(\exists x. k \text{ dvd } x*n \wedge m = p*x) \vee (\exists y. k \text{ dvd } m*y \wedge n = p*y)$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-power-dvd-prod*:  
 assumes  $pc: p^c \text{ dvd } m*n$  **and**  $p: \text{prime-elem } p$   
 shows  $\exists a \ b. a+b = c \wedge p^a \text{ dvd } m \wedge p^b \text{ dvd } n$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-power-dvd-cases*:  
 assumes  $p^c \text{ dvd } m * n$  **and**  $a + b = \text{Suc } c$  **and**  $\text{prime-elem } p$   
 shows  $p^a \text{ dvd } m \vee p^b \text{ dvd } n$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-not-unit' [simp]*:  
 ASSUMPTION  $(\text{prime-elem } x) \implies \neg \text{is-unit } x$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-dvd-power-iff*:  
 assumes  $\text{prime-elem } p$   
 shows  $p \text{ dvd } a^{\wedge} n \longleftrightarrow p \text{ dvd } a \wedge n > 0$   
 $\langle \text{proof} \rangle$

**lemma** *prime-power-dvd-multD*:  
 assumes  $\text{prime-elem } p$   
 assumes  $p^{\wedge} n \text{ dvd } a * b$  **and**  $n > 0$  **and**  $\neg p \text{ dvd } a$   
 shows  $p^{\wedge} n \text{ dvd } b$   
 $\langle \text{proof} \rangle$

**end**

## 1.2 Generalized primes: normalized prime elements

**context** *normalization-semidom*  
**begin**

**lemma** *irreducible-normalized-divisors*:  
 assumes  $\text{irreducible } x \ y \text{ dvd } x \text{ normalize } y = y$   
 shows  $y = 1 \vee y = \text{normalize } x$   
 $\langle \text{proof} \rangle$

**lemma** *irreducible-normalize-iff [simp]*:  $\text{irreducible } (\text{normalize } x) = \text{irreducible } x$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-normalize-iff [simp]*:  $\text{prime-elem } (\text{normalize } x) = \text{prime-elem } x$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-associated*:

**assumes** *prime-elem p and prime-elem q and q dvd p*

**shows** *normalize q = normalize p*

*<proof>*

**definition** *prime* :: '*a*  $\Rightarrow$  bool **where**

*prime p  $\longleftrightarrow$  prime-elem p  $\wedge$  normalize p = p*

**lemma** *not-prime-0* [*simp*]:  $\neg$ *prime 0* *<proof>*

**lemma** *not-prime-unit*: *is-unit x  $\Rightarrow$   $\neg$ prime x*

*<proof>*

**lemma** *not-prime-1* [*simp*]:  $\neg$ *prime 1* *<proof>*

**lemma** *primeI*: *prime-elem x  $\Rightarrow$  normalize x = x  $\Rightarrow$  prime x*

*<proof>*

**lemma** *prime-imp-prime-elem* [*dest*]: *prime p  $\Rightarrow$  prime-elem p*

*<proof>*

**lemma** *normalize-prime*: *prime p  $\Rightarrow$  normalize p = p*

*<proof>*

**lemma** *prime-normalize-iff* [*simp*]: *prime (normalize p)  $\longleftrightarrow$  prime-elem p*

*<proof>*

**lemma** *prime-power-iff*:

*prime (p  $\wedge$  n)  $\longleftrightarrow$  prime p  $\wedge$  n = 1*

*<proof>*

**lemma** *prime-imp-nonzero* [*simp*]:

*ASSUMPTION (prime x)  $\Rightarrow$  x  $\neq$  0*

*<proof>*

**lemma** *prime-imp-not-one* [*simp*]:

*ASSUMPTION (prime x)  $\Rightarrow$  x  $\neq$  1*

*<proof>*

**lemma** *prime-not-unit'* [*simp*]:

*ASSUMPTION (prime x)  $\Rightarrow$   $\neg$ is-unit x*

*<proof>*

**lemma** *prime-normalize'* [*simp*]: *ASSUMPTION (prime x)  $\Rightarrow$  normalize x = x*

*<proof>*

**lemma** *unit-factor-prime*: *prime x  $\Rightarrow$  unit-factor x = 1*

*<proof>*

**lemma** *unit-factor-prime'* [simp]: ASSUMPTION (*prime*  $x$ )  $\implies$  *unit-factor*  $x = 1$

*<proof>*

**lemma** *prime-imp-prime-elem'* [simp]: ASSUMPTION (*prime*  $x$ )  $\implies$  *prime-elem*  $x$

*<proof>*

**lemma** *prime-dvd-multD*: *prime*  $p \implies p \text{ dvd } a * b \implies p \text{ dvd } a \vee p \text{ dvd } b$

*<proof>*

**lemma** *prime-dvd-mult-iff*: *prime*  $p \implies p \text{ dvd } a * b \longleftrightarrow p \text{ dvd } a \vee p \text{ dvd } b$

*<proof>*

**lemma** *prime-dvd-power*:

*prime*  $p \implies p \text{ dvd } x ^ n \implies p \text{ dvd } x$

*<proof>*

**lemma** *prime-dvd-power-iff*:

*prime*  $p \implies n > 0 \implies p \text{ dvd } x ^ n \longleftrightarrow p \text{ dvd } x$

*<proof>*

**lemma** *prime-dvd-prod-mset-iff*: *prime*  $p \implies p \text{ dvd prod-mset } A \longleftrightarrow (\exists x. x \in \# A \wedge p \text{ dvd } x)$

*<proof>*

**lemma** *prime-dvd-prod-iff*: *finite*  $A \implies \text{prime } p \implies p \text{ dvd prod } f A \longleftrightarrow (\exists x \in A. p \text{ dvd } f x)$

*<proof>*

**lemma** *primes-dvd-imp-eq*:

**assumes** *prime*  $p$  *prime*  $q$   $p \text{ dvd } q$

**shows**  $p = q$

*<proof>*

**lemma** *prime-dvd-prod-mset-primes-iff*:

**assumes** *prime*  $p \wedge q. q \in \# A \implies \text{prime } q$

**shows**  $p \text{ dvd prod-mset } A \longleftrightarrow p \in \# A$

*<proof>*

**lemma** *prod-mset-primes-dvd-imp-subset*:

**assumes** *prod-mset*  $A \text{ dvd prod-mset } B \wedge p. p \in \# A \implies \text{prime } p \wedge p. p \in \# B \implies \text{prime } p$

**shows**  $A \subseteq \# B$

*<proof>*

**lemma** *prod-mset-dvd-prod-mset-primes-iff*:

**assumes**  $\wedge x. x \in \# A \implies \text{prime } x \wedge x. x \in \# B \implies \text{prime } x$

**shows** *prod-mset*  $A \text{ dvd prod-mset } B \longleftrightarrow A \subseteq \# B$

$\langle \text{proof} \rangle$

**lemma** *is-unit-prod-mset-primes-iff*:

**assumes**  $\bigwedge x. x \in \# A \implies \text{prime } x$

**shows**  $\text{is-unit } (\text{prod-mset } A) \longleftrightarrow A = \{\#\}$

$\langle \text{proof} \rangle$

**lemma** *prod-mset-primes-irreducible-imp-prime*:

**assumes** *irred*:  $\text{irreducible } (\text{prod-mset } A)$

**assumes** *A*:  $\bigwedge x. x \in \# A \implies \text{prime } x$

**assumes** *B*:  $\bigwedge x. x \in \# B \implies \text{prime } x$

**assumes** *C*:  $\bigwedge x. x \in \# C \implies \text{prime } x$

**assumes** *dvd*:  $\text{prod-mset } A \text{ dvd } \text{prod-mset } B * \text{prod-mset } C$

**shows**  $\text{prod-mset } A \text{ dvd } \text{prod-mset } B \vee \text{prod-mset } A \text{ dvd } \text{prod-mset } C$

$\langle \text{proof} \rangle$

**lemma** *prod-mset-primes-finite-divisor-powers*:

**assumes** *A*:  $\bigwedge x. x \in \# A \implies \text{prime } x$

**assumes** *B*:  $\bigwedge x. x \in \# B \implies \text{prime } x$

**assumes**  $A \neq \{\#\}$

**shows**  $\text{finite } \{n. \text{prod-mset } A \wedge^n \text{ dvd } \text{prod-mset } B\}$

$\langle \text{proof} \rangle$

**end**

### 1.3 In a semiring with GCD, each irreducible element is a prime element

**context** *semiring-gcd*

**begin**

**lemma** *irreducible-imp-prime-elem-gcd*:

**assumes** *irreducible*  $x$

**shows** *prime-elem*  $x$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-imp-coprime*:

**assumes** *prime-elem*  $p \neg p \text{ dvd } n$

**shows** *coprime*  $p n$

$\langle \text{proof} \rangle$

**lemma** *prime-imp-coprime*:

**assumes** *prime*  $p \neg p \text{ dvd } n$

**shows** *coprime*  $p n$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-imp-power-coprime*:

*prime-elem*  $p \implies \neg p \text{ dvd } a \implies \text{coprime } a (p \wedge^m)$

$\langle \text{proof} \rangle$

**lemma** *prime-imp-power-coprime*:  
 $\text{prime } p \implies \neg p \text{ dvd } a \implies \text{coprime } a (p \wedge m)$   
 <proof>

**lemma** *prime-elem-divprod-pow*:  
**assumes** *p*: *prime-elem* *p* **and** *ab*: *coprime* *a* *b* **and** *pab*:  $p \wedge n \text{ dvd } a * b$   
**shows**  $p \wedge n \text{ dvd } a \vee p \wedge n \text{ dvd } b$   
 <proof>

**lemma** *primes-coprime*:  
 $\text{prime } p \implies \text{prime } q \implies p \neq q \implies \text{coprime } p \ q$   
 <proof>

**end**

## 1.4 Factorial semirings: algebraic structures with unique prime factorizations

**class** *factorial-semiring* = *normalization-semidom* +  
**assumes** *prime-factorization-exists*:  
 $x \neq 0 \implies \exists A. (\forall x. x \in \# A \longrightarrow \text{prime-elem } x) \wedge \text{normalize } (\text{prod-mset } A) = \text{normalize } x$

Alternative characterization

**lemma** (in *normalization-semidom*) *factorial-semiring-altI-aux*:  
**assumes** *finite-divisors*:  $\bigwedge x. x \neq 0 \implies \text{finite } \{y. y \text{ dvd } x \wedge \text{normalize } y = y\}$   
**assumes** *irreducible-imp-prime-elem*:  $\bigwedge x. \text{irreducible } x \implies \text{prime-elem } x$   
**assumes**  $x \neq 0$   
**shows**  $\exists A. (\forall x. x \in \# A \longrightarrow \text{prime-elem } x) \wedge \text{normalize } (\text{prod-mset } A) = \text{normalize } x$   
 <proof>

**lemma** *factorial-semiring-altI*:  
**assumes** *finite-divisors*:  $\bigwedge x::'a. x \neq 0 \implies \text{finite } \{y. y \text{ dvd } x \wedge \text{normalize } y = y\}$   
**assumes** *irreducible-imp-prime*:  $\bigwedge x::'a. \text{irreducible } x \implies \text{prime-elem } x$   
**shows** *OFCLASS*('a :: *normalization-semidom*, *factorial-semiring-class*)  
 <proof>

Properties

**context** *factorial-semiring*  
**begin**

**lemma** *prime-factorization-exists'*:  
**assumes**  $x \neq 0$   
**obtains** *A* **where**  $\bigwedge x. x \in \# A \implies \text{prime } x \text{ normalize } (\text{prod-mset } A) = \text{normalize } x$   
 <proof>

**lemma** *irreducible-imp-prime-elem*:

**assumes** *irreducible*  $x$

**shows** *prime-elem*  $x$

$\langle$ *proof* $\rangle$

**lemma** *finite-divisor-powers*:

**assumes**  $y \neq 0$   $\neg$ *is-unit*  $x$

**shows** *finite*  $\{n. x^n \text{ dvd } y\}$

$\langle$ *proof* $\rangle$

**lemma** *finite-prime-divisors*:

**assumes**  $x \neq 0$

**shows** *finite*  $\{p. \text{prime } p \wedge p \text{ dvd } x\}$

$\langle$ *proof* $\rangle$

**lemma** *infinite-unit-divisor-powers*:

**assumes**  $y \neq 0$

**assumes** *is-unit*  $x$

**shows** *infinite*  $\{n. x^n \text{ dvd } y\}$

$\langle$ *proof* $\rangle$

**corollary** *is-unit-iff-infinite-divisor-powers*:

**assumes**  $y \neq 0$

**shows** *is-unit*  $x \longleftrightarrow$  *infinite*  $\{n. x^n \text{ dvd } y\}$

$\langle$ *proof* $\rangle$

**lemma** *prime-elem-iff-irreducible*: *prime-elem*  $x \longleftrightarrow$  *irreducible*  $x$

$\langle$ *proof* $\rangle$

**lemma** *prime-divisor-exists*:

**assumes**  $a \neq 0$   $\neg$ *is-unit*  $a$

**shows**  $\exists b. b \text{ dvd } a \wedge \text{prime } b$

$\langle$ *proof* $\rangle$

**lemma** *prime-divisors-induct* [*case-names zero unit factor*]:

**assumes**  $P\ 0 \wedge x. \text{is-unit } x \implies P\ x \wedge p. \text{prime } p \implies P\ x \implies P\ (p * x)$

**shows**  $P\ x$

$\langle$ *proof* $\rangle$

**lemma** *no-prime-divisors-imp-unit*:

**assumes**  $a \neq 0 \wedge b. b \text{ dvd } a \implies \text{normalize } b = b \implies \neg \text{prime-elem } b$

**shows** *is-unit*  $a$

$\langle$ *proof* $\rangle$

**lemma** *prime-divisorE*:

**assumes**  $a \neq 0$  **and**  $\neg \text{is-unit } a$

**obtains**  $p$  **where** *prime*  $p$  **and**  $p \text{ dvd } a$

$\langle$ *proof* $\rangle$

**definition** *multiplicity* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  nat **where**  
*multiplicity* p x = (if finite {n. p  $\wedge$  n dvd x} then Max {n. p  $\wedge$  n dvd x} else 0)

**lemma** *multiplicity-dvd*: p  $\wedge$  *multiplicity* p x dvd x  
 <proof>

**lemma** *multiplicity-dvd'*: n  $\leq$  *multiplicity* p x  $\implies$  p  $\wedge$  n dvd x  
 <proof>

**context**  
 fixes x p :: 'a  
 assumes xp: x  $\neq$  0  $\neg$ is-unit p  
**begin**

**lemma** *multiplicity-eq-Max*: *multiplicity* p x = Max {n. p  $\wedge$  n dvd x}  
 <proof>

**lemma** *multiplicity-geI*:  
 assumes p  $\wedge$  n dvd x  
 shows *multiplicity* p x  $\geq$  n  
 <proof>

**lemma** *multiplicity-lessI*:  
 assumes  $\neg$ p  $\wedge$  n dvd x  
 shows *multiplicity* p x < n  
 <proof>

**lemma** *power-dvd-iff-le-multiplicity*:  
 p  $\wedge$  n dvd x  $\longleftrightarrow$  n  $\leq$  *multiplicity* p x  
 <proof>

**lemma** *multiplicity-eq-zero-iff*:  
 shows *multiplicity* p x = 0  $\longleftrightarrow$   $\neg$ p dvd x  
 <proof>

**lemma** *multiplicity-gt-zero-iff*:  
 shows *multiplicity* p x > 0  $\longleftrightarrow$  p dvd x  
 <proof>

**lemma** *multiplicity-decompose*:  
 $\neg$ p dvd (x div p  $\wedge$  *multiplicity* p x)  
 <proof>

**lemma** *multiplicity-decompose'*:  
 obtains y **where** x = p  $\wedge$  *multiplicity* p x \* y  $\neg$ p dvd y  
 <proof>

**end**



**lemma** *multiplicity-zero* [simp]: *multiplicity p 0 = 0*  
 ⟨proof⟩

**lemma** *prime-elem-multiplicity-eq-zero-iff*:  
*prime-elem p  $\implies x \neq 0 \implies \text{multiplicity } p \ x = 0 \longleftrightarrow \neg p \ \text{dvd } x$*   
 ⟨proof⟩

**lemma** *prime-multiplicity-other*:  
**assumes** *prime p prime q p  $\neq$  q*  
**shows** *multiplicity p q = 0*  
 ⟨proof⟩

**lemma** *prime-multiplicity-gt-zero-iff*:  
*prime-elem p  $\implies x \neq 0 \implies \text{multiplicity } p \ x > 0 \longleftrightarrow p \ \text{dvd } x$*   
 ⟨proof⟩

**lemma** *multiplicity-unit-left*: *is-unit p  $\implies \text{multiplicity } p \ x = 0$*   
 ⟨proof⟩

**lemma** *multiplicity-unit-right*:  
**assumes** *is-unit x*  
**shows** *multiplicity p x = 0*  
 ⟨proof⟩

**lemma** *multiplicity-one* [simp]: *multiplicity p 1 = 0*  
 ⟨proof⟩

**lemma** *multiplicity-eqI*:  
**assumes** *p  $\wedge$  n dvd x  $\neg$  p  $\wedge$  Suc n dvd x*  
**shows** *multiplicity p x = n*  
 ⟨proof⟩

**context**  
**fixes** *x p :: 'a*  
**assumes** *xp: x  $\neq$  0  $\neg$  is-unit p*  
**begin**

**lemma** *multiplicity-times-same*:  
**assumes** *p  $\neq$  0*  
**shows** *multiplicity p (p \* x) = Suc (multiplicity p x)*  
 ⟨proof⟩

**end**

**lemma** *multiplicity-same-power'*: *multiplicity p (p  $\wedge$  n) = (if p = 0  $\vee$  is-unit p then 0 else n)*  
 ⟨proof⟩

**lemma** *multiplicity-same-power*:

$p \neq 0 \implies \neg \text{is-unit } p \implies \text{multiplicity } p (p \wedge n) = n$   
*<proof>*

**lemma** *multiplicity-prime-elem-times-other*:

**assumes** *prime-elem*  $p \neg p \text{ dvd } q$   
**shows**  $\text{multiplicity } p (q * x) = \text{multiplicity } p x$   
*<proof>*

**lemma** *multiplicity-self*:

**assumes**  $p \neq 0 \neg \text{is-unit } p$   
**shows**  $\text{multiplicity } p p = 1$   
*<proof>*

**lemma** *multiplicity-times-unit-left*:

**assumes** *is-unit*  $c$   
**shows**  $\text{multiplicity } (c * p) x = \text{multiplicity } p x$   
*<proof>*

**lemma** *multiplicity-times-unit-right*:

**assumes** *is-unit*  $c$   
**shows**  $\text{multiplicity } p (c * x) = \text{multiplicity } p x$   
*<proof>*

**lemma** *multiplicity-normalize-left* [*simp*]:

$\text{multiplicity } (\text{normalize } p) x = \text{multiplicity } p x$   
*<proof>*

**lemma** *multiplicity-normalize-right* [*simp*]:

$\text{multiplicity } p (\text{normalize } x) = \text{multiplicity } p x$   
*<proof>*

**lemma** *multiplicity-prime* [*simp*]: *prime-elem*  $p \implies \text{multiplicity } p p = 1$

*<proof>*

**lemma** *multiplicity-prime-power* [*simp*]: *prime-elem*  $p \implies \text{multiplicity } p (p \wedge n)$

$= n$   
*<proof>*

**lift-definition** *prime-factorization* ::  $'a \Rightarrow 'a \text{ multiset}$  **is**

$\lambda x p. \text{ if prime } p \text{ then multiplicity } p x \text{ else } 0$   
*<proof>*

**abbreviation** *prime-factors* ::  $'a \Rightarrow 'a \text{ set}$  **where**

$\text{prime-factors } a \equiv \text{set-mset } (\text{prime-factorization } a)$

**lemma** *count-prime-factorization-nonprime*:

$\neg \text{prime } p \implies \text{count } (\text{prime-factorization } x) p = 0$

$\langle \text{proof} \rangle$

**lemma** *count-prime-factorization-prime:*

*prime*  $p \implies \text{count } (\text{prime-factorization } x) \ p = \text{multiplicity } p \ x$

$\langle \text{proof} \rangle$

**lemma** *count-prime-factorization:*

*count*  $(\text{prime-factorization } x) \ p = (\text{if } \text{prime } p \text{ then } \text{multiplicity } p \ x \text{ else } 0)$

$\langle \text{proof} \rangle$

**lemma** *dvd-imp-multiplicity-le:*

**assumes**  $a \text{ dvd } b \ b \neq 0$

**shows**  $\text{multiplicity } p \ a \leq \text{multiplicity } p \ b$

$\langle \text{proof} \rangle$

**lemma** *prime-power-inj:*

**assumes** *prime*  $a \ a^m = a^n$

**shows**  $m = n$

$\langle \text{proof} \rangle$

**lemma** *prime-power-inj':*

**assumes** *prime*  $p \ \text{prime } q$

**assumes**  $p^m = q^n \ m > 0 \ n > 0$

**shows**  $p = q \ m = n$

$\langle \text{proof} \rangle$

**lemma** *prime-power-eq-one-iff* [simp]: *prime*  $p \implies p^n = 1 \longleftrightarrow n = 0$

$\langle \text{proof} \rangle$

**lemma** *one-eq-prime-power-iff* [simp]: *prime*  $p \implies 1 = p^n \longleftrightarrow n = 0$

$\langle \text{proof} \rangle$

**lemma** *prime-power-inj'':*

**assumes** *prime*  $p \ \text{prime } q$

**shows**  $p^m = q^n \longleftrightarrow (m = 0 \wedge n = 0) \vee (p = q \wedge m = n)$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-0* [simp]: *prime-factorization*  $0 = \{\#\}$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-empty-iff:*

*prime-factorization*  $x = \{\#\} \longleftrightarrow x = 0 \vee \text{is-unit } x$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-unit:*

**assumes** *is-unit*  $x$

**shows** *prime-factorization*  $x = \{\#\}$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-1* [simp]: *prime-factorization 1 = {#}*  
 ⟨proof⟩

**lemma** *prime-factorization-times-prime*:  
 assumes  $x \neq 0$  *prime*  $p$   
 shows *prime-factorization*  $(p * x) = \{ \#p\# \} + \text{prime-factorization } x$   
 ⟨proof⟩

**lemma** *prod-mset-prime-factorization-weak*:  
 assumes  $x \neq 0$   
 shows *normalize* (*prod-mset* (*prime-factorization*  $x$ )) = *normalize*  $x$   
 ⟨proof⟩

**lemma** *in-prime-factors-iff*:  
 $p \in \text{prime-factors } x \longleftrightarrow x \neq 0 \wedge p \text{ dvd } x \wedge \text{prime } p$   
 ⟨proof⟩

**lemma** *in-prime-factors-imp-prime* [intro]:  
 $p \in \text{prime-factors } x \implies \text{prime } p$   
 ⟨proof⟩

**lemma** *in-prime-factors-imp-dvd* [dest]:  
 $p \in \text{prime-factors } x \implies p \text{ dvd } x$   
 ⟨proof⟩

**lemma** *prime-factorsI*:  
 $x \neq 0 \implies \text{prime } p \implies p \text{ dvd } x \implies p \in \text{prime-factors } x$   
 ⟨proof⟩

**lemma** *prime-factors-dvd*:  
 $x \neq 0 \implies \text{prime-factors } x = \{ p. \text{prime } p \wedge p \text{ dvd } x \}$   
 ⟨proof⟩

**lemma** *prime-factors-multiplicity*:  
 $\text{prime-factors } n = \{ p. \text{prime } p \wedge \text{multiplicity } p \ n > 0 \}$   
 ⟨proof⟩

**lemma** *prime-factorization-prime*:  
 assumes *prime*  $p$   
 shows *prime-factorization*  $p = \{ \#p\# \}$   
 ⟨proof⟩

**lemma** *prime-factorization-prod-mset-primes*:  
 assumes  $\bigwedge p. p \in \# \ A \implies \text{prime } p$   
 shows *prime-factorization* (*prod-mset*  $A$ ) =  $A$   
 ⟨proof⟩

**lemma** *prime-factorization-cong*:  
 $\text{normalize } x = \text{normalize } y \implies \text{prime-factorization } x = \text{prime-factorization } y$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique*:

**assumes**  $x \neq 0 \ y \neq 0$

**shows**  $\text{prime-factorization } x = \text{prime-factorization } y \longleftrightarrow \text{normalize } x = \text{normalize } y$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-normalize [simp]*:

$\text{prime-factorization } (\text{normalize } x) = \text{prime-factorization } x$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-eqI-strong*:

**assumes**  $\bigwedge p. p \in \# P \implies \text{prime } p \ \text{prod-mset } P = n$

**shows**  $\text{prime-factorization } n = P$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-eqI*:

**assumes**  $\bigwedge p. p \in \# P \implies \text{prime } p \ \text{normalize } (\text{prod-mset } P) = \text{normalize } n$

**shows**  $\text{prime-factorization } n = P$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-mult*:

**assumes**  $x \neq 0 \ y \neq 0$

**shows**  $\text{prime-factorization } (x * y) = \text{prime-factorization } x + \text{prime-factorization } y$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-prod*:

**assumes**  $\text{finite } A \ \bigwedge x. x \in A \implies f \ x \neq 0$

**shows**  $\text{prime-factorization } (\text{prod } f \ A) = (\sum n \in A. \text{prime-factorization } (f \ n))$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-multiplicity-mult-distrib*:

**assumes**  $\text{prime-elem } p \ x \neq 0 \ y \neq 0$

**shows**  $\text{multiplicity } p \ (x * y) = \text{multiplicity } p \ x + \text{multiplicity } p \ y$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-multiplicity-prod-mset-distrib*:

**assumes**  $\text{prime-elem } p \ 0 \notin \# A$

**shows**  $\text{multiplicity } p \ (\text{prod-mset } A) = \text{sum-mset } (\text{image-mset } (\text{multiplicity } p) \ A)$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-multiplicity-power-distrib*:

**assumes**  $\text{prime-elem } p \ x \neq 0$

**shows**  $\text{multiplicity } p \ (x \wedge^n) = n * \text{multiplicity } p \ x$

$\langle \text{proof} \rangle$

**lemma** *prime-elem-multiplicity-prod-distrib*:

**assumes** *prime-elem*  $p \neq 0 \notin f \cdot A$  *finite*  $A$

**shows**  $\text{multiplicity } p (\text{prod } f \ A) = (\sum_{x \in A. \text{multiplicity } p (f \ x)})$

*<proof>*

**lemma** *multiplicity-distinct-prime-power*:

$\text{prime } p \implies \text{prime } q \implies p \neq q \implies \text{multiplicity } p (q \wedge n) = 0$

*<proof>*

**lemma** *prime-factorization-prime-power*:

$\text{prime } p \implies \text{prime-factorization } (p \wedge n) = \text{replicate-mset } n \ p$

*<proof>*

**lemma** *prime-factorization-subset-iff-dvd*:

**assumes** *[simp]*:  $x \neq 0 \ y \neq 0$

**shows**  $\text{prime-factorization } x \subseteq \# \text{ prime-factorization } y \longleftrightarrow x \text{ dvd } y$

*<proof>*

**lemma** *prime-factorization-subset-imp-dvd*:

$x \neq 0 \implies (\text{prime-factorization } x \subseteq \# \text{ prime-factorization } y) \implies x \text{ dvd } y$

*<proof>*

**lemma** *prime-factorization-divide*:

**assumes**  $b \text{ dvd } a$

**shows**  $\text{prime-factorization } (a \text{ div } b) = \text{prime-factorization } a - \text{prime-factorization } b$

*<proof>*

**lemma** *zero-not-in-prime-factors* *[simp]*:  $0 \notin \text{prime-factors } x$

*<proof>*

**lemma** *prime-prime-factors*:

$\text{prime } p \implies \text{prime-factors } p = \{p\}$

*<proof>*

**lemma** *prime-factors-product*:

$x \neq 0 \implies y \neq 0 \implies \text{prime-factors } (x * y) = \text{prime-factors } x \cup \text{prime-factors } y$

*<proof>*

**lemma** *dvd-prime-factors* *[intro]*:

$y \neq 0 \implies x \text{ dvd } y \implies \text{prime-factors } x \subseteq \text{prime-factors } y$

*<proof>*

**lemma** *multiplicity-le-imp-dvd*:

**assumes**  $x \neq 0 \ \bigwedge p. \text{prime } p \implies \text{multiplicity } p \ x \leq \text{multiplicity } p \ y$

**shows**  $x \text{ dvd } y$

*<proof>*

**lemma** *dvd-multiplicity-eq*:

$x \neq 0 \implies y \neq 0 \implies x \text{ dvd } y \iff (\forall p. \text{multiplicity } p \ x \leq \text{multiplicity } p \ y)$   
 $\langle \text{proof} \rangle$

**lemma** *multiplicity-eq-imp-eq*:

**assumes**  $x \neq 0 \ y \neq 0$   
**assumes**  $\bigwedge p. \text{prime } p \implies \text{multiplicity } p \ x = \text{multiplicity } p \ y$   
**shows**  $\text{normalize } x = \text{normalize } y$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique'*:

**assumes**  $\forall p \in \# M. \text{prime } p \ \forall p \in \# N. \text{prime } p \ (\prod i \in \# M. i) = (\prod i \in \# N. i)$   
**shows**  $M = N$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique''*:

**assumes**  $\forall p \in \# M. \text{prime } p \ \forall p \in \# N. \text{prime } p \ \text{normalize } (\prod i \in \# M. i) = \text{normalize } (\prod i \in \# N. i)$   
**shows**  $M = N$   
 $\langle \text{proof} \rangle$

**lemma** *multiplicity-cong*:

$(\bigwedge r. p \wedge r \text{ dvd } a \iff p \wedge r \text{ dvd } b) \implies \text{multiplicity } p \ a = \text{multiplicity } p \ b$   
 $\langle \text{proof} \rangle$

**lemma** *not-dvd-imp-multiplicity-0*:

**assumes**  $\neg p \text{ dvd } x$   
**shows**  $\text{multiplicity } p \ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *multiplicity-zero-left [simp]*:  $\text{multiplicity } 0 \ x = 0$

$\langle \text{proof} \rangle$

**lemma** *inj-on-Prod-primes*:

**assumes**  $\bigwedge P \ p. P \in A \implies p \in P \implies \text{prime } p$   
**assumes**  $\bigwedge P. P \in A \implies \text{finite } P$   
**shows**  $\text{inj-on } \text{Prod } A$   
 $\langle \text{proof} \rangle$

**lemma** *divides-primelow-weak*:

**assumes**  $\text{prime } p \ \text{and } a \text{ dvd } p \wedge n$   
**obtains**  $m \ \text{where } m \leq n \ \text{and } \text{normalize } a = \text{normalize } (p \wedge m)$   
 $\langle \text{proof} \rangle$

**lemma** *divide-out-primelow-ex*:

**assumes**  $n \neq 0 \ \exists p \in \text{prime-factors } n. P \ p$   
**obtains**  $p \ k \ n' \ \text{where } P \ p \ \text{prime } p \ p \text{ dvd } n \ \neg p \text{ dvd } n' \ k > 0 \ n = p \wedge k * n'$   
 $\langle \text{proof} \rangle$

**lemma** *divide-out-primelow*:

**assumes**  $n \neq 0 \neg \text{is-unit } n$

**obtains**  $p \ k \ n'$  **where** *prime*  $p \ p \ \text{dvd } n \neg p \ \text{dvd } n' \ k > 0 \ n = p \wedge k * n'$

*<proof>*

## 1.5 GCD and LCM computation with unique factorizations

**definition** *gcd-factorial*  $a \ b =$  (if  $a = 0$  then normalize  $b$

else if  $b = 0$  then normalize  $a$

else normalize (prod-mset (prime-factorization  $a \cap \#$  prime-factorization  $b$ )))

**definition** *lcm-factorial*  $a \ b =$  (if  $a = 0 \vee b = 0$  then 0

else normalize (prod-mset (prime-factorization  $a \cup \#$  prime-factorization  $b$ )))

**definition** *Gcd-factorial*  $A =$

(if  $A \subseteq \{0\}$  then 0 else normalize (prod-mset (Inf (prime-factorization ‘ ( $A - \{0\}$ ))))))

**definition** *Lcm-factorial*  $A =$

(if  $A = \{0\}$  then 1

else if  $0 \notin A \wedge \text{subset-mset.bdd-above}$  (prime-factorization ‘ ( $A - \{0\}$ )) then

normalize (prod-mset (Sup (prime-factorization ‘  $A$ )))

else

0)

**lemma** *prime-factorization-gcd-factorial*:

**assumes** [simp]:  $a \neq 0 \ b \neq 0$

**shows** prime-factorization (gcd-factorial  $a \ b$ ) = prime-factorization  $a \cap \#$  prime-factorization  $b$

*<proof>*

**lemma** *prime-factorization-lcm-factorial*:

**assumes** [simp]:  $a \neq 0 \ b \neq 0$

**shows** prime-factorization (lcm-factorial  $a \ b$ ) = prime-factorization  $a \cup \#$  prime-factorization  $b$

*<proof>*

**lemma** *prime-factorization-Gcd-factorial*:

**assumes**  $\neg A \subseteq \{0\}$

**shows** prime-factorization (Gcd-factorial  $A$ ) = Inf (prime-factorization ‘ ( $A - \{0\}$ )))

*<proof>*

**lemma** *prime-factorization-Lcm-factorial*:

**assumes**  $0 \notin A \ \text{subset-mset.bdd-above}$  (prime-factorization ‘  $A$ )

**shows** prime-factorization (Lcm-factorial  $A$ ) = Sup (prime-factorization ‘  $A$ )

*<proof>*



**lemma** *gcd-factorial-commute*:  $\text{gcd-factorial } a \ b = \text{gcd-factorial } b \ a$   
 ⟨proof⟩

**lemma** *gcd-factorial-dvd1*:  $\text{gcd-factorial } a \ b \ \text{dvd } a$   
 ⟨proof⟩

**lemma** *gcd-factorial-dvd2*:  $\text{gcd-factorial } a \ b \ \text{dvd } b$   
 ⟨proof⟩

**lemma** *normalize-gcd-factorial [simp]*:  $\text{normalize } (\text{gcd-factorial } a \ b) = \text{gcd-factorial } a \ b$   
 ⟨proof⟩

**lemma** *normalize-lcm-factorial [simp]*:  $\text{normalize } (\text{lcm-factorial } a \ b) = \text{lcm-factorial } a \ b$   
 ⟨proof⟩

**lemma** *gcd-factorial-greatest*:  $c \ \text{dvd} \ \text{gcd-factorial } a \ b$  **if**  $c \ \text{dvd} \ a$   $c \ \text{dvd} \ b$  **for**  $a \ b \ c$   
 ⟨proof⟩

**lemma** *lcm-factorial-gcd-factorial*:  
 $\text{lcm-factorial } a \ b = \text{normalize } (a * b \ \text{div} \ \text{gcd-factorial } a \ b)$  **for**  $a \ b$   
 ⟨proof⟩

**lemma** *normalize-Gcd-factorial*:  
 $\text{normalize } (\text{Gcd-factorial } A) = \text{Gcd-factorial } A$   
 ⟨proof⟩

**lemma** *Gcd-factorial-eq-0-iff*:  
 $\text{Gcd-factorial } A = 0 \iff A \subseteq \{0\}$   
 ⟨proof⟩

**lemma** *Gcd-factorial-dvd*:  
**assumes**  $x \in A$   
**shows**  $\text{Gcd-factorial } A \ \text{dvd} \ x$   
 ⟨proof⟩

**lemma** *Gcd-factorial-greatest*:  
**assumes**  $\bigwedge y. y \in A \implies x \ \text{dvd} \ y$   
**shows**  $x \ \text{dvd} \ \text{Gcd-factorial } A$   
 ⟨proof⟩

**lemma** *normalize-Lcm-factorial*:  
 $\text{normalize } (\text{Lcm-factorial } A) = \text{Lcm-factorial } A$   
 ⟨proof⟩

**lemma** *Lcm-factorial-eq-0-iff*:  
 $\text{Lcm-factorial } A = 0 \iff 0 \in A \vee \neg \text{subset-mset.bdd-above } (\text{prime-factorization } A)$

```

    <proof>

lemma dvd-Lcm-factorial:
  assumes  $x \in A$ 
  shows  $x \text{ dvd } \text{Lcm-factorial } A$ 
  <proof>

lemma Lcm-factorial-least:
  assumes  $\bigwedge y. y \in A \implies y \text{ dvd } x$ 
  shows  $\text{Lcm-factorial } A \text{ dvd } x$ 
  <proof>

lemmas gcd-lcm-factorial =
  gcd-factorial-dvd1 gcd-factorial-dvd2 gcd-factorial-greatest
  normalize-gcd-factorial lcm-factorial-gcd-factorial
  normalize-Gcd-factorial Gcd-factorial-dvd Gcd-factorial-greatest
  normalize-Lcm-factorial dvd-Lcm-factorial Lcm-factorial-least

end

class factorial-semiring-gcd = factorial-semiring + gcd + Gcd +
  assumes gcd-eq-gcd-factorial:  $\text{gcd } a \ b = \text{gcd-factorial } a \ b$ 
  and lcm-eq-lcm-factorial:  $\text{lcm } a \ b = \text{lcm-factorial } a \ b$ 
  and Gcd-eq-Gcd-factorial:  $\text{Gcd } A = \text{Gcd-factorial } A$ 
  and Lcm-eq-Lcm-factorial:  $\text{Lcm } A = \text{Lcm-factorial } A$ 
begin

lemma prime-factorization-gcd:
  assumes [simp]:  $a \neq 0 \ b \neq 0$ 
  shows  $\text{prime-factorization } (\text{gcd } a \ b) = \text{prime-factorization } a \cap \# \text{prime-factorization } b$ 
  <proof>

lemma prime-factorization-lcm:
  assumes [simp]:  $a \neq 0 \ b \neq 0$ 
  shows  $\text{prime-factorization } (\text{lcm } a \ b) = \text{prime-factorization } a \cup \# \text{prime-factorization } b$ 
  <proof>

lemma prime-factorization-Gcd:
  assumes  $\text{Gcd } A \neq 0$ 
  shows  $\text{prime-factorization } (\text{Gcd } A) = \text{Inf } (\text{prime-factorization } ` (A - \{0\}))$ 
  <proof>

lemma prime-factorization-Lcm:
  assumes  $\text{Lcm } A \neq 0$ 
  shows  $\text{prime-factorization } (\text{Lcm } A) = \text{Sup } (\text{prime-factorization } ` A)$ 
  <proof>

```

**lemma** *prime-factors-gcd* [*simp*]:  
 $a \neq 0 \implies b \neq 0 \implies \text{prime-factors } (\text{gcd } a \ b) =$   
 $\text{prime-factors } a \cap \text{prime-factors } b$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factors-lcm* [*simp*]:  
 $a \neq 0 \implies b \neq 0 \implies \text{prime-factors } (\text{lcm } a \ b) =$   
 $\text{prime-factors } a \cup \text{prime-factors } b$   
 $\langle \text{proof} \rangle$

**subclass** *semiring-gcd*  
 $\langle \text{proof} \rangle$

**subclass** *semiring-Gcd*  
 $\langle \text{proof} \rangle$

**lemma**  
**assumes**  $x \neq 0 \ y \neq 0$   
**shows** *gcd-eq-factorial'*:  
 $\text{gcd } x \ y = \text{normalize } (\prod p \in \text{prime-factors } x \cap \text{prime-factors } y.$   
 $\quad p \wedge \min (\text{multiplicity } p \ x) (\text{multiplicity } p \ y)) \ (\text{is } - = ?\text{rhs1})$   
**and** *lcm-eq-factorial'*:  
 $\text{lcm } x \ y = \text{normalize } (\prod p \in \text{prime-factors } x \cup \text{prime-factors } y.$   
 $\quad p \wedge \max (\text{multiplicity } p \ x) (\text{multiplicity } p \ y)) \ (\text{is } - = ?\text{rhs2})$   
 $\langle \text{proof} \rangle$

**lemma**  
**assumes**  $x \neq 0 \ y \neq 0 \ \text{prime } p$   
**shows** *multiplicity-gcd*:  $\text{multiplicity } p \ (\text{gcd } x \ y) = \min (\text{multiplicity } p \ x)$   
 $(\text{multiplicity } p \ y)$   
**and** *multiplicity-lcm*:  $\text{multiplicity } p \ (\text{lcm } x \ y) = \max (\text{multiplicity } p \ x)$   
 $(\text{multiplicity } p \ y)$   
 $\langle \text{proof} \rangle$

**lemma** *gcd-lcm-distrib*:  
 $\text{gcd } x \ (\text{lcm } y \ z) = \text{lcm } (\text{gcd } x \ y) (\text{gcd } x \ z)$   
 $\langle \text{proof} \rangle$

**lemma** *lcm-gcd-distrib*:  
 $\text{lcm } x \ (\text{gcd } y \ z) = \text{gcd } (\text{lcm } x \ y) (\text{lcm } x \ z)$   
 $\langle \text{proof} \rangle$

**end**

**class** *factorial-ring-gcd* = *factorial-semiring-gcd* + *idom*  
**begin**

**subclass** *ring-gcd*  $\langle \text{proof} \rangle$

**subclass** *idom-divide*  $\langle \text{proof} \rangle$

**end**

**class** *factorial-semiring-multiplicative* =  
*factorial-semiring* + *normalization-semidom-multiplicative*  
**begin**

**lemma** *normalize-prod-mset-primes*:

$(\bigwedge p. p \in \# A \implies \text{prime } p) \implies \text{normalize } (\text{prod-mset } A) = \text{prod-mset } A$   
 $\langle \text{proof} \rangle$

**lemma** *prod-mset-prime-factorization*:

**assumes**  $x \neq 0$

**shows**  $\text{prod-mset } (\text{prime-factorization } x) = \text{normalize } x$

$\langle \text{proof} \rangle$

**lemma** *prime-decomposition: unit-factor*  $x * \text{prod-mset } (\text{prime-factorization } x) = x$

$\langle \text{proof} \rangle$

**lemma** *prod-prime-factors*:

**assumes**  $x \neq 0$

**shows**  $(\prod p \in \text{prime-factors } x. p \wedge \text{multiplicity } p x) = \text{normalize } x$

$\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique''*:

**assumes**  $S\text{-eq: } S = \{p. 0 < f p\}$

**and** *finite*  $S$

**and**  $S: \forall p \in S. \text{prime } p \text{ normalize } n = (\prod p \in S. p \wedge f p)$

**shows**  $S = \text{prime-factors } n \wedge (\forall p. \text{prime } p \longrightarrow f p = \text{multiplicity } p n)$

$\langle \text{proof} \rangle$

**lemma** *divides-primelow*:

**assumes** *prime*  $p$  **and**  $a \text{ dvd } p \wedge n$

**obtains**  $m$  **where**  $m \leq n$  **and**  $\text{normalize } a = p \wedge m$

$\langle \text{proof} \rangle$

**lemma** *Ex-other-prime-factor*:

**assumes**  $n \neq 0$  **and**  $\neg(\exists k. \text{normalize } n = p \wedge k) \text{ prime } p$

**shows**  $\exists q \in \text{prime-factors } n. q \neq p$

$\langle \text{proof} \rangle$

Now a string of results due to Maya Kdzioka

**lemma** *multiplicity-dvd-iff-dvd*:

**assumes**  $x \neq 0$

**shows**  $p \wedge k \text{ dvd } x \longleftrightarrow p \wedge k \text{ dvd } p \wedge \text{multiplicity } p x$

$\langle \text{proof} \rangle$

```

lemma multiplicity-decomposeI:
  assumes  $x = p^{\wedge}k * x'$  and  $\neg p \text{ dvd } x'$  and  $p \neq 0$ 
  shows  $\text{multiplicity } p \ x = k$ 
   $\langle \text{proof} \rangle$ 

lemma multiplicity-sum-lt:
  assumes  $\text{multiplicity } p \ a < \text{multiplicity } p \ b$   $a \neq 0$   $b \neq 0$ 
  shows  $\text{multiplicity } p \ (a + b) = \text{multiplicity } p \ a$ 
   $\langle \text{proof} \rangle$ 

corollary multiplicity-sum-min:
  assumes  $\text{multiplicity } p \ a \neq \text{multiplicity } p \ b$   $a \neq 0$   $b \neq 0$ 
  shows  $\text{multiplicity } p \ (a + b) = \min (\text{multiplicity } p \ a) (\text{multiplicity } p \ b)$ 
   $\langle \text{proof} \rangle$ 

end

lifting-update multiset.lifting
lifting-forget multiset.lifting

end

```

## 2 Abstract euclidean algorithm in euclidean (semi)rings

```

theory Euclidean-Algorithm
  imports Factorial-Ring
begin

```

### 2.1 Generic construction of the (simple) euclidean algorithm

```

class normalization-euclidean-semiring = euclidean-semiring + normalization-semidom
begin

```

```

lemma euclidean-size-normalize [simp]:
   $\text{euclidean-size } (\text{normalize } a) = \text{euclidean-size } a$ 
   $\langle \text{proof} \rangle$ 

```

```

context
begin

```

```

qualified function gcd :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
  where  $\text{gcd } a \ b = (\text{if } b = 0 \text{ then } \text{normalize } a \text{ else } \text{gcd } b \ (a \bmod b))$ 
   $\langle \text{proof} \rangle$ 
termination
   $\langle \text{proof} \rangle$ 

```

```

declare gcd.simps [simp del]

```

```

lemma eucl-induct [case-names zero mod]:
  assumes H1:  $\bigwedge b. P\ b\ 0$ 
  and H2:  $\bigwedge a\ b. b \neq 0 \implies P\ b\ (a \bmod b) \implies P\ a\ b$ 
  shows  $P\ a\ b$ 
<proof> lemma gcd-0:
   $\text{gcd}\ a\ 0 = \text{normalize}\ a$ 
<proof> lemma gcd-mod:
   $a \neq 0 \implies \text{gcd}\ a\ (b \bmod a) = \text{gcd}\ b\ a$ 
<proof> definition lcm :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
  where  $\text{lcm}\ a\ b = \text{normalize}\ (a * b \text{ div } \text{gcd}\ a\ b)$ 

qualified definition Lcm :: 'a set  $\Rightarrow$  'a — Somewhat complicated definition of
Lcm that has the advantage of working for infinite sets as well
where
[code del]:  $\text{Lcm}\ A = (\text{if } \exists l. l \neq 0 \wedge (\forall a \in A. a \text{ dvd } l) \text{ then}$ 
   $\text{let } l = \text{SOME } l. l \neq 0 \wedge (\forall a \in A. a \text{ dvd } l) \wedge \text{euclidean-size } l =$ 
   $(\text{LEAST } n. \exists l. l \neq 0 \wedge (\forall a \in A. a \text{ dvd } l) \wedge \text{euclidean-size } l = n)$ 
   $\text{in } \text{normalize } l$ 
   $\text{else } 0)$ 

qualified definition Gcd :: 'a set  $\Rightarrow$  'a
where [code del]:  $\text{Gcd}\ A = \text{Lcm}\ \{d. \forall a \in A. d \text{ dvd } a\}$ 

lemma semiring-gcd:
  class.semiring-gcd one zero times gcd lcm
  divide plus minus unit-factor normalize
<proof>

interpretation semiring-gcd one zero times gcd lcm
  divide plus minus unit-factor normalize
<proof>

lemma semiring-Gcd:
  class.semiring-Gcd one zero times gcd lcm Gcd Lcm
  divide plus minus unit-factor normalize
<proof>

end

interpretation semiring-Gcd one zero times
  Euclidean-Algorithm.gcd Euclidean-Algorithm.lcm Euclidean-Algorithm.Gcd Eu-
clidean-Algorithm.Lcm
  divide plus minus unit-factor normalize
<proof>

subclass factorial-semiring
<proof>

lemma Gcd-eucl-set [code]:

```

*Euclidean-Algorithm.Gcd* (set *xs*) = fold *Euclidean-Algorithm.gcd xs 0*  
 ⟨proof⟩

**lemma** *Lcm-eucl-set* [code]:  
*Euclidean-Algorithm.Lcm* (set *xs*) = fold *Euclidean-Algorithm.lcm xs 1*  
 ⟨proof⟩

**end**

**lemma** *prime-elem-int-abs-iff* [simp]:  
 fixes *p* :: *int*  
 shows *prime-elem* |*p*|  $\longleftrightarrow$  *prime-elem p*  
 ⟨proof⟩

**lemma** *prime-elem-int-minus-iff* [simp]:  
 fixes *p* :: *int*  
 shows *prime-elem* (− *p*)  $\longleftrightarrow$  *prime-elem p*  
 ⟨proof⟩

**lemma** *prime-int-iff*:  
 fixes *p* :: *int*  
 shows *prime p*  $\longleftrightarrow$  *p* > 0 ∧ *prime-elem p*  
 ⟨proof⟩

## 2.2 The (simple) euclidean algorithm as gcd computation

**class** *euclidean-semiring-gcd* = *normalization-euclidean-semiring* + *gcd* + *Gcd* +  
 assumes *gcd-eucl*: *Euclidean-Algorithm.gcd* = *GCD.gcd*  
 and *lcm-eucl*: *Euclidean-Algorithm.lcm* = *GCD.lcm*  
 assumes *Gcd-eucl*: *Euclidean-Algorithm.Gcd* = *GCD.Gcd*  
 and *Lcm-eucl*: *Euclidean-Algorithm.Lcm* = *GCD.Lcm*  
**begin**

**subclass** *semiring-gcd*  
 ⟨proof⟩

**subclass** *semiring-Gcd*  
 ⟨proof⟩

**subclass** *factorial-semiring-gcd*  
 ⟨proof⟩

**lemma** *gcd-mod-right* [simp]:  
 $a \neq 0 \implies \text{gcd } a (b \bmod a) = \text{gcd } a b$   
 ⟨proof⟩

**lemma** *gcd-mod-left* [simp]:  
 $b \neq 0 \implies \text{gcd } (a \bmod b) b = \text{gcd } a b$   
 ⟨proof⟩

```

lemma euclidean-size-gcd-le1 [simp]:
  assumes  $a \neq 0$ 
  shows  $\text{euclidean-size } (\text{gcd } a \ b) \leq \text{euclidean-size } a$ 
  <proof>

lemma euclidean-size-gcd-le2 [simp]:
   $b \neq 0 \implies \text{euclidean-size } (\text{gcd } a \ b) \leq \text{euclidean-size } b$ 
  <proof>

lemma euclidean-size-gcd-less1:
  assumes  $a \neq 0$  and  $\neg a \text{ dvd } b$ 
  shows  $\text{euclidean-size } (\text{gcd } a \ b) < \text{euclidean-size } a$ 
  <proof>

lemma euclidean-size-gcd-less2:
  assumes  $b \neq 0$  and  $\neg b \text{ dvd } a$ 
  shows  $\text{euclidean-size } (\text{gcd } a \ b) < \text{euclidean-size } b$ 
  <proof>

lemma euclidean-size-lcm-le1:
  assumes  $a \neq 0$  and  $b \neq 0$ 
  shows  $\text{euclidean-size } a \leq \text{euclidean-size } (\text{lcm } a \ b)$ 
  <proof>

lemma euclidean-size-lcm-le2:
   $a \neq 0 \implies b \neq 0 \implies \text{euclidean-size } b \leq \text{euclidean-size } (\text{lcm } a \ b)$ 
  <proof>

lemma euclidean-size-lcm-less1:
  assumes  $b \neq 0$  and  $\neg b \text{ dvd } a$ 
  shows  $\text{euclidean-size } a < \text{euclidean-size } (\text{lcm } a \ b)$ 
  <proof>

lemma euclidean-size-lcm-less2:
  assumes  $a \neq 0$  and  $\neg a \text{ dvd } b$ 
  shows  $\text{euclidean-size } b < \text{euclidean-size } (\text{lcm } a \ b)$ 
  <proof>

end

lemma factorial-euclidean-semiring-gcdI:
  OFCLASS('a::{factorial-semiring-gcd, normalization-euclidean-semiring}, euclidean-semiring-gcd-class)
  <proof>

```

### 2.3 The extended euclidean algorithm

```

class euclidean-ring-gcd = euclidean-semiring-gcd + idom
begin

```



```

subclass euclidean-ring ⟨proof⟩
subclass ring-gcd ⟨proof⟩
subclass factorial-ring-gcd ⟨proof⟩

function euclid-ext-aux :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ ('a × 'a) × 'a
  where euclid-ext-aux s' s t' t r' r = (
    if r = 0 then let c = 1 div unit-factor r' in ((s' * c, t' * c), normalize r')
    else let q = r' div r
      in euclid-ext-aux s (s' - q * s) t (t' - q * t) r (r' mod r)
  )
  ⟨proof⟩
termination
  ⟨proof⟩

abbreviation (input) euclid-ext :: 'a ⇒ 'a ⇒ ('a × 'a) × 'a
  where euclid-ext ≡ euclid-ext-aux 1 0 0 1

lemma
  assumes gcd r' r = gcd a b
  assumes s' * a + t' * b = r'
  assumes s * a + t * b = r
  assumes euclid-ext-aux s' s t' t r' r = ((x, y), c)
  shows euclid-ext-aux-eq-gcd: c = gcd a b
  and euclid-ext-aux-bezout: x * a + y * b = gcd a b
  ⟨proof⟩

declare euclid-ext-aux.simps [simp del]

definition bezout-coefficients :: 'a ⇒ 'a ⇒ 'a × 'a
  where [code]: bezout-coefficients a b = fst (euclid-ext a b)

lemma bezout-coefficients-0:
  bezout-coefficients a 0 = (1 div unit-factor a, 0)
  ⟨proof⟩

lemma bezout-coefficients-left-0:
  bezout-coefficients 0 a = (0, 1 div unit-factor a)
  ⟨proof⟩

lemma bezout-coefficients:
  assumes bezout-coefficients a b = (x, y)
  shows x * a + y * b = gcd a b
  ⟨proof⟩

lemma bezout-coefficients-fst-snd:
  fst (bezout-coefficients a b) * a + snd (bezout-coefficients a b) * b = gcd a b
  ⟨proof⟩

lemma euclid-ext-eq [simp]:

```

```

    euclid-ext a b = (bezout-coefficients a b, gcd a b) (is ?p = ?q)
  <proof>

declare euclid-ext-eq [symmetric, code-unfold]

end

class normalization-euclidean-semiring-multiplicative =
  normalization-euclidean-semiring + normalization-semidom-multiplicative
begin

subclass factorial-semiring-multiplicative <proof>

end

class field-gcd =
  field + unique-euclidean-ring + euclidean-ring-gcd + normalization-semidom-multiplicative
begin

subclass normalization-euclidean-semiring-multiplicative <proof>

subclass normalization-euclidean-semiring <proof>

subclass semiring-gcd-mult-normalize <proof>

end

## 2.4 Typical instances

instance nat :: normalization-euclidean-semiring <proof>

instance nat :: euclidean-semiring-gcd
  <proof>

instance nat :: normalization-euclidean-semiring-multiplicative <proof>

lemma prime-factorization-Suc-0 [simp]: prime-factorization (Suc 0) = {#}
  <proof>

instance int :: normalization-euclidean-semiring <proof>

instance int :: euclidean-ring-gcd
  <proof>

instance int :: normalization-euclidean-semiring-multiplicative <proof>

lemma (in idom) prime-CHAR-semidom:
  assumes CHAR('a) > 0
  shows prime CHAR('a)

```

$\langle proof \rangle$

end

### 3 Primes

**theory** *Primes*  
**imports** *Euclidean-Algorithm*  
**begin**

#### 3.1 Primes on *nat* and *int*

**lemma** *Suc-0-not-prime-nat* [*simp*]:  $\neg \text{prime } (\text{Suc } 0)$   
 $\langle proof \rangle$

**lemma** *prime-ge-2-nat*:  
 $p \geq 2$  **if** *prime* *p* **for**  $p :: \text{nat}$   
 $\langle proof \rangle$

**lemma** *prime-ge-2-int*:  
 $p \geq 2$  **if** *prime* *p* **for**  $p :: \text{int}$   
 $\langle proof \rangle$

**lemma** *prime-ge-0-int*:  $\text{prime } p \implies p \geq (0 :: \text{int})$   
 $\langle proof \rangle$

**lemma** *prime-gt-0-nat*:  $\text{prime } p \implies p > (0 :: \text{nat})$   
 $\langle proof \rangle$

**lemma** *prime-gt-0-int*:  $\text{prime } p \implies p > (0 :: \text{int})$   
 $\langle proof \rangle$

**lemma** *prime-ge-1-nat*:  $\text{prime } p \implies p \geq (1 :: \text{nat})$   
 $\langle proof \rangle$

**lemma** *prime-ge-Suc-0-nat*:  $\text{prime } p \implies p \geq \text{Suc } 0$   
 $\langle proof \rangle$

**lemma** *prime-ge-1-int*:  $\text{prime } p \implies p \geq (1 :: \text{int})$   
 $\langle proof \rangle$

**lemma** *prime-gt-1-nat*:  $\text{prime } p \implies p > (1 :: \text{nat})$   
 $\langle proof \rangle$

**lemma** *prime-gt-Suc-0-nat*:  $\text{prime } p \implies p > \text{Suc } 0$   
 $\langle proof \rangle$

**lemma** *prime-gt-1-int*:  $\text{prime } p \implies p > (1::\text{int})$   
 ⟨proof⟩

**lemma** *prime-natI*:  
 $\text{prime } p \text{ if } p \geq 2 \text{ and } \bigwedge m n. p \text{ dvd } m * n \implies p \text{ dvd } m \vee p \text{ dvd } n \text{ for } p :: \text{nat}$   
 ⟨proof⟩

**lemma** *prime-intI*:  
 $\text{prime } p \text{ if } p \geq 2 \text{ and } \bigwedge m n. p \text{ dvd } m * n \implies p \text{ dvd } m \vee p \text{ dvd } n \text{ for } p :: \text{int}$   
 ⟨proof⟩

**lemma** *prime-elem-nat-iff* [simp]:  
 $\text{prime-elem } n \longleftrightarrow \text{prime } n \text{ for } n :: \text{nat}$   
 ⟨proof⟩

**lemma** *prime-elem-iff-prime-abs* [simp]:  
 $\text{prime-elem } k \longleftrightarrow \text{prime } |k| \text{ for } k :: \text{int}$   
 ⟨proof⟩

**lemma** *prime-nat-int-transfer* [simp]:  
 $\text{prime } (\text{int } n) \longleftrightarrow \text{prime } n \text{ (is } ?P \longleftrightarrow ?Q)$   
 ⟨proof⟩

**lemma** *prime-nat-iff-prime* [simp]:  
 $\text{prime } (\text{nat } k) \longleftrightarrow \text{prime } k$   
 ⟨proof⟩

**lemma** *prime-int-nat-transfer*:  
 $\text{prime } k \longleftrightarrow k \geq 0 \wedge \text{prime } (\text{nat } k)$   
 ⟨proof⟩

**lemma** *prime-nat-naiveI*:  
 $\text{prime } p \text{ if } p \geq 2 \text{ and dvd: } \bigwedge n. n \text{ dvd } p \implies n = 1 \vee n = p \text{ for } p :: \text{nat}$   
 ⟨proof⟩

**lemma** *prime-int-naiveI*:  
 $\text{prime } p \text{ if } p \geq 2 \text{ and dvd: } \bigwedge k. k \text{ dvd } p \implies |k| = 1 \vee |k| = p \text{ for } p :: \text{int}$   
 ⟨proof⟩

**lemma** *prime-nat-iff*:  
 $\text{prime } (n :: \text{nat}) \longleftrightarrow (1 < n \wedge (\forall m. m \text{ dvd } n \longrightarrow m = 1 \vee m = n))$   
 ⟨proof⟩

**lemma** *prime-nat-iff'*:  
 $\text{prime } (p :: \text{nat}) \longleftrightarrow p > 1 \wedge (\forall n \in \{2..<p\}. \neg n \text{ dvd } p)$   
 ⟨proof⟩

**lemma** *prime-int-iff*:  
 $\text{prime } (n::\text{int}) \longleftrightarrow (1 < n \wedge (\forall m. m \geq 0 \wedge m \text{ dvd } n \longrightarrow m = 1 \vee m = n))$

⟨proof⟩

**lemma** *prime-int-iff'*:

*prime* ( $p :: \text{int}$ )  $\longleftrightarrow p > 1 \wedge (\forall n \in \{2..<p\}. \neg n \text{ dvd } p)$  (**is**  $?P \longleftrightarrow ?Q$ )  
⟨proof⟩

**lemma** *prime-nat-not-dvd*:

**assumes** *prime*  $p$   $p > n$   $n \neq (1::\text{nat})$   
**shows**  $\neg n \text{ dvd } p$   
⟨proof⟩

**lemma** *prime-int-not-dvd*:

**assumes** *prime*  $p$   $p > n$   $n > (1::\text{int})$   
**shows**  $\neg n \text{ dvd } p$   
⟨proof⟩

**lemma** *prime-odd-nat*: *prime*  $p \implies p > (2::\text{nat}) \implies \text{odd } p$

⟨proof⟩

**lemma** *prime-odd-int*: *prime*  $p \implies p > (2::\text{int}) \implies \text{odd } p$

⟨proof⟩

**lemma** *prime-int-altdef*:

*prime*  $p = (1 < p \wedge (\forall m::\text{int}. m \geq 0 \longrightarrow m \text{ dvd } p \longrightarrow m = 1 \vee m = p))$   
⟨proof⟩

**lemma** *not-prime-eq-prod-nat*:

**assumes**  $m > 1 \neg \text{prime } (m::\text{nat})$   
**shows**  $\exists n k. n = m * k \wedge 1 < m \wedge m < n \wedge 1 < k \wedge k < n$   
⟨proof⟩

### 3.2 Make prime naively executable

**lemma** *prime-int-numeral-eq* [simp]:

*prime* (numeral  $m :: \text{int}$ )  $\longleftrightarrow \text{prime}$  (numeral  $m :: \text{nat}$ )  
⟨proof⟩

**class** *check-prime-by-range* = *normalization-semidom* + *discrete-linordered-semidom*  
+

**assumes** *prime-iff*:  $\langle \text{prime } a \longleftrightarrow 1 < a \wedge (\forall d \in \{2..a \text{ div } 2\}. \neg d \text{ dvd } a) \rangle$   
**begin**

**lemma** *two-is-prime* [simp]:

$\langle \text{prime } 2 \rangle$   
⟨proof⟩

**end**

```

lemma divisor-less-eq-half-nat:
   $\langle m \leq n \text{ div } 2 \rangle$  if  $\langle m \text{ dvd } n \rangle$   $\langle m < n \rangle$  for  $m\ n :: \text{nat}$ 
   $\langle \text{proof} \rangle$ 

instance nat :: check-prime-by-range
   $\langle \text{proof} \rangle$ 

lemma two-is-prime-nat [simp]:
   $\langle \text{prime } (2 :: \text{nat}) \rangle$ 
   $\langle \text{proof} \rangle$ 

lemma divisor-less-eq-half-int:
   $\langle k \leq l \text{ div } 2 \rangle$  if  $\langle k \text{ dvd } l \rangle$   $\langle k < l \rangle$   $\langle l \geq 0 \rangle$   $\langle k \geq 0 \rangle$  for  $k\ l :: \text{int}$ 
   $\langle \text{proof} \rangle$ 

instance int :: check-prime-by-range
   $\langle \text{proof} \rangle$ 

lemma prime-nat-numeral-eq [simp]: — TODO Sieve Of Erathosthenes might
speed this up
   $\text{prime } (\text{numeral } m :: \text{nat}) \longleftrightarrow$ 
   $(1 :: \text{nat}) < \text{numeral } m \wedge$ 
   $(\forall n :: \text{nat} \in \text{set } [2..<\text{Suc } (\text{numeral } m \text{ div } 2)]. \neg n \text{ dvd numeral } m)$ 
   $\langle \text{proof} \rangle$ 

context check-prime-by-range
begin

definition check-divisors ::  $\langle 'a \Rightarrow 'a \Rightarrow \text{bool} \rangle$ 
  where  $\langle \text{check-divisors } l\ u\ a \longleftrightarrow (\forall d \in \{l..u\}. \neg d \text{ dvd } a) \rangle$ 

lemma check-divisors-rec [code]:
   $\langle \text{check-divisors } l\ u\ a \longleftrightarrow u < l \vee (\neg l \text{ dvd } a \wedge \text{check-divisors } (l + 1)\ u\ a) \rangle$ 
   $\langle \text{proof} \rangle$ 

lemma prime-eq-check-divisors [code]:
   $\langle \text{prime } a \longleftrightarrow a > 1 \wedge \text{check-divisors } 2\ (a \text{ div } 2)\ a \rangle$ 
   $\langle \text{proof} \rangle$ 

end

```

### 3.3 Largest exponent of a prime factor

```

lemma prime-factor-nat:
   $n \neq (1 :: \text{nat}) \implies \exists p. \text{prime } p \wedge p \text{ dvd } n$ 
   $\langle \text{proof} \rangle$ 

lemma prime-factor-int:
  fixes  $k :: \text{int}$ 

```

**assumes**  $|k| \neq 1$   
**obtains**  $p$  **where**  $\text{prime } p \text{ } p \text{ } dvd \text{ } k$   
 $\langle proof \rangle$

Possibly duplicates other material, but avoid the complexities of multisets.

**lemma** *prime-power-cancel-less*:  
**assumes**  $\text{prime } p$  **and**  $\text{eq: } m * (p \wedge k) = m' * (p \wedge k')$  **and**  $\text{less: } k < k'$  **and**  $\neg p \text{ } dvd \text{ } m$   
**shows**  $\text{False}$   
 $\langle proof \rangle$

**lemma** *prime-power-cancel*:  
**assumes**  $\text{prime } p$  **and**  $\text{eq: } m * (p \wedge k) = m' * (p \wedge k')$  **and**  $\neg p \text{ } dvd \text{ } m \neg p \text{ } dvd \text{ } m'$   
**shows**  $k = k'$   
 $\langle proof \rangle$

**lemma** *prime-power-cancel2*:  
**assumes**  $\text{prime } p \text{ } m * (p \wedge k) = m' * (p \wedge k') \neg p \text{ } dvd \text{ } m \neg p \text{ } dvd \text{ } m'$   
**obtains**  $m = m' \text{ } k = k'$   
 $\langle proof \rangle$

**lemma** *prime-power-canonical*:  
**fixes**  $m :: \text{nat}$   
**assumes**  $\text{prime } p \text{ } m > 0$   
**shows**  $\exists k \text{ } n. \neg p \text{ } dvd \text{ } n \wedge m = n * p \wedge k$   
 $\langle proof \rangle$

### 3.4 Infinitely many primes

**lemma** *next-prime-bound*:  $\exists p :: \text{nat}. \text{prime } p \wedge n < p \wedge p \leq \text{fact } n + 1$   
 $\langle proof \rangle$

**lemma** *bigger-prime*:  $\exists p. \text{prime } p \wedge p > (n :: \text{nat})$   
 $\langle proof \rangle$

**lemma** *primes-infinite*:  $\neg (\text{finite } \{(p :: \text{nat}). \text{prime } p\})$   
 $\langle proof \rangle$

### 3.5 Powers of Primes

Versions for type nat only

**lemma** *prime-product*:  
**fixes**  $p :: \text{nat}$   
**assumes**  $\text{prime } (p * q)$   
**shows**  $p = 1 \vee q = 1$   
 $\langle proof \rangle$

**lemma** *prime-power-mult-nat*:  
**fixes**  $p :: \text{nat}$   
**assumes**  $p$ : *prime*  $p$  **and**  $xy$ :  $x * y = p \wedge k$   
**shows**  $\exists i j. x = p \wedge i \wedge y = p \wedge j$   
 $\langle \text{proof} \rangle$

**lemma** *prime-power-exp-nat*:  
**fixes**  $p :: \text{nat}$   
**assumes**  $p$ : *prime*  $p$  **and**  $n$ :  $n \neq 0$   
**and**  $xn$ :  $x \wedge n = p \wedge k$  **shows**  $\exists i. x = p \wedge i$   
 $\langle \text{proof} \rangle$

**lemma** *divides-primexp-nat*:  
**fixes**  $p :: \text{nat}$   
**assumes**  $p$ : *prime*  $p$   
**shows**  $d \text{ dvd } p \wedge k \longleftrightarrow (\exists i \leq k. d = p \wedge i)$   
 $\langle \text{proof} \rangle$

**lemma** *gcd-prime-int*:  
**assumes** *prime* ( $p :: \text{int}$ )  
**shows**  $\text{gcd } p \ k = (\text{if } p \text{ dvd } k \text{ then } p \text{ else } 1)$   
 $\langle \text{proof} \rangle$

### 3.6 Chinese Remainder Theorem Variants

**lemma** *bezout-gcd-nat*:  
**fixes**  $a :: \text{nat}$  **shows**  $\exists x y. a * x - b * y = \text{gcd } a \ b \vee b * x - a * y = \text{gcd } a \ b$   
 $\langle \text{proof} \rangle$

**lemma** *gcd-bezout-sum-nat*:  
**fixes**  $a :: \text{nat}$   
**assumes**  $a * x + b * y = d$   
**shows**  $\text{gcd } a \ b \text{ dvd } d$   
 $\langle \text{proof} \rangle$

A binary form of the Chinese Remainder Theorem.

**lemma** *chinese-remainder*:  
**fixes**  $a :: \text{nat}$  **assumes**  $ab$ : *coprime*  $a \ b$  **and**  $a$ :  $a \neq 0$  **and**  $b$ :  $b \neq 0$   
**shows**  $\exists x \ q1 \ q2. x = u + q1 * a \wedge x = v + q2 * b$   
 $\langle \text{proof} \rangle$

Primality

**lemma** *coprime-bezout-strong*:  
**fixes**  $a :: \text{nat}$  **assumes** *coprime*  $a \ b$   $b \neq 1$   
**shows**  $\exists x y. a * x = b * y + 1$   
 $\langle \text{proof} \rangle$

**lemma** *bezout-prime*:  
**assumes**  $p$ : *prime*  $p$  **and**  $pa$ :  $\neg p \text{ dvd } a$



**shows**  $\exists x y. a*x = \text{Suc } (p*y)$   
 $\langle \text{proof} \rangle$

### 3.7 Multiplicity and primality for natural numbers and integers

**lemma** *prime-factors-gt-0-nat*:  
 $p \in \text{prime-factors } x \implies p > (0::\text{nat})$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factors-gt-0-int*:  
 $p \in \text{prime-factors } x \implies p > (0::\text{int})$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factors-ge-0-int* [elim]:  
**fixes**  $n :: \text{int}$   
**shows**  $p \in \text{prime-factors } n \implies p \geq 0$   
 $\langle \text{proof} \rangle$

**lemma** *prod-mset-prime-factorization-int*:  
**fixes**  $n :: \text{int}$   
**assumes**  $n > 0$   
**shows**  $\text{prod-mset } (\text{prime-factorization } n) = n$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-exists-nat*:  
 $n > 0 \implies (\exists M. (\forall p::\text{nat} \in \text{set-mset } M. \text{prime } p) \wedge n = (\prod i \in \# M. i))$   
 $\langle \text{proof} \rangle$

**lemma** *prod-mset-prime-factorization-nat* [simp]:  
 $(n::\text{nat}) > 0 \implies \text{prod-mset } (\text{prime-factorization } n) = n$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-nat*:  
 $n > (0::\text{nat}) \implies n = (\prod p \in \text{prime-factors } n. p \wedge \text{multiplicity } p n)$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-int*:  
 $n > (0::\text{int}) \implies n = (\prod p \in \text{prime-factors } n. p \wedge \text{multiplicity } p n)$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique-nat*:  
**fixes**  $f :: \text{nat} \Rightarrow -$   
**assumes**  $S\text{-eq}: S = \{p. 0 < f p\}$   
**and** *finite*  $S$   
**and**  $S: \forall p \in S. \text{prime } p \implies n = (\prod p \in S. p \wedge f p)$   
**shows**  $S = \text{prime-factors } n \wedge (\forall p. \text{prime } p \longrightarrow f p = \text{multiplicity } p n)$   
 $\langle \text{proof} \rangle$

**lemma** *prime-factorization-unique-int*:

**fixes**  $f :: \text{int} \Rightarrow -$

**assumes**  $S\text{-eq}$ :  $S = \{p. 0 < f\ p\}$

**and** *finite*  $S$

**and**  $S$ :  $\forall p \in S. \text{prime } p \text{ abs } n = (\prod_{p \in S}. p \wedge f\ p)$

**shows**  $S = \text{prime-factors } n \wedge (\forall p. \text{prime } p \longrightarrow f\ p = \text{multiplicity } p\ n)$

$\langle \text{proof} \rangle$

**lemma** *prime-factors-characterization-nat*:

$S = \{p. 0 < f\ (p::\text{nat})\} \Longrightarrow$

$\text{finite } S \Longrightarrow \forall p \in S. \text{prime } p \Longrightarrow n = (\prod_{p \in S}. p \wedge f\ p) \Longrightarrow \text{prime-factors } n = S$

$\langle \text{proof} \rangle$

**lemma** *prime-factors-characterization'-nat*:

$\text{finite } \{p. 0 < f\ (p::\text{nat})\} \Longrightarrow$

$(\forall p. 0 < f\ p \longrightarrow \text{prime } p) \Longrightarrow$

$\text{prime-factors } (\prod p \mid 0 < f\ p. p \wedge f\ p) = \{p. 0 < f\ p\}$

$\langle \text{proof} \rangle$

**lemma** *prime-factors-characterization-int*:

$S = \{p. 0 < f\ (p::\text{int})\} \Longrightarrow \text{finite } S \Longrightarrow$

$\forall p \in S. \text{prime } p \Longrightarrow \text{abs } n = (\prod_{p \in S}. p \wedge f\ p) \Longrightarrow \text{prime-factors } n = S$

$\langle \text{proof} \rangle$

**lemma** *abs-prod*:  $\text{abs } (\text{prod } f\ A :: 'a :: \text{linordered-idom}) = \text{prod } (\lambda x. \text{abs } (f\ x))\ A$

$\langle \text{proof} \rangle$

**lemma** *primes-characterization'-int* [rule-format]:

$\text{finite } \{p. p \geq 0 \wedge 0 < f\ (p::\text{int})\} \Longrightarrow \forall p. 0 < f\ p \longrightarrow \text{prime } p \Longrightarrow$

$\text{prime-factors } (\prod p \mid p \geq 0 \wedge 0 < f\ p. p \wedge f\ p) = \{p. p \geq 0 \wedge 0 < f\ p\}$

$\langle \text{proof} \rangle$

**lemma** *multiplicity-characterization-nat*:

$S = \{p. 0 < f\ (p::\text{nat})\} \Longrightarrow \text{finite } S \Longrightarrow \forall p \in S. \text{prime } p \Longrightarrow \text{prime } p \Longrightarrow$

$n = (\prod_{p \in S}. p \wedge f\ p) \Longrightarrow \text{multiplicity } p\ n = f\ p$

$\langle \text{proof} \rangle$

**lemma** *multiplicity-characterization'-nat*:  $\text{finite } \{p. 0 < f\ (p::\text{nat})\} \longrightarrow$

$(\forall p. 0 < f\ p \longrightarrow \text{prime } p) \longrightarrow \text{prime } p \longrightarrow$

$\text{multiplicity } p\ (\prod p \mid 0 < f\ p. p \wedge f\ p) = f\ p$

$\langle \text{proof} \rangle$

**lemma** *multiplicity-characterization-int*:  $S = \{p. 0 < f\ (p::\text{int})\} \Longrightarrow$

$\text{finite } S \Longrightarrow \forall p \in S. \text{prime } p \Longrightarrow \text{prime } p \Longrightarrow n = (\prod_{p \in S}. p \wedge f\ p) \Longrightarrow$

$\text{multiplicity } p\ n = f\ p$

$\langle \text{proof} \rangle$

**lemma** *multiplicity-characterization'-int* [rule-format]:

$finite \{p. p \geq 0 \wedge 0 < f (p::int)\} \implies$   
 $(\forall p. 0 < f p \longrightarrow prime p) \implies prime p \implies$   
 $multiplicity p (\prod p \mid p \geq 0 \wedge 0 < f p. p \wedge f p) = f p$   
 $\langle proof \rangle$

**lemma** *multiplicity-one-nat* [simp]:  $multiplicity p (Suc 0) = 0$   
 $\langle proof \rangle$

**lemma** *multiplicity-eq-nat*:  
**fixes**  $x$  **and**  $y::nat$   
**assumes**  $x > 0 \ y > 0 \ \wedge p. prime p \implies multiplicity p x = multiplicity p y$   
**shows**  $x = y$   
 $\langle proof \rangle$

**lemma** *multiplicity-eq-int*:  
**fixes**  $x \ y :: int$   
**assumes**  $x > 0 \ y > 0 \ \wedge p. prime p \implies multiplicity p x = multiplicity p y$   
**shows**  $x = y$   
 $\langle proof \rangle$

**lemma** *multiplicity-prod-prime-powers*:  
**assumes**  $finite S \ \wedge x. x \in S \implies prime x \ prime p$   
**shows**  $multiplicity p (\prod p \in S. p \wedge f p) = (if p \in S then f p else 0)$   
 $\langle proof \rangle$

**lemma** *prime-factorization-prod-mset*:  
**assumes**  $0 \notin \# A$   
**shows**  $prime-factorization (prod-mset A) = \sum \#(image-mset prime-factorization A)$   
 $\langle proof \rangle$

**lemma** *prime-factors-prod*:  
**assumes**  $finite A$  **and**  $0 \notin f \text{ ' } A$   
**shows**  $prime-factors (prod f A) = \bigcup ((prime-factors \circ f) \text{ ' } A)$   
 $\langle proof \rangle$

**lemma** *prime-factors-fact*:  
 $prime-factors (fact n) = \{p \in \{2..n\}. prime p\}$  (**is**  $?M = ?N$ )  
 $\langle proof \rangle$

**lemma** *prime-dvd-fact-iff*:  
**assumes**  $prime p$   
**shows**  $p \text{ dvd } fact n \longleftrightarrow p \leq n$   
 $\langle proof \rangle$

**lemma** *dvd-choose-prime*:  
**assumes**  $kn: k < n$  **and**  $k: k \neq 0$  **and**  $n: n \neq 0$  **and**  $prime-n: prime n$   
**shows**  $n \text{ dvd } (n \text{ choose } k)$   
 $\langle proof \rangle$

```

lemma (in ring-1) minus-power-prime-CHAR:
  assumes  $p = \text{CHAR}('a)$  prime  $p$ 
  shows  $(-x :: 'a) \wedge p = -(x \wedge p)$ 
   $\langle \text{proof} \rangle$ 

```

### 3.8 Rings and fields with prime characteristic

We introduce some type classes for rings and fields with prime characteristic.

```

class semiring-prime-char = semiring-1 +
  assumes prime-char-aux:  $\exists n. \text{prime } n \wedge \text{of-nat } n = (0 :: 'a)$ 
begin

```

```

lemma CHAR-pos [intro, simp]:  $\text{CHAR}('a) > 0$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma CHAR-nonzero [simp]:  $\text{CHAR}('a) \neq 0$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma CHAR-prime [intro, simp]: prime  $\text{CHAR}('a)$ 
   $\langle \text{proof} \rangle$ 

```

**end**

```

lemma semiring-prime-charI [intro?]:
  prime  $\text{CHAR}('a :: \text{semiring-1}) \implies \text{OFCLASS}('a, \text{semiring-prime-char-class})$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma idom-prime-charI [intro?]:
  assumes  $\text{CHAR}('a :: \text{idom}) > 0$ 
  shows  $\text{OFCLASS}('a, \text{semiring-prime-char-class})$ 
   $\langle \text{proof} \rangle$ 

```

```

class comm-semiring-prime-char = comm-semiring-1 + semiring-prime-char
class comm-ring-prime-char = comm-ring-1 + semiring-prime-char
begin
subclass comm-semiring-prime-char  $\langle \text{proof} \rangle$ 
end
class idom-prime-char = idom + semiring-prime-char
begin
subclass comm-ring-prime-char  $\langle \text{proof} \rangle$ 
end

```

```

class field-prime-char = field +
  assumes pos-char-exists:  $\exists n > 0. \text{of-nat } n = (0 :: 'a)$ 
begin
subclass idom-prime-char
   $\langle \text{proof} \rangle$ 
end

```

**lemma** *field-prime-charI* [intro?]:  
 $n > 0 \implies \text{of-nat } n = (0 :: 'a :: \text{field}) \implies \text{OFCLASS}('a, \text{field-prime-char-class})$   
 <proof>

**lemma** *field-prime-charI'* [intro?]:  
 $\text{CHAR}('a :: \text{field}) > 0 \implies \text{OFCLASS}('a, \text{field-prime-char-class})$   
 <proof>

### 3.9 Finite fields

**class** *finite-field* = *field-prime-char* + *finite*

**lemma** *finite-fieldI* [intro?]:  
 assumes *finite* (*UNIV* :: '*a* :: field set)  
 shows  $\text{OFCLASS}('a, \text{finite-field-class})$   
 <proof>

On a finite field with  $n$  elements, taking the  $n$ -th power of an element is the identity. This is an obvious consequence of the fact that the multiplicative group of the field is a finite group of order  $n - 1$ , so  $x^n = 1$  for any non-zero  $x$ .

Note that this result is sharp in the sense that the multiplicative group of a finite field is cyclic, i.e. it contains an element of order  $n - 1$ . (We don't prove this here.)

**lemma** *finite-field-power-card-eq-same*:  
 fixes  $x :: 'a :: \text{finite-field}$   
 shows  $x^{\text{card } (\text{UNIV} :: 'a \text{ set})} = x$   
 <proof>

**lemma** *finite-field-power-card-power-eq-same*:  
 fixes  $x :: 'a :: \text{finite-field}$   
 assumes  $m = \text{card } (\text{UNIV} :: 'a \text{ set}) \wedge n$   
 shows  $x^m = x^n$   
 <proof>

**class** *enum-finite-field* = *finite-field* +  
 fixes *enum-finite-field* :: nat  $\Rightarrow 'a$   
 assumes *enum-finite-field*:  $\text{enum-finite-field } \cdot \{.. < \text{card } (\text{UNIV} :: 'a \text{ set})\} = \text{UNIV}$   
**begin**

**lemma** *inj-on-enum-finite-field*: *inj-on enum-finite-field*  $\{.. < \text{card } (\text{UNIV} :: 'a \text{ set})\}$   
 <proof>

**end**

To get rid of the pending sort hypotheses, we prove that the field with 2 elements is indeed a finite field.

```

typedef gf2 = {0, 1 :: nat}
  <proof>

```

```

setup-lifting type-definition-gf2

```

```

instantiation gf2 :: field
begin
lift-definition zero-gf2 :: gf2 is 0 <proof>
lift-definition one-gf2 :: gf2 is 1 <proof>
lift-definition uminus-gf2 :: gf2  $\Rightarrow$  gf2 is  $\lambda x. x$  <proof>
lift-definition plus-gf2 :: gf2  $\Rightarrow$  gf2  $\Rightarrow$  gf2 is  $\lambda x y. \text{if } x = y \text{ then } 0 \text{ else } 1$  <proof>
lift-definition minus-gf2 :: gf2  $\Rightarrow$  gf2  $\Rightarrow$  gf2 is  $\lambda x y. \text{if } x = y \text{ then } 0 \text{ else } 1$  <proof>
lift-definition times-gf2 :: gf2  $\Rightarrow$  gf2  $\Rightarrow$  gf2 is  $\lambda x y. x * y$  <proof>
lift-definition inverse-gf2 :: gf2  $\Rightarrow$  gf2 is  $\lambda x. x$  <proof>
lift-definition divide-gf2 :: gf2  $\Rightarrow$  gf2  $\Rightarrow$  gf2 is  $\lambda x y. x * y$  <proof>

```

```

instance
  <proof>

```

```

end

```

```

instance gf2 :: finite-field
  <proof>

```

### 3.10 The Freshman's Dream in rings of prime characteristic

```

lemma (in comm-semiring-1) freshman-dream:

```

```

  fixes x y :: 'a and n :: nat
  assumes prime CHAR('a)
  assumes n-def: n = CHAR('a)
  shows (x + y) ^ n = x ^ n + y ^ n
  <proof>

```

```

lemma (in comm-semiring-1) freshman-dream':

```

```

  assumes [simp]: prime CHAR('a) and m = CHAR('a) ^ n
  shows (x + y :: 'a) ^ m = x ^ m + y ^ m
  <proof>

```

```

lemma (in comm-semiring-1) freshman-dream-sum:

```

```

  fixes f :: 'b  $\Rightarrow$  'a
  assumes prime CHAR('a) and n = CHAR('a)
  shows sum f A ^ n = sum ( $\lambda i. f i$  ^ n) A
  <proof>

```

```

lemma (in comm-semiring-1) freshman-dream-sum':

```

```

  fixes f :: 'b  $\Rightarrow$  'a
  assumes prime CHAR('a) m = CHAR('a) ^ n
  shows sum f A ^ m = sum ( $\lambda i. f i$  ^ m) A
  <proof>

```

```

lemmas prime-imp-coprime-nat = prime-imp-coprime[where ?'a = nat]
lemmas prime-imp-coprime-int = prime-imp-coprime[where ?'a = int]
lemmas prime-dvd-mult-nat = prime-dvd-mult-iff[where ?'a = nat]
lemmas prime-dvd-mult-int = prime-dvd-mult-iff[where ?'a = int]
lemmas prime-dvd-mult-eq-nat = prime-dvd-mult-iff[where ?'a = nat]
lemmas prime-dvd-mult-eq-int = prime-dvd-mult-iff[where ?'a = int]
lemmas prime-dvd-power-nat = prime-dvd-power[where ?'a = nat]
lemmas prime-dvd-power-int = prime-dvd-power[where ?'a = int]
lemmas prime-dvd-power-nat-iff = prime-dvd-power-iff[where ?'a = nat]
lemmas prime-dvd-power-int-iff = prime-dvd-power-iff[where ?'a = int]
lemmas prime-imp-power-coprime-nat = prime-imp-power-coprime[where ?'a =
nat]
lemmas prime-imp-power-coprime-int = prime-imp-power-coprime[where ?'a =
int]
lemmas primes-coprime-nat = primes-coprime[where ?'a = nat]
lemmas primes-coprime-int = primes-coprime[where ?'a = nat]
lemmas prime-divprod-pow-nat = prime-elem-divprod-pow[where ?'a = nat]
lemmas prime-exp = prime-elem-power-iff[where ?'a = nat]

end

```

## 4 Polynomials as type over a ring structure

```

theory Polynomial
imports
  Complex-Main
  HOL-Library.More-List
  HOL-Library.Infinite-Set
  Primes
begin

context semidom-modulo
begin

lemma not-dvd-imp-mod-neq-0:
   $\langle a \bmod b \neq 0 \rangle$  if  $\langle \neg b \text{ dvd } a \rangle$ 
   $\langle \text{proof} \rangle$ 

end

```

### 4.1 Auxiliary: operations for lists (later) representing coefficients

```

definition cCons :: 'a::zero  $\Rightarrow$  'a list  $\Rightarrow$  'a list (infixr  $\langle \#\# \rangle$  65)
  where  $x \#\# xs = (\text{if } xs = [] \wedge x = 0 \text{ then } [] \text{ else } x \# xs)$ 

```

**lemma** *cCons-0-Nil-eq* [simp]:  $0 \#\# [] = []$   
 ⟨proof⟩

**lemma** *cCons-Cons-eq* [simp]:  $x \#\# y \# ys = x \# y \# ys$   
 ⟨proof⟩

**lemma** *cCons-append-Cons-eq* [simp]:  $x \#\# xs @ y \# ys = x \# xs @ y \# ys$   
 ⟨proof⟩

**lemma** *cCons-not-0-eq* [simp]:  $x \neq 0 \implies x \#\# xs = x \# xs$   
 ⟨proof⟩

**lemma** *strip-while-not-0-Cons-eq* [simp]:  
 $strip\_while (\lambda x. x = 0) (x \# xs) = x \#\# strip\_while (\lambda x. x = 0) xs$   
 ⟨proof⟩

**lemma** *tl-cCons* [simp]:  $tl (x \#\# xs) = xs$   
 ⟨proof⟩

## 4.2 Definition of type *poly*

**typedef** (overloaded) *'a poly* =  $\{f :: nat \Rightarrow 'a::zero. \forall_{\infty} n. f\ n = 0\}$   
**morphisms** *coeff Abs-poly*  
 ⟨proof⟩

**setup-lifting** *type-definition-poly*

**lemma** *poly-eq-iff*:  $p = q \longleftrightarrow (\forall n. coeff\ p\ n = coeff\ q\ n)$   
 ⟨proof⟩

**lemma** *poly-eqI*:  $(\bigwedge n. coeff\ p\ n = coeff\ q\ n) \implies p = q$   
 ⟨proof⟩

**lemma** *MOST-coeff-eq-0*:  $\forall_{\infty} n. coeff\ p\ n = 0$   
 ⟨proof⟩

**lemma** *coeff-Abs-poly*:  
**assumes**  $\bigwedge i. i > n \implies f\ i = 0$   
**shows**  $coeff\ (Abs\_poly\ f) = f$   
 ⟨proof⟩

## 4.3 Degree of a polynomial

**definition** *degree* :: *'a::zero poly*  $\Rightarrow$  *nat*  
**where**  $degree\ p = (LEAST\ n. \forall i > n. coeff\ p\ i = 0)$

**lemma** *degree-cong*:  
**assumes**  $\bigwedge i. coeff\ p\ i = 0 \longleftrightarrow coeff\ q\ i = 0$   
**shows**  $degree\ p = degree\ q$



$\langle proof \rangle$

**lemma** *coeff-Abs-poly-If-le*:

*coeff (Abs-poly ( $\lambda i. \text{if } i \leq n \text{ then } f \ i \text{ else } 0$ )) = ( $\lambda i. \text{if } i \leq n \text{ then } f \ i \text{ else } 0$ )*  
 $\langle proof \rangle$

**lemma** *coeff-eq-0*:

**assumes** *degree p < n*  
**shows** *coeff p n = 0*  
 $\langle proof \rangle$

**lemma** *le-degree*: *coeff p n  $\neq$  0  $\implies$  n  $\leq$  degree p*  
 $\langle proof \rangle$

**lemma** *degree-le*:  $\forall i > n. \text{coeff } p \ i = 0 \implies \text{degree } p \leq n$   
 $\langle proof \rangle$

**lemma** *less-degree-imp*: *n < degree p  $\implies$   $\exists i > n. \text{coeff } p \ i \neq 0$*   
 $\langle proof \rangle$

**lemma** *poly-eqI2*:

**assumes** *degree p = degree q and  $\bigwedge i. i \leq \text{degree } p \implies \text{coeff } p \ i = \text{coeff } q \ i$*   
**shows** *p = q*  
 $\langle proof \rangle$

#### 4.4 The zero polynomial

**instantiation** *poly* :: (*zero*) *zero*  
**begin**

**lift-definition** *zero-poly* :: '*a poly*  
**is**  $\lambda-. 0$   
 $\langle proof \rangle$

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

**end**

**lemma** *coeff-0 [simp]*: *coeff 0 n = 0*  
 $\langle proof \rangle$

**lemma** *degree-0 [simp]*: *degree 0 = 0*  
 $\langle proof \rangle$

**lemma** *leading-coeff-neq-0*:

**assumes** *p  $\neq$  0*  
**shows** *coeff p (degree p)  $\neq$  0*  
 $\langle proof \rangle$

**lemma** *leading-coeff-0-iff* [simp]:  $\text{coeff } p (\text{degree } p) = 0 \iff p = 0$   
 ⟨proof⟩

**lemma** *degree-lessI*:  
 assumes  $p \neq 0 \vee n > 0 \forall k \geq n. \text{coeff } p k = 0$   
 shows  $\text{degree } p < n$   
 ⟨proof⟩

**lemma** *eq-zero-or-degree-less*:  
 assumes  $\text{degree } p \leq n$  and  $\text{coeff } p n = 0$   
 shows  $p = 0 \vee \text{degree } p < n$   
 ⟨proof⟩

**lemma** *coeff-0-degree-minus-1*:  $\text{coeff } rrr \ dr = 0 \implies \text{degree } rrr \leq dr \implies \text{degree } rrr \leq dr - 1$   
 ⟨proof⟩

## 4.5 List-style constructor for polynomials

**lift-definition** *pCons* ::  $'a::\text{zero} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly}$   
 is  $\lambda a \ p. \text{case-nat } a (\text{coeff } p)$   
 ⟨proof⟩

**lemmas** *coeff-pCons = pCons.rep-eq*

**lemma** *coeff-pCons'*:  $\text{poly.coeff } (pCons \ c \ p) \ n = (\text{if } n = 0 \text{ then } c \text{ else } \text{poly.coeff } p \ (n - 1))$   
 ⟨proof⟩

**lemma** *coeff-pCons-0* [simp]:  $\text{coeff } (pCons \ a \ p) \ 0 = a$   
 ⟨proof⟩

**lemma** *coeff-pCons-Suc* [simp]:  $\text{coeff } (pCons \ a \ p) \ (\text{Suc } n) = \text{coeff } p \ n$   
 ⟨proof⟩

**lemma** *degree-pCons-le*:  $\text{degree } (pCons \ a \ p) \leq \text{Suc } (\text{degree } p)$   
 ⟨proof⟩

**lemma** *degree-pCons-eq*:  $p \neq 0 \implies \text{degree } (pCons \ a \ p) = \text{Suc } (\text{degree } p)$   
 ⟨proof⟩

**lemma** *degree-pCons-0*:  $\text{degree } (pCons \ a \ 0) = 0$   
 ⟨proof⟩

**lemma** *degree-pCons-eq-if* [simp]:  $\text{degree } (pCons \ a \ p) = (\text{if } p = 0 \text{ then } 0 \text{ else } \text{Suc } (\text{degree } p))$   
 ⟨proof⟩

**lemma** *pCons-0-0* [simp]:  $pCons \ 0 \ 0 = 0$

$\langle \text{proof} \rangle$

**lemma** *pCons-eq-iff* [*simp*]:  $pCons\ a\ p = pCons\ b\ q \longleftrightarrow a = b \wedge p = q$   
 $\langle \text{proof} \rangle$

**lemma** *pCons-eq-0-iff* [*simp*]:  $pCons\ a\ p = 0 \longleftrightarrow a = 0 \wedge p = 0$   
 $\langle \text{proof} \rangle$

**lemma** *pCons-cases* [*cases type: poly*]:  
**obtains**  $(pCons)\ a\ q$  **where**  $p = pCons\ a\ q$   
 $\langle \text{proof} \rangle$

**lemma** *pCons-induct* [*case-names 0 pCons, induct type: poly*]:  
**assumes** *zero*:  $P\ 0$   
**assumes** *pCons*:  $\bigwedge a\ p.\ a \neq 0 \vee p \neq 0 \implies P\ p \implies P\ (pCons\ a\ p)$   
**shows**  $P\ p$   
 $\langle \text{proof} \rangle$

**lemma** *degree-eq-zeroE*:  
**fixes**  $p :: 'a::zero\ poly$   
**assumes** *degree*  $p = 0$   
**obtains**  $a$  **where**  $p = pCons\ a\ 0$   
 $\langle \text{proof} \rangle$

## 4.6 Quickcheck generator for polynomials

**quickcheck-generator** *poly constructors*:  $0 :: -\ poly, pCons$

## 4.7 List-style syntax for polynomials

**syntax**

$-poly :: args \Rightarrow 'a\ poly$  ( $\langle \langle \text{indent}=2\ \text{notation}=\langle \text{mixfix polynomial enumeration} \rangle \rangle \rangle$ )

**syntax-consts**

$-poly \equiv pCons$

**translations**

$[x, xs:] \equiv CONST\ pCons\ x\ [xs:]$

$[x:] \equiv CONST\ pCons\ x\ 0$

**lemma** *degree-0-id*:

**assumes** *degree*  $p = 0$

**shows**  $[:\text{coeff}\ p\ 0:] = p$

$\langle \text{proof} \rangle$

**lemma** *degree0-coeffs*:  $\text{degree}\ p = 0 \implies \exists\ a.\ p = [:\ a:]$   
 $\langle \text{proof} \rangle$

**lemma** *degree1-coeffs*:

**fixes**  $p :: 'a::zero\ poly$

**assumes** *degree*  $p = 1$

**obtains**  $a\ b$  **where**  $p = [: b, a :]$   $a \neq 0$   
 $\langle proof \rangle$

**lemma** *degree2-coeffs*:  
**fixes**  $p :: 'a::zero\ poly$   
**assumes**  $degree\ p = 2$   
**obtains**  $a\ b\ c$  **where**  $p = [: c, b, a :]$   $a \neq 0$   
 $\langle proof \rangle$

## 4.8 Representation of polynomials by lists of coefficients

**primrec**  $Poly :: 'a::zero\ list \Rightarrow 'a\ poly$   
**where**  
 $[code-post]: Poly\ [] = 0$   
 $| [code-post]: Poly\ (a \# as) = pCons\ a\ (Poly\ as)$

**lemma** *Poly-replicate-0*  $[simp]: Poly\ (replicate\ n\ 0) = 0$   
 $\langle proof \rangle$

**lemma** *Poly-eq-0*:  $Poly\ as = 0 \longleftrightarrow (\exists n. as = replicate\ n\ 0)$   
 $\langle proof \rangle$

**lemma** *Poly-append-replicate-zero*  $[simp]: Poly\ (as @ replicate\ n\ 0) = Poly\ as$   
 $\langle proof \rangle$

**lemma** *Poly-snoc-zero*  $[simp]: Poly\ (as @ [0]) = Poly\ as$   
 $\langle proof \rangle$

**lemma** *Poly-cCons-eq-pCons-Poly*  $[simp]: Poly\ (a \#\# p) = pCons\ a\ (Poly\ p)$   
 $\langle proof \rangle$

**lemma** *Poly-on-rev-starting-with-0*  $[simp]: hd\ as = 0 \implies Poly\ (rev\ (tl\ as)) = Poly\ (rev\ as)$   
 $\langle proof \rangle$

**lemma** *degree-Poly*:  $degree\ (Poly\ xs) \leq length\ xs$   
 $\langle proof \rangle$

**lemma** *coeff-Poly-eq*  $[simp]: coeff\ (Poly\ xs) = nth-default\ 0\ xs$   
 $\langle proof \rangle$

**definition**  $coeffs :: 'a\ poly \Rightarrow 'a::zero\ list$   
**where**  $coeffs\ p = (if\ p = 0\ then\ []\ else\ map\ (\lambda i. coeff\ p\ i)\ [0 ..< Suc\ (degree\ p)])$

**lemma** *coeffs-eq-Nil*  $[simp]: coeffs\ p = [] \longleftrightarrow p = 0$   
 $\langle proof \rangle$

**lemma** *not-0-coeffs-not-Nil*:  $p \neq 0 \implies coeffs\ p \neq []$   
 $\langle proof \rangle$

**lemma** *coeffs-0-eq-Nil* [simp]:  $\text{coeffs } 0 = []$   
 ⟨proof⟩

**lemma** *coeffs-pCons-eq-cCons* [simp]:  $\text{coeffs } (\text{pCons } a \ p) = a \ \#\# \ \text{coeffs } p$   
 ⟨proof⟩

**lemma** *length-coeffs*:  $p \neq 0 \implies \text{length } (\text{coeffs } p) = \text{degree } p + 1$   
 ⟨proof⟩

**lemma** *coeffs-nth*:  $p \neq 0 \implies n \leq \text{degree } p \implies \text{coeffs } p \ ! \ n = \text{coeff } p \ n$   
 ⟨proof⟩

**lemma** *coeff-in-coeffs*:  $p \neq 0 \implies n \leq \text{degree } p \implies \text{coeff } p \ n \in \text{set } (\text{coeffs } p)$   
 ⟨proof⟩

**lemma** *not-0-cCons-eq* [simp]:  $p \neq 0 \implies a \ \#\# \ \text{coeffs } p = a \ \# \ \text{coeffs } p$   
 ⟨proof⟩

**lemma** *Poly-coeffs* [simp, code abstype]:  $\text{Poly } (\text{coeffs } p) = p$   
 ⟨proof⟩

**lemma** *coeffs-Poly* [simp]:  $\text{coeffs } (\text{Poly } as) = \text{strip-while } (\text{HOL.eq } 0) \ as$   
 ⟨proof⟩

**lemma** *no-trailing-coeffs* [simp]:  
 $\text{no-trailing } (\text{HOL.eq } 0) \ (\text{coeffs } p)$   
 ⟨proof⟩

**lemma** *strip-while-coeffs* [simp]:  
 $\text{strip-while } (\text{HOL.eq } 0) \ (\text{coeffs } p) = \text{coeffs } p$   
 ⟨proof⟩

**lemma** *coeffs-eq-iff*:  $p = q \longleftrightarrow \text{coeffs } p = \text{coeffs } q$   
 (is  $?P \longleftrightarrow ?Q$ )  
 ⟨proof⟩

**lemma** *nth-default-coeffs-eq*:  $\text{nth-default } 0 \ (\text{coeffs } p) = \text{coeff } p$   
 ⟨proof⟩

**lemma** *range-coeff*:  $\text{range } (\text{coeff } p) = \text{insert } 0 \ (\text{set } (\text{coeffs } p))$   
 ⟨proof⟩

**lemma** [code]:  $\text{coeff } p = \text{nth-default } 0 \ (\text{coeffs } p)$   
 ⟨proof⟩

**lemma** *coeffs-eqI*:  
 assumes *coeff*:  $\bigwedge n. \text{coeff } p \ n = \text{nth-default } 0 \ xs \ n$   
 assumes *zero*:  $\text{no-trailing } (\text{HOL.eq } 0) \ xs$

**shows**  $\text{coeffs } p = xs$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{degree-eq-length-coeffs}$  [code]:  $\text{degree } p = \text{length } (\text{coeffs } p) - 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{length-coeffs-degree}$ :  $p \neq 0 \implies \text{length } (\text{coeffs } p) = \text{Suc } (\text{degree } p)$   
 $\langle \text{proof} \rangle$

**lemma** [code abstract]:  $\text{coeffs } 0 = []$   
 $\langle \text{proof} \rangle$

**lemma** [code abstract]:  $\text{coeffs } (p\text{Cons } a \ p) = a \ \#\# \ \text{coeffs } p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{set-coeffs-subset-singleton-0-iff}$  [simp]:  
 $\text{set } (\text{coeffs } p) \subseteq \{0\} \longleftrightarrow p = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{set-coeffs-not-only-0}$  [simp]:  
 $\text{set } (\text{coeffs } p) \neq \{0\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{forall-coeffs-conv}$ :  
 $(\forall n. \ P \ (\text{coeff } p \ n)) \longleftrightarrow (\forall c \in \text{set } (\text{coeffs } p). \ P \ c) \ \text{if } P \ 0$   
 $\langle \text{proof} \rangle$

**instantiation**  $\text{poly} :: (\{zero, equal\}) \ \text{equal}$   
**begin**

**definition** [code]:  $\text{HOL.equal } (p :: 'a \ \text{poly}) \ q \longleftrightarrow \text{HOL.equal } (\text{coeffs } p) \ (\text{coeffs } q)$

**instance**  
 $\langle \text{proof} \rangle$

**end**

**lemma** [code nbe]:  $\text{HOL.equal } (p :: - \ \text{poly}) \ p \longleftrightarrow \text{True}$   
 $\langle \text{proof} \rangle$

**definition**  $\text{is-zero} :: 'a :: \text{zero } \text{poly} \Rightarrow \text{bool}$   
**where** [code]:  $\text{is-zero } p \longleftrightarrow \text{List.null } (\text{coeffs } p)$

**lemma**  $\text{is-zero-null}$  [code-abbrev]:  $\text{is-zero } p \longleftrightarrow p = 0$   
 $\langle \text{proof} \rangle$

Reconstructing the polynomial from the list

**definition**  $\text{poly-of-list} :: 'a :: \text{comm-monoid-add } \text{list} \Rightarrow 'a \ \text{poly}$   
**where** [simp]:  $\text{poly-of-list} = \text{Poly}$

**lemma** *poly-of-list-impl* [code abstract]:  $\text{coeffs } (\text{poly-of-list } as) = \text{strip-while } (HOL.eq\ 0) \text{ as}$   
 $\langle \text{proof} \rangle$

## 4.9 Fold combinator for polynomials

**definition** *fold-coeffs* ::  $('a::zero \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'a \text{ poly} \Rightarrow 'b \Rightarrow 'b$   
**where**  $\text{fold-coeffs } f\ p = \text{foldr } f\ (\text{coeffs } p)$

**lemma** *fold-coeffs-0-eq* [simp]:  $\text{fold-coeffs } f\ 0 = id$   
 $\langle \text{proof} \rangle$

**lemma** *fold-coeffs-pCons-eq* [simp]:  $f\ 0 = id \implies \text{fold-coeffs } f\ (pCons\ a\ p) = f\ a \circ \text{fold-coeffs } f\ p$   
 $\langle \text{proof} \rangle$

**lemma** *fold-coeffs-pCons-0-0-eq* [simp]:  $\text{fold-coeffs } f\ (pCons\ 0\ 0) = id$   
 $\langle \text{proof} \rangle$

**lemma** *fold-coeffs-pCons-coeff-not-0-eq* [simp]:  
 $a \neq 0 \implies \text{fold-coeffs } f\ (pCons\ a\ p) = f\ a \circ \text{fold-coeffs } f\ p$   
 $\langle \text{proof} \rangle$

**lemma** *fold-coeffs-pCons-not-0-0-eq* [simp]:  
 $p \neq 0 \implies \text{fold-coeffs } f\ (pCons\ a\ p) = f\ a \circ \text{fold-coeffs } f\ p$   
 $\langle \text{proof} \rangle$

## 4.10 Canonical morphism on polynomials – evaluation

**definition** *poly* ::  $\langle 'a::comm-semiring-0\ \text{poly} \Rightarrow 'a \Rightarrow 'a \rangle$   
**where**  $\langle \text{poly } p\ a = \text{horner-sum } id\ a\ (\text{coeffs } p) \rangle$

**lemma** *poly-eq-fold-coeffs*:  
 $\langle \text{poly } p = \text{fold-coeffs } (\lambda a\ f\ x. a + x * f\ x)\ p\ (\lambda x. 0) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-0* [simp]:  $\text{poly } 0\ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *poly-pCons* [simp]:  $\text{poly } (pCons\ a\ p)\ x = a + x * \text{poly } p\ x$   
 $\langle \text{proof} \rangle$

**lemma** *poly-altdef*:  $\text{poly } p\ x = (\sum i \leq \text{degree } p. \text{coeff } p\ i * x^i)$   
**for**  $x :: 'a::\{comm-semiring-0, semiring-1\}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-0-coeff-0*:  $\text{poly } p\ 0 = \text{coeff } p\ 0$   
 $\langle \text{proof} \rangle$

**lemma** *poly-zero*:  
**fixes**  $p :: 'a :: \text{comm-ring-1 poly}$   
**assumes**  $x: \text{poly } p \ x = 0$  **shows**  $p = 0 \longleftrightarrow \text{degree } p = 0$   
 $\langle \text{proof} \rangle$

#### 4.11 Monomials

**lift-definition** *monom*  $:: 'a \Rightarrow \text{nat} \Rightarrow 'a::\text{zero poly}$   
**is**  $\lambda a \ m \ n. \text{if } m = n \text{ then } a \text{ else } 0$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-monom* [simp]:  $\text{coeff } (\text{monom } a \ m) \ n = (\text{if } m = n \text{ then } a \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *monom-0*:  $\text{monom } a \ 0 = [:a:]$   
 $\langle \text{proof} \rangle$

**lemma** *monom-Suc*:  $\text{monom } a \ (\text{Suc } n) = p\text{Cons } 0 \ (\text{monom } a \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *monom-eq-0* [simp]:  $\text{monom } 0 \ n = 0$   
 $\langle \text{proof} \rangle$

**lemma** *monom-eq-0-iff* [simp]:  $\text{monom } a \ n = 0 \longleftrightarrow a = 0$   
 $\langle \text{proof} \rangle$

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

**lemma** *degree-monom-le*:  $\text{degree } (\text{monom } a \ n) \leq n$   
 $\langle \text{proof} \rangle$

**lemma** *degree-monom-eq*:  $a \neq 0 \implies \text{degree } (\text{monom } a \ n) = n$   
 $\langle \text{proof} \rangle$

**lemma** *coeffs-monom* [code abstract]:  
 $\text{coeffs } (\text{monom } a \ n) = (\text{if } a = 0 \text{ then } [] \text{ else replicate } n \ 0 \ @ [a])$   
 $\langle \text{proof} \rangle$

**lemma** *fold-coeffs-monom* [simp]:  $a \neq 0 \implies \text{fold-coeffs } f \ (\text{monom } a \ n) = f \ 0 \ \sim n \circ f \ a$   
 $\langle \text{proof} \rangle$

**lemma** *poly-monom*:  $\text{poly } (\text{monom } a \ n) \ x = a * x \wedge n$   
**for**  $a \ x :: 'a::\text{comm-semiring-1}$   
 $\langle \text{proof} \rangle$

**lemma** *monom-eq-iff'*:  $\text{monom } c \ n = \text{monom } d \ m \longleftrightarrow c = d \wedge (c = 0 \vee n = m)$



$\langle proof \rangle$

**lemma** *monom-eq-const-iff*:  $monom\ c\ n = [:d:] \longleftrightarrow c = d \wedge (c = 0 \vee n = 0)$   
 $\langle proof \rangle$

## 4.12 Leading coefficient

**abbreviation** *lead-coeff*:: 'a::zero poly  $\Rightarrow$  'a  
**where** *lead-coeff*  $p \equiv coeff\ p\ (degree\ p)$

**lemma** *lead-coeff-pCons[simp]*:  
 $p \neq 0 \implies lead-coeff\ (pCons\ a\ p) = lead-coeff\ p$   
 $p = 0 \implies lead-coeff\ (pCons\ a\ p) = a$   
 $\langle proof \rangle$

**lemma** *lead-coeff-monom [simp]*:  $lead-coeff\ (monom\ c\ n) = c$   
 $\langle proof \rangle$

**lemma** *last-coeffs-eq-coeff-degree*:  
 $last\ (coeffs\ p) = lead-coeff\ p$  **if**  $p \neq 0$   
 $\langle proof \rangle$

**lemma** *lead-coeff-list-def*:  
 $lead-coeff\ p = (if\ coeffs\ p = []\ then\ 0\ else\ last\ (coeffs\ p))$   
 $\langle proof \rangle$

## 4.13 Addition and subtraction

**instantiation** *poly* :: (comm-monoid-add) comm-monoid-add  
**begin**

**lift-definition** *plus-poly* :: 'a poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly  
**is**  $\lambda p\ q\ n. coeff\ p\ n + coeff\ q\ n$   
 $\langle proof \rangle$

**lemma** *coeff-add [simp]*:  $coeff\ (p + q)\ n = coeff\ p\ n + coeff\ q\ n$   
 $\langle proof \rangle$

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

**end**

**instantiation** *poly* :: (cancel-comm-monoid-add) cancel-comm-monoid-add  
**begin**

**lift-definition** *minus-poly* :: 'a poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly  
**is**  $\lambda p\ q\ n. coeff\ p\ n - coeff\ q\ n$   
 $\langle proof \rangle$

**lemma** *coeff-diff* [simp]:  $\text{coeff } (p - q) \ n = \text{coeff } p \ n - \text{coeff } q \ n$   
 ⟨proof⟩

**instance**  
 ⟨proof⟩

**end**

**instantiation** *poly* :: (ab-group-add) ab-group-add  
**begin**

**lift-definition** *uminus-poly* :: 'a poly  $\Rightarrow$  'a poly  
**is**  $\lambda p \ n. - \text{coeff } p \ n$   
 ⟨proof⟩

**lemma** *coeff-minus* [simp]:  $\text{coeff } (- p) \ n = - \text{coeff } p \ n$   
 ⟨proof⟩

**instance**  
 ⟨proof⟩

**end**

**lemma** *add-pCons* [simp]:  $p\text{Cons } a \ p + p\text{Cons } b \ q = p\text{Cons } (a + b) \ (p + q)$   
 ⟨proof⟩

**lemma** *minus-pCons* [simp]:  $- p\text{Cons } a \ p = p\text{Cons } (- a) \ (- p)$   
 ⟨proof⟩

**lemma** *diff-pCons* [simp]:  $p\text{Cons } a \ p - p\text{Cons } b \ q = p\text{Cons } (a - b) \ (p - q)$   
 ⟨proof⟩

**lemma** *degree-add-le-max*:  $\text{degree } (p + q) \leq \max (\text{degree } p) (\text{degree } q)$   
 ⟨proof⟩

**lemma** *degree-add-le*:  $\text{degree } p \leq n \implies \text{degree } q \leq n \implies \text{degree } (p + q) \leq n$   
 ⟨proof⟩

**lemma** *degree-add-less*:  $\text{degree } p < n \implies \text{degree } q < n \implies \text{degree } (p + q) < n$   
 ⟨proof⟩

**lemma** *degree-add-eq-right*: **assumes**  $\text{degree } p < \text{degree } q$  **shows**  $\text{degree } (p + q) = \text{degree } q$   
 ⟨proof⟩

**lemma** *degree-add-eq-left*:  $\text{degree } q < \text{degree } p \implies \text{degree } (p + q) = \text{degree } p$   
 ⟨proof⟩

**lemma** *degree-minus* [simp]:  $\text{degree } (- p) = \text{degree } p$

$\langle \text{proof} \rangle$

**lemma** *lead-coeff-add-le*:  $\text{degree } p < \text{degree } q \implies \text{lead-coeff } (p + q) = \text{lead-coeff } q$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-minus*:  $\text{lead-coeff } (-p) = - \text{lead-coeff } p$   
 $\langle \text{proof} \rangle$

**lemma** *degree-diff-le-max*:  $\text{degree } (p - q) \leq \max (\text{degree } p) (\text{degree } q)$   
**for**  $p \ q :: 'a::\text{ab-group-add poly}$   
 $\langle \text{proof} \rangle$

**lemma** *degree-diff-le*:  $\text{degree } p \leq n \implies \text{degree } q \leq n \implies \text{degree } (p - q) \leq n$   
**for**  $p \ q :: 'a::\text{ab-group-add poly}$   
 $\langle \text{proof} \rangle$

**lemma** *degree-diff-less*:  $\text{degree } p < n \implies \text{degree } q < n \implies \text{degree } (p - q) < n$   
**for**  $p \ q :: 'a::\text{ab-group-add poly}$   
 $\langle \text{proof} \rangle$

**lemma** *add-monom*:  $\text{monom } a \ n + \text{monom } b \ n = \text{monom } (a + b) \ n$   
 $\langle \text{proof} \rangle$

**lemma** *diff-monom*:  $\text{monom } a \ n - \text{monom } b \ n = \text{monom } (a - b) \ n$   
 $\langle \text{proof} \rangle$

**lemma** *minus-monom*:  $- \text{monom } a \ n = \text{monom } (-a) \ n$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-sum*:  $\text{coeff } (\sum x \in A. p \ x) \ i = (\sum x \in A. \text{coeff } (p \ x) \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *monom-sum*:  $\text{monom } (\sum x \in A. a \ x) \ n = (\sum x \in A. \text{monom } (a \ x) \ n)$   
 $\langle \text{proof} \rangle$

**fun** *plus-coeffs* ::  $'a::\text{comm-monoid-add list} \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{list}$   
**where**

$\text{plus-coeffs } xs \ [] = xs$   
 $| \text{plus-coeffs } [] \ ys = ys$   
 $| \text{plus-coeffs } (x \ \# \ xs) \ (y \ \# \ ys) = (x + y) \ \#\# \ \text{plus-coeffs } xs \ ys$

**lemma** *coeffs-plus-eq-plus-coeffs* [code abstract]:  
 $\text{coeffs } (p + q) = \text{plus-coeffs } (\text{coeffs } p) \ (\text{coeffs } q)$   
 $\langle \text{proof} \rangle$

**lemma** *coeffs-uminus* [code abstract]:  
 $\text{coeffs } (-p) = \text{map } \text{uminus } (\text{coeffs } p)$   
 $\langle \text{proof} \rangle$

**lemma** *[code]*:  $p - q = p + - q$   
**for**  $p\ q :: 'a::ab-group-add\ poly$   
 $\langle proof \rangle$

**lemma** *poly-add [simp]*:  $poly\ (p + q)\ x = poly\ p\ x + poly\ q\ x$   
 $\langle proof \rangle$

**lemma** *poly-minus [simp]*:  $poly\ (-\ p)\ x = -\ poly\ p\ x$   
**for**  $x :: 'a::comm-ring$   
 $\langle proof \rangle$

**lemma** *poly-diff [simp]*:  $poly\ (p - q)\ x = poly\ p\ x - poly\ q\ x$   
**for**  $x :: 'a::comm-ring$   
 $\langle proof \rangle$

**lemma** *poly-sum*:  $poly\ (\sum_{k \in A} p\ k)\ x = (\sum_{k \in A} poly\ (p\ k)\ x)$   
 $\langle proof \rangle$

**lemma** *poly-sum-list*:  $poly\ (\sum_{p \leftarrow ps} p)\ y = (\sum_{p \leftarrow ps} poly\ p\ y)$   
 $\langle proof \rangle$

**lemma** *poly-sum-mset*:  $poly\ (\sum_{x \in \#A} p\ x)\ y = (\sum_{x \in \#A} poly\ (p\ x)\ y)$   
 $\langle proof \rangle$

**lemma** *degree-sum-le*:  $finite\ S \implies (\bigwedge p. p \in S \implies degree\ (f\ p) \leq n) \implies degree\ (sum\ f\ S) \leq n$   
 $\langle proof \rangle$

**lemma** *degree-sum-less*:  
**assumes**  $\bigwedge x. x \in A \implies degree\ (f\ x) < n\ n > 0$   
**shows**  $degree\ (sum\ f\ A) < n$   
 $\langle proof \rangle$

**lemma** *poly-as-sum-of-monoms'*:  
**assumes**  $degree\ p \leq n$   
**shows**  $(\sum_{i \leq n} monom\ (coeff\ p\ i)\ i) = p$   
 $\langle proof \rangle$

**lemma** *poly-as-sum-of-monoms*:  $(\sum_{i \leq degree\ p} monom\ (coeff\ p\ i)\ i) = p$   
 $\langle proof \rangle$

**lemma** *Poly-snoc*:  $Poly\ (xs\ @\ [x]) = Poly\ xs + monom\ x\ (length\ xs)$   
 $\langle proof \rangle$

#### 4.14 Multiplication by a constant, polynomial multiplication and the unit polynomial

**lift-definition** *smult* ::  $'a::comm-semiring-0 \Rightarrow 'a\ poly \Rightarrow 'a\ poly$   
**is**  $\lambda a\ p\ n. a * coeff\ p\ n$

$\langle proof \rangle$

**lemma** *coeff-smult [simp]: coeff (smult a p) n = a \* coeff p n*  
 $\langle proof \rangle$

**lemma** *degree-smult-le: degree (smult a p)  $\leq$  degree p*  
 $\langle proof \rangle$

**lemma** *smult-smult [simp]: smult a (smult b p) = smult (a \* b) p*  
 $\langle proof \rangle$

**lemma** *smult-0-right [simp]: smult a 0 = 0*  
 $\langle proof \rangle$

**lemma** *smult-0-left [simp]: smult 0 p = 0*  
 $\langle proof \rangle$

**lemma** *smult-1-left [simp]: smult (1::'a::comm-semiring-1) p = p*  
 $\langle proof \rangle$

**lemma** *smult-add-right: smult a (p + q) = smult a p + smult a q*  
 $\langle proof \rangle$

**lemma** *smult-add-left: smult (a + b) p = smult a p + smult b p*  
 $\langle proof \rangle$

**lemma** *smult-minus-right [simp]: smult a (- p) = - smult a p*  
**for** *a :: 'a::comm-ring*  
 $\langle proof \rangle$

**lemma** *smult-minus-left [simp]: smult (- a) p = - smult a p*  
**for** *a :: 'a::comm-ring*  
 $\langle proof \rangle$

**lemma** *smult-diff-right: smult a (p - q) = smult a p - smult a q*  
**for** *a :: 'a::comm-ring*  
 $\langle proof \rangle$

**lemma** *smult-diff-left: smult (a - b) p = smult a p - smult b p*  
**for** *a b :: 'a::comm-ring*  
 $\langle proof \rangle$

**lemmas** *smult-distrib =*  
*smult-add-left smult-add-right*  
*smult-diff-left smult-diff-right*

**lemma** *smult-pCons [simp]: smult a (pCons b p) = pCons (a \* b) (smult a p)*  
 $\langle proof \rangle$

**lemma** *smult-monom*:  $smult\ a\ (monom\ b\ n) = monom\ (a * b)\ n$   
 ⟨proof⟩

**lemma** *smult-Poly*:  $smult\ c\ (Poly\ xs) = Poly\ (map\ ((*)\ c)\ xs)$   
 ⟨proof⟩

**lemma** *degree-smult-eq* [simp]:  $degree\ (smult\ a\ p) = (if\ a = 0\ then\ 0\ else\ degree\ p)$   
 for  $a :: 'a :: \{comm-semiring-0, semiring-no-zero-divisors\}$   
 ⟨proof⟩

**lemma** *smult-eq-0-iff* [simp]:  $smult\ a\ p = 0 \longleftrightarrow a = 0 \vee p = 0$   
 for  $a :: 'a :: \{comm-semiring-0, semiring-no-zero-divisors\}$   
 ⟨proof⟩

**lemma** *coeffs-smult* [code abstract]:  
 $coeffs\ (smult\ a\ p) = (if\ a = 0\ then\ []\ else\ map\ (Groups.times\ a)\ (coeffs\ p))$   
 for  $p :: 'a :: \{comm-semiring-0, semiring-no-zero-divisors\}$  poly  
 ⟨proof⟩

**lemma** *smult-eq-iff*:  
 fixes  $b :: 'a :: field$   
 assumes  $b \neq 0$   
 shows  $smult\ a\ p = smult\ b\ q \longleftrightarrow smult\ (a / b)\ p = q$   
 (is  $?lhs \longleftrightarrow ?rhs$ )  
 ⟨proof⟩

**lemma** *smult-cancel*:  
 fixes  $p :: 'a :: idom\ poly$   
 assumes  $c \neq 0$  and  $smult$ :  $smult\ c\ p = smult\ c\ q$   
 shows  $p = q$   
 ⟨proof⟩

**instantiation**  $poly :: (comm-semiring-0)\ comm-semiring-0$   
**begin**

**definition**  $p * q = fold-coeffs\ (\lambda a\ p.\ smult\ a\ q + pCons\ 0\ p)\ p\ 0$

**lemma** *mult-poly-0-left*:  $(0 :: 'a\ poly) * q = 0$   
 ⟨proof⟩

**lemma** *mult-pCons-left* [simp]:  $pCons\ a\ p * q = smult\ a\ q + pCons\ 0\ (p * q)$   
 ⟨proof⟩

**lemma** *mult-poly-0-right*:  $p * (0 :: 'a\ poly) = 0$   
 ⟨proof⟩

**lemma** *mult-pCons-right* [simp]:  $p * pCons\ a\ q = smult\ a\ p + pCons\ 0\ (p * q)$   
 ⟨proof⟩

**lemmas** *mult-poly-0 = mult-poly-0-left mult-poly-0-right*

**lemma** *mult-smult-left [simp]: smult a p \* q = smult a (p \* q)*  
 ⟨proof⟩

**lemma** *mult-smult-right [simp]: p \* smult a q = smult a (p \* q)*  
 ⟨proof⟩

**lemma** *mult-poly-add-left: (p + q) \* r = p \* r + q \* r*  
**for** *p q r :: 'a poly*  
 ⟨proof⟩

**instance**  
 ⟨proof⟩

**end**

**lemma** *coeff-mult-degree-sum:*  
*coeff (p \* q) (degree p + degree q) = coeff p (degree p) \* coeff q (degree q)*  
 ⟨proof⟩

**instance** *poly :: ({comm-semiring-0, semiring-no-zero-divisors}) semiring-no-zero-divisors*  
 ⟨proof⟩

**instance** *poly :: (comm-semiring-0-cancel) comm-semiring-0-cancel* ⟨proof⟩

**lemma** *coeff-mult: coeff (p \* q) n = (∑ i ≤ n. coeff p i \* coeff q (n - i))*  
 ⟨proof⟩

**lemma** *coeff-mult-0: coeff (p \* q) 0 = coeff p 0 \* coeff q 0*  
 ⟨proof⟩

**lemma** *degree-mult-le: degree (p \* q) ≤ degree p + degree q*  
 ⟨proof⟩

**lemma** *mult-monom: monom a m \* monom b n = monom (a \* b) (m + n)*  
 ⟨proof⟩

**instantiation** *poly :: (comm-semiring-1) comm-semiring-1*  
**begin**

**lift-definition** *one-poly :: 'a poly*  
**is** *λn. of-bool (n = 0)*  
 ⟨proof⟩

**lemma** *coeff-1 [simp]:*  
*coeff 1 n = of-bool (n = 0)*  
 ⟨proof⟩

**lemma** *one-pCons*:

$1 = [:1:]$

$\langle proof \rangle$

**lemma** *pCons-one*:

$[:1:] = 1$

$\langle proof \rangle$

**instance**

$\langle proof \rangle$

**end**

**lemma** *poly-1* [*simp*]:

$poly\ 1\ x = 1$

$\langle proof \rangle$

**lemma** *one-poly-eq-simps* [*simp*]:

$1 = [:1:] \longleftrightarrow True$

$[:1:] = 1 \longleftrightarrow True$

$\langle proof \rangle$

**lemma** *degree-1* [*simp*]:

$degree\ 1 = 0$

$\langle proof \rangle$

**lemma** *coeffs-1-eq* [*simp*, *code abstract*]:

$coeffs\ 1 = [1]$

$\langle proof \rangle$

**lemma** *smult-one* [*simp*]:

$smult\ c\ 1 = [:c:]$

$\langle proof \rangle$

**lemma** *smult-sum*:  $smult\ (\sum i \in S. f\ i)\ p = (\sum i \in S. smult\ (f\ i)\ p)$

$\langle proof \rangle$

**lemma** *smult-power*:  $(smult\ a\ p) \wedge^n = smult\ (a \wedge^n)\ (p \wedge^n)$

$\langle proof \rangle$

**lemma** *monom-eq-1* [*simp*]:

$monom\ 1\ 0 = 1$

$\langle proof \rangle$

**lemma** *monom-eq-1-iff*:

$monom\ c\ n = 1 \longleftrightarrow c = 1 \wedge n = 0$

$\langle proof \rangle$

**lemma** *monom-altdef*:



$\text{monom } c \ n = \text{smult } c \ ([:0, 1:] \wedge n)$   
 $\langle \text{proof} \rangle$

**lemma** *degree-sum-list-le*:  $(\bigwedge p . p \in \text{set } ps \implies \text{degree } p \leq n)$   
 $\implies \text{degree } (\text{sum-list } ps) \leq n$   
 $\langle \text{proof} \rangle$

**lemma** *degree-prod-list-le*:  $\text{degree } (\text{prod-list } ps) \leq \text{sum-list } (\text{map } \text{degree } ps)$   
 $\langle \text{proof} \rangle$

**instance** *poly* ::  $(\{\text{comm-semiring-1}, \text{semiring-1-no-zero-divisors}\}) \text{ semiring-1-no-zero-divisors}$   
 $\langle \text{proof} \rangle$

**instance** *poly* ::  $(\text{comm-ring}) \text{ comm-ring}$   $\langle \text{proof} \rangle$   
**instance** *poly* ::  $(\text{comm-ring-1}) \text{ comm-ring-1}$   $\langle \text{proof} \rangle$   
**instance** *poly* ::  $(\text{comm-ring-1}) \text{ comm-semiring-1-cancel}$   $\langle \text{proof} \rangle$

**lemma** *prod-smult*:  $(\prod_{x \in A}. \text{smult } (c \ x) \ (p \ x)) = \text{smult } (\text{prod } c \ A) \ (\text{prod } p \ A)$   
 $\langle \text{proof} \rangle$

**lemma** *degree-power-le*:  $\text{degree } (p \wedge n) \leq \text{degree } p * n$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-0-power*:  $\text{coeff } (p \wedge n) \ 0 = \text{coeff } p \ 0 \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *poly-smult [simp]*:  $\text{poly } (\text{smult } a \ p) \ x = a * \text{poly } p \ x$   
 $\langle \text{proof} \rangle$

**lemma** *poly-mult [simp]*:  $\text{poly } (p * q) \ x = \text{poly } p \ x * \text{poly } q \ x$   
 $\langle \text{proof} \rangle$

**lemma** *poly-power [simp]*:  $\text{poly } (p \wedge n) \ x = \text{poly } p \ x \wedge n$   
**for**  $p :: 'a :: \text{comm-semiring-1}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *poly-prod*:  $\text{poly } (\prod_{k \in A}. p \ k) \ x = (\prod_{k \in A}. \text{poly } (p \ k) \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-prod-list*:  $\text{poly } (\prod_{p \leftarrow ps}. p) \ y = (\prod_{p \leftarrow ps}. \text{poly } p \ y)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-prod-mset*:  $\text{poly } (\prod_{x \in \#A}. p \ x) \ y = (\prod_{x \in \#A}. \text{poly } (p \ x) \ y)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-const-pow*:  $[ : c : ] \wedge n = [ : c \wedge n : ]$   
 $\langle \text{proof} \rangle$

**lemma** *monom-power*:  $\text{monom } c \ n \wedge k = \text{monom } (c \wedge k) \ (n * k)$   
 $\langle \text{proof} \rangle$

**lemma** *degree-prod-sum-le*:  $\text{finite } S \implies \text{degree } (\text{prod } f \ S) \leq \text{sum } (\text{degree } \circ f) \ S$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-0-prod-list*:  $\text{coeff } (\text{prod-list } xs) \ 0 = \text{prod-list } (\text{map } (\lambda p. \text{coeff } p \ 0) \ xs)$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-monom-mult*:  $\text{coeff } (\text{monom } c \ n * p) \ k = (\text{if } k < n \text{ then } 0 \text{ else } c * \text{coeff } p \ (k - n))$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-monom-Suc*:  $\text{coeff } (\text{monom } a \ (\text{Suc } d) * p) \ (\text{Suc } i) = \text{coeff } (\text{monom } a \ d * p) \ i$   
 $\langle \text{proof} \rangle$

**lemma** *monom-1-dvd-iff'*:  $\text{monom } 1 \ n \ \text{dvd } p \longleftrightarrow (\forall k < n. \text{coeff } p \ k = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-sum-monom*:  
**assumes**  $n: n \leq d$   
**shows**  $\text{coeff } (\sum i \leq d. \text{monom } (f \ i) \ i) \ n = f \ n$  (**is** ?l = -)  
 $\langle \text{proof} \rangle$

## 4.15 Mapping polynomials

**definition** *map-poly* ::  $('a :: \text{zero} \Rightarrow 'b :: \text{zero}) \Rightarrow 'a \ \text{poly} \Rightarrow 'b \ \text{poly}$   
**where**  $\text{map-poly } f \ p = \text{Poly } (\text{map } f \ (\text{coeffs } p))$

**lemma** *map-poly-0* [*simp*]:  $\text{map-poly } f \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *map-poly-1*:  $\text{map-poly } f \ 1 = [:f \ 1:]$   
 $\langle \text{proof} \rangle$

**lemma** *map-poly-1'* [*simp*]:  $f \ 1 = 1 \implies \text{map-poly } f \ 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-map-poly*:  
**assumes**  $f \ 0 = 0$   
**shows**  $\text{coeff } (\text{map-poly } f \ p) \ n = f \ (\text{coeff } p \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-map-poly-nz*:  
**assumes**  $f \ (\text{lead-coeff } p) \neq 0 \ f \ 0 = 0$   
**shows**  $\text{lead-coeff } (\text{map-poly } f \ p) = f \ (\text{lead-coeff } p)$   
 $\langle \text{proof} \rangle$

**lemma** *coeffs-map-poly* [*code abstract*]:  
 $\text{coeffs } (\text{map-poly } f \ p) = \text{strip-while } ((=) \ 0) \ (\text{map } f \ (\text{coeffs } p))$

$\langle proof \rangle$

**lemma** *coeffs-map-poly'*:

**assumes**  $\bigwedge x. x \neq 0 \implies f\ x \neq 0$

**shows**  $coeffs\ (map\text{-}poly\ f\ p) = map\ f\ (coeffs\ p)$

$\langle proof \rangle$

**lemma** *set-coeffs-map-poly*:

$(\bigwedge x. f\ x = 0 \longleftrightarrow x = 0) \implies set\ (coeffs\ (map\text{-}poly\ f\ p)) = f\ `set\ (coeffs\ p)$

$\langle proof \rangle$

**lemma** *degree-map-poly*:

**assumes**  $\bigwedge x. x \neq 0 \implies f\ x \neq 0$

**shows**  $degree\ (map\text{-}poly\ f\ p) = degree\ p$

$\langle proof \rangle$

**lemma** *map-poly-eq-0-iff*:

**assumes**  $f\ 0 = 0 \wedge \bigwedge x. x \in set\ (coeffs\ p) \implies x \neq 0 \implies f\ x \neq 0$

**shows**  $map\text{-}poly\ f\ p = 0 \longleftrightarrow p = 0$

$\langle proof \rangle$

**lemma** *map-poly-smult*:

**assumes**  $f\ 0 = 0 \wedge c\ x. f\ (c * x) = f\ c * f\ x$

**shows**  $map\text{-}poly\ f\ (smult\ c\ p) = smult\ (f\ c)\ (map\text{-}poly\ f\ p)$

$\langle proof \rangle$

**lemma** *map-poly-pCons*:

**assumes**  $f\ 0 = 0$

**shows**  $map\text{-}poly\ f\ (pCons\ c\ p) = pCons\ (f\ c)\ (map\text{-}poly\ f\ p)$

$\langle proof \rangle$

**lemma** *map-poly-map-poly*:

**assumes**  $f\ 0 = 0 \wedge g\ 0 = 0$

**shows**  $map\text{-}poly\ f\ (map\text{-}poly\ g\ p) = map\text{-}poly\ (f \circ g)\ p$

$\langle proof \rangle$

**lemma** *map-poly-id [simp]*:  $map\text{-}poly\ id\ p = p$

$\langle proof \rangle$

**lemma** *map-poly-id' [simp]*:  $map\text{-}poly\ (\lambda x. x)\ p = p$

$\langle proof \rangle$

**lemma** *map-poly-cong*:

**assumes**  $(\bigwedge x. x \in set\ (coeffs\ p) \implies f\ x = g\ x)$

**shows**  $map\text{-}poly\ f\ p = map\text{-}poly\ g\ p$

$\langle proof \rangle$

**lemma** *map-poly-monom*:  $f\ 0 = 0 \implies map\text{-}poly\ f\ (monom\ c\ n) = monom\ (f\ c)\ n$

$\langle proof \rangle$

**lemma** *map-poly-idI*:

**assumes**  $\bigwedge x. x \in \text{set } (\text{coeffs } p) \implies f\ x = x$

**shows**  $\text{map-poly } f\ p = p$

*<proof>*

**lemma** *map-poly-idI'*:

**assumes**  $\bigwedge x. x \in \text{set } (\text{coeffs } p) \implies f\ x = x$

**shows**  $p = \text{map-poly } f\ p$

*<proof>*

**lemma** *smult-conv-map-poly*:  $\text{smult } c\ p = \text{map-poly } (\lambda x. c * x)\ p$

*<proof>*

**lemma** *poly-cn timer*:  $\text{cn timer } (\text{poly } p\ z) = \text{poly } (\text{map-poly } \text{cn timer } p)\ (\text{cn timer } z)$

*<proof>*

**lemma** *poly-cn timer-real*:

**assumes**  $\bigwedge n. \text{poly.co timer } p\ n \in \mathbb{R}$

**shows**  $\text{cn timer } (\text{poly } p\ z) = \text{poly } p\ (\text{cn timer } z)$

*<proof>*

**lemma** *real-poly-cn timer-root-iff*:

**assumes**  $\bigwedge n. \text{poly.co timer } p\ n \in \mathbb{R}$

**shows**  $\text{poly } p\ (\text{cn timer } z) = 0 \iff \text{poly } p\ z = 0$

*<proof>*

**lemma** *sum-to-poly*:  $(\sum x \in A. [:f\ x:]) = [: \sum x \in A. f\ x:]$

*<proof>*

**lemma** *diff-to-poly*:  $[:c:] - [:d:] = [:c - d:]$

*<proof>*

**lemma** *mult-to-poly*:  $[:c:] * [:d:] = [:c * d:]$

*<proof>*

**lemma** *prod-to-poly*:  $(\prod x \in A. [:f\ x:]) = [: \prod x \in A. f\ x:]$

*<proof>*

**lemma** *poly-map-poly-cn timer* [simp]:  $\text{poly } (\text{map-poly } \text{cn timer } p)\ x = \text{cn timer } (\text{poly } p\ (\text{cn timer } x))$

*<proof>*

**lemma** *map-poly-degree-eq*:

**assumes**  $f\ (\text{lead-co timer } p) \neq 0$

**shows**  $\text{degree } (\text{map-poly } f\ p) = \text{degree } p$

*<proof>*

**lemma** *map-poly-degree-less*:

**assumes**  $f\ (\text{lead-co timer } p) = 0\ \text{degree } p \neq 0$

**shows**  $\text{degree } (\text{map-poly } f \ p) < \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma** *map-poly-degree-leq*:  
**shows**  $\text{degree } (\text{map-poly } f \ p) \leq \text{degree } p$   
 $\langle \text{proof} \rangle$

## 4.16 Conversions

**lemma** *of-nat-poly*:  $\text{of-nat } n = [\text{of-nat } n:]$   
 $\langle \text{proof} \rangle$

**lemma** *of-nat-monom*:  $\text{of-nat } n = \text{monom } (\text{of-nat } n) \ 0$   
 $\langle \text{proof} \rangle$

**lemma** *degree-of-nat [simp]*:  $\text{degree } (\text{of-nat } n) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-of-nat [simp]*:  $\text{lead-coeff } (\text{of-nat } n) = \text{of-nat } n$   
 $\langle \text{proof} \rangle$

**lemma** *of-int-poly*:  $\text{of-int } k = [\text{of-int } k:]$   
 $\langle \text{proof} \rangle$

**lemma** *of-int-monom*:  $\text{of-int } k = \text{monom } (\text{of-int } k) \ 0$   
 $\langle \text{proof} \rangle$

**lemma** *degree-of-int [simp]*:  $\text{degree } (\text{of-int } k) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-of-int [simp]*:  $\text{lead-coeff } (\text{of-int } k) = \text{of-int } k$   
 $\langle \text{proof} \rangle$

**lemma** *poly-of-nat [simp]*:  $\text{poly } (\text{of-nat } n) \ x = \text{of-nat } n$   
 $\langle \text{proof} \rangle$

**lemma** *poly-of-int [simp]*:  $\text{poly } (\text{of-int } n) \ x = \text{of-int } n$   
 $\langle \text{proof} \rangle$

**lemma** *poly-numeral [simp]*:  $\text{poly } (\text{numeral } n) \ x = \text{numeral } n$   
 $\langle \text{proof} \rangle$

**lemma** *numeral-poly*:  $\text{numeral } n = [\text{numeral } n:]$   
 $\langle \text{proof} \rangle$

**lemma** *numeral-monom*:  
 $\text{numeral } n = \text{monom } (\text{numeral } n) \ 0$   
 $\langle \text{proof} \rangle$

**lemma** *degree-numeral* [simp]:

*degree* (numeral  $n$ ) = 0

⟨proof⟩

**lemma** *lead-coeff-numeral* [simp]:

*lead-coeff* (numeral  $n$ ) = numeral  $n$

⟨proof⟩

**lemma** *coeff-linear-poly-power*:

**fixes**  $c :: 'a :: \text{semiring-1}$

**assumes**  $i \leq n$

**shows**  $\text{coeff } ([a, b] \wedge n) i = \text{of\_nat } (n \text{ choose } i) * b \wedge i * a \wedge (n - i)$

⟨proof⟩

## 4.17 Lemmas about divisibility

**lemma** *dvd-smult*:

**assumes**  $p \text{ dvd } q$

**shows**  $p \text{ dvd smult } a \ q$

⟨proof⟩

**lemma** *dvd-smult-cancel*:  $p \text{ dvd smult } a \ q \implies a \neq 0 \implies p \text{ dvd } q$

**for**  $a :: 'a :: \text{field}$

⟨proof⟩

**lemma** *dvd-smult-iff*:  $a \neq 0 \implies p \text{ dvd smult } a \ q \longleftrightarrow p \text{ dvd } q$

**for**  $a :: 'a :: \text{field}$

⟨proof⟩

**lemma** *smult-dvd-cancel*:

**assumes**  $\text{smult } a \ p \text{ dvd } q$

**shows**  $p \text{ dvd } q$

⟨proof⟩

**lemma** *smult-dvd*:  $p \text{ dvd } q \implies a \neq 0 \implies \text{smult } a \ p \text{ dvd } q$

**for**  $a :: 'a :: \text{field}$

⟨proof⟩

**lemma** *smult-dvd-iff*:  $\text{smult } a \ p \text{ dvd } q \longleftrightarrow (\text{if } a = 0 \text{ then } q = 0 \text{ else } p \text{ dvd } q)$

**for**  $a :: 'a :: \text{field}$

⟨proof⟩

**lemma** *is-unit-smult-iff*:  $\text{smult } c \ p \text{ dvd } 1 \longleftrightarrow c \text{ dvd } 1 \wedge p \text{ dvd } 1$

⟨proof⟩

## 4.18 Polynomials form an integral domain

**instance** *poly* :: (*idom*) *idom* ⟨proof⟩

**instance** *poly* :: (*{ring-char-0, comm-ring-1}*) *ring-char-0*

$\langle \text{proof} \rangle$

**lemma** *semiring-char-poly* [simp]:  $\text{CHAR}('a :: \text{comm-semiring-1 poly}) = \text{CHAR}('a)$   
 $\langle \text{proof} \rangle$

**instance** *poly* :: ( $\{\text{semiring-prime-char}, \text{comm-semiring-1}\}$ ) *semiring-prime-char*  
 $\langle \text{proof} \rangle$

**instance** *poly* :: ( $\{\text{comm-semiring-prime-char}, \text{comm-semiring-1}\}$ ) *comm-semiring-prime-char*  
 $\langle \text{proof} \rangle$

**instance** *poly* :: ( $\{\text{comm-ring-prime-char}, \text{comm-semiring-1}\}$ ) *comm-ring-prime-char*  
 $\langle \text{proof} \rangle$

**instance** *poly* :: ( $\{\text{idom-prime-char}, \text{comm-semiring-1}\}$ ) *idom-prime-char*  
 $\langle \text{proof} \rangle$

**lemma** *degree-mult-eq*:  $p \neq 0 \implies q \neq 0 \implies \text{degree } (p * q) = \text{degree } p + \text{degree } q$   
**for**  $p \ q :: 'a :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *degree-prod-sum-eq*:  
 $(\bigwedge x. x \in A \implies f \ x \neq 0) \implies$   
 $\text{degree } (\text{prod } f \ A :: 'a :: \text{idom poly}) = (\sum_{x \in A}. \text{degree } (f \ x))$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-imp-degree*:  
 $\langle \text{degree } x \leq \text{degree } y \rangle \text{ if } \langle x \text{ dvd } y \rangle \langle x \neq 0 \rangle \langle y \neq 0 \rangle$   
**for**  $x \ y :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *degree-prod-eq-sum-degree*:  
**fixes**  $A :: 'a \text{ set}$   
**and**  $f :: 'a \Rightarrow 'b :: \text{idom poly}$   
**assumes**  $f0: \forall i \in A. f \ i \neq 0$   
**shows**  $\text{degree } (\prod_{i \in A}. (f \ i)) = (\sum_{i \in A}. \text{degree } (f \ i))$   
 $\langle \text{proof} \rangle$

**lemma** *degree-mult-eq-0*:  
 $\text{degree } (p * q) = 0 \iff p = 0 \vee q = 0 \vee (p \neq 0 \wedge q \neq 0 \wedge \text{degree } p = 0 \wedge \text{degree } q = 0)$   
**for**  $p \ q :: 'a :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *degree-power-eq*:  $p \neq 0 \implies \text{degree } ((p :: 'a :: \text{idom poly}) ^ n) = n * \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma** *degree-mult-right-le*:  
**fixes**  $p \ q :: 'a :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
**assumes**  $q \neq 0$   
**shows**  $\text{degree } p \leq \text{degree } (p * q)$

$\langle \text{proof} \rangle$

**lemma** *coeff-degree-mult*:  $\text{coeff } (p * q) (\text{degree } (p * q)) = \text{coeff } q (\text{degree } q) * \text{coeff } p (\text{degree } p)$   
**for**  $p \ q :: 'a::\{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *dvd-imp-degree-le*:  $p \text{ dvd } q \implies q \neq 0 \implies \text{degree } p \leq \text{degree } q$   
**for**  $p \ q :: 'a::\{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *divides-degree*:  
**fixes**  $p \ q :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
**assumes**  $p \text{ dvd } q$   
**shows**  $\text{degree } p \leq \text{degree } q \vee q = 0$   
 $\langle \text{proof} \rangle$

**lemma** *const-poly-dvd-iff*:  
**fixes**  $c :: 'a::\{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$   
**shows**  $[:c:] \text{ dvd } p \longleftrightarrow (\forall n. c \text{ dvd } \text{coeff } p \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *const-poly-dvd-const-poly-iff* [simp]:  $[:a:] \text{ dvd } [:b:] \longleftrightarrow a \text{ dvd } b$   
**for**  $a \ b :: 'a::\{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-mult*:  $\text{lead-coeff } (p * q) = \text{lead-coeff } p * \text{lead-coeff } q$   
**for**  $p \ q :: 'a::\{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-prod*:  $\text{lead-coeff } (\text{prod } f \ A) = (\prod_{x \in A.} \text{lead-coeff } (f \ x))$   
**for**  $f :: 'a \Rightarrow 'b::\{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-smult*:  $\text{lead-coeff } (\text{smult } c \ p) = c * \text{lead-coeff } p$   
**for**  $p :: 'a::\{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-1* [simp]:  $\text{lead-coeff } 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *lead-coeff-power*:  $\text{lead-coeff } (p \wedge n) = \text{lead-coeff } p \wedge n$   
**for**  $p :: 'a::\{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

## 4.19 Polynomials form an ordered integral domain

**definition** *pos-poly* ::  $'a::\text{linordered-semidom}$  *poly*  $\Rightarrow$  *bool*  
**where**  $\text{pos-poly } p \longleftrightarrow 0 < \text{coeff } p (\text{degree } p)$



**lemma** *pos-poly-pCons*:  $\text{pos-poly } (p\text{Cons } a \ p) \longleftrightarrow \text{pos-poly } p \vee (p = 0 \wedge 0 < a)$   
 $\langle \text{proof} \rangle$

**lemma** *not-pos-poly-0* [simp]:  $\neg \text{pos-poly } 0$   
 $\langle \text{proof} \rangle$

**lemma** *pos-poly-add*:  $\text{pos-poly } p \implies \text{pos-poly } q \implies \text{pos-poly } (p + q)$   
 $\langle \text{proof} \rangle$

**lemma** *pos-poly-mult*:  $\text{pos-poly } p \implies \text{pos-poly } q \implies \text{pos-poly } (p * q)$   
 $\langle \text{proof} \rangle$

**lemma** *pos-poly-total*:  $p = 0 \vee \text{pos-poly } p \vee \text{pos-poly } (-p)$   
**for**  $p :: 'a::\text{linordered-idom}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *pos-poly-coeffs* [code]:  $\text{pos-poly } p \longleftrightarrow (\text{let } as = \text{coeffs } p \text{ in } as \neq [] \wedge \text{last } as > 0)$   
**(is** *?lhs*  $\longleftrightarrow$  *?rhs*)  
 $\langle \text{proof} \rangle$

**instantiation** *poly* :: (*linordered-idom*) *linordered-idom*  
**begin**

**definition**  $x < y \longleftrightarrow \text{pos-poly } (y - x)$

**definition**  $x \leq y \longleftrightarrow x = y \vee \text{pos-poly } (y - x)$

**definition**  $|x::'a \ \text{poly}| = (\text{if } x < 0 \text{ then } -x \text{ else } x)$

**definition**  $\text{sgn } (x::'a \ \text{poly}) = (\text{if } x = 0 \text{ then } 0 \text{ else if } 0 < x \text{ then } 1 \text{ else } -1)$

**instance**  
 $\langle \text{proof} \rangle$

**end**

TODO: Simplification rules for comparisons

## 4.20 Synthetic division and polynomial roots

### 4.20.1 Synthetic division

Synthetic division is simply division by the linear polynomial  $x - c$ .

**definition** *synthetic-divmod* ::  $'a::\text{comm-semiring-0}$  *poly*  $\Rightarrow 'a \Rightarrow 'a \ \text{poly} \times 'a$   
**where**  $\text{synthetic-divmod } p \ c = \text{fold-coeffs } (\lambda a \ (q, r). (p\text{Cons } r \ q, a + c * r)) \ p$   
 $(0, 0)$

**definition** *synthetic-div* :: 'a::comm-semiring-0 poly  $\Rightarrow$  'a  $\Rightarrow$  'a poly  
**where** *synthetic-div* p c = fst (*synthetic-divmod* p c)

**lemma** *synthetic-divmod-0* [simp]: *synthetic-divmod* 0 c = (0, 0)  
 <proof>

**lemma** *synthetic-divmod-pCons* [simp]:  
*synthetic-divmod* (pCons a p) c = ( $\lambda(q, r). (pCons r q, a + c * r)$ ) (*synthetic-divmod* p c)  
 <proof>

**lemma** *synthetic-div-0* [simp]: *synthetic-div* 0 c = 0  
 <proof>

**lemma** *synthetic-div-unique-lemma*: smult c p = pCons a p  $\implies$  p = 0  
 <proof>

**lemma** *snd-synthetic-divmod*: snd (*synthetic-divmod* p c) = poly p c  
 <proof>

**lemma** *synthetic-div-pCons* [simp]:  
*synthetic-div* (pCons a p) c = pCons (poly p c) (*synthetic-div* p c)  
 <proof>

**lemma** *synthetic-div-eq-0-iff*: *synthetic-div* p c = 0  $\longleftrightarrow$  degree p = 0  
 <proof>

**lemma** *degree-synthetic-div*: degree (*synthetic-div* p c) = degree p - 1  
 <proof>

**lemma** *synthetic-div-correct*:  
 p + smult c (*synthetic-div* p c) = pCons (poly p c) (*synthetic-div* p c)  
 <proof>

**lemma** *synthetic-div-unique*: p + smult c q = pCons r q  $\implies$  r = poly p c  $\wedge$  q = *synthetic-div* p c  
 <proof>

**lemma** *synthetic-div-correct'*:  $[-c, 1:] * \text{synthetic-div } p \text{ c} + [: \text{poly } p \text{ c}:] = p$   
**for** c :: 'a::comm-ring-1  
 <proof>

#### 4.20.2 Polynomial roots

**lemma** *poly-eq-0-iff-dvd*: poly p c = 0  $\longleftrightarrow$   $[-c, 1:] \text{ dvd } p$   
 (is ?lhs  $\longleftrightarrow$  ?rhs)  
**for** c :: 'a::comm-ring-1  
 <proof>

**lemma** *dvd-iff-poly-eq-0*:  $[c, 1:] \text{ dvd } p \longleftrightarrow \text{poly } p \ (-c) = 0$   
**for**  $c :: 'a::\text{comm-ring-1}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-roots-finite*:  $p \neq 0 \implies \text{finite } \{x. \text{poly } p \ x = 0\}$   
**for**  $p :: 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *poly-eq-poly-eq-iff*:  $\text{poly } p = \text{poly } q \longleftrightarrow p = q$   
**(is**  $?lhs \longleftrightarrow ?rhs$   
**for**  $p \ q :: 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}, \text{ring-char-0}\}$  *poly*  
 $\langle \text{proof} \rangle$

A nice extension rule for polynomials.

**lemma** *poly-ext*:  
**fixes**  $p \ q :: 'a :: \{\text{ring-char-0}, \text{idom}\}$  *poly*  
**assumes**  $\bigwedge x. \text{poly } p \ x = \text{poly } q \ x$  **shows**  $p = q$   
 $\langle \text{proof} \rangle$

Copied from non-negative variants.

**lemma** *coeff-linear-power-neg[simp]*:  
**fixes**  $a :: 'a::\text{comm-ring-1}$   
**shows**  $\text{coeff } ([a, -1:] \wedge n) \ n = (-1) \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *degree-linear-power-neg[simp]*:  
**fixes**  $a :: 'a::\{\text{idom}, \text{comm-ring-1}\}$   
**shows**  $\text{degree } ([a, -1:] \wedge n) = n$   
 $\langle \text{proof} \rangle$

**lemma** *poly-all-0-iff-0*:  $(\forall x. \text{poly } p \ x = 0) \longleftrightarrow p = 0$   
**for**  $p :: 'a::\{\text{ring-char-0}, \text{comm-ring-1}, \text{ring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *card-poly-roots-bound*:  
**fixes**  $p :: 'a::\{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$  *poly*  
**assumes**  $p \neq 0$   
**shows**  $\text{card } \{x. \text{poly } p \ x = 0\} \leq \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma** *poly-eqI-degree*:  
**fixes**  $p \ q :: 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$  *poly*  
**assumes**  $\bigwedge x. x \in A \implies \text{poly } p \ x = \text{poly } q \ x$   
**assumes**  $\text{card } A > \text{degree } p \ \text{card } A > \text{degree } q$   
**shows**  $p = q$   
 $\langle \text{proof} \rangle$

**lemma** *poly-eqI-degree-lead-coeff*:  
**fixes**  $p \ q :: 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$  *poly*

**assumes**  $\text{poly.coeff } p \ n = \text{poly.coeff } q \ n \ \text{card } A \geq n \ \text{degree } p \leq n \ \text{degree } q \leq n$   
**assumes**  $\bigwedge z. z \in A \implies \text{poly } p \ z = \text{poly } q \ z$   
**shows**  $p = q$   
 $\langle \text{proof} \rangle$

#### 4.20.3 Order of polynomial roots

**definition**  $\text{order} :: 'a::\text{idom} \Rightarrow 'a \ \text{poly} \Rightarrow \text{nat}$   
**where**  $\text{order } a \ p = (\text{LEAST } n. \neg [:-a, 1:] \wedge \text{Suc } n \ \text{dvd } p)$

**lemma**  $\text{coeff-linear-power: coeff } ([:-a, 1:] \wedge n) \ n = 1$   
**for**  $a :: 'a::\text{comm-semiring-1}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{degree-linear-power: degree } ([:-a, 1:] \wedge n) = n$   
**for**  $a :: 'a::\text{comm-semiring-1}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-1: } [:-a, 1:] \wedge \text{order } a \ p \ \text{dvd } p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-2:}$   
**assumes**  $p \neq 0$   
**shows**  $\neg [:-a, 1:] \wedge \text{Suc } (\text{order } a \ p) \ \text{dvd } p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order: } p \neq 0 \implies [:-a, 1:] \wedge \text{order } a \ p \ \text{dvd } p \wedge \neg [:-a, 1:] \wedge \text{Suc } (\text{order } a \ p) \ \text{dvd } p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-degree:}$   
**assumes**  $p: p \neq 0$   
**shows**  $\text{order } a \ p \leq \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-root: } \text{poly } p \ a = 0 \longleftrightarrow p = 0 \vee \text{order } a \ p \neq 0 \ (\text{is } ?lhs = ?rhs)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-0I: } \text{poly } p \ a \neq 0 \implies \text{order } a \ p = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-unique-lemma:}$   
**fixes**  $p :: 'a::\text{idom} \ \text{poly}$   
**assumes**  $[:-a, 1:] \wedge n \ \text{dvd } p \wedge \neg [:-a, 1:] \wedge \text{Suc } n \ \text{dvd } p$   
**shows**  $\text{order } a \ p = n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{order-mult:}$   
**assumes**  $p * q \neq 0$  **shows**  $\text{order } a \ (p * q) = \text{order } a \ p + \text{order } a \ q$

$\langle proof \rangle$

**lemma** *order-smult*:

**assumes**  $c \neq 0$

**shows**  $order\ x\ (smult\ c\ p) = order\ x\ p$

$\langle proof \rangle$

**lemma** *order-gt-0-iff*:  $p \neq 0 \implies order\ x\ p > 0 \longleftrightarrow poly\ p\ x = 0$

$\langle proof \rangle$

**lemma** *order-eq-0-iff*:  $p \neq 0 \implies order\ x\ p = 0 \longleftrightarrow poly\ p\ x \neq 0$

$\langle proof \rangle$

Next three lemmas contributed by Wenda Li

**lemma** *order-1-eq-0* [*simp*]:  $order\ x\ 1 = 0$

$\langle proof \rangle$

**lemma** *order-uminus*[*simp*]:  $order\ x\ (-p) = order\ x\ p$

$\langle proof \rangle$

**lemma** *order-power-n-n*:  $order\ a\ ([: -a, 1:]^{\wedge} n) = n$

$\langle proof \rangle$

**lemma** *order-0-monom* [*simp*]:  $c \neq 0 \implies order\ 0\ (monom\ c\ n) = n$

$\langle proof \rangle$

**lemma** *dvd-imp-order-le*:  $q \neq 0 \implies p\ dvd\ q \implies Polynomial.order\ a\ p \leq Polynomial.order\ a\ q$

$\langle proof \rangle$

Now justify the standard squarefree decomposition, i.e.  $f / gcd\ f\ f'$ .

**lemma** *order-divides*:  $[: -a, 1:]^{\wedge} n\ dvd\ p \longleftrightarrow p = 0 \vee n \leq order\ a\ p$

$\langle proof \rangle$

**lemma** *order-decomp*:

**assumes**  $p \neq 0$

**shows**  $\exists q. p = [: -a, 1:]^{\wedge} order\ a\ p * q \wedge \neg [: -a, 1:]^{\wedge} n\ dvd\ q$

$\langle proof \rangle$

**lemma** *monom-1-dvd-iff*:  $p \neq 0 \implies monom\ 1\ n\ dvd\ p \longleftrightarrow n \leq order\ 0\ p$

$\langle proof \rangle$

**lemma** *poly-root-order-induct* [*case-names 0 no-roots root*]:

**fixes**  $p :: 'a :: idom\ poly$

**assumes**  $P\ 0 \wedge p. (\bigwedge x. poly\ p\ x \neq 0) \implies P\ p$

$\bigwedge p\ x\ n. n > 0 \implies poly\ p\ x \neq 0 \implies P\ p \implies P\ ([: -x, 1:]^{\wedge} n * p)$

**shows**  $P\ p$

$\langle proof \rangle$

```

context
  includes multiset.lifting
begin

lift-definition proots :: ('a :: idom) poly  $\Rightarrow$  'a multiset is
   $\lambda(p :: 'a \text{ poly}) (x :: 'a). \text{ if } p = 0 \text{ then } 0 \text{ else order } x \text{ } p$ 
   $\langle \text{proof} \rangle$ 

lemma proots-0 [simp]: proots (0 :: 'a :: idom poly) = {#}
   $\langle \text{proof} \rangle$ 

lemma proots-1 [simp]: proots (1 :: 'a :: idom poly) = {#}
   $\langle \text{proof} \rangle$ 

lemma proots-const [simp]: proots [: x :] = 0
   $\langle \text{proof} \rangle$ 

lemma proots-numeral [simp]: proots (numeral n) = 0
   $\langle \text{proof} \rangle$ 

lemma count-proots [simp]:
   $p \neq 0 \implies \text{count } (\text{proots } p) \text{ } a = \text{order } a \text{ } p$ 
   $\langle \text{proof} \rangle$ 

lemma set-count-proots [simp]:
   $p \neq 0 \implies \text{set-mset } (\text{proots } p) = \{x. \text{poly } p \text{ } x = 0\}$ 
   $\langle \text{proof} \rangle$ 

lemma proots-uminus [simp]: proots (-p) = proots p
   $\langle \text{proof} \rangle$ 

lemma proots-smult [simp]:  $c \neq 0 \implies \text{proots } (\text{smult } c \text{ } p) = \text{proots } p$ 
   $\langle \text{proof} \rangle$ 

lemma proots-mult:
  assumes  $p \neq 0 \text{ } q \neq 0$ 
  shows  $\text{proots } (p * q) = \text{proots } p + \text{proots } q$ 
   $\langle \text{proof} \rangle$ 

lemma proots-prod:
  assumes  $\bigwedge x. x \in A \implies f \text{ } x \neq 0$ 
  shows  $\text{proots } (\prod_{x \in A}. f \text{ } x) = (\sum_{x \in A}. \text{proots } (f \text{ } x))$ 
   $\langle \text{proof} \rangle$ 

lemma proots-prod-mset:
  assumes  $0 \notin \# A$ 
  shows  $\text{proots } (\prod_{p \in \# A}. p) = (\sum_{p \in \# A}. \text{proots } p)$ 

```

$\langle \text{proof} \rangle$

**lemma** *proots-prod-list*:

**assumes**  $0 \notin \text{set } ps$

**shows**  $\text{proots } (\prod p \leftarrow ps. p) = (\sum p \leftarrow ps. \text{proots } p)$

$\langle \text{proof} \rangle$

**lemma** *proots-power*:  $\text{proots } (p \wedge n) = \text{repeat-mset } n (\text{proots } p)$

$\langle \text{proof} \rangle$

**lemma** *proots-linear-factor* [simp]:  $\text{proots } [:x, 1:] = \{\#-x\# \}$

$\langle \text{proof} \rangle$

**lemma** *size-proots-le*:  $\text{size } (\text{proots } p) \leq \text{degree } p$

$\langle \text{proof} \rangle$

**end**

## 4.21 Additional induction rules on polynomials

An induction rule for induction over the roots of a polynomial with a certain property. (e.g. all positive roots)

**lemma** *poly-root-induct* [case-names 0 no-roots root]:

**fixes**  $p :: 'a :: \text{idom } \text{poly}$

**assumes**  $Q \ 0$

**and**  $\bigwedge p. (\bigwedge a. P \ a \implies \text{poly } p \ a \neq 0) \implies Q \ p$

**and**  $\bigwedge a \ p. P \ a \implies Q \ p \implies Q \ ([:a, -1:] * p)$

**shows**  $Q \ p$

$\langle \text{proof} \rangle$

**lemma** *dropWhile-replicate-append*:

$\text{dropWhile } ((=) \ a) (\text{replicate } n \ a @ \text{ys}) = \text{dropWhile } ((=) \ a) \ \text{ys}$

$\langle \text{proof} \rangle$

**lemma** *Poly-append-replicate-0*:  $\text{Poly } (xs @ \text{replicate } n \ 0) = \text{Poly } xs$

$\langle \text{proof} \rangle$

An induction rule for simultaneous induction over two polynomials, prepending one coefficient in each step.

**lemma** *poly-induct2* [case-names 0 pCons]:

**assumes**  $P \ 0 \ 0 \ \bigwedge a \ p \ b \ q. P \ p \ q \implies P \ (pCons \ a \ p) \ (pCons \ b \ q)$

**shows**  $P \ p \ q$

$\langle \text{proof} \rangle$

## 4.22 Composition of polynomials

**definition** *pcompose* ::  $'a :: \text{comm-semiring-0 } \text{poly} \Rightarrow 'a \ \text{poly} \Rightarrow 'a \ \text{poly}$

**where**  $pcompose \ p \ q = \text{fold-coeffs } (\lambda a \ c. [:a:] + q * c) \ p \ 0$

**notation**  $pcompose$  (**infixl**  $\circ_p$  71)

**lemma**  $pcompose-0$  [*simp*]:  $pcompose\ 0\ q = 0$   
 $\langle proof \rangle$

**lemma**  $pcompose-pCons$ :  $pcompose\ (pCons\ a\ p)\ q = [:a:] + q * pcompose\ p\ q$   
 $\langle proof \rangle$

**lemma**  $pcompose-altdef$ :  $pcompose\ p\ q = poly\ (map-poly\ (\lambda x. [:x:])\ p)\ q$   
 $\langle proof \rangle$

**lemma**  $coeff-pcompose-0$  [*simp*]:  
 $coeff\ (pcompose\ p\ q)\ 0 = poly\ p\ (coeff\ q\ 0)$   
 $\langle proof \rangle$

**lemma**  $pcompose-1$ :  $pcompose\ 1\ p = 1$   
**for**  $p :: 'a::comm-semiring-1\ poly$   
 $\langle proof \rangle$

**lemma**  $poly-pcompose$ :  $poly\ (pcompose\ p\ q)\ x = poly\ p\ (poly\ q\ x)$   
 $\langle proof \rangle$

**lemma**  $degree-pcompose-le$ :  $degree\ (pcompose\ p\ q) \leq degree\ p * degree\ q$   
 $\langle proof \rangle$

**lemma**  $pcompose-add$ :  $pcompose\ (p + q)\ r = pcompose\ p\ r + pcompose\ q\ r$   
**for**  $p\ q\ r :: 'a::\{comm-semiring-0, ab-semigroup-add\}\ poly$   
 $\langle proof \rangle$

**lemma**  $pcompose-uminus$ :  $pcompose\ (-p)\ r = -pcompose\ p\ r$   
**for**  $p\ r :: 'a::comm-ring\ poly$   
 $\langle proof \rangle$

**lemma**  $pcompose-diff$ :  $pcompose\ (p - q)\ r = pcompose\ p\ r - pcompose\ q\ r$   
**for**  $p\ q\ r :: 'a::comm-ring\ poly$   
 $\langle proof \rangle$

**lemma**  $pcompose-smult$ :  $pcompose\ (smult\ a\ p)\ r = smult\ a\ (pcompose\ p\ r)$   
**for**  $p\ r :: 'a::comm-semiring-0\ poly$   
 $\langle proof \rangle$

**lemma**  $pcompose-mult$ :  $pcompose\ (p * q)\ r = pcompose\ p\ r * pcompose\ q\ r$   
**for**  $p\ q\ r :: 'a::comm-semiring-0\ poly$   
 $\langle proof \rangle$

**lemma**  $pcompose-assoc$ :  $pcompose\ p\ (pcompose\ q\ r) = pcompose\ (pcompose\ p\ q)\ r$   
**for**  $p\ q\ r :: 'a::comm-semiring-0\ poly$   
 $\langle proof \rangle$



**lemma** *pcompose-idR[simp]*:  $pcompose\ p\ [:0, 1:] = p$   
**for**  $p :: 'a::comm-semiring-1\ poly$   
 $\langle proof \rangle$

**lemma** *pcompose-sum*:  $pcompose\ (sum\ f\ A)\ p = sum\ (\lambda i. pcompose\ (f\ i)\ p)\ A$   
 $\langle proof \rangle$

**lemma** *pcompose-prod*:  $pcompose\ (prod\ f\ A)\ p = prod\ (\lambda i. pcompose\ (f\ i)\ p)\ A$   
 $\langle proof \rangle$

**lemma** *pcompose-const [simp]*:  $pcompose\ [:a:]\ q = [:a:]$   
 $\langle proof \rangle$

**lemma** *pcompose-0'*:  $pcompose\ p\ 0 = [:coeff\ p\ 0:]$   
 $\langle proof \rangle$

**lemma** *pcompose-coeff-0*:  
 $coeff\ (pcompose\ p\ q)\ 0 = poly\ p\ (coeff\ q\ 0)$   
 $\langle proof \rangle$

**lemma** *pcompose-pCons-0*:  $pcompose\ p\ [:a:] = [:poly\ p\ a:]$   
 $\langle proof \rangle$

**lemma** *degree-pcompose*:  $degree\ (pcompose\ p\ q) = degree\ p * degree\ q$   
**for**  $p\ q :: 'a::\{comm-semiring-0, semiring-no-zero-divisors\}\ poly$   
 $\langle proof \rangle$

**lemma** *pcompose-eq-0*:  
**fixes**  $p\ q :: 'a::\{comm-semiring-0, semiring-no-zero-divisors\}\ poly$   
**assumes**  $pcompose\ p\ q = 0\ degree\ q > 0$   
**shows**  $p = 0$   
 $\langle proof \rangle$

**lemma** *pcompose-eq-0-iff*:  
**fixes**  $p\ q :: 'a::\{comm-semiring-0, semiring-no-zero-divisors\}\ poly$   
**assumes**  $degree\ q > 0$   
**shows**  $pcompose\ p\ q = 0 \longleftrightarrow p = 0$   
 $\langle proof \rangle$

**lemma** *coeff-pcompose-linear*:  
 $coeff\ (pcompose\ p\ [:0, a :: 'a :: comm-semiring-1:])\ i = a ^ i * coeff\ p\ i$   
 $\langle proof \rangle$

**lemma** *lead-coeff-comp*:  
**fixes**  $p\ q :: 'a::\{comm-semiring-1, semiring-no-zero-divisors\}\ poly$   
**assumes**  $degree\ q > 0$   
**shows**  $lead-coeff\ (pcompose\ p\ q) = lead-coeff\ p * lead-coeff\ q ^ (degree\ p)$   
 $\langle proof \rangle$

**lemma** *coeff-pcompose-monom-linear* [simp]:  
**fixes**  $p :: 'a :: \text{comm-ring-1 poly}$   
**shows**  $\text{coeff } (\text{pcompose } p (\text{monom } c (\text{Suc } 0))) k = c \wedge k * \text{coeff } p k$   
 $\langle \text{proof} \rangle$

**lemma** *of-nat-mult-conv-smult*:  $\text{of-nat } n * P = \text{smult } (\text{of-nat } n) P$   
 $\langle \text{proof} \rangle$

**lemma** *numeral-mult-conv-smult*:  $\text{numeral } n * P = \text{smult } (\text{numeral } n) P$   
 $\langle \text{proof} \rangle$

**lemma** *sum-order-le-degree*:  
**assumes**  $p \neq 0$   
**shows**  $(\sum x \mid \text{poly } p \ x = 0. \text{ order } x \ p) \leq \text{degree } p$   
 $\langle \text{proof} \rangle$

## 4.23 Closure properties of coefficients

**context**  
**fixes**  $R :: 'a :: \text{comm-semiring-1 set}$   
**assumes**  $R-0$ :  $0 \in R$   
**assumes**  $R\text{-plus}$ :  $\bigwedge x \ y. x \in R \implies y \in R \implies x + y \in R$   
**assumes**  $R\text{-mult}$ :  $\bigwedge x \ y. x \in R \implies y \in R \implies x * y \in R$   
**begin**

**lemma** *coeff-mult-semiring-closed*:  
**assumes**  $\bigwedge i. \text{coeff } p \ i \in R \ \bigwedge i. \text{coeff } q \ i \in R$   
**shows**  $\text{coeff } (p * q) \ i \in R$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-pcompose-semiring-closed*:  
**assumes**  $\bigwedge i. \text{coeff } p \ i \in R \ \bigwedge i. \text{coeff } q \ i \in R$   
**shows**  $\text{coeff } (\text{pcompose } p \ q) \ i \in R$   
 $\langle \text{proof} \rangle$

**end**

## 4.24 Shifting polynomials

**definition** *poly-shift* ::  $\text{nat} \Rightarrow 'a::\text{zero poly} \Rightarrow 'a \text{ poly}$   
**where**  $\text{poly-shift } n \ p = \text{Abs-poly } (\lambda i. \text{coeff } p \ (i + n))$

**lemma** *nth-default-drop*:  $\text{nth-default } x \ (\text{drop } n \ xs) \ m = \text{nth-default } x \ xs \ (m + n)$   
 $\langle \text{proof} \rangle$

**lemma** *nth-default-take*:  $\text{nth-default } x \ (\text{take } n \ xs) \ m = (\text{if } m < n \text{ then } \text{nth-default } x \ xs \ m \text{ else } x)$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-poly-shift*:  $\text{coeff } (\text{poly-shift } n \ p) \ i = \text{coeff } p \ (i + n)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-shift-id* [simp]:  $\text{poly-shift } 0 = (\lambda x. x)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-shift-0* [simp]:  $\text{poly-shift } n \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *poly-shift-1*:  $\text{poly-shift } n \ 1 = (\text{if } n = 0 \text{ then } 1 \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-shift-monom*:  $\text{poly-shift } n \ (\text{monom } c \ m) = (\text{if } m \geq n \text{ then } \text{monom } c \ (m - n) \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *coeffs-shift-poly* [code abstract]:  
 $\text{coeffs } (\text{poly-shift } n \ p) = \text{drop } n \ (\text{coeffs } p)$   
 $\langle \text{proof} \rangle$

## 4.25 Truncating polynomials

**definition** *poly-cutoff*  
**where**  $\text{poly-cutoff } n \ p = \text{Abs-poly } (\lambda k. \text{if } k < n \text{ then } \text{coeff } p \ k \text{ else } 0)$

**lemma** *coeff-poly-cutoff*:  $\text{coeff } (\text{poly-cutoff } n \ p) \ k = (\text{if } k < n \text{ then } \text{coeff } p \ k \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-cutoff-0* [simp]:  $\text{poly-cutoff } n \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *poly-cutoff-1* [simp]:  $\text{poly-cutoff } n \ 1 = (\text{if } n = 0 \text{ then } 0 \text{ else } 1)$   
 $\langle \text{proof} \rangle$

**lemma** *coeffs-poly-cutoff* [code abstract]:  
 $\text{coeffs } (\text{poly-cutoff } n \ p) = \text{strip-while } ((=) \ 0) \ (\text{take } n \ (\text{coeffs } p))$   
 $\langle \text{proof} \rangle$

## 4.26 Reflecting polynomials

**definition** *reflect-poly* ::  $'a::\text{zero poly} \Rightarrow 'a \text{ poly}$   
**where**  $\text{reflect-poly } p = \text{Poly } (\text{rev } (\text{coeffs } p))$

**lemma** *coeffs-reflect-poly* [code abstract]:  
 $\text{coeffs } (\text{reflect-poly } p) = \text{rev } (\text{dropWhile } ((=) \ 0) \ (\text{coeffs } p))$   
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-0* [simp]:  $\text{reflect-poly } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-1* [simp]: *reflect-poly 1 = 1*

*<proof>*

**lemma** *coeff-reflect-poly*:

*coeff (reflect-poly p) n = (if n > degree p then 0 else coeff p (degree p - n))*

*<proof>*

**lemma** *coeff-0-reflect-poly-0-iff* [simp]: *coeff (reflect-poly p) 0 = 0  $\longleftrightarrow$  p = 0*

*<proof>*

**lemma** *reflect-poly-at-0-eq-0-iff* [simp]: *poly (reflect-poly p) 0 = 0  $\longleftrightarrow$  p = 0*

*<proof>*

**lemma** *reflect-poly-pCons'*:

*p  $\neq$  0  $\implies$  reflect-poly (pCons c p) = reflect-poly p + monom c (Suc (degree p))*

*<proof>*

**lemma** *reflect-poly-const* [simp]: *reflect-poly [:a:] = [:a:]*

*<proof>*

**lemma** *poly-reflect-poly-nz*:

*x  $\neq$  0  $\implies$  poly (reflect-poly p) x = x  $^{\wedge}$  degree p \* poly p (inverse x)*

**for** *x :: 'a::field*

*<proof>*

**lemma** *coeff-0-reflect-poly* [simp]: *coeff (reflect-poly p) 0 = lead-coeff p*

*<proof>*

**lemma** *poly-reflect-poly-0* [simp]: *poly (reflect-poly p) 0 = lead-coeff p*

*<proof>*

**lemma** *reflect-poly-reflect-poly* [simp]: *coeff p 0  $\neq$  0  $\implies$  reflect-poly (reflect-poly p) = p*

*<proof>*

**lemma** *degree-reflect-poly-le*: *degree (reflect-poly p)  $\leq$  degree p*

*<proof>*

**lemma** *reflect-poly-pCons*: *a  $\neq$  0  $\implies$  reflect-poly (pCons a p) = Poly (rev (a # coeffs p))*

*<proof>*

**lemma** *degree-reflect-poly-eq* [simp]: *coeff p 0  $\neq$  0  $\implies$  degree (reflect-poly p) = degree p*

*<proof>*

**lemma** *reflect-poly-eq-0-iff* [simp]: *reflect-poly p = 0  $\longleftrightarrow$  p = 0*

*<proof>*

**lemma** *reflect-poly-mult*:  $\text{reflect-poly } (p * q) = \text{reflect-poly } p * \text{reflect-poly } q$   
**for**  $p \ q :: 'a :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-smult*:  $\text{reflect-poly } (\text{smult } c \ p) = \text{smult } c \ (\text{reflect-poly } p)$   
**for**  $p :: 'a :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-power*:  $\text{reflect-poly } (p \wedge n) = \text{reflect-poly } p \wedge n$   
**for**  $p :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-prod*:  $\text{reflect-poly } (\text{prod } f \ A) = \text{prod } (\lambda x. \text{reflect-poly } (f \ x)) \ A$   
**for**  $f :: - \Rightarrow :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-prod-list*:  $\text{reflect-poly } (\text{prod-list } xs) = \text{prod-list } (\text{map } \text{reflect-poly } xs)$   
**for**  $xs :: :: \{\text{comm-semiring-0}, \text{semiring-no-zero-divisors}\}$  *poly list*  
 $\langle \text{proof} \rangle$

**lemma** *reflect-poly-Poly-nz*:  
 $\text{no-trailing } (\text{HOL.eq } 0) \ xs \Longrightarrow \text{reflect-poly } (\text{Poly } xs) = \text{Poly } (\text{rev } xs)$   
 $\langle \text{proof} \rangle$

**lemmas** *reflect-poly-simps* =  
 $\text{reflect-poly-0}$   $\text{reflect-poly-1}$   $\text{reflect-poly-const}$   $\text{reflect-poly-smult}$   $\text{reflect-poly-mult}$   
 $\text{reflect-poly-power}$   $\text{reflect-poly-prod}$   $\text{reflect-poly-prod-list}$

## 4.27 Derivatives

**function** *pderiv* ::  $('a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\})$  *poly*  $\Rightarrow 'a$  *poly*  
**where**  $\text{pderiv } (pCons \ a \ p) = (\text{if } p = 0 \text{ then } 0 \text{ else } p + pCons \ 0 \ (\text{pderiv } p))$   
 $\langle \text{proof} \rangle$

**termination** *pderiv*  
 $\langle \text{proof} \rangle$

**declare** *pderiv.simps*[*simp del*]

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

**lemma** *pderiv-pCons*:  $\text{pderiv } (pCons \ a \ p) = p + pCons \ 0 \ (\text{pderiv } p)$   
 $\langle \text{proof} \rangle$

**lemma** *pderiv-1* [simp]: *pderiv* 1 = 0  
 ⟨proof⟩

**lemma** *pderiv-of-nat* [simp]: *pderiv* (of-nat *n*) = 0  
**and** *pderiv-numeral* [simp]: *pderiv* (numeral *m*) = 0  
 ⟨proof⟩

**lemma** *coeff-pderiv*: *coeff* (*pderiv* *p*) *n* = of-nat (Suc *n*) \* *coeff* *p* (Suc *n*)  
 ⟨proof⟩

**fun** *pderiv-coeffs-code* :: 'a::{comm-semiring-1,semiring-no-zero-divisors} ⇒ 'a list  
 ⇒ 'a list  
**where**  
   *pderiv-coeffs-code* *f* (*x* # *xs*) = cCons (*f* \* *x*) (*pderiv-coeffs-code* (*f*+1) *xs*)  
   | *pderiv-coeffs-code* *f* [] = []

**definition** *pderiv-coeffs* :: 'a::{comm-semiring-1,semiring-no-zero-divisors} list ⇒  
 'a list  
**where** *pderiv-coeffs* *xs* = *pderiv-coeffs-code* 1 (tl *xs*)

**lemma** *pderiv-coeffs-code*:  
   *nth-default* 0 (*pderiv-coeffs-code* *f* *xs*) *n* = (*f* + of-nat *n*) \* *nth-default* 0 *xs* *n*  
 ⟨proof⟩

**lemma** *coeffs-pderiv-code* [code abstract]: *coeffs* (*pderiv* *p*) = *pderiv-coeffs* (*coeffs* *p*)  
 ⟨proof⟩

**lemma** *pderiv-eq-0-iff*: *pderiv* *p* = 0 ⟷ *degree* *p* = 0  
**for** *p* :: 'a::{comm-semiring-1,semiring-no-zero-divisors,semiring-char-0} poly  
 ⟨proof⟩

**lemma** *degree-pderiv*: *degree* (*pderiv* *p*) = *degree* *p* - 1  
**for** *p* :: 'a::{comm-semiring-1,semiring-no-zero-divisors,semiring-char-0} poly  
 ⟨proof⟩

**lemma** *not-dvd-pderiv*:  
**fixes** *p* :: 'a::{comm-semiring-1,semiring-no-zero-divisors,semiring-char-0} poly  
**assumes** *degree* *p* ≠ 0  
**shows** ¬ *p* dvd *pderiv* *p*  
 ⟨proof⟩

**lemma** *dvd-pderiv-iff* [simp]: *p* dvd *pderiv* *p* ⟷ *degree* *p* = 0  
**for** *p* :: 'a::{comm-semiring-1,semiring-no-zero-divisors,semiring-char-0} poly  
 ⟨proof⟩

**lemma** *pderiv-singleton* [simp]: *pderiv* [:*a*:] = 0  
 ⟨proof⟩

**lemma** *pderiv-add*:  $pderiv (p + q) = pderiv p + pderiv q$   
 $\langle proof \rangle$

**lemma** *pderiv-minus*:  $pderiv (- p :: 'a :: idom poly) = - pderiv p$   
 $\langle proof \rangle$

**lemma** *pderiv-diff*:  $pderiv ((p :: - :: idom poly) - q) = pderiv p - pderiv q$   
 $\langle proof \rangle$

**lemma** *pderiv-smult*:  $pderiv (smult a p) = smult a (pderiv p)$   
 $\langle proof \rangle$

**lemma** *pderiv-mult*:  $pderiv (p * q) = p * pderiv q + q * pderiv p$   
 $\langle proof \rangle$

**lemma** *pderiv-power-Suc*:  $pderiv (p \wedge Suc n) = smult (of-nat (Suc n)) (p \wedge n) * pderiv p$   
 $\langle proof \rangle$

**lemma** *pderiv-power*:  
 $pderiv (p \wedge n) = smult (of-nat n) (p \wedge (n - 1)) * pderiv p$   
 $\langle proof \rangle$

**lemma** *pderiv-monom*:  
 $pderiv (monom c n) = monom (of-nat n * c) (n - 1)$   
 $\langle proof \rangle$

**lemma** *pderiv-pcompose*:  $pderiv (pcompose p q) = pcompose (pderiv p) q * pderiv q$   
 $\langle proof \rangle$

**lemma** *pderiv-prod*:  $pderiv (prod f (as)) = (\sum a \in as. prod f (as - \{a\}) * pderiv (f a))$   
 $\langle proof \rangle$

**lemma** *coeff-higher-pderiv*:  
 $coeff ((pderiv \wedge m) f) n = pochhammer (of-nat (Suc n)) m * coeff f (n + m)$   
 $\langle proof \rangle$

**lemma** *higher-pderiv-0 [simp]*:  $(pderiv \wedge n) 0 = 0$   
 $\langle proof \rangle$

**lemma** *higher-pderiv-add*:  $(pderiv \wedge n) (p + q) = (pderiv \wedge n) p + (pderiv \wedge n) q$   
 $\langle proof \rangle$

**lemma** *higher-pderiv-smult*:  $(pderiv \wedge n) (smult c p) = smult c ((pderiv \wedge n) p)$   
 $\langle proof \rangle$

**lemma** *higher-pderiv-monom*:

$m \leq n + 1 \implies (\text{pderiv } \sim m) (\text{monom } c \ n) = \text{monom } (\text{pochhammer } (\text{int } n - \text{int } m + 1) \ m * c) (n - m)$   
 $\langle \text{proof} \rangle$

**lemma** *higher-pderiv-monom-eq-zero*:

$m > n + 1 \implies (\text{pderiv } \sim m) (\text{monom } c \ n) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *higher-pderiv-sum*:  $(\text{pderiv } \sim n) (\text{sum } f \ A) = (\sum_{x \in A}. (\text{pderiv } \sim n) (f \ x))$

$\langle \text{proof} \rangle$

**lemma** *higher-pderiv-sum-mset*:  $(\text{pderiv } \sim n) (\text{sum-mset } A) = (\sum_{p \in \#A}. (\text{pderiv } \sim n) \ p)$

$\langle \text{proof} \rangle$

**lemma** *higher-pderiv-sum-list*:  $(\text{pderiv } \sim n) (\text{sum-list } ps) = (\sum_{p \leftarrow ps}. (\text{pderiv } \sim n) \ p)$

$\langle \text{proof} \rangle$

**lemma** *degree-higher-pderiv*:  $\text{Polynomial.degree } ((\text{pderiv } \sim n) \ p) = \text{Polynomial.degree } p - n$

**for**  $p :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}, \text{semiring-char-0}\}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *DERIV-pow2*:  $\text{DERIV } (\lambda x. x \wedge \text{Suc } n) \ x := \text{real } (\text{Suc } n) * (x \wedge n)$

$\langle \text{proof} \rangle$

**declare** *DERIV-pow2* [simp] *DERIV-pow* [simp]

**lemma** *DERIV-add-const*:  $\text{DERIV } f \ x := D \implies \text{DERIV } (\lambda x. a + f \ x :: 'a :: \text{real-normed-field}) \ x := D$

$\langle \text{proof} \rangle$

**lemma** *poly-DERIV* [simp]:  $\text{DERIV } (\lambda x. \text{poly } p \ x) \ x := \text{poly } (\text{pderiv } p) \ x$

$\langle \text{proof} \rangle$

**lemma** *poly-isCont*[simp]:

**fixes**  $x :: 'a :: \text{real-normed-field}$

**shows**  $\text{isCont } (\lambda x. \text{poly } p \ x) \ x$

$\langle \text{proof} \rangle$

**lemma** *tendsto-poly* [tendsto-intros]:  $(f \longrightarrow a) \ F \implies ((\lambda x. \text{poly } p \ (f \ x)) \longrightarrow \text{poly } p \ a) \ F$

**for**  $f :: - \Rightarrow 'a :: \text{real-normed-field}$

$\langle \text{proof} \rangle$



**lemma** *continuous-within-poly*: *continuous* (at  $z$  within  $s$ ) (*poly*  $p$ )  
**for**  $z :: 'a::\{\text{real-normed-field}\}$   
 $\langle \text{proof} \rangle$

**lemma** *continuous-poly* [*continuous-intros*]: *continuous*  $F f \implies \text{continuous } F (\lambda x. \text{poly } p (f x))$   
**for**  $f :: - \Rightarrow 'a::\text{real-normed-field}$   
 $\langle \text{proof} \rangle$

**lemma** *continuous-on-poly* [*continuous-intros*]:  
**fixes**  $p :: 'a :: \{\text{real-normed-field}\}$  *poly*  
**assumes** *continuous-on*  $A f$   
**shows** *continuous-on*  $A (\lambda x. \text{poly } p (f x))$   
 $\langle \text{proof} \rangle$

Consequences of the derivative theorem above.

**lemma** *poly-differentiable[simp]*:  $(\lambda x. \text{poly } p x)$  *differentiable* (at  $x$ )  
**for**  $x :: \text{real}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-IVT-pos*:  $a < b \implies \text{poly } p a < 0 \implies 0 < \text{poly } p b \implies \exists x. a < x \wedge x < b \wedge \text{poly } p x = 0$   
**for**  $a b :: \text{real}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-IVT-neg*:  $a < b \implies 0 < \text{poly } p a \implies \text{poly } p b < 0 \implies \exists x. a < x \wedge x < b \wedge \text{poly } p x = 0$   
**for**  $a b :: \text{real}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-IVT*:  $a < b \implies \text{poly } p a * \text{poly } p b < 0 \implies \exists x > a. x < b \wedge \text{poly } p x = 0$   
**for**  $p :: \text{real poly}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-MVT*:  $a < b \implies \exists x. a < x \wedge x < b \wedge \text{poly } p b - \text{poly } p a = (b - a) * \text{poly } (pderiv p) x$   
**for**  $a b :: \text{real}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-MVT'*:  
**fixes**  $a b :: \text{real}$   
**assumes**  $\{\min a b.. \max a b\} \subseteq A$   
**shows**  $\exists x \in A. \text{poly } p b - \text{poly } p a = (b - a) * \text{poly } (pderiv p) x$   
 $\langle \text{proof} \rangle$

**lemma** *poly-pinfty-gt-lc*:  
**fixes**  $p :: \text{real poly}$   
**assumes** *lead-coeff*  $p > 0$

**shows**  $\exists n. \forall x \geq n. \text{poly } p \ x \geq \text{lead-coeff } p$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-monic*:

**fixes**  $p \ q :: 'a :: \text{idom } \text{poly}$   
**assumes**  $\text{monic} : \text{lead-coeff } p = 1$  **and**  $p \ \text{dvd} \ (\text{smult } c \ q)$  **and**  $c \neq 0$   
**shows**  $p \ \text{dvd} \ q$   $\langle \text{proof} \rangle$

**lemma** *lemma-order-pderiv1*:

$\text{pderiv } ([:- a, 1:] \wedge \text{Suc } n * q)$   
 $= [:- a, 1:] \wedge \text{Suc } n * \text{pderiv } q + \text{smult } (\text{of-nat } (\text{Suc } n)) (q * [:- a, 1:] \wedge n)$   
 $\langle \text{proof} \rangle$

**lemma** *order-pderiv*:

**fixes**  $p :: 'a :: \{\text{idom}, \text{semiring-char-0}\} \text{poly}$   
**assumes**  $p \neq 0$   $\text{poly } p \ x = 0$   
**shows**  $\text{order } x \ p = \text{Suc } (\text{order } x \ (\text{pderiv } p))$   $\langle \text{proof} \rangle$

**lemma** *lemma-order-pderiv*:

**fixes**  $p :: 'a :: \text{field-char-0 } \text{poly}$   
**assumes**  $n : 0 < n$   
**and**  $\text{pd} : \text{pderiv } p \neq 0$   
**and**  $\text{pe} : p = [:- a, 1:] \wedge n * q$   
**and**  $\text{nd} : \neg [:- a, 1:] \ \text{dvd} \ q$   
**shows**  $n = \text{Suc } (\text{order } a \ (\text{pderiv } p))$   
 $\langle \text{proof} \rangle$

**lemma** *poly-squarefree-decomp-order*:

**fixes**  $p :: 'a :: \text{field-char-0 } \text{poly}$   
**assumes**  $\text{pderiv } p \neq 0$   
**and**  $p : p = q * d$   
**and**  $p' : \text{pderiv } p = e * d$   
**and**  $d : d = r * p + s * \text{pderiv } p$   
**shows**  $\text{order } a \ q = (\text{if } \text{order } a \ p = 0 \text{ then } 0 \text{ else } 1)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-squarefree-decomp-order2*:

$\text{pderiv } p \neq 0 \implies p = q * d \implies \text{pderiv } p = e * d \implies$   
 $d = r * p + s * \text{pderiv } p \implies \forall a. \text{order } a \ q = (\text{if } \text{order } a \ p = 0 \text{ then } 0 \text{ else } 1)$   
**for**  $p :: 'a :: \text{field-char-0 } \text{poly}$   
 $\langle \text{proof} \rangle$

**lemma** *order-pderiv2*:

$\text{pderiv } p \neq 0 \implies \text{order } a \ p \neq 0 \implies \text{order } a \ (\text{pderiv } p) = n \longleftrightarrow \text{order } a \ p = \text{Suc } n$   
**for**  $p :: 'a :: \text{field-char-0 } \text{poly}$   
 $\langle \text{proof} \rangle$

**definition** *rsquarefree* ::  $'a :: \text{idom } \text{poly} \Rightarrow \text{bool}$

**where**  $rsquarefree\ p \longleftrightarrow p \neq 0 \wedge (\forall a. order\ a\ p = 0 \vee order\ a\ p = 1)$

**lemma** *pderiv-iszero*:  $pderiv\ p = 0 \implies \exists h. p = [:h:]$   
**for**  $p :: 'a::\{semidom, semiring-char-0\}$  *poly*  
 $\langle proof \rangle$

**lemma** *rsquarefree-roots*:  $rsquarefree\ p \longleftrightarrow (\forall a. \neg (poly\ p\ a = 0 \wedge poly\ (pderiv\ p)\ a = 0))$   
**for**  $p :: 'a::field-char-0$  *poly*  
 $\langle proof \rangle$

**lemma** *rsquarefree-root-order*:  
**assumes**  $rsquarefree\ p$   $poly\ p\ z = 0$   $p \neq 0$   
**shows**  $order\ z\ p = 1$   
 $\langle proof \rangle$

**lemma** *poly-squarefree-decomp*:  
**fixes**  $p :: 'a::field-char-0$  *poly*  
**assumes**  $pderiv\ p \neq 0$   
**and**  $p = q * d$   
**and**  $pderiv\ p = e * d$   
**and**  $d = r * p + s * pderiv\ p$   
**shows**  $rsquarefree\ q \wedge (\forall a. poly\ q\ a = 0 \longleftrightarrow poly\ p\ a = 0)$   
 $\langle proof \rangle$

**lemma** *has-field-derivative-poly* [*derivative-intros*]:  
**assumes**  $(f\ has-field-derivative\ f')$   $(at\ x\ within\ A)$   
**shows**  $((\lambda x. poly\ p\ (f\ x))\ has-field-derivative\ (f' * poly\ (pderiv\ p)\ (f\ x)))\ (at\ x\ within\ A)$   
 $\langle proof \rangle$

## 4.28 Algebraic numbers

**lemma** *intpolyE*:  
**assumes**  $\bigwedge i. poly.coeff\ p\ i \in \mathbb{Z}$   
**obtains**  $q$  **where**  $p = map-poly\ of-int\ q$   
 $\langle proof \rangle$

**lemma** *ratpolyE*:  
**assumes**  $\bigwedge i. poly.coeff\ p\ i \in \mathbb{Q}$   
**obtains**  $q$  **where**  $p = map-poly\ of-rat\ q$   
 $\langle proof \rangle$

Algebraic numbers can be defined in two equivalent ways: all real numbers that are roots of rational polynomials or of integer polynomials. The Algebraic-Numbers AFP entry uses the rational definition, but we need the integer definition.

The equivalence is obvious since any rational polynomial can be multiplied with the LCM of its coefficients, yielding an integer polynomial with the

same roots.

**definition** *algebraic* :: 'a :: field-char-0  $\Rightarrow$  bool  
**where** *algebraic*  $x \longleftrightarrow (\exists p. (\forall i. \text{coeff } p \ i \in \mathbb{Z}) \wedge p \neq 0 \wedge \text{poly } p \ x = 0)$

**lemma** *algebraicI*:  $(\bigwedge i. \text{coeff } p \ i \in \mathbb{Z}) \Rightarrow p \neq 0 \Rightarrow \text{poly } p \ x = 0 \Rightarrow \text{algebraic } x$   
 $\langle \text{proof} \rangle$

**lemma** *algebraicE*:  
**assumes** *algebraic*  $x$   
**obtains**  $p$  **where**  $\bigwedge i. \text{coeff } p \ i \in \mathbb{Z} \ p \neq 0 \ \text{poly } p \ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *algebraic-altdef*: *algebraic*  $x \longleftrightarrow (\exists p. (\forall i. \text{coeff } p \ i \in \mathbb{Q}) \wedge p \neq 0 \wedge \text{poly } p \ x = 0)$   
**for**  $p :: 'a :: \text{field-char-0}$  *poly*  
 $\langle \text{proof} \rangle$

**lemma** *algebraicI'*:  $(\bigwedge i. \text{coeff } p \ i \in \mathbb{Q}) \Rightarrow p \neq 0 \Rightarrow \text{poly } p \ x = 0 \Rightarrow \text{algebraic } x$   
 $\langle \text{proof} \rangle$

**lemma** *algebraicE'*:  
**assumes** *algebraic*  $(x :: 'a :: \text{field-char-0})$   
**obtains**  $p$  **where**  $p \neq 0 \ \text{poly } (\text{map-poly of-int } p) \ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *algebraicE'-nonzero*:  
**assumes** *algebraic*  $(x :: 'a :: \text{field-char-0}) \ x \neq 0$   
**obtains**  $p$  **where**  $p \neq 0 \ \text{coeff } p \ 0 \neq 0 \ \text{poly } (\text{map-poly of-int } p) \ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *rat-imp-algebraic*:  $x \in \mathbb{Q} \Rightarrow \text{algebraic } x$   
 $\langle \text{proof} \rangle$

**lemma** *algebraic-0* [*simp, intro*]: *algebraic* 0  
**and** *algebraic-1* [*simp, intro*]: *algebraic* 1  
**and** *algebraic-numeral* [*simp, intro*]: *algebraic* (numeral  $n$ )  
**and** *algebraic-of-nat* [*simp, intro*]: *algebraic* (of-nat  $k$ )  
**and** *algebraic-of-int* [*simp, intro*]: *algebraic* (of-int  $m$ )  
 $\langle \text{proof} \rangle$

**lemma** *algebraic-ii* [*simp, intro*]: *algebraic*  $i$   
 $\langle \text{proof} \rangle$

**lemma** *algebraic-minus* [*intro*]:  
**assumes** *algebraic*  $x$   
**shows** *algebraic*  $(-x)$   
 $\langle \text{proof} \rangle$

**lemma** *algebraic-minus-iff* [simp]:  
 $\text{algebraic } (-x) \longleftrightarrow \text{algebraic } (x :: 'a :: \text{field-char-0})$   
 ⟨proof⟩

**lemma** *algebraic-inverse* [intro]:  
 assumes *algebraic* *x*  
 shows *algebraic* (*inverse* *x*)  
 ⟨proof⟩

**lemma** *algebraic-root*:  
 assumes *algebraic* *y*  
 and *poly* *p* *x* = *y* and  $\forall i. \text{coeff } p \ i \in \mathbb{Z}$  and *lead-coeff* *p* = 1 and *degree* *p*  
 > 0  
 shows *algebraic* *x*  
 ⟨proof⟩

**lemma** *algebraic-abs-real* [simp]:  
 $\text{algebraic } |x :: \text{real}| \longleftrightarrow \text{algebraic } x$   
 ⟨proof⟩

**lemma** *algebraic-nth-root-real* [intro]:  
 assumes *algebraic* *x*  
 shows *algebraic* (*root* *n* *x*)  
 ⟨proof⟩

**lemma** *algebraic-sqrt* [intro]: *algebraic* *x*  $\implies$  *algebraic* (*sqrt* *x*)  
 ⟨proof⟩

**lemma** *algebraic-csqrt* [intro]: *algebraic* *x*  $\implies$  *algebraic* (*csqrt* *x*)  
 ⟨proof⟩

**lemma** *algebraic-cnj* [intro]:  
 assumes *algebraic* *x*  
 shows *algebraic* (*cnj* *x*)  
 ⟨proof⟩

**lemma** *algebraic-cnj-iff* [simp]: *algebraic* (*cnj* *x*)  $\longleftrightarrow$  *algebraic* *x*  
 ⟨proof⟩

**lemma** *algebraic-of-real* [intro]:  
 assumes *algebraic* *x*  
 shows *algebraic* (*of-real* *x*)  
 ⟨proof⟩

**lemma** *algebraic-of-real-iff* [simp]:  
 $\text{algebraic } (\text{of-real } x :: 'a :: \{\text{real-algebra-1}, \text{field-char-0}\}) \longleftrightarrow \text{algebraic } x$   
 ⟨proof⟩

## 4.29 Algebraic integers

**inductive** *algebraic-int* :: 'a :: field  $\Rightarrow$  bool **where**  
 $\llbracket \text{lead-coeff } p = 1; \forall i. \text{coeff } p \ i \in \mathbb{Z}; \text{poly } p \ x = 0 \rrbracket \Longrightarrow \text{algebraic-int } x$

**lemma** *algebraic-int-altdef-ipoly*:

**fixes**  $x :: 'a :: \text{field-char-0}$

**shows**  $\text{algebraic-int } x \longleftrightarrow (\exists p. \text{poly } (\text{map-poly of-int } p) \ x = 0 \wedge \text{lead-coeff } p = 1)$

*<proof>*

**theorem** *rational-algebraic-int-is-int*:

**assumes** *algebraic-int*  $x$  **and**  $x \in \mathbb{Q}$

**shows**  $x \in \mathbb{Z}$

*<proof>*

**lemma** *algebraic-int-imp-algebraic* [*dest*]: *algebraic-int*  $x \Longrightarrow \text{algebraic } x$

*<proof>*

**lemma** *int-imp-algebraic-int*:

**assumes**  $x \in \mathbb{Z}$

**shows** *algebraic-int*  $x$

*<proof>*

**lemma** *algebraic-int-0* [*simp, intro*]: *algebraic-int* 0

**and** *algebraic-int-1* [*simp, intro*]: *algebraic-int* 1

**and** *algebraic-int-numeral* [*simp, intro*]: *algebraic-int* (numeral  $n$ )

**and** *algebraic-int-of-nat* [*simp, intro*]: *algebraic-int* (of-nat  $k$ )

**and** *algebraic-int-of-int* [*simp, intro*]: *algebraic-int* (of-int  $m$ )

*<proof>*

**lemma** *algebraic-int-ii* [*simp, intro*]: *algebraic-int*  $i$

*<proof>*

**lemma** *algebraic-int-minus* [*intro*]:

**assumes** *algebraic-int*  $x$

**shows** *algebraic-int*  $(-x)$

*<proof>*

**lemma** *algebraic-int-minus-iff* [*simp*]:

$\text{algebraic-int } (-x) \longleftrightarrow \text{algebraic-int } (x :: 'a :: \text{field-char-0})$

*<proof>*

**lemma** *algebraic-int-inverse* [*intro*]:

**assumes**  $\text{poly } p \ x = 0$  **and**  $\forall i. \text{coeff } p \ i \in \mathbb{Z}$  **and**  $\text{coeff } p \ 0 = 1$

**shows** *algebraic-int* (inverse  $x$ )

*<proof>*

**lemma** *algebraic-int-root*:

**assumes** *algebraic-int*  $y$

**and**  $\text{poly } p \ x = y$  **and**  $\forall i. \text{coeff } p \ i \in \mathbb{Z}$  **and**  $\text{lead-coeff } p = 1$  **and**  $\text{degree } p > 0$   
**shows**  $\text{algebraic-int } x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-abs-real}$  [simp]:  
 $\text{algebraic-int } |x :: \text{real}| \longleftrightarrow \text{algebraic-int } x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-nth-root-real}$  [intro]:  
**assumes**  $\text{algebraic-int } x$   
**shows**  $\text{algebraic-int } (\text{root } n \ x)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-sqrt}$  [intro]:  $\text{algebraic-int } x \implies \text{algebraic-int } (\text{sqrt } x)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-csqrt}$  [intro]:  $\text{algebraic-int } x \implies \text{algebraic-int } (\text{csqrt } x)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-cnj}$  [intro]:  
**assumes**  $\text{algebraic-int } x$   
**shows**  $\text{algebraic-int } (\text{cnj } x)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-cnj-iff}$  [simp]:  $\text{algebraic-int } (\text{cnj } x) \longleftrightarrow \text{algebraic-int } x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-of-real}$  [intro]:  
**assumes**  $\text{algebraic-int } x$   
**shows**  $\text{algebraic-int } (\text{of-real } x)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{algebraic-int-of-real-iff}$  [simp]:  
 $\text{algebraic-int } (\text{of-real } x :: 'a :: \{\text{field-char-0}, \text{real-algebra-1}\}) \longleftrightarrow \text{algebraic-int } x$   
 $\langle \text{proof} \rangle$

## 4.30 Division of polynomials

### 4.30.1 Division in general

**instantiation**  $\text{poly} :: (\text{idom-divide}) \text{idom-divide}$   
**begin**

**fun**  $\text{divide-poly-main} :: 'a \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ poly}$   
**where**  
 $\text{divide-poly-main } lc \ q \ r \ d \ dr \ (\text{Suc } n) =$   
 $(\text{let } cr = \text{coeff } r \ dr; a = cr \ \text{div } lc; \text{mon} = \text{monom } a \ n \ \text{in}$   
 $\text{if } \text{False} \vee a * lc = cr \text{ then } \text{--- False} \vee \text{is only because of problem in}$   
 function-package

```

      divide-poly-main
      lc
      (q + mon)
      (r - mon * d)
      d (dr - 1) n else 0)
| divide-poly-main lc q r d dr 0 = q

```

**definition** *divide-poly* :: 'a poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly  
**where** *divide-poly* f g =  
 (if g = 0 then 0  
 else  
 divide-poly-main (coeff g (degree g)) 0 f g (degree f)  
 (1 + length (coeffs f) - length (coeffs g)))

**lemma** *divide-poly-main*:  
**assumes** d: d  $\neq$  0 lc = coeff d (degree d)  
**and** degree (d \* r)  $\leq$  dr divide-poly-main lc q (d \* r) d dr n = q'  
**and** n = 1 + dr - degree d  $\vee$  dr = 0  $\wedge$  n = 0  $\wedge$  d \* r = 0  
**shows** q' = q + r  
 <proof>

**lemma** *divide-poly-main-0*: divide-poly-main 0 0 r d dr n = 0  
 <proof>

**lemma** *divide-poly*:  
**assumes** g: g  $\neq$  0  
**shows** (f \* g) div g = (f :: 'a poly)  
 <proof>

**lemma** *divide-poly-0*: f div 0 = 0  
**for** f :: 'a poly  
 <proof>

**instance**  
 <proof>

**end**

**instance** poly :: (idom-divide) algebraic-semidom <proof>

**lemma** *div-const-poly-conv-map-poly*:  
**assumes** [:c:] dvd p  
**shows** p div [:c:] = map-poly ( $\lambda x. x \text{ div } c$ ) p  
 <proof>

**lemma** *is-unit-monom-0*:  
**fixes** a :: 'a::field  
**assumes** a  $\neq$  0  
**shows** is-unit (monom a 0)



$\langle proof \rangle$

**lemma** *is-unit-triv*:  $a \neq 0 \implies is\_unit\ [a:]$   
  **for**  $a :: 'a::field$   
   $\langle proof \rangle$

**lemma** *is-unit-iff-degree*:  
  **fixes**  $p :: 'a::field\ poly$   
  **assumes**  $p \neq 0$   
  **shows**  $is\_unit\ p \longleftrightarrow degree\ p = 0$   
    **(is ?lhs  $\longleftrightarrow$  ?rhs)**  
   $\langle proof \rangle$

**lemma** *is-unit-pCons-iff*:  $is\_unit\ (pCons\ a\ p) \longleftrightarrow p = 0 \wedge a \neq 0$   
  **for**  $p :: 'a::field\ poly$   
   $\langle proof \rangle$

**lemma** *is-unit-monom-trivial*:  $is\_unit\ p \implies monom\ (coeff\ p\ (degree\ p))\ 0 = p$   
  **for**  $p :: 'a::field\ poly$   
   $\langle proof \rangle$

**lemma** *is-unit-const-poly-iff*:  $[c:]\ dvd\ 1 \longleftrightarrow c\ dvd\ 1$   
  **for**  $c :: 'a::\{comm\_semiring-1, semiring-no-zero-divisors\}$   
   $\langle proof \rangle$

**lemma** *is-unit-polyE*:  
  **fixes**  $p :: 'a :: \{comm\_semiring-1, semiring-no-zero-divisors\}\ poly$   
  **assumes**  $p\ dvd\ 1$   
  **obtains**  $c$  **where**  $p = [c:]\ c\ dvd\ 1$   
   $\langle proof \rangle$

**lemma** *is-unit-polyE'*:  
  **fixes**  $p :: 'a::field\ poly$   
  **assumes**  $is\_unit\ p$   
  **obtains**  $a$  **where**  $p = monom\ a\ 0$  **and**  $a \neq 0$   
   $\langle proof \rangle$

**lemma** *is-unit-poly-iff*:  $p\ dvd\ 1 \longleftrightarrow (\exists c. p = [c:] \wedge c\ dvd\ 1)$   
  **for**  $p :: 'a::\{comm\_semiring-1, semiring-no-zero-divisors\}\ poly$   
   $\langle proof \rangle$

**lemma** *coprime-poly-0*:  
   $poly\ p\ x \neq 0 \vee poly\ q\ x \neq 0$  **if**  $coprime\ p\ q$   
  **for**  $x :: 'a :: field$   
   $\langle proof \rangle$

**lemma** *root-imp-reducible-poly*:  
  **fixes**  $x :: 'a :: field$   
  **assumes**  $poly\ p\ x = 0$  **and**  $degree\ p > 1$

**shows**  $\neg \text{irreducible } p$   
 $\langle \text{proof} \rangle$

**lemma** *reducible-polyI*:  
**fixes**  $p :: 'a :: \text{field poly}$   
**assumes**  $p = q * r \text{ degree } q > 0 \text{ degree } r > 0$   
**shows**  $\neg \text{irreducible } p$   
 $\langle \text{proof} \rangle$

#### 4.30.2 Pseudo-Division

This part is by René Thiemann and Akihisa Yamada.

**fun** *pseudo-divmod-main* ::  
 $'a :: \text{comm-ring-1} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ poly} \times 'a \text{ poly}$   
**where**  
 $\text{pseudo-divmod-main } lc \ q \ r \ d \ dr \ (\text{Suc } n) =$   
 $(\text{let}$   
 $\quad rr = \text{smult } lc \ r;$   
 $\quad qq = \text{coeff } r \ dr;$   
 $\quad rrr = rr - \text{monom } qq \ n * d;$   
 $\quad qqq = \text{smult } lc \ q + \text{monom } qq \ n$   
 $\quad \text{in } \text{pseudo-divmod-main } lc \ qqq \ rrr \ d \ (dr - 1) \ n)$   
 $| \text{pseudo-divmod-main } lc \ q \ r \ d \ dr \ 0 = (q, r)$

**definition** *pseudo-divmod* ::  $'a :: \text{comm-ring-1} \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly} \times 'a \text{ poly}$   
**where**  $\text{pseudo-divmod } p \ q \equiv$   
 $\text{if } q = 0 \text{ then } (0, p)$   
 $\text{else}$   
 $\text{pseudo-divmod-main } (\text{coeff } q \ (\text{degree } q)) \ 0 \ p \ q \ (\text{degree } p)$   
 $(1 + \text{length } (\text{coeffs } p) - \text{length } (\text{coeffs } q))$

**lemma** *pseudo-divmod-main*:  
**assumes**  $d: d \neq 0 \ lc = \text{coeff } d \ (\text{degree } d)$   
**and**  $\text{degree } r \leq dr \ \text{pseudo-divmod-main } lc \ q \ r \ d \ dr \ n = (q', r')$   
**and**  $n = 1 + dr - \text{degree } d \vee dr = 0 \wedge n = 0 \wedge r = 0$   
**shows**  $(r' = 0 \vee \text{degree } r' < \text{degree } d) \wedge \text{smult } (lc \hat{\ } n) \ (d * q + r) = d * q' + r'$   
 $\langle \text{proof} \rangle$

**lemma** *pseudo-divmod*:  
**assumes**  $g: g \neq 0$   
**and**  $*: \text{pseudo-divmod } f \ g = (q, r)$   
**shows**  $\text{smult } (\text{coeff } g \ (\text{degree } g)) \ (\text{Suc } (\text{degree } f) - \text{degree } g)) \ f = g * q + r$  (is ?A)  
**and**  $r = 0 \vee \text{degree } r < \text{degree } g$  (is ?B)  
 $\langle \text{proof} \rangle$

**definition** *pseudo-mod-main*  $lc \ r \ d \ dr \ n = \text{snd } (\text{pseudo-divmod-main } lc \ 0 \ r \ d \ dr \ n)$

**lemma** *snd-pseudo-divmod-main*:

*snd (pseudo-divmod-main lc q r d dr n) = snd (pseudo-divmod-main lc q' r d dr n)*  
*<proof>*

**definition** *pseudo-mod* :: 'a::{comm-ring-1,semiring-1-no-zero-divisors} poly  $\Rightarrow$  'a poly  
*poly  $\Rightarrow$  'a poly*  
*where pseudo-mod f g = snd (pseudo-divmod f g)*

**lemma** *pseudo-mod*:

*fixes f g :: 'a::{comm-ring-1,semiring-1-no-zero-divisors} poly*  
*defines r  $\equiv$  pseudo-mod f g*  
*assumes g: g  $\neq$  0*  
*shows  $\exists a q. a \neq 0 \wedge \text{smult } a f = g * q + r \wedge r = 0 \vee \text{degree } r < \text{degree } g$*   
*<proof>*

**lemma** *fst-pseudo-divmod-main-as-divide-poly-main*:

*assumes d: d  $\neq$  0*  
*defines lc: lc  $\equiv$  coeff d (degree d)*  
*shows fst (pseudo-divmod-main lc q r d dr n) =*  
*divide-poly-main lc (smult (lc<sup>n</sup>) q) (smult (lc<sup>n</sup>) r) d dr n*  
*<proof>*

### 4.30.3 Division in polynomials over fields

**lemma** *pseudo-divmod-field*:

*fixes g :: 'a::field poly*  
*assumes g: g  $\neq$  0*  
*and \*: pseudo-divmod f g = (q,r)*  
*defines c  $\equiv$  coeff g (degree g)  $\wedge$  (Suc (degree f) - degree g)*  
*shows f = g \* smult (1/c) q + smult (1/c) r*  
*<proof>*

**lemma** *divide-poly-main-field*:

*fixes d :: 'a::field poly*  
*assumes d: d  $\neq$  0*  
*defines lc: lc  $\equiv$  coeff d (degree d)*  
*shows divide-poly-main lc q r d dr n =*  
*fst (pseudo-divmod-main lc (smult ((1 / lc)<sup>n</sup>) q) (smult ((1 / lc)<sup>n</sup>) r) d dr n)*  
*<proof>*

**lemma** *divide-poly-field*:

*fixes f g :: 'a::field poly*  
*defines f'  $\equiv$  smult ((1 / coeff g (degree g))  $\wedge$  (Suc (degree f) - degree g)) f*  
*shows f div g = fst (pseudo-divmod f' g)*  
*<proof>*

**instantiation** *poly* :: (*{ semidom-divide-unit-factor, idom-divide }*) *normalization-semidom*  
**begin**

**definition** *unit-factor-poly* :: 'a *poly*  $\Rightarrow$  'a *poly*  
**where** *unit-factor-poly* *p* = [:*unit-factor* (*lead-coeff* *p*):]

**definition** *normalize-poly* :: 'a *poly*  $\Rightarrow$  'a *poly*  
**where** *normalize* *p* = *p* *div* [:*unit-factor* (*lead-coeff* *p*):]

**instance**  
*<proof>*

**end**

**instance** *poly* :: (*{ semidom-divide-unit-factor, idom-divide, normalization-semidom-multiplicative }*)  
*normalization-semidom-multiplicative*  
*<proof>*

**lemma** *normalize-poly-eq-map-poly*: *normalize* *p* = *map-poly* ( $\lambda x. x \text{ div } \text{unit-factor}$   
(*lead-coeff* *p*)) *p*  
*<proof>*

**lemma** *coeff-normalize* [*simp*]:  
*coeff* (*normalize* *p*) *n* = *coeff* *p* *n* *div* *unit-factor* (*lead-coeff* *p*)  
*<proof>*

**class** *field-unit-factor* = *field* + *unit-factor* +  
**assumes** *unit-factor-field* [*simp*]: *unit-factor* = *id*  
**begin**

**subclass** *semidom-divide-unit-factor*  
*<proof>*

**end**

**lemma** *unit-factor-pCons*:  
*unit-factor* (*pCons* *a* *p*) = (if *p* = 0 then [:*unit-factor* *a*:] else *unit-factor* *p*)  
*<proof>*

**lemma** *normalize-monom* [*simp*]: *normalize* (*monom* *a* *n*) = *monom* (*normalize*  
*a*) *n*  
*<proof>*

**lemma** *unit-factor-monom* [*simp*]: *unit-factor* (*monom* *a* *n*) = [:*unit-factor* *a*:]  
*<proof>*

**lemma** *normalize-const-poly*: *normalize* [:*c*:] = [:*normalize* *c*:]  
*<proof>*

```

lemma normalize-smult:
  fixes  $c :: 'a :: \{normalization-semidom-multiplicative, idom-divide\}$ 
  shows  $normalize (smult\ c\ p) = smult (normalize\ c) (normalize\ p)$ 
   $\langle proof \rangle$ 

instantiation poly :: (field) idom-modulo
begin

definition modulo-poly :: 'a poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly
  where mod-poly-def:  $f\ mod\ g =$ 
    (if  $g = 0$  then  $f$  else  $pseudo-mod (smult ((1 / lead-coeff\ g) ^ (Suc (degree\ f) -$ 
     $degree\ g))\ f)\ g$ )

instance
   $\langle proof \rangle$ 

end

lemma pseudo-divmod-eq-div-mod:
   $\langle pseudo-divmod\ f\ g = (f\ div\ g, f\ mod\ g) \rangle$  if  $\langle lead-coeff\ g = 1 \rangle$ 
   $\langle proof \rangle$ 

lemma degree-mod-less-degree:
   $\langle degree\ (x\ mod\ y) < degree\ y \rangle$  if  $\langle y \neq 0 \rangle$   $\langle \neg\ y\ dvd\ x \rangle$ 
   $\langle proof \rangle$ 

instantiation poly :: (field) unique-euclidean-ring
begin

definition euclidean-size-poly :: 'a poly  $\Rightarrow$  nat
  where euclidean-size-poly  $p = (if\ p = 0\ then\ 0\ else\ 2 ^ degree\ p)$ 

definition division-segment-poly :: 'a poly  $\Rightarrow$  'a poly
  where [simp]: division-segment-poly  $p = 1$ 

instance  $\langle proof \rangle$ 

end

lemma euclidean-relation-polyI [case-names by0 divides euclidean-relation]:
   $\langle (x\ div\ y, x\ mod\ y) = (q, r) \rangle$ 
  if by0:  $\langle y = 0 \implies q = 0 \wedge r = x \rangle$ 
  and divides:  $\langle y \neq 0 \implies y\ dvd\ x \implies r = 0 \wedge x = q * y \rangle$ 
  and euclidean-relation:  $\langle y \neq 0 \implies \neg\ y\ dvd\ x \implies degree\ r < degree\ y \wedge x = q$ 
   $*\ y + r \rangle$ 
   $\langle proof \rangle$ 

lemma div-poly-eq-0-iff:
   $\langle x\ div\ y = 0 \iff x = 0 \vee y = 0 \vee degree\ x < degree\ y \rangle$  for  $x\ y :: 'a::field\ poly$ 

```

$\langle \text{proof} \rangle$

**lemma** *div-poly-less*:

$\langle x \text{ div } y = 0 \rangle$  **if**  $\langle \text{degree } x < \text{degree } y \rangle$  **for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *mod-poly-less*:

$\langle x \text{ mod } y = x \rangle$  **if**  $\langle \text{degree } x < \text{degree } y \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *degree-div-less*:

$\langle \text{degree } (x \text{ div } y) < \text{degree } x \rangle$   
**if**  $\langle \text{degree } x > 0 \rangle \langle \text{degree } y > 0 \rangle$   
**for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *degree-mod-less'*:  $b \neq 0 \implies a \text{ mod } b \neq 0 \implies \text{degree } (a \text{ mod } b) < \text{degree } b$   
 $\langle \text{proof} \rangle$

**lemma** *degree-mod-less*:  $y \neq 0 \implies x \text{ mod } y = 0 \vee \text{degree } (x \text{ mod } y) < \text{degree } y$   
 $\langle \text{proof} \rangle$

**lemma** *div-smult-left*:  $\langle \text{smult } a \ x \text{ div } y = \text{smult } a \ (x \text{ div } y) \rangle$  (**is** ?Q)  
**and** *mod-smult-left*:  $\langle \text{smult } a \ x \text{ mod } y = \text{smult } a \ (x \text{ mod } y) \rangle$  (**is** ?R)  
**for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-div-minus-left [simp]*:  $(- \ x) \text{ div } y = - \ (x \text{ div } y)$   
**for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-mod-minus-left [simp]*:  $(- \ x) \text{ mod } y = - \ (x \text{ mod } y)$   
**for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-div-add-left*:  $\langle (x + y) \text{ div } z = x \text{ div } z + y \text{ div } z \rangle$  (**is** ?Q)  
**and** *poly-mod-add-left*:  $\langle (x + y) \text{ mod } z = x \text{ mod } z + y \text{ mod } z \rangle$  (**is** ?R)  
**for**  $x \ y \ z :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-div-diff-left*:  $(x - y) \text{ div } z = x \text{ div } z - y \text{ div } z$   
**for**  $x \ y \ z :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-mod-diff-left*:  $(x - y) \text{ mod } z = x \text{ mod } z - y \text{ mod } z$   
**for**  $x \ y \ z :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *div-smult-right*:  $\langle x \text{ div } \text{smult } a \ y = \text{smult } (\text{inverse } a) \ (x \text{ div } y) \rangle$  (**is** ?Q)

**and** *mod-smult-right*:  $\langle x \text{ mod } \text{smult } a \ y = (\text{if } a = 0 \text{ then } x \text{ else } x \text{ mod } y) \rangle$  (**is** ?R)  
 $\langle \text{proof} \rangle$

**lemma** *mod-mult-unit-eq*:  
 $\langle x \text{ mod } (z * y) = x \text{ mod } y \rangle$   
**if**  $\langle \text{is-unit } z \rangle$   
**for**  $x \ y \ z :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *poly-div-minus-right* [*simp*]:  $x \text{ div } (- y) = - (x \text{ div } y)$   
**for**  $x \ y :: 'a::\text{field poly}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-mod-minus-right* [*simp*]:  $x \text{ mod } (- y) = x \text{ mod } y$   
**for**  $x \ y :: 'a::\text{field poly}$   
 $\langle \text{proof} \rangle$

**lemma** *poly-div-mult-right*:  $\langle x \text{ div } (y * z) = (x \text{ div } y) \text{ div } z \rangle$  (**is** ?Q)  
**and** *poly-mod-mult-right*:  $\langle x \text{ mod } (y * z) = y * (x \text{ div } y \text{ mod } z) + x \text{ mod } y \rangle$  (**is** ?R)  
**for**  $x \ y \ z :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-pCons-imp-dvd-pCons-mod*:  
 $\langle y \text{ dvd } p\text{Cons } a \ (x \text{ mod } y) \rangle$  **if**  $\langle y \text{ dvd } p\text{Cons } a \ x \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *degree-less-if-less-eqI*:  
 $\langle \text{degree } x < \text{degree } y \rangle$  **if**  $\langle \text{degree } x \leq \text{degree } y \rangle \langle \text{coeff } x \ (\text{degree } y) = 0 \rangle \langle x \neq 0 \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *div-pCons-eq*:  
 $\langle p\text{Cons } a \ p \text{ div } q = (\text{if } q = 0 \text{ then } 0 \text{ else } p\text{Cons } (\text{coeff } (p\text{Cons } a \ (p \text{ mod } q))$   
 $(\text{degree } q) / \text{lead-coeff } q) \ (p \text{ div } q)) \rangle$  (**is** ?Q)  
**and** *mod-pCons-eq*:  
 $\langle p\text{Cons } a \ p \text{ mod } q = (\text{if } q = 0 \text{ then } p\text{Cons } a \ p \text{ else } p\text{Cons } a \ (p \text{ mod } q) - \text{smult}$   
 $(\text{coeff } (p\text{Cons } a \ (p \text{ mod } q)) \ (\text{degree } q) / \text{lead-coeff } q) \ q) \rangle$  (**is** ?R)  
**for**  $x \ y :: \langle 'a::\text{field poly} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *div-mod-fold-coeffs*:  
 $(p \text{ div } q, p \text{ mod } q) =$   
 $(\text{if } q = 0 \text{ then } (0, p)$   
 $\text{else}$   
 $\text{fold-coeffs}$   
 $(\lambda a \ (s, r).$   
 $\text{let } b = \text{coeff } (p\text{Cons } a \ r) \ (\text{degree } q) / \text{coeff } q \ (\text{degree } q)$   
 $\text{in } (p\text{Cons } b \ s, p\text{Cons } a \ r - \text{smult } b \ q)) \ p \ (0, 0))$   
 $\langle \text{proof} \rangle$

```

lemma mod-pCons:
  fixes a :: 'a::field
    and x y :: 'a::field poly
  assumes y: y ≠ 0
  defines b ≡ coeff (pCons a (x mod y)) (degree y) / coeff y (degree y)
  shows (pCons a x) mod y = pCons a (x mod y) - smult b y
  ⟨proof⟩

```

#### 4.30.4 List-based versions for fast implementation

```

fun minus-poly-rev-list :: 'a :: group-add list ⇒ 'a list ⇒ 'a list
  where
    minus-poly-rev-list (x # xs) (y # ys) = (x - y) # (minus-poly-rev-list xs ys)
  | minus-poly-rev-list xs [] = xs
  | minus-poly-rev-list [] (y # ys) = []

```

```

fun pseudo-divmod-main-list ::
  'a::comm-ring-1 ⇒ 'a list ⇒ 'a list ⇒ 'a list ⇒ nat ⇒ 'a list × 'a list
  where
    pseudo-divmod-main-list lc q r d (Suc n) =
      (let
        rr = map ((* ) lc) r;
        a = hd r;
        qq = cCons a (map ((* ) lc) q);
        rrr = tl (if a = 0 then rr else minus-poly-rev-list rr (map ((* ) a) d))
      in pseudo-divmod-main-list lc qq rrr d n)
  | pseudo-divmod-main-list lc q r d 0 = (q, r)

```

```

fun pseudo-mod-main-list :: 'a::comm-ring-1 ⇒ 'a list ⇒ 'a list ⇒ nat ⇒ 'a list
  where
    pseudo-mod-main-list lc r d (Suc n) =
      (let
        rr = map ((* ) lc) r;
        a = hd r;
        rrr = tl (if a = 0 then rr else minus-poly-rev-list rr (map ((* ) a) d))
      in pseudo-mod-main-list lc rrr d n)
  | pseudo-mod-main-list lc r d 0 = r

```

```

fun divmod-poly-one-main-list ::
  'a::comm-ring-1 list ⇒ 'a list ⇒ 'a list ⇒ nat ⇒ 'a list × 'a list
  where
    divmod-poly-one-main-list q r d (Suc n) =
      (let
        a = hd r;
        qq = cCons a q;
        rr = tl (if a = 0 then r else minus-poly-rev-list r (map ((* ) a) d))
      in divmod-poly-one-main-list qq rr d n)

```



```

| divmod-poly-one-main-list q r d 0 = (q, r)

fun mod-poly-one-main-list :: 'a::comm-ring-1 list  $\Rightarrow$  'a list  $\Rightarrow$  nat  $\Rightarrow$  'a list
where
  mod-poly-one-main-list r d (Suc n) =
    (let
      a = hd r;
      rr = tl (if a = 0 then r else minus-poly-rev-list r (map ((*) a) d))
    in mod-poly-one-main-list rr d n)
| mod-poly-one-main-list r d 0 = r

definition pseudo-divmod-list :: 'a::comm-ring-1 list  $\Rightarrow$  'a list  $\Rightarrow$  'a list  $\times$  'a list
where pseudo-divmod-list p q =
  (if q = [] then ([], p)
   else
    (let rq = rev q;
      (qu, re) = pseudo-divmod-main-list (hd rq) [] (rev p) rq (1 + length p -
length q)
    in (qu, rev re)))

definition pseudo-mod-list :: 'a::comm-ring-1 list  $\Rightarrow$  'a list  $\Rightarrow$  'a list
where pseudo-mod-list p q =
  (if q = [] then p
   else
    (let
      rq = rev q;
      re = pseudo-mod-main-list (hd rq) (rev p) rq (1 + length p - length q)
    in rev re))

lemma minus-zero-does-nothing: minus-poly-rev-list x (map ((*) 0) y) = x
for x :: 'a::ring list
<proof>

lemma length-minus-poly-rev-list [simp]: length (minus-poly-rev-list xs ys) = length
xs
<proof>

lemma if-0-minus-poly-rev-list:
  (if a = 0 then x else minus-poly-rev-list x (map ((*) a) y)) =
    minus-poly-rev-list x (map ((*) a) y)
for a :: 'a::ring
<proof>

lemma Poly-append: Poly (a @ b) = Poly a + monom 1 (length a) * Poly b
for a :: 'a::comm-semiring-1 list
<proof>

lemma minus-poly-rev-list: length p  $\geq$  length q  $\implies$ 
  Poly (rev (minus-poly-rev-list (rev p) (rev q))) =

```

*Poly p* − *monom 1 (length p − length q) \* Poly q*  
**for** *p q* :: 'a :: comm-ring-1 list  
 ⟨proof⟩

**lemma** *smult-monom-mult*: *smult a (monom b n \* f) = monom (a \* b) n \* f*  
 ⟨proof⟩

**lemma** *head-minus-poly-rev-list*:  
*length d ≤ length r ⇒ d ≠ [] ⇒*  
*hd (minus-poly-rev-list (map ((\*) (last d)) r) (map ((\*) (hd r)) (rev d))) = 0*  
**for** *d r* :: 'a::comm-ring list  
 ⟨proof⟩

**lemma** *Poly-map*: *Poly (map ((\*) a) p) = smult a (Poly p)*  
 ⟨proof⟩

**lemma** *last-coeff-is-hd*: *xs ≠ [] ⇒ coeff (Poly xs) (length xs − 1) = hd (rev xs)*  
 ⟨proof⟩

**lemma** *pseudo-divmod-main-list-invar*:  
**assumes** *leading-nonzero*: *last d ≠ 0*  
**and** *lc*: *last d = lc*  
**and** *d ≠ []*  
**and** *pseudo-divmod-main-list lc q (rev r) (rev d) n = (q', rev r')*  
**and** *n = 1 + length r − length d*  
**shows** *pseudo-divmod-main lc (monom 1 n \* Poly q) (Poly r) (Poly d) (length r − 1) n =*  
*(Poly q', Poly r')*  
 ⟨proof⟩

**lemma** *pseudo-divmod-impl* [code]:  
*pseudo-divmod f g = map-prod poly-of-list poly-of-list (pseudo-divmod-list (coeffs f) (coeffs g))*  
**for** *f g* :: 'a::comm-ring-1 poly  
 ⟨proof⟩

**lemma** *pseudo-mod-main-list*:  
*snd (pseudo-divmod-main-list l q xs ys n) = pseudo-mod-main-list l xs ys n*  
 ⟨proof⟩

**lemma** *pseudo-mod-impl*[code]: *pseudo-mod f g = poly-of-list (pseudo-mod-list (coeffs f) (coeffs g))*  
 ⟨proof⟩

#### 4.30.5 Improved Code-Equations for Polynomial (Pseudo) Division

**lemma** *pdivmod-via-pseudo-divmod*:  
 ⟨*f div g, f mod g*⟩ =

```

    (if g = 0 then (0, f)
     else
      let
        ilc = inverse (lead-coeff g);
        h = smult ilc g;
        (q,r) = pseudo-divmod f h
      in (smult ilc q, r))
  (is ⟨?l = ?r⟩)
⟨proof⟩

```

**lemma** *pdivmod-via-pseudo-divmod-list*:

```

(f div g, f mod g) =
  (let cg = coeffs g in
   if cg = [] then (0, f)
   else
    let
      cf = coeffs f;
      ilc = inverse (last cg);
      ch = map ((* ) ilc) cg;
      (q, r) = pseudo-divmod-main-list 1 [] (rev cf) (rev ch) (1 + length cf -
length cg)
    in (poly-of-list (map ((* ) ilc) q), poly-of-list (rev r)))
  )
⟨proof⟩

```

**lemma** *pseudo-divmod-main-list-1*: *pseudo-divmod-main-list 1 = divmod-poly-one-main-list*  
 ⟨proof⟩

**fun** *divide-poly-main-list* :: 'a::idom-divide ⇒ 'a list ⇒ 'a list ⇒ 'a list ⇒ nat ⇒ 'a list

**where**

```

  divide-poly-main-list lc q r d (Suc n) =
    (let
      cr = hd r
      in if cr = 0 then divide-poly-main-list lc (cCons cr q) (tl r) d n else let
        a = cr div lc;
        qq = cCons a q;
        rr = minus-poly-rev-list r (map ((* ) a) d)
        in if hd rr = 0 then divide-poly-main-list lc qq (tl rr) d n else [])
    | divide-poly-main-list lc q r d 0 = q

```

**lemma** *divide-poly-main-list-simp* [simp]:

```

  divide-poly-main-list lc q r d (Suc n) =
    (let
      cr = hd r;
      a = cr div lc;
      qq = cCons a q;
      rr = minus-poly-rev-list r (map ((* ) a) d)
      in if hd rr = 0 then divide-poly-main-list lc qq (tl rr) d n else [])
  )
⟨proof⟩

```

**declare** *divide-poly-main-list.simps*(1)[*simp del*]

**definition** *divide-poly-list* :: 'a::idom-divide poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly

**where** *divide-poly-list* f g =  
 (let cg = coeffs g in  
 if cg = [] then g  
 else  
 let  
 cf = coeffs f;  
 cgr = rev cg  
 in poly-of-list (divide-poly-main-list (hd cgr) [] (rev cf) cgr (1 + length cf  
 - length cg)))

**lemmas** *pdivmod-via-divmod-list* = *pdivmod-via-pseudo-divmod-list*[*unfolded pseudo-divmod-main-list-1*]

**lemma** *mod-poly-one-main-list*: *snd* (divmod-poly-one-main-list q r d n) = *mod-poly-one-main-list*  
 r d n  
 <proof>

**lemma** *mod-poly-code* [code]:

*f mod g* =  
 (let cg = coeffs g in  
 if cg = [] then f  
 else  
 let  
 cf = coeffs f;  
 ilc = inverse (last cg);  
 ch = map ((\*) ilc) cg;  
 r = mod-poly-one-main-list (rev cf) (rev ch) (1 + length cf - length cg)  
 in poly-of-list (rev r))  
 (is - = ?rhs)  
 <proof>

**definition** *div-field-poly-impl* :: 'a :: field poly  $\Rightarrow$  'a poly  $\Rightarrow$  'a poly

**where** *div-field-poly-impl* f g =  
 (let cg = coeffs g in  
 if cg = [] then 0  
 else  
 let  
 cf = coeffs f;  
 ilc = inverse (last cg);  
 ch = map ((\*) ilc) cg;  
 q = fst (divmod-poly-one-main-list [] (rev cf) (rev ch) (1 + length cf -  
 length cg))  
 in poly-of-list ((map ((\*) ilc) q)))

We do not declare the following lemma as code equation, since then polynomial division on non-fields will no longer be executable. However, a code-

unfold is possible, since *div-field-poly-impl* is a bit more efficient than the generic polynomial division.

**lemma** *div-field-poly-impl*[code-unfold]: (*div*) = *div-field-poly-impl*  
 ⟨proof⟩

**lemma** *divide-poly-main-list*:

**assumes** *lc0*: *lc* ≠ 0

**and** *lc*: last *d* = *lc*

**and** *d*: *d* ≠ []

**and** *n* = (1 + length *r* - length *d*)

**shows** Poly (*divide-poly-main-list* *lc* *q* (rev *r*) (rev *d*) *n*) =

*divide-poly-main* *lc* (monom 1 *n* \* Poly *q*) (Poly *r*) (Poly *d*) (length *r* - 1) *n*

⟨proof⟩

**lemma** *divide-poly-list*[code]: *f* div *g* = *divide-poly-list* *f* *g*

⟨proof⟩

**lemma** *poly-mod*:

*poly* (*p* mod *q*) *x* = *poly* *p* *x* if *poly* *q* *x* = 0

⟨proof⟩

### 4.31 Primality and irreducibility in polynomial rings

**lemma** *prod-mset-const-poly*: (∏ *x* ∈ #*A*. [:*f* *x*:]) = [:*prod-mset* (*image-mset* *f* *A*):]

⟨proof⟩

**lemma** *irreducible-const-poly-iff*:

**fixes** *c* :: 'a :: {comm-semiring-1, semiring-no-zero-divisors}

**shows** irreducible [:*c*:] ⟷ irreducible *c*

⟨proof⟩

**lemma** *lift-prime-elem-poly*:

**assumes** *prime-elem* (*c* :: 'a :: semidom)

**shows** *prime-elem* [:*c*:]

⟨proof⟩

**lemma** *prime-elem-const-poly-iff*:

**fixes** *c* :: 'a :: semidom

**shows** *prime-elem* [:*c*:] ⟷ *prime-elem* *c*

⟨proof⟩

### 4.32 Content and primitive part of a polynomial

**definition** *content* :: 'a::semiring-gcd poly ⇒ 'a

**where** *content* *p* = gcd-list (coeffs *p*)

**lemma** *content-eq-fold-coeffs* [code]: *content* *p* = fold-coeffs gcd *p* 0

⟨proof⟩

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

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

**lemma** *content-const* [simp]: *content [:c:] = normalize c*  
 ⟨proof⟩

**lemma** *const-poly-dvd-iff-dvd-content*: *[:c:] dvd p  $\longleftrightarrow$  c dvd content p*  
**for** *c :: 'a::semiring-gcd*  
 ⟨proof⟩

**lemma** *content-dvd* [simp]: *[:content p:] dvd p*  
 ⟨proof⟩

**lemma** *content-dvd-coeff* [simp]: *content p dvd coeff p n*  
 ⟨proof⟩

**lemma** *content-dvd-coeffs*: *c  $\in$  set (coeffs p)  $\implies$  content p dvd c*  
 ⟨proof⟩

**lemma** *normalize-content* [simp]: *normalize (content p) = content p*  
 ⟨proof⟩

**lemma** *is-unit-content-iff* [simp]: *is-unit (content p)  $\longleftrightarrow$  content p = 1*  
 ⟨proof⟩

**lemma** *content-smult* [simp]:  
**fixes** *c :: 'a :: {normalization-semidom-multiplicative, semiring-gcd}*  
**shows** *content (smult c p) = normalize c \* content p*  
 ⟨proof⟩

**lemma** *content-eq-zero-iff* [simp]: *content p = 0  $\longleftrightarrow$  p = 0*  
 ⟨proof⟩

**definition** *primitive-part* :: *'a :: semiring-gcd poly  $\Rightarrow$  'a poly*  
**where** *primitive-part p = map-poly ( $\lambda x. x \text{ div content } p$ ) p*

**lemma** *primitive-part-0* [simp]: *primitive-part 0 = 0*  
 ⟨proof⟩

**lemma** *content-times-primitive-part* [simp]: *smult (content p) (primitive-part p) = p*  
**for** *p :: 'a :: semiring-gcd poly*  
 ⟨proof⟩

**lemma** *primitive-part-eq-0-iff* [simp]: *primitive-part p = 0  $\longleftrightarrow$  p = 0*  
 ⟨proof⟩

```

lemma content-primitive-part [simp]:
  fixes  $p :: 'a :: \{\text{normalization-semidom-multiplicative, semiring-gcd}\}$  poly
  assumes  $p \neq 0$ 
  shows  $\text{content } (\text{primitive-part } p) = 1$ 
<proof>

lemma content-decompose:
  obtains  $p' :: 'a :: \{\text{normalization-semidom-multiplicative, semiring-gcd}\}$  poly
  where  $p = \text{smult } (\text{content } p) \ p'$   $\text{content } p' = 1$ 
<proof>

lemma content-dvd-contentI [intro]:  $p \text{ dvd } q \implies \text{content } p \text{ dvd } \text{content } q$ 
<proof>

lemma primitive-part-const-poly [simp]:  $\text{primitive-part } [:x:] = [: \text{unit-factor } x:]$ 
<proof>

lemma primitive-part-prim:  $\text{content } p = 1 \implies \text{primitive-part } p = p$ 
<proof>

lemma degree-primitive-part [simp]:  $\text{degree } (\text{primitive-part } p) = \text{degree } p$ 
<proof>

lemma smult-content-normalize-primitive-part [simp]:
  fixes  $p :: 'a :: \{\text{normalization-semidom-multiplicative, semiring-gcd, idom-divide}\}$  poly
  shows  $\text{smult } (\text{content } p) (\text{normalize } (\text{primitive-part } p)) = \text{normalize } p$ 
<proof>

context
begin

private

lemma content-1-mult:
  fixes  $f \ g :: 'a :: \{\text{semiring-gcd, factorial-semiring}\}$  poly
  assumes  $\text{content } f = 1 \ \text{content } g = 1$ 
  shows  $\text{content } (f * g) = 1$ 
<proof>

lemma content-mult:
  fixes  $p \ q :: 'a :: \{\text{factorial-semiring, semiring-gcd, normalization-semidom-multiplicative}\}$  poly
  shows  $\text{content } (p * q) = \text{content } p * \text{content } q$ 
<proof>

end

```

**lemma** *primitive-part-mult*:

**fixes**  $p\ q :: 'a :: \{\text{factorial-semiring, semiring-Gcd, ring-gcd, idom-divide,}$   
 $\text{normalization-semidom-multiplicative}\}$  *poly*  
**shows**  $\text{primitive-part } (p * q) = \text{primitive-part } p * \text{primitive-part } q$   
 $\langle \text{proof} \rangle$

**lemma** *primitive-part-smult*:

**fixes**  $p :: 'a :: \{\text{factorial-semiring, semiring-Gcd, ring-gcd, idom-divide,}$   
 $\text{normalization-semidom-multiplicative}\}$  *poly*  
**shows**  $\text{primitive-part } (\text{smult } a\ p) = \text{smult } (\text{unit-factor } a) (\text{primitive-part } p)$   
 $\langle \text{proof} \rangle$

**lemma** *primitive-part-dvd-primitive-partI* [*intro*]:

**fixes**  $p\ q :: 'a :: \{\text{factorial-semiring, semiring-Gcd, ring-gcd, idom-divide,}$   
 $\text{normalization-semidom-multiplicative}\}$  *poly*  
**shows**  $p\ \text{dvd}\ q \implies \text{primitive-part } p\ \text{dvd}\ \text{primitive-part } q$   
 $\langle \text{proof} \rangle$

**lemma** *content-prod-mset*:

**fixes**  $A :: 'a :: \{\text{factorial-semiring, semiring-Gcd, normalization-semidom-multiplicative}\}$   
*poly multiset*  
**shows**  $\text{content } (\text{prod-mset } A) = \text{prod-mset } (\text{image-mset } \text{content } A)$   
 $\langle \text{proof} \rangle$

**lemma** *content-prod-eq-1-iff*:

**fixes**  $p\ q :: 'a :: \{\text{factorial-semiring, semiring-Gcd, normalization-semidom-multiplicative}\}$   
*poly*  
**shows**  $\text{content } (p * q) = 1 \iff \text{content } p = 1 \wedge \text{content } q = 1$   
 $\langle \text{proof} \rangle$

### 4.33 A typeclass for algebraically closed fields

Since the required sort constraints are not available inside the class, we have to resort to a somewhat awkward way of writing the definition of algebraically closed fields:

**class** *alg-closed-field* = *field* +

**assumes**  $\text{alg-closed: } n > 0 \implies f\ n \neq 0 \implies \exists x. (\sum_{k \leq n}. f\ k * x^k) = 0$

We can then however easily show the equivalence to the proper definition:

**lemma** *alg-closed-imp-poly-has-root*:

**assumes**  $\text{degree } (p :: 'a :: \text{alg-closed-field } \text{poly}) > 0$   
**shows**  $\exists x. \text{poly } p\ x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *alg-closedI* [*Pure.intro*]:

**assumes**  $\bigwedge p :: 'a\ \text{poly. degree } p > 0 \implies \text{lead-coeff } p = 1 \implies \exists x. \text{poly } p\ x = 0$   
**shows**  $\text{OFCLASS}('a :: \text{field, alg-closed-field-class})$   
 $\langle \text{proof} \rangle$



**lemma** (in *alg-closed-field*) *nth-root-exists*:

**assumes**  $n > 0$

**shows**  $\exists y. y^n = (x :: 'a)$

*<proof>*

We can now prove by induction that every polynomial of degree  $n$  splits into a product of  $n$  linear factors:

**lemma** *alg-closed-imp-factorization*:

**fixes**  $p :: 'a :: \text{alg-closed-field poly}$

**assumes**  $p \neq 0$

**shows**  $\exists A. \text{size } A = \text{degree } p \wedge p = \text{smult } (\text{lead-coeff } p) (\prod_{x \in \#A. [: -x, 1:] )$

*<proof>*

As an alternative characterisation of algebraic closure, one can also say that any polynomial of degree at least 2 splits into non-constant factors:

**lemma** *alg-closed-imp-reducible*:

**assumes**  $\text{degree } (p :: 'a :: \text{alg-closed-field poly}) > 1$

**shows**  $\neg \text{irreducible } p$

*<proof>*

When proving algebraic closure through reducibility, we can assume w.l.o.g. that the polynomial is monic and has a non-zero constant coefficient:

**lemma** *alg-closedI-reducible*:

**assumes**  $\bigwedge p :: 'a \text{ poly. } \text{degree } p > 1 \implies \text{lead-coeff } p = 1 \implies \text{coeff } p \ 0 \neq 0 \implies \neg \text{irreducible } p$

**shows**  $\text{OFCLASS}('a :: \text{field}, \text{alg-closed-field-class})$

*<proof>*

Using a clever Tschirnhausen transformation mentioned e.g. in the article by Nowak [1], we can also assume w.l.o.g. that the coefficient  $a_{n-1}$  is zero.

**lemma** *alg-closedI-reducible-coeff-deg-minus-one-eq-0*:

**assumes**  $\bigwedge p :: 'a \text{ poly. } \text{degree } p > 1 \implies \text{lead-coeff } p = 1 \implies \text{coeff } p \ (\text{degree } p - 1) = 0 \implies$

$\text{coeff } p \ 0 \neq 0 \implies \neg \text{irreducible } p$

**shows**  $\text{OFCLASS}('a :: \text{field-char-0}, \text{alg-closed-field-class})$

*<proof>*

As a consequence of the full factorisation lemma proven above, we can also show that any polynomial with at least two different roots splits into two non-constant coprime factors:

**lemma** *alg-closed-imp-poly-splits-coprime*:

**assumes**  $\text{degree } (p :: 'a :: \{\text{alg-closed-field}\} \text{ poly}) > 1$

**assumes**  $\text{poly } p \ x = 0 \ \text{poly } p \ y = 0 \ x \neq y$

**obtains**  $r \ s \ \text{where } \text{degree } r > 0 \ \text{degree } s > 0 \ \text{coprime } r \ s \ p = r * s$

*<proof>*

### 4.34 Polynomials and limits

```

lemma filterlim-poly-at-infinity:
  fixes  $p::'a::\text{real-normed-field}$  poly
  assumes degree  $p > 0$ 
  shows filterlim (poly  $p$ ) at-infinity at-infinity
  <proof>

lemma poly-divide-tendsto-aux:
  fixes  $p::'a::\text{real-normed-field}$  poly
  shows  $((\lambda x. \text{poly } p \ x / x^{\text{degree } p})) \longrightarrow \text{lead-coeff } p$  at-infinity
  <proof>

lemma filterlim-power-at-infinity:
  assumes  $n \neq 0$ 
  shows filterlim  $(\lambda x::'a::\text{real-normed-field}. x^n)$  at-infinity at-infinity
  <proof>

lemma poly-divide-tendsto-0-at-infinity:
  fixes  $p::'a::\text{real-normed-field}$  poly
  assumes degree  $p > \text{degree } q$ 
  shows  $((\lambda x. \text{poly } q \ x / \text{poly } p \ x) \longrightarrow 0)$  at-infinity
  <proof>

lemma poly-eventually-not-zero:
  fixes  $p::\text{real}$  poly
  assumes  $p \neq 0$ 
  shows eventually  $(\lambda x. \text{poly } p \ x \neq 0)$  at-infinity
  <proof>

no-notation cCons (infixr  $\langle \# \# \rangle$  65)

end

```

## 5 A formalization of formal power series

```

theory Formal-Power-Series
imports
  Complex-Main
  Euclidean-Algorithm
  Primes
  HOL-Library.FuncSet
  HOL-Library.Multiset
begin

```

### 5.1 The type of formal power series

```

typedef  $'a \text{ fps} = \{f :: \text{nat} \Rightarrow 'a. \text{True}\}$ 
morphisms fps-nth Abs-fps

```

$\langle proof \rangle$

**notation**  $fps\text{-}nth$  (**infixl**  $\langle \$ \rangle$  75)

**lemma**  $expand\text{-}fps\text{-}eq$ :  $p = q \longleftrightarrow (\forall n. p \$ n = q \$ n)$   
 $\langle proof \rangle$

**lemmas**  $fps\text{-}eq\text{-}iff = expand\text{-}fps\text{-}eq$

**lemma**  $fps\text{-}ext$ :  $(\bigwedge n. p \$ n = q \$ n) \implies p = q$   
 $\langle proof \rangle$

**lemma**  $fps\text{-}nth\text{-}Abs\text{-}fps$  [*simp*]:  $Abs\text{-}fps\ f \$ n = f\ n$   
 $\langle proof \rangle$

Definition of the basic elements 0 and 1 and the basic operations of addition, negation and multiplication.

**instantiation**  $fps :: (zero)\ zero$

**begin**

**definition**  $fps\text{-}zero\text{-}def$ :  $0 = Abs\text{-}fps\ (\lambda n. 0)$

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

**end**

**lemma**  $fps\text{-}zero\text{-}nth$  [*simp*]:  $0 \$ n = 0$   
 $\langle proof \rangle$

**lemma**  $fps\text{-}nonzero\text{-}nth$ :  $f \neq 0 \longleftrightarrow (\exists n. f \$ n \neq 0)$   
 $\langle proof \rangle$

**lemma**  $fps\text{-}nonzero\text{-}nth\text{-}minimal$ :  $f \neq 0 \longleftrightarrow (\exists n. f \$ n \neq 0 \wedge (\forall m < n. f \$ m = 0))$   
(**is**  $?lhs \longleftrightarrow ?rhs$ )  
 $\langle proof \rangle$

**lemma**  $fps\text{-}nonzeroI$ :  $f \$ n \neq 0 \implies f \neq 0$   
 $\langle proof \rangle$

**instantiation**  $fps :: (\{one, zero\})\ one$

**begin**

**definition**  $fps\text{-}one\text{-}def$ :  $1 = Abs\text{-}fps\ (\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } 0)$

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

**end**

**lemma**  $fps\text{-}one\text{-}nth$  [*simp*]:  $1 \$ n = (\text{if } n = 0 \text{ then } 1 \text{ else } 0)$   
 $\langle proof \rangle$

**instantiation**  $fps :: (plus)\ plus$

**begin**

**definition**  $fps\text{-}plus\text{-}def$ :  $(+) = (\lambda f\ g. Abs\text{-}fps\ (\lambda n. f \$ n + g \$ n))$

```

instance ⟨proof⟩
end

lemma fps-add-nth [simp]:  $(f + g) \$ n = f \$ n + g \$ n$ 
  ⟨proof⟩

instantiation fps :: (minus) minus
begin
  definition fps-minus-def:  $(-) = (\lambda f\ g.\ \text{Abs-fps}\ (\lambda n.\ f \$ n - g \$ n))$ 
  instance ⟨proof⟩
end

lemma fps-sub-nth [simp]:  $(f - g) \$ n = f \$ n - g \$ n$ 
  ⟨proof⟩

instantiation fps :: (uminus) uminus
begin
  definition fps-uminus-def: uminus =  $(\lambda f.\ \text{Abs-fps}\ (\lambda n.\ -(f \$ n)))$ 
  instance ⟨proof⟩
end

lemma fps-neg-nth [simp]:  $(- f) \$ n = -(f \$ n)$ 
  ⟨proof⟩

lemma fps-neg-0 [simp]:  $-(0 :: 'a :: \text{group-add fps}) = 0$ 
  ⟨proof⟩

instantiation fps :: ( $\{\text{comm-monoid-add}, \text{times}\}$ ) times
begin
  definition fps-times-def:  $(*) = (\lambda f\ g.\ \text{Abs-fps}\ (\lambda n.\ \sum_{i=0..n} f \$ i * g \$ (n - i)))$ 
  instance ⟨proof⟩
end

lemma fps-mult-nth:  $(f * g) \$ n = (\sum_{i=0..n} f \$ i * g \$ (n - i))$ 
  ⟨proof⟩

lemma fps-mult-nth-0 [simp]:  $(f * g) \$ 0 = f \$ 0 * g \$ 0$ 
  ⟨proof⟩

lemma fps-mult-nth-1:  $(f * g) \$ 1 = f \$ 0 * g \$ 1 + f \$ 1 * g \$ 0$ 
  ⟨proof⟩

lemma fps-mult-nth-1' [simp]:  $(f * g) \$ \text{Suc } 0 = f \$ 0 * g \$ \text{Suc } 0 + f \$ \text{Suc } 0 * g \$ 0$ 
  ⟨proof⟩

lemmas mult-nth-0 = fps-mult-nth-0
lemmas mult-nth-1 = fps-mult-nth-1

```

**instance** *fps* :: (*{ comm-monoid-add, mult-zero }*) *mult-zero*  
 <proof>

**declare** *atLeastAtMost-iff* [*presburger*]  
**declare** *Bex-def* [*presburger*]  
**declare** *Ball-def* [*presburger*]

**lemma** *mult-delta-left*:  
 fixes *x y* :: '*a*::*mult-zero*  
 shows (if *b* then *x* else 0) \* *y* = (if *b* then *x* \* *y* else 0)  
 <proof>

**lemma** *mult-delta-right*:  
 fixes *x y* :: '*a*::*mult-zero*  
 shows *x* \* (if *b* then *y* else 0) = (if *b* then *x* \* *y* else 0)  
 <proof>

**lemma** *fps-one-mult*:  
 fixes *f* :: '*a*::*{ comm-monoid-add, mult-zero, monoid-mult }* *fps*  
 shows 1 \* *f* = *f*  
 and *f* \* 1 = *f*  
 <proof>

## 5.2 Subdegrees

**definition** *subdegree* :: ('*a*::*zero*) *fps*  $\Rightarrow$  *nat* **where**  
*subdegree f* = (if *f* = 0 then 0 else LEAST *n*. *f*\$*n*  $\neq$  0)

**lemma** *subdegreeI*:  
 assumes *f* \$ *d*  $\neq$  0 and  $\bigwedge i. i < d \implies f \$ i = 0$   
 shows *subdegree f* = *d*  
 <proof>

**lemma** *nth-subdegree-nonzero* [*simp,intro*]: *f*  $\neq$  0  $\implies f \$ \text{subdegree } f \neq 0$   
 <proof>

**lemma** *nth-less-subdegree-zero* [*dest*]: *n* < *subdegree f*  $\implies f \$ n = 0$   
 <proof>

**lemma** *subdegree-geI*:  
 assumes *f*  $\neq$  0  $\bigwedge i. i < n \implies f \$ i = 0$   
 shows *subdegree f*  $\geq n$   
 <proof>

**lemma** *subdegree-greaterI*:  
 assumes *f*  $\neq$  0  $\bigwedge i. i \leq n \implies f \$ i = 0$   
 shows *subdegree f* > *n*  
 <proof>

**lemma** *subdegree-leI*:

$f \ \$ \ n \neq 0 \implies \text{subdegree } f \leq n$

*<proof>*

**lemma** *subdegree-0 [simp]*:  $\text{subdegree } 0 = 0$

*<proof>*

**lemma** *subdegree-1 [simp]*:  $\text{subdegree } 1 = 0$

*<proof>*

**lemma** *subdegree-eq-0-iff*:  $\text{subdegree } f = 0 \longleftrightarrow f = 0 \vee f \ \$ \ 0 \neq 0$

*<proof>*

**lemma** *subdegree-eq-0 [simp]*:  $f \ \$ \ 0 \neq 0 \implies \text{subdegree } f = 0$

*<proof>*

**lemma** *nth-subdegree-zero-iff [simp]*:  $f \ \$ \ \text{subdegree } f = 0 \longleftrightarrow f = 0$

*<proof>*

**lemma** *fps-nonzero-subdegree-nonzeroI*:  $\text{subdegree } f > 0 \implies f \neq 0$

*<proof>*

**lemma** *subdegree-uminus [simp]*:

$\text{subdegree } (-(f :: ('a :: \text{group-add}) \text{fps})) = \text{subdegree } f$

*<proof>*

**lemma** *subdegree-minus-commute [simp]*:

**fixes**  $f :: ('a :: \text{group-add}) \text{fps}$

**shows**  $\text{subdegree } (f - g) = \text{subdegree } (g - f)$

*<proof>*

**lemma** *subdegree-add-ge'*:

**fixes**  $f \ g :: ('a :: \text{monoid-add}) \text{fps}$

**assumes**  $f + g \neq 0$

**shows**  $\text{subdegree } (f + g) \geq \min (\text{subdegree } f) (\text{subdegree } g)$

*<proof>*

**lemma** *subdegree-add-ge*:

**assumes**  $f \neq -(g :: ('a :: \text{group-add}) \text{fps})$

**shows**  $\text{subdegree } (f + g) \geq \min (\text{subdegree } f) (\text{subdegree } g)$

*<proof>*

**lemma** *subdegree-add-eq1*:

**assumes**  $f \neq 0$

**and**  $\text{subdegree } f < \text{subdegree } (g :: ('a :: \text{monoid-add}) \text{fps})$

**shows**  $\text{subdegree } (f + g) = \text{subdegree } f$

*<proof>*

**lemma** *subdegree-add-eq2*:

**assumes**  $g \neq 0$   
**and**  $\text{subdegree } g < \text{subdegree } (f :: 'a :: \text{monoid-add fps})$   
**shows**  $\text{subdegree } (f + g) = \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-diff-eq1*:  
**assumes**  $f \neq 0$   
**and**  $\text{subdegree } f < \text{subdegree } (g :: 'a :: \text{group-add fps})$   
**shows**  $\text{subdegree } (f - g) = \text{subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-diff-eq1-cancel*:  
**assumes**  $f \neq 0$   
**and**  $\text{subdegree } f < \text{subdegree } (g :: 'a :: \text{cancel-comm-monoid-add fps})$   
**shows**  $\text{subdegree } (f - g) = \text{subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-diff-eq2*:  
**assumes**  $g \neq 0$   
**and**  $\text{subdegree } g < \text{subdegree } (f :: 'a :: \text{group-add fps})$   
**shows**  $\text{subdegree } (f - g) = \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-diff-ge [simp]*:  
**assumes**  $f \neq (g :: 'a :: \text{group-add fps})$   
**shows**  $\text{subdegree } (f - g) \geq \min (\text{subdegree } f) (\text{subdegree } g)$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-diff-ge'*:  
**fixes**  $f g :: 'a :: \text{comm-monoid-diff fps}$   
**assumes**  $f - g \neq 0$   
**shows**  $\text{subdegree } (f - g) \geq \text{subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *nth-subdegree-mult-left [simp]*:  
**fixes**  $f g :: ('a :: \{\text{mult-zero, comm-monoid-add}\}) \text{ fps}$   
**shows**  $(f * g) \$ (\text{subdegree } f) = f \$ \text{subdegree } f * g \$ 0$   
 $\langle \text{proof} \rangle$

**lemma** *nth-subdegree-mult-right [simp]*:  
**fixes**  $f g :: ('a :: \{\text{mult-zero, comm-monoid-add}\}) \text{ fps}$   
**shows**  $(f * g) \$ (\text{subdegree } g) = f \$ 0 * g \$ \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *nth-subdegree-mult [simp]*:  
**fixes**  $f g :: ('a :: \{\text{mult-zero, comm-monoid-add}\}) \text{ fps}$   
**shows**  $(f * g) \$ (\text{subdegree } f + \text{subdegree } g) = f \$ \text{subdegree } f * g \$ \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-nth-eq0*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\}$  *fps*  
**assumes**  $n < \text{subdegree } f + \text{subdegree } g$   
**shows**  $(f * g) \$ n = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-subdegree-ge*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\}$  *fps*  
**assumes**  $f * g \neq 0$   
**shows**  $\text{subdegree } (f * g) \geq \text{subdegree } f + \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-mult'*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\}$  *fps*  
**assumes**  $f \$ \text{subdegree } f * g \$ \text{subdegree } g \neq 0$   
**shows**  $\text{subdegree } (f * g) = \text{subdegree } f + \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-mult [simp]*:  
**fixes**  $f\ g :: 'a :: \{\text{semiring-no-zero-divisors}\}$  *fps*  
**assumes**  $f \neq 0\ g \neq 0$   
**shows**  $\text{subdegree } (f * g) = \text{subdegree } f + \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-nth-conv-upto-subdegree-left*:  
**fixes**  $f\ g :: ('a :: \{\text{mult-zero}, \text{comm-monoid-add}\})$  *fps*  
**shows**  $(f * g) \$ n = (\sum_{i=\text{subdegree } f..n} f \$ i * g \$ (n - i))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-nth-conv-upto-subdegree-right*:  
**fixes**  $f\ g :: ('a :: \{\text{mult-zero}, \text{comm-monoid-add}\})$  *fps*  
**shows**  $(f * g) \$ n = (\sum_{i=0..n - \text{subdegree } g} f \$ i * g \$ (n - i))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-nth-conv-inside-subdegrees*:  
**fixes**  $f\ g :: ('a :: \{\text{mult-zero}, \text{comm-monoid-add}\})$  *fps*  
**shows**  $(f * g) \$ n = (\sum_{i=\text{subdegree } f..n - \text{subdegree } g} f \$ i * g \$ (n - i))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-nth-outside-subdegrees*:  
**fixes**  $f\ g :: ('a :: \{\text{mult-zero}, \text{comm-monoid-add}\})$  *fps*  
**shows**  $n < \text{subdegree } f \implies (f * g) \$ n = 0$   
**and**  $n < \text{subdegree } g \implies (f * g) \$ n = 0$   
 $\langle \text{proof} \rangle$

### 5.3 Ring structure

**instance** *fps* :: (*semigroup-add*) *semigroup-add*  
 $\langle \text{proof} \rangle$



```

instance fps :: (ab-semigroup-add) ab-semigroup-add
⟨proof⟩

instance fps :: (monoid-add) monoid-add
⟨proof⟩

instance fps :: (comm-monoid-add) comm-monoid-add
⟨proof⟩

instance fps :: (cancel-semigroup-add) cancel-semigroup-add
⟨proof⟩

instance fps :: (cancel-ab-semigroup-add) cancel-ab-semigroup-add
⟨proof⟩

instance fps :: (cancel-comm-monoid-add) cancel-comm-monoid-add ⟨proof⟩

instance fps :: (group-add) group-add
⟨proof⟩

instance fps :: (ab-group-add) ab-group-add
⟨proof⟩

instance fps :: (zero-neq-one) zero-neq-one
⟨proof⟩

lemma fps-mult-assoc-lemma:
  fixes k :: nat
    and f :: nat ⇒ nat ⇒ nat ⇒ 'a::comm-monoid-add
  shows (∑ j=0..k. ∑ i=0..j. f i (j - i) (n - j)) =
    (∑ j=0..k. ∑ i=0..k - j. f j i (n - j - i))
  ⟨proof⟩

instance fps :: (semiring-0) semiring-0
⟨proof⟩

instance fps :: (semiring-0-cancel) semiring-0-cancel ⟨proof⟩

lemma fps-mult-commute-lemma:
  fixes n :: nat
    and f :: nat ⇒ nat ⇒ 'a::comm-monoid-add
  shows (∑ i=0..n. f i (n - i)) = (∑ i=0..n. f (n - i) i)
  ⟨proof⟩

instance fps :: (comm-semiring-0) comm-semiring-0
⟨proof⟩

instance fps :: (comm-semiring-0-cancel) comm-semiring-0-cancel ⟨proof⟩

```

**instance** *fps* :: (*semiring-1*) *semiring-1*  
 ⟨*proof*⟩

**instance** *fps* :: (*comm-semiring-1*) *comm-semiring-1*  
 ⟨*proof*⟩

**instance** *fps* :: (*semiring-1-cancel*) *semiring-1-cancel* ⟨*proof*⟩

**lemma** *fps-square-nth*:  $(f^{\wedge}2) \$ n = (\sum k \leq n. f \$ k * f \$ (n - k))$   
 ⟨*proof*⟩

**lemma** *fps-sum-nth*:  $sum f S \$ n = sum (\lambda k. (f k) \$ n) S$   
 ⟨*proof*⟩

**definition** *fps-const* *c* = *Abs-fps* ( $\lambda n. \text{if } n = 0 \text{ then } c \text{ else } 0$ )

**lemma** *fps-nth-fps-const* [*simp*]: *fps-const* *c* \$ *n* = (*if* *n* = 0 *then* *c* *else* 0)  
 ⟨*proof*⟩

**lemma** *fps-const-0-eq-0* [*simp*]: *fps-const* 0 = 0  
 ⟨*proof*⟩

**lemma** *fps-const-nonzero-eq-nonzero*:  $c \neq 0 \implies \text{fps-const } c \neq 0$   
 ⟨*proof*⟩

**lemma** *fps-const-eq-0-iff* [*simp*]: *fps-const* *c* = 0  $\longleftrightarrow c = 0$   
 ⟨*proof*⟩

**lemma** *fps-const-1-eq-1* [*simp*]: *fps-const* 1 = 1  
 ⟨*proof*⟩

**lemma** *fps-const-eq-1-iff* [*simp*]: *fps-const* *c* = 1  $\longleftrightarrow c = 1$   
 ⟨*proof*⟩

**lemma** *subdegree-fps-const* [*simp*]: *subdegree* (*fps-const* *c*) = 0  
 ⟨*proof*⟩

**lemma** *fps-const-neg* [*simp*]:  $- (\text{fps-const } (c :: 'a :: \text{group-add})) = \text{fps-const } (- c)$   
 ⟨*proof*⟩

**lemma** *fps-const-add* [*simp*]:  $\text{fps-const } (c :: 'a :: \text{monoid-add}) + \text{fps-const } d = \text{fps-const } (c + d)$   
 ⟨*proof*⟩

**lemma** *fps-const-add-left*: *fps-const* (*c* :: 'a :: *monoid-add*) + *f* =  
*Abs-fps* ( $\lambda n. \text{if } n = 0 \text{ then } c + f \$ 0 \text{ else } f \$ n$ )  
 ⟨*proof*⟩

**lemma** *fps-const-add-right*:  $f + \text{fps-const } (c::'a::\text{monoid-add}) =$   
 $\text{Abs-fps } (\lambda n. \text{ if } n = 0 \text{ then } f\$0 + c \text{ else } f\$n)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-sub* [simp]:  $\text{fps-const } (c::'a::\text{group-add}) - \text{fps-const } d = \text{fps-const } (c - d)$   
 $\langle \text{proof} \rangle$

**lemmas** *fps-const-minus* = *fps-const-sub*

**lemma** *fps-const-mult*[simp]:  
**fixes**  $c\ d :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$   
**shows**  $\text{fps-const } c * \text{fps-const } d = \text{fps-const } (c * d)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-mult-left*:  
 $\text{fps-const } (c::'a::\{\text{comm-monoid-add}, \text{mult-zero}\}) * f = \text{Abs-fps } (\lambda n. c * f\$n)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-mult-right*:  
 $f * \text{fps-const } (c::'a::\{\text{comm-monoid-add}, \text{mult-zero}\}) = \text{Abs-fps } (\lambda n. f\$n * c)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-left-const-nth* [simp]:  
 $(\text{fps-const } (c::'a::\{\text{comm-monoid-add}, \text{mult-zero}\}) * f)\$n = c * f\$n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-right-const-nth* [simp]:  
 $(f * \text{fps-const } (c::'a::\{\text{comm-monoid-add}, \text{mult-zero}\}))\$n = f\$n * c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-power* [simp]:  $\text{fps-const } c ^ n = \text{fps-const } (c ^ n)$   
 $\langle \text{proof} \rangle$

**instance** *fps* :: (*ring*) *ring*  $\langle \text{proof} \rangle$

**instance** *fps* :: (*comm-ring*) *comm-ring*  $\langle \text{proof} \rangle$

**instance** *fps* :: (*ring-1*) *ring-1*  $\langle \text{proof} \rangle$

**instance** *fps* :: (*comm-ring-1*) *comm-ring-1*  $\langle \text{proof} \rangle$

**instance** *fps* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*  
 $\langle \text{proof} \rangle$

**instance** *fps* :: (*semiring-1-no-zero-divisors*) *semiring-1-no-zero-divisors*  $\langle \text{proof} \rangle$

**instance** *fps* :: (*{cancel-semigroup-add, semiring-no-zero-divisors-cancel}*)  
*semiring-no-zero-divisors-cancel*  
*<proof>*

**instance** *fps* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors* *<proof>*

**instance** *fps* :: (*ring-1-no-zero-divisors*) *ring-1-no-zero-divisors* *<proof>*

**instance** *fps* :: (*idom*) *idom* *<proof>*

**lemma** *fps-of-nat*: *fps-const (of-nat c) = of-nat c*  
*<proof>*

**lemma** *fps-of-int*: *fps-const (of-int c) = of-int c*  
*<proof>*

**lemma** *semiring-char-fps* [*simp*]: *CHAR('a :: comm-semiring-1 fps) = CHAR('a)*  
*<proof>*

**instance** *fps* :: (*{semiring-prime-char, comm-semiring-1}*) *semiring-prime-char*  
*<proof>*

**instance** *fps* :: (*{comm-semiring-prime-char, comm-semiring-1}*) *comm-semiring-prime-char*  
*<proof>*

**instance** *fps* :: (*{comm-ring-prime-char, comm-semiring-1}*) *comm-ring-prime-char*  
*<proof>*

**instance** *fps* :: (*{idom-prime-char, comm-semiring-1}*) *idom-prime-char*  
*<proof>*

**lemma** *fps-numeral-fps-const*: *numeral k = fps-const (numeral k)*  
*<proof>*

**lemmas** *numeral-fps-const = fps-numeral-fps-const*

**lemma** *neg-numeral-fps-const*:  
*(- numeral k :: 'a :: ring-1 fps) = fps-const (- numeral k)*  
*<proof>*

**lemma** *fps-numeral-nth*: *numeral n \$ i = (if i = 0 then numeral n else 0)*  
*<proof>*

**lemma** *fps-numeral-nth-0* [*simp*]: *numeral n \$ 0 = numeral n*  
*<proof>*

**lemma** *subdegree-numeral* [*simp*]: *subdegree (numeral n) = 0*  
*<proof>*

**lemma** *fps-nth-of-nat* [*simp*]:  
*(of-nat c) \$ n = (if n=0 then of-nat c else 0)*  
*<proof>*

**lemma** *fps-nth-of-int* [simp]:  
 (of-int c) \$ n = (if n=0 then of-int c else 0)  
 <proof>

**lemma** *fps-mult-of-nat-nth* [simp]:  
 shows (of-nat k \* f) \$ n = of-nat k \* f \$ n  
 and (f \* of-nat k) \$ n = f \$ n \* of-nat k  
 <proof>

**lemma** *fps-mult-of-int-nth* [simp]:  
 shows (of-int k \* f) \$ n = of-int k \* f \$ n  
 and (f \* of-int k) \$ n = f \$ n \* of-int k  
 <proof>

**lemma** *numeral-neq-fps-zero* [simp]: (numeral f :: 'a :: field-char-0 fps) ≠ 0  
 <proof>

**instance** *fps* :: (semiring-char-0) semiring-char-0  
 <proof>

**lemma** *subdegree-power-ge*:  
 $f^n \neq 0 \implies \text{subdegree } (f^n) \geq n * \text{subdegree } f$   
 <proof>

**lemma** *fps-pow-nth-below-subdegree*:  
 $k < n * \text{subdegree } f \implies (f^n) \$ k = 0$   
 <proof>

**lemma** *fps-pow-base* [simp]:  
 $(f^n) \$ (n * \text{subdegree } f) = (f \$ \text{subdegree } f)^n$   
 <proof>

**lemma** *subdegree-power-eqI*:  
 fixes f :: 'a :: semiring-1 fps  
 shows  $(f \$ \text{subdegree } f)^n \neq 0 \implies \text{subdegree } (f^n) = n * \text{subdegree } f$   
 <proof>

**lemma** *subdegree-power* [simp]:  
 $\text{subdegree } ((f :: ('a :: \text{semiring-1-no-zero-divisors}) \text{fps})^n) = n * \text{subdegree } f$   
 <proof>

**lemma** *subdegree-prod*:  
 fixes f :: 'a ⇒ 'b :: idom fps  
 assumes  $\bigwedge x. x \in A \implies f x \neq 0$   
 shows  $\text{subdegree } (\prod_{x \in A} f x) = (\sum_{x \in A} \text{subdegree } (f x))$   
 <proof>

**lemma** *minus-one-power-iff*:  $(- (1 :: 'a::ring-1)) ^ n = (\text{if even } n \text{ then } 1 \text{ else } - 1)$   
 ⟨proof⟩

**definition** *fps-X* = *Abs-fps* ( $\lambda n. \text{if } n = 1 \text{ then } 1 \text{ else } 0$ )

**lemma** *subdegree-fps-X [simp]*: *subdegree* (*fps-X* :: ('a :: zero-neq-one) *fps*) = 1  
 ⟨proof⟩

**lemma** *fps-X-mult-nth [simp]*:  
 fixes *f* :: 'a::{\i comm-monoid-add, mult-zero, monoid-mult} *fps*  
 shows  $(\text{fps-X} * f) \$ n = (\text{if } n = 0 \text{ then } 0 \text{ else } f \$ (n - 1))$   
 ⟨proof⟩

**lemma** *fps-X-mult-right-nth [simp]*:  
 fixes *a* :: 'a::{\i comm-monoid-add, mult-zero, monoid-mult} *fps*  
 shows  $(a * \text{fps-X}) \$ n = (\text{if } n = 0 \text{ then } 0 \text{ else } a \$ (n - 1))$   
 ⟨proof⟩

**lemma** *fps-mult-fps-X-commute*:  
 fixes *a* :: 'a::{\i comm-monoid-add, mult-zero, monoid-mult} *fps*  
 shows  $\text{fps-X} * a = a * \text{fps-X}$   
 ⟨proof⟩

**lemma** *fps-mult-fps-X-power-commute*:  $\text{fps-X} ^ k * a = a * \text{fps-X} ^ k$   
 ⟨proof⟩

**lemma** *fps-subdegree-mult-fps-X*:  
 fixes *f* :: 'a::{\i comm-monoid-add, mult-zero, monoid-mult} *fps*  
 assumes  $f \neq 0$   
 shows  $\text{subdegree} (\text{fps-X} * f) = \text{subdegree } f + 1$   
 and  $\text{subdegree} (f * \text{fps-X}) = \text{subdegree } f + 1$   
 ⟨proof⟩

**lemma** *fps-mult-fps-X-nonzero*:  
 fixes *f* :: 'a::{\i comm-monoid-add, mult-zero, monoid-mult} *fps*  
 assumes  $f \neq 0$   
 shows  $\text{fps-X} * f \neq 0$   
 and  $f * \text{fps-X} \neq 0$   
 ⟨proof⟩

**lemma** *fps-mult-fps-X-power-nonzero*:  
 assumes  $f \neq 0$   
 shows  $\text{fps-X} ^ n * f \neq 0$   
 and  $f * \text{fps-X} ^ n \neq 0$   
 ⟨proof⟩

**lemma** *fps-X-power-iff*:  $\text{fps-X} ^ n = \text{Abs-fps} (\lambda m. \text{if } m = n \text{ then } 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fps-X-nth[simp]*:  $\text{fps-X } n = (\text{if } n = 1 \text{ then } 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fps-X-power-nth[simp]*:  $(\text{fps-X}^k) \text{ } n = (\text{if } n = k \text{ then } 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fps-X-power-subdegree*:  $\text{subdegree } (\text{fps-X}^n) = n$   
 ⟨proof⟩

**lemma** *fps-X-power-mult-nth*:  
 $(\text{fps-X}^k * f) \text{ } n = (\text{if } n < k \text{ then } 0 \text{ else } f \text{ } (n - k))$   
 ⟨proof⟩

**lemma** *fps-X-power-mult-right-nth*:  
 $(f * \text{fps-X}^k) \text{ } n = (\text{if } n < k \text{ then } 0 \text{ else } f \text{ } (n - k))$   
 ⟨proof⟩

**lemma** *fps-subdegree-mult-fps-X-power*:  
 assumes  $f \neq 0$   
 shows  $\text{subdegree } (\text{fps-X}^n * f) = \text{subdegree } f + n$   
 and  $\text{subdegree } (f * \text{fps-X}^n) = \text{subdegree } f + n$   
 ⟨proof⟩

**lemma** *fps-mult-fps-X-plus-1-nth*:  
 $((1 + \text{fps-X}) * a) \text{ } n = (\text{if } n = 0 \text{ then } (a \text{ } n :: 'a :: \text{semiring-1}) \text{ else } a \text{ } n + a \text{ } (n - 1))$   
 ⟨proof⟩

**lemma** *fps-mult-right-fps-X-plus-1-nth*:  
 fixes  $a :: 'a :: \text{semiring-1}$   $\text{fps}$   
 shows  $(a * (1 + \text{fps-X})) \text{ } n = (\text{if } n = 0 \text{ then } a \text{ } n \text{ else } a \text{ } n + a \text{ } (n - 1))$   
 ⟨proof⟩

**lemma** *fps-X-neq-fps-const [simp]*:  $(\text{fps-X} :: 'a :: \text{zero-neq-one fps}) \neq \text{fps-const } c$   
 ⟨proof⟩

**lemma** *fps-X-neq-zero [simp]*:  $(\text{fps-X} :: 'a :: \text{zero-neq-one fps}) \neq 0$   
 ⟨proof⟩

**lemma** *fps-X-neq-one [simp]*:  $(\text{fps-X} :: 'a :: \text{zero-neq-one fps}) \neq 1$   
 ⟨proof⟩

**lemma** *fps-X-neq-numeral [simp]*:  $\text{fps-X} \neq \text{numeral } c$   
 ⟨proof⟩

**lemma** *fps-X-pow-eq-fps-X-pow-iff [simp]*:  $\text{fps-X}^m = \text{fps-X}^n \longleftrightarrow m = n$   
 ⟨proof⟩

## 5.4 Shifting and slicing

**definition**  $\text{fps-shift} :: \text{nat} \Rightarrow 'a \text{ fps} \Rightarrow 'a \text{ fps}$  **where**

$$\text{fps-shift } n \ f = \text{Abs-fps } (\lambda i. f \ \$ \ (i + n))$$

**lemma**  $\text{fps-shift-nth}$   $[\text{simp}]$ :  $\text{fps-shift } n \ f \ \$ \ i = f \ \$ \ (i + n)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-0}$   $[\text{simp}]$ :  $\text{fps-shift } 0 \ f = f$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-zero}$   $[\text{simp}]$ :  $\text{fps-shift } n \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-one}$ :  $\text{fps-shift } n \ 1 = (\text{if } n = 0 \text{ then } 1 \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-fps-const}$ :  $\text{fps-shift } n \ (\text{fps-const } c) = (\text{if } n = 0 \text{ then } \text{fps-const } c \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-numeral}$ :  $\text{fps-shift } n \ (\text{numeral } c) = (\text{if } n = 0 \text{ then } \text{numeral } c \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-fps-X}$   $[\text{simp}]$ :  
 $n \geq 1 \implies \text{fps-shift } n \ \text{fps-X} = (\text{if } n = 1 \text{ then } 1 \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-fps-X-power}$   $[\text{simp}]$ :  
 $n \leq m \implies \text{fps-shift } n \ (\text{fps-X} \wedge^m) = \text{fps-X} \wedge^{(m - n)}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-subdegree}$   $[\text{simp}]$ :  
 $n \leq \text{subdegree } f \implies \text{subdegree } (\text{fps-shift } n \ f) = \text{subdegree } f - n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-fps-shift}$ :  
 $\text{fps-shift } (m + n) \ f = \text{fps-shift } m \ (\text{fps-shift } n \ f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-fps-shift-reorder}$ :  
 $\text{fps-shift } m \ (\text{fps-shift } n \ f) = \text{fps-shift } n \ (\text{fps-shift } m \ f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-rev-shift}$ :  
 $m \leq n \implies \text{fps-shift } n \ (\text{Abs-fps } (\lambda k. \text{if } k < m \text{ then } 0 \text{ else } f \ \$ \ (k - m))) = \text{fps-shift } (n - m) \ f$   
 $m > n \implies \text{fps-shift } n \ (\text{Abs-fps } (\lambda k. \text{if } k < m \text{ then } 0 \text{ else } f \ \$ \ (k - m))) = \text{Abs-fps } (\lambda k. \text{if } k < m - n \text{ then } 0 \text{ else } f \ \$ \ (k - (m - n)))$



$\langle proof \rangle$

**lemma** *fps-shift-add*:

$fps\_shift\ n\ (f + g) = fps\_shift\ n\ f + fps\_shift\ n\ g$

$\langle proof \rangle$

**lemma** *fps-shift-diff*:

$fps\_shift\ n\ (f - g) = fps\_shift\ n\ f - fps\_shift\ n\ g$

$\langle proof \rangle$

**lemma** *fps-shift-uminus*:

$fps\_shift\ n\ (-f) = -\ fps\_shift\ n\ f$

$\langle proof \rangle$

**lemma** *fps-shift-mult*:

**assumes**  $n \leq subdegree\ (g :: 'b :: \{comm-monoid-add, mult-zero\}\ fps)$

**shows**  $fps\_shift\ n\ (h * g) = h * fps\_shift\ n\ g$

$\langle proof \rangle$

**lemma** *fps-shift-mult-right-noncomm*:

**assumes**  $n \leq subdegree\ (g :: 'b :: \{comm-monoid-add, mult-zero\}\ fps)$

**shows**  $fps\_shift\ n\ (g * h) = fps\_shift\ n\ g * h$

$\langle proof \rangle$

**lemma** *fps-shift-mult-right*:

**assumes**  $n \leq subdegree\ (g :: 'b :: comm-semiring-0\ fps)$

**shows**  $fps\_shift\ n\ (g * h) = h * fps\_shift\ n\ g$

$\langle proof \rangle$

**lemma** *fps-shift-mult-both*:

**fixes**  $f\ g :: 'a :: \{comm-monoid-add, mult-zero\}\ fps$

**assumes**  $m \leq subdegree\ f\ n \leq subdegree\ g$

**shows**  $fps\_shift\ m\ f * fps\_shift\ n\ g = fps\_shift\ (m+n)\ (f * g)$

$\langle proof \rangle$

**lemma** *fps-shift-subdegree-zero-iff* [simp]:

$fps\_shift\ (subdegree\ f)\ f = 0 \longleftrightarrow f = 0$

$\langle proof \rangle$

**lemma** *fps-shift-times-fps-X*:

**fixes**  $f\ g :: 'a :: \{comm-monoid-add, mult-zero, monoid-mult\}\ fps$

**shows**  $1 \leq subdegree\ f \implies fps\_shift\ 1\ f * fps\_X = f$

$\langle proof \rangle$

**lemma** *fps-shift-times-fps-X'* [simp]:

**fixes**  $f :: 'a :: \{comm-monoid-add, mult-zero, monoid-mult\}\ fps$

**shows**  $fps\_shift\ 1\ (f * fps\_X) = f$

$\langle proof \rangle$

**lemma** *fps-shift-times-fps-X''*:  
**fixes**  $f :: 'a :: \{comm-monoid-add, mult-zero, monoid-mult\}$  *fps*  
**shows**  $1 \leq n \implies \text{fps-shift } n (f * \text{fps-X}) = \text{fps-shift } (n - 1) f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-times-fps-X-power*:  
 $n \leq \text{subdegree } f \implies \text{fps-shift } n f * \text{fps-X}^{\wedge n} = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-times-fps-X-power' [simp]*:  
 $\text{fps-shift } n (f * \text{fps-X}^{\wedge n}) = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-times-fps-X-power''*:  
 $m \leq n \implies \text{fps-shift } n (f * \text{fps-X}^{\wedge m}) = \text{fps-shift } (n - m) f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-times-fps-X-power'''*:  
 $m > n \implies \text{fps-shift } n (f * \text{fps-X}^{\wedge m}) = f * \text{fps-X}^{\wedge (m - n)}$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-decompose*:  
 $f = \text{fps-shift } (\text{subdegree } f) f * \text{fps-X}^{\wedge \text{subdegree } f}$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-decompose'*:  
 $n \leq \text{subdegree } f \implies f = \text{fps-shift } n f * \text{fps-X}^{\wedge n}$   
 $\langle \text{proof} \rangle$

**instantiation** *fps* :: (zero) unit-factor  
**begin**  
**definition** *fps-unit-factor-def [simp]*:  
 $\text{unit-factor } f = \text{fps-shift } (\text{subdegree } f) f$   
**instance**  $\langle \text{proof} \rangle$   
**end**

**lemma** *fps-unit-factor-zero-iff*:  $\text{unit-factor } (f :: 'a :: \text{zero } \text{fps}) = 0 \longleftrightarrow f = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-nth-0*:  $f \neq 0 \implies \text{unit-factor } f \$ 0 \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-X-unit-factor*:  $\text{unit-factor } (\text{fps-X} :: 'a :: \text{zero-neq-one } \text{fps}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fps-X-power-unit-factor*:  $\text{unit-factor } (\text{fps-X}^{\wedge n}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-decompose*:

$f = \text{unit-factor } f * \text{fps-}X \wedge \text{subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-decompose'*:  
 $f = \text{fps-}X \wedge \text{subdegree } f * \text{unit-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-uminus*:  
 $\text{unit-factor } (-f) = - \text{unit-factor } (f :: 'a :: \text{group-add } \text{fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-shift*:  
**assumes**  $n \leq \text{subdegree } f$   
**shows**  $\text{unit-factor } (\text{fps-shift } n \ f) = \text{unit-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult-fps-X*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add}, \text{monoid-mult}, \text{mult-zero}\} \ \text{fps}$   
**shows**  $\text{unit-factor } (\text{fps-}X * f) = \text{unit-factor } f$   
**and**  $\text{unit-factor } (f * \text{fps-}X) = \text{unit-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult-fps-X-power*:  
**shows**  $\text{unit-factor } (\text{fps-}X \wedge n * f) = \text{unit-factor } f$   
**and**  $\text{unit-factor } (f * \text{fps-}X \wedge n) = \text{unit-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult-unit-factor*:  
**fixes**  $f \ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\} \ \text{fps}$   
**shows**  $\text{unit-factor } (f * \text{unit-factor } g) = \text{unit-factor } (f * g)$   
**and**  $\text{unit-factor } (\text{unit-factor } f * g) = \text{unit-factor } (f * g)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult-both-unit-factor*:  
**fixes**  $f \ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\} \ \text{fps}$   
**shows**  $\text{unit-factor } (\text{unit-factor } f * \text{unit-factor } g) = \text{unit-factor } (f * g)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult'*:  
**fixes**  $f \ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\} \ \text{fps}$   
**assumes**  $f \ \$ \ \text{subdegree } f * g \ \$ \ \text{subdegree } g \neq 0$   
**shows**  $\text{unit-factor } (f * g) = \text{unit-factor } f * \text{unit-factor } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-factor-mult*:  
**fixes**  $f \ g :: 'a :: \text{semiring-no-zero-divisors } \text{fps}$   
**shows**  $\text{unit-factor } (f * g) = \text{unit-factor } f * \text{unit-factor } g$   
 $\langle \text{proof} \rangle$

**definition**  $\text{fps-cutoff } n \ f = \text{Abs-fps } (\lambda i. \text{ if } i < n \text{ then } f\$i \text{ else } 0)$

**lemma**  $\text{fps-cutoff-nth [simp]: } \text{fps-cutoff } n \ f \ \$ \ i = (\text{if } i < n \text{ then } f\$i \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-zero-iff: } \text{fps-cutoff } n \ f = 0 \longleftrightarrow (f = 0 \vee n \leq \text{subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-0 [simp]: } \text{fps-cutoff } 0 \ f = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-zero [simp]: } \text{fps-cutoff } n \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-one: } \text{fps-cutoff } n \ 1 = (\text{if } n = 0 \text{ then } 0 \text{ else } 1)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-fps-const: } \text{fps-cutoff } n \ (\text{fps-const } c) = (\text{if } n = 0 \text{ then } 0 \text{ else } \text{fps-const } c)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-numeral: } \text{fps-cutoff } n \ (\text{numeral } c) = (\text{if } n = 0 \text{ then } 0 \text{ else numeral } c)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-cutoff:}$   
 $\text{fps-shift } n \ f * \text{fps-X}^n + \text{fps-cutoff } n \ f = f$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-shift-cutoff':}$   
 $\text{fps-X}^n * \text{fps-shift } n \ f + \text{fps-cutoff } n \ f = f$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-left-mult-nth:}$   
 $k < n \implies (\text{fps-cutoff } n \ f * g) \$ k = (f * g) \$ k$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-add: } \text{fps-cutoff } n \ (f + g :: 'a :: \text{monoid-add fps}) = \text{fps-cutoff } n \ f + \text{fps-cutoff } n \ g$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-diff: } \text{fps-cutoff } n \ (f - g :: 'a :: \text{group-add fps}) = \text{fps-cutoff } n \ f - \text{fps-cutoff } n \ g$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-uminus: } \text{fps-cutoff } n \ (-f :: 'a :: \text{group-add fps}) = -\text{fps-cutoff } n \ f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cutoff-right-mult-nth*:  
**assumes**  $k < n$   
**shows**  $(f * \text{fps-cutoff } n \ g) \$ k = (f * g) \$ k$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cutoff-eq-fps-cutoff-iff*:  
 $\text{fps-cutoff } n \ f = \text{fps-cutoff } n \ g \longleftrightarrow (\forall k < n. \text{fps-nth } f \ k = \text{fps-nth } g \ k)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-conv-fps-X-power-mult-fps-shift*:  
**assumes**  $f = 0 \vee \text{subdegree } f \geq n$   
**shows**  $f = \text{fps-X}^n * \text{fps-shift } n \ f$   
 $\langle \text{proof} \rangle$

## 5.5 Metrizability

**instantiation** *fps* :: (*minus*, *zero*) *dist*  
**begin**

**definition**  
 $\text{dist-fps-def: } \text{dist } (a :: 'a \text{ fps}) \ b = (\text{if } a = b \text{ then } 0 \text{ else inverse } (2^{\wedge} \text{subdegree } (a - b)))$

**lemma** *dist-fps-ge0*:  $\text{dist } (a :: 'a \text{ fps}) \ b \geq 0$   
 $\langle \text{proof} \rangle$

**instance**  $\langle \text{proof} \rangle$

**end**

**instantiation** *fps* :: (*group-add*) *metric-space*  
**begin**

**definition** *uniformity-fps-def* [code del]:  
 $(\text{uniformity} :: ('a \text{ fps} \times 'a \text{ fps}) \text{ filter}) = (\text{INF } e \in \{0 < ..\}. \text{principal } \{(x, y). \text{dist } x \ y < e\})$

**definition** *open-fps-def'* [code del]:  
 $\text{open } (U :: 'a \text{ fps set}) \longleftrightarrow (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U) \text{uniformity})$

**lemma** *dist-fps-sym*:  $\text{dist } (a :: 'a \text{ fps}) \ b = \text{dist } b \ a$   
 $\langle \text{proof} \rangle$

**instance**  
 $\langle \text{proof} \rangle$

**end**

**declare** *uniformity-Abort*[**where** 'a='a :: *group-add fps*, *code*]

**lemma** *open-fps-def*: *open* (*S* :: 'a::*group-add fps set*) = ( $\forall a \in S. \exists r. r > 0 \wedge \{y. \text{dist } y \ a < r\} \subseteq S$ )  
 <proof>

Topology

## 5.6 The topology of formal power series

A set of formal power series is open iff for any power series  $f$  in it, there exists some number  $n$  such that all power series that agree with  $f$  on the first  $n$  components are also in it.

**lemma** *open-fps-iff*:  
 $\text{open } A \longleftrightarrow (\forall F \in A. \exists n. \{G. \text{fps-cutoff } n \ G = \text{fps-cutoff } n \ F\} \subseteq A)$   
 <proof>

**lemma** *open-fps-cutoff*: *open*  $\{H. \text{fps-cutoff } N \ H = \text{fps-cutoff } N \ G\}$   
 <proof>

**lemma** *eventually-fps-nth-eq-nhds-fps-strong*:  
 $\text{eventually } (\lambda g. \forall k \leq n. \text{fps-nth } g \ k = \text{fps-nth } f \ k) \ (\text{nhds } f)$   
 <proof>

**lemma** *eventually-fps-nth-eq-nhds-fps*:  $\text{eventually } (\lambda g. \text{fps-nth } g \ k = \text{fps-nth } f \ k) \ (\text{nhds } f)$   
 <proof>

A family of formal power series  $f_x$  tends to a limit series  $g$  at some filter  $F$  iff for any  $N \geq 0$ , the set of  $x$  for which  $f_x$  and  $G$  agree on the first  $N$  coefficients is in  $F$ .

For a sequence  $(f_i)_{n \geq 0}$  this means that  $f_i \longrightarrow G$  iff for any  $N \geq 0$ ,  $f_x$  and  $G$  agree for all but finitely many  $x$ .

**lemma** *tendsto-fps-iff*:  
 $\text{filterlim } f \ (\text{nhds } (g :: 'a :: \text{group-add fps})) \ F \longleftrightarrow$   
 $(\forall n. \text{eventually } (\lambda x. \text{fps-nth } (f \ x) \ n = \text{fps-nth } g \ n) \ F)$   
 <proof>

**lemma** *tendsto-fpsI*:  
**assumes**  $\bigwedge n. \text{eventually } (\lambda x. \text{fps-nth } (f \ x) \ n = \text{fps-nth } G \ n) \ F$   
**shows**  $\text{filterlim } f \ (\text{nhds } (G :: 'a :: \text{group-add fps})) \ F$   
 <proof>

The infinite sums and justification of the notation in textbooks.

**lemma** *reals-power-lt-ex*:  
**fixes**  $x \ y :: \text{real}$   
**assumes**  $x > 0$

and  $y1: y > 1$   
 shows  $\exists k > 0. (1/y)^k < x$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sum-rep-nth*:  $(\text{sum } (\lambda i. \text{fps-const}(a\$i) * \text{fps-X}^i) \{0..m\})\$n = (\text{if } n \leq m \text{ then } a\$n \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-notation*:  $(\lambda n. \text{sum } (\lambda i. \text{fps-const}(a\$i) * \text{fps-X}^i) \{0..n\}) \longrightarrow a$   
 (is  $?s \longrightarrow a$ )  
 $\langle \text{proof} \rangle$

## 5.7 Division

**declare** *sum.cong*[*fundef-cong*]

**fun** *fps-left-inverse-constructor* ::  
 'a::{comm-monoid-add,times,uminus}  $\text{fps} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a$   
**where**  
 $\text{fps-left-inverse-constructor } f \ a \ 0 = a$   
 $|\ \text{fps-left-inverse-constructor } f \ a \ (\text{Suc } n) =$   
 $\quad - \text{sum } (\lambda i. \text{fps-left-inverse-constructor } f \ a \ i * f\$ (\text{Suc } n - i)) \{0..n\} * a$

— This will construct a left inverse for f in case that  $x * f \$ 0 = 1$

**abbreviation** *fps-left-inverse*  $\equiv (\lambda f \ x. \text{Abs-fps } (\text{fps-left-inverse-constructor } f \ x))$

**fun** *fps-right-inverse-constructor* ::  
 'a::{comm-monoid-add,times,uminus}  $\text{fps} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a$   
**where**  
 $\text{fps-right-inverse-constructor } f \ a \ 0 = a$   
 $|\ \text{fps-right-inverse-constructor } f \ a \ n =$   
 $\quad - a * \text{sum } (\lambda i. f\$i * \text{fps-right-inverse-constructor } f \ a \ (n - i)) \{1..n\}$

— This will construct a right inverse for f in case that  $f \$ 0 * y = 1$

**abbreviation** *fps-right-inverse*  $\equiv (\lambda f \ y. \text{Abs-fps } (\text{fps-right-inverse-constructor } f \ y))$

**instantiation** *fps* :: ( $\{ \text{comm-monoid-add}, \text{inverse}, \text{times}, \text{uminus} \}$ ) *inverse*  
**begin**

— For backwards compatibility.

**abbreviation** *natfun-inverse*:: 'a  $\text{fps} \Rightarrow \text{nat} \Rightarrow 'a$   
**where** *natfun-inverse*  $f \equiv \text{fps-right-inverse-constructor } f \ (\text{inverse } (f\$0))$

**definition** *fps-inverse-def*:  $\text{inverse } f = \text{Abs-fps } (\text{natfun-inverse } f)$

— With scalars from a (possibly non-commutative) ring, this defines a right inverse. Furthermore, if scalars are of class *mult-zero* and satisfy condition  $\text{inverse } 0 = 0$ , then this will evaluate to zero when the zeroth term is zero.

**definition** *fps-divide-def*:  $f \text{ div } g = \text{fps-shift } (\text{subdegree } g) (f * \text{inverse } (\text{unit-factor } g))$

g))

— If scalars are of class *mult-zero* and satisfy condition *inverse 0 = 0*, then div by zero will equal zero.

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

**end**

**lemma** *fps-lr-inverse-0-iff*:

$(fps\text{-}left\text{-}inverse\ f\ x)\ \$\ 0 = 0 \longleftrightarrow x = 0$   
 $(fps\text{-}right\text{-}inverse\ f\ x)\ \$\ 0 = 0 \longleftrightarrow x = 0$   
 $\langle proof \rangle$

**lemma** *fps-inverse-0-iff'*:  $(inverse\ f)\ \$\ 0 = 0 \longleftrightarrow inverse\ (f\ \$\ 0) = 0$   
 $\langle proof \rangle$

**lemma** *fps-inverse-0-iff[simp]*:  $(inverse\ f)\ \$\ 0 = (0::'a::division\text{-}ring) \longleftrightarrow f\ \$\ 0 = 0$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-nth-0*:

$(fps\text{-}left\text{-}inverse\ f\ x)\ \$\ 0 = x\ (fps\text{-}right\text{-}inverse\ f\ x)\ \$\ 0 = x$   
 $\langle proof \rangle$

**lemma** *fps-inverse-nth-0 [simp]*:  $(inverse\ f)\ \$\ 0 = inverse\ (f\ \$\ 0)$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-starting0*:

**fixes**  $f :: 'a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero,uminus\}$  *fps*  
**and**  $g :: 'b::\{ab\text{-}group\text{-}add,mult\text{-}zero\}$  *fps*  
**shows**  $fps\text{-}left\text{-}inverse\ f\ 0 = 0$   
**and**  $fps\text{-}right\text{-}inverse\ g\ 0 = 0$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-eq0-imp-starting0*:

$fps\text{-}left\text{-}inverse\ f\ x = 0 \implies x = 0$   
 $fps\text{-}right\text{-}inverse\ f\ x = 0 \implies x = 0$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-eq-0-iff*:

**fixes**  $x :: 'a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero,uminus\}$   
**and**  $y :: 'b::\{ab\text{-}group\text{-}add,mult\text{-}zero\}$   
**shows**  $fps\text{-}left\text{-}inverse\ f\ x = 0 \longleftrightarrow x = 0$   
**and**  $fps\text{-}right\text{-}inverse\ g\ y = 0 \longleftrightarrow y = 0$   
 $\langle proof \rangle$

**lemma** *fps-inverse-eq-0-iff'*:

**fixes**  $f :: 'a::\{ab\text{-}group\text{-}add,inverse,mult\text{-}zero\}$  *fps*  
**shows**  $inverse\ f = 0 \longleftrightarrow inverse\ (f\ \$\ 0) = 0$



$\langle \text{proof} \rangle$

**lemma** *fps-inverse-eq-0-iff*[simp]:  $\text{inverse } f = (0 :: ('a :: \text{division-ring}) \text{fps}) \longleftrightarrow f \$ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemmas** *fps-inverse-eq-0'* = *iffD2*[*OF fps-inverse-eq-0-iff*]  
**lemmas** *fps-inverse-eq-0* = *iffD2*[*OF fps-inverse-eq-0-iff*]

**lemma** *fps-const-lr-inverse*:  
**fixes**  $a :: 'a :: \{\text{ab-group-add, mult-zero}\}$   
**and**  $b :: 'b :: \{\text{comm-monoid-add, mult-zero, uminus}\}$   
**shows**  $\text{fps-left-inverse } (\text{fps-const } a) \ x = \text{fps-const } x$   
**and**  $\text{fps-right-inverse } (\text{fps-const } b) \ y = \text{fps-const } y$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-inverse*:  
**fixes**  $a :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus}\}$   
**shows**  $\text{inverse } (\text{fps-const } a) = \text{fps-const } (\text{inverse } a)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-lr-inverse-zero*:  
**fixes**  $x :: 'a :: \{\text{ab-group-add, mult-zero}\}$   
**and**  $y :: 'b :: \{\text{comm-monoid-add, mult-zero, uminus}\}$   
**shows**  $\text{fps-left-inverse } 0 \ x = \text{fps-const } x$   
**and**  $\text{fps-right-inverse } 0 \ y = \text{fps-const } y$   
 $\langle \text{proof} \rangle$

**lemma** *fps-inverse-zero-conv-fps-const*:  
 $\text{inverse } (0 :: 'a :: \{\text{comm-monoid-add, mult-zero, uminus, inverse}\} \text{fps}) = \text{fps-const } (\text{inverse } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-inverse-zero'*:  
**assumes**  $\text{inverse } (0 :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus}\}) = 0$   
**shows**  $\text{inverse } (0 :: 'a \text{fps}) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-inverse-zero* [simp]:  
 $\text{inverse } (0 :: 'a :: \text{division-ring } \text{fps}) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-lr-inverse-one*:  
**fixes**  $x :: 'a :: \{\text{ab-group-add, mult-zero, one}\}$   
**and**  $y :: 'b :: \{\text{comm-monoid-add, mult-zero, uminus, one}\}$   
**shows**  $\text{fps-left-inverse } 1 \ x = \text{fps-const } x$   
**and**  $\text{fps-right-inverse } 1 \ y = \text{fps-const } y$   
 $\langle \text{proof} \rangle$

**lemma** *fps-lr-inverse-one-one*:

*fps-left-inverse* 1 1 = (1 :: 'a :: {ab-group-add, mult-zero, one} fps)

*fps-right-inverse* 1 1 = (1 :: 'b :: {comm-monoid-add, mult-zero, uminus, one} fps)

⟨proof⟩

**lemma** *fps-inverse-one'*:

**assumes** *inverse* (1 :: 'a :: {comm-monoid-add, inverse, mult-zero, uminus, one}) = 1

**shows** *inverse* (1 :: 'a fps) = 1

⟨proof⟩

**lemma** *fps-inverse-one [simp]*: *inverse* (1 :: 'a :: division-ring fps) = 1

⟨proof⟩

**lemma** *fps-lr-inverse-minus*:

**fixes** *f* :: 'a :: ring-1 fps

**shows** *fps-left-inverse* (−*f*) (−*x*) = − *fps-left-inverse* *f* *x*

**and** *fps-right-inverse* (−*f*) (−*x*) = − *fps-right-inverse* *f* *x*

⟨proof⟩

**lemma** *fps-inverse-minus [simp]*: *inverse* (−*f*) = − *inverse* (*f* :: 'a :: division-ring fps)

⟨proof⟩

**lemma** *fps-left-inverse*:

**fixes** *f* :: 'a :: ring-1 fps

**assumes** *f0*: *x* \* *f*\$0 = 1

**shows** *fps-left-inverse* *f* *x* \* *f* = 1

⟨proof⟩

**lemma** *fps-right-inverse*:

**fixes** *f* :: 'a :: ring-1 fps

**assumes** *f0*: *f*\$0 \* *y* = 1

**shows** *f* \* *fps-right-inverse* *f* *y* = 1

⟨proof⟩

It is possible in a ring for an element to have a left inverse but not a right inverse, or vice versa. But when an element has both, they must be the same.

**lemma** *fps-left-inverse-eq-fps-right-inverse*:

**fixes** *f* :: 'a :: ring-1 fps

**assumes** *f0*: *x* \* *f*\$0 = 1 *f* \$ 0 \* *y* = 1

— These assumptions imply that *x* equals *y*, but no need to assume that.

**shows** *fps-left-inverse* *f* *x* = *fps-right-inverse* *f* *y*

⟨proof⟩

**lemma** *fps-left-inverse-eq-fps-right-inverse-comm*:

**fixes** *f* :: 'a :: comm-ring-1 fps

**assumes** *f0*: *x* \* *f*\$0 = 1

**shows** *fps-left-inverse* *f* *x* = *fps-right-inverse* *f* *x*

$\langle \text{proof} \rangle$

**lemma** *fps-left-inverse'*:

**fixes**  $f :: 'a::\text{ring-1 } \text{fps}$

**assumes**  $x * f\$0 = 1 \ f\$0 * y = 1$

— These assumptions imply  $x$  equals  $y$ , but no need to assume that.

**shows**  $\text{fps-right-inverse } f \ y * f = 1$

$\langle \text{proof} \rangle$

**lemma** *fps-right-inverse'*:

**fixes**  $f :: 'a::\text{ring-1 } \text{fps}$

**assumes**  $x * f\$0 = 1 \ f\$0 * y = 1$

— These assumptions imply  $x$  equals  $y$ , but no need to assume that.

**shows**  $f * \text{fps-left-inverse } f \ x = 1$

$\langle \text{proof} \rangle$

**lemma** *inverse-mult-eq-1* [intro]:

**assumes**  $f\$0 \neq (0 :: 'a::\text{division-ring})$

**shows**  $\text{inverse } f * f = 1$

$\langle \text{proof} \rangle$

**lemma** *inverse-mult-eq-1'*:

**assumes**  $f\$0 \neq (0 :: 'a::\text{division-ring})$

**shows**  $f * \text{inverse } f = 1$

$\langle \text{proof} \rangle$

**lemma** *fps-mult-left-inverse-unit-factor*:

**fixes**  $f :: 'a::\text{ring-1 } \text{fps}$

**assumes**  $x * f \ \$ \ \text{subdegree } f = 1$

**shows**  $\text{fps-left-inverse } (\text{unit-factor } f) \ x * f = \text{fps-}X \wedge \text{subdegree } f$

$\langle \text{proof} \rangle$

**lemma** *fps-mult-right-inverse-unit-factor*:

**fixes**  $f :: 'a::\text{ring-1 } \text{fps}$

**assumes**  $f \ \$ \ \text{subdegree } f * y = 1$

**shows**  $f * \text{fps-right-inverse } (\text{unit-factor } f) \ y = \text{fps-}X \wedge \text{subdegree } f$

$\langle \text{proof} \rangle$

**lemma** *fps-mult-right-inverse-unit-factor-divring*:

$(f :: 'a::\text{division-ring } \text{fps}) \neq 0 \implies f * \text{inverse } (\text{unit-factor } f) = \text{fps-}X \wedge \text{subdegree } f$

$\langle \text{proof} \rangle$

**lemma** *fps-left-inverse-idempotent-ring1*:

**fixes**  $f :: 'a::\text{ring-1 } \text{fps}$

**assumes**  $x * f\$0 = 1 \ y * x = 1$

— These assumptions imply  $y$  equals  $f\$0$ , but no need to assume that.

**shows**  $\text{fps-left-inverse } (\text{fps-left-inverse } f \ x) \ y = f$

$\langle \text{proof} \rangle$

**lemma** *fps-left-inverse-idempotent-comm-ring1*:  
**fixes**  $f :: 'a::comm-ring-1\ fps$   
**assumes**  $x * f\$0 = 1$   
**shows**  $fps\_left\_inverse\ (fps\_left\_inverse\ f\ x)\ (f\$0) = f$   
 $\langle proof \rangle$

**lemma** *fps-right-inverse-idempotent-ring1*:  
**fixes**  $f :: 'a::ring-1\ fps$   
**assumes**  $f\$0 * x = 1\ x * y = 1$   
— These assumptions imply  $y$  equals  $f\$0$ , but no need to assume that.  
**shows**  $fps\_right\_inverse\ (fps\_right\_inverse\ f\ x)\ y = f$   
 $\langle proof \rangle$

**lemma** *fps-right-inverse-idempotent-comm-ring1*:  
**fixes**  $f :: 'a::comm-ring-1\ fps$   
**assumes**  $f\$0 * x = 1$   
**shows**  $fps\_right\_inverse\ (fps\_right\_inverse\ f\ x)\ (f\$0) = f$   
 $\langle proof \rangle$

**lemma** *fps-inverse-idempotent[intro, simp]*:  
 $f\$0 \neq (0::'a::division-ring) \implies inverse\ (inverse\ f) = f$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-unique-ring1*:  
**fixes**  $f\ g :: 'a :: ring-1\ fps$   
**assumes**  $fg: f * g = 1\ g\$0 * f\$0 = 1$   
**shows**  $fps\_left\_inverse\ g\ (f\$0) = f$   
**and**  $fps\_right\_inverse\ f\ (g\$0) = g$   
 $\langle proof \rangle$

**lemma** *fps-lr-inverse-unique-divring*:  
**fixes**  $f\ g :: 'a :: division-ring\ fps$   
**assumes**  $fg: f * g = 1$   
**shows**  $fps\_left\_inverse\ g\ (f\$0) = f$   
**and**  $fps\_right\_inverse\ f\ (g\$0) = g$   
 $\langle proof \rangle$

**lemma** *fps-inverse-unique*:  
**fixes**  $f\ g :: 'a :: division-ring\ fps$   
**assumes**  $fg: f * g = 1$   
**shows**  $inverse\ f = g$   
 $\langle proof \rangle$

**lemma** *inverse-fps-numeral*:  
 $inverse\ (numeral\ n :: ('a :: field-char-0)\ fps) = fps\_const\ (inverse\ (numeral\ n))$   
 $\langle proof \rangle$

**lemma** *inverse-fps-of-nat*:

*inverse* (of-nat  $n :: 'a :: \{semiring-1, times, uminus, inverse\}$   $fps$ ) =  
*fps-const* (*inverse* (of-nat  $n$ ))  
 ⟨proof⟩

**lemma** *fps-lr-inverse-mult-ring1*:  
 fixes  $f\ g :: 'a :: ring-1\ fps$   
 assumes  $x: x * f\$0 = 1\ f\$0 * x = 1$   
 and  $y: y * g\$0 = 1\ g\$0 * y = 1$   
 shows  $fps\text{-}left\text{-}inverse\ (f * g)\ (y*x) = fps\text{-}left\text{-}inverse\ g\ y * fps\text{-}left\text{-}inverse\ f\ x$   
 and  $fps\text{-}right\text{-}inverse\ (f * g)\ (y*x) = fps\text{-}right\text{-}inverse\ g\ y * fps\text{-}right\text{-}inverse\ f\ x$   
 ⟨proof⟩

**lemma** *fps-lr-inverse-mult-divring*:  
 fixes  $f\ g :: 'a :: division\text{-}ring\ fps$   
 shows  $fps\text{-}left\text{-}inverse\ (f * g)\ (inverse\ ((f*g)\$0)) =$   
 $fps\text{-}left\text{-}inverse\ g\ (inverse\ (g\$0)) * fps\text{-}left\text{-}inverse\ f\ (inverse\ (f\$0))$   
 and  $fps\text{-}right\text{-}inverse\ (f * g)\ (inverse\ ((f*g)\$0)) =$   
 $fps\text{-}right\text{-}inverse\ g\ (inverse\ (g\$0)) * fps\text{-}right\text{-}inverse\ f\ (inverse\ (f\$0))$   
 ⟨proof⟩

**lemma** *fps-inverse-mult-divring*:  
 $inverse\ (f * g) = inverse\ g * inverse\ (f :: 'a :: division\text{-}ring\ fps)$   
 ⟨proof⟩

**lemma** *fps-inverse-mult*:  $inverse\ (f * g :: 'a :: field\ fps) = inverse\ f * inverse\ g$   
 ⟨proof⟩

**lemma** *inverse-prod-fps*:  $inverse\ (prod\ f\ A) = (\prod_{x \in A}. inverse\ (f\ x) :: 'a :: field\ fps)$   
 ⟨proof⟩

**lemma** *fps-lr-inverse-gp-ring1*:  
 fixes  $ones\ ones\text{-}inv :: 'a :: ring-1\ fps$   
 defines  $ones \equiv Abs\text{-}fps\ (\lambda n. 1)$   
 and  $ones\text{-}inv \equiv Abs\text{-}fps\ (\lambda n. if\ n=0\ then\ 1\ else\ if\ n=1\ then\ -\ 1\ else\ 0)$   
 shows  $fps\text{-}left\text{-}inverse\ ones\ 1 = ones\text{-}inv$   
 and  $fps\text{-}right\text{-}inverse\ ones\ 1 = ones\text{-}inv$   
 ⟨proof⟩

**lemma** *fps-lr-inverse-gp-ring1'*:  
 fixes  $ones :: 'a :: ring-1\ fps$   
 defines  $ones \equiv Abs\text{-}fps\ (\lambda n. 1)$   
 shows  $fps\text{-}left\text{-}inverse\ ones\ 1 = 1 - fps\text{-}X$   
 and  $fps\text{-}right\text{-}inverse\ ones\ 1 = 1 - fps\text{-}X$   
 ⟨proof⟩

**lemma** *fps-inverse-gp*:  
 $inverse\ (Abs\text{-}fps(\lambda n. (1 :: 'a :: division\text{-}ring))) =$

*Abs-fps* ( $\lambda n. \text{if } n = 0 \text{ then } 1 \text{ else if } n = 1 \text{ then } -1 \text{ else } 0$ )  
 <proof>

**lemma** *fps-inverse-gp'*: *inverse* (*Abs-fps* ( $\lambda n. 1 :: 'a :: \text{division-ring}$ ))) =  $1 - \text{fps-X}$   
 <proof>

**lemma** *fps-lr-inverse-one-minus-fps-X*:  
**fixes** *ones* ::  $'a :: \text{ring-1}$  *fps*  
**defines** *ones*  $\equiv \text{Abs-fps } (\lambda n. 1)$   
**shows** *fps-left-inverse* ( $1 - \text{fps-X}$ )  $1 = \text{ones}$   
**and** *fps-right-inverse* ( $1 - \text{fps-X}$ )  $1 = \text{ones}$   
 <proof>

**lemma** *fps-inverse-one-minus-fps-X*:  
**fixes** *ones* ::  $'a :: \text{division-ring}$  *fps*  
**defines** *ones*  $\equiv \text{Abs-fps } (\lambda n. 1)$   
**shows** *inverse* ( $1 - \text{fps-X}$ ) = *ones*  
 <proof>

**lemma** *fps-lr-one-over-one-minus-fps-X-squared*:  
**shows** *fps-left-inverse* ( $(1 - \text{fps-X})^2$ ) ( $1 :: 'a :: \text{ring-1}$ ) = *Abs-fps* ( $\lambda n. \text{of-nat } (n+1)$ )  
*fps-right-inverse* ( $(1 - \text{fps-X})^2$ ) ( $1 :: 'a$ ) = *Abs-fps* ( $\lambda n. \text{of-nat } (n+1)$ )  
 <proof>

**lemma** *fps-one-over-one-minus-fps-X-squared'*:  
**assumes** *inverse* ( $1 :: 'a :: \{\text{ring-1}, \text{inverse}\}$ ) =  $1$   
**shows** *inverse* ( $(1 - \text{fps-X})^2 :: 'a \text{ fps}$ ) = *Abs-fps* ( $\lambda n. \text{of-nat } (n+1)$ )  
 <proof>

**lemma** *fps-one-over-one-minus-fps-X-squared*:  
*inverse* ( $(1 - \text{fps-X})^2 :: 'a :: \text{division-ring fps}$ ) = *Abs-fps* ( $\lambda n. \text{of-nat } (n+1)$ )  
 <proof>

**lemma** *fps-lr-inverse-fps-X-plus1*:  
*fps-left-inverse* ( $1 + \text{fps-X}$ ) ( $1 :: 'a :: \text{ring-1}$ ) = *Abs-fps* ( $\lambda n. (-1)^n$ )  
*fps-right-inverse* ( $1 + \text{fps-X}$ ) ( $1 :: 'a$ ) = *Abs-fps* ( $\lambda n. (-1)^n$ )  
 <proof>

**lemma** *fps-inverse-fps-X-plus1'*:  
**assumes** *inverse* ( $1 :: 'a :: \{\text{ring-1}, \text{inverse}\}$ ) =  $1$   
**shows** *inverse* ( $1 + \text{fps-X}$ ) = *Abs-fps* ( $\lambda n. (-1)^n$ )  
 <proof>

**lemma** *fps-inverse-fps-X-plus1*:  
*inverse* ( $1 + \text{fps-X}$ ) = *Abs-fps* ( $\lambda n. (-1)^n$ )  
 <proof>

**lemma** *subdegree-lr-inverse*:

**fixes**  $x :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}, \text{uminus}\}$   
**and**  $y :: 'b :: \{\text{ab-group-add}, \text{mult-zero}\}$   
**shows**  $\text{subdegree } (\text{fps-left-inverse } f) x = 0$   
**and**  $\text{subdegree } (\text{fps-right-inverse } g) y = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{subdegree-inverse}$  [simp]:  
**fixes**  $f :: 'a :: \{\text{ab-group-add}, \text{inverse}, \text{mult-zero}\}$   $\text{fps}$   
**shows**  $\text{subdegree } (\text{inverse } f) = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-right-inverse-constructor-rec}$ :  
 $n > 0 \implies \text{fps-right-inverse-constructor } f \ a \ n =$   
 $\quad -a * \text{sum } (\lambda i. \text{fps-nth } f \ i * \text{fps-right-inverse-constructor } f \ a \ (n - i))$   
 $\{1..n\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-right-inverse-constructor-cong}$ :  
**assumes**  $\bigwedge k. k \leq n \implies \text{fps-nth } f \ k = \text{fps-nth } g \ k$   
**shows**  $\text{fps-right-inverse-constructor } f \ c \ n = \text{fps-right-inverse-constructor } g \ c \ n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-cutoff-inverse}$ :  
**fixes**  $f :: 'a :: \text{field}$   $\text{fps}$   
**assumes**  $\text{fps-nth } f \ 0 \neq 0$   
**shows**  $\text{fps-cutoff } n \ (\text{inverse } (\text{fps-cutoff } n \ f)) = \text{fps-cutoff } n \ (\text{inverse } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{tendsto-inverse-fps-aux}$ :  
**fixes**  $f :: 'a :: \text{field}$   $\text{fps}$   
**assumes**  $\text{fps-nth } f \ 0 \neq 0$   
**shows**  $((\lambda f. \text{inverse } f) \longrightarrow \text{inverse } f) \ (at \ f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{tendsto-inverse-fps}$  [tendsto-intros]:  
**fixes**  $g :: 'a :: \text{field}$   $\text{fps}$   
**assumes**  $(f \longrightarrow g) \ F$   
**assumes**  $\text{fps-nth } g \ 0 \neq 0$   
**shows**  $((\lambda x. \text{inverse } (f \ x)) \longrightarrow \text{inverse } g) \ F$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-div-zero}$  [simp]:  
 $0 \text{ div } (g :: 'a :: \{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}) \ \text{fps} = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-div-by-zero'}$ :  
**fixes**  $g :: 'a :: \{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}$   $\text{fps}$   
**assumes**  $\text{inverse } (0 :: 'a) = 0$   
**shows**  $g \text{ div } 0 = 0$

$\langle \text{proof} \rangle$

**lemma** *fps-div-by-zero* [simp]:  $(g :: 'a :: \text{division-ring } \text{fps}) \text{ div } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-unit'*:  $\text{subdegree } g = 0 \implies f \text{ div } g = f * \text{inverse } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-unit*:  $g \$ 0 \neq 0 \implies f \text{ div } g = f * \text{inverse } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-nth-0'*:  
 $\text{subdegree } (g :: 'a :: \text{division-ring } \text{fps}) = 0 \implies (f \text{ div } g) \$ 0 = f \$ 0 / (g \$ 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-nth-0* [simp]:  
 $g \$ 0 \neq 0 \implies (f \text{ div } g) \$ 0 = f \$ 0 / (g \$ 0 :: - :: \text{division-ring})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-nth-below*:  
**fixes**  $f g :: 'a :: \{\text{comm-monoid-add}, \text{uminus}, \text{mult-zero}, \text{inverse}\} \text{fps}$   
**shows**  $n < \text{subdegree } f - \text{subdegree } g \implies (f \text{ div } g) \$ n = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-nth-base*:  
**fixes**  $f g :: 'a :: \text{division-ring } \text{fps}$   
**assumes**  $\text{subdegree } g \leq \text{subdegree } f$   
**shows**  $(f \text{ div } g) \$ (\text{subdegree } f - \text{subdegree } g) = f \$ \text{subdegree } f * \text{inverse } (g \$ \text{subdegree } g)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-subdegree-ge*:  
**fixes**  $f g :: 'a :: \{\text{comm-monoid-add}, \text{uminus}, \text{mult-zero}, \text{inverse}\} \text{fps}$   
**assumes**  $f / g \neq 0$   
**shows**  $\text{subdegree } (f / g) \geq \text{subdegree } f - \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-subdegree*:  
**fixes**  $f g :: 'a :: \text{division-ring } \text{fps}$   
**assumes**  $f \neq 0 \ g \neq 0 \ \text{subdegree } g \leq \text{subdegree } f$   
**shows**  $\text{subdegree } (f / g) = \text{subdegree } f - \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-shift-numer*:  
**fixes**  $f g :: 'a :: \{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\} \text{fps}$   
**assumes**  $n \leq \text{subdegree } f$   
**shows**  $\text{fps-shift } n \ f / g = \text{fps-shift } n \ (f / g)$   
 $\langle \text{proof} \rangle$



**lemma** *fps-divide-shift-denom*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$  *fps*  
**assumes**  $n \leq \text{subdegree } g \leq \text{subdegree } f$   
**shows**  $f / \text{fps-shift } n\ g = \text{Abs-fps } (\lambda k. \text{if } k < n \text{ then } 0 \text{ else } (f/g) \$ (k-n))$   
*<proof>*

**lemma** *fps-divide-unit-factor-numer*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$  *fps*  
**shows**  $\text{unit-factor } f / g = \text{fps-shift } (\text{subdegree } f) (f/g)$   
*<proof>*

**lemma** *fps-divide-unit-factor-denom*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$  *fps*  
**assumes**  $\text{subdegree } g \leq \text{subdegree } f$   
**shows**  
 $f / \text{unit-factor } g = \text{Abs-fps } (\lambda k. \text{if } k < \text{subdegree } g \text{ then } 0 \text{ else } (f/g) \$ (k - \text{subdegree } g))$   
*<proof>*

**lemma** *fps-divide-unit-factor-both'*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$  *fps*  
**assumes**  $\text{subdegree } g \leq \text{subdegree } f$   
**shows**  $\text{unit-factor } f / \text{unit-factor } g = \text{fps-shift } (\text{subdegree } f - \text{subdegree } g) (f / g)$   
*<proof>*

**lemma** *fps-divide-unit-factor-both*:

**fixes**  $f\ g :: 'a::\text{division-ring } \text{fps}$   
**assumes**  $\text{subdegree } g \leq \text{subdegree } f$   
**shows**  $\text{unit-factor } f / \text{unit-factor } g = \text{unit-factor } (f / g)$   
*<proof>*

**lemma** *fps-divide-self*:

$(f::'a::\text{division-ring } \text{fps}) \neq 0 \implies f / f = 1$   
*<proof>*

**lemma** *fps-divide-add*:

**fixes**  $f\ g\ h :: 'a::\{\text{semiring-0}, \text{inverse}, \text{uminus}\}$  *fps*  
**shows**  $(f + g) / h = f / h + g / h$   
*<proof>*

**lemma** *fps-divide-diff*:

**fixes**  $f\ g\ h :: 'a::\{\text{ring}, \text{inverse}\}$  *fps*  
**shows**  $(f - g) / h = f / h - g / h$   
*<proof>*

**lemma** *fps-divide-uminus*:

**fixes**  $f\ g\ h :: 'a::\{\text{ring}, \text{inverse}\}$  *fps*  
**shows**  $(-f) / g = -(f / g)$

$\langle \text{proof} \rangle$

**lemma** *fps-divide-uminus'*:  
 fixes  $f\ g\ h :: 'a::\text{division-ring}\ \text{fps}$   
 shows  $f / (-g) = -(f / g)$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-times*:  
 fixes  $f\ g\ h :: 'a::\{\text{semiring-0},\text{inverse},\text{uminus}\}\ \text{fps}$   
 assumes  $\text{subdegree}\ h \leq \text{subdegree}\ g$   
 shows  $(f * g) / h = f * (g / h)$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-times2*:  
 fixes  $f\ g\ h :: 'a::\{\text{comm-semiring-0},\text{inverse},\text{uminus}\}\ \text{fps}$   
 assumes  $\text{subdegree}\ h \leq \text{subdegree}\ f$   
 shows  $(f * g) / h = (f / h) * g$   
  $\langle \text{proof} \rangle$

**lemma** *fps-times-divide-eq*:  
 fixes  $f\ g :: 'a::\text{field}\ \text{fps}$   
 assumes  $g \neq 0$  and  $\text{subdegree}\ f \geq \text{subdegree}\ g$   
 shows  $f \text{ div } g * g = f$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-times-eq*:  
  $(g :: 'a::\text{division-ring}\ \text{fps}) \neq 0 \implies (f * g) \text{ div } g = f$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-by-mult'*:  
 fixes  $f\ g\ h :: 'a::\text{division-ring}\ \text{fps}$   
 assumes  $\text{subdegree}\ h \leq \text{subdegree}\ f$   
 shows  $f / (g * h) = f / h / g$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-by-mult*:  
 fixes  $f\ g\ h :: 'a::\text{field}\ \text{fps}$   
 assumes  $\text{subdegree}\ g \leq \text{subdegree}\ f$   
 shows  $f / (g * h) = f / g / h$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-cancel*:  
 fixes  $f\ g\ h :: 'a::\text{division-ring}\ \text{fps}$   
 shows  $h \neq 0 \implies (f * h) \text{ div } (g * h) = f \text{ div } g$   
  $\langle \text{proof} \rangle$

**lemma** *fps-divide-1'*:  
 fixes  $a :: 'a::\{\text{comm-monoid-add},\text{inverse},\text{mult-zero},\text{uminus},\text{zero-neq-one},\text{monoid-mult}\}\ \text{fps}$

**assumes**  $\text{inverse } (1 :: 'a) = 1$   
**shows**  $a / 1 = a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-1* [simp]:  $(a :: 'a :: \text{division-ring fps}) / 1 = a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-X'*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}, \text{zero-neq-one}, \text{monoid-mult}\}$   
 $\text{fps}$   
**assumes**  $\text{inverse } (1 :: 'a) = 1$   
**shows**  $f / \text{fps-X} = \text{fps-shift } 1 \ f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-X* [simp]:  $a / \text{fps-X} = \text{fps-shift } 1 \ (a :: 'a :: \text{division-ring fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-X-power'*:  
**fixes**  $f :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\} \text{ fps}$   
**assumes**  $\text{inverse } (1 :: 'a) = 1$   
**shows**  $f / (\text{fps-X} \wedge n) = \text{fps-shift } n \ f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-X-power* [simp]:  $a / (\text{fps-X} \wedge n) = \text{fps-shift } n \ (a :: 'a :: \text{division-ring fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-shift-denom-conv-times-fps-X-power*:  
**fixes**  $f \ g :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\} \text{ fps}$   
**assumes**  $n \leq \text{subdegree } g \ \text{subdegree } g \leq \text{subdegree } f$   
**shows**  $f / \text{fps-shift } n \ g = f / g * \text{fps-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-unit-factor-denom-conv-times-fps-X-power*:  
**fixes**  $f \ g :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\} \text{ fps}$   
**assumes**  $\text{subdegree } g \leq \text{subdegree } f$   
**shows**  $f / \text{unit-factor } g = f / g * \text{fps-X} \wedge \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-altdef'*:  
**fixes**  $f :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\} \text{ fps}$   
**assumes**  $\text{inverse } (1 :: 'a) = 1$   
**shows**  $\text{fps-shift } n \ f = f \text{ div } \text{fps-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-altdef*:  
 $\text{fps-shift } n \ f = (f :: 'a :: \text{division-ring fps}) \text{ div } \text{fps-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-div-fps-X-power-nth'*:

**fixes**  $f :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\}$  *fps*

**assumes**  $\text{inverse } (1 :: 'a) = 1$

**shows**  $(f \text{ div } \text{fps-X}^n) \$ k = f \$ (k + n)$

*<proof>*

**lemma** *fps-div-fps-X-power-nth*:  $((f :: 'a :: \text{division-ring fps}) \text{ div } \text{fps-X}^n) \$ k = f \$ (k + n)$

*<proof>*

**lemma** *fps-div-fps-X-nth'*:

**fixes**  $f :: 'a :: \{\text{semiring-1}, \text{inverse}, \text{uminus}\}$  *fps*

**assumes**  $\text{inverse } (1 :: 'a) = 1$

**shows**  $(f \text{ div } \text{fps-X}) \$ k = f \$ \text{Suc } k$

*<proof>*

**lemma** *fps-div-fps-X-nth*:  $((f :: 'a :: \text{division-ring fps}) \text{ div } \text{fps-X}) \$ k = f \$ \text{Suc } k$

*<proof>*

**lemma** *divide-fps-const'*:

**fixes**  $c :: 'a :: \{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$

**shows**  $f / \text{fps-const } c = f * \text{fps-const } (\text{inverse } c)$

*<proof>*

**lemma** *divide-fps-const [simp]*:

**fixes**  $c :: 'a :: \{\text{comm-semiring-0}, \text{inverse}, \text{uminus}\}$

**shows**  $f / \text{fps-const } c = \text{fps-const } (\text{inverse } c) * f$

*<proof>*

**lemma** *fps-const-divide*:  $\text{fps-const } (x :: - :: \text{division-ring}) / \text{fps-const } y = \text{fps-const } (x / y)$

*<proof>*

**lemma** *fps-numeral-divide-divide*:

$x / \text{numeral } b / \text{numeral } c = (x / \text{numeral } (b * c)) :: 'a :: \text{field fps}$

*<proof>*

**lemma** *fps-numeral-mult-divide*:

$\text{numeral } b * x / \text{numeral } c = (\text{numeral } b / \text{numeral } c * x) :: 'a :: \text{field fps}$

*<proof>*

**lemmas** *fps-numeral-simps* =

*fps-numeral-divide-divide fps-numeral-mult-divide inverse-fps-numeral neg-numeral-fps-const*

**lemma** *fps-is-left-unit-iff-zeroth-is-left-unit*:

**fixes**  $f :: 'a :: \text{ring-1 fps}$

**shows**  $(\exists g. 1 = f * g) \longleftrightarrow (\exists k. 1 = f \$ 0 * k)$

*<proof>*

**lemma** *fps-is-right-unit-iff-zeroth-is-right-unit*:  
**fixes**  $f :: 'a :: \text{ring-1 } \text{fps}$   
**shows**  $(\exists g. 1 = g * f) \longleftrightarrow (\exists k. 1 = k * f \$ 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-is-unit-iff* [simp]:  $(f :: 'a :: \text{field } \text{fps}) \text{ dvd } 1 \longleftrightarrow f \$ 0 \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-eq-0-left*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, zero-neq-one, mult-zero}\} \text{fps}$   
**assumes**  $\exists g. 1 = f * g$   
**shows**  $\text{subdegree } f = 0$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-eq-0-right*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, zero-neq-one, mult-zero}\} \text{fps}$   
**assumes**  $\exists g. 1 = g * f$   
**shows**  $\text{subdegree } f = 0$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-eq-0'* [simp]:  $(f :: 'a :: \text{field } \text{fps}) \text{ dvd } 1 \implies \text{subdegree } f = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-dvd1-left-trivial-unit-factor*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, zero-neq-one, mult-zero}\} \text{fps}$   
**assumes**  $\exists g. 1 = f * g$   
**shows**  $\text{unit-factor } f = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-dvd1-right-trivial-unit-factor*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, zero-neq-one, mult-zero}\} \text{fps}$   
**assumes**  $\exists g. 1 = g * f$   
**shows**  $\text{unit-factor } f = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-dvd1-trivial-unit-factor*:  
 $(f :: 'a :: \text{comm-semiring-1 } \text{fps}) \text{ dvd } 1 \implies \text{unit-factor } f = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-dvd-left*:  
**fixes**  $f :: 'a :: \text{division-ring } \text{fps}$   
**assumes**  $f \$ 0 \neq 0$   
**shows**  $\exists g. 1 = f * g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-unit-dvd-right*:  
**fixes**  $f :: 'a :: \text{division-ring } \text{fps}$   
**assumes**  $f \$ 0 \neq 0$   
**shows**  $\exists g. 1 = g * f$

$\langle \text{proof} \rangle$

**lemma** *fps-unit-dvd [simp]:*  $(f \ \$ \ 0 :: 'a :: \text{field}) \neq 0 \implies f \ \text{dvd} \ g$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-left-imp-subdegree-le:*  
**fixes**  $f \ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\} \ \text{fps}$   
**assumes**  $\exists k. g = f * k \ g \neq 0$   
**shows**  $\text{subdegree } f \leq \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-right-imp-subdegree-le:*  
**fixes**  $f \ g :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}\} \ \text{fps}$   
**assumes**  $\exists k. g = k * f \ g \neq 0$   
**shows**  $\text{subdegree } f \leq \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *dvd-imp-subdegree-le:*  
 $f \ \text{dvd} \ g \implies g \neq 0 \implies \text{subdegree } f \leq \text{subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-le-imp-dvd-left-ring1:*  
**fixes**  $f \ g :: 'a :: \text{ring-1} \ \text{fps}$   
**assumes**  $\exists y. f \ \$ \ \text{subdegree } f * y = 1 \ \text{subdegree } f \leq \text{subdegree } g$   
**shows**  $\exists k. g = f * k$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-le-imp-dvd-left-divring:*  
**fixes**  $f \ g :: 'a :: \text{division-ring} \ \text{fps}$   
**assumes**  $f \neq 0 \ \text{subdegree } f \leq \text{subdegree } g$   
**shows**  $\exists k. g = f * k$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-le-imp-dvd-right-ring1:*  
**fixes**  $f \ g :: 'a :: \text{ring-1} \ \text{fps}$   
**assumes**  $\exists x. x * f \ \$ \ \text{subdegree } f = 1 \ \text{subdegree } f \leq \text{subdegree } g$   
**shows**  $\exists k. g = k * f$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-le-imp-dvd-right-divring:*  
**fixes**  $f \ g :: 'a :: \text{division-ring} \ \text{fps}$   
**assumes**  $f \neq 0 \ \text{subdegree } f \leq \text{subdegree } g$   
**shows**  $\exists k. g = k * f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-dvd-iff:*  
**assumes**  $(f :: 'a :: \text{field} \ \text{fps}) \neq 0 \ g \neq 0$   
**shows**  $f \ \text{dvd} \ g \longleftrightarrow \text{subdegree } f \leq \text{subdegree } g$   
 $\langle \text{proof} \rangle$

```

lemma subdegree-div':
  fixes   p q :: 'a::division-ring fps
  assumes  $\exists k. p = k * q$ 
  shows    $\text{subdegree } (p \text{ div } q) = \text{subdegree } p - \text{subdegree } q$ 
  <proof>

lemma subdegree-div:
  fixes   p q :: 'a :: field fps
  assumes   q dvd p
  shows      $\text{subdegree } (p \text{ div } q) = \text{subdegree } p - \text{subdegree } q$ 
  <proof>

lemma subdegree-div-unit':
  fixes   p q :: 'a :: {ab-group-add,mult-zero,inverse} fps
  assumes  $q \ \$ \ 0 \neq 0 \ \& \ p \ \$ \ \text{subdegree } p * \text{inverse } (q \ \$ \ 0) \neq 0$ 
  shows    $\text{subdegree } (p \text{ div } q) = \text{subdegree } p$ 
  <proof>

lemma subdegree-div-unit'':
  fixes   p q :: 'a :: {ring-no-zero-divisors,inverse} fps
  assumes  $q \ \$ \ 0 \neq 0 \ \& \ \text{inverse } (q \ \$ \ 0) \neq 0$ 
  shows    $\text{subdegree } (p \text{ div } q) = \text{subdegree } p$ 
  <proof>

lemma subdegree-div-unit:
  fixes   p q :: 'a :: division-ring fps
  assumes  $q \ \$ \ 0 \neq 0$ 
  shows    $\text{subdegree } (p \text{ div } q) = \text{subdegree } p$ 
  <proof>

instantiation fps :: ({comm-semiring-1,inverse,uminus}) modulo
begin

definition fps-mod-def:
  f mod g = (if g = 0 then f else
    let h = unit-factor g in fps-cutoff (subdegree g) (f * inverse h) * h)

instance <proof>

end

lemma fps-mod-zero [simp]:
  (f::'a::{comm-semiring-1,inverse,uminus} fps) mod 0 = f
  <proof>

lemma fps-mod-eq-zero:
  assumes  $g \neq 0$  and  $\text{subdegree } f \geq \text{subdegree } g$ 
  shows    $f \text{ mod } g = 0$ 

```

$\langle \text{proof} \rangle$

**lemma** *fps-mod-unit* [simp]:  $g \neq 0 \implies f \bmod g = 0$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-mod*:  
  **assumes**  $\text{subdegree } (f::'a::\text{field } \text{fps}) < \text{subdegree } g$   
  **shows**  $\text{subdegree } (f \bmod g) = \text{subdegree } f$   
 $\langle \text{proof} \rangle$

**instance** *fps* :: (field) idom-modulo  
 $\langle \text{proof} \rangle$

**instantiation** *fps* :: (field) normalization-semidom-multiplicative  
**begin**

**definition** *fps-normalize-def* [simp]:  
   $\text{normalize } f = (\text{if } f = 0 \text{ then } 0 \text{ else } \text{fps-X}^{\wedge} \text{subdegree } f)$

**instance**  $\langle \text{proof} \rangle$

**end**

## 5.8 Computing reciprocals via Hensel lifting

**lemma** *inverse-fps-hensel-lifting*:  
  **fixes**  $F \ G :: 'a :: \text{field } \text{fps}$  **and**  $n :: \text{nat}$   
  **assumes**  $G \text{-eq: } \text{fps-cutoff } n \ G = \text{fps-cutoff } n \ (\text{inverse } F)$   
  **assumes**  $\text{unit: } \text{fps-nth } F \ 0 \neq 0$   
  **shows**  $\text{fps-cutoff } (2*n) \ (\text{inverse } F) = \text{fps-cutoff } (2*n) \ (G * (2 - F * G))$   
 $\langle \text{proof} \rangle$

**lemma** *inverse-fps-hensel-lifting'*:  
  **fixes**  $F \ G :: 'a :: \text{field } \text{fps}$  **and**  $n :: \text{nat}$   
  **assumes**  $G \text{-eq: } \text{fps-cutoff } n \ G = \text{fps-cutoff } n \ (\text{inverse } F)$   
  **assumes**  $\text{unit: } \text{fps-nth } F \ 0 \neq 0$   
  **defines**  $P \equiv \text{fps-shift } n \ (F * G - 1)$   
  **shows**  $\text{fps-cutoff } (2*n) \ (\text{inverse } F) = \text{fps-cutoff } (2*n) \ (G * (1 - \text{fps-X}^{\wedge} n * P))$   
 $\langle \text{proof} \rangle$

## 5.9 Euclidean division

**instantiation** *fps* :: (field) euclidean-ring-cancel  
**begin**

**definition** *fps-euclidean-size-def*:  
   $\text{euclidean-size } f = (\text{if } f = 0 \text{ then } 0 \text{ else } 2^{\wedge} \text{subdegree } f)$

**instance**  $\langle \text{proof} \rangle$



```

end

instance fps :: (field) normalization-euclidean-semiring ⟨proof⟩

instantiation fps :: (field) euclidean-ring-gcd
begin
definition fps-gcd-def: (gcd :: 'a fps ⇒ -) = Euclidean-Algorithm.gcd
definition fps-lcm-def: (lcm :: 'a fps ⇒ -) = Euclidean-Algorithm.lcm
definition fps-Gcd-def: (Gcd :: 'a fps set ⇒ -) = Euclidean-Algorithm.Gcd
definition fps-Lcm-def: (Lcm :: 'a fps set ⇒ -) = Euclidean-Algorithm.Lcm
instance ⟨proof⟩
end

lemma fps-gcd:
  assumes [simp]: f ≠ 0 g ≠ 0
  shows gcd f g = fps-X ^ min (subdegree f) (subdegree g)
  ⟨proof⟩

lemma fps-gcd-altdef: gcd f g =
  (if f = 0 ∧ g = 0 then 0 else
   if f = 0 then fps-X ^ subdegree g else
   if g = 0 then fps-X ^ subdegree f else
   fps-X ^ min (subdegree f) (subdegree g))
  ⟨proof⟩

lemma fps-lcm:
  assumes [simp]: f ≠ 0 g ≠ 0
  shows lcm f g = fps-X ^ max (subdegree f) (subdegree g)
  ⟨proof⟩

lemma fps-lcm-altdef: lcm f g =
  (if f = 0 ∨ g = 0 then 0 else fps-X ^ max (subdegree f) (subdegree g))
  ⟨proof⟩

lemma fps-Gcd:
  assumes A - {0} ≠ {}
  shows Gcd A = fps-X ^ (INF f∈A-{0}. subdegree f)
  ⟨proof⟩

lemma fps-Gcd-altdef: Gcd A =
  (if A ⊆ {0} then 0 else fps-X ^ (INF f∈A-{0}. subdegree f))
  ⟨proof⟩

lemma fps-Lcm:
  assumes A ≠ {} 0 ∉ A bdd-above (subdegree `A)
  shows Lcm A = fps-X ^ (SUP f∈A. subdegree f)
  ⟨proof⟩

```

**lemma** *fps-Lcm-altdef*:

$Lcm\ A =$   
 (if  $0 \in A \vee \neg bdd\text{-}above\ (subdegree\ 'A)$  then 0 else  
 if  $A = \{\}$  then 1 else  $fps\text{-}X \wedge (SUP\ f \in A.\ subdegree\ f)$ )  
 $\langle proof \rangle$

## 5.10 Formal Derivatives

**definition** *fps-deriv*  $f = Abs\text{-}fps\ (\lambda n.\ of\text{-}nat\ (n + 1) * f\ \$\ (n + 1))$

**lemma** *fps-deriv-nth[simp]*:  $fps\text{-}deriv\ f\ \$\ n = of\text{-}nat\ (n + 1) * f\ \$\ (n + 1)$   
 $\langle proof \rangle$

**lemma** *fps-0th-higher-deriv*:

$(fps\text{-}deriv\ \frown n)\ f\ \$\ 0 = fact\ n * f\ \$\ n$   
 $\langle proof \rangle$

**lemma** *fps-deriv-mult[simp]*:

$fps\text{-}deriv\ (f * g) = f * fps\text{-}deriv\ g + fps\text{-}deriv\ f * g$   
 $\langle proof \rangle$

**lemma** *fps-deriv-fps-X[simp]*:  $fps\text{-}deriv\ fps\text{-}X = 1$

$\langle proof \rangle$

**lemma** *fps-deriv-neg[simp]*:

$fps\text{-}deriv\ (-\ (f::\ 'a::ring\text{-}1\ fps)) = -\ (fps\text{-}deriv\ f)$   
 $\langle proof \rangle$

**lemma** *fps-deriv-add[simp]*:  $fps\text{-}deriv\ (f + g) = fps\text{-}deriv\ f + fps\text{-}deriv\ g$

$\langle proof \rangle$

**lemma** *fps-deriv-sub[simp]*:

$fps\text{-}deriv\ ((f::\ 'a::ring\text{-}1\ fps) - g) = fps\text{-}deriv\ f - fps\text{-}deriv\ g$   
 $\langle proof \rangle$

**lemma** *fps-deriv-const[simp]*:  $fps\text{-}deriv\ (fps\text{-}const\ c) = 0$

$\langle proof \rangle$

**lemma** *fps-deriv-of-nat [simp]*:  $fps\text{-}deriv\ (of\text{-}nat\ n) = 0$

$\langle proof \rangle$

**lemma** *fps-deriv-of-int [simp]*:  $fps\text{-}deriv\ (of\text{-}int\ n) = 0$

$\langle proof \rangle$

**lemma** *fps-deriv-numeral [simp]*:  $fps\text{-}deriv\ (numeral\ n) = 0$

$\langle proof \rangle$

**lemma** *fps-deriv-mult-const-left[simp]*:

$fps\text{-}deriv\ (fps\text{-}const\ c * f) = fps\text{-}const\ c * fps\text{-}deriv\ f$

$\langle \text{proof} \rangle$

**lemma** *fps-deriv-linear*[simp]:

$\text{fps-deriv } (\text{fps-const } a * f + \text{fps-const } b * g) =$   
 $\text{fps-const } a * \text{fps-deriv } f + \text{fps-const } b * \text{fps-deriv } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-0*[simp]:  $\text{fps-deriv } 0 = 0$

$\langle \text{proof} \rangle$

**lemma** *fps-deriv-1*[simp]:  $\text{fps-deriv } 1 = 0$

$\langle \text{proof} \rangle$

**lemma** *fps-deriv-mult-const-right*[simp]:

$\text{fps-deriv } (f * \text{fps-const } c) = \text{fps-deriv } f * \text{fps-const } c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-sum*:

$\text{fps-deriv } (\text{sum } f \ S) = \text{sum } (\lambda i. \text{fps-deriv } (f \ i)) \ S$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-eq-0-iff* [simp]:

$\text{fps-deriv } f = 0 \iff f = \text{fps-const } (f\$0 :: 'a::\{\text{semiring-no-zero-divisors}, \text{semiring-char-0}\})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-eq-iff*:

**fixes**  $f \ g :: 'a::\{\text{ring-1-no-zero-divisors}, \text{semiring-char-0}\}$  *fps*  
**shows**  $\text{fps-deriv } f = \text{fps-deriv } g \iff (f = \text{fps-const}(f\$0 - g\$0) + g)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-eq-iff-ex*:

**fixes**  $f \ g :: 'a::\{\text{ring-1-no-zero-divisors}, \text{semiring-char-0}\}$  *fps*  
**shows**  $(\text{fps-deriv } f = \text{fps-deriv } g) \iff (\exists c. f = \text{fps-const } c + g)$   
 $\langle \text{proof} \rangle$

**fun** *fps-nth-deriv* ::  $\text{nat} \Rightarrow 'a::\text{semiring-1}$  *fps*  $\Rightarrow 'a$  *fps*

**where**

$\text{fps-nth-deriv } 0 \ f = f$   
 $|\ \text{fps-nth-deriv } (\text{Suc } n) \ f = \text{fps-nth-deriv } n \ (\text{fps-deriv } f)$

**lemma** *fps-nth-deriv-commute*:  $\text{fps-nth-deriv } (\text{Suc } n) \ f = \text{fps-deriv } (\text{fps-nth-deriv } n \ f)$

$\langle \text{proof} \rangle$

**lemma** *fps-nth-deriv-linear*[simp]:

$\text{fps-nth-deriv } n \ (\text{fps-const } a * f + \text{fps-const } b * g) =$   
 $\text{fps-const } a * \text{fps-nth-deriv } n \ f + \text{fps-const } b * \text{fps-nth-deriv } n \ g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-nth-deriv-neg*[simp]:

$$\text{fps-nth-deriv } n \text{ } (- (f :: 'a::\text{ring-1 } \text{fps})) = - (\text{fps-nth-deriv } n \text{ } f)$$

*<proof>*

**lemma** *fps-nth-deriv-add*[simp]:

$$\text{fps-nth-deriv } n \text{ } ((f :: 'a::\text{ring-1 } \text{fps}) + g) = \text{fps-nth-deriv } n \text{ } f + \text{fps-nth-deriv } n \text{ } g$$

*<proof>*

**lemma** *fps-nth-deriv-sub*[simp]:

$$\text{fps-nth-deriv } n \text{ } ((f :: 'a::\text{ring-1 } \text{fps}) - g) = \text{fps-nth-deriv } n \text{ } f - \text{fps-nth-deriv } n \text{ } g$$

*<proof>*

**lemma** *fps-nth-deriv-0*[simp]: *fps-nth-deriv* *n* 0 = 0

*<proof>*

**lemma** *fps-nth-deriv-1*[simp]: *fps-nth-deriv* *n* 1 = (if *n* = 0 then 1 else 0)

*<proof>*

**lemma** *fps-nth-deriv-const*[simp]:

$$\text{fps-nth-deriv } n \text{ } (\text{fps-const } c) = (\text{if } n = 0 \text{ then } \text{fps-const } c \text{ else } 0)$$

*<proof>*

**lemma** *fps-nth-deriv-mult-const-left*[simp]:

$$\text{fps-nth-deriv } n \text{ } (\text{fps-const } c * f) = \text{fps-const } c * \text{fps-nth-deriv } n \text{ } f$$

*<proof>*

**lemma** *fps-nth-deriv-mult-const-right*[simp]:

$$\text{fps-nth-deriv } n \text{ } (f * \text{fps-const } c) = \text{fps-nth-deriv } n \text{ } f * \text{fps-const } c$$

*<proof>*

**lemma** *fps-nth-deriv-sum*:

$$\text{fps-nth-deriv } n \text{ } (\text{sum } f \text{ } S) = \text{sum } (\lambda i. \text{fps-nth-deriv } n \text{ } (f \text{ } i :: 'a::\text{ring-1 } \text{fps})) \text{ } S$$

*<proof>*

**lemma** *fps-deriv-maclauren-0*:

$$(\text{fps-nth-deriv } k \text{ } (f :: 'a::\text{comm-semiring-1 } \text{fps})) \$ 0 = \text{of-nat } (\text{fact } k) * f \$ k$$

*<proof>*

**lemma** *fps-deriv-lr-inverse*:

**fixes** *x y* :: '*a*::ring-1

**assumes** *x* \* *f*\$0 = 1 *f*\$0 \* *y* = 1

— These assumptions imply *x* equals *y*, but no need to assume that.

**shows** *fps-deriv* (*fps-left-inverse* *f* *x*) =

$$- \text{fps-left-inverse } f \text{ } x * \text{fps-deriv } f * \text{fps-left-inverse } f \text{ } x$$

**and** *fps-deriv* (*fps-right-inverse* *f* *y*) =

$$- \text{fps-right-inverse } f \text{ } y * \text{fps-deriv } f * \text{fps-right-inverse } f \text{ } y$$

*<proof>*

**lemma** *fps-deriv-lr-inverse-comm*:  
**fixes**  $x :: 'a::comm-ring-1$   
**assumes**  $x * f\$0 = 1$   
**shows**  $fps-deriv (fps-left-inverse f x) = - fps-deriv f * (fps-left-inverse f x)^2$   
**and**  $fps-deriv (fps-right-inverse f x) = - fps-deriv f * (fps-right-inverse f x)^2$   
 $\langle proof \rangle$

**lemma** *fps-inverse-deriv-divring*:  
**fixes**  $a :: 'a::division-ring fps$   
**assumes**  $a\$0 \neq 0$   
**shows**  $fps-deriv (inverse a) = - inverse a * fps-deriv a * inverse a$   
 $\langle proof \rangle$

**lemma** *fps-inverse-deriv*:  
**fixes**  $a :: 'a::field fps$   
**assumes**  $a\$0 \neq 0$   
**shows**  $fps-deriv (inverse a) = - fps-deriv a * (inverse a)^2$   
 $\langle proof \rangle$

**lemma** *fps-inverse-deriv'*:  
**fixes**  $a :: 'a::field fps$   
**assumes**  $a\$0 : a \$ 0 \neq 0$   
**shows**  $fps-deriv (inverse a) = - fps-deriv a / a^2$   
 $\langle proof \rangle$

**lemma** *fps-divide-deriv*:  
**assumes**  $b \text{ dvd } (a :: 'a :: field fps)$   
**shows**  $fps-deriv (a / b) = (fps-deriv a * b - a * fps-deriv b) / b^2$   
 $\langle proof \rangle$

**lemma** *fps-nth-deriv-fps-X[simp]*:  $fps-nth-deriv n fps-X = (if n = 0 then fps-X else if n=1 then 1 else 0)$   
 $\langle proof \rangle$

## 5.11 Powers

**lemma** *fps-power-zeroth*:  $(a^{\wedge}n) \$ 0 = (a\$0)^{\wedge}n$   
 $\langle proof \rangle$

**lemma** *fps-power-zeroth-eq-one*:  $a\$0 = 1 \implies a^{\wedge}n \$ 0 = 1$   
 $\langle proof \rangle$

**lemma** *fps-power-first*:  
**fixes**  $a :: 'a::comm-semiring-1 fps$   
**shows**  $(a^{\wedge}n) \$ 1 = of-nat n * (a\$0)^{\wedge}(n-1) * a\$1$   
 $\langle proof \rangle$

**lemma** *fps-power-first-eq*:  $a \$ 0 = 1 \implies a^{\wedge}n \$ 1 = of-nat n * a\$1$

$\langle proof \rangle$

**lemma** *fps-power-first-eq'*:

**assumes**  $a \$ 1 = 1$

**shows**  $a \wedge n \$ 1 = of\_nat\ n * (a \$ 0) \wedge (n-1)$

$\langle proof \rangle$

**lemmas** *startsby-one-power* = *fps-power-zeroth-eq-one*

**lemma** *startsby-zero-power*:  $a \$ 0 = 0 \implies n > 0 \implies a \wedge n \$ 0 = 0$

$\langle proof \rangle$

**lemma** *startsby-power*:  $a \$ 0 = v \implies a \wedge n \$ 0 = v \wedge n$

$\langle proof \rangle$

**lemma** *startsby-nonzero-power*:

**fixes**  $a :: 'a::semiring-1-no-zero-divisors\ fps$

**shows**  $a \$ 0 \neq 0 \implies a \wedge n \$ 0 \neq 0$

$\langle proof \rangle$

**lemma** *startsby-zero-power-iff*[*simp*]:

$a \wedge n \$ 0 = (0 :: 'a::semiring-1-no-zero-divisors) \longleftrightarrow n \neq 0 \wedge a \$ 0 = 0$

$\langle proof \rangle$

**lemma** *startsby-zero-power-prefix*:

**assumes**  $a0: a \$ 0 = 0$

**shows**  $\forall n < k. a \wedge k \$ n = 0$

$\langle proof \rangle$

**lemma** *startsby-zero-sum-depends*:

**assumes**  $a0: a \$ 0 = 0$

**and**  $kn: n \geq k$

**shows**  $sum\ (\lambda i. (a \wedge i) \$ k)\ \{0 .. n\} = sum\ (\lambda i. (a \wedge i) \$ k)\ \{0 .. k\}$

$\langle proof \rangle$

**lemma** *startsby-zero-power-nth-same*:

**assumes**  $a0: a \$ 0 = 0$

**shows**  $a \wedge n \$ n = (a \$ 1) \wedge n$

$\langle proof \rangle$

**lemma** *fps-lr-inverse-power*:

**fixes**  $a :: 'a::ring-1\ fps$

**assumes**  $x * a \$ 0 = 1\ a \$ 0 * x = 1$

**shows**  $fps\_left\_inverse\ (a \wedge n)\ (x \wedge n) = fps\_left\_inverse\ a\ x \wedge n$

**and**  $fps\_right\_inverse\ (a \wedge n)\ (x \wedge n) = fps\_right\_inverse\ a\ x \wedge n$

$\langle proof \rangle$

**lemma** *fps-inverse-power*:

**fixes**  $a :: 'a::division-ring\ fps$

**shows**  $\text{inverse } (a \wedge n) = \text{inverse } a \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-power'*:  
**fixes**  $a :: 'a::\text{comm-semiring-1}$  *fps*  
**shows**  $\text{fps-deriv } (a \wedge n) = (\text{of-nat } n) * \text{fps-deriv } a * a \wedge (n - 1)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-power*:  
**fixes**  $a :: 'a::\text{comm-semiring-1}$  *fps*  
**shows**  $\text{fps-deriv } (a \wedge n) = \text{fps-const } (\text{of-nat } n) * \text{fps-deriv } a * a \wedge (n - 1)$   
 $\langle \text{proof} \rangle$

## 5.12 Finite and infinite products

**lemma** *fps-prod-nth'*:  
**assumes** *finite A*  
**shows**  $\text{fps-nth } (\prod x \in A. f\ x)\ n = (\sum X \in \text{multisets-of-size } A\ n. \prod x \in A. \text{fps-nth } (f\ x)\ (\text{count } X\ x))$   
 $\langle \text{proof} \rangle$

**theorem** *tendsto-prod-fps*:  
**fixes**  $f :: \text{nat} \Rightarrow 'a :: \{\text{idom},\ \text{t2-space}\}$  *fps*  
**assumes**  $[\text{simp}]: \bigwedge k. f\ k \neq 0$   
**assumes**  $g: \bigwedge n\ k. k > g\ n \implies \text{subdegree } (f\ k - 1) > n$   
**defines**  $P \equiv \text{Abs-fps } (\lambda n. (\sum X \in \text{multisets-of-size } \{..g\ n\}\ n. \prod i \leq g\ n. \text{fps-nth } (f\ i)\ (\text{count } X\ i)))$   
**shows**  $(\lambda n. \prod k \leq n. f\ k) \longrightarrow P$   
 $\langle \text{proof} \rangle$

## 5.13 Integration

**definition** *fps-integral*  $:: 'a::\{\text{semiring-1},\ \text{inverse}\}$  *fps*  $\Rightarrow 'a \Rightarrow 'a$  *fps*  
**where**  $\text{fps-integral } a\ a0 =$   
 $\text{Abs-fps } (\lambda n. \text{if } n=0 \text{ then } a0 \text{ else } \text{inverse } (\text{of-nat } n) * a \$ (n - 1))$

**abbreviation** *fps-integral0*  $a \equiv \text{fps-integral } a\ 0$

**lemma** *fps-integral-nth-0-Suc*  $[\text{simp}]$ :  
**fixes**  $a :: 'a::\{\text{semiring-1},\ \text{inverse}\}$  *fps*  
**shows**  $\text{fps-integral } a\ a0\ \$\ 0 = a0$   
**and**  $\text{fps-integral } a\ a0\ \$\ \text{Suc } n = \text{inverse } (\text{of-nat } (\text{Suc } n)) * a\ \$\ n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral-conv-plus-const*:  
 $\text{fps-integral } a\ a0 = \text{fps-integral } a\ 0 + \text{fps-const } a0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-deriv-fps-integral*:  
**fixes**  $a :: 'a::\{\text{division-ring},\ \text{ring-char-0}\}$  *fps*

**shows**  $\text{fps-deriv } (\text{fps-integral } a \ a0) = a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-deriv}$ :  
**fixes**  $a :: 'a::\{\text{division-ring}, \text{ring-char-0}\}$   $\text{fps}$   
**shows**  $\text{fps-integral0 } (\text{fps-deriv } a) = a - \text{fps-const } (a\$0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral-deriv}$ :  
**fixes**  $a :: 'a::\{\text{division-ring}, \text{ring-char-0}\}$   $\text{fps}$   
**shows**  $\text{fps-integral } (\text{fps-deriv } a) \ (a\$0) = a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-zero}$ :  
 $\text{fps-integral0 } (0::'a::\{\text{semiring-1}, \text{inverse}\}) \ \text{fps} = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-fps-const}'$ :  
**fixes**  $c :: 'a::\{\text{semiring-1}, \text{inverse}\}$   
**assumes**  $\text{inverse } (1::'a) = 1$   
**shows**  $\text{fps-integral0 } (\text{fps-const } c) = \text{fps-const } c * \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-fps-const}$ :  
**fixes**  $c :: 'a::\text{division-ring}$   
**shows**  $\text{fps-integral0 } (\text{fps-const } c) = \text{fps-const } c * \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-one}'$ :  
**assumes**  $\text{inverse } (1::'a::\{\text{semiring-1}, \text{inverse}\}) = 1$   
**shows**  $\text{fps-integral0 } (1::'a \ \text{fps}) = \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-one}$ :  
 $\text{fps-integral0 } (1::'a::\text{division-ring} \ \text{fps}) = \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-fps-const-mult-left}$ :  
**fixes**  $a :: 'a::\text{division-ring} \ \text{fps}$   
**shows**  $\text{fps-integral0 } (\text{fps-const } c * a) = \text{fps-const } c * \text{fps-integral0 } a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-fps-const-mult-right}$ :  
**fixes**  $a :: 'a::\{\text{semiring-1}, \text{inverse}\} \ \text{fps}$   
**shows**  $\text{fps-integral0 } (a * \text{fps-const } c) = \text{fps-integral0 } a * \text{fps-const } c$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-integral0-neg}$ :  
**fixes**  $a :: 'a::\{\text{ring-1}, \text{inverse}\} \ \text{fps}$



**shows**  $\text{fps-integral0 } (-a) = - \text{fps-integral0 } a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-add*:  
 $\text{fps-integral0 } (a+b) = \text{fps-integral0 } a + \text{fps-integral0 } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-linear*:  
**fixes**  $a \ b :: 'a::\text{division-ring}$   
**shows**  $\text{fps-integral0 } (\text{fps-const } a * f + \text{fps-const } b * g) =$   
 $\text{fps-const } a * \text{fps-integral0 } f + \text{fps-const } b * \text{fps-integral0 } g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-linear2*:  
 $\text{fps-integral0 } (f * \text{fps-const } a + g * \text{fps-const } b) =$   
 $\text{fps-integral0 } f * \text{fps-const } a + \text{fps-integral0 } g * \text{fps-const } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral-linear*:  
**fixes**  $a \ b \ a0 \ b0 :: 'a::\text{division-ring}$   
**shows**  
 $\text{fps-integral } (\text{fps-const } a * f + \text{fps-const } b * g) (a*a0 + b*b0) =$   
 $\text{fps-const } a * \text{fps-integral } f \ a0 + \text{fps-const } b * \text{fps-integral } g \ b0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-sub*:  
**fixes**  $a \ b :: 'a::\{\text{ring-1}, \text{inverse}\} \text{fps}$   
**shows**  $\text{fps-integral0 } (a-b) = \text{fps-integral0 } a - \text{fps-integral0 } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-of-nat*:  
 $\text{fps-integral0 } (\text{of-nat } n :: 'a::\text{division-ring fps}) = \text{of-nat } n * \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-sum*:  
 $\text{fps-integral0 } (\text{sum } f \ S) = \text{sum } (\lambda i. \text{fps-integral0 } (f \ i)) \ S$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-by-parts*:  
**fixes**  $a \ b :: 'a::\{\text{division-ring}, \text{ring-char-0}\} \text{fps}$   
**shows**  
 $\text{fps-integral0 } (a * b) =$   
 $a * \text{fps-integral0 } b - \text{fps-integral0 } (\text{fps-deriv } a * \text{fps-integral0 } b)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-fps-X*:  
 $\text{fps-integral0 } (\text{fps-X} :: 'a::\{\text{semiring-1}, \text{inverse}\} \text{fps}) =$   
 $\text{fps-const } (\text{inverse } (\text{of-nat } 2)) * \text{fps-X}^2$   
 $\langle \text{proof} \rangle$

**lemma** *fps-integral0-fps-X-power*:  

$$\text{fps-integral0 } ((\text{fps-X} :: 'a :: \{\text{semiring-1}, \text{inverse}\} \text{ fps}) \hat{\ } n) =$$

$$\text{fps-const } (\text{inverse } (\text{of-nat } (\text{Suc } n))) * \text{fps-X} \hat{\ } \text{Suc } n$$
 $\langle \text{proof} \rangle$

## 5.14 Composition

**definition** *fps-compose* ::  $'a :: \text{semiring-1} \text{ fps} \Rightarrow 'a \text{ fps} \Rightarrow 'a \text{ fps}$  (**infixl**  $\langle \text{oo} \rangle$  55)  
**where**  $a \text{ oo } b = \text{Abs-fps } (\lambda n. \text{sum } (\lambda i. a \$ i * (b \hat{\ } i \$ n)) \{0..n\})$

**lemma** *fps-compose-nth*:  $(a \text{ oo } b) \$ n = \text{sum } (\lambda i. a \$ i * (b \hat{\ } i \$ n)) \{0..n\}$   
 $\langle \text{proof} \rangle$

**lemma** *fps-compose-nth-0 [simp]*:  $(f \text{ oo } g) \$ 0 = f \$ 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-compose-fps-X [simp]*:  $a \text{ oo fps-X} = (a :: 'a :: \text{comm-ring-1} \text{ fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-const-compose [simp]*:  $\text{fps-const } (a :: 'a :: \text{comm-ring-1}) \text{ oo } b = \text{fps-const } a$   
 $\langle \text{proof} \rangle$

**lemma** *numeral-compose [simp]*:  $(\text{numeral } k :: 'a :: \text{comm-ring-1} \text{ fps}) \text{ oo } b = \text{numeral } k$   
 $\langle \text{proof} \rangle$

**lemma** *neg-numeral-compose [simp]*:  $(- \text{ numeral } k :: 'a :: \text{comm-ring-1} \text{ fps}) \text{ oo } b = - \text{ numeral } k$   
 $\langle \text{proof} \rangle$

**lemma** *fps-X-fps-compose-startby0 [simp]*:  $a \$ 0 = 0 \implies \text{fps-X oo } a = (a :: 'a :: \text{comm-ring-1} \text{ fps})$   
 $\langle \text{proof} \rangle$

## 5.15 Rules from Herbert Wilf's Generatingfunctionology

### 5.15.1 Rule 1

**lemma** *fps-power-mult-eq-shift*:  

$$\text{fps-X} \hat{\ } \text{Suc } k * \text{Abs-fps } (\lambda n. a (n + \text{Suc } k)) =$$

$$\text{Abs-fps } a - \text{sum } (\lambda i. \text{fps-const } (a \$ i :: 'a :: \text{comm-ring-1}) * \text{fps-X} \hat{\ } i) \{0 .. k\}$$
**(is ?lhs = ?rhs)**  
 $\langle \text{proof} \rangle$

### 5.15.2 Rule 2

**definition** *fps-XD* =  $(*) \text{ fps-X} \circ \text{fps-deriv}$

**lemma** *fps-XD-add[simp]:fps-XD*  $(a + b) = \text{fps-XD } a + \text{fps-XD } (b :: 'a::\text{comm-ring-1 } \text{fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XD-mult-const[simp]:fps-XD*  $(\text{fps-const } (c :: 'a::\text{comm-ring-1}) * a) = \text{fps-const } c * \text{fps-XD } a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XD-linear[simp]: fps-XD*  $(\text{fps-const } c * a + \text{fps-const } d * b) = \text{fps-const } c * \text{fps-XD } a + \text{fps-const } d * \text{fps-XD } (b :: 'a::\text{comm-ring-1 } \text{fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XDN-linear:*  
 $(\text{fps-XD } \sim^n) (\text{fps-const } c * a + \text{fps-const } d * b) = \text{fps-const } c * (\text{fps-XD } \sim^n) a + \text{fps-const } d * (\text{fps-XD } \sim^n) (b :: 'a::\text{comm-ring-1 } \text{fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-fps-X-deriv-shift:*  $\text{fps-X} * \text{fps-deriv } a = \text{Abs-fps } (\lambda n. \text{of-nat } n * a \$ n)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-mult-fps-XD-shift:*  
 $(\text{fps-XD } \sim^k) (a :: 'a::\text{comm-ring-1 } \text{fps}) = \text{Abs-fps } (\lambda n. (\text{of-nat } n \wedge k) * a \$ n)$   
 $\langle \text{proof} \rangle$

### 5.15.3 Rule 3

Rule 3 is trivial and is given by `fps_times_def`.

### 5.15.4 Rule 5 — summation and “division” by $1 - X$

**lemma** *fps-divide-fps-X-minus1-sum-lemma:*  
 $a = ((1 :: 'a::\text{ring-1 } \text{fps}) - \text{fps-X}) * \text{Abs-fps } (\lambda n. \text{sum } (\lambda i. a \$ i) \{0..n\})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-fps-X-minus1-sum-ring1:*  
**assumes** *inverse*  $1 = (1 :: 'a::\{\text{ring-1}, \text{inverse}\})$   
**shows**  $a / ((1 :: 'a \text{fps}) - \text{fps-X}) = \text{Abs-fps } (\lambda n. \text{sum } (\lambda i. a \$ i) \{0..n\})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-divide-fps-X-minus1-sum:*  
 $a / ((1 :: 'a::\text{division-ring } \text{fps}) - \text{fps-X}) = \text{Abs-fps } (\lambda n. \text{sum } (\lambda i. a \$ i) \{0..n\})$   
 $\langle \text{proof} \rangle$

### 5.15.5 Rule 4 in its more general form

This generalizes Rule 3 for an arbitrary finite product of FPS, also the relevant instance of powers of a FPS.

**definition**  $\text{natpermute } n \ k = \{l :: \text{nat list. length } l = k \wedge \text{sum-list } l = n\}$

**lemma**  $\text{natlist-trivial-1}$ :  $\text{natpermute } n \ 1 = \{[n]\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natlist-trivial-Suc0}$  [simp]:  $\text{natpermute } n \ (\text{Suc } 0) = \{[n]\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{append-natpermute-less-eq}$ :  
**assumes**  $xs \ @ \ ys \in \text{natpermute } n \ k$   
**shows**  $\text{sum-list } xs \leq n$   
**and**  $\text{sum-list } ys \leq n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natpermute-split}$ :  
**assumes**  $h \leq k$   
**shows**  $\text{natpermute } n \ k =$   
 $(\bigcup m \in \{0..n\}. \{l1 \ @ \ l2 \mid l1 \ l2. l1 \in \text{natpermute } m \ h \wedge l2 \in \text{natpermute } (n -$   
 $m) \ (k - h)\})$   
**(is**  $?L = ?R$  **is**  $- = (\bigcup m \in \{0..n\}. ?S \ m))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natpermute-0}$ :  $\text{natpermute } n \ 0 = (\text{if } n = 0 \text{ then } \{\} \text{ else } \{\})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natpermute-0'}$ [simp]:  $\text{natpermute } 0 \ k = (\text{if } k = 0 \text{ then } \{\} \text{ else } \{\text{replicate } k \ 0\})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natpermute-finite}$ :  $\text{finite } (\text{natpermute } n \ k)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{natpermute-contain-maximal}$ :  
 $\{xs \in \text{natpermute } n \ (k + 1). n \in \text{set } xs\} = (\bigcup i \in \{0 .. k\}. \{\text{replicate } (k + 1) \ 0\}$   
 $[i := n]\})$   
**(is**  $?A = ?B$   
 $\langle \text{proof} \rangle$

The general form.

**lemma**  $\text{fps-prod-nth}$ :  
**fixes**  $m :: \text{nat}$   
**and**  $a :: \text{nat} \Rightarrow 'a :: \text{comm-ring-1 fps}$   
**shows**  $(\text{prod } a \ \{0 .. m\}) \ \$ \ n =$   
 $\text{sum } (\lambda v. \text{prod } (\lambda j. (a \ j) \ \$ \ (v!j)) \ \{0..m\}) \ (\text{natpermute } n \ (m+1))$   
**(is**  $?P \ m \ n$   
 $\langle \text{proof} \rangle$

The special form for powers.

**lemma**  $\text{fps-power-nth-Suc}$ :

**fixes**  $m :: \text{nat}$   
**and**  $a :: 'a :: \text{comm-ring-1 fps}$   
**shows**  $(a \wedge^{\text{Suc } m}) \$ n = \text{sum } (\lambda v. \text{prod } (\lambda j. a \$ (v!j)) \{0..m\}) (\text{natpermute } n \text{ } (m+1))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-nth*:  
**fixes**  $m :: \text{nat}$   
**and**  $a :: 'a :: \text{comm-ring-1 fps}$   
**shows**  $(a \wedge^m) \$ n =$   
 $(\text{if } m=0 \text{ then } 1 \$ n \text{ else } \text{sum } (\lambda v. \text{prod } (\lambda j. a \$ (v!j)) \{0..m-1\}) (\text{natpermute } n \text{ } m))$   
 $\langle \text{proof} \rangle$

**lemmas** *fps-nth-power-0 = fps-power-zeroth*

**lemma** *natpermute-max-card*:  
**assumes**  $n0: n \neq 0$   
**shows**  $\text{card } \{xs \in \text{natpermute } n \text{ } (k+1). n \in \text{set } xs\} = k+1$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-Suc-nth*:  
**fixes**  $f :: 'a :: \text{comm-ring-1 fps}$   
**assumes**  $k: k > 0$   
**shows**  $(f \wedge^{\text{Suc } m}) \$ k =$   
 $\text{of-nat } (\text{Suc } m) * (f \$ k * (f \$ 0) \wedge^m) +$   
 $(\sum v \in \{v \in \text{natpermute } k \text{ } (m+1). k \notin \text{set } v\}. \prod j = 0..m. f \$ v ! j)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-Suc-eqD*:  
**fixes**  $f g :: 'a :: \{idom, \text{semiring-char-0}\} \text{ fps}$   
**assumes**  $f \wedge^{\text{Suc } m} = g \wedge^{\text{Suc } m} f \$ 0 = g \$ 0 f \$ 0 \neq 0$   
**shows**  $f = g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-Suc-eqD'*:  
**fixes**  $f g :: 'a :: \{idom, \text{semiring-char-0}\} \text{ fps}$   
**assumes**  $f \wedge^{\text{Suc } m} = g \wedge^{\text{Suc } m} f \$ \text{subdegree } f = g \$ \text{subdegree } g$   
**shows**  $f = g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-eqD'*:  
**fixes**  $f g :: 'a :: \{idom, \text{semiring-char-0}\} \text{ fps}$   
**assumes**  $f \wedge^m = g \wedge^m f \$ \text{subdegree } f = g \$ \text{subdegree } g \text{ } m > 0$   
**shows**  $f = g$   
 $\langle \text{proof} \rangle$

**lemma** *fps-power-eqD*:  
**fixes**  $f g :: 'a :: \{idom, \text{semiring-char-0}\} \text{ fps}$

**assumes**  $f \wedge m = g \wedge m \ f \ \$ \ 0 = g \ \$ \ 0 \ f \ \$ \ 0 \neq 0 \ m > 0$   
**shows**  $f = g$   
 $\langle proof \rangle$

**lemma** *fps-compose-inj-right*:  
**assumes**  $a0: a\$0 = (0::'a::idom)$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $(b \text{ oo } a = c \text{ oo } a) \longleftrightarrow b = c$   
**(is**  $?lhs \longleftrightarrow ?rhs)$   
 $\langle proof \rangle$

## 5.16 Radicals

**declare** *prod.cong* [*fundef-cong*]

**function** *radical* ::  $(nat \Rightarrow 'a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a::field \text{ fps} \Rightarrow nat \Rightarrow 'a$   
**where**  
 $radical \ r \ 0 \ a \ 0 = 1$   
 $| radical \ r \ 0 \ a \ (Suc \ n) = 0$   
 $| radical \ r \ (Suc \ k) \ a \ 0 = r \ (Suc \ k) \ (a\$0)$   
 $| radical \ r \ (Suc \ k) \ a \ (Suc \ n) =$   
 $(a\$ \ Suc \ n - sum \ (\lambda xs. prod \ (\lambda j. radical \ r \ (Suc \ k) \ a \ (xs \ ! \ j)) \ \{0..k\})$   
 $\{xs. xs \in natpermute \ (Suc \ n) \ (Suc \ k) \wedge Suc \ n \notin set \ xs\}) /$   
 $(of-nat \ (Suc \ k) * (radical \ r \ (Suc \ k) \ a \ 0)^\wedge k)$   
 $\langle proof \rangle$

**termination** *radical*  
 $\langle proof \rangle$

**definition** *fps-radical*  $r \ n \ a = Abs-fps \ (radical \ r \ n \ a)$

**lemma** *radical-0* [*simp*]:  $\bigwedge n. \ 0 < n \implies radical \ r \ 0 \ a \ n = 0$   
 $\langle proof \rangle$

**lemma** *fps-radical0* [*simp*]:  $fps-radical \ r \ 0 \ a = 1$   
 $\langle proof \rangle$

**lemma** *fps-radical-nth-0* [*simp*]:  $fps-radical \ r \ n \ a \ \$ \ 0 = (if \ n = 0 \ then \ 1 \ else \ r \ n \ (a\$0))$   
 $\langle proof \rangle$

**lemma** *fps-radical-power-nth* [*simp*]:  
**assumes**  $r: (r \ k \ (a\$0))^\wedge k = a\$0$   
**shows**  $fps-radical \ r \ k \ a^\wedge k \ \$ \ 0 = (if \ k = 0 \ then \ 1 \ else \ a\$0)$   
 $\langle proof \rangle$

**lemma** *power-radical*:  
**fixes**  $a::'a::field-char-0 \text{ fps}$   
**assumes**  $a0: a\$0 \neq 0$

**shows**  $(r (Suc\ k) (a\$0)) \wedge Suc\ k = a\$0 \longleftrightarrow (fps-radical\ r\ (Suc\ k)\ a) \wedge (Suc\ k)$   
 $= a$   
**(is**  $?lhs \longleftrightarrow ?rhs$   
 $\langle proof \rangle$

**lemma radical-unique:**  
**assumes**  $r0: (r (Suc\ k) (b\$0)) \wedge Suc\ k = b\$0$   
**and**  $a0: r (Suc\ k) (b\$0 :: 'a::field-char-0) = a\$0$   
**and**  $b0: b\$0 \neq 0$   
**shows**  $a \wedge (Suc\ k) = b \longleftrightarrow a = fps-radical\ r\ (Suc\ k)\ b$   
**(is**  $?lhs \longleftrightarrow ?rhs$  **is**  $- \longleftrightarrow a = ?r$   
 $\langle proof \rangle$

**lemma radical-power:**  
**assumes**  $r0: r (Suc\ k) ((a\$0) \wedge Suc\ k) = a\$0$   
**and**  $a0: (a\$0 :: 'a::field-char-0) \neq 0$   
**shows**  $(fps-radical\ r\ (Suc\ k) (a \wedge Suc\ k)) = a$   
 $\langle proof \rangle$

**lemma fps-deriv-radical':**  
**fixes**  $a :: 'a::field-char-0\ fps$   
**assumes**  $r0: (r (Suc\ k) (a\$0)) \wedge Suc\ k = a\$0$   
**and**  $a0: a\$0 \neq 0$   
**shows**  $fps-deriv\ (fps-radical\ r\ (Suc\ k)\ a) =$   
 $fps-deriv\ a / ((of-nat\ (Suc\ k)) * (fps-radical\ r\ (Suc\ k)\ a) \wedge k)$   
 $\langle proof \rangle$

**lemma fps-deriv-radical:**  
**fixes**  $a :: 'a::field-char-0\ fps$   
**assumes**  $r0: (r (Suc\ k) (a\$0)) \wedge Suc\ k = a\$0$   
**and**  $a0: a\$0 \neq 0$   
**shows**  $fps-deriv\ (fps-radical\ r\ (Suc\ k)\ a) =$   
 $fps-deriv\ a / (fps-const\ (of-nat\ (Suc\ k)) * (fps-radical\ r\ (Suc\ k)\ a) \wedge k)$   
 $\langle proof \rangle$

**lemma radical-mult-distrib:**  
**fixes**  $a :: 'a::field-char-0\ fps$   
**assumes**  $k: k > 0$   
**and**  $ra0: r\ k\ (a\ \$\ 0) \wedge k = a\ \$\ 0$   
**and**  $rb0: r\ k\ (b\ \$\ 0) \wedge k = b\ \$\ 0$   
**and**  $a0: a\ \$\ 0 \neq 0$   
**and**  $b0: b\ \$\ 0 \neq 0$   
**shows**  $r\ k\ ((a * b)\ \$\ 0) = r\ k\ (a\ \$\ 0) * r\ k\ (b\ \$\ 0) \longleftrightarrow$   
 $fps-radical\ r\ k\ (a * b) = fps-radical\ r\ k\ a * fps-radical\ r\ k\ b$   
**(is**  $?lhs \longleftrightarrow ?rhs$   
 $\langle proof \rangle$

**lemma** *radical-divide*:  
**fixes**  $a :: 'a::field-char-0\ fps$   
**assumes**  $kp: k > 0$   
**and**  $ra0: (r\ k\ (a\ \$\ 0))\ \wedge^k = a\ \$\ 0$   
**and**  $rb0: (r\ k\ (b\ \$\ 0))\ \wedge^k = b\ \$\ 0$   
**and**  $a0: a\ \$\ 0 \neq 0$   
**and**  $b0: b\ \$\ 0 \neq 0$   
**shows**  $r\ k\ ((a\ \$\ 0) / (b\ \$\ 0)) = r\ k\ (a\ \$\ 0) / r\ k\ (b\ \$\ 0) \longleftrightarrow$   
 $fps-radical\ r\ k\ (a/b) = fps-radical\ r\ k\ a / fps-radical\ r\ k\ b$   
**(is**  $?lhs = ?rhs)$   
 $\langle proof \rangle$

**lemma** *radical-inverse*:  
**fixes**  $a :: 'a::field-char-0\ fps$   
**assumes**  $k: k > 0$   
**and**  $ra0: r\ k\ (a\ \$\ 0)\ \wedge^k = a\ \$\ 0$   
**and**  $r1: (r\ k\ 1)\ \wedge^k = 1$   
**and**  $a0: a\ \$\ 0 \neq 0$   
**shows**  $r\ k\ (inverse\ (a\ \$\ 0)) = r\ k\ 1 / (r\ k\ (a\ \$\ 0)) \longleftrightarrow$   
 $fps-radical\ r\ k\ (inverse\ a) = fps-radical\ r\ k\ 1 / fps-radical\ r\ k\ a$   
 $\langle proof \rangle$

## 5.17 Chain rule

**lemma** *fps-compose-deriv*:  
**fixes**  $a :: 'a::idom\ fps$   
**assumes**  $b0: b\ \$\ 0 = 0$   
**shows**  $fps-deriv\ (a\ oo\ b) = ((fps-deriv\ a)\ oo\ b) * fps-deriv\ b$   
 $\langle proof \rangle$

**lemma** *fps-poly-sum-fps-X*:  
**assumes**  $\forall i > n. a\ \$\ i = 0$   
**shows**  $a = sum\ (\lambda i. fps-const\ (a\ \$\ i) * fps-X\ \wedge^i)\ \{0..n\}\ (\text{is } a = ?r)$   
 $\langle proof \rangle$

## 5.18 Compositional inverses

**fun**  $compinv :: 'a\ fps \Rightarrow nat \Rightarrow 'a::field$   
**where**  
 $compinv\ a\ 0 = fps-X\ \$\ 0$   
 $| compinv\ a\ (Suc\ n) =$   
 $(fps-X\ \$\ Suc\ n - sum\ (\lambda i. (compinv\ a\ i) * (a\ \wedge^i)\ \$\ Suc\ n)\ \{0..n\}) / (a\ \$\ 1)\ \wedge^{Suc\ n}$

**definition**  $fps-inv\ a = Abs-fps\ (compinv\ a)$

**lemma** *fps-inv*:  
**assumes**  $a0: a\ \$\ 0 = 0$   
**and**  $a1: a\ \$\ 1 \neq 0$



**shows**  $\text{fps-inv } a \text{ oo } a = \text{fps-}X$   
 $\langle \text{proof} \rangle$

**fun**  $\text{gcompinv} :: 'a \text{ fps} \Rightarrow 'a \text{ fps} \Rightarrow \text{nat} \Rightarrow 'a::\text{field}$   
**where**  
 $\text{gcompinv } b \ a \ 0 = b\$0$   
 $| \text{gcompinv } b \ a \ (\text{Suc } n) =$   
 $(b\$ \text{Suc } n - \text{sum } (\lambda i. (\text{gcompinv } b \ a \ i) * (a \hat{~} i) \$ \text{Suc } n) \ \{0 .. n\}) / (a\$1) \hat{~} \text{Suc } n$

**definition**  $\text{fps-ginv } b \ a = \text{Abs-fps } (\text{gcompinv } b \ a)$

**lemma**  $\text{fps-ginv}$ :  
**assumes**  $a0: a\$0 = 0$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $\text{fps-ginv } b \ a \text{ oo } a = b$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-inv-ginv}$ :  $\text{fps-inv} = \text{fps-ginv } \text{fps-}X$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-compose-1[simp]}$ :  $1 \text{ oo } a = 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-compose-0[simp]}$ :  $0 \text{ oo } a = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-compose-0-right[simp]}$ :  $a \text{ oo } 0 = \text{fps-const } (a \$ 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-compose-add-distrib}$ :  $(a + b) \text{ oo } c = (a \text{ oo } c) + (b \text{ oo } c)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-compose-sum-distrib}$ :  $(\text{sum } f \ S) \text{ oo } a = \text{sum } (\lambda i. f \ i \text{ oo } a) \ S$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{convolution-eq}$ :  
 $\text{sum } (\lambda i. a \ (i :: \text{nat}) * b \ (n - i)) \ \{0 .. n\} =$   
 $\text{sum } (\lambda (i,j). a \ i * b \ j) \ \{(i,j). i \leq n \wedge j \leq n \wedge i + j = n\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{product-composition-lemma}$ :  
**assumes**  $c0: c\$0 = (0::'a::\text{idom})$   
**and**  $d0: d\$0 = 0$   
**shows**  $((a \text{ oo } c) * (b \text{ oo } d))\$n =$   
 $\text{sum } (\lambda (k,m). a\$k * b\$m * (c \hat{~} k * d \hat{~} m) \$ n) \ \{(k,m). k + m \leq n\} \ (\text{is } ?l = ?r)$   
 $\langle \text{proof} \rangle$

**lemma** *sum-pair-less-iff*:

$sum (\lambda((k::nat),m). a\ k * b\ m * c\ (k + m)) \{(k,m). k + m \leq n\} =$   
 $sum (\lambda s. sum (\lambda i. a\ i * b\ (s - i) * c\ s) \{0..s\}) \{0..n\}$   
 $(is\ ?l = ?r)$   
 $\langle proof \rangle$

**lemma** *fps-compose-mult-distrib-lemma*:

**assumes**  $c0: c\$0 = (0::'a::idom)$   
**shows**  $((a\ oo\ c) * (b\ oo\ c))\$n = sum (\lambda s. sum (\lambda i. a\$i * b\$ (s - i) * (c\hat{s}) \$ n)$   
 $\{0..s\}) \{0..n\}$   
 $\langle proof \rangle$

**lemma** *fps-compose-mult-distrib*:

**assumes**  $c0: c \$ 0 = (0::'a::idom)$   
**shows**  $(a * b) oo\ c = (a\ oo\ c) * (b\ oo\ c)$   
 $\langle proof \rangle$

**lemma** *fps-compose-prod-distrib*:

**assumes**  $c0: c\$0 = (0::'a::idom)$   
**shows**  $prod\ a\ S\ oo\ c = prod (\lambda k. a\ k\ oo\ c)\ S$   
 $\langle proof \rangle$

**lemma** *fps-compose-divide*:

**assumes**  $[simp]: g\ dvd\ f\ h\ \$\ 0 = 0$   
**shows**  $fps-compose\ f\ h = fps-compose\ (f / g :: 'a :: field\ fps)\ h * fps-compose$   
 $g\ h$   
 $\langle proof \rangle$

**lemma** *fps-compose-divide-distrib*:

**assumes**  $g\ dvd\ f\ h\ \$\ 0 = 0$   $fps-compose\ g\ h \neq 0$   
**shows**  $fps-compose\ (f / g :: 'a :: field\ fps)\ h = fps-compose\ f\ h / fps-compose$   
 $g\ h$   
 $\langle proof \rangle$

**lemma** *fps-compose-power*:

**assumes**  $c0: c\$0 = (0::'a::idom)$   
**shows**  $(a\ oo\ c)^{\wedge}n = a^{\wedge}n\ oo\ c$   
 $\langle proof \rangle$

**lemma** *fps-compose-uminus*:  $-(a::'a::ring-1\ fps)\ oo\ c = -(a\ oo\ c)$

$\langle proof \rangle$

**lemma** *fps-compose-sub-distrib*:  $(a - b) oo\ (c::'a::ring-1\ fps) = (a\ oo\ c) - (b\ oo\ c)$

$\langle proof \rangle$

**lemma** *fps-X-fps-compose*:  $fps-X\ oo\ a = Abs-fps (\lambda n. if\ n = 0\ then\ (0::'a::comm-ring-1)$   
 $else\ a\$n)$

$\langle proof \rangle$

**lemma** *fps-compose-eq-0-iff*:

**fixes**  $F\ G :: 'a :: idom\ fps$

**assumes**  $fps\_nth\ G\ 0 = 0$

**shows**  $fps\_compose\ F\ G = 0 \longleftrightarrow F = 0 \vee (G = 0 \wedge fps\_nth\ F\ 0 = 0)$

$\langle proof \rangle$

**lemma** *subdegree-fps-compose [simp]*:

**fixes**  $F\ G :: 'a :: idom\ fps$

**assumes**  $[simp]: fps\_nth\ G\ 0 = 0$

**shows**  $subdegree\ (fps\_compose\ F\ G) = subdegree\ F * subdegree\ G$

$\langle proof \rangle$

**lemma** *fps-inverse-compose*:

**assumes**  $b0: (b\$0 :: 'a::field) = 0$

**and**  $a0: a\$0 \neq 0$

**shows**  $inverse\ a\ oo\ b = inverse\ (a\ oo\ b)$

$\langle proof \rangle$

**lemma** *fps-divide-compose*:

**assumes**  $c0: (c\$0 :: 'a::field) = 0$

**and**  $b0: b\$0 \neq 0$

**shows**  $(a/b)\ oo\ c = (a\ oo\ c) / (b\ oo\ c)$

$\langle proof \rangle$

**lemma** *gp*:

**assumes**  $a0: a\$0 = (0 :: 'a::field)$

**shows**  $(Abs\_fps\ (\lambda n. 1))\ oo\ a = 1 / (1 - a)$

**(is ?one oo a = -)**

$\langle proof \rangle$

**lemma** *fps-compose-radical*:

**assumes**  $b0: b\$0 = (0 :: 'a::field-char-0)$

**and**  $ra0: r\ (Suc\ k)\ (a\$0) \wedge Suc\ k = a\$0$

**and**  $a0: a\$0 \neq 0$

**shows**  $fps\_radical\ r\ (Suc\ k)\ a\ oo\ b = fps\_radical\ r\ (Suc\ k)\ (a\ oo\ b)$

$\langle proof \rangle$

**lemma** *fps-const-mult-apply-left*:  $fps\_const\ c * (a\ oo\ b) = (fps\_const\ c * a)\ oo\ b$

$\langle proof \rangle$

**lemma** *fps-const-mult-apply-right*:

$(a\ oo\ b) * fps\_const\ (c :: 'a::comm-semiring-1) = (fps\_const\ c * a)\ oo\ b$

$\langle proof \rangle$

**lemma** *fps-compose-assoc*:

**assumes**  $c0: c\$0 = (0 :: 'a::idom)$

**and**  $b0: b\$0 = 0$

**shows**  $a \text{ oo } (b \text{ oo } c) = a \text{ oo } b \text{ oo } c$  (**is**  $?l = ?r$ )  
 $\langle \text{proof} \rangle$

**lemma** *fps-X-power-compose*:  
**assumes**  $a0: a\$0=0$   
**shows**  $\text{fps-X}^k \text{ oo } a = (a::'a::\text{idom fps})^k$   
**(is**  $?l = ?r$ )  
 $\langle \text{proof} \rangle$

**lemma** *fps-inv-right*:  
**assumes**  $a0: a\$0 = 0$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $a \text{ oo fps-inv } a = \text{fps-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fps-inv-deriv*:  
**assumes**  $a0: a\$0 = (0::'a::\text{field})$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $\text{fps-deriv } (\text{fps-inv } a) = \text{inverse } (\text{fps-deriv } a \text{ oo fps-inv } a)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-inv-idempotent*:  
**assumes**  $a0: a\$0 = 0$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $\text{fps-inv } (\text{fps-inv } a) = a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-ginv-ginv*:  
**assumes**  $a0: a\$0 = 0$   
**and**  $a1: a\$1 \neq 0$   
**and**  $c0: c\$0 = 0$   
**and**  $c1: c\$1 \neq 0$   
**shows**  $\text{fps-ginv } b \text{ (fps-ginv } c \text{ a)} = b \text{ oo } a \text{ oo fps-inv } c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-ginv-deriv*:  
**assumes**  $a0: a\$0 = (0::'a::\text{field})$   
**and**  $a1: a\$1 \neq 0$   
**shows**  $\text{fps-deriv } (\text{fps-ginv } b \text{ a}) = (\text{fps-deriv } b / \text{fps-deriv } a) \text{ oo fps-ginv fps-X } a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-compose-linear*:  
 $\text{fps-compose } (f :: 'a :: \text{comm-ring-1 fps}) (\text{fps-const } c * \text{fps-X}) = \text{Abs-fps } (\lambda n. c \hat{\ } n * f \$ n)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-compose-uminus'*:  
 $\text{fps-compose } f \text{ (-fps-X :: 'a :: comm-ring-1 fps)} = \text{Abs-fps } (\lambda n. (-1) \hat{\ } n * f \$ n)$

$\langle proof \rangle$   
**lemma** *fps-nth-compose-linear* [simp]:  
 fixes  $f :: 'a :: comm-ring-1$  *fps*  
 shows  $fps\text{-}nth\ (fps\text{-}compose\ f\ (fps\text{-}const\ c * fps\text{-}X))\ n = c^{\wedge} n * fps\text{-}nth\ f\ n$   
 $\langle proof \rangle$

## 5.19 Elementary series

### 5.19.1 Exponential series

**definition** *fps-exp*  $x = Abs\text{-}fps\ (\lambda n. x^{\wedge} n / of\text{-}nat\ (fact\ n))$

**lemma** *fps-exp-deriv*[simp]:  $fps\text{-}deriv\ (fps\text{-}exp\ a) = fps\text{-}const\ (a::'a::field\text{-}char-0) * fps\text{-}exp\ a$   
 (is  $?l = ?r$ )  
 $\langle proof \rangle$

**lemma** *fps-exp-unique-ODE*:  
 $fps\text{-}deriv\ a = fps\text{-}const\ c * a \longleftrightarrow a = fps\text{-}const\ (a\$0) * fps\text{-}exp\ (c::'a::field\text{-}char-0)$   
 (is  $?lhs \longleftrightarrow ?rhs$ )  
 $\langle proof \rangle$

**lemma** *fps-exp-add-mult*:  $fps\text{-}exp\ (a + b) = fps\text{-}exp\ (a::'a::field\text{-}char-0) * fps\text{-}exp\ b$   
 (is  $?l = ?r$ )  
 $\langle proof \rangle$

**lemma** *fps-exp-nth*[simp]:  $fps\text{-}exp\ a \$ n = a^{\wedge} n / of\text{-}nat\ (fact\ n)$   
 $\langle proof \rangle$

**lemma** *fps-exp-0*[simp]:  $fps\text{-}exp\ (0::'a::field) = 1$   
 $\langle proof \rangle$

**lemma** *fps-exp-neg*:  $fps\text{-}exp\ (-a) = inverse\ (fps\text{-}exp\ (a::'a::field\text{-}char-0))$   
 $\langle proof \rangle$

**lemma** *fps-exp-nth-deriv*[simp]:  
 $fps\text{-}nth\text{-}deriv\ n\ (fps\text{-}exp\ (a::'a::field\text{-}char-0)) = (fps\text{-}const\ a)^{\wedge} n * (fps\text{-}exp\ a)$   
 $\langle proof \rangle$

**lemma** *fps-X-compose-fps-exp*[simp]:  $fps\text{-}X\ oo\ fps\text{-}exp\ (a::'a::field) = fps\text{-}exp\ a - 1$   
 $\langle proof \rangle$

**lemma** *fps-inv-fps-exp-compose*:  
 assumes  $a: a \neq 0$   
 shows  $fps\text{-}inv\ (fps\text{-}exp\ a - 1) oo\ (fps\text{-}exp\ a - 1) = fps\text{-}X$   
 and  $(fps\text{-}exp\ a - 1) oo\ fps\text{-}inv\ (fps\text{-}exp\ a - 1) = fps\text{-}X$   
 $\langle proof \rangle$

**lemma** *fps-exp-power-mult*:  $(fps\text{-}exp\ (c::'a::field\text{-}char-0))^{\wedge} n = fps\text{-}exp\ (of\text{-}nat\ n * c)$

c)  
 ⟨proof⟩

**lemma** *radical-fps-exp*:  
 assumes  $r: r \text{ (Suc } k) \text{ } 1 = 1$   
 shows  $\text{fps-radical } r \text{ (Suc } k) \text{ (fps-exp (c::'a::field-char-0))} = \text{fps-exp (c / of-nat (Suc } k))$   
 ⟨proof⟩

**lemma** *fps-exp-compose-linear* [simp]:  
 $\text{fps-exp (d::'a::field-char-0) oo (fps-const c * fps-X)} = \text{fps-exp (c * d)}$   
 ⟨proof⟩

**lemma** *fps-fps-exp-compose-minus* [simp]:  
 $\text{fps-compose (fps-exp c) (-fps-X)} = \text{fps-exp (-c :: 'a :: field-char-0)}$   
 ⟨proof⟩

**lemma** *fps-exp-eq-iff* [simp]:  $\text{fps-exp } c = \text{fps-exp } d \longleftrightarrow c = (d :: 'a :: \text{field-char-0})$   
 ⟨proof⟩

**lemma** *fps-exp-eq-fps-const-iff* [simp]:  
 $\text{fps-exp (c :: 'a :: field-char-0)} = \text{fps-const } c' \longleftrightarrow c = 0 \wedge c' = 1$   
 ⟨proof⟩

**lemma** *fps-exp-neq-0* [simp]:  $\neg \text{fps-exp (c :: 'a :: field-char-0)} = 0$   
 ⟨proof⟩

**lemma** *fps-exp-eq-1-iff* [simp]:  $\text{fps-exp (c :: 'a :: field-char-0)} = 1 \longleftrightarrow c = 0$   
 ⟨proof⟩

**lemma** *fps-exp-neq-numeral-iff* [simp]:  
 $\text{fps-exp (c :: 'a :: field-char-0)} = \text{numeral } n \longleftrightarrow c = 0 \wedge n = \text{Num.One}$   
 ⟨proof⟩

### 5.19.2 Logarithmic series

**lemma** *Abs-fps-if-0*:  
 $\text{Abs-fps } (\lambda n. \text{ if } n = 0 \text{ then } (v::'a::\text{ring-1}) \text{ else } f \text{ } n) =$   
 $\text{fps-const } v + \text{fps-X} * \text{Abs-fps } (\lambda n. f \text{ (Suc } n))$   
 ⟨proof⟩

**definition** *fps-ln* ::  $'a::\text{field-char-0} \Rightarrow 'a \text{ fps}$   
 where  $\text{fps-ln } c = \text{fps-const (1/c)} * \text{Abs-fps } (\lambda n. \text{ if } n = 0 \text{ then } 0 \text{ else } (-1) ^ (n - 1) / \text{of-nat } n)$

**lemma** *fps-ln-deriv*:  $\text{fps-deriv (fps-ln } c) = \text{fps-const (1/c)} * \text{inverse (1 + fps-X)}$   
 ⟨proof⟩

**lemma** *fps-ln-nth*:  $\text{fps-ln } c \text{ \$ } n = (\text{if } n = 0 \text{ then } 0 \text{ else } 1/c * ((-1) ^ (n - 1) /$

*of-nat n*)  
 ⟨proof⟩

**lemma** *fps-ln-0* [simp]: *fps-ln c \$ 0 = 0* ⟨proof⟩

**lemma** *fps-ln-fps-exp-inv*:  
 fixes *a* :: 'a::field-char-0  
 assumes *a*: *a* ≠ 0  
 shows *fps-ln a = fps-inv (fps-exp a - 1)* (is ?l = ?r)  
 ⟨proof⟩

**lemma** *fps-ln-mult-add*:  
 assumes *c0*: *c* ≠ 0  
 and *d0*: *d* ≠ 0  
 shows *fps-ln c + fps-ln d = fps-const (c+d) \* fps-ln (c\*d)*  
 (is ?r = ?l)  
 ⟨proof⟩

**lemma** *fps-X-dvd-fps-ln* [simp]: *fps-X dvd fps-ln c*  
 ⟨proof⟩

### 5.19.3 Binomial series

**definition** *fps-binomial a = Abs-fps (λn. a gchoose n)*

**lemma** *fps-binomial-nth*[simp]: *fps-binomial a \$ n = a gchoose n*  
 ⟨proof⟩

**lemma** *fps-binomial-ODE-unique*:  
 fixes *c* :: 'a::field-char-0  
 shows *fps-deriv a = (fps-const c \* a) / (1 + fps-X) ⟷ a = fps-const (a\$0) \* fps-binomial c*  
 (is ?lhs ⟷ ?rhs)  
 ⟨proof⟩

**lemma** *fps-binomial-ODE-unique'*:  
 (*fps-deriv a = fps-const c \* a / (1 + fps-X) ∧ a \$ 0 = 1*) ⟷ (*a = fps-binomial c*)  
 ⟨proof⟩

**lemma** *fps-binomial-deriv*: *fps-deriv (fps-binomial c) = fps-const c \* fps-binomial c / (1 + fps-X)*  
 ⟨proof⟩

**lemma** *fps-binomial-add-mult*: *fps-binomial (c+d) = fps-binomial c \* fps-binomial d* (is ?l = ?r)  
 ⟨proof⟩

**lemma** *fps-binomial-minus-one*: *fps-binomial (- 1) = inverse (1 + fps-X)*

(is ?l = inverse ?r)  
 <proof>

**lemma** *fps-binomial-of-nat*:  $\text{fps-binomial } (\text{of-nat } n) = (1 + \text{fps-X} :: 'a :: \text{field-char-0 } \text{fps})^{\wedge} n$   
 <proof>

**lemma** *fps-binomial-0 [simp]*:  $\text{fps-binomial } 0 = 1$   
 <proof>

**lemma** *fps-binomial-power*:  $\text{fps-binomial } a^{\wedge} n = \text{fps-binomial } (\text{of-nat } n * a)$   
 <proof>

**lemma** *fps-binomial-1*:  $\text{fps-binomial } 1 = 1 + \text{fps-X}$   
 <proof>

**lemma** *fps-binomial-minus-of-nat*:  
 $\text{fps-binomial } (- \text{of-nat } n) = \text{inverse } ((1 + \text{fps-X} :: 'a :: \text{field-char-0 } \text{fps})^{\wedge} n)$   
 <proof>

**lemma** *one-minus-const-fps-X-power*:  
 $c \neq 0 \implies (1 - \text{fps-const } c * \text{fps-X})^{\wedge} n =$   
 $\text{fps-compose } (\text{fps-binomial } (\text{of-nat } n)) (-\text{fps-const } c * \text{fps-X})$   
 <proof>

**lemma** *one-minus-fps-X-const-neg-power*:  
 $\text{inverse } ((1 - \text{fps-const } c * \text{fps-X})^{\wedge} n) =$   
 $\text{fps-compose } (\text{fps-binomial } (-\text{of-nat } n)) (-\text{fps-const } c * \text{fps-X})$   
 <proof>

**lemma** *fps-X-plus-const-power*:  
 $c \neq 0 \implies (\text{fps-X} + \text{fps-const } c)^{\wedge} n =$   
 $\text{fps-const } (c^{\wedge} n) * \text{fps-compose } (\text{fps-binomial } (\text{of-nat } n)) (\text{fps-const } (\text{inverse } c)$   
 $* \text{fps-X})$   
 <proof>

**lemma** *fps-X-plus-const-neg-power*:  
 $c \neq 0 \implies \text{inverse } ((\text{fps-X} + \text{fps-const } c)^{\wedge} n) =$   
 $\text{fps-const } (\text{inverse } c^{\wedge} n) * \text{fps-compose } (\text{fps-binomial } (-\text{of-nat } n)) (\text{fps-const } (\text{inverse } c) * \text{fps-X})$   
 <proof>

**lemma** *one-minus-const-fps-X-neg-power'*:  
**fixes**  $c :: 'a :: \text{field-char-0}$   
**assumes**  $n > 0$   
**shows**  $\text{inverse } ((1 - \text{fps-const } c * \text{fps-X})^{\wedge} n) = \text{Abs-fps } (\lambda k. \text{of-nat } ((n + k - 1) \text{ choose } k) * c^{\wedge} k)$   
 <proof>



Vandermonde's Identity as a consequence.

**lemma** *gbinomial-Vandermonde*:

$sum (\lambda k. (a \text{ gchoose } k) * (b \text{ gchoose } (n - k))) \{0..n\} = (a + b) \text{ gchoose } n$   
 $\langle proof \rangle$

**lemma** *binomial-Vandermonde*:

$sum (\lambda k. (a \text{ choose } k) * (b \text{ choose } (n - k))) \{0..n\} = (a + b) \text{ choose } n$   
 $\langle proof \rangle$

**lemma** *binomial-Vandermonde-same*:  $sum (\lambda k. (n \text{ choose } k)^2) \{0..n\} = (2 * n) \text{ choose } n$   
 $\langle proof \rangle$

**lemma** *Vandermonde-pochhammer-lemma*:

**fixes**  $a :: 'a::field-char-0$   
**assumes**  $b: \bigwedge j. j < n \implies b \neq \text{of-nat } j$   
**shows**  $sum (\lambda k. (\text{pochhammer } (- a) k * \text{pochhammer } (- (\text{of-nat } n)) k) /$   
 $(\text{of-nat } (\text{fact } k) * \text{pochhammer } (b - \text{of-nat } n + 1) k)) \{0..n\} =$   
 $\text{pochhammer } (- (a + b)) n / \text{pochhammer } (- b) n$   
**(is ?l = ?r)**  
 $\langle proof \rangle$

**lemma** *Vandermonde-pochhammer*:

**fixes**  $a :: 'a::field-char-0$   
**assumes**  $c: \forall i \in \{0..< n\}. c \neq - \text{of-nat } i$   
**shows**  $sum (\lambda k. (\text{pochhammer } a k * \text{pochhammer } (- (\text{of-nat } n)) k) /$   
 $(\text{of-nat } (\text{fact } k) * \text{pochhammer } c k)) \{0..n\} = \text{pochhammer } (c - a) n / \text{pochhammer } c n$   
 $\langle proof \rangle$

#### 5.19.4 Trigonometric functions

**definition** *fps-sin* ( $c::'a::field-char-0$ ) =

$Abs\text{-fps } (\lambda n. \text{ if even } n \text{ then } 0 \text{ else } (- 1) ^ ((n - 1) \text{ div } 2) * c ^ n / (\text{of-nat } (\text{fact } n)))$

**definition** *fps-cos* ( $c::'a::field-char-0$ ) =

$Abs\text{-fps } (\lambda n. \text{ if even } n \text{ then } (- 1) ^ (n \text{ div } 2) * c ^ n / (\text{of-nat } (\text{fact } n)) \text{ else } 0)$

**lemma** *fps-sin-0* [simp]:  $\text{fps-sin } 0 = 0$

$\langle proof \rangle$

**lemma** *fps-cos-0* [simp]:  $\text{fps-cos } 0 = 1$

$\langle proof \rangle$

**lemma** *fps-sin-deriv*:

$\text{fps-deriv } (\text{fps-sin } c) = \text{fps-const } c * \text{fps-cos } c$

**(is ?lhs = ?rhs)**

$\langle proof \rangle$

**lemma** *fps-cos-deriv*:  $\text{fps-deriv } (\text{fps-cos } c) = \text{fps-const } (-\ c) * (\text{fps-sin } c)$   
 (is ?lhs = ?rhs)  
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-cos-sum-of-squares*:  $(\text{fps-cos } c)^2 + (\text{fps-sin } c)^2 = 1$   
 (is ?lhs = -)  
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-nth-0* [simp]:  $\text{fps-sin } c \$ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-nth-1* [simp]:  $\text{fps-sin } c \$ \text{Suc } 0 = c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-nth-add-2*:  
 $\text{fps-sin } c \$ (n + 2) = - (c * c * \text{fps-sin } c \$ n / (\text{of-nat } (n + 1) * \text{of-nat } (n + 2)))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-nth-0* [simp]:  $\text{fps-cos } c \$ 0 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-nth-1* [simp]:  $\text{fps-cos } c \$ \text{Suc } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-nth-add-2*:  
 $\text{fps-cos } c \$ (n + 2) = - (c * c * \text{fps-cos } c \$ n / (\text{of-nat } (n + 1) * \text{of-nat } (n + 2)))$   
 $\langle \text{proof} \rangle$

**lemma** *nat-add-1-add-1*:  $(n::\text{nat}) + 1 + 1 = n + 2$   
 $\langle \text{proof} \rangle$

**lemma** *eq-fps-sin*:  
 assumes  $a0: a \$ 0 = 0$   
 and  $a1: a \$ 1 = c$   
 and  $a2: \text{fps-deriv } (\text{fps-deriv } a) = - (\text{fps-const } c * \text{fps-const } c * a)$   
 shows  $\text{fps-sin } c = a$   
 $\langle \text{proof} \rangle$

**lemma** *eq-fps-cos*:  
 assumes  $a0: a \$ 0 = 1$   
 and  $a1: a \$ 1 = 0$   
 and  $a2: \text{fps-deriv } (\text{fps-deriv } a) = - (\text{fps-const } c * \text{fps-const } c * a)$   
 shows  $\text{fps-cos } c = a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-add*:  $\text{fps-sin } (a + b) = \text{fps-sin } a * \text{fps-cos } b + \text{fps-cos } a * \text{fps-sin } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-add*:  $\text{fps-cos } (a + b) = \text{fps-cos } a * \text{fps-cos } b - \text{fps-sin } a * \text{fps-sin } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-even*:  $\text{fps-sin } (-c) = - \text{fps-sin } c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-odd*:  $\text{fps-cos } (-c) = \text{fps-cos } c$   
 $\langle \text{proof} \rangle$

**definition** *fps-tan*  $c = \text{fps-sin } c / \text{fps-cos } c$

**lemma** *fps-tan-0* [*simp*]:  $\text{fps-tan } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-tan-deriv*:  $\text{fps-deriv } (\text{fps-tan } c) = \text{fps-const } c / (\text{fps-cos } c)^2$   
 $\langle \text{proof} \rangle$

Connection to *fps-exp* over the complex numbers — Euler and de Moivre.

**lemma** *fps-exp-ii-sin-cos*:  $\text{fps-exp } (i * c) = \text{fps-cos } c + \text{fps-const } i * \text{fps-sin } c$   
 $(\text{is } ?l = ?r)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-exp-minus-ii-sin-cos*:  $\text{fps-exp } (- (i * c)) = \text{fps-cos } c - \text{fps-const } i * \text{fps-sin } c$   
 $\langle \text{proof} \rangle$

**lemma** *fps-cos-fps-exp-ii*:  $\text{fps-cos } c = (\text{fps-exp } (i * c) + \text{fps-exp } (- i * c)) / \text{fps-const } 2$   
 $\langle \text{proof} \rangle$

**lemma** *fps-sin-fps-exp-ii*:  $\text{fps-sin } c = (\text{fps-exp } (i * c) - \text{fps-exp } (- i * c)) / \text{fps-const } (2 * i)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-tan-fps-exp-ii*:  
 $\text{fps-tan } c = (\text{fps-exp } (i * c) - \text{fps-exp } (- i * c)) /$   
 $(\text{fps-const } i * (\text{fps-exp } (i * c) + \text{fps-exp } (- i * c)))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-demoivre*:  
 $(\text{fps-cos } a + \text{fps-const } i * \text{fps-sin } a)^{\wedge} n =$   
 $\text{fps-cos } (\text{of-nat } n * a) + \text{fps-const } i * \text{fps-sin } (\text{of-nat } n * a)$   
 $\langle \text{proof} \rangle$

## 5.20 Hypergeometric series

**definition** *fps-hypergeo as bs* (*c*::'a::field-char-0) =  

$$\text{Abs-fps } (\lambda n. (\text{foldl } (\lambda r a. r * \text{pochhammer } a \ n) \ 1 \ \text{as} * c^{\wedge} n) /$$

$$(\text{foldl } (\lambda r b. r * \text{pochhammer } b \ n) \ 1 \ \text{bs} * \text{of-nat } (\text{fact } n)))$$

**lemma** *fps-hypergeo-nth[simp]*: *fps-hypergeo as bs c* \$ *n* =  

$$(\text{foldl } (\lambda r a. r * \text{pochhammer } a \ n) \ 1 \ \text{as} * c^{\wedge} n) /$$

$$(\text{foldl } (\lambda r b. r * \text{pochhammer } b \ n) \ 1 \ \text{bs} * \text{of-nat } (\text{fact } n))$$

$$\langle \text{proof} \rangle$$

**lemma** *foldl-mult-start*:  
**fixes** *v* :: 'a::comm-ring-1  
**shows** *foldl* ( $\lambda x r. r * f \ x$ ) *v as* \* *x* = *foldl* ( $\lambda x r. r * f \ x$ ) (*v* \* *x*) *as*  

$$\langle \text{proof} \rangle$$

**lemma** *foldr-mult-foldl*:  
**fixes** *v* :: 'a::comm-ring-1  
**shows** *foldr* ( $\lambda x r. r * f \ x$ ) *as v* = *foldl* ( $\lambda x r. r * f \ x$ ) *v as*  

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-nth-alt*:  

$$\text{fps-hypergeo as bs c } \$ \ n = \text{foldr } (\lambda a r. r * \text{pochhammer } a \ n) \ \text{as } (c^{\wedge} n) /$$

$$\text{foldr } (\lambda b r. r * \text{pochhammer } b \ n) \ \text{bs } (\text{of-nat } (\text{fact } n))$$

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-fps-exp[simp]*: *fps-hypergeo* [] [] *c* = *fps-exp c*  

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-1-0[simp]*: *fps-hypergeo* [1] [] *c* =  $1 / (1 - \text{fps-const } c * \text{fps-X})$   

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-B[simp]*: *fps-hypergeo* [-*a*] [] (- 1) = *fps-binomial a*  

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-0[simp]*: *fps-hypergeo as bs c* \$ 0 = 1  

$$\langle \text{proof} \rangle$$

**lemma** *foldl-prod-prod*:  

$$\text{foldl } (\lambda (r::'b::\text{comm-ring-1}) (x::'a::\text{comm-ring-1}). r * f \ x) \ v \ \text{as} * \text{foldl } (\lambda r x. r * g \ x) \ w \ \text{as} =$$

$$\text{foldl } (\lambda r x. r * f \ x * g \ x) \ (v * w) \ \text{as}$$

$$\langle \text{proof} \rangle$$

**lemma** *fps-hypergeo-rec*:  

$$\text{fps-hypergeo as bs c } \$ \ \text{Suc } n = ((\text{foldl } (\lambda r a. r * (a + \text{of-nat } n)) \ c \ \text{as}) /$$

$$(\text{foldl } (\lambda r b. r * (b + \text{of-nat } n)) \ (\text{of-nat } (\text{Suc } n)) \ \text{bs})) * \text{fps-hypergeo as bs c } \$ \ n$$

$$\langle \text{proof} \rangle$$

**lemma** *fps-XD-nth[simp]*:  $\text{fps-XD } a \ \$ \ n = \text{of-nat } n * a\$n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XD-0th[simp]*:  $\text{fps-XD } a \ \$ \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XD-Suc[simp]*:  $\text{fps-XD } a \ \$ \ \text{Suc } n = \text{of-nat } (\text{Suc } n) * a \ \$ \ \text{Suc } n$   
 $\langle \text{proof} \rangle$

**definition** *fps-XDp*  $c \ a = \text{fps-XD } a + \text{fps-const } c * a$

**lemma** *fps-XDp-nth[simp]*:  $\text{fps-XDp } c \ a \ \$ \ n = (c + \text{of-nat } n) * a\$n$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XDp-commute*:  $\text{fps-XDp } b \circ \text{fps-XDp } (c::'a::\text{comm-ring-1}) = \text{fps-XDp } c \circ \text{fps-XDp } b$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XDp0 [simp]*:  $\text{fps-XDp } 0 = \text{fps-XD}$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XDp-fps-integral [simp]*:  
**fixes**  $a :: 'a::\{\text{division-ring}, \text{ring-char-0}\}$  *fps*  
**shows**  $\text{fps-XDp } 0 \ (\text{fps-integral } a \ c) = \text{fps-X} * a$   
 $\langle \text{proof} \rangle$

**lemma** *fps-hypergeo-minus-nat*:  
 $\text{fps-hypergeo } [- \text{of-nat } n] \ [- \text{of-nat } (n + m)] \ (c::'a::\text{field-char-0}) \ \$ \ k =$   
 $(\text{if } k \leq n \text{ then}$   
 $\quad \text{pochhammer } (- \text{of-nat } n) \ k * c ^ k / (\text{pochhammer } (- \text{of-nat } (n + m)) \ k * \text{of-nat } (\text{fact } k))$   
 $\quad \text{else } 0)$   
 $\text{fps-hypergeo } [- \text{of-nat } m] \ [- \text{of-nat } (m + n)] \ (c::'a::\text{field-char-0}) \ \$ \ k =$   
 $(\text{if } k \leq m \text{ then}$   
 $\quad \text{pochhammer } (- \text{of-nat } m) \ k * c ^ k / (\text{pochhammer } (- \text{of-nat } (m + n)) \ k * \text{of-nat } (\text{fact } k))$   
 $\quad \text{else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *pochhammer-rec-if*:  $\text{pochhammer } a \ n = (\text{if } n = 0 \text{ then } 1 \text{ else } a * \text{pochhammer } (a + 1) \ (n - 1))$   
 $\langle \text{proof} \rangle$

**lemma** *fps-XDp-foldr-nth [simp]*:  $\text{foldr } (\lambda c \ r. \ \text{fps-XDp } c \circ r) \ cs \ (\lambda c. \ \text{fps-XDp } c \ a)$   
 $c0 \ \$ \ n =$   
 $\text{foldr } (\lambda c \ r. \ (c + \text{of-nat } n) * r) \ cs \ (c0 + \text{of-nat } n) * a\$n$   
 $\langle \text{proof} \rangle$

**lemma** *genric-fps-XDp-foldr-nth*:

```

assumes  $f: \forall n\ c\ a. f\ c\ a\ \$\ n = (of\_nat\ n + k\ c) * a\$n$ 
shows  $foldr\ (\lambda c\ r. f\ c\ \circ r)\ cs\ (\lambda c. g\ c\ a)\ c0\ \$\ n =$ 
 $foldr\ (\lambda c\ r. (k\ c + of\_nat\ n) * r)\ cs\ (g\ c0\ a\ \$\ n)$ 
 $\langle proof \rangle$ 

lemma dist-less-imp-nth-equal:
  assumes  $dist\ f\ g < inverse\ (2\ ^i)$ 
  and  $j \leq i$ 
shows  $f\ \$\ j = g\ \$\ j$ 
 $\langle proof \rangle$ 

lemma nth-equal-imp-dist-less:
  assumes  $\bigwedge j. j \leq i \implies f\ \$\ j = g\ \$\ j$ 
shows  $dist\ f\ g < inverse\ (2\ ^i)$ 
 $\langle proof \rangle$ 

lemma dist-less-eq-nth-equal:  $dist\ f\ g < inverse\ (2\ ^i) \longleftrightarrow (\forall j \leq i. f\ \$\ j = g\ \$\ j)$ 
 $\langle proof \rangle$ 

instance fps :: (comm-ring-1) complete-space
 $\langle proof \rangle$ 

bundle fps-syntax
begin
notation fps-nth (infixl  $\langle \$ \rangle\ 75$ )
end

unbundle no fps-syntax

end

```

## 6 Converting polynomials to formal power series

```

theory Polynomial-FPS
  imports Polynomial Formal-Power-Series
begin

context
  includes fps-syntax
begin

definition fps-of-poly where
  fps-of-poly  $p = Abs\_fps\ (coeff\ p)$ 

lemma fps-of-poly-eq-iff:  $fps\_of\_poly\ p = fps\_of\_poly\ q \longleftrightarrow p = q$ 
 $\langle proof \rangle$ 

```

**lemma** *fps-of-poly-nth* [simp]: *fps-of-poly* *p* \$ *n* = *coeff p n*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-const*: *fps-of-poly* [:*c*:] = *fps-const c*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-0* [simp]: *fps-of-poly 0* = 0  
 ⟨*proof*⟩

**lemma** *fps-of-poly-1* [simp]: *fps-of-poly 1* = 1  
 ⟨*proof*⟩

**lemma** *fps-of-poly-1'* [simp]: *fps-of-poly* [:1:] = 1  
 ⟨*proof*⟩

**lemma** *fps-of-poly-numeral* [simp]: *fps-of-poly* (*numeral n*) = *numeral n*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-numeral'* [simp]: *fps-of-poly* [:*numeral n*:] = *numeral n*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-fps-X* [simp]: *fps-of-poly* [:0, 1:] = *fps-X*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-add*: *fps-of-poly* (*p* + *q*) = *fps-of-poly p* + *fps-of-poly q*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-diff*: *fps-of-poly* (*p* − *q*) = *fps-of-poly p* − *fps-of-poly q*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-uminus*: *fps-of-poly* (−*p*) = −*fps-of-poly p*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-mult*: *fps-of-poly* (*p* \* *q*) = *fps-of-poly p* \* *fps-of-poly q*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-smult*:  
*fps-of-poly* (*smult c p*) = *fps-const c* \* *fps-of-poly p*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-sum*: *fps-of-poly* (*sum f A*) = *sum* (λ*x*. *fps-of-poly* (*f x*)) *A*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-sum-list*: *fps-of-poly* (*sum-list xs*) = *sum-list* (*map fps-of-poly xs*)  
 ⟨*proof*⟩

**lemma** *fps-of-poly-prod*: *fps-of-poly* (*prod f A*) = *prod* (λ*x*. *fps-of-poly* (*f x*)) *A*  
 ⟨*proof*⟩

**lemma** *fps-of-poly-prod-list*:  $\text{fps-of-poly } (\text{prod-list } xs) = \text{prod-list } (\text{map } \text{fps-of-poly } xs)$   
 ⟨proof⟩

**lemma** *fps-of-poly-pCons*:  
 $\text{fps-of-poly } (p\text{Cons } (c :: 'a :: \text{semiring-1}) p) = \text{fps-const } c + \text{fps-of-poly } p * \text{fps-X}$   
 ⟨proof⟩

**lemma** *fps-of-poly-pderiv*:  $\text{fps-of-poly } (p\text{deriv } p) = \text{fps-deriv } (\text{fps-of-poly } p)$   
 ⟨proof⟩

**lemma** *fps-of-poly-power*:  $\text{fps-of-poly } (p \wedge n) = \text{fps-of-poly } p \wedge n$   
 ⟨proof⟩

**lemma** *fps-of-poly-monom*:  $\text{fps-of-poly } (\text{monom } (c :: 'a :: \text{comm-ring-1}) n) = \text{fps-const } c * \text{fps-X} \wedge n$   
 ⟨proof⟩

**lemma** *fps-of-poly-monom'*:  $\text{fps-of-poly } (\text{monom } (1 :: 'a :: \text{comm-ring-1}) n) = \text{fps-X} \wedge n$   
 ⟨proof⟩

**lemma** *fps-of-poly-div*:  
 assumes  $(q :: 'a :: \text{field poly}) \text{ dvd } p$   
 shows  $\text{fps-of-poly } (p \text{ div } q) = \text{fps-of-poly } p / \text{fps-of-poly } q$   
 ⟨proof⟩

**lemma** *fps-of-poly-divide-numeral*:  
 $\text{fps-of-poly } (\text{smult } (\text{inverse } (\text{numeral } c :: 'a :: \text{field})) p) = \text{fps-of-poly } p / \text{numeral } c$   
 ⟨proof⟩

**lemma** *subdegree-fps-of-poly*:  
 assumes  $p \neq 0$   
 defines  $n \equiv \text{Polynomial.order } 0 p$   
 shows  $\text{subdegree } (\text{fps-of-poly } p) = n$   
 ⟨proof⟩

**lemma** *fps-of-poly-dvd*:  
 assumes  $p \text{ dvd } q$   
 shows  $\text{fps-of-poly } (p :: 'a :: \text{field poly}) \text{ dvd } \text{fps-of-poly } q$   
 ⟨proof⟩

**lemmas** *fps-of-poly-simps* =  
 $\text{fps-of-poly-0 } \text{fps-of-poly-1 } \text{fps-of-poly-numeral } \text{fps-of-poly-const } \text{fps-of-poly-fps-X}$   
 $\text{fps-of-poly-add } \text{fps-of-poly-diff } \text{fps-of-poly-uminus } \text{fps-of-poly-mult } \text{fps-of-poly-smult}$



*fps-of-poly-sum fps-of-poly-sum-list fps-of-poly-prod fps-of-poly-prod-list  
 fps-of-poly-pCons fps-of-poly-pderiv fps-of-poly-power fps-of-poly-monom  
 fps-of-poly-divide-numeral*

**lemma** *fps-of-poly-pcompose:*

**assumes** *coeff q 0 = (0 :: 'a :: idom)*

**shows** *fps-of-poly (pcompose p q) = fps-compose (fps-of-poly p) (fps-of-poly q)*  
*<proof>*

**lemmas** *reify-fps-atom =*

*fps-of-poly-0 fps-of-poly-1 ' fps-of-poly-numeral' fps-of-poly-const fps-of-poly-fps-X*

The following simproc can reduce the equality of two polynomial FPSs to equality of the respective polynomials. A polynomial FPS is one that only has finitely many non-zero coefficients and can therefore be written as *fps-of-poly p* for some polynomial *p*.

This may sound trivial, but it covers a number of annoying side conditions like  $1 + \text{fps-X} \neq 0$  that would otherwise not be solved automatically.

*<ML>*

**lemma** *fps-of-poly-linear: fps-of-poly [:a,1 :: 'a :: field:] = fps-X + fps-const a*  
*<proof>*

**lemma** *fps-of-poly-linear': fps-of-poly [:1,a :: 'a :: field:] = 1 + fps-const a \* fps-X*  
*<proof>*

**lemma** *fps-of-poly-cutoff [simp]:*

*fps-of-poly (poly-cutoff n p) = fps-cutoff n (fps-of-poly p)*  
*<proof>*

**lemma** *fps-of-poly-shift [simp]: fps-of-poly (poly-shift n p) = fps-shift n (fps-of-poly p)*

*<proof>*

**definition** *poly-subdegree :: 'a::zero poly  $\Rightarrow$  nat where*

*poly-subdegree p = subdegree (fps-of-poly p)*

**lemma** *coeff-less-poly-subdegree:*

*k < poly-subdegree p  $\implies$  coeff p k = 0*  
*<proof>*

**definition** *prefix-length :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a list  $\Rightarrow$  nat where*

*prefix-length P xs = length (takeWhile P xs)*

**primrec** *prefix-length-aux :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  nat  $\Rightarrow$  'a list  $\Rightarrow$  nat where*

*prefix-length-aux P acc [] = acc*

|  $\text{prefix-length-aux } P \text{ acc } (x \# xs) = (\text{if } P \ x \text{ then } \text{prefix-length-aux } P \ (\text{Suc acc}) \ xs \text{ else acc})$

**lemma** *prefix-length-aux-correct*:  $\text{prefix-length-aux } P \text{ acc } xs = \text{prefix-length } P \ xs + \text{acc}$   
 <proof>

**lemma** *prefix-length-code* [code]:  $\text{prefix-length } P \ xs = \text{prefix-length-aux } P \ 0 \ xs$   
 <proof>

**lemma** *prefix-length-le-length*:  $\text{prefix-length } P \ xs \leq \text{length } xs$   
 <proof>

**lemma** *prefix-length-less-length*:  $(\exists x \in \text{set } xs. \neg P \ x) \implies \text{prefix-length } P \ xs < \text{length } xs$   
 <proof>

**lemma** *nth-prefix-length*:  
 $(\exists x \in \text{set } xs. \neg P \ x) \implies \neg P \ (xs \ ! \ \text{prefix-length } P \ xs)$   
 <proof>

**lemma** *nth-less-prefix-length*:  
 $n < \text{prefix-length } P \ xs \implies P \ (xs \ ! \ n)$   
 <proof>

**lemma** *poly-subdegree-code* [code]:  $\text{poly-subdegree } p = \text{prefix-length } ((=) \ 0) \ (\text{coeffs } p)$   
 <proof>

**end**

Truncation of formal power series: all monomials  $cx^k$  with  $k \geq n$  are removed; the remainder is a polynomial of degree at most  $n - 1$ .

**lift-definition** *truncate-fps* ::  $\text{nat} \Rightarrow 'a \text{ fps} \Rightarrow 'a :: \text{zero poly}$  **is**  
 $\lambda n \ F \ k. \text{if } k \geq n \text{ then } 0 \text{ else } \text{fps-nth } F \ k$   
 <proof>

**lemma** *coeff-truncate-fps'* [simp]:  
 $k \geq n \implies \text{coeff } (\text{truncate-fps } n \ F) \ k = 0$   
 $k < n \implies \text{coeff } (\text{truncate-fps } n \ F) \ k = \text{fps-nth } F \ k$   
 <proof>

**lemma** *coeff-truncate-fps*:  $\text{coeff } (\text{truncate-fps } n \ F) \ k = (\text{if } k < n \text{ then } \text{fps-nth } F \ k \text{ else } 0)$   
 <proof>

**lemma** *truncate-0-fps* [simp]:  $\text{truncate-fps } 0 \ F = 0$   
 <proof>

**lemma** *degree-truncate-fps*:  $n > 0 \implies \text{degree } (\text{truncate-fps } n \ F) < n$   
 $\langle \text{proof} \rangle$

**lemma** *truncate-fps-0* [simp]:  $\text{truncate-fps } n \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *truncate-fps-add*:  $\text{truncate-fps } n \ (f + g) = \text{truncate-fps } n \ f + \text{truncate-fps } n \ g$   
 $\langle \text{proof} \rangle$

**lemma** *truncate-fps-diff*:  $\text{truncate-fps } n \ (f - g) = \text{truncate-fps } n \ f - \text{truncate-fps } n \ g$   
 $\langle \text{proof} \rangle$

**lemma** *truncate-fps-uminus*:  $\text{truncate-fps } n \ (-f) = -\text{truncate-fps } n \ f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-of-poly-truncate* [simp]:  $\text{fps-of-poly } (\text{truncate-fps } n \ f) = \text{fps-cutoff } n \ f$   
 $\langle \text{proof} \rangle$

end

## 7 A formalization of formal Laurent series

**theory** *Formal-Laurent-Series*

**imports**

*Polynomial-FPS*

**begin**

### 7.1 The type of formal Laurent series

#### 7.1.1 Type definition

**typedef** (overloaded) *'a fls* =  $\{f::\text{int} \Rightarrow 'a::\text{zero}. \forall_{\infty} n::\text{nat}. f \ (- \ \text{int } n) = 0\}$   
**morphisms** *fls-nth Abs-fls*  
 $\langle \text{proof} \rangle$

**setup-lifting** *type-definition-fls*

**unbundle** *fps-syntax*

**notation** *fls-nth* (infixl  $\langle \$\$ \rangle$  75)

**lemmas** *fls-eqI* = *iffD1*[*OF fls-nth-inject*, *OF iffD2*, *OF fun-eq-iff*, *OF allI*]

**lemma** *fls-eq-iff*:  $f = g \longleftrightarrow (\forall n. f \ \$\$ \ n = g \ \$\$ \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *nth-Abs-fls* [simp]:  $\forall_{\infty} n. f \ (- \ \text{int } n) = 0 \implies \text{Abs-fls } f \ \$\$ \ n = f \ n$

$\langle \text{proof} \rangle$

**lemmas** *nth-Abs-fls-finite-nonzero-neg-nth* = *nth-Abs-fls*[*OF iffD2*, *OF eventually-cofinite*]

**lemmas** *nth-Abs-fls-ex-nat-lower-bound* = *nth-Abs-fls*[*OF iffD2*, *OF MOST-nat*]

**lemmas** *nth-Abs-fls-nat-lower-bound* = *nth-Abs-fls-ex-nat-lower-bound*[*OF exI*]

**lemma** *nth-Abs-fls-ex-lower-bound*:

**assumes**  $\exists N. \forall n < N. f\ n = 0$

**shows**  $\text{Abs-fls } f\ \$\$ n = f\ n$

$\langle \text{proof} \rangle$

**lemmas** *nth-Abs-fls-lower-bound* = *nth-Abs-fls-ex-lower-bound*[*OF exI*]

**lemmas** *MOST-fls-neg-nth-eq-0* [*simp*] = *CollectD*[*OF fls-nth*]

**lemmas** *fls-finite-nonzero-neg-nth* = *iffD1*[*OF eventually-cofinite MOST-fls-neg-nth-eq-0*]

**lemma** *fls-nth-vanishes-below-natE*:

**fixes**  $f :: 'a :: \text{zero } \text{fls}$

**obtains**  $N :: \text{nat}$

**where**  $\forall n > N. f\ \$\$ (-\text{int } n) = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-nth-vanishes-belowE*:

**fixes**  $f :: 'a :: \text{zero } \text{fls}$

**obtains**  $N :: \text{int}$

**where**  $\forall n < N. f\ \$\$ n = 0$

$\langle \text{proof} \rangle$

### 7.1.2 Definition of basic Laurent series

**instantiation** *fls* :: (*zero*) *zero*

**begin**

**lift-definition** *zero-fls* :: '*a fls* is  $\lambda\cdot. 0$ '  $\langle \text{proof} \rangle$

**instance**  $\langle \text{proof} \rangle$

**end**

**lemma** *fls-zero-nth* [*simp*]:  $0\ \$\$ n = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-zero-eqI*:  $(\bigwedge n. f\ \$\$ n = 0) \implies f = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-nonzeroI*:  $f\ \$\$ n \neq 0 \implies f \neq 0$

$\langle \text{proof} \rangle$

**lemma** *fls-nonzero-nth*:  $f \neq 0 \longleftrightarrow (\exists n. f\ \$\$ n \neq 0)$

$\langle \text{proof} \rangle$

**lemma** *fls-trivial-delta-eq-zero* [simp]:  $b = 0 \implies \text{Abs-fls } (\lambda n. \text{ if } n=a \text{ then } b \text{ else } 0) = 0$

*<proof>*

**lemma** *fls-delta-nth* [simp]:

$\text{Abs-fls } (\lambda n. \text{ if } n=a \text{ then } b \text{ else } 0) \ \$\$ \ n = (\text{ if } n=a \text{ then } b \text{ else } 0)$

*<proof>*

**instantiation** *fls* :: ( $\{zero, one\}$ ) *one*

**begin**

**lift-definition** *one-fls* :: 'a *fls* **is**  $\lambda k. \text{ if } k = 0 \text{ then } 1 \text{ else } 0$

*<proof>*

**instance** *<proof>*

**end**

**lemma** *fls-one-nth* [simp]:

$1 \ \$\$ \ n = (\text{ if } n = 0 \text{ then } 1 \text{ else } 0)$

*<proof>*

**instance** *fls* :: (*zero-neq-one*) *zero-neq-one*

*<proof>*

**definition** *fls-const* :: 'a::*zero*  $\Rightarrow$  'a *fls*

**where** *fls-const*  $c \equiv \text{Abs-fls } (\lambda n. \text{ if } n = 0 \text{ then } c \text{ else } 0)$

**lemma** *fls-const-nth* [simp]: *fls-const*  $c \ \$\$ \ n = (\text{ if } n = 0 \text{ then } c \text{ else } 0)$

*<proof>*

**lemma** *fls-const-0* [simp]: *fls-const*  $0 = 0$

*<proof>*

**lemma** *fls-const-nonzero*:  $c \neq 0 \implies \text{fls-const } c \neq 0$

*<proof>*

**lemma** *fls-const-eq-0-iff* [simp]: *fls-const*  $c = 0 \longleftrightarrow c = 0$

*<proof>*

**lemma** *fls-const-1* [simp]: *fls-const*  $1 = 1$

*<proof>*

**lemma** *fls-const-eq-1-iff* [simp]: *fls-const*  $c = 1 \longleftrightarrow c = 1$

*<proof>*

**lift-definition** *fls-X* :: 'a::( $\{zero, one\}$ ) *fls*

**is**  $\lambda n. \text{ if } n = 1 \text{ then } 1 \text{ else } 0$

*<proof>*

**lemma** *fls-X-nth* [simp]:

*fls-X*  $\ \$\$ \ n = (\text{ if } n = 1 \text{ then } 1 \text{ else } 0)$

$\langle \text{proof} \rangle$

**lemma** *fls-X-nonzero* [simp]: (*fls-X* :: 'a :: zero-neq-one fls)  $\neq 0$   
 $\langle \text{proof} \rangle$

**lift-definition** *fls-X-inv* :: 'a::{zero,one} fls  
**is**  $\lambda n. \text{if } n = -1 \text{ then } 1 \text{ else } 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-inv-nth* [simp]:  
*fls-X-inv* \$\$  $n = (\text{if } n = -1 \text{ then } 1 \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-inv-nonzero* [simp]: (*fls-X-inv* :: 'a :: zero-neq-one fls)  $\neq 0$   
 $\langle \text{proof} \rangle$

## 7.2 Subdegrees

**lemma** *unique-fls-subdegree*:  
**assumes**  $f \neq 0$   
**shows**  $\exists! n. f \$\$ n \neq 0 \wedge (\forall m. f \$\$ m \neq 0 \longrightarrow n \leq m)$   
 $\langle \text{proof} \rangle$

**definition** *fls-subdegree* :: ('a::zero) fls  $\Rightarrow$  int  
**where** *fls-subdegree*  $f \equiv (\text{if } f = 0 \text{ then } 0 \text{ else } \text{LEAST } n::\text{int}. f \$\$ n \neq 0)$

**lemma** *fls-zero-subdegree* [simp]: *fls-subdegree* 0 = 0  
 $\langle \text{proof} \rangle$

**lemma** *nth-fls-subdegree-nonzero* [simp]:  $f \neq 0 \implies f \$\$ \text{fls-subdegree } f \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *nth-fls-subdegree-zero-iff*:  $(f \$\$ \text{fls-subdegree } f = 0) \longleftrightarrow (f = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-leI*:  $f \$\$ n \neq 0 \implies \text{fls-subdegree } f \leq n$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-leI'*:  $f \$\$ n \neq 0 \implies n \leq m \implies \text{fls-subdegree } f \leq m$   
 $\langle \text{proof} \rangle$

**lemma** *fls-eq0-below-subdegree* [simp]:  $n < \text{fls-subdegree } f \implies f \$\$ n = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-geI*:  $f \neq 0 \implies (\bigwedge k. k < n \implies f \$\$ k = 0) \implies n \leq \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-ge0I*:  $(\bigwedge k. k < 0 \implies f \$\$ k = 0) \implies 0 \leq \text{fls-subdegree } f$

$\langle \text{proof} \rangle$

**lemma** *fls-subdegree-greaterI*:  
**assumes**  $f \neq 0 \wedge k. k \leq n \implies f \ \$\$ k = 0$   
**shows**  $n < \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-eqI*:  $f \ \$\$ n \neq 0 \implies (\bigwedge k. k < n \implies f \ \$\$ k = 0) \implies \text{fls-subdegree } f = n$   
 $\langle \text{proof} \rangle$

**lemma** *fls-delta-subdegree [simp]*:  
 $b \neq 0 \implies \text{fls-subdegree } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) = a$   
 $\langle \text{proof} \rangle$

**lemma** *fls-delta0-subdegree*:  $\text{fls-subdegree } (\text{Abs-fls } (\lambda n. \text{if } n=0 \text{ then } a \text{ else } 0)) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-one-subdegree [simp]*:  $\text{fls-subdegree } 1 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-const-subdegree [simp]*:  $\text{fls-subdegree } (\text{fls-const } c) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-subdegree [simp]*:  $\text{fls-subdegree } (\text{fls-X}::'a::\{\text{zero-neq-one}\} \text{ fls}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-inv-subdegree [simp]*:  $\text{fls-subdegree } (\text{fls-X-inv}::'a::\{\text{zero-neq-one}\} \text{ fls}) = -1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-eq-above-subdegreeI*:  
**assumes**  $N \leq \text{fls-subdegree } f \wedge N \leq \text{fls-subdegree } g \wedge k \geq N. f \ \$\$ k = g \ \$\$ k$   
**shows**  $f = g$   
 $\langle \text{proof} \rangle$

## 7.3 Shifting

### 7.3.1 Shift definition

**definition** *fls-shift* ::  $\text{int} \Rightarrow ('a::\text{zero}) \text{ fls} \Rightarrow 'a \text{ fls}$   
**where**  $\text{fls-shift } n \ f \equiv \text{Abs-fls } (\lambda k. f \ \$\$ (k+n))$

— Since the index set is unbounded in both directions, we can shift in either direction.

**lemma** *fls-shift-nth [simp]*:  $\text{fls-shift } m \ f \ \$\$ n = f \ \$\$ (n+m)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-shift-eq-iff*:  $(\text{fls-shift } m \ f = \text{fls-shift } m \ g) \longleftrightarrow (f = g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-shift-0* [simp]:  $\text{fls-shift } 0 \ f = f$   
 ⟨proof⟩

**lemma** *fls-shift-subdegree* [simp]:  
 $f \neq 0 \implies \text{fls-subdegree } (\text{fls-shift } n \ f) = \text{fls-subdegree } f - n$   
 ⟨proof⟩

**lemma** *fls-shift-fls-shift* [simp]:  $\text{fls-shift } m \ (\text{fls-shift } k \ f) = \text{fls-shift } (k+m) \ f$   
 ⟨proof⟩

**lemma** *fls-shift-fls-shift-reorder*:  
 $\text{fls-shift } m \ (\text{fls-shift } k \ f) = \text{fls-shift } k \ (\text{fls-shift } m \ f)$   
 ⟨proof⟩

**lemma** *fls-shift-zero* [simp]:  $\text{fls-shift } m \ 0 = 0$   
 ⟨proof⟩

**lemma** *fls-shift-eq0-iff*:  $\text{fls-shift } m \ f = 0 \longleftrightarrow f = 0$   
 ⟨proof⟩

**lemma** *fls-shift-eq-1-iff*:  $\text{fls-shift } n \ f = 1 \longleftrightarrow f = \text{fls-shift } (-n) \ 1$   
 ⟨proof⟩

**lemma** *fls-shift-nonneg-subdegree*:  $m \leq \text{fls-subdegree } f \implies \text{fls-subdegree } (\text{fls-shift } m \ f) \geq 0$   
 ⟨proof⟩

**lemma** *fls-shift-delta*:  
 $\text{fls-shift } m \ (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) = \text{Abs-fls } (\lambda n. \text{if } n=a-m \text{ then } b \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fls-shift-const*:  
 $\text{fls-shift } m \ (\text{fls-const } c) = \text{Abs-fls } (\lambda n. \text{if } n=-m \text{ then } c \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fls-shift-const-nth*:  
 $\text{fls-shift } m \ (\text{fls-const } c) \ \$\$ \ n = (\text{if } n=-m \text{ then } c \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fls-X-conv-shift-1*:  $\text{fls-X} = \text{fls-shift } (-1) \ 1$   
 ⟨proof⟩

**lemma** *fls-X-shift-to-one* [simp]:  $\text{fls-shift } 1 \ \text{fls-X} = 1$   
 ⟨proof⟩

**lemma** *fls-X-inv-conv-shift-1*:  $\text{fls-X-inv} = \text{fls-shift } 1 \ 1$   
 ⟨proof⟩



**lemma** *fls-X-inv-shift-to-one* [simp]:  $\text{fls-shift } (-1) \text{ fls-X-inv} = 1$   
 <proof>

**lemma** *fls-X-fls-X-inv-conv*:  
 $\text{fls-X} = \text{fls-shift } (-2) \text{ fls-X-inv fls-X-inv} = \text{fls-shift } 2 \text{ fls-X}$   
 <proof>

### 7.3.2 Base factor

Similarly to the *unit-factor* for formal power series, we can decompose a formal Laurent series as a power of the implied variable times a series of subdegree 0. (See lemma *fls-base-factor-X-power-decompose*.) But we will call this something other *unit-factor* because it will not satisfy assumption *is-unit-unit-factor* of *semidom-divide-unit-factor*.

**definition** *fls-base-factor* ::  $(\text{'a}::\text{zero}) \text{ fls} \Rightarrow \text{'a fls}$   
**where** *fls-base-factor-def*[simp]:  $\text{fls-base-factor } f = \text{fls-shift } (\text{fls-subdegree } f) f$

**lemma** *fls-base-factor-nth*:  $\text{fls-base-factor } f \text{ \textit{\$ \$} } n = f \text{ \textit{\$ \$} } (n + \text{fls-subdegree } f)$   
 <proof>

**lemma** *fls-base-factor-nonzero* [simp]:  $f \neq 0 \implies \text{fls-base-factor } f \neq 0$   
 <proof>

**lemma** *fls-base-factor-subdegree* [simp]:  $\text{fls-subdegree } (\text{fls-base-factor } f) = 0$   
 <proof>

**lemma** *fls-base-factor-base* [simp]:  
 $\text{fls-base-factor } f \text{ \textit{\$ \$} } \text{fls-subdegree } (\text{fls-base-factor } f) = f \text{ \textit{\$ \$} } \text{fls-subdegree } f$   
 <proof>

**lemma** *fls-conv-base-factor-shift-subdegree*:  
 $f = \text{fls-shift } (-\text{fls-subdegree } f) (\text{fls-base-factor } f)$   
 <proof>

**lemma** *fls-base-factor-idem*:  
 $\text{fls-base-factor } (\text{fls-base-factor } (f::\text{'a}::\text{zero fls})) = \text{fls-base-factor } f$   
 <proof>

**lemma** *fls-base-factor-zero*:  $\text{fls-base-factor } (0::\text{'a}::\text{zero fls}) = 0$   
 <proof>

**lemma** *fls-base-factor-zero-iff*:  $\text{fls-base-factor } (f::\text{'a}::\text{zero fls}) = 0 \iff f = 0$   
 <proof>

**lemma** *fls-base-factor-nth-0*:  $f \neq 0 \implies \text{fls-base-factor } f \text{ \textit{\$ \$} } 0 \neq 0$   
 <proof>

**lemma** *fls-base-factor-one*:  $\text{fls-base-factor } (1 :: 'a :: \{\text{zero}, \text{one}\} \text{ fls}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-const*:  $\text{fls-base-factor } (\text{fls-const } c) = \text{fls-const } c$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-delta*:  
 $\text{fls-base-factor } (\text{Abs-fls } (\lambda n. \text{ if } n=a \text{ then } c \text{ else } 0)) = \text{fls-const } c$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-X*:  $\text{fls-base-factor } (\text{fls-X} :: 'a :: \{\text{zero-neq-one}\} \text{ fls}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-X-inv*:  $\text{fls-base-factor } (\text{fls-X-inv} :: 'a :: \{\text{zero-neq-one}\} \text{ fls}) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-shift* [simp]:  $\text{fls-base-factor } (\text{fls-shift } n \ f) = \text{fls-base-factor } f$   
 $\langle \text{proof} \rangle$

## 7.4 Conversion between formal power and Laurent series

### 7.4.1 Converting Laurent to power series

We can truncate a Laurent series at index 0 to create a power series, called the regular part.

**lift-definition** *fls-regpart* ::  $('a :: \text{zero}) \text{ fls} \Rightarrow 'a \text{ fps}$   
**is**  $\lambda f. \text{ Abs-fps } (\lambda n. f \ (\text{int } n))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-nth* [simp]:  $\text{fls-regpart } f \ \$ \ n = f \ \$\$ \ (\text{int } n)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-zero* [simp]:  $\text{fls-regpart } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-one* [simp]:  $\text{fls-regpart } 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-Abs-fls*:  
 $\forall_{\infty} n. F \ (- \ \text{int } n) = 0 \implies \text{fls-regpart } (\text{Abs-fls } F) = \text{Abs-fps } (\lambda n. F \ (\text{int } n))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-delta*:  
 $\text{fls-regpart } (\text{Abs-fls } (\lambda n. \text{ if } n=a \text{ then } b \text{ else } 0)) =$   
 $(\text{if } a < 0 \text{ then } 0 \text{ else } \text{Abs-fps } (\lambda n. \text{ if } n=\text{nat } a \text{ then } b \text{ else } 0))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-const* [simp]:  $\text{fls-regpart } (\text{fls-const } c) = \text{fps-const } c$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-fls-X [simp]: fls-regpart fls-X = fps-X*  
 ⟨proof⟩

**lemma** *fls-regpart-fls-X-inv [simp]: fls-regpart fls-X-inv = 0*  
 ⟨proof⟩

**lemma** *fls-regpart-eq0-imp-nonpos-subdegree:*  
**assumes** *fls-regpart f = 0*  
**shows** *fls-subdegree f ≤ 0*  
 ⟨proof⟩

**lemma** *fls-subdegree-lt-fls-regpart-subdegree:*  
*fls-subdegree f ≤ int (subdegree (fls-regpart f))*  
 ⟨proof⟩

**lemma** *fls-regpart-subdegree-conv:*  
**assumes** *fls-subdegree f ≥ 0*  
**shows** *subdegree (fls-regpart f) = nat (fls-subdegree f)*  
 — This is the best we can do since if the subdegree is negative, we might still have the bad luck that the term at index 0 is equal to 0.  
 ⟨proof⟩

**lemma** *fls-eq-conv-fps-eqI:*  
**assumes** *0 ≤ fls-subdegree f 0 ≤ fls-subdegree g fls-regpart f = fls-regpart g*  
**shows** *f = g*  
 ⟨proof⟩

**lemma** *fls-regpart-shift-conv-fps-shift:*  
*m ≥ 0 ⇒ fls-regpart (fls-shift m f) = fps-shift (nat m) (fls-regpart f)*  
 ⟨proof⟩

**lemma** *fps-shift-fls-regpart-conv-fls-shift:*  
*fps-shift m (fls-regpart f) = fls-regpart (fls-shift m f)*  
 ⟨proof⟩

**lemma** *fps-unit-factor-fls-regpart:*  
*fls-subdegree f ≥ 0 ⇒ unit-factor (fls-regpart f) = fls-regpart (fls-base-factor f)*  
 ⟨proof⟩

The terms below the zeroth form a polynomial in the inverse of the implied variable, called the principle part.

**lift-definition** *fls-prpart :: ('a::zero) fls ⇒ 'a poly*  
**is** *λf. Abs-poly (λn. if n = 0 then 0 else f (− int n))*  
 ⟨proof⟩

**lemma** *fls-prpart-coeff [simp]: coeff (fls-prpart f) n = (if n = 0 then 0 else f \$\$ (− int n))*  
 ⟨proof⟩

**lemma** *fls-prpart-eq0-iff*:  $(\text{fls-prpart } f = 0) \longleftrightarrow (\text{fls-subdegree } f \geq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart0* [simp]:  $\text{fls-prpart } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-one* [simp]:  $\text{fls-prpart } 1 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-delta*:  
 $\text{fls-prpart } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) =$   
 $(\text{if } a < 0 \text{ then Poly (replicate (nat } (-a)) \ 0 \ @ \ [b]) \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-const* [simp]:  $\text{fls-prpart } (\text{fls-const } c) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-X* [simp]:  $\text{fls-prpart } \text{fls-X} = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-X-inv*:  $\text{fls-prpart } \text{fls-X-inv} = [:0, 1:]$   
 $\langle \text{proof} \rangle$

**lemma** *degree-fls-prpart* [simp]:  
 $\text{degree } (\text{fls-prpart } f) = \text{nat } (-\text{fls-subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-shift*:  
**assumes**  $m \leq 0$   
**shows**  $\text{fls-prpart } (\text{fls-shift } m \ f) = \text{pCons } 0 \ (\text{poly-shift } (\text{Suc } (\text{nat } (-m)))$   
 $(\text{fls-prpart } f))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-prpart-base-factor*:  $\text{fls-prpart } (\text{fls-base-factor } f) = 0$   
 $\langle \text{proof} \rangle$

The essential data of a formal Laurant series resides from the subdegree up.

**abbreviation** *fls-base-factor-to-fps* ::  $('a::\text{zero}) \text{ fls} \Rightarrow 'a \text{ fps}$   
**where**  $\text{fls-base-factor-to-fps } f \equiv \text{fls-regpart } (\text{fls-base-factor } f)$

**lemma** *fls-base-factor-to-fps-conv-fps-shift*:  
**assumes**  $\text{fls-subdegree } f \geq 0$   
**shows**  $\text{fls-base-factor-to-fps } f = \text{fps-shift } (\text{nat } (\text{fls-subdegree } f)) \ (\text{fls-regpart } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-nth*:  
 $\text{fls-base-factor-to-fps } f \ \$ \ n = f \ \$\$ \ (\text{fls-subdegree } f + \text{int } n)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-base*:  $f \neq 0 \implies \text{fls-base-factor-to-fps } f \ \$ \ 0 \neq 0$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-nonzero*:  $f \neq 0 \implies \text{fls-base-factor-to-fps } f \neq 0$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-subdegree* [simp]:  $\text{subdegree } (\text{fls-base-factor-to-fps } f) = 0$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-trivial*:  
 $\text{fls-subdegree } f = 0 \implies \text{fls-base-factor-to-fps } f = \text{fls-regpart } f$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-zero*:  $\text{fls-base-factor-to-fps } 0 = 0$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-one*:  $\text{fls-base-factor-to-fps } 1 = 1$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-delta*:  
 $\text{fls-base-factor-to-fps } (\text{Abs-fls } (\lambda n. \text{ if } n=a \text{ then } c \text{ else } 0)) = \text{fps-const } c$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-const*:  
 $\text{fls-base-factor-to-fps } (\text{fls-const } c) = \text{fps-const } c$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-X*:  
 $\text{fls-base-factor-to-fps } (\text{fls-X}::'a::\{\text{zero-neq-one}\} \text{ fls}) = 1$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-X-inv*:  
 $\text{fls-base-factor-to-fps } (\text{fls-X-inv}::'a::\{\text{zero-neq-one}\} \text{ fls}) = 1$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-shift*:  
 $\text{fls-base-factor-to-fps } (\text{fls-shift } m \ f) = \text{fls-base-factor-to-fps } f$   
 ⟨proof⟩

**lemma** *fls-base-factor-to-fps-base-factor*:  
 $\text{fls-base-factor-to-fps } (\text{fls-base-factor } f) = \text{fls-base-factor-to-fps } f$   
 ⟨proof⟩

**lemma** *fps-unit-factor-fls-base-factor*:  
 $\text{unit-factor } (\text{fls-base-factor-to-fps } f) = \text{fls-base-factor-to-fps } f$   
 ⟨proof⟩

## 7.4.2 Converting power to Laurent series

We can extend a power series by 0s below to create a Laurent series.

**definition**  $\text{fps-to-fls} :: ('a::\text{zero}) \text{fps} \Rightarrow 'a \text{fls}$   
**where**  $\text{fps-to-fls } f \equiv \text{Abs-fls } (\lambda k::\text{int}. \text{if } k < 0 \text{ then } 0 \text{ else } f \$ (\text{nat } k))$

**lemma**  $\text{fps-to-fls-nth}$  [simp]:  
 $(\text{fps-to-fls } f) \$\$ n = (\text{if } n < 0 \text{ then } 0 \text{ else } f \$ (\text{nat } n))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-eq-imp-fps-eq}$ :  
**assumes**  $\text{fps-to-fls } f = \text{fps-to-fls } g$   
**shows**  $f = g$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-eq-iff}$  [simp]:  $\text{fps-to-fls } f = \text{fps-to-fls } g \longleftrightarrow f = g$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-zero-to-fls}$  [simp]:  $\text{fps-to-fls } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-nonzeroI}$ :  $f \neq 0 \implies \text{fps-to-fls } f \neq 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-one-to-fls}$  [simp]:  $\text{fps-to-fls } 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-Abs-fps}$ :  
 $\text{fps-to-fls } (\text{Abs-fps } F) = \text{Abs-fls } (\lambda n. \text{if } n < 0 \text{ then } 0 \text{ else } F (\text{nat } n))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-delta-to-fls}$ :  
 $\text{fps-to-fls } (\text{Abs-fps } (\lambda n. \text{if } n = a \text{ then } b \text{ else } 0)) = \text{Abs-fls } (\lambda n. \text{if } n = \text{int } a \text{ then } b \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-const-to-fls}$  [simp]:  $\text{fps-to-fls } (\text{fps-const } c) = \text{fls-const } c$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-X-to-fls}$  [simp]:  $\text{fps-to-fls } \text{fps-X } f = \text{fls-X } f$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-eq-0-iff}$  [simp]:  $(\text{fps-to-fls } f = 0) \longleftrightarrow (f = 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-to-fls-eq-1-iff}$  [simp]:  $\text{fps-to-fls } f = 1 \longleftrightarrow f = 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-subdegree-fls-to-fps-gt0}$ :  $\text{fls-subdegree } (\text{fps-to-fls } f) \geq 0$

$\langle \text{proof} \rangle$

**lemma** *fls-subdegree-fls-to-fps*:  $\text{fls-subdegree } (\text{fps-to-fls } f) = \text{int } (\text{subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-shift-to-fls* [simp]:  
 $n \leq \text{subdegree } f \implies \text{fps-to-fls } (\text{fps-shift } n \ f) = \text{fls-shift } (\text{int } n) (\text{fps-to-fls } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-fps-to-fls*:  $\text{fls-base-factor } (\text{fps-to-fls } f) = \text{fps-to-fls } (\text{unit-factor } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-to-fls-trivial* [simp]:  
 $\text{fls-subdegree } f \geq 0 \implies \text{fps-to-fls } (\text{fls-regpart } f) = f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-regpart-fps-trivial* [simp]:  $\text{fls-regpart } (\text{fps-to-fls } f) = f$   
 $\langle \text{proof} \rangle$

**lemma** *fps-to-fls-base-factor-to-fps*:  
 $\text{fps-to-fls } (\text{fls-base-factor-to-fps } f) = \text{fls-base-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-conv-base-factor-to-fps-shift-subdegree*:  
 $f = \text{fls-shift } (-\text{fls-subdegree } f) (\text{fps-to-fls } (\text{fls-base-factor-to-fps } f))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-to-fls*:  
 $\text{fls-base-factor-to-fps } (\text{fps-to-fls } f) = \text{unit-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-as-fps*:  
fixes  $f :: 'a :: \text{zero fls}$  and  $n :: \text{int}$   
assumes  $n: n \geq -\text{fls-subdegree } f$   
obtains  $f'$  where  $f = \text{fls-shift } n (\text{fps-to-fls } f')$   
 $\langle \text{proof} \rangle$

**lemma** *fls-as-fps'*:  
fixes  $f :: 'a :: \text{zero fls}$  and  $n :: \text{int}$   
assumes  $n: n \geq -\text{fls-subdegree } f$   
shows  $\exists f'. f = \text{fls-shift } n (\text{fps-to-fls } f')$   
 $\langle \text{proof} \rangle$

**abbreviation**

$\text{fls-regpart-as-fls } f \equiv \text{fps-to-fls } (\text{fls-regpart } f)$

**abbreviation**

$\text{fls-prpart-as-fls } f \equiv$   
 $\text{fls-shift } (-\text{fls-subdegree } f) (\text{fps-to-fls } (\text{fps-of-poly } (\text{reflect-poly } (\text{fls-prpart } f))))$

**lemma** *fls-regpart-as-fls-nth*:  
 $fls-regpart-as-fls\ f\ \$\$ n = (if\ n < 0\ then\ 0\ else\ f\ \$\$ n)$   
 $\langle proof \rangle$

**lemma** *fls-regpart-idem*:  
 $fls-regpart\ (fls-regpart-as-fls\ f) = fls-regpart\ f$   
 $\langle proof \rangle$

**lemma** *fls-prpart-as-fls-nth*:  
 $fls-prpart-as-fls\ f\ \$\$ n = (if\ n < 0\ then\ f\ \$\$ n\ else\ 0)$   
 $\langle proof \rangle$

**lemma** *fls-prpart-idem* [simp]:  $fls-prpart\ (fls-prpart-as-fls\ f) = fls-prpart\ f$   
 $\langle proof \rangle$

**lemma** *fls-regpart-prpart*:  $fls-regpart\ (fls-prpart-as-fls\ f) = 0$   
 $\langle proof \rangle$

**lemma** *fls-prpart-regpart*:  $fls-prpart\ (fls-regpart-as-fls\ f) = 0$   
 $\langle proof \rangle$

## 7.5 Algebraic structures

### 7.5.1 Addition

**instantiation** *fls* :: (*monoid-add*) *plus*  
**begin**  
**lift-definition** *plus-fls* :: '*a fls*  $\Rightarrow$  '*a fls*  $\Rightarrow$  '*a fls* **is**  $\lambda f\ g\ n. f\ n + g\ n$   
 $\langle proof \rangle$   
**instance**  $\langle proof \rangle$   
**end**

**lemma** *fls-plus-nth* [simp]:  $(f + g)\ \$\$ n = f\ \$\$ n + g\ \$\$ n$   
 $\langle proof \rangle$

**lemma** *fls-plus-const*:  $fls-const\ x + fls-const\ y = fls-const\ (x+y)$   
 $\langle proof \rangle$

**lemma** *fls-plus-subdegree*:  
 $f + g \neq 0 \implies fls-subdegree\ (f + g) \geq \min\ (fls-subdegree\ f)\ (fls-subdegree\ g)$   
 $\langle proof \rangle$

**lemma** *fls-shift-plus* [simp]:  
 $fls-shift\ m\ (f + g) = (fls-shift\ m\ f) + (fls-shift\ m\ g)$   
 $\langle proof \rangle$

**lemma** *fls-regpart-plus* [simp]:  $fls-regpart\ (f + g) = fls-regpart\ f + fls-regpart\ g$   
 $\langle proof \rangle$



**lemma** *fls-prpart-plus* [simp]:  $\text{fls-prpart } (f + g) = \text{fls-prpart } f + \text{fls-prpart } g$   
 ⟨proof⟩

**lemma** *fls-decompose-reg-pr-parts*:  
**fixes**  $f :: 'a :: \text{monoid-add fls}$   
**defines**  $R \equiv \text{fls-regpart-as-fls } f$   
**and**  $P \equiv \text{fls-prpart-as-fls } f$   
**shows**  $f = P + R$   
**and**  $f = R + P$   
 ⟨proof⟩

**lemma** *fps-to-fls-plus* [simp]:  $\text{fps-to-fls } (f + g) = \text{fps-to-fls } f + \text{fps-to-fls } g$   
 ⟨proof⟩

**instance** *fls* :: (*monoid-add*) *monoid-add*  
 ⟨proof⟩

**instance** *fls* :: (*comm-monoid-add*) *comm-monoid-add*  
 ⟨proof⟩

**lemma** *fls-nth-sum*:  $\text{fls-nth } (\sum_{x \in A. f \ x}) \ n = (\sum_{x \in A. \text{fls-nth } (f \ x) \ n})$   
 ⟨proof⟩

## 7.5.2 Subtraction and negatives

**instantiation** *fls* :: (*group-add*) *minus*  
**begin**  
**lift-definition** *minus-fls* :: '*a fls*  $\Rightarrow$  '*a fls*  $\Rightarrow$  '*a fls* **is**  $\lambda f \ g \ n. f \ n - g \ n$   
 ⟨proof⟩  
**instance** ⟨proof⟩  
**end**

**lemma** *fls-minus-nth* [simp]:  $(f - g) \ \$\$ \ n = f \ \$\$ \ n - g \ \$\$ \ n$   
 ⟨proof⟩

**lemma** *fls-minus-const*:  $\text{fls-const } x - \text{fls-const } y = \text{fls-const } (x - y)$   
 ⟨proof⟩

**lemma** *fls-subdegree-minus*:  
 $f - g \neq 0 \implies \text{fls-subdegree } (f - g) \geq \min (\text{fls-subdegree } f) (\text{fls-subdegree } g)$   
 ⟨proof⟩

**lemma** *fls-shift-minus* [simp]:  $\text{fls-shift } m \ (f - g) = (\text{fls-shift } m \ f) - (\text{fls-shift } m \ g)$   
 ⟨proof⟩

**lemma** *fls-regpart-minus* [simp]:  $\text{fls-regpart } (f - g) = \text{fls-regpart } f - \text{fls-regpart } g$   
 ⟨proof⟩

**lemma** *fls-prpart-minus* [simp] : *fls-prpart* ( $f - g$ ) = *fls-prpart*  $f - fls-prpart$   $g$   
 ⟨proof⟩

**lemma** *fps-to-fls-minus* [simp]: *fps-to-fls* ( $f - g$ ) = *fps-to-fls*  $f - fps-to-fls$   $g$   
 ⟨proof⟩

**instantiation** *fls* :: (group-add) *uminus*  
**begin**  
**lift-definition** *uminus-fls* :: 'a *fls*  $\Rightarrow$  'a *fls* **is**  $\lambda f n. - f n$   
 ⟨proof⟩  
**instance** ⟨proof⟩  
**end**

**lemma** *fls-uminus-nth* [simp]:  $(-f) \$\$ n = - (f \$\$ n)$   
 ⟨proof⟩

**lemma** *fls-const-uminus*[simp]: *fls-const*  $(-x) = -fls-const$   $x$   
 ⟨proof⟩

**lemma** *fls-shift-uminus* [simp]: *fls-shift*  $m (-f) = - (fls-shift$   $m f)$   
 ⟨proof⟩

**lemma** *fls-regpart-uminus* [simp]: *fls-regpart*  $(-f) = - fls-regpart$   $f$   
 ⟨proof⟩

**lemma** *fls-prpart-uminus* [simp] : *fls-prpart*  $(-f) = - fls-prpart$   $f$   
 ⟨proof⟩

**lemma** *fps-to-fls-uminus* [simp]: *fps-to-fls*  $(-f) = - fps-to-fls$   $f$   
 ⟨proof⟩

**instance** *fls* :: (group-add) *group-add*  
 ⟨proof⟩

**instance** *fls* :: (ab-group-add) *ab-group-add*  
 ⟨proof⟩

**lemma** *fls-uminus-subdegree* [simp]: *fls-subdegree*  $(-f) = fls-subdegree$   $f$   
 ⟨proof⟩

**lemma** *fls-subdegree-minus-sym*: *fls-subdegree*  $(g - f) = fls-subdegree$   $(f - g)$   
 ⟨proof⟩

**lemma** *fls-regpart-sub-prpart*: *fls-regpart*  $(f - fls-prpart-as-fls$   $f) = fls-regpart$   $f$   
 ⟨proof⟩

**lemma** *fls-prpart-sub-regpart*: *fls-prpart*  $(f - fls-regpart-as-fls$   $f) = fls-prpart$   $f$   
 ⟨proof⟩

### 7.5.3 Multiplication

**instantiation** *fls* :: (*{comm-monoid-add, times}*) *times*

**begin**

**definition** *fls-times-def*:

(\*) = ( $\lambda f g.$   
 $\text{fls-shift}$   
 $(- (\text{fls-subdegree } f + \text{fls-subdegree } g))$   
 $(\text{fps-to-fls } (\text{fls-base-factor-to-fps } f * \text{fls-base-factor-to-fps } g))$   
 $)$

**instance** *<proof>*

**end**

**lemma** *fls-times-nth-eq0*:  $n < \text{fls-subdegree } f + \text{fls-subdegree } g \implies (f * g) \$\$ n = 0$

*<proof>*

**lemma** *fls-times-nth*:

**fixes** *f df g dg*

**defines**  $df \equiv \text{fls-subdegree } f$  **and**  $dg \equiv \text{fls-subdegree } g$

**shows**  $(f * g) \$\$ n = (\sum_{i=df+dg..n} f \$\$ (i - dg) * g \$\$ (dg + n - i))$

**and**  $(f * g) \$\$ n = (\sum_{i=df..n-dg} f \$\$ i * g \$\$ (n - i))$

**and**  $(f * g) \$\$ n = (\sum_{i=dg..n-df} f \$\$ (df + i - dg) * g \$\$ (dg + n - df - i))$

**and**  $(f * g) \$\$ n = (\sum_{i=0..n-(df+dg)} f \$\$ (df + i) * g \$\$ (n - df - i))$

*<proof>*

**lemma** *fls-times-base [simp]*:

$(f * g) \$\$ (\text{fls-subdegree } f + \text{fls-subdegree } g) =$

$(f \$\$ \text{fls-subdegree } f) * (g \$\$ \text{fls-subdegree } g)$

*<proof>*

**instance** *fls* :: (*{comm-monoid-add, mult-zero}*) *mult-zero*

*<proof>*

**lemma** *fls-mult-one*:

**fixes** *f* :: 'a::(*{comm-monoid-add, mult-zero, monoid-mult}*) *fls*

**shows**  $1 * f = f$

**and**  $f * 1 = f$

*<proof>*

**lemma** *fls-mult-const-nth [simp]*:

**fixes** *f* :: 'a::(*{comm-monoid-add, mult-zero}*) *fls*

**shows**  $(\text{fls-const } x * f) \$\$ n = x * f \$\$ n$

**and**  $(f * \text{fls-const } x) \$\$ n = f \$\$ n * x$

*<proof>*

**lemma** *fls-const-mult-const[simp]*:

**fixes** *x y* :: 'a::(*{comm-monoid-add, mult-zero}*)

**shows**  $\text{fls-const } x * \text{fls-const } y = \text{fls-const } (x*y)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-add-eq1*:  
**assumes**  $f \neq 0$   $\text{fls-subdegree } f < \text{fls-subdegree } g$   
**shows**  $\text{fls-subdegree } (f + g) = \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-add-eq2*:  
**assumes**  $g \neq 0$   $\text{fls-subdegree } g < \text{fls-subdegree } f$   
**shows**  $\text{fls-subdegree } (f + g) = \text{fls-subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-diff-eq1*:  
**assumes**  $f \neq 0$   $\text{fls-subdegree } f < \text{fls-subdegree } g$   
**shows**  $\text{fls-subdegree } (f - g) = \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-diff-eq2*:  
**assumes**  $g \neq 0$   $\text{fls-subdegree } g < \text{fls-subdegree } f$   
**shows**  $\text{fls-subdegree } (f - g) = \text{fls-subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *nat-minus-fls-subdegree-plus-const-eq*:  
 $\text{nat } (-\text{fls-subdegree } (F + \text{fls-const } c)) = \text{nat } (-\text{fls-subdegree } F)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-subdegree-ge*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, mult-zero}\}$   $\text{fls}$   
**assumes**  $f * g \neq 0$   
**shows**  $\text{fls-subdegree } (f * g) \geq \text{fls-subdegree } f + \text{fls-subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-subdegree-ge-0*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, mult-zero}\}$   $\text{fls}$   
**assumes**  $\text{fls-subdegree } f \geq 0$   $\text{fls-subdegree } g \geq 0$   
**shows**  $\text{fls-subdegree } (f * g) \geq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-nonzero-base-subdegree-eq*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, mult-zero}\}$   $\text{fls}$   
**assumes**  $f \neq 0$   $\text{fls-subdegree } f * g \neq 0$   
**shows**  $\text{fls-subdegree } (f * g) = \text{fls-subdegree } f + \text{fls-subdegree } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-mult [simp]*:  
**fixes**  $f\ g :: 'a :: \text{semiring-no-zero-divisors}$   $\text{fls}$   
**assumes**  $f \neq 0$   $g \neq 0$   
**shows**  $\text{fls-subdegree } (f * g) = \text{fls-subdegree } f + \text{fls-subdegree } g$

$\langle \text{proof} \rangle$

**lemma** *fls-shifted-times-simps*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $f * (\text{fls-shift } n\ g) = \text{fls-shift } n\ (f * g) \ (\text{fls-shift } n\ f) * g = \text{fls-shift } n\ (f * g)$

$\langle \text{proof} \rangle$

**lemma** *fls-shifted-times-transfer*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $\text{fls-shift } n\ f * g = f * \text{fls-shift } n\ g$

$\langle \text{proof} \rangle$

**lemma** *fls-times-both-shifted-simp*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $(\text{fls-shift } m\ f) * (\text{fls-shift } n\ g) = \text{fls-shift } (m+n)\ (f * g)$

$\langle \text{proof} \rangle$

**lemma** *fls-base-factor-mult-base-factor*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $\text{fls-base-factor } (f * \text{fls-base-factor } g) = \text{fls-base-factor } (f * g)$

**and**  $\text{fls-base-factor } (\text{fls-base-factor } f * g) = \text{fls-base-factor } (f * g)$

$\langle \text{proof} \rangle$

**lemma** *fls-base-factor-mult-both-base-factor*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $\text{fls-base-factor } (\text{fls-base-factor } f * \text{fls-base-factor } g) = \text{fls-base-factor } (f * g)$

$\langle \text{proof} \rangle$

$\langle \text{proof} \rangle$

**lemma** *fls-base-factor-mult*:

**fixes**  $f\ g :: 'a::\text{semiring-no-zero-divisors}$  *fls*

**shows**  $\text{fls-base-factor } (f * g) = \text{fls-base-factor } f * \text{fls-base-factor } g$

$\langle \text{proof} \rangle$

**lemma** *fls-times-conv-base-factor-times*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**

$f * g =$

$\text{fls-shift } (-(\text{fls-subdegree } f + \text{fls-subdegree } g))\ (\text{fls-base-factor } f * \text{fls-base-factor } g)$

$\langle \text{proof} \rangle$

$\langle \text{proof} \rangle$

**lemma** *fls-times-base-factor-conv-shifted-times*:

— Convenience form of lemma *fls-times-both-shifted-simp*.

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**

$\text{fls-base-factor } f * \text{fls-base-factor } g = \text{fls-shift } (\text{fls-subdegree } f + \text{fls-subdegree } g)$

$(f * g)$

$\langle \text{proof} \rangle$

**lemma** *fls-times-conv-regpart*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*  
**assumes**  $\text{fls-subdegree } f \geq 0 \text{ fls-subdegree } g \geq 0$   
**shows**  $\text{fls-regpart } (f * g) = \text{fls-regpart } f * \text{fls-regpart } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-mult-conv-unit-factor*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*  
**shows**  
 $\text{fls-base-factor-to-fps } (f * g) =$   
 $\text{unit-factor } (\text{fls-base-factor-to-fps } f * \text{fls-base-factor-to-fps } g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-mult'*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*  
**assumes**  $(f \text{ $$$ fls-subdegree } f) * (g \text{ $$$ fls-subdegree } g) \neq 0$   
**shows**  $\text{fls-base-factor-to-fps } (f * g) = \text{fls-base-factor-to-fps } f * \text{fls-base-factor-to-fps } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-mult*:  
**fixes**  $f\ g :: 'a::\text{semiring-no-zero-divisors}$  *fls*  
**shows**  $\text{fls-base-factor-to-fps } (f * g) = \text{fls-base-factor-to-fps } f * \text{fls-base-factor-to-fps } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-times-conv-fps-times*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*  
**assumes**  $\text{fls-subdegree } f \geq 0 \text{ fls-subdegree } g \geq 0$   
**shows**  $f * g = \text{fps-to-fls } (\text{fls-regpart } f * \text{fls-regpart } g)$   
 $\langle \text{proof} \rangle$

**lemma** *fps-times-conv-fls-times*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fps*  
**shows**  $f * g = \text{fls-regpart } (\text{fps-to-fls } f * \text{fps-to-fls } g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-times-fps-to-fls*:  
**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fps*  
**shows**  $\text{fps-to-fls } (f * g) = \text{fps-to-fls } f * \text{fps-to-fls } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-times-conv-shift*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**shows**  $\text{fls-X} * f = \text{fls-shift } (-1) f f * \text{fls-X} = \text{fls-shift } (-1) f$   
 $\langle \text{proof} \rangle$

**lemmas**  $\text{fls-X-times-comm} = \text{trans-sym}[OF \text{ fls-X-times-conv-shift}]$

**lemma** *fls-subdegree-mult-fls-X*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**assumes**  $f \neq 0$   
**shows**  $\text{fls-subdegree } (\text{fls-X} * f) = \text{fls-subdegree } f + 1$   
**and**  $\text{fls-subdegree } (f * \text{fls-X}) = \text{fls-subdegree } f + 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-fls-X-nonzero*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**assumes**  $f \neq 0$   
**shows**  $\text{fls-X} * f \neq 0$   
**and**  $f * \text{fls-X} \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-mult-fls-X*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{monoid-mult}, \text{mult-zero}\}$  *fls*  
**shows**  $\text{fls-base-factor } (\text{fls-X} * f) = \text{fls-base-factor } f$   
**and**  $\text{fls-base-factor } (f * \text{fls-X}) = \text{fls-base-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-inv-times-conv-shift*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**shows**  $\text{fls-X-inv} * f = \text{fls-shift } 1 \ f \ f * \text{fls-X-inv} = \text{fls-shift } 1 \ f$   
 $\langle \text{proof} \rangle$

**lemmas** *fls-X-inv-times-comm* = *trans-sym*[*OF fls-X-inv-times-conv-shift*]

**lemma** *fls-subdegree-mult-fls-X-inv*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**assumes**  $f \neq 0$   
**shows**  $\text{fls-subdegree } (\text{fls-X-inv} * f) = \text{fls-subdegree } f - 1$   
**and**  $\text{fls-subdegree } (f * \text{fls-X-inv}) = \text{fls-subdegree } f - 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-fls-X-inv-nonzero*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{monoid-mult}\}$  *fls*  
**assumes**  $f \neq 0$   
**shows**  $\text{fls-X-inv} * f \neq 0$   
**and**  $f * \text{fls-X-inv} \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-mult-fls-X-inv*:  
**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{monoid-mult}, \text{mult-zero}\}$  *fls*  
**shows**  $\text{fls-base-factor } (\text{fls-X-inv} * f) = \text{fls-base-factor } f$   
**and**  $\text{fls-base-factor } (f * \text{fls-X-inv}) = \text{fls-base-factor } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-assoc-subdegree-ge-0*:

**fixes**  $f\ g\ h :: 'a::\text{semiring-0}\ \text{fls}$   
**assumes**  $\text{fls-subdegree}\ f \geq 0\ \text{fls-subdegree}\ g \geq 0\ \text{fls-subdegree}\ h \geq 0$   
**shows**  $f * g * h = f * (g * h)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-assoc-base-factor*:  
**fixes**  $a\ b\ c :: 'a::\text{semiring-0}\ \text{fls}$   
**shows**  
 $\text{fls-base-factor}\ a * \text{fls-base-factor}\ b * \text{fls-base-factor}\ c =$   
 $\text{fls-base-factor}\ a * (\text{fls-base-factor}\ b * \text{fls-base-factor}\ c)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-distrib-subdegree-ge-0*:  
**fixes**  $f\ g\ h :: 'a::\text{semiring-0}\ \text{fls}$   
**assumes**  $\text{fls-subdegree}\ f \geq 0\ \text{fls-subdegree}\ g \geq 0\ \text{fls-subdegree}\ h \geq 0$   
**shows**  $(f + g) * h = f * h + g * h$   
**and**  $h * (f + g) = h * f + h * g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-distrib-base-factor*:  
**fixes**  $a\ b\ c :: 'a::\text{semiring-0}\ \text{fls}$   
**shows**  
 $\text{fls-base-factor}\ a * (\text{fls-base-factor}\ b + \text{fls-base-factor}\ c) =$   
 $\text{fls-base-factor}\ a * \text{fls-base-factor}\ b + \text{fls-base-factor}\ a * \text{fls-base-factor}\ c$   
 $\langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{semiring-0})\ \text{semiring-0}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-commute-subdegree-ge-0*:  
**fixes**  $f\ g :: 'a::\text{comm-semiring-0}\ \text{fls}$   
**assumes**  $\text{fls-subdegree}\ f \geq 0\ \text{fls-subdegree}\ g \geq 0$   
**shows**  $f * g = g * f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-mult-commute-base-factor*:  
**fixes**  $a\ b\ c :: 'a::\text{comm-semiring-0}\ \text{fls}$   
**shows**  $\text{fls-base-factor}\ a * \text{fls-base-factor}\ b = \text{fls-base-factor}\ b * \text{fls-base-factor}\ a$   
 $\langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{comm-semiring-0})\ \text{comm-semiring-0}$   
 $\langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{semiring-1})\ \text{semiring-1}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-of-nat*:  $(\text{of-nat}\ n :: 'a::\text{semiring-1}\ \text{fls}) = \text{fls-const}\ (\text{of-nat}\ n)$   
 $\langle \text{proof} \rangle$



**lemma** *fls-of-nat-nth*: *of-nat*  $n$   $\$ \$$   $k = (if\ k=0\ then\ of-nat\ n\ else\ 0)$   
 ⟨*proof*⟩

**lemma** *fls-mult-of-nat-nth* [*simp*]:  
 shows  $(of-nat\ k * f)\ \$ \$\ n = of-nat\ k * f\ \$ \$\ n$   
 and  $(f * of-nat\ k)\ \$ \$\ n = f\ \$ \$\ n * of-nat\ k$   
 ⟨*proof*⟩

**lemma** *fls-subdegree-of-nat* [*simp*]: *fls-subdegree* (*of-nat*  $n$ ) = 0  
 ⟨*proof*⟩

**lemma** *fls-shift-of-nat-nth*:  
*fls-shift*  $k$  (*of-nat*  $a$ )  $\$ \$$   $n = (if\ n=-k\ then\ of-nat\ a\ else\ 0)$   
 ⟨*proof*⟩

**lemma** *fls-base-factor-of-nat* [*simp*]:  
*fls-base-factor* (*of-nat*  $n :: 'a::semiring-1\ fls$ ) = (*of-nat*  $n :: 'a\ fls$ )  
 ⟨*proof*⟩

**lemma** *fls-regpart-of-nat* [*simp*]: *fls-regpart* (*of-nat*  $n$ ) = (*of-nat*  $n :: 'a::semiring-1\ fps$ )  
 ⟨*proof*⟩

**lemma** *fls-prpart-of-nat* [*simp*]: *fls-prpart* (*of-nat*  $n$ ) = 0  
 ⟨*proof*⟩

**lemma** *fls-base-factor-to-fps-of-nat*:  
*fls-base-factor-to-fps* (*of-nat*  $n$ ) = (*of-nat*  $n :: 'a::semiring-1\ fps$ )  
 ⟨*proof*⟩

**lemma** *fps-to-fls-of-nat*:  
*fps-to-fls* (*of-nat*  $n$ ) = (*of-nat*  $n :: 'a::semiring-1\ fls$ )  
 ⟨*proof*⟩

**lemma** *fps-to-fls-numeral* [*simp*]: *fps-to-fls* (*numeral*  $n$ ) = *numeral*  $n$   
 ⟨*proof*⟩

**lemma** *fls-const-power*: *fls-const* ( $a \wedge b$ ) = *fls-const*  $a \wedge b$   
 ⟨*proof*⟩

**lemma** *fls-const-numeral* [*simp*]: *fls-const* (*numeral*  $n$ ) = *numeral*  $n$   
 ⟨*proof*⟩

**lemma** *fls-mult-of-numeral-nth* [*simp*]:  
 shows  $(numeral\ k * f)\ \$ \$\ n = numeral\ k * f\ \$ \$\ n$   
 and  $(f * numeral\ k)\ \$ \$\ n = f\ \$ \$\ n * numeral\ k$   
 ⟨*proof*⟩

**lemma** *fls-nth-numeral'* [*simp*]:

$\text{numeral } n \text{ } \$\$ \ 0 = \text{numeral } n \ k \neq 0 \implies \text{numeral } n \text{ } \$\$ \ k = 0$   
 $\langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{comm-semiring-1}) \text{ comm-semiring-1}$   
 $\langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{ring}) \text{ ring} \langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{comm-ring}) \text{ comm-ring} \langle \text{proof} \rangle$

**instance**  $\text{fls} :: (\text{ring-1}) \text{ ring-1} \langle \text{proof} \rangle$

**lemma**  $\text{fls-of-int-nonneg} : (\text{of-int } (\text{int } n) :: 'a::\text{ring-1} \text{ fls}) = \text{fls-const } (\text{of-int } (\text{int } n))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-of-int} : (\text{of-int } i :: 'a::\text{ring-1} \text{ fls}) = \text{fls-const } (\text{of-int } i)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-of-int-nth} : \text{of-int } n \text{ } \$\$ \ k = (\text{if } k=0 \text{ then } \text{of-int } n \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-mult-of-int-nth} \text{ [simp]} :$   
**shows**  $(\text{of-int } k * f) \text{ } \$\$ \ n = \text{of-int } k * f \$\$ n$   
**and**  $(f * \text{of-int } k) \text{ } \$\$ \ n = f \$\$ n * \text{of-int } k$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-subdegree-of-int} \text{ [simp]} : \text{fls-subdegree } (\text{of-int } i) = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-shift-of-int-nth} :$   
 $\text{fls-shift } k \text{ } (\text{of-int } i) \text{ } \$\$ \ n = (\text{if } n=-k \text{ then } \text{of-int } i \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-base-factor-of-int} \text{ [simp]} :$   
 $\text{fls-base-factor } (\text{of-int } i :: 'a::\text{ring-1} \text{ fls}) = (\text{of-int } i :: 'a \text{ fls})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-regpart-of-int} \text{ [simp]} :$   
 $\text{fls-regpart } (\text{of-int } i) = (\text{of-int } i :: 'a::\text{ring-1} \text{ fps})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-prpart-of-int} \text{ [simp]} : \text{fls-prpart } (\text{of-int } n) = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-base-factor-to-fps-of-int} :$   
 $\text{fls-base-factor-to-fps } (\text{of-int } i) = (\text{of-int } i :: 'a::\text{ring-1} \text{ fps})$   
 $\langle \text{proof} \rangle$

**lemma** *fps-to-fls-of-int*:

*fps-to-fls* (*of-int* *i*) = (*of-int* *i* :: 'a::ring-1 *fls*)  
 ⟨*proof*⟩

**instance** *fls* :: (*comm-ring-1*) *comm-ring-1* ⟨*proof*⟩

**instance** *fls* :: (*semiring-no-zero-divisors*) *semiring-no-zero-divisors*  
 ⟨*proof*⟩

**instance** *fls* :: (*semiring-1-no-zero-divisors*) *semiring-1-no-zero-divisors* ⟨*proof*⟩

**instance** *fls* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors* ⟨*proof*⟩

**instance** *fls* :: (*ring-1-no-zero-divisors*) *ring-1-no-zero-divisors* ⟨*proof*⟩

**instance** *fls* :: (*idom*) *idom* ⟨*proof*⟩

**lemma** *semiring-char-fls* [*simp*]:  $\text{CHAR}('a :: \text{comm-semiring-1 } fls) = \text{CHAR}('a)$   
 ⟨*proof*⟩

**instance** *fls* :: ({*semiring-prime-char*, *comm-semiring-1*}) *semiring-prime-char*  
 ⟨*proof*⟩

**instance** *fls* :: ({*comm-semiring-prime-char*, *comm-semiring-1*}) *comm-semiring-prime-char*  
 ⟨*proof*⟩

**instance** *fls* :: ({*comm-ring-prime-char*, *comm-semiring-1*}) *comm-ring-prime-char*  
 ⟨*proof*⟩

**instance** *fls* :: ({*idom-prime-char*, *comm-semiring-1*}) *idom-prime-char*  
 ⟨*proof*⟩

**lemma** *fls-subdegree-numeral* [*simp*]:  $\text{fls-subdegree } (\text{numeral } n) = 0$   
 ⟨*proof*⟩

**lemma** *fls-regpart-numeral* [*simp*]:  $\text{fls-regpart } (\text{numeral } n) = \text{numeral } n$   
 ⟨*proof*⟩

## 7.5.4 Powers

**lemma** *fls-subdegree-prod*:

**fixes** *F* :: 'a  $\Rightarrow$  'b :: *field-char-0 fls*

**assumes**  $\bigwedge x. x \in I \Rightarrow F x \neq 0$

**shows**  $\text{fls-subdegree } (\prod_{x \in I} F x) = (\sum_{x \in I} \text{fls-subdegree } (F x))$

⟨*proof*⟩

**lemma** *fls-subdegree-prod'*:

**fixes** *F* :: 'a  $\Rightarrow$  'b :: *field-char-0 fls*

**assumes**  $\bigwedge x. x \in I \Rightarrow \text{fls-subdegree } (F x) \neq 0$

**shows**  $\text{fls-subdegree } (\prod_{x \in I} F x) = (\sum_{x \in I} \text{fls-subdegree } (F x))$

⟨*proof*⟩

**lemma** *fls-pow-subdegree-ge*:

$f^n \neq 0 \implies \text{fls-subdegree } (f^n) \geq n * \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-pow-nth-below-subdegree*:

$k < n * \text{fls-subdegree } f \implies (f^n) \$\$ k = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-pow-base [simp]*:

$(f^n) \$\$ (n * \text{fls-subdegree } f) = (f \$\$ \text{fls-subdegree } f)^n$   
 $\langle \text{proof} \rangle$

**lemma** *fls-pow-subdegree-eqI*:

$(f \$\$ \text{fls-subdegree } f)^n \neq 0 \implies \text{fls-subdegree } (f^n) = n * \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-unit-base-subdegree-power*:

$x * f \$\$ \text{fls-subdegree } f = 1 \implies \text{fls-subdegree } (f^n) = n * \text{fls-subdegree } f$   
 $f \$\$ \text{fls-subdegree } f * y = 1 \implies \text{fls-subdegree } (f^n) = n * \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-dvd1-subdegree-power*:

$f \$\$ \text{fls-subdegree } f \text{ dvd } 1 \implies \text{fls-subdegree } (f^n) = n * \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-pow-subdegree-ge0*:

**assumes**  $\text{fls-subdegree } f \geq 0$   
**shows**  $\text{fls-subdegree } (f^n) \geq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-pow*:

**fixes**  $f :: 'a::\text{semiring-1-no-zero-divisors fls}$   
**shows**  $\text{fls-subdegree } (f^n) = n * \text{fls-subdegree } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-shifted-pow*:

$(\text{fls-shift } m \ f)^n = \text{fls-shift } (n*m) \ (f^n)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-pow-conv-fps-pow*:

**assumes**  $\text{fls-subdegree } f \geq 0$   
**shows**  $f^n = \text{fps-to-fls } ( (\text{fls-regpart } f)^n )$   
 $\langle \text{proof} \rangle$

**lemma** *fps-to-fls-power*:  $\text{fps-to-fls } (f^n) = \text{fps-to-fls } f^n$

$\langle \text{proof} \rangle$

**lemma** *fls-pow-conv-regpart*:

*fls-subdegree*  $f \geq 0 \implies \text{fls-regpart } (f \wedge n) = (\text{fls-regpart } f) \wedge n$   
 ⟨proof⟩

These two lemmas show that shifting 1 is equivalent to powers of the implied variable.

**lemma** *fls-X-power-conv-shift-1*:  $\text{fls-X} \wedge n = \text{fls-shift } (-n) \ 1$   
 ⟨proof⟩

**lemma** *fls-X-inv-power-conv-shift-1*:  $\text{fls-X-inv} \wedge n = \text{fls-shift } n \ 1$   
 ⟨proof⟩

**abbreviation** *fls-X-intpow*  $\equiv (\lambda i. \text{fls-shift } (-i) \ 1)$

— Unifies *fls-X* and *fls-X-inv* so that *fls-X-intpow* returns the equivalent of the implied variable raised to the supplied integer argument of *fls-X-intpow*, whether positive or negative.

**lemma** *fls-X-intpow-nonzero[simp]*:  $(\text{fls-X-intpow } i :: 'a::\text{zero-neq-one } \text{fls}) \neq 0$   
 ⟨proof⟩

**lemma** *fls-X-intpow-power*:  $(\text{fls-X-intpow } i) \wedge n = \text{fls-X-intpow } (n * i)$   
 ⟨proof⟩

**lemma** *fls-X-power-nth [simp]*:  $\text{fls-X} \wedge n \ \$\$ k = (\text{if } k=n \text{ then } 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fls-X-inv-power-nth [simp]*:  $\text{fls-X-inv} \wedge n \ \$\$ k = (\text{if } k=-n \text{ then } 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *fls-X-pow-nonzero[simp]*:  $(\text{fls-X} \wedge n :: 'a :: \text{semiring-1 } \text{fls}) \neq 0$   
 ⟨proof⟩

**lemma** *fls-X-inv-pow-nonzero[simp]*:  $(\text{fls-X-inv} \wedge n :: 'a :: \text{semiring-1 } \text{fls}) \neq 0$   
 ⟨proof⟩

**lemma** *fls-subdegree-fls-X-pow [simp]*:  $\text{fls-subdegree } (\text{fls-X} \wedge n) = n$   
 ⟨proof⟩

**lemma** *fls-subdegree-fls-X-inv-pow [simp]*:  $\text{fls-subdegree } (\text{fls-X-inv} \wedge n) = -n$   
 ⟨proof⟩

**lemma** *fls-subdegree-fls-X-intpow [simp]*:  
 $\text{fls-subdegree } ((\text{fls-X-intpow } i) :: 'a::\text{zero-neq-one } \text{fls}) = i$   
 ⟨proof⟩

**lemma** *fls-X-pow-conv-fps-X-pow*:  $\text{fls-regpart } (\text{fls-X} \wedge n) = \text{fps-X} \wedge n$   
 ⟨proof⟩

**lemma** *fls-X-inv-pow-regpart*:  $n > 0 \implies \text{fls-regpart } (\text{fls-X-inv} \wedge n) = 0$   
 ⟨proof⟩

**lemma** *fls-X-intpow-regpart*:

*fls-regpart (fls-X-intpow i) = (if i ≥ 0 then fls-X ^ nat i else 0)*  
 ⟨proof⟩

**lemma** *fls-X-power-times-conv-shift*:

*fls-X ^ n \* f = fls-shift (-int n) f f \* fls-X ^ n = fls-shift (-int n) f*  
 ⟨proof⟩

**lemma** *fls-X-inv-power-times-conv-shift*:

*fls-X-inv ^ n \* f = fls-shift (int n) f f \* fls-X-inv ^ n = fls-shift (int n) f*  
 ⟨proof⟩

**lemma** *fls-X-intpow-times-conv-shift*:

**fixes** *f :: 'a::semiring-1 fls*  
**shows** *fls-X-intpow i \* f = fls-shift (-i) f f \* fls-X-intpow i = fls-shift (-i) f*  
 ⟨proof⟩

**lemmas** *fls-X-power-times-comm* = *trans-sym[OF fls-X-power-times-conv-shift]*

**lemmas** *fls-X-inv-power-times-comm* = *trans-sym[OF fls-X-inv-power-times-conv-shift]*

**lemma** *fls-X-intpow-times-comm*:

**fixes** *f :: 'a::semiring-1 fls*  
**shows** *fls-X-intpow i \* f = f \* fls-X-intpow i*  
 ⟨proof⟩

**lemma** *fls-X-intpow-times-fls-X-intpow*:

*(fls-X-intpow i :: 'a::semiring-1 fls) \* fls-X-intpow j = fls-X-intpow (i+j)*  
 ⟨proof⟩

**lemma** *fls-X-intpow-diff-conv-times*:

*fls-X-intpow (i-j) = (fls-X-intpow i :: 'a::semiring-1 fls) \* fls-X-intpow (-j)*  
 ⟨proof⟩

**lemma** *fls-mult-fls-X-power-nonzero*:

**assumes** *f ≠ 0*  
**shows** *fls-X ^ n \* f ≠ 0 f \* fls-X ^ n ≠ 0*  
 ⟨proof⟩

**lemma** *fls-mult-fls-X-inv-power-nonzero*:

**assumes** *f ≠ 0*  
**shows** *fls-X-inv ^ n \* f ≠ 0 f \* fls-X-inv ^ n ≠ 0*  
 ⟨proof⟩

**lemma** *fls-mult-fls-X-intpow-nonzero*:

**fixes** *f :: 'a::semiring-1 fls*  
**assumes** *f ≠ 0*  
**shows** *fls-X-intpow i \* f ≠ 0 f \* fls-X-intpow i ≠ 0*  
 ⟨proof⟩

**lemma** *fls-subdegree-mult-fls-X-power*:

**assumes**  $f \neq 0$

**shows**  $\text{fls-subdegree } (\text{fls-X} \wedge n * f) = \text{fls-subdegree } f + n$

**and**  $\text{fls-subdegree } (f * \text{fls-X} \wedge n) = \text{fls-subdegree } f + n$

*<proof>*

**lemma** *fls-subdegree-mult-fls-X-inv-power*:

**assumes**  $f \neq 0$

**shows**  $\text{fls-subdegree } (\text{fls-X-inv} \wedge n * f) = \text{fls-subdegree } f - n$

**and**  $\text{fls-subdegree } (f * \text{fls-X-inv} \wedge n) = \text{fls-subdegree } f - n$

*<proof>*

**lemma** *fls-subdegree-mult-fls-X-intpow*:

**fixes**  $f :: 'a::\text{semiring-1}$  *fls*

**assumes**  $f \neq 0$

**shows**  $\text{fls-subdegree } (\text{fls-X-intpow } i * f) = \text{fls-subdegree } f + i$

**and**  $\text{fls-subdegree } (f * \text{fls-X-intpow } i) = \text{fls-subdegree } f + i$

*<proof>*

**lemma** *fls-X-shift*:

$\text{fls-shift } (-\text{int } n) \text{ fls-X} = \text{fls-X} \wedge \text{Suc } n$

$\text{fls-shift } (\text{int } (\text{Suc } n)) \text{ fls-X} = \text{fls-X-inv} \wedge n$

*<proof>*

**lemma** *fls-X-inv-shift*:

$\text{fls-shift } (\text{int } n) \text{ fls-X-inv} = \text{fls-X-inv} \wedge \text{Suc } n$

$\text{fls-shift } (-\text{int } (\text{Suc } n)) \text{ fls-X-inv} = \text{fls-X} \wedge n$

*<proof>*

**lemma** *fls-X-power-base-factor*:  $\text{fls-base-factor } (\text{fls-X} \wedge n) = 1$

*<proof>*

**lemma** *fls-X-inv-power-base-factor*:  $\text{fls-base-factor } (\text{fls-X-inv} \wedge n) = 1$

*<proof>*

**lemma** *fls-X-intpow-base-factor*:  $\text{fls-base-factor } (\text{fls-X-intpow } i) = 1$

*<proof>*

**lemma** *fls-base-factor-mult-fls-X-power*:

**shows**  $\text{fls-base-factor } (\text{fls-X} \wedge n * f) = \text{fls-base-factor } f$

**and**  $\text{fls-base-factor } (f * \text{fls-X} \wedge n) = \text{fls-base-factor } f$

*<proof>*

**lemma** *fls-base-factor-mult-fls-X-inv-power*:

**shows**  $\text{fls-base-factor } (\text{fls-X-inv} \wedge n * f) = \text{fls-base-factor } f$

**and**  $\text{fls-base-factor } (f * \text{fls-X-inv} \wedge n) = \text{fls-base-factor } f$

*<proof>*

**lemma** *fls-base-factor-mult-fls-X-intpow*:

**fixes**  $f :: 'a::\text{semiring-1}$  *fls*

**shows**  $\text{fls-base-factor } (\text{fls-X-intpow } i * f) = \text{fls-base-factor } f$

**and**  $\text{fls-base-factor } (f * \text{fls-X-intpow } i) = \text{fls-base-factor } f$

$\langle \text{proof} \rangle$

**lemma** *fls-X-power-base-factor-to-fps*:  $\text{fls-base-factor-to-fps } (\text{fls-X} \wedge n) = 1$

$\langle \text{proof} \rangle$

**lemma** *fls-X-inv-power-base-factor-to-fps*:  $\text{fls-base-factor-to-fps } (\text{fls-X-inv} \wedge n) = 1$

$\langle \text{proof} \rangle$

**lemma** *fls-X-intpow-base-factor-to-fps*:  $\text{fls-base-factor-to-fps } (\text{fls-X-intpow } i) = 1$

$\langle \text{proof} \rangle$

**lemma** *fls-base-factor-X-power-decompose*:

**fixes**  $f :: 'a::\text{semiring-1}$  *fls*

**shows**  $f = \text{fls-base-factor } f * \text{fls-X-intpow } (\text{fls-subdegree } f)$

**and**  $f = \text{fls-X-intpow } (\text{fls-subdegree } f) * \text{fls-base-factor } f$

$\langle \text{proof} \rangle$

**lemma** *fls-normalized-product-of-inverses*:

**assumes**  $f * g = 1$

**shows**  $\text{fls-base-factor } f * \text{fls-base-factor } g =$

$\text{fls-X} \wedge (\text{nat } (-(\text{fls-subdegree } f + \text{fls-subdegree } g)))$

**and**  $\text{fls-base-factor } f * \text{fls-base-factor } g =$

$\text{fls-X-intpow } (-(\text{fls-subdegree } f + \text{fls-subdegree } g))$

$\langle \text{proof} \rangle$

**lemma** *fls-fps-normalized-product-of-inverses*:

**assumes**  $f * g = 1$

**shows**  $\text{fls-base-factor-to-fps } f * \text{fls-base-factor-to-fps } g =$

$\text{fps-X} \wedge (\text{nat } (-(\text{fls-subdegree } f + \text{fls-subdegree } g)))$

$\langle \text{proof} \rangle$

### 7.5.5 Inverses

**abbreviation** *fls-left-inverse* ::

$'a::\{\text{comm-monoid-add}, \text{uminus}, \text{times}\}$  *fls*  $\Rightarrow 'a \Rightarrow 'a$  *fls*

**where**

$\text{fls-left-inverse } f \ x \equiv$

$\text{fls-shift } (\text{fls-subdegree } f) (\text{fps-to-fls } (\text{fps-left-inverse } (\text{fls-base-factor-to-fps } f) \ x))$

**abbreviation** *fls-right-inverse* ::

$'a::\{\text{comm-monoid-add}, \text{uminus}, \text{times}\}$  *fls*  $\Rightarrow 'a \Rightarrow 'a$  *fls*

**where**

$\text{fls-right-inverse } f \ y \equiv$

$\text{fls-shift } (\text{fls-subdegree } f) (\text{fps-to-fls } (\text{fps-right-inverse } (\text{fls-base-factor-to-fps } f) \ y))$



y))

**instantiation** *fls* :: ({*comm-monoid-add*,*uminus*,*times*,*inverse*}) *inverse*

**begin**

**definition** *fls-divide-def*:

$f \text{ div } g =$   
 $\text{fls-shift } (\text{fls-subdegree } g - \text{fls-subdegree } f) ($   
 $\text{fps-to-fls } ((\text{fls-base-factor-to-fps } f) \text{ div } (\text{fls-base-factor-to-fps } g))$   
 $)$

**definition** *fls-inverse-def*:

$\text{inverse } f = \text{fls-shift } (\text{fls-subdegree } f) (\text{fps-to-fls } (\text{inverse } (\text{fls-base-factor-to-fps } f)))$

**instance**  $\langle \text{proof} \rangle$

**end**

**lemma** *fls-inverse-def'*:

$\text{inverse } f = \text{fls-right-inverse } f (\text{inverse } (f \text{ $$ fls-subdegree } f))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-base*:

$\text{fls-left-inverse } f \ x \text{ $$ } (-\text{fls-subdegree } f) = x$   
 $\text{fls-right-inverse } f \ y \text{ $$ } (-\text{fls-subdegree } f) = y$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-base*:

$f \neq 0 \implies \text{inverse } f \text{ $$ } (-\text{fls-subdegree } f) = \text{inverse } (f \text{ $$ fls-subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-starting0*:

**fixes**  $f :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}, \text{uminus}\}$  *fls*  
**and**  $g :: 'b :: \{\text{ab-group-add}, \text{mult-zero}\}$  *fls*  
**shows**  $\text{fls-left-inverse } f \ 0 = 0$   
**and**  $\text{fls-right-inverse } g \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-eq0-imp-starting0*:

$\text{fls-left-inverse } f \ x = 0 \implies x = 0$   
 $\text{fls-right-inverse } f \ x = 0 \implies x = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-eq0-iff*:

**fixes**  $x :: 'a :: \{\text{comm-monoid-add}, \text{mult-zero}, \text{uminus}\}$   
**and**  $y :: 'b :: \{\text{ab-group-add}, \text{mult-zero}\}$   
**shows**  $\text{fls-left-inverse } f \ x = 0 \longleftrightarrow x = 0$   
**and**  $\text{fls-right-inverse } g \ y = 0 \longleftrightarrow y = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-eq0-iff'*:

**fixes**  $f :: 'a::\{ab\text{-group-add}, inverse, mult\text{-zero}\}$   $fls$   
**shows**  $inverse\ f = 0 \longleftrightarrow (inverse\ (f\ \$\$ fls\text{-subdegree}\ f) = 0)$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-eq-0-iff}[simp]$ :  
 $inverse\ f = (0::('a::division\text{-ring})\ fls) \longleftrightarrow f\ \$\$ fls\text{-subdegree}\ f = 0$   
 $\langle proof \rangle$

**lemmas**  $fls\text{-inverse-eq-0}' = iffD2[OF\ fls\text{-inverse-eq-0-iff}]$   
**lemmas**  $fls\text{-inverse-eq-0} = iffD2[OF\ fls\text{-inverse-eq-0-iff}]$

**lemma**  $fls\text{-lr-inverse-const}$ :  
**fixes**  $a :: 'a::\{ab\text{-group-add}, mult\text{-zero}\}$   
**and**  $b :: 'b::\{comm\text{-monoid-add}, mult\text{-zero}, uminus\}$   
**shows**  $fls\text{-left-inverse}\ (fls\text{-const}\ a)\ x = fls\text{-const}\ x$   
**and**  $fls\text{-right-inverse}\ (fls\text{-const}\ b)\ y = fls\text{-const}\ y$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-const}$ :  
**fixes**  $a :: 'a::\{comm\text{-monoid-add}, inverse, mult\text{-zero}, uminus\}$   
**shows**  $inverse\ (fls\text{-const}\ a) = fls\text{-const}\ (inverse\ a)$   
 $\langle proof \rangle$

**lemma**  $fls\text{-lr-inverse-of-nat}$ :  
**fixes**  $x :: 'a::\{ring\text{-1}, mult\text{-zero}\}$   
**and**  $y :: 'b::\{semiring\text{-1}, uminus\}$   
**shows**  $fls\text{-left-inverse}\ (of\text{-nat}\ n)\ x = fls\text{-const}\ x$   
**and**  $fls\text{-right-inverse}\ (of\text{-nat}\ n)\ y = fls\text{-const}\ y$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-of-nat}$ :  
 $inverse\ (of\text{-nat}\ n :: 'a :: \{semiring\text{-1}, inverse, uminus\}\ fls) = fls\text{-const}\ (inverse\ (of\text{-nat}\ n))$   
 $\langle proof \rangle$

**lemma**  $fls\text{-lr-inverse-of-int}$ :  
**fixes**  $x :: 'a::\{ring\text{-1}, mult\text{-zero}\}$   
**shows**  $fls\text{-left-inverse}\ (of\text{-int}\ n)\ x = fls\text{-const}\ x$   
**and**  $fls\text{-right-inverse}\ (of\text{-int}\ n)\ x = fls\text{-const}\ x$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-of-int}$ :  
 $inverse\ (of\text{-int}\ n :: 'a :: \{ring\text{-1}, inverse, uminus\}\ fls) = fls\text{-const}\ (inverse\ (of\text{-int}\ n))$   
 $\langle proof \rangle$

**lemma**  $fls\text{-lr-inverse-zero}$ :  
**fixes**  $x :: 'a::\{ab\text{-group-add}, mult\text{-zero}\}$   
**and**  $y :: 'b::\{comm\text{-monoid-add}, mult\text{-zero}, uminus\}$

**shows**  $\text{fls-left-inverse } 0 \ x = \text{fls-const } x$   
**and**  $\text{fls-right-inverse } 0 \ y = \text{fls-const } y$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-zero-conv-fls-const}$ :  
 $\text{inverse } (0 :: 'a :: \{\text{comm-monoid-add, mult-zero, uminus, inverse}\} \text{ fls}) = \text{fls-const } (\text{inverse } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-zero}'$ :  
**assumes**  $\text{inverse } (0 :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus}\}) = 0$   
**shows**  $\text{inverse } (0 :: 'a \text{ fls}) = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-zero [simp]}$ :  $\text{inverse } (0 :: 'a :: \text{division-ring fls}) = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-base2}$ :  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, mult-zero, uminus, inverse}\} \text{ fls}$   
**shows**  $\text{inverse } f \ \$\$ (-\text{fls-subdegree } f) = \text{inverse } (f \ \$\$ \text{fls-subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-one}$ :  
**fixes**  $x :: 'a :: \{\text{ab-group-add, mult-zero, one}\}$   
**and**  $y :: 'b :: \{\text{comm-monoid-add, mult-zero, uminus, one}\}$   
**shows**  $\text{fls-left-inverse } 1 \ x = \text{fls-const } x$   
**and**  $\text{fls-right-inverse } 1 \ y = \text{fls-const } y$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-one-one}$ :  
 $\text{fls-left-inverse } 1 \ 1 =$   
 $(1 :: 'a :: \{\text{ab-group-add, mult-zero, one}\} \text{ fls})$   
 $\text{fls-right-inverse } 1 \ 1 =$   
 $(1 :: 'b :: \{\text{comm-monoid-add, mult-zero, uminus, one}\} \text{ fls})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-one}$ :  
**assumes**  $\text{inverse } (1 :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus, one}\}) = 1$   
**shows**  $\text{inverse } (1 :: 'a \text{ fls}) = 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-left-inverse-delta}$ :  
**fixes**  $b :: 'a :: \{\text{ab-group-add, mult-zero}\}$   
**assumes**  $b \neq 0$   
**shows**  $\text{fls-left-inverse } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) \ x =$   
 $\text{Abs-fls } (\lambda n. \text{if } n=-a \text{ then } x \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-right-inverse-delta}$ :

**fixes**  $b :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}, \text{uminus}\}$   
**assumes**  $b \neq 0$   
**shows**  $\text{fls-right-inverse } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) \ x =$   
 $\text{Abs-fls } (\lambda n. \text{if } n=-a \text{ then } x \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-delta-nonzero*:  
**fixes**  $b :: 'a::\{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}$   
**assumes**  $b \neq 0$   
**shows**  $\text{inverse } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) =$   
 $\text{Abs-fls } (\lambda n. \text{if } n=-a \text{ then } \text{inverse } b \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-delta*:  
**fixes**  $b :: 'a::\text{division-ring}$   
**shows**  $\text{inverse } (\text{Abs-fls } (\lambda n. \text{if } n=a \text{ then } b \text{ else } 0)) =$   
 $\text{Abs-fls } (\lambda n. \text{if } n=-a \text{ then } \text{inverse } b \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-X*:  
**fixes**  $x :: 'a::\{\text{ab-group-add}, \text{mult-zero}, \text{zero-neq-one}\}$   
**and**  $y :: 'b::\{\text{comm-monoid-add}, \text{uminus}, \text{mult-zero}, \text{zero-neq-one}\}$   
**shows**  $\text{fls-left-inverse } \text{fls-X } x = \text{fls-shift } 1 \ (\text{fls-const } x)$   
**and**  $\text{fls-right-inverse } \text{fls-X } y = \text{fls-shift } 1 \ (\text{fls-const } y)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-X'*:  
**fixes**  $x :: 'a::\{\text{ab-group-add}, \text{mult-zero}, \text{zero-neq-one}, \text{monoid-mult}\}$   
**and**  $y :: 'b::\{\text{comm-monoid-add}, \text{uminus}, \text{mult-zero}, \text{zero-neq-one}, \text{monoid-mult}\}$   
**shows**  $\text{fls-left-inverse } \text{fls-X } x = \text{fls-const } x * \text{fls-X-inv}$   
**and**  $\text{fls-right-inverse } \text{fls-X } y = \text{fls-const } y * \text{fls-X-inv}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-X'*:  
**assumes**  $\text{inverse } 1 = (1 :: 'a::\{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}, \text{zero-neq-one}\})$   
**shows**  $\text{inverse } (\text{fls-X} :: 'a \text{ fls}) = \text{fls-X-inv}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-X*:  $\text{inverse } (\text{fls-X} :: 'a::\text{division-ring fls}) = \text{fls-X-inv}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-X-inv*:  
**fixes**  $x :: 'a::\{\text{ab-group-add}, \text{mult-zero}, \text{zero-neq-one}\}$   
**and**  $y :: 'b::\{\text{comm-monoid-add}, \text{uminus}, \text{mult-zero}, \text{zero-neq-one}\}$   
**shows**  $\text{fls-left-inverse } \text{fls-X-inv } x = \text{fls-shift } (-1) \ (\text{fls-const } x)$   
**and**  $\text{fls-right-inverse } \text{fls-X-inv } y = \text{fls-shift } (-1) \ (\text{fls-const } y)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-X-inv'*:

**fixes**  $x :: 'a::\{ab\text{-group-add}, mult\text{-zero}, zero\text{-neq-one}, monoid\text{-mult}\}$   
**and**  $y :: 'b::\{comm\text{-monoid-add}, uminus, mult\text{-zero}, zero\text{-neq-one}, monoid\text{-mult}\}$   
**shows**  $fls\text{-left-inverse } fls\text{-X-inv } x = fls\text{-const } x * fls\text{-X}$   
**and**  $fls\text{-right-inverse } fls\text{-X-inv } y = fls\text{-const } y * fls\text{-X}$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-X-inv}'$ :  
**assumes**  $inverse\ 1 = (1 :: 'a::\{comm\text{-monoid-add}, inverse, mult\text{-zero}, uminus, zero\text{-neq-one}\})$   
**shows**  $inverse\ (fls\text{-X-inv}::'a\ fls) = fls\text{-X}$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-X-inv}$ :  $inverse\ (fls\text{-X-inv}::'a::division\text{-ring } fls) = fls\text{-X}$   
 $\langle proof \rangle$

**lemma**  $fls\text{-lr-inverse-subdegree}$ :  
**assumes**  $x \neq 0$   
**shows**  $fls\text{-subdegree } (fls\text{-left-inverse } f\ x) = -\ fls\text{-subdegree } f$   
**and**  $fls\text{-subdegree } (fls\text{-right-inverse } f\ x) = -\ fls\text{-subdegree } f$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-subdegree}'$ :  
 $inverse\ (f\ \$\$ fls\text{-subdegree } f) \neq 0 \implies fls\text{-subdegree } (inverse\ f) = -\ fls\text{-subdegree } f$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-subdegree } [simp]$ :  
**fixes**  $f :: 'a::division\text{-ring } fls$   
**shows**  $fls\text{-subdegree } (inverse\ f) = -\ fls\text{-subdegree } f$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-subdegree-base-nonzero}$ :  
**assumes**  $f \neq 0$   $inverse\ (f\ \$\$ fls\text{-subdegree } f) \neq 0$   
**shows**  $inverse\ f\ \$\$ (fls\text{-subdegree } (inverse\ f)) = inverse\ (f\ \$\$ fls\text{-subdegree } f)$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-subdegree-base}$ :  
**fixes**  $f :: 'a::\{ab\text{-group-add}, inverse, mult\text{-zero}\} fls$   
**shows**  $inverse\ f\ \$\$ (fls\text{-subdegree } (inverse\ f)) = inverse\ (f\ \$\$ fls\text{-subdegree } f)$   
 $\langle proof \rangle$

**lemma**  $fls\text{-lr-inverse-subdegree-0}$ :  
**assumes**  $fls\text{-subdegree } f = 0$   
**shows**  $fls\text{-subdegree } (fls\text{-left-inverse } f\ x) \geq 0$   
**and**  $fls\text{-subdegree } (fls\text{-right-inverse } f\ x) \geq 0$   
 $\langle proof \rangle$

**lemma**  $fls\text{-inverse-subdegree-0}$ :  
 $fls\text{-subdegree } f = 0 \implies fls\text{-subdegree } (inverse\ f) \geq 0$   
 $\langle proof \rangle$

**lemma** *fls-lr-inverse-shift-nonzero*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add,mult-zero,uminus}\}$  *fls*

**assumes**  $f \neq 0$

**shows**  $\text{fls-left-inverse } (\text{fls-shift } m \ f) \ x = \text{fls-shift } (-m) \ (\text{fls-left-inverse } f \ x)$

**and**  $\text{fls-right-inverse } (\text{fls-shift } m \ f) \ x = \text{fls-shift } (-m) \ (\text{fls-right-inverse } f \ x)$

*<proof>*

**lemma** *fls-inverse-shift-nonzero*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add,inverse,mult-zero,uminus}\}$  *fls*

**assumes**  $f \neq 0$

**shows**  $\text{inverse } (\text{fls-shift } m \ f) = \text{fls-shift } (-m) \ (\text{inverse } f)$

*<proof>*

**lemma** *fls-inverse-shift*:

**fixes**  $f :: 'a::\text{division-ring}$  *fls*

**shows**  $\text{inverse } (\text{fls-shift } m \ f) = \text{fls-shift } (-m) \ (\text{inverse } f)$

*<proof>*

**lemma** *fls-left-inverse-base-factor*:

**fixes**  $x :: 'a::\{\text{ab-group-add,mult-zero}\}$

**assumes**  $x \neq 0$

**shows**  $\text{fls-left-inverse } (\text{fls-base-factor } f) \ x = \text{fls-base-factor } (\text{fls-left-inverse } f \ x)$

*<proof>*

**lemma** *fls-right-inverse-base-factor*:

**fixes**  $y :: 'a::\{\text{comm-monoid-add,mult-zero,uminus}\}$

**assumes**  $y \neq 0$

**shows**  $\text{fls-right-inverse } (\text{fls-base-factor } f) \ y = \text{fls-base-factor } (\text{fls-right-inverse } f \ y)$

*<proof>*

**lemma** *fls-inverse-base-factor'*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add,inverse,mult-zero,uminus}\}$  *fls*

**assumes**  $\text{inverse } (f \ \$\$ \ \text{fls-subdegree } f) \neq 0$

**shows**  $\text{inverse } (\text{fls-base-factor } f) = \text{fls-base-factor } (\text{inverse } f)$

*<proof>*

**lemma** *fls-inverse-base-factor*:

**fixes**  $f :: 'a::\{\text{ab-group-add,inverse,mult-zero}\}$  *fls*

**shows**  $\text{inverse } (\text{fls-base-factor } f) = \text{fls-base-factor } (\text{inverse } f)$

*<proof>*

**lemma** *fls-lr-inverse-regpart*:

**assumes**  $\text{fls-subdegree } f = 0$

**shows**  $\text{fls-regpart } (\text{fls-left-inverse } f \ x) = \text{fps-left-inverse } (\text{fls-regpart } f) \ x$

**and**  $\text{fls-regpart } (\text{fls-right-inverse } f \ y) = \text{fps-right-inverse } (\text{fls-regpart } f) \ y$

*<proof>*

**lemma** *fls-inverse-regpart*:  
**assumes** *fls-subdegree*  $f = 0$   
**shows**  $\text{fls-regpart } (\text{inverse } f) = \text{inverse } (\text{fls-regpart } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-left-inverse*:  
**fixes**  $x :: 'a :: \{\text{ab-group-add, mult-zero}\}$   
**shows**  $\text{fls-base-factor-to-fps } (\text{fls-left-inverse } f) x =$   
 $\text{fps-left-inverse } (\text{fls-base-factor-to-fps } f) x$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-right-inverse-nonzero*:  
**fixes**  $y :: 'a :: \{\text{comm-monoid-add, mult-zero, uminus}\}$   
**assumes**  $y \neq 0$   
**shows**  $\text{fls-base-factor-to-fps } (\text{fls-right-inverse } f) y =$   
 $\text{fps-right-inverse } (\text{fls-base-factor-to-fps } f) y$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-right-inverse*:  
**fixes**  $y :: 'a :: \{\text{ab-group-add, mult-zero}\}$   
**shows**  $\text{fls-base-factor-to-fps } (\text{fls-right-inverse } f) y =$   
 $\text{fps-right-inverse } (\text{fls-base-factor-to-fps } f) y$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-inverse-nonzero*:  
**fixes**  $f :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus}\}$  *fls*  
**assumes**  $\text{inverse } (f \ \$\$ \text{fls-subdegree } f) \neq 0$   
**shows**  $\text{fls-base-factor-to-fps } (\text{inverse } f) = \text{inverse } (\text{fls-base-factor-to-fps } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-base-factor-to-fps-inverse*:  
**fixes**  $f :: 'a :: \{\text{ab-group-add, inverse, mult-zero}\}$  *fls*  
**shows**  $\text{fls-base-factor-to-fps } (\text{inverse } f) = \text{inverse } (\text{fls-base-factor-to-fps } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-fps-to-fls*:  
**assumes**  $\text{subdegree } f = 0$   
**shows**  $\text{fls-left-inverse } (\text{fps-to-fls } f) x = \text{fps-to-fls } (\text{fps-left-inverse } f) x$   
**and**  $\text{fls-right-inverse } (\text{fps-to-fls } f) x = \text{fps-to-fls } (\text{fps-right-inverse } f) x$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-fps-to-fls*:  
 $\text{subdegree } f = 0 \implies \text{inverse } (\text{fps-to-fls } f) = \text{fps-to-fls } (\text{inverse } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-X-power*:  
**fixes**  $x :: 'a :: \text{ring-1}$   
**and**  $y :: 'b :: \{\text{semiring-1, uminus}\}$   
**shows**  $\text{fls-left-inverse } (\text{fls-} X^n) x = \text{fls-shift } n (\text{fls-const } x)$

**and**  $\text{fls-right-inverse } (\text{fls-X} \wedge n) \ y = \text{fls-shift } n \ (\text{fls-const } y)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-X-power}'$ :  
**fixes**  $x :: 'a::\text{ring-1}$   
**and**  $y :: 'b::\{\text{semiring-1}, \text{uminus}\}$   
**shows**  $\text{fls-left-inverse } (\text{fls-X} \wedge n) \ x = \text{fls-const } x * \text{fls-X-inv} \wedge n$   
**and**  $\text{fls-right-inverse } (\text{fls-X} \wedge n) \ y = \text{fls-const } y * \text{fls-X-inv} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-X-power}'$ :  
**assumes**  $\text{inverse } 1 = (1 :: 'a::\{\text{semiring-1}, \text{uminus}, \text{inverse}\})$   
**shows**  $\text{inverse } ((\text{fls-X} \wedge n) :: 'a \ \text{fls}) = \text{fls-X-inv} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-X-power}$ :  
 $\text{inverse } ((\text{fls-X} :: 'a::\text{division-ring fls}) \wedge n) = \text{fls-X-inv} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-X-inv-power}$ :  
**fixes**  $x :: 'a::\text{ring-1}$   
**and**  $y :: 'b::\{\text{semiring-1}, \text{uminus}\}$   
**shows**  $\text{fls-left-inverse } (\text{fls-X-inv} \wedge n) \ x = \text{fls-shift } (-n) \ (\text{fls-const } x)$   
**and**  $\text{fls-right-inverse } (\text{fls-X-inv} \wedge n) \ y = \text{fls-shift } (-n) \ (\text{fls-const } y)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-X-inv-power}'$ :  
**fixes**  $x :: 'a::\text{ring-1}$   
**and**  $y :: 'b::\{\text{semiring-1}, \text{uminus}\}$   
**shows**  $\text{fls-left-inverse } (\text{fls-X-inv} \wedge n) \ x = \text{fls-const } x * \text{fls-X} \wedge n$   
**and**  $\text{fls-right-inverse } (\text{fls-X-inv} \wedge n) \ y = \text{fls-const } y * \text{fls-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-X-inv-power}'$ :  
**assumes**  $\text{inverse } 1 = (1 :: 'a::\{\text{semiring-1}, \text{uminus}, \text{inverse}\})$   
**shows**  $\text{inverse } ((\text{fls-X-inv} \wedge n) :: 'a \ \text{fls}) = \text{fls-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-inverse-X-inv-power}$ :  
 $\text{inverse } ((\text{fls-X-inv} :: 'a::\text{division-ring fls}) \wedge n) = \text{fls-X} \wedge n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-lr-inverse-X-intpow}$ :  
**fixes**  $x :: 'a::\text{ring-1}$   
**and**  $y :: 'b::\{\text{semiring-1}, \text{uminus}\}$   
**shows**  $\text{fls-left-inverse } (\text{fls-X-intpow } i) \ x = \text{fls-shift } i \ (\text{fls-const } x)$   
**and**  $\text{fls-right-inverse } (\text{fls-X-intpow } i) \ y = \text{fls-shift } i \ (\text{fls-const } y)$   
 $\langle \text{proof} \rangle$



**lemma** *fls-lr-inverse-X-intpow'*:  
**fixes**  $x :: 'a::ring-1$   
**and**  $y :: 'b::\{semiring-1, uminus\}$   
**shows**  $fls-left-inverse (fls-X-intpow i) x = fls-const x * fls-X-intpow (-i)$   
**and**  $fls-right-inverse (fls-X-intpow i) y = fls-const y * fls-X-intpow (-i)$   
 $\langle proof \rangle$

**lemma** *fls-inverse-X-intpow'*:  
**assumes**  $inverse\ 1 = (1 :: 'a::\{semiring-1, uminus, inverse\})$   
**shows**  $inverse (fls-X-intpow i :: 'a\ fls) = fls-X-intpow (-i)$   
 $\langle proof \rangle$

**lemma** *fls-inverse-X-intpow*:  
 $inverse (fls-X-intpow i :: 'a::division-ring\ fls) = fls-X-intpow (-i)$   
 $\langle proof \rangle$

**lemma** *fls-left-inverse*:  
**fixes**  $f :: 'a::ring-1\ fls$   
**assumes**  $x * f\ \$\$ fls-subdegree\ f = 1$   
**shows**  $fls-left-inverse\ f\ x * f = 1$   
 $\langle proof \rangle$

**lemma** *fls-right-inverse*:  
**fixes**  $f :: 'a::ring-1\ fls$   
**assumes**  $f\ \$\$ fls-subdegree\ f * y = 1$   
**shows**  $f * fls-right-inverse\ f\ y = 1$   
 $\langle proof \rangle$

**lemma** *fls-left-inverse-eq-fls-right-inverse*:  
**fixes**  $f :: 'a::ring-1\ fls$   
**assumes**  $x * f\ \$\$ fls-subdegree\ f = 1\ f\ \$\$ fls-subdegree\ f * y = 1$   
— These assumptions imply  $x$  equals  $y$ , but no need to assume that.  
**shows**  $fls-left-inverse\ f\ x = fls-right-inverse\ f\ y$   
 $\langle proof \rangle$

**lemma** *fls-left-inverse-eq-inverse*:  
**fixes**  $f :: 'a::division-ring\ fls$   
**shows**  $fls-left-inverse\ f\ (inverse\ (f\ \$\$ fls-subdegree\ f)) = inverse\ f$   
 $\langle proof \rangle$

**lemma** *fls-right-inverse-eq-inverse*:  
**fixes**  $f :: 'a::division-ring\ fls$   
**shows**  $fls-right-inverse\ f\ (inverse\ (f\ \$\$ fls-subdegree\ f)) = inverse\ f$   
 $\langle proof \rangle$

**lemma** *fls-left-inverse-eq-fls-right-inverse-comm*:  
**fixes**  $f :: 'a::comm-ring-1\ fls$   
**assumes**  $x * f\ \$\$ fls-subdegree\ f = 1$   
**shows**  $fls-left-inverse\ f\ x = fls-right-inverse\ f\ x$   
 $\langle proof \rangle$

**lemma** *fls-left-inverse'*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1\ f \ \$\$ fls-subdegree\ f * y = 1$

— These assumptions imply  $x$  equals  $y$ , but no need to assume that.

**shows**  $fls-right-inverse\ f\ y * f = 1$

*<proof>*

**lemma** *fls-right-inverse'*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1\ f \ \$\$ fls-subdegree\ f * y = 1$

— These assumptions imply  $x$  equals  $y$ , but no need to assume that.

**shows**  $f * fls-left-inverse\ f\ x = 1$

*<proof>*

**lemma** *fls-mult-left-inverse-base-factor*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $x * (f \ \$\$ fls-subdegree\ f) = 1$

**shows**  $fls-left-inverse\ (fls-base-factor\ f)\ x * f = fls-X-intpow\ (fls-subdegree\ f)$

*<proof>*

**lemma** *fls-mult-right-inverse-base-factor*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $(f \ \$\$ fls-subdegree\ f) * y = 1$

**shows**  $f * fls-right-inverse\ (fls-base-factor\ f)\ y = fls-X-intpow\ (fls-subdegree\ f)$

*<proof>*

**lemma** *fls-mult-inverse-base-factor*:

**fixes**  $f :: 'a::division-ring\ fls$

**assumes**  $f \neq 0$

**shows**  $f * inverse\ (fls-base-factor\ f) = fls-X-intpow\ (fls-subdegree\ f)$

*<proof>*

**lemma** *fls-left-inverse-idempotent-ring1*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1\ y * x = 1$

— These assumptions imply  $y$  equals  $f \ \$\$ fls-subdegree\ f$ , but no need to assume that.

**shows**  $fls-left-inverse\ (fls-left-inverse\ f\ x)\ y = f$

*<proof>*

**lemma** *fls-left-inverse-idempotent-comm-ring1*:

**fixes**  $f :: 'a::comm-ring-1\ fls$

**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1$

**shows**  $fls-left-inverse\ (fls-left-inverse\ f\ x)\ (f \ \$\$ fls-subdegree\ f) = f$

*<proof>*

**lemma** *fls-right-inverse-idempotent-ring1*:

**fixes**  $f :: 'a::ring-1\ fls$

**assumes**  $f \text{ fls-subdegree } f * x = 1 \ x * y = 1$   
 — These assumptions imply  $y$  equals  $f \text{ fls-subdegree } f$ , but no need to assume that.  
**shows**  $\text{fls-right-inverse } (\text{fls-right-inverse } f \ x) \ y = f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-right-inverse-idempotent-comm-ring1*:  
**fixes**  $f :: 'a::\text{comm-ring-1} \ \text{fls}$   
**assumes**  $f \text{ fls-subdegree } f * x = 1$   
**shows**  $\text{fls-right-inverse } (\text{fls-right-inverse } f \ x) \ (f \text{ fls-subdegree } f) = f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-unique-ring1*:  
**fixes**  $f \ g :: 'a :: \text{ring-1} \ \text{fls}$   
**assumes**  $fg: f * g = 1 \ g \text{ fls-subdegree } g * f \text{ fls-subdegree } f = 1$   
**shows**  $\text{fls-left-inverse } g \ (f \text{ fls-subdegree } f) = f$   
**and**  $\text{fls-right-inverse } f \ (g \text{ fls-subdegree } g) = g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-unique-divring*:  
**fixes**  $f \ g :: 'a :: \text{division-ring} \ \text{fls}$   
**assumes**  $fg: f * g = 1$   
**shows**  $\text{fls-left-inverse } g \ (f \text{ fls-subdegree } f) = f$   
**and**  $\text{fls-right-inverse } f \ (g \text{ fls-subdegree } g) = g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-minus*:  
**fixes**  $f :: 'a::\text{ring-1} \ \text{fls}$   
**shows**  $\text{fls-left-inverse } (-f) \ (-x) = - \text{fls-left-inverse } f \ x$   
**and**  $\text{fls-right-inverse } (-f) \ (-x) = - \text{fls-right-inverse } f \ x$   
 $\langle \text{proof} \rangle$

**lemma** *fls-inverse-minus [simp]*:  $\text{inverse } (-f) = - \text{inverse } f :: 'a :: \text{division-ring} \ \text{fls}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-mult-ring1*:  
**fixes**  $f \ g :: 'a::\text{ring-1} \ \text{fls}$   
**assumes**  $x: x * f \text{ fls-subdegree } f = 1 \ f \text{ fls-subdegree } f * x = 1$   
**and**  $y: y * g \text{ fls-subdegree } g = 1 \ g \text{ fls-subdegree } g * y = 1$   
**shows**  $\text{fls-left-inverse } (f * g) \ (y*x) = \text{fls-left-inverse } g \ y * \text{fls-left-inverse } f \ x$   
**and**  $\text{fls-right-inverse } (f * g) \ (y*x) = \text{fls-right-inverse } g \ y * \text{fls-right-inverse } f \ x$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-power-ring1*:  
**fixes**  $f :: 'a::\text{ring-1} \ \text{fls}$   
**assumes**  $x: x * f \text{ fls-subdegree } f = 1 \ f \text{ fls-subdegree } f * x = 1$   
**shows**  $\text{fls-left-inverse } (f \wedge n) \ (x \wedge n) = (\text{fls-left-inverse } f \ x) \wedge n$

$\text{fls-right-inverse } (f \wedge n) (x \wedge n) = (\text{fls-right-inverse } f x) \wedge n$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-convert-times-inverse*:  
**fixes**  $f g :: 'a :: \{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}$  *fls*  
**shows**  $f / g = f * \text{inverse } g$   
 $\langle \text{proof} \rangle$

**instance** *fls* :: (*division-ring*) *division-ring*  
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-mult-divring*:  
**fixes**  $f g :: 'a :: \text{division-ring } \text{fls}$   
**and**  $df dg :: \text{int}$   
**defines**  $df \equiv \text{fls-subdegree } f$   
**and**  $dg \equiv \text{fls-subdegree } g$   
**shows**  $\text{fls-left-inverse } (f * g) (\text{inverse } ((f * g) \$\$ (df + dg))) =$   
 $\text{fls-left-inverse } g (\text{inverse } (g \$\$ dg)) * \text{fls-left-inverse } f (\text{inverse } (f \$\$ df))$   
**and**  $\text{fls-right-inverse } (f * g) (\text{inverse } ((f * g) \$\$ (df + dg))) =$   
 $\text{fls-right-inverse } g (\text{inverse } (g \$\$ dg)) * \text{fls-right-inverse } f (\text{inverse } (f \$\$ df))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-lr-inverse-power-divring*:  
 $\text{fls-left-inverse } (f \wedge n) ((\text{inverse } (f \$\$ \text{fls-subdegree } f)) \wedge n) =$   
 $(\text{fls-left-inverse } f (\text{inverse } (f \$\$ \text{fls-subdegree } f))) \wedge n$  (**is** ?P)  
**and**  $\text{fls-right-inverse } (f \wedge n) ((\text{inverse } (f \$\$ \text{fls-subdegree } f)) \wedge n) =$   
 $(\text{fls-right-inverse } f (\text{inverse } (f \$\$ \text{fls-subdegree } f))) \wedge n$  (**is** ?Q)  
**for**  $f :: 'a :: \text{division-ring } \text{fls}$   
 $\langle \text{proof} \rangle$

**lemma** *one-plus-fls-X-powi-eq*:  
 $(1 + \text{fls-X}) \text{ powi } n = \text{fps-to-fls } (\text{fps-binomial } (\text{of-int } n :: 'a :: \text{field-char-0}))$   
 $\langle \text{proof} \rangle$

**instance** *fls* :: (*field*) *field*  
 $\langle \text{proof} \rangle$

**instance** *fls* :: ( $\{\text{field-prime-char}, \text{comm-semiring-1}\}$ ) *field-prime-char*  
 $\langle \text{proof} \rangle$

**instance** *fls* :: (*semiring-char-0*) *semiring-char-0*  
 $\langle \text{proof} \rangle$

**instance** *fls* :: (*field-char-0*) *field-char-0*  $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-power-int [simp]*:  
**fixes**  $F :: 'a :: \text{field } \text{fls}$   
**shows**  $\text{fls-subdegree } (F \text{ powi } n) = n * \text{fls-subdegree } F$

$\langle proof \rangle$

### 7.5.6 Division

**lemma** *fls-divide-nth-below*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{uminus}, \text{times}, \text{inverse}\}\ \text{fls}$

**shows**  $n < \text{fls-subdegree } f - \text{fls-subdegree } g \implies (f \text{ div } g) \text{ \textit{fls} } n = 0$

$\langle proof \rangle$

**lemma** *fls-divide-nth-base*:

**fixes**  $f\ g :: 'a::\text{division-ring}\ \text{fls}$

**shows**

$(f \text{ div } g) \text{ \textit{fls} } (\text{fls-subdegree } f - \text{fls-subdegree } g) =$

$f \text{ \textit{fls} } \text{fls-subdegree } f / g \text{ \textit{fls} } \text{fls-subdegree } g$

$\langle proof \rangle$

**lemma** *fls-div-zero [simp]*:

$0 \text{ div } (g :: 'a :: \{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}\ \text{fls}) = 0$

$\langle proof \rangle$

**lemma** *fls-div-by-zero*:

**fixes**  $g :: 'a::\{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}\}\ \text{fls}$

**assumes**  $\text{inverse } (0 :: 'a) = 0$

**shows**  $g \text{ div } 0 = 0$

$\langle proof \rangle$

**lemma** *fls-divide-times*:

**fixes**  $f\ g :: 'a::\{\text{semiring-0}, \text{inverse}, \text{uminus}\}\ \text{fls}$

**shows**  $(f * g) / h = f * (g / h)$

$\langle proof \rangle$

**lemma** *fls-divide-times2*:

**fixes**  $f\ g :: 'a::\{\text{comm-semiring-0}, \text{inverse}, \text{uminus}\}\ \text{fls}$

**shows**  $(f * g) / h = (f / h) * g$

$\langle proof \rangle$

**lemma** *fls-divide-subdegree-ge*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{uminus}, \text{times}, \text{inverse}\}\ \text{fls}$

**assumes**  $f / g \neq 0$

**shows**  $\text{fls-subdegree } (f / g) \geq \text{fls-subdegree } f - \text{fls-subdegree } g$

$\langle proof \rangle$

**lemma** *fls-divide-subdegree*:

**fixes**  $f\ g :: 'a::\text{division-ring}\ \text{fls}$

**assumes**  $f \neq 0\ g \neq 0$

**shows**  $\text{fls-subdegree } (f / g) = \text{fls-subdegree } f - \text{fls-subdegree } g$

$\langle proof \rangle$

**lemma** *fls-divide-shift-numer-nonzero*:

**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, inverse, times, uminus}\}$   $\text{fls}$   
**assumes**  $f \neq 0$   
**shows**  $\text{fls-shift } m\ f / g = \text{fls-shift } m\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-shift-numer*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, inverse, mult-zero, uminus}\}$   $\text{fls}$   
**shows**  $\text{fls-shift } m\ f / g = \text{fls-shift } m\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-shift-denom-nonzero*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, inverse, times, uminus}\}$   $\text{fls}$   
**assumes**  $g \neq 0$   
**shows**  $f / \text{fls-shift } m\ g = \text{fls-shift } (-m)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-shift-denom*:  
**fixes**  $f\ g :: 'a :: \text{division-ring fls}$   
**shows**  $f / \text{fls-shift } m\ g = \text{fls-shift } (-m)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-shift-both-nonzero*:  
**fixes**  $f\ g :: 'a :: \{\text{comm-monoid-add, inverse, times, uminus}\}$   $\text{fls}$   
**assumes**  $f \neq 0\ g \neq 0$   
**shows**  $\text{fls-shift } n\ f / \text{fls-shift } m\ g = \text{fls-shift } (n-m)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-shift-both [simp]*:  
**fixes**  $f\ g :: 'a :: \text{division-ring fls}$   
**shows**  $\text{fls-shift } n\ f / \text{fls-shift } m\ g = \text{fls-shift } (n-m)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-base-factor-numer*:  
 $\text{fls-base-factor } f / g = \text{fls-shift } (\text{fls-subdegree } f)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-base-factor-denom*:  
 $f / \text{fls-base-factor } g = \text{fls-shift } (-\text{fls-subdegree } g)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-base-factor'*:  
 $\text{fls-base-factor } f / \text{fls-base-factor } g = \text{fls-shift } (\text{fls-subdegree } f - \text{fls-subdegree } g)\ (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-base-factor*:  
**fixes**  $f\ g :: 'a :: \text{division-ring fls}$   
**shows**  $\text{fls-base-factor } f / \text{fls-base-factor } g = \text{fls-base-factor } (f/g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-regpart*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{comm-monoid-add}, \text{uminus}, \text{mult-zero}\}$  *fls*  
**assumes**  $\text{fls-subdegree } f \geq 0 \text{ fls-subdegree } g \geq 0$   
**shows**  $\text{fls-regpart } (f / g) = \text{fls-regpart } f / \text{fls-regpart } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-fls-base-factor-to-fps'*:

**fixes**  $f\ g :: 'a::\{\text{comm-monoid-add}, \text{uminus}, \text{inverse}, \text{mult-zero}\}$  *fls*  
**shows**  
 $\text{fls-base-factor-to-fps } f / \text{fls-base-factor-to-fps } g =$   
 $\text{fls-regpart } (\text{fls-shift } (\text{fls-subdegree } f - \text{fls-subdegree } g) (f / g))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-fls-base-factor-to-fps*:

**fixes**  $f\ g :: 'a::\text{division-ring}$  *fls*  
**shows**  $\text{fls-base-factor-to-fps } f / \text{fls-base-factor-to-fps } g = \text{fls-base-factor-to-fps } (f / g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-fps-to-fls*:

**fixes**  $f\ g :: 'a::\{\text{inverse}, \text{ab-group-add}, \text{mult-zero}\}$  *fps*  
**assumes**  $\text{subdegree } f \geq \text{subdegree } g$   
**shows**  $\text{fps-to-fls } f / \text{fps-to-fls } g = \text{fps-to-fls } (f / g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-1'*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}, \text{zero-neq-one}, \text{monoid-mult}\}$  *fls*  
**assumes**  $\text{inverse } (1::'a) = 1$   
**shows**  $f / 1 = f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-1 [simp]*:  $a / 1 = (a::'a::\text{division-ring } \text{fls})$

$\langle \text{proof} \rangle$

**lemma** *fls-const-divide-const*:

**fixes**  $x\ y :: 'a::\text{division-ring}$   
**shows**  $\text{fls-const } x / \text{fls-const } y = \text{fls-const } (x / y)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-X'*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{inverse}, \text{mult-zero}, \text{uminus}, \text{zero-neq-one}, \text{monoid-mult}\}$  *fls*  
**assumes**  $\text{inverse } (1::'a) = 1$   
**shows**  $f / \text{fls-X} = \text{fls-shift } 1\ f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-X [simp]*:

```

fixes f :: 'a::division-ring fls
shows f / fls-X = fls-shift 1 f
⟨proof⟩

lemma fls-divide-X-power':
  fixes f :: 'a::{semiring-1,inverse,uminus} fls
  assumes inverse (1::'a) = 1
  shows f / (fls-X ^ n) = fls-shift n f
⟨proof⟩

lemma fls-divide-X-power [simp]:
  fixes f :: 'a::division-ring fls
  shows f / (fls-X ^ n) = fls-shift n f
⟨proof⟩

lemma fls-divide-X-inv':
  fixes f :: 'a::{comm-monoid-add,inverse,mult-zero,uminus,zero-neg-one,monoid-mult}
  fls
  assumes inverse (1::'a) = 1
  shows f / fls-X-inv = fls-shift (-1) f
⟨proof⟩

lemma fls-divide-X-inv [simp]:
  fixes f :: 'a::division-ring fls
  shows f / fls-X-inv = fls-shift (-1) f
⟨proof⟩

lemma fls-divide-X-inv-power':
  fixes f :: 'a::{semiring-1,inverse,uminus} fls
  assumes inverse (1::'a) = 1
  shows f / (fls-X-inv ^ n) = fls-shift (-int n) f
⟨proof⟩

lemma fls-divide-X-inv-power [simp]:
  fixes f :: 'a::division-ring fls
  shows f / (fls-X-inv ^ n) = fls-shift (-int n) f
⟨proof⟩

lemma fls-divide-X-intpow':
  fixes f :: 'a::{semiring-1,inverse,uminus} fls
  assumes inverse (1::'a) = 1
  shows f / (fls-X-intpow i) = fls-shift i f
⟨proof⟩

lemma fls-divide-X-intpow-conv-times':
  fixes f :: 'a::{semiring-1,inverse,uminus} fls
  assumes inverse (1::'a) = 1
  shows f / (fls-X-intpow i) = f * fls-X-intpow (-i)
⟨proof⟩

```



**lemma** *fls-divide-X-intpow*:  
**fixes**  $f :: 'a::\text{division-ring } fls$   
**shows**  $f / (\text{fls-X-intpow } i) = \text{fls-shift } i \ f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-X-intpow-conv-times*:  
**fixes**  $f :: 'a::\text{division-ring } fls$   
**shows**  $f / (\text{fls-X-intpow } i) = f * \text{fls-X-intpow } (-i)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-intpow-div-fls-X-intpow-semiring1*:  
**assumes**  $\text{inverse } (1 :: 'a::\{\text{semiring-1}, \text{inverse}, \text{uminus}\}) = 1$   
**shows**  $(\text{fls-X-intpow } i :: 'a \text{ fls}) / \text{fls-X-intpow } j = \text{fls-X-intpow } (i-j)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-X-intpow-div-fls-X-intpow*:  
 $(\text{fls-X-intpow } i :: 'a::\text{division-ring } fls) / \text{fls-X-intpow } j = \text{fls-X-intpow } (i-j)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-add*:  
**fixes**  $f \ g \ h :: 'a::\{\text{semiring-0}, \text{inverse}, \text{uminus}\} \text{ fls}$   
**shows**  $(f + g) / h = f / h + g / h$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-diff*:  
**fixes**  $f \ g \ h :: 'a::\{\text{ring}, \text{inverse}\} \text{ fls}$   
**shows**  $(f - g) / h = f / h - g / h$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-uminus*:  
**fixes**  $f \ g \ h :: 'a::\{\text{ring}, \text{inverse}\} \text{ fls}$   
**shows**  $(-f) / g = -(f / g)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-divide-uminus'*:  
**fixes**  $f \ g \ h :: 'a::\text{division-ring } fls$   
**shows**  $f / (-g) = -(f / g)$   
 $\langle \text{proof} \rangle$

### 7.5.7 Units

**lemma** *fls-is-left-unit-iff-base-is-left-unit*:  
**fixes**  $f :: 'a :: \text{ring-1-no-zero-divisors } fls$   
**shows**  $(\exists g. 1 = f * g) \longleftrightarrow (\exists k. 1 = f \ \$\$ \text{fls-subdegree } f * k)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-is-right-unit-iff-base-is-right-unit*:  
**fixes**  $f :: 'a :: \text{ring-1-no-zero-divisors } fls$

**shows**  $(\exists g. 1 = g * f) \longleftrightarrow (\exists k. 1 = k * f \text{ \textit{fls-subdegree f}})$   
 $\langle \textit{proof} \rangle$

## 7.6 Composition

**definition**  $\textit{fls-compose-fps} :: 'a :: \textit{field fls} \Rightarrow 'a \textit{ fps} \Rightarrow 'a \textit{ fls}$  **where**  
 $\textit{fls-compose-fps } F \ G =$   
 $\textit{fps-to-fls } (\textit{fps-compose } (\textit{fls-base-factor-to-fps } F) \ G) * \textit{fps-to-fls } G \textit{ powi fls-subdegree }$   
 $F$

**lemma**  $\textit{fps-compose-of-nat} \text{ [simp]: } \textit{fps-compose } (\textit{of-nat } n :: 'a :: \textit{comm-ring-1 fps})$   
 $H = \textit{of-nat } n$   
**and**  $\textit{fps-compose-of-int} \text{ [simp]: } \textit{fps-compose } (\textit{of-int } i) \ H = \textit{of-int } i$   
 $\langle \textit{proof} \rangle$

**lemmas**  $\text{ [simp] } = \textit{fps-to-fls-of-nat } \textit{fps-to-fls-of-int}$

**lemma**  $\textit{fls-compose-fps-0} \text{ [simp]: } \textit{fls-compose-fps } 0 \ H = 0$   
**and**  $\textit{fls-compose-fps-1} \text{ [simp]: } \textit{fls-compose-fps } 1 \ H = 1$   
**and**  $\textit{fls-compose-fps-const} \text{ [simp]: } \textit{fls-compose-fps } (\textit{fls-const } c) \ H = \textit{fls-const } c$   
**and**  $\textit{fls-compose-fps-of-nat} \text{ [simp]: } \textit{fls-compose-fps } (\textit{of-nat } n) \ H = \textit{of-nat } n$   
**and**  $\textit{fls-compose-fps-of-int} \text{ [simp]: } \textit{fls-compose-fps } (\textit{of-int } i) \ H = \textit{of-int } i$   
**and**  $\textit{fls-compose-fps-X} \text{ [simp]: } \textit{fls-compose-fps fls-X } F = \textit{fps-to-fls } F$   
 $\langle \textit{proof} \rangle$

**lemma**  $\textit{fls-compose-fps-0-right}$ :  
 $\textit{fls-compose-fps } F \ 0 = (\textit{if } 0 \leq \textit{fls-subdegree } F \text{ then } \textit{fls-const } (F \text{ \textit{fls-subdegree } 0}) \text{ else } 0)$   
 $\langle \textit{proof} \rangle$

**lemma**  $\textit{fls-compose-fps-shift}$ :  
**assumes**  $H \neq 0$   
**shows**  $\textit{fls-compose-fps } (\textit{fls-shift } n \ F) \ H = \textit{fls-compose-fps } F \ H * \textit{fps-to-fls } H$   
 $\textit{powi } (-n)$   
 $\langle \textit{proof} \rangle$

**lemma**  $\textit{fls-compose-fps-to-fls} \text{ [simp]:}$   
**assumes**  $\text{ [simp]: } G \neq 0 \textit{ fps-nth } G \ 0 = 0$   
**shows**  $\textit{fls-compose-fps } (\textit{fps-to-fls } F) \ G = \textit{fps-to-fls } (\textit{fps-compose } F \ G)$   
 $\langle \textit{proof} \rangle$

**lemma**  $\textit{fls-compose-fps-mult}$ :  
**assumes**  $\text{ [simp]: } H \neq 0 \textit{ fps-nth } H \ 0 = 0$   
**shows**  $\textit{fls-compose-fps } (F * G) \ H = \textit{fls-compose-fps } F \ H * \textit{fls-compose-fps } G$   
 $H$   
 $\langle \textit{proof} \rangle$

**lemma**  $\textit{fls-compose-fps-power}$ :  
**assumes**  $\text{ [simp]: } G \neq 0 \textit{ fps-nth } G \ 0 = 0$   
**shows**  $\textit{fls-compose-fps } (F \wedge n) \ G = \textit{fls-compose-fps } F \ G \wedge n$

$\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-add*:

**assumes**  $[simp]: H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (F + G) \ H = \text{fls-compose-fps } F \ H + \text{fls-compose-fps } G \ H$   
 $\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-uminus*  $[simp]: \text{fls-compose-fps } (-F) \ H = -\text{fls-compose-fps } F \ H$

$\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-diff*:

**assumes**  $[simp]: H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (F - G) \ H = \text{fls-compose-fps } F \ H - \text{fls-compose-fps } G \ H$   
 $\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-eq-0-iff*:

**assumes**  $H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } F \ H = 0 \longleftrightarrow F = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-inverse*:

**assumes**  $[simp]: H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (\text{inverse } F) \ H = \text{inverse } (\text{fls-compose-fps } F \ H)$

$\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-divide*:

**assumes**  $[simp]: H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (F / G) \ H = \text{fls-compose-fps } F \ H / \text{fls-compose-fps } G \ H$   
 $\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-powi*:

**assumes**  $[simp]: H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (F \text{ powi } n) \ H = \text{fls-compose-fps } F \ H \text{ powi } n$

$\langle \text{proof} \rangle$

**lemma** *fls-compose-fps-assoc*:

**assumes**  $[simp]: G \neq 0 \text{ fps-nth } G \ 0 = 0 \ H \neq 0 \text{ fps-nth } H \ 0 = 0$

**shows**  $\text{fls-compose-fps } (\text{fls-compose-fps } F \ G) \ H = \text{fls-compose-fps } F \ (\text{fls-compose-fps } G \ H)$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-pos-iff*:  $\text{subdegree } F > 0 \longleftrightarrow F \neq 0 \wedge \text{fps-nth } F \ 0 = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-X-power-int*  $[simp]: \text{fls-X powi } n = (\text{fls-X-intpow } n :: 'a :: \text{division-ring})$

*fls*)  
 ⟨proof⟩

**lemma** *fls-const-power-int*: *fls-const* (*c powi n*) = *fls-const* (*c :: 'a :: division-ring*)  
*powi n*  
 ⟨proof⟩

**lemma** *fls-nth-fls-compose-fps-linear*:  
**fixes** *c :: 'a :: field*  
**assumes** [*simp*]: *c ≠ 0*  
**shows** *fls-compose-fps F (fps-const c \* fps-X) \$\$ n = F \$\$ n \* c powi n*  
 ⟨proof⟩

**lemma** *fls-const-transfer* [*transfer-rule*]:  
*rel-fun* (=) (*pcr-fls* (=))  
 (λ*c n. if n = 0 then c else 0*) *fls-const*  
 ⟨proof⟩

**lemma** *fls-shift-transfer* [*transfer-rule*]:  
*rel-fun* (=) (*rel-fun* (*pcr-fls* (=)) (*pcr-fls* (=)))  
 (λ*n f k. f (k+n)*) *fls-shift*  
 ⟨proof⟩

**lift-definition** *fls-compose-power* :: '*a* :: zero *fls* ⇒ *nat* ⇒ '*a fls* **is**  
 λ*f d n. if d > 0 ∧ int d dvd n then f (n div int d) else 0*  
 ⟨proof⟩

**lemma** *fls-nth-compose-power*:  
**assumes** *d > 0*  
**shows** *fls-compose-power f d \$\$ n = (if int d dvd n then f \$\$ (n div int d) else 0)*  
 ⟨proof⟩

**lemma** *fls-compose-power-0-left* [*simp*]: *fls-compose-power 0 d = 0*  
 ⟨proof⟩

**lemma** *fls-compose-power-1-left* [*simp*]: *d > 0 ⇒ fls-compose-power 1 d = 1*  
 ⟨proof⟩

**lemma** *fls-compose-power-const-left* [*simp*]:  
*d > 0 ⇒ fls-compose-power (fls-const c) d = fls-const c*  
 ⟨proof⟩

**lemma** *fls-compose-power-shift* [*simp*]:  
*d > 0 ⇒ fls-compose-power (fls-shift n f) d = fls-shift (d \* n) (fls-compose-power f d)*  
 ⟨proof⟩

**lemma** *fls-compose-power-X-intpow* [simp]:

$d > 0 \implies \text{fls-compose-power } (\text{fls-X-intpow } n) \ d = \text{fls-X-intpow } (\text{int } d * n)$

*<proof>*

**lemma** *fls-compose-power-X* [simp]:

$d > 0 \implies \text{fls-compose-power } \text{fls-X} \ d = \text{fls-X-intpow } (\text{int } d)$

*<proof>*

**lemma** *fls-compose-power-X-inv* [simp]:

$d > 0 \implies \text{fls-compose-power } \text{fls-X-inv} \ d = \text{fls-X-intpow } (-\text{int } d)$

*<proof>*

**lemma** *fls-compose-power-0-right* [simp]:  $\text{fls-compose-power } f \ 0 = 0$

*<proof>*

**lemma** *fls-compose-power-add* [simp]:

$\text{fls-compose-power } (f + g) \ d = \text{fls-compose-power } f \ d + \text{fls-compose-power } g \ d$

*<proof>*

**lemma** *fls-compose-power-diff* [simp]:

$\text{fls-compose-power } (f - g) \ d = \text{fls-compose-power } f \ d - \text{fls-compose-power } g \ d$

*<proof>*

**lemma** *fls-compose-power-uminus* [simp]:

$\text{fls-compose-power } (-f) \ d = -\text{fls-compose-power } f \ d$

*<proof>*

**lemma** *fps-nth-compose-X-power*:

$\text{fps-nth } (f \text{ oo } (\text{fps-X}^\wedge d)) \ n = (\text{if } d \text{ dvd } n \text{ then } \text{fps-nth } f \ (n \text{ div } d) \text{ else } 0)$

*<proof>*

**lemma** *fls-compose-power-fps-to-fls*:

**assumes**  $d > 0$

**shows**  $\text{fls-compose-power } (\text{fps-to-fls } f) \ d = \text{fps-to-fls } (\text{fps-compose } f \ (\text{fps-X}^\wedge d))$

*<proof>*

**lemma** *fls-compose-power-mult* [simp]:

$\text{fls-compose-power } (f * g :: 'a :: \text{idom } \text{fls}) \ d = \text{fls-compose-power } f \ d * \text{fls-compose-power } g \ d$

*<proof>*

**lemma** *fls-compose-power-power* [simp]:

**assumes**  $d > 0 \vee n > 0$

**shows**  $\text{fls-compose-power } (f^\wedge n :: 'a :: \text{idom } \text{fls}) \ d = \text{fls-compose-power } f \ d^\wedge n$

*<proof>*

**lemma** *fls-nth-compose-power'* [simp]:

$d = 0 \vee \neg d \text{ dvd } n \implies \text{fls-compose-power } f \ d \ \$\$ \text{int } n = 0$

$d \text{ dvd } n \implies d > 0 \implies \text{fls-compose-power } f \text{ } d \text{ } \$\$ \text{ int } n = f \text{ } \$\$ \text{ int } (n \text{ div } d)$   
 $\langle \text{proof} \rangle$

**lemma** *subdegree-fls-compose-fps* [simp]:  
**fixes**  $G :: 'a :: \text{field } \text{fps}$   
**assumes** [simp]:  $\text{fps-nth } G \text{ } 0 = 0$   
**shows**  $\text{fls-subdegree } (\text{fls-compose-fps } F \text{ } G) = \text{fls-subdegree } F * \text{subdegree } G$   
 $\langle \text{proof} \rangle$

## 7.7 Formal differentiation and integration

### 7.7.1 Derivative

**definition**  $\text{fls-deriv } f = \text{Abs-fls } (\lambda n. \text{of-int } (n+1) * f \$\$ (n+1))$

**lemma** *fls-deriv-nth*[simp]:  $\text{fls-deriv } f \text{ } \$\$ n = \text{of-int } (n+1) * f \$\$ (n+1)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-residue*:  $\text{fls-deriv } f \text{ } \$\$ -1 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-const*[simp]:  $\text{fls-deriv } (\text{fls-const } x) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-of-nat*[simp]:  $\text{fls-deriv } (\text{of-nat } n) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-of-int*[simp]:  $\text{fls-deriv } (\text{of-int } i) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-zero*[simp]:  $\text{fls-deriv } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-one*[simp]:  $\text{fls-deriv } 1 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-numeral* [simp]:  $\text{fls-deriv } (\text{numeral } n) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-subdegree'*:  
**assumes**  $\text{of-int } (\text{fls-subdegree } f) * f \text{ } \$\$ \text{fls-subdegree } f \neq 0$   
**shows**  $\text{fls-subdegree } (\text{fls-deriv } f) = \text{fls-subdegree } f - 1$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-subdegree0*:  
**assumes**  $\text{fls-subdegree } f = 0$   
**shows**  $\text{fls-subdegree } (\text{fls-deriv } f) \geq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-subdegree-deriv'*:

**fixes**  $f :: 'a::\text{ring-1-no-zero-divisors}$   $\text{fls}$   
**assumes**  $(\text{of-int } (\text{fls-subdegree } f) :: 'a) \neq 0$   
**shows**  $\text{fls-subdegree } (\text{fls-deriv } f) = \text{fls-subdegree } f - 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-subdegree-deriv}$ :  
**fixes**  $f :: 'a::\{\text{ring-1-no-zero-divisors}, \text{ring-char-0}\}$   $\text{fls}$   
**assumes**  $\text{fls-subdegree } f \neq 0$   
**shows**  $\text{fls-subdegree } (\text{fls-deriv } f) = \text{fls-subdegree } f - 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fps-deriv-fls-regpart}$ :  $\text{fps-deriv } (\text{fls-regpart } F) = \text{fls-regpart } (\text{fls-deriv } F)$   
 $\langle \text{proof} \rangle$

Shifting is like multiplying by a power of the implied variable, and so satisfies a product-like rule.

**lemma**  $\text{fls-deriv-shift}$ :  
 $\text{fls-deriv } (\text{fls-shift } n \ f) = \text{of-int } (-n) * \text{fls-shift } (n+1) \ f + \text{fls-shift } n \ (\text{fls-deriv } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-deriv-X}$   $[\text{simp}]$ :  $\text{fls-deriv } \text{fls-X} = 1$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-deriv-X-inv}$   $[\text{simp}]$ :  $\text{fls-deriv } \text{fls-X-inv} = - (\text{fls-X-inv}^2)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-deriv-delta}$ :  
 $\text{fls-deriv } (\text{Abs-fls } (\lambda n. \text{if } n=m \text{ then } c \text{ else } 0)) =$   
 $\text{Abs-fls } (\lambda n. \text{if } n=m-1 \text{ then } \text{of-int } m * c \text{ else } 0)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-deriv-base-factor}$ :  
 $\text{fls-deriv } (\text{fls-base-factor } f) =$   
 $\text{of-int } (-\text{fls-subdegree } f) * \text{fls-shift } (\text{fls-subdegree } f + 1) \ f +$   
 $\text{fls-shift } (\text{fls-subdegree } f) \ (\text{fls-deriv } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-regpart-deriv}$ :  $\text{fls-regpart } (\text{fls-deriv } f) = \text{fps-deriv } (\text{fls-regpart } f)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{fls-prpart-deriv}$ :  
**fixes**  $f :: 'a :: \{\text{comm-ring-1}, \text{ring-no-zero-divisors}\}$   $\text{fls}$   
— Commutativity and no zero divisors are required by the definition of  $\text{pderiv}$ .  
**shows**  $\text{fls-prpart } (\text{fls-deriv } f) = - \text{pCons } 0 \ (\text{pCons } 0 \ (\text{pderiv } (\text{fls-prpart } f)))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pderiv-fls-prpart}$ :  
 $\text{pderiv } (\text{fls-prpart } f) = - \text{poly-shift } 2 \ (\text{fls-prpart } (\text{fls-deriv } f))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-fps-to-fls*:  $\text{fls-deriv } (\text{fps-to-fls } f) = \text{fps-to-fls } (\text{fps-deriv } f)$   
 $\langle \text{proof} \rangle$

### 7.7.2 Algebraic rules of the derivative

**lemma** *fls-deriv-add* [*simp*]:  $\text{fls-deriv } (f+g) = \text{fls-deriv } f + \text{fls-deriv } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-sub* [*simp*]:  $\text{fls-deriv } (f-g) = \text{fls-deriv } f - \text{fls-deriv } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-neg* [*simp*]:  $\text{fls-deriv } (-f) = - \text{fls-deriv } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-mult* [*simp*]:  
 $\text{fls-deriv } (f*g) = f * \text{fls-deriv } g + \text{fls-deriv } f * g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-mult-const-left*:  
 $\text{fls-deriv } (\text{fls-const } c * f) = \text{fls-const } c * \text{fls-deriv } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-linear*:  
 $\text{fls-deriv } (\text{fls-const } a * f + \text{fls-const } b * g) =$   
 $\text{fls-const } a * \text{fls-deriv } f + \text{fls-const } b * \text{fls-deriv } g$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-mult-const-right*:  
 $\text{fls-deriv } (f * \text{fls-const } c) = \text{fls-deriv } f * \text{fls-const } c$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-linear2*:  
 $\text{fls-deriv } (f * \text{fls-const } a + g * \text{fls-const } b) =$   
 $\text{fls-deriv } f * \text{fls-const } a + \text{fls-deriv } g * \text{fls-const } b$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-sum*:  
 $\text{fls-deriv } (\text{sum } f \ S) = \text{sum } (\lambda i. \text{fls-deriv } (f \ i)) \ S$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-power*:  
**fixes**  $f :: 'a::\text{comm-ring-1} \ \text{fls}$   
**shows**  $\text{fls-deriv } (f^{\wedge} n) = \text{of-nat } n * f^{\wedge}(n-1) * \text{fls-deriv } f$   
 $\langle \text{proof} \rangle$

**lemma** *fls-deriv-X-power*:  
 $\text{fls-deriv } (\text{fls-X } ^{\wedge} n) = \text{of-nat } n * \text{fls-X } ^{\wedge}(n-1)$   
 $\langle \text{proof} \rangle$



**lemma** *fls-deriv-X-inv-power*:

$fls-deriv (fls-X-inv \wedge n) = -\ of-nat\ n * fls-X-inv \wedge (Suc\ n)$   
 $\langle proof \rangle$

**lemma** *fls-deriv-X-intpow*:

$fls-deriv (fls-X-intpow\ i) = of-int\ i * fls-X-intpow\ (i-1)$   
 $\langle proof \rangle$

**lemma** *fls-deriv-lr-inverse*:

**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1\ f \ \$\$ fls-subdegree\ f * y = 1$   
 — These assumptions imply  $x$  equals  $y$ , but no need to assume that.  
**shows**  $fls-deriv (fls-left-inverse\ f\ x) =$   
 $- fls-left-inverse\ f\ x * fls-deriv\ f * fls-left-inverse\ f\ x$   
**and**  $fls-deriv (fls-right-inverse\ f\ y) =$   
 $- fls-right-inverse\ f\ y * fls-deriv\ f * fls-right-inverse\ f\ y$   
 $\langle proof \rangle$

**lemma** *fls-deriv-lr-inverse-comm*:

**fixes**  $x\ y :: 'a::comm-ring-1$   
**assumes**  $x * f \ \$\$ fls-subdegree\ f = 1$   
**shows**  $fls-deriv (fls-left-inverse\ f\ x) = - fls-deriv\ f * (fls-left-inverse\ f\ x)^2$   
**and**  $fls-deriv (fls-right-inverse\ f\ x) = - fls-deriv\ f * (fls-right-inverse\ f\ x)^2$   
 $\langle proof \rangle$

**lemma** *fls-inverse-deriv-divring*:

**fixes**  $a :: 'a::division-ring\ fls$   
**shows**  $fls-deriv (inverse\ a) = - inverse\ a * fls-deriv\ a * inverse\ a$   
 $\langle proof \rangle$

**lemma** *fls-inverse-deriv*:

**fixes**  $a :: 'a::field\ fls$   
**shows**  $fls-deriv (inverse\ a) = - fls-deriv\ a * (inverse\ a)^2$   
 $\langle proof \rangle$

**lemma** *fls-inverse-deriv'*:

**fixes**  $a :: 'a::field\ fls$   
**shows**  $fls-deriv (inverse\ a) = - fls-deriv\ a / a^2$   
 $\langle proof \rangle$

### 7.7.3 Equality of derivatives

**lemma** *fls-deriv-eq-0-iff*:

$fls-deriv\ f = 0 \iff f = fls-const\ (f \ \$\$ 0 :: 'a::\{ring-1-no-zero-divisors,ring-char-0\})$   
 $\langle proof \rangle$

**lemma** *fls-deriv-eq-iff*:

**fixes**  $f\ g :: 'a::\{ring-1-no-zero-divisors,ring-char-0\}\ fls$   
**shows**  $fls-deriv\ f = fls-deriv\ g \iff (f = fls-const(f \ \$\$ 0 - g \ \$\$ 0) + g)$

$\langle proof \rangle$

**lemma** *fls-deriv-eq-iff-ex*:

**fixes**  $f\ g :: 'a::\{\text{ring-1-no-zero-divisors}, \text{ring-char-0}\}$  *fls*

**shows**  $(\text{fls-deriv } f = \text{fls-deriv } g) \longleftrightarrow (\exists c. f = \text{fls-const } c + g)$

$\langle proof \rangle$

#### 7.7.4 Residues

**definition** *fls-residue-def[simp]*:  $\text{fls-residue } f \equiv f \text{ \texttt{\$} \$} - 1$

**lemma** *fls-residue-deriv*:  $\text{fls-residue } (\text{fls-deriv } f) = 0$

$\langle proof \rangle$

**lemma** *fls-residue-add*:  $\text{fls-residue } (f+g) = \text{fls-residue } f + \text{fls-residue } g$

$\langle proof \rangle$

**lemma** *fls-residue-times-deriv*:

$\text{fls-residue } (\text{fls-deriv } f * g) = - \text{fls-residue } (f * \text{fls-deriv } g)$

$\langle proof \rangle$

**lemma** *fls-residue-power-series*:  $\text{fls-subdegree } f \geq 0 \implies \text{fls-residue } f = 0$

$\langle proof \rangle$

**lemma** *fls-residue-fls-X-intpow*:

$\text{fls-residue } (\text{fls-X-intpow } i) = (\text{if } i = -1 \text{ then } 1 \text{ else } 0)$

$\langle proof \rangle$

**lemma** *fls-residue-shift-nth*:

**fixes**  $f :: 'a::\text{semiring-1}$  *fls*

**shows**  $f \text{ \texttt{\$} \$} n = \text{fls-residue } (\text{fls-X-intpow } (-n-1) * f)$

$\langle proof \rangle$

**lemma** *fls-residue-fls-const-times*:

**fixes**  $f :: 'a::\{\text{comm-monoid-add}, \text{mult-zero}\}$  *fls*

**shows**  $\text{fls-residue } (\text{fls-const } c * f) = c * \text{fls-residue } f$

**and**  $\text{fls-residue } (f * \text{fls-const } c) = \text{fls-residue } f * c$

$\langle proof \rangle$

**lemma** *fls-residue-of-int-times*:

**fixes**  $f :: 'a::\text{ring-1}$  *fls*

**shows**  $\text{fls-residue } (\text{of-int } i * f) = \text{of-int } i * \text{fls-residue } f$

**and**  $\text{fls-residue } (f * \text{of-int } i) = \text{fls-residue } f * \text{of-int } i$

$\langle proof \rangle$

**lemma** *fls-residue-deriv-times-lr-inverse-eq-subdegree*:

**fixes**  $f\ g :: 'a::\text{ring-1}$  *fls*

**assumes**  $y * (f \text{ \texttt{\$} \$} \text{fls-subdegree } f) = 1$   $(f \text{ \texttt{\$} \$} \text{fls-subdegree } f) * y = 1$

**shows**  $\text{fls-residue } (\text{fls-deriv } f * \text{fls-right-inverse } f\ y) = \text{of-int } (\text{fls-subdegree } f)$

**and**  $\text{fls-residue } (\text{fls-deriv } f * \text{fls-left-inverse } f \ y) = \text{of-int } (\text{fls-subdegree } f)$   
**and**  $\text{fls-residue } (\text{fls-left-inverse } f \ y * \text{fls-deriv } f) = \text{of-int } (\text{fls-subdegree } f)$   
**and**  $\text{fls-residue } (\text{fls-right-inverse } f \ y * \text{fls-deriv } f) = \text{of-int } (\text{fls-subdegree } f)$   
 $\langle \text{proof} \rangle$

**lemma** *fls-residue-deriv-times-inverse-eq-subdegree*:  
**fixes**  $f \ g :: 'a::\text{division-ring } \text{fls}$   
**shows**  $\text{fls-residue } (\text{fls-deriv } f * \text{inverse } f) = \text{of-int } (\text{fls-subdegree } f)$   
**and**  $\text{fls-residue } (\text{inverse } f * \text{fls-deriv } f) = \text{of-int } (\text{fls-subdegree } f)$   
 $\langle \text{proof} \rangle$

### 7.7.5 Integral definition and basic properties

**definition** *fls-integral* ::  $'a::\{\text{ring-1}, \text{inverse}\} \text{fls} \Rightarrow 'a \text{fls}$   
**where**  $\text{fls-integral } a = \text{Abs-fls } (\lambda n. \text{if } n=0 \text{ then } 0 \text{ else } \text{inverse } (\text{of-int } n) * a \$\$ (n - 1))$

**lemma** *fls-integral-nth* [*simp*]:  
 $\text{fls-integral } a \$\$ n = (\text{if } n=0 \text{ then } 0 \text{ else } \text{inverse } (\text{of-int } n) * a \$\$ (n-1))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-conv-fps-zeroth-integral*:  
**assumes**  $\text{fls-subdegree } a \geq 0$   
**shows**  $\text{fls-integral } a = \text{fps-to-fls } (\text{fps-integral0 } (\text{fls-regpart } a))$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-zero* [*simp*]:  $\text{fls-integral } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-const'*:  
**fixes**  $x :: 'a::\{\text{ring-1}, \text{inverse}\}$   
**assumes**  $\text{inverse } (1::'a) = 1$   
**shows**  $\text{fls-integral } (\text{fls-const } x) = \text{fls-const } x * \text{fls-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-const*:  
**fixes**  $x :: 'a::\text{division-ring}$   
**shows**  $\text{fls-integral } (\text{fls-const } x) = \text{fls-const } x * \text{fls-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-of-nat'*:  
**assumes**  $\text{inverse } (1::'a::\{\text{ring-1}, \text{inverse}\}) = 1$   
**shows**  $\text{fls-integral } (\text{of-nat } n :: 'a \text{fls}) = \text{of-nat } n * \text{fls-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-of-nat*:  
 $\text{fls-integral } (\text{of-nat } n :: 'a::\text{division-ring } \text{fls}) = \text{of-nat } n * \text{fls-X}$   
 $\langle \text{proof} \rangle$

**lemma** *fls-integral-of-int'*:

**assumes**  $\text{inverse } (1 :: 'a :: \{\text{ring-1}, \text{inverse}\}) = 1$

**shows**  $\text{fls-integral } (\text{of-int } i :: 'a \text{ fls}) = \text{of-int } i * \text{fls-}X$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-of-int*:

$\text{fls-integral } (\text{of-int } i :: 'a :: \text{division-ring fls}) = \text{of-int } i * \text{fls-}X$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-one'*:

**assumes**  $\text{inverse } (1 :: 'a :: \{\text{ring-1}, \text{inverse}\}) = 1$

**shows**  $\text{fls-integral } (1 :: 'a \text{ fls}) = \text{fls-}X$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-one*:  $\text{fls-integral } (1 :: 'a :: \text{division-ring fls}) = \text{fls-}X$

$\langle \text{proof} \rangle$

**lemma** *fls-subdegree-integral-ge*:

$\text{fls-integral } f \neq 0 \implies \text{fls-subdegree } (\text{fls-integral } f) \geq \text{fls-subdegree } f + 1$

$\langle \text{proof} \rangle$

**lemma** *fls-subdegree-integral*:

**fixes**  $f :: 'a :: \{\text{division-ring}, \text{ring-char-0}\} \text{ fls}$

**assumes**  $f \neq 0 \text{ fls-subdegree } f \neq -1$

**shows**  $\text{fls-subdegree } (\text{fls-integral } f) = \text{fls-subdegree } f + 1$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-X [simp]*:

$\text{fls-integral } (\text{fls-}X :: 'a :: \{\text{ring-1}, \text{inverse}\} \text{ fls}) =$

$\text{fls-const } (\text{inverse } (\text{of-int } 2)) * \text{fls-}X^2$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-X-power*:

$\text{fls-integral } (\text{fls-}X \wedge n :: 'a :: \{\text{ring-1}, \text{inverse}\} \text{ fls}) =$

$\text{fls-const } (\text{inverse } (\text{of-nat } (\text{Suc } n))) * \text{fls-}X \wedge \text{Suc } n$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-X-power-char0*:

$\text{fls-integral } (\text{fls-}X \wedge n :: 'a :: \{\text{ring-char-0}, \text{inverse}\} \text{ fls}) =$

$\text{inverse } (\text{of-nat } (\text{Suc } n)) * \text{fls-}X \wedge \text{Suc } n$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-X-inv [simp]*:  $\text{fls-integral } (\text{fls-}X\text{-inv} :: 'a :: \{\text{ring-1}, \text{inverse}\} \text{ fls}) = 0$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-X-inv-power*:

**assumes**  $n \geq 2$

**shows**

$$\text{fls-integral } (\text{fls-X-inv } ^n :: 'a :: \{\text{ring-1}, \text{inverse}\} \text{ fls}) =$$

$$\text{fls-const } (\text{inverse } (\text{of-int } (1 - \text{int } n))) * \text{fls-X-inv } ^{(n-1)}$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-X-inv-power-char0*:

**assumes**  $n \geq 2$

**shows**

$$\text{fls-integral } (\text{fls-X-inv } ^n :: 'a :: \{\text{ring-char-0}, \text{inverse}\} \text{ fls}) =$$

$$\text{inverse } (\text{of-int } (1 - \text{int } n)) * \text{fls-X-inv } ^{(n-1)}$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-X-inv-power'*:

**assumes**  $n \geq 1$

**shows**

$$\text{fls-integral } (\text{fls-X-inv } ^n :: 'a :: \text{division-ring fls}) =$$

$$- \text{fls-const } (\text{inverse } (\text{of-nat } (n-1))) * \text{fls-X-inv } ^{(n-1)}$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-X-inv-power-char0'*:

**assumes**  $n \geq 1$

**shows**

$$\text{fls-integral } (\text{fls-X-inv } ^n :: 'a :: \{\text{division-ring}, \text{ring-char-0}\} \text{ fls}) =$$

$$- \text{inverse } (\text{of-nat } (n-1)) * \text{fls-X-inv } ^{(n-1)}$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-delta*:

**assumes**  $m \neq -1$

**shows**

$$\text{fls-integral } (\text{Abs-fls } (\lambda n. \text{if } n=m \text{ then } c \text{ else } 0)) =$$

$$\text{Abs-fls } (\lambda n. \text{if } n=m+1 \text{ then } \text{inverse } (\text{of-int } (m+1)) * c \text{ else } 0)$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-regpart-integral*:

$$\text{fls-regpart } (\text{fls-integral } f) = \text{fps-integral0 } (\text{fls-regpart } f)$$

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-fps-to-fls*:

$$\text{fls-integral } (\text{fps-to-fls } f) = \text{fps-to-fls } (\text{fps-integral0 } f)$$

$$\langle \text{proof} \rangle$$

### 7.7.6 Algebraic rules of the integral

**lemma** *fls-integral-add [simp]*:  $\text{fls-integral } (f+g) = \text{fls-integral } f + \text{fls-integral } g$   

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-sub [simp]*:  $\text{fls-integral } (f-g) = \text{fls-integral } f - \text{fls-integral } g$   

$$\langle \text{proof} \rangle$$

**lemma** *fls-integral-neg [simp]*:  $\text{fls-integral } (-f) = - \text{fls-integral } f$

$\langle \text{proof} \rangle$

**lemma** *fls-integral-mult-const-left*:

*fls-integral* (*fls-const* *c* \* *f*) = *fls-const* *c* \* *fls-integral* (*f* :: 'a::division-ring *fls*)

$\langle \text{proof} \rangle$

**lemma** *fls-integral-mult-const-left-comm*:

**fixes** *f* :: 'a::{comm-ring-1,inverse} *fls*

**shows** *fls-integral* (*fls-const* *c* \* *f*) = *fls-const* *c* \* *fls-integral* *f*

$\langle \text{proof} \rangle$

**lemma** *fls-integral-linear*:

**fixes** *f g* :: 'a::division-ring *fls*

**shows**

*fls-integral* (*fls-const* *a* \* *f* + *fls-const* *b* \* *g*) =

*fls-const* *a* \* *fls-integral* *f* + *fls-const* *b* \* *fls-integral* *g*

$\langle \text{proof} \rangle$

**lemma** *fls-integral-linear-comm*:

**fixes** *f g* :: 'a::{comm-ring-1,inverse} *fls*

**shows**

*fls-integral* (*fls-const* *a* \* *f* + *fls-const* *b* \* *g*) =

*fls-const* *a* \* *fls-integral* *f* + *fls-const* *b* \* *fls-integral* *g*

$\langle \text{proof} \rangle$

**lemma** *fls-integral-mult-const-right*:

*fls-integral* (*f* \* *fls-const* *c*) = *fls-integral* *f* \* *fls-const* *c*

$\langle \text{proof} \rangle$

**lemma** *fls-integral-linear2*:

*fls-integral* (*f* \* *fls-const* *a* + *g* \* *fls-const* *b*) =

*fls-integral* *f* \* *fls-const* *a* + *fls-integral* *g* \* *fls-const* *b*

$\langle \text{proof} \rangle$

**lemma** *fls-integral-sum*:

*fls-integral* (*sum* *f* *S*) = *sum* ( $\lambda i.$  *fls-integral* (*f* *i*)) *S*

$\langle \text{proof} \rangle$

### 7.7.7 Derivatives of integrals and vice versa

**lemma** *fls-integral-fls-deriv*:

**fixes** *a* :: 'a::{division-ring,ring-char-0} *fls*

**shows** *fls-integral* (*fls-deriv* *a*) + *fls-const* (*a*\$\$\$0) = *a*

$\langle \text{proof} \rangle$

**lemma** *fls-deriv-fls-integral*:

**fixes** *a* :: 'a::{division-ring,ring-char-0} *fls*

**assumes** *fls-residue* *a* = 0

**shows** *fls-deriv* (*fls-integral* *a*) = *a*

*<proof>*

Series with zero residue are precisely the derivatives.

**lemma** *fls-residue-nonzero-ex-antiderivative*:  
  **fixes**  $f :: 'a :: \{\text{division-ring}, \text{ring-char-0}\}$  *fls*  
  **assumes** *fls-residue*  $f = 0$   
  **shows**  $\exists F. \text{fls-deriv } F = f$   
  *<proof>*

**lemma** *fls-ex-antiderivative-residue-nonzero*:  
  **assumes**  $\exists F. \text{fls-deriv } F = f$   
  **shows** *fls-residue*  $f = 0$   
  *<proof>*

**lemma** *fls-residue-nonzero-ex-antiderivative-iff*:  
  **fixes**  $f :: 'a :: \{\text{division-ring}, \text{ring-char-0}\}$  *fls*  
  **shows** *fls-residue*  $f = 0 \longleftrightarrow (\exists F. \text{fls-deriv } F = f)$   
  *<proof>*

## 7.8 Topology

**instantiation** *fls* :: (*group-add*) *metric-space*  
**begin**

**definition** *dist-fls-def*:  
  *dist* ( $a :: 'a \text{ fls}$ )  $b =$   
    (*if*  $a = b$   
      *then* 0  
      *else if* *fls-subdegree* ( $a - b$ )  $\geq 0$   
        *then* *inverse* ( $2 \wedge \text{nat } (\text{fls-subdegree } (a - b))$ )  
        *else*  $2 \wedge \text{nat } (-\text{fls-subdegree } (a - b))$   
    )

**lemma** *dist-fls-ge0*: *dist* ( $a :: 'a \text{ fls}$ )  $b \geq 0$   
  *<proof>*

**definition** *uniformity-fls-def* [*code del*]:  
  (*uniformity* :: ( $'a \text{ fls} \times 'a \text{ fls}$ ) *filter*) = (*INF*  $e \in \{0 <..\}$ . *principal*  $\{(x, y). \text{dist } x \ y < e\}$ )

**definition** *open-fls-def'* [*code del*]:  
  *open* ( $U :: 'a \text{ fls set}$ )  $\longleftrightarrow (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U)$   
  *uniformity*)

**lemma** *dist-fls-sym*: *dist* ( $a :: 'a \text{ fls}$ )  $b = \text{dist } b \ a$   
  *<proof>*

**context**

**begin**

**private lemma** *instance-helper*:

**fixes**  $a\ b\ c :: 'a\ fls$

**assumes** *neg*:  $a \neq b\ a \neq c$

**and** *dist-ineq*:  $dist\ a\ b > dist\ a\ c$

**shows**  $fls\text{-}subdegree\ (a - b) < fls\text{-}subdegree\ (a - c)$

$\langle proof \rangle$

**instance**

$\langle proof \rangle$

**end**

**end**

**declare** *uniformity-Abort*[**where**  $'a = 'a :: group\text{-}add\ fls$ , *code*]

**lemma** *open-fls-def*:

$open\ (S :: 'a :: group\text{-}add\ fls\ set) = (\forall a \in S. \exists r. r > 0 \wedge \{y. dist\ y\ a < r\} \subseteq S)$

$\langle proof \rangle$

## 7.9 Notation

**bundle** *fps-syntax*

**begin**

**notation** *fls-nth* (**infixl**  $\langle \$\$ \rangle\ 75$ )

**end**

**unbundle** *no fps-syntax*

**end**

## 8 The fraction field of any integral domain

**theory** *Fraction-Field*

**imports** *Main*

**begin**

### 8.1 General fractions construction

#### 8.1.1 Construction of the type of fractions

**context** *idom* **begin**

**definition** *fractrel* ::  $'a \times 'a \Rightarrow 'a * 'a \Rightarrow bool$  **where**

$fractrel = (\lambda x\ y. snd\ x \neq 0 \wedge snd\ y \neq 0 \wedge fst\ x * snd\ y = fst\ y * snd\ x)$

**lemma** *fractrel-iff* [*simp*]:

$fractrel\ x\ y \longleftrightarrow snd\ x \neq 0 \wedge snd\ y \neq 0 \wedge fst\ x * snd\ y = fst\ y * snd\ x$



$\langle proof \rangle$

**lemma** *symp-fractrel*: *symp fractrel*  
 $\langle proof \rangle$

**lemma** *transp-fractrel*: *transp fractrel*  
 $\langle proof \rangle$

**lemma** *part-equivp-fractrel*: *part-equivp fractrel*  
 $\langle proof \rangle$

**end**

**quotient-type (overloaded)**  $'a \text{ fract} = 'a :: idom \times 'a / \text{partial: fractrel}$   
 $\langle proof \rangle$

### 8.1.2 Representation and basic operations

**lift-definition** *Fract* ::  $'a :: idom \Rightarrow 'a \Rightarrow 'a \text{ fract}$   
**is**  $\lambda a \ b. \text{if } b = 0 \text{ then } (0, 1) \text{ else } (a, b)$   
 $\langle proof \rangle$

**lemma** *Fract-cases* [*cases type: fract*]:  
**obtains**  $(\text{Fract}) \ a \ b$  **where**  $q = \text{Fract } a \ b \ b \neq 0$   
 $\langle proof \rangle$

**lemma** *Fract-induct* [*case-names Fract, induct type: fract*]:  
 $(\bigwedge a \ b. b \neq 0 \Longrightarrow P (\text{Fract } a \ b)) \Longrightarrow P \ q$   
 $\langle proof \rangle$

**lemma** *eq-fract*:  
**shows**  $\bigwedge a \ b \ c \ d. b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow \text{Fract } a \ b = \text{Fract } c \ d \longleftrightarrow a * d = c * b$   
**and**  $\bigwedge a. \text{Fract } a \ 0 = \text{Fract } 0 \ 1$   
**and**  $\bigwedge a \ c. \text{Fract } 0 \ a = \text{Fract } 0 \ c$   
 $\langle proof \rangle$

**instantiation** *fract* ::  $(idom) \text{ comm-ring-1}$   
**begin**

**lift-definition** *zero-fract* ::  $'a \text{ fract}$  **is**  $(0, 1)$   $\langle proof \rangle$

**lemma** *Zero-fract-def*:  $0 = \text{Fract } 0 \ 1$   
 $\langle proof \rangle$

**lift-definition** *one-fract* ::  $'a \text{ fract}$  **is**  $(1, 1)$   $\langle proof \rangle$

**lemma** *One-fract-def*:  $1 = \text{Fract } 1 \ 1$   
 $\langle proof \rangle$

**lift-definition** *plus-fract* :: 'a fract  $\Rightarrow$  'a fract  $\Rightarrow$  'a fract  
**is**  $\lambda q r. (fst\ q * snd\ r + fst\ r * snd\ q, snd\ q * snd\ r)$   
 $\langle proof \rangle$

**lemma** *add-fract* [*simp*]:  
 $\llbracket b \neq 0; d \neq 0 \rrbracket \Longrightarrow Fract\ a\ b + Fract\ c\ d = Fract\ (a * d + c * b)\ (b * d)$   
 $\langle proof \rangle$

**lift-definition** *uminus-fract* :: 'a fract  $\Rightarrow$  'a fract  
**is**  $\lambda x. (-\ fst\ x, snd\ x)$   
 $\langle proof \rangle$

**lemma** *minus-fract* [*simp*]:  
**fixes**  $a\ b :: 'a::idom$   
**shows**  $- Fract\ a\ b = Fract\ (-\ a)\ b$   
 $\langle proof \rangle$

**lemma** *minus-fract-cancel* [*simp*]:  $Fract\ (-\ a)\ (-\ b) = Fract\ a\ b$   
 $\langle proof \rangle$

**definition** *diff-fract-def*:  $q - r = q + -\ (r::'a\ fract)$

**lemma** *diff-fract* [*simp*]:  
 $\llbracket b \neq 0; d \neq 0 \rrbracket \Longrightarrow Fract\ a\ b - Fract\ c\ d = Fract\ (a * d - c * b)\ (b * d)$   
 $\langle proof \rangle$

**lift-definition** *times-fract* :: 'a fract  $\Rightarrow$  'a fract  $\Rightarrow$  'a fract  
**is**  $\lambda q r. (fst\ q * fst\ r, snd\ q * snd\ r)$   
 $\langle proof \rangle$

**lemma** *mult-fract* [*simp*]:  $Fract\ (a::'a::idom)\ b * Fract\ c\ d = Fract\ (a * c)\ (b * d)$   
 $\langle proof \rangle$

**lemma** *mult-fract-cancel*:  
 $c \neq 0 \Longrightarrow Fract\ (c * a)\ (c * b) = Fract\ a\ b$   
 $\langle proof \rangle$

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

**end**

**lemma** *of-nat-fract*:  $of\_nat\ k = Fract\ (of\_nat\ k)\ 1$   
 $\langle proof \rangle$

**lemma** *Fract-of-nat-eq*:  $Fract\ (of\_nat\ k)\ 1 = of\_nat\ k$   
 $\langle proof \rangle$

**lemma** *fract-collapse*:

```

    Fract 0 k = 0
    Fract 1 1 = 1
    Fract k 0 = 0
  <proof>

```

```

lemma fract-expand:
  0 = Fract 0 1
  1 = Fract 1 1
  <proof>

```

```

lemma Fract-cases-nonzero:
  obtains (Fract) a b where q = Fract a b and b ≠ 0 and a ≠ 0
    | (0) q = 0
  <proof>

```

### 8.1.3 The field of rational numbers

```

context idom
begin

```

```

subclass ring-no-zero-divisors <proof>

```

```

end

```

```

instantiation fract :: (idom) field
begin

```

```

lift-definition inverse-fract :: 'a fract ⇒ 'a fract
  is  $\lambda x.$  if fst x = 0 then (0, 1) else (snd x, fst x)
  <proof>

```

```

lemma inverse-fract [simp]: inverse (Fract a b) = Fract (b::'a::idom) a
  <proof>

```

```

definition divide-fract-def: q div r = q * inverse (r:: 'a fract)

```

```

lemma divide-fract [simp]: Fract a b div Fract c d = Fract (a * d) (b * c)
  <proof>

```

```

instance
  <proof>

```

```

end

```

### 8.1.4 The ordered field of fractions over an ordered idom

```

instantiation fract :: (linordered-idom) linorder
begin

```

```

lemma less-eq-fract-respect:

```

**fixes**  $a\ b\ a'\ b'\ c\ d\ c'\ d' :: 'a$   
**assumes**  $neg: b \neq 0\ b' \neq 0\ d \neq 0\ d' \neq 0$   
**assumes**  $eq1: a * b' = a' * b$   
**assumes**  $eq2: c * d' = c' * d$   
**shows**  $((a * d) * (b * d) \leq (c * b) * (b * d)) \longleftrightarrow ((a' * d') * (b' * d') \leq (c' * b') * (b' * d'))$   
 $\langle proof \rangle$

**lift-definition**  $less-eq-fract :: 'a\ fract \Rightarrow 'a\ fract \Rightarrow bool$   
**is**  $\lambda q\ r. (fst\ q * snd\ r) * (snd\ q * snd\ r) \leq (fst\ r * snd\ q) * (snd\ q * snd\ r)$   
 $\langle proof \rangle$

**definition**  $less-fract-def: z < (w::'a\ fract) \longleftrightarrow z \leq w \wedge \neg w \leq z$

**lemma**  $le-fract\ [simp]:$   
 $\llbracket b \neq 0; d \neq 0 \rrbracket \Longrightarrow Fract\ a\ b \leq Fract\ c\ d \longleftrightarrow (a * d) * (b * d) \leq (c * b) * (b * d)$   
 $\langle proof \rangle$

**lemma**  $less-fract\ [simp]:$   
 $\llbracket b \neq 0; d \neq 0 \rrbracket \Longrightarrow Fract\ a\ b < Fract\ c\ d \longleftrightarrow (a * d) * (b * d) < (c * b) * (b * d)$   
 $\langle proof \rangle$

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

**end**

**instantiation**  $fract :: (linordered-idom)\ linordered-field$   
**begin**

**definition**  $abs-fract-def2:$   
 $|q| = (if\ q < 0\ then\ -q\ else\ (q::'a\ fract))$

**definition**  $sgn-fract-def:$   
 $sgn\ (q::'a\ fract) = (if\ q = 0\ then\ 0\ else\ if\ 0 < q\ then\ 1\ else\ -1)$

**theorem**  $abs-fract\ [simp]: |Fract\ a\ b| = Fract\ |a|\ |b|$   
 $\langle proof \rangle$

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

**end**

**instantiation**  $fract :: (linordered-idom)\ distrib-lattice$   
**begin**

**definition**  $inf-fract-def:$

```

    (inf :: 'a fract  $\Rightarrow$  'a fract  $\Rightarrow$  'a fract) = min

definition sup-fract-def:
  (sup :: 'a fract  $\Rightarrow$  'a fract  $\Rightarrow$  'a fract) = max

instance
  <proof>

end

lemma fract-induct-pos [case-names Fract]:
  fixes P :: 'a::linordered-idom fract  $\Rightarrow$  bool
  assumes step:  $\bigwedge a b. 0 < b \implies P (Fract a b)$ 
  shows P q
  <proof>

lemma zero-less-Fract-iff:  $0 < b \implies 0 < Fract a b \longleftrightarrow 0 < a$ 
  <proof>

lemma Fract-less-zero-iff:  $0 < b \implies Fract a b < 0 \longleftrightarrow a < 0$ 
  <proof>

lemma zero-le-Fract-iff:  $0 < b \implies 0 \leq Fract a b \longleftrightarrow 0 \leq a$ 
  <proof>

lemma Fract-le-zero-iff:  $0 < b \implies Fract a b \leq 0 \longleftrightarrow a \leq 0$ 
  <proof>

lemma one-less-Fract-iff:  $0 < b \implies 1 < Fract a b \longleftrightarrow b < a$ 
  <proof>

lemma Fract-less-one-iff:  $0 < b \implies Fract a b < 1 \longleftrightarrow a < b$ 
  <proof>

lemma one-le-Fract-iff:  $0 < b \implies 1 \leq Fract a b \longleftrightarrow b \leq a$ 
  <proof>

lemma Fract-le-one-iff:  $0 < b \implies Fract a b \leq 1 \longleftrightarrow a \leq b$ 
  <proof>

end

```

## 9 Fundamental Theorem of Algebra

```

theory Fundamental-Theorem-Algebra
imports Polynomial Complex-Main
begin

```

## 9.1 More lemmas about module of complex numbers

The triangle inequality for cmod

**lemma** *complex-mod-triangle-sub*:  $cmod\ w \leq cmod\ (w + z) + norm\ z$   
 $\langle proof \rangle$

## 9.2 Basic lemmas about polynomials

**lemma** *poly-bound-exists*:

**fixes**  $p :: 'a::\{comm-semiring-0, real-normed-div-algebra\}$  *poly*  
**shows**  $\exists m. m > 0 \wedge (\forall z. norm\ z \leq r \longrightarrow norm\ (poly\ p\ z) \leq m)$   
 $\langle proof \rangle$

Offsetting the variable in a polynomial gives another of same degree

**definition** *offset-poly* ::  $'a::comm-semiring-0$  *poly*  $\Rightarrow 'a \Rightarrow 'a$  *poly*  
**where** *offset-poly*  $p\ h = fold-coeffs\ (\lambda a\ q. smult\ h\ q + pCons\ a\ q)\ p\ 0$

**lemma** *offset-poly-0*: *offset-poly*  $0\ h = 0$   
 $\langle proof \rangle$

**lemma** *offset-poly-pCons*:

*offset-poly*  $(pCons\ a\ p)\ h =$   
 $smult\ h\ (offset-poly\ p\ h) + pCons\ a\ (offset-poly\ p\ h)$   
 $\langle proof \rangle$

**lemma** *offset-poly-single* [simp]: *offset-poly*  $[:a:] h = [:a:]$   
 $\langle proof \rangle$

**lemma** *poly-offset-poly*: *poly*  $(offset-poly\ p\ h)\ x = poly\ p\ (h + x)$   
 $\langle proof \rangle$

**lemma** *offset-poly-eq-0-lemma*:  $smult\ c\ p + pCons\ a\ p = 0 \implies p = 0$   
 $\langle proof \rangle$

**lemma** *offset-poly-eq-0-iff* [simp]: *offset-poly*  $p\ h = 0 \longleftrightarrow p = 0$   
 $\langle proof \rangle$

**lemma** *degree-offset-poly* [simp]: *degree*  $(offset-poly\ p\ h) = degree\ p$   
 $\langle proof \rangle$

**definition** *psize*  $p = (if\ p = 0\ then\ 0\ else\ Suc\ (degree\ p))$

**lemma** *psize-eq-0-iff* [simp]: *psize*  $p = 0 \longleftrightarrow p = 0$   
 $\langle proof \rangle$

**lemma** *poly-offset*:

**fixes**  $p :: 'a::comm-ring-1$  *poly*  
**shows**  $\exists q. psize\ q = psize\ p \wedge (\forall x. poly\ q\ x = poly\ p\ (a + x))$   
 $\langle proof \rangle$

An alternative useful formulation of completeness of the reals

**lemma** *real-sup-exists*:

**assumes** *ex*:  $\exists x. P\ x$

**and** *bz*:  $\exists z. \forall x. P\ x \longrightarrow x < z$

**shows**  $\exists s::\text{real}. \forall y. (\exists x. P\ x \wedge y < x) \longleftrightarrow y < s$

*<proof>*

### 9.3 Fundamental theorem of algebra

**lemma** *unimodular-reduce-norm*:

**assumes** *md*:  $\text{cmod } z = 1$

**shows**  $\text{cmod } (z + 1) < 1 \vee \text{cmod } (z - 1) < 1 \vee \text{cmod } (z + i) < 1 \vee \text{cmod } (z - i) < 1$

*<proof>*

Hence we can always reduce modulus of  $1 + b z^n$  if nonzero

**lemma** *reduce-poly-simple*:

**assumes** *b*:  $b \neq 0$

**and** *n*:  $n \neq 0$

**shows**  $\exists z. \text{cmod } (1 + b * z^n) < 1$

*<proof>*

Bolzano-Weierstrass type property for closed disc in complex plane.

**lemma** *metric-bound-lemma*:  $\text{cmod } (x - y) \leq |\text{Re } x - \text{Re } y| + |\text{Im } x - \text{Im } y|$

*<proof>*

**lemma** *Bolzano-Weierstrass-complex-disc*:

**assumes** *r*:  $\forall n. \text{cmod } (s\ n) \leq r$

**shows**  $\exists f\ z. \text{strict-mono } (f :: \text{nat} \Rightarrow \text{nat}) \wedge (\forall e > 0. \exists N. \forall n \geq N. \text{cmod } (s\ (f\ n) - z) < e)$

*<proof>*

Polynomial is continuous.

**lemma** *poly-cont*:

**fixes** *p* :: 'a::{comm-semiring-0,real-normed-div-algebra} poly

**assumes** *ep*:  $e > 0$

**shows**  $\exists d > 0. \forall w. 0 < \text{norm } (w - z) \wedge \text{norm } (w - z) < d \longrightarrow \text{norm } (\text{poly } p\ w - \text{poly } p\ z) < e$

*<proof>*

Hence a polynomial attains minimum on a closed disc in the complex plane.

**lemma** *poly-minimum-modulus-disc*:  $\exists z. \forall w. \text{cmod } w \leq r \longrightarrow \text{cmod } (\text{poly } p\ z) \leq \text{cmod } (\text{poly } p\ w)$

*<proof>*

Nonzero polynomial in z goes to infinity as z does.

**lemma** *poly-infinity*:

**fixes** *p*:: 'a::{comm-semiring-0,real-normed-div-algebra} poly

**assumes**  $ex: p \neq 0$   
**shows**  $\exists r. \forall z. r \leq \text{norm } z \longrightarrow d \leq \text{norm } (\text{poly } (pCons \ a \ p) \ z)$   
 $\langle \text{proof} \rangle$

Hence polynomial's modulus attains its minimum somewhere.

**lemma** *poly-minimum-modulus*:  $\exists z. \forall w. \text{cmod } (\text{poly } p \ z) \leq \text{cmod } (\text{poly } p \ w)$   
 $\langle \text{proof} \rangle$

Constant function (non-syntactic characterization).

**definition** *constant*  $f \longleftrightarrow (\forall x \ y. f \ x = f \ y)$

**lemma** *nonconstant-length*:  $\neg \text{constant } (\text{poly } p) \implies \text{psize } p \geq 2$   
 $\langle \text{proof} \rangle$

**lemma** *poly-replicate-append*:  $\text{poly } (\text{monom } 1 \ n \ * \ p) \ (x::'a::\text{comm-ring-1}) = x^{\wedge n} \ * \ \text{poly } p \ x$   
 $\langle \text{proof} \rangle$

Decomposition of polynomial, skipping zero coefficients after the first.

**lemma** *poly-decompose-lemma*:  
**assumes**  $nz: \neg (\forall z. z \neq 0 \longrightarrow \text{poly } p \ z = (0::'a::\text{idom}))$   
**shows**  $\exists k \ a \ q. a \neq 0 \wedge \text{Suc } (\text{psize } q + k) = \text{psize } p \wedge (\forall z. \text{poly } p \ z = z^{\wedge k} \ * \ \text{poly } (pCons \ a \ q) \ z)$   
 $\langle \text{proof} \rangle$

**lemma** *poly-decompose*:  
**fixes**  $p :: 'a::\text{idom} \ \text{poly}$   
**assumes**  $nc: \neg \text{constant } (\text{poly } p)$   
**shows**  $\exists k \ a \ q. a \neq 0 \wedge k \neq 0 \wedge$   
 $\text{psize } q + k + 1 = \text{psize } p \wedge$   
 $(\forall z. \text{poly } p \ z = \text{poly } p \ 0 + z^{\wedge k} \ * \ \text{poly } (pCons \ a \ q) \ z)$   
 $\langle \text{proof} \rangle$

Fundamental theorem of algebra

**theorem** *fundamental-theorem-of-algebra*:  
**assumes**  $nc: \neg \text{constant } (\text{poly } p)$   
**shows**  $\exists z::\text{complex}. \text{poly } p \ z = 0$   
 $\langle \text{proof} \rangle$

Alternative version with a syntactic notion of constant polynomial.

**lemma** *fundamental-theorem-of-algebra-alt*:  
**assumes**  $nc: \neg (\exists a \ l. a \neq 0 \wedge l = 0 \wedge p = pCons \ a \ l)$   
**shows**  $\exists z. \text{poly } p \ z = (0::\text{complex})$   
 $\langle \text{proof} \rangle$

## 9.4 Nullstellensatz, degrees and divisibility of polynomials

**lemma** *nullstellensatz-lemma*:



**fixes**  $p :: \text{complex poly}$   
**assumes**  $\forall x. \text{poly } p \ x = 0 \longrightarrow \text{poly } q \ x = 0$   
**and**  $\text{degree } p = n$   
**and**  $n \neq 0$   
**shows**  $p \text{ dvd } (q \wedge n)$   
 $\langle \text{proof} \rangle$

**lemma** *nullstellensatz-univariate*:  
 $(\forall x. \text{poly } p \ x = (0 :: \text{complex}) \longrightarrow \text{poly } q \ x = 0) \longleftrightarrow$   
 $p \text{ dvd } (q \wedge (\text{degree } p)) \vee (p = 0 \wedge q = 0)$   
 $\langle \text{proof} \rangle$

Useful lemma

**lemma** *constant-degree*:  
**fixes**  $p :: 'a :: \{\text{idom}, \text{ring-char-0}\} \text{ poly}$   
**shows**  $\text{constant } (\text{poly } p) \longleftrightarrow \text{degree } p = 0$  (**is**  $?lhs = ?rhs$ )  
 $\langle \text{proof} \rangle$

**lemma** *complex-poly-decompose*:  
 $\text{smult } (\text{lead-coeff } p) \ (\prod z | \text{poly } p \ z = 0. [-z, 1:] \wedge \text{order } z \ p) = (p :: \text{complex poly})$   
 $\langle \text{proof} \rangle$

**instance** *complex* :: *alg-closed-field*  
 $\langle \text{proof} \rangle$

**lemma** *size-roots-complex*:  $\text{size } (\text{roots } (p :: \text{complex poly})) = \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma** *complex-poly-decompose-multiset*:  
 $\text{smult } (\text{lead-coeff } p) \ (\prod x \in \# \text{roots } p. [-x, 1:]) = (p :: \text{complex poly})$   
 $\langle \text{proof} \rangle$

**lemma** *complex-poly-decompose'*:  
**obtains** **root** **where**  $\text{smult } (\text{lead-coeff } p) \ (\prod i < \text{degree } p. [-\text{root } i, 1:]) = (p :: \text{complex poly})$   
 $\langle \text{proof} \rangle$

**lemma** *complex-poly-decompose-rsquarefree*:  
**assumes**  $\text{rsquarefree } p$   
**shows**  $\text{smult } (\text{lead-coeff } p) \ (\prod z | \text{poly } p \ z = 0. [-z, 1:]) = (p :: \text{complex poly})$   
 $\langle \text{proof} \rangle$

Arithmetic operations on multivariate polynomials.

**lemma** *mpoly-base-conv*:  
**fixes**  $x :: 'a :: \text{comm-ring-1}$   
**shows**  $0 = \text{poly } 0 \ x \ c = \text{poly } [:c:] \ x \ x = \text{poly } [:0, 1:] \ x$   
 $\langle \text{proof} \rangle$

**lemma** *mpoly-norm-conv*:

```

fixes  $x :: 'a::comm-ring-1$ 
shows  $poly\ [0:]\ x = poly\ 0\ x\ poly\ [:poly\ 0\ y:]\ x = poly\ 0\ x$ 
 $\langle proof \rangle$ 

lemma mpoly-sub-conv:
fixes  $x :: 'a::comm-ring-1$ 
shows  $poly\ p\ x - poly\ q\ x = poly\ p\ x + -1 * poly\ q\ x$ 
 $\langle proof \rangle$ 

lemma poly-pad-rule:  $poly\ p\ x = 0 \implies poly\ (pCons\ 0\ p)\ x = 0$ 
 $\langle proof \rangle$ 

lemma poly-cancel-eq-conv:
fixes  $x :: 'a::field$ 
shows  $x = 0 \implies a \neq 0 \implies y = 0 \longleftrightarrow a * y - b * x = 0$ 
 $\langle proof \rangle$ 

lemma poly-divides-pad-rule:
fixes  $p:: ('a::comm-ring-1)\ poly$ 
assumes  $pq: p\ dvd\ q$ 
shows  $p\ dvd\ (pCons\ 0\ q)$ 
 $\langle proof \rangle$ 

lemma poly-divides-conv0:
fixes  $p:: 'a::field\ poly$ 
assumes  $lqpq: degree\ q < degree\ p$  and  $lq: p \neq 0$ 
shows  $p\ dvd\ q \longleftrightarrow q = 0$ 
 $\langle proof \rangle$ 

lemma poly-divides-conv1:
fixes  $p :: 'a::field\ poly$ 
assumes  $a0: a \neq 0$ 
and  $pp': p\ dvd\ p'$ 
and  $grp': smult\ a\ q - p' = r$ 
shows  $p\ dvd\ q \longleftrightarrow p\ dvd\ r$ 
 $\langle proof \rangle$ 

lemma basic-cqe-conv1:
 $(\exists x. poly\ p\ x = 0 \wedge poly\ 0\ x \neq 0) \longleftrightarrow False$ 
 $(\exists x. poly\ 0\ x \neq 0) \longleftrightarrow False$ 
 $(\exists x. poly\ [:c:]\ x \neq 0) \longleftrightarrow c \neq 0$ 
 $(\exists x. poly\ 0\ x = 0) \longleftrightarrow True$ 
 $(\exists x. poly\ [:c:]\ x = 0) \longleftrightarrow c = 0$ 
 $\langle proof \rangle$ 

lemma basic-cqe-conv2:
assumes  $l: p \neq 0$ 
shows  $\exists x. poly\ (pCons\ a\ (pCons\ b\ p))\ x = (0::complex)$ 
 $\langle proof \rangle$ 

```

**lemma** *basic-cqe-conv-2b*:  $(\exists x. \text{poly } p \ x \neq (0::\text{complex})) \longleftrightarrow p \neq 0$   
 <proof>

**lemma** *basic-cqe-conv3*:  
 fixes  $p \ q :: \text{complex poly}$   
 assumes  $l: p \neq 0$   
 shows  $(\exists x. \text{poly } (p\text{Cons } a \ p) \ x = 0 \wedge \text{poly } q \ x \neq 0) \longleftrightarrow \neg (p\text{Cons } a \ p) \text{ dvd } (q \wedge_{\text{psize } p})$   
 <proof>

**lemma** *basic-cqe-conv4*:  
 fixes  $p \ q :: \text{complex poly}$   
 assumes  $h: \bigwedge x. \text{poly } (q \wedge^n) \ x = \text{poly } r \ x$   
 shows  $p \text{ dvd } (q \wedge^n) \longleftrightarrow p \text{ dvd } r$   
 <proof>

**lemma** *poly-const-conv*:  
 fixes  $x :: 'a::\text{comm-ring-1}$   
 shows  $\text{poly } [:c:] \ x = y \longleftrightarrow c = y$   
 <proof>

**end**

**theory** *Group-Closure*  
**imports**  
*Main*  
**begin**

**context** *ab-group-add*  
**begin**

**inductive-set** *group-closure* ::  $'a \text{ set} \Rightarrow 'a \text{ set}$  **for**  $S$   
**where** *base*:  $s \in \text{insert } 0 \ S \Longrightarrow s \in \text{group-closure } S$   
 | *diff*:  $s \in \text{group-closure } S \Longrightarrow t \in \text{group-closure } S \Longrightarrow s - t \in \text{group-closure } S$

**lemma** *zero-in-group-closure* [*simp*]:  
 $0 \in \text{group-closure } S$   
 <proof>

**lemma** *group-closure-minus-iff* [*simp*]:  
 $s \in \text{group-closure } S \longleftrightarrow s \in \text{group-closure } S$   
 <proof>

**lemma** *group-closure-add*:  
 $s + t \in \text{group-closure } S$  **if**  $s \in \text{group-closure } S$  **and**  $t \in \text{group-closure } S$   
 <proof>

```

lemma group-closure-empty [simp]:
  group-closure {} = {0}
  ⟨proof⟩

lemma group-closure-insert-zero [simp]:
  group-closure (insert 0 S) = group-closure S
  ⟨proof⟩

end

context comm-ring-1
begin

lemma group-closure-scalar-mult-left:
  of-nat n * s ∈ group-closure S if s ∈ group-closure S
  ⟨proof⟩

lemma group-closure-scalar-mult-right:
  s * of-nat n ∈ group-closure S if s ∈ group-closure S
  ⟨proof⟩

end

lemma group-closure-abs-iff [simp]:
  |s| ∈ group-closure S  $\longleftrightarrow$  s ∈ group-closure S for s :: int
  ⟨proof⟩

lemma group-closure-mult-left:
  s * t ∈ group-closure S if s ∈ group-closure S for s t :: int
  ⟨proof⟩

lemma group-closure-mult-right:
  s * t ∈ group-closure S if t ∈ group-closure S for s t :: int
  ⟨proof⟩

context idom
begin

lemma group-closure-mult-all-eq:
  group-closure (times k ' S) = times k ' group-closure S
  ⟨proof⟩

end

lemma Gcd-group-closure-eq-Gcd:
  Gcd (group-closure S) = Gcd S for S :: int set
  ⟨proof⟩

lemma group-closure-sum:

```

```

fixes  $S :: \text{int set}$ 
assumes  $X: \text{finite } X \ X \neq \{\} \ X \subseteq S$ 
shows  $(\sum_{x \in X}. a \ x * x) \in \text{group-closure } S$ 
 $\langle \text{proof} \rangle$ 

lemma Gcd-group-closure-in-group-closure:
   $\text{Gcd } (\text{group-closure } S) \in \text{group-closure } S$  for  $S :: \text{int set}$ 
 $\langle \text{proof} \rangle$ 

lemma Gcd-in-group-closure:
   $\text{Gcd } S \in \text{group-closure } S$  for  $S :: \text{int set}$ 
 $\langle \text{proof} \rangle$ 

lemma group-closure-eq:
   $\text{group-closure } S = \text{range } (\text{times } (\text{Gcd } S))$  for  $S :: \text{int set}$ 
 $\langle \text{proof} \rangle$ 

end

theory Normalized-Fraction
imports
  Main
  Euclidean-Algorithm
  Fraction-Field
begin

lemma unit-factor-1-imp-normalized:  $\text{unit-factor } x = 1 \implies \text{normalize } x = x$ 
 $\langle \text{proof} \rangle$ 

definition quot-to-fract ::  $'a \times 'a \Rightarrow 'a :: \text{idom fract}$  where
   $\text{quot-to-fract} = (\lambda(a,b). \text{Fraction-Field.Fract } a \ b)$ 

definition normalize-quot ::  $'a :: \{\text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$ 
 $\times 'a \Rightarrow 'a \times 'a$  where
   $\text{normalize-quot} =$ 
     $(\lambda(a,b). \text{if } b = 0 \text{ then } (0,1) \text{ else let } d = \text{gcd } a \ b * \text{unit-factor } b \text{ in } (a \text{ div } d, b \text{ div } d))$ 

lemma normalize-quot-zero [simp]:
   $\text{normalize-quot } (a, 0) = (0, 1)$ 
 $\langle \text{proof} \rangle$ 

lemma normalize-quot-proj:
   $\text{fst } (\text{normalize-quot } (a, b)) = a \text{ div } (\text{gcd } a \ b * \text{unit-factor } b)$ 
   $\text{snd } (\text{normalize-quot } (a, b)) = \text{normalize } b \text{ div gcd } a \ b$  if  $b \neq 0$ 
 $\langle \text{proof} \rangle$ 

definition normalized-fracts ::  $( 'a :: \{\text{ring-gcd}, \text{idom-divide}\} \times 'a)$  set where

```

$normalized\_fracts = \{(a,b). \text{ coprime } a \ b \wedge \text{ unit-factor } b = 1\}$

**lemma** *not-normalized-fracts-0-denom* [simp]:  $(a, 0) \notin normalized\_fracts$   
 <proof>

**lemma** *unit-factor-snd-normalize-quot* [simp]:  
 $\text{unit-factor } (\text{snd } (\text{normalize-quot } x)) = 1$   
 <proof>

**lemma** *snd-normalize-quot-nonzero* [simp]:  $\text{snd } (\text{normalize-quot } x) \neq 0$   
 <proof>

**lemma** *normalize-quot-aux*:  
 fixes  $a \ b$   
 assumes  $b \neq 0$   
 defines  $d \equiv \text{gcd } a \ b * \text{unit-factor } b$   
 shows  $a = \text{fst } (\text{normalize-quot } (a,b)) * d \ b = \text{snd } (\text{normalize-quot } (a,b)) * d$   
 $d \ \text{dvd} \ a \ d \ \text{dvd} \ b \ d \neq 0$   
 <proof>

**lemma** *normalize-quotE*:  
 assumes  $b \neq 0$   
 obtains  $d$  **where**  $a = \text{fst } (\text{normalize-quot } (a,b)) * d \ b = \text{snd } (\text{normalize-quot } (a,b)) * d$   
 $d \ \text{dvd} \ a \ d \ \text{dvd} \ b \ d \neq 0$   
 <proof>

**lemma** *normalize-quotE'*:  
 assumes  $\text{snd } x \neq 0$   
 obtains  $d$  **where**  $\text{fst } x = \text{fst } (\text{normalize-quot } x) * d \ \text{snd } x = \text{snd } (\text{normalize-quot } x) * d$   
 $d \ \text{dvd} \ \text{fst } x \ d \ \text{dvd} \ \text{snd } x \ d \neq 0$   
 <proof>

**lemma** *coprime-normalize-quot*:  
 $\text{coprime } (\text{fst } (\text{normalize-quot } x)) (\text{snd } (\text{normalize-quot } x))$   
 <proof>

**lemma** *normalize-quot-in-normalized-fracts* [simp]:  $\text{normalize-quot } x \in \text{normalized-fracts}$   
 <proof>

**lemma** *normalize-quot-eq-iff*:  
 assumes  $b \neq 0 \ d \neq 0$   
 shows  $\text{normalize-quot } (a,b) = \text{normalize-quot } (c,d) \longleftrightarrow a * d = b * c$   
 <proof>

**lemma** *normalize-quot-eq-iff'*:  
 assumes  $\text{snd } x \neq 0 \ \text{snd } y \neq 0$

**shows**  $\text{normalize-quot } x = \text{normalize-quot } y \longleftrightarrow \text{fst } x * \text{snd } y = \text{snd } x * \text{fst } y$   
 $\langle \text{proof} \rangle$

**lemma** *normalize-quot-id*:  $x \in \text{normalized-fracts} \implies \text{normalize-quot } x = x$   
 $\langle \text{proof} \rangle$

**lemma** *normalize-quot-idem* [simp]:  $\text{normalize-quot } (\text{normalize-quot } x) = \text{normalize-quot } x$   
 $\langle \text{proof} \rangle$

**lemma** *fractrel-iff-normalize-quot-eq*:  
 $\text{fractrel } x \ y \longleftrightarrow \text{normalize-quot } x = \text{normalize-quot } y \wedge \text{snd } x \neq 0 \wedge \text{snd } y \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *fractrel-normalize-quot-left*:  
**assumes**  $\text{snd } x \neq 0$   
**shows**  $\text{fractrel } (\text{normalize-quot } x) \ y \longleftrightarrow \text{fractrel } x \ y$   
 $\langle \text{proof} \rangle$

**lemma** *fractrel-normalize-quot-right*:  
**assumes**  $\text{snd } x \neq 0$   
**shows**  $\text{fractrel } y \ (\text{normalize-quot } x) \longleftrightarrow \text{fractrel } y \ x$   
 $\langle \text{proof} \rangle$

**lift-definition** *quot-of-fract* ::  
 $'a :: \{\text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\} \text{ fract} \Rightarrow 'a \times 'a$   
**is** *normalize-quot*  
 $\langle \text{proof} \rangle$

**lemma** *quot-to-fract-quot-of-fract* [simp]:  $\text{quot-to-fract } (\text{quot-of-fract } x) = x$   
 $\langle \text{proof} \rangle$

**lemma** *quot-of-fract-quot-to-fract*:  $\text{quot-of-fract } (\text{quot-to-fract } x) = \text{normalize-quot } x$   
 $\langle \text{proof} \rangle$

**lemma** *quot-of-fract-quot-to-fract'*:  
 $x \in \text{normalized-fracts} \implies \text{quot-of-fract } (\text{quot-to-fract } x) = x$   
 $\langle \text{proof} \rangle$

**lemma** *quot-of-fract-in-normalized-fracts* [simp]:  $\text{quot-of-fract } x \in \text{normalized-fracts}$   
 $\langle \text{proof} \rangle$

**lemma** *normalize-quotI*:  
**assumes**  $a * d = b * c \ \text{snd } a \neq 0 \ (c, d) \in \text{normalized-fracts}$   
**shows**  $\text{normalize-quot } (a, b) = (c, d)$   
 $\langle \text{proof} \rangle$

**lemma** *td-normalized-fract*:  
*type-definition quot-of-fract quot-to-fract normalized-fracts*  
*<proof>*

**lemma** *quot-of-fract-add-aux*:  
**assumes**  $\text{snd } x \neq 0 \text{ snd } y \neq 0$   
**shows**  $(\text{fst } x * \text{snd } y + \text{fst } y * \text{snd } x) * (\text{snd } (\text{normalize-quot } x) * \text{snd } (\text{normalize-quot } y)) =$   
 $\text{snd } x * \text{snd } y * (\text{fst } (\text{normalize-quot } x) * \text{snd } (\text{normalize-quot } y) +$   
 $\text{snd } (\text{normalize-quot } x) * \text{fst } (\text{normalize-quot } y))$   
*<proof>*

**locale** *fract-as-normalized-quot*  
**begin**  
**setup-lifting** *td-normalized-fract*  
**end**

**lemma** *quot-of-fract-add*:  
 $\text{quot-of-fract } (x + y) =$   
 $(\text{let } (a,b) = \text{quot-of-fract } x; (c,d) = \text{quot-of-fract } y$   
 $\text{in } \text{normalize-quot } (a * d + b * c, b * d))$   
*<proof>*

**lemma** *quot-of-fract-uminus*:  
 $\text{quot-of-fract } (-x) = (\text{let } (a,b) = \text{quot-of-fract } x \text{ in } (-a, b))$   
*<proof>*

**lemma** *quot-of-fract-diff*:  
 $\text{quot-of-fract } (x - y) =$   
 $(\text{let } (a,b) = \text{quot-of-fract } x; (c,d) = \text{quot-of-fract } y$   
 $\text{in } \text{normalize-quot } (a * d - b * c, b * d))$  (**is** - = ?*rhs*)  
*<proof>*

**lemma** *normalize-quot-mult-coprime*:  
**assumes**  $\text{coprime } a \ b \ \text{coprime } c \ d \ \text{unit-factor } b = 1 \ \text{unit-factor } d = 1$   
**defines**  $e \equiv \text{fst } (\text{normalize-quot } (a, d))$  **and**  $f \equiv \text{snd } (\text{normalize-quot } (a, d))$   
**and**  $g \equiv \text{fst } (\text{normalize-quot } (c, b))$  **and**  $h \equiv \text{snd } (\text{normalize-quot } (c, b))$   
**shows**  $\text{normalize-quot } (a * c, b * d) = (e * g, f * h)$   
*<proof>*

**lemma** *normalize-quot-mult*:  
**assumes**  $\text{snd } x \neq 0 \text{ snd } y \neq 0$   
**shows**  $\text{normalize-quot } (\text{fst } x * \text{fst } y, \text{snd } x * \text{snd } y) = \text{normalize-quot } (\text{fst } (\text{normalize-quot } x) * \text{fst } (\text{normalize-quot } y),$   
 $\text{snd } (\text{normalize-quot } x) * \text{snd } (\text{normalize-quot } y))$   
*<proof>*



**lemma** *quot-of-fract-mult*:

*quot-of-fract* ( $x * y$ ) =  
 (let ( $a, b$ ) = *quot-of-fract*  $x$ ; ( $c, d$ ) = *quot-of-fract*  $y$ ;  
       ( $e, f$ ) = *normalize-quot* ( $a, d$ ); ( $g, h$ ) = *normalize-quot* ( $c, b$ )  
       in ( $e * g, f * h$ ))  
 <proof>

**lemma** *normalize-quot-0* [*simp*]:

*normalize-quot* ( $0, x$ ) = ( $0, 1$ ) *normalize-quot* ( $x, 0$ ) = ( $0, 1$ )  
 <proof>

**lemma** *normalize-quot-eq-0-iff* [*simp*]:  $\text{fst } (\text{normalize-quot } x) = 0 \iff \text{fst } x = 0$

$\vee \text{snd } x = 0$   
 <proof>

**lemma** *fst-quot-of-fract-0-imp*:  $\text{fst } (\text{quot-of-fract } x) = 0 \implies \text{snd } (\text{quot-of-fract } x) = 1$

<proof>

**lemma** *normalize-quot-swap*:

**assumes**  $a \neq 0$   $b \neq 0$   
**defines**  $a' \equiv \text{fst } (\text{normalize-quot } (a, b))$  **and**  $b' \equiv \text{snd } (\text{normalize-quot } (a, b))$   
**shows**  $\text{normalize-quot } (b, a) = (b' \text{ div unit-factor } a', a' \text{ div unit-factor } a')$   
 <proof>

**lemma** *quot-of-fract-inverse*:

*quot-of-fract* (*inverse*  $x$ ) =  
 (let ( $a, b$ ) = *quot-of-fract*  $x$ ;  $d = \text{unit-factor } a$   
       in if  $d = 0$  then ( $0, 1$ ) else ( $b \text{ div } d, a \text{ div } d$ ))  
 <proof>

**lemma** *normalize-quot-div-unit-left*:

**fixes**  $x$   $y$   $u$   
**assumes** *is-unit*  $u$   
**defines**  $x' \equiv \text{fst } (\text{normalize-quot } (x, y))$  **and**  $y' \equiv \text{snd } (\text{normalize-quot } (x, y))$   
**shows**  $\text{normalize-quot } (x \text{ div } u, y) = (x' \text{ div } u, y')$   
 <proof>

**lemma** *normalize-quot-div-unit-right*:

**fixes**  $x$   $y$   $u$   
**assumes** *is-unit*  $u$   
**defines**  $x' \equiv \text{fst } (\text{normalize-quot } (x, y))$  **and**  $y' \equiv \text{snd } (\text{normalize-quot } (x, y))$   
**shows**  $\text{normalize-quot } (x, y \text{ div } u) = (x' * u, y')$   
 <proof>

**lemma** *normalize-quot-normalize-left*:

**fixes**  $x$   $y$   $u$   
**defines**  $x' \equiv \text{fst } (\text{normalize-quot } (x, y))$  **and**  $y' \equiv \text{snd } (\text{normalize-quot } (x, y))$   
**shows**  $\text{normalize-quot } (\text{normalize } x, y) = (x' \text{ div unit-factor } x, y')$

*<proof>*

**lemma** *normalize-quot-normalize-right*:

**fixes**  $x\ y\ u$

**defines**  $x' \equiv \text{fst } (\text{normalize-quot } (x, y))$  **and**  $y' \equiv \text{snd } (\text{normalize-quot } (x, y))$

**shows**  $\text{normalize-quot } (x, \text{normalize } y) = (x' * \text{unit-factor } y, y')$

*<proof>*

**lemma** *quot-of-fract-0 [simp]*:  $\text{quot-of-fract } 0 = (0, 1)$

*<proof>*

**lemma** *quot-of-fract-1 [simp]*:  $\text{quot-of-fract } 1 = (1, 1)$

*<proof>*

**lemma** *quot-of-fract-divide*:

$\text{quot-of-fract } (x / y) = (\text{if } y = 0 \text{ then } (0, 1) \text{ else}$

$(\text{let } (a,b) = \text{quot-of-fract } x; (c,d) = \text{quot-of-fract } y;$

$(e,f) = \text{normalize-quot } (a,c); (g,h) = \text{normalize-quot } (d,b)$

$\text{in } (e * g, f * h)))$  **(is - = ?rhs)**

*<proof>*

**lemma** *snd-quot-of-fract-nonzero [simp]*:  $\text{snd } (\text{quot-of-fract } x) \neq 0$

*<proof>*

**lemma** *Fract-quot-of-fract [simp]*:  $\text{Fract } (\text{fst } (\text{quot-of-fract } x)) (\text{snd } (\text{quot-of-fract } x)) = x$

*<proof>*

**lemma** *snd-quot-of-fract-Fract-whole*:

**assumes**  $y \text{ dvd } x$

**shows**  $\text{snd } (\text{quot-of-fract } (\text{Fract } x\ y)) = 1$

*<proof>*

**lemma** *fst-quot-of-fract-eq-0-iff [simp]*:  $\text{fst } (\text{quot-of-fract } x) = 0 \longleftrightarrow x = 0$

*<proof>*

**lemma** *coprime-quot-of-fract*:

$\text{coprime } (\text{fst } (\text{quot-of-fract } x)) (\text{snd } (\text{quot-of-fract } x))$

*<proof>*

**lemma** *unit-factor-snd-quot-of-fract*:  $\text{unit-factor } (\text{snd } (\text{quot-of-fract } x)) = 1$

*<proof>*

**lemma** *normalize-snd-quot-of-fract*:  $\text{normalize } (\text{snd } (\text{quot-of-fract } x)) = \text{snd } (\text{quot-of-fract } x)$

*<proof>*

**end**

## 10 $n$ -th powers and roots of naturals

```
theory Nth-Powers
  imports Primes
begin
```

### 10.1 The set of $n$ -th powers

**definition** *is-nth-power* ::  $\text{nat} \Rightarrow 'a :: \text{monoid-mult} \Rightarrow \text{bool}$  **where**  
*is-nth-power*  $n\ x \longleftrightarrow (\exists y. x = y \wedge n)$

**lemma** *is-nth-power-nth-power* [*simp, intro*]: *is-nth-power*  $n\ (x \wedge n)$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-powerI* [*intro?*]:  $x = y \wedge n \implies \text{is-nth-power } n\ x$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-powerE*: *is-nth-power*  $n\ x \implies (\bigwedge y. x = y \wedge n \implies P) \implies P$   
 $\langle \text{proof} \rangle$

**abbreviation** *is-square* **where** *is-square*  $\equiv \text{is-nth-power } 2$

**lemma** *is-zeroth-power* [*simp*]: *is-nth-power*  $0\ x \longleftrightarrow x = 1$   
 $\langle \text{proof} \rangle$

**lemma** *is-first-power* [*simp*]: *is-nth-power*  $1\ x$   
 $\langle \text{proof} \rangle$

**lemma** *is-first-power'* [*simp*]: *is-nth-power*  $(\text{Suc } 0)\ x$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-0* [*simp*]:  $n > 0 \implies \text{is-nth-power } n\ (0 :: 'a :: \text{semiring-1})$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-0-iff* [*simp*]: *is-nth-power*  $n\ (0 :: 'a :: \text{semiring-1}) \longleftrightarrow n > 0$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-1* [*simp*]: *is-nth-power*  $n\ 1$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-Suc-0* [*simp*]: *is-nth-power*  $n\ (\text{Suc } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-conv-multiplicity*:  
**fixes**  $x :: 'a :: \{\text{factorial-semiring, normalization-semidom-multiplicative}\}$   
**assumes**  $n > 0$   
**shows**  $\text{is-nth-power } n\ (\text{normalize } x) \longleftrightarrow (\forall p. \text{prime } p \longrightarrow n \text{ dvd multiplicity } p\ x)$   
 $\langle \text{proof} \rangle$

**lemma** *is-nth-power-conv-multiplicity-nat*:

**assumes**  $n > 0$

**shows**  $\text{is-nth-power } n \ (x :: \text{nat}) \longleftrightarrow (\forall p. \text{prime } p \longrightarrow n \text{ dvd multiplicity } p \ x)$

*<proof>*

**lemma** *is-nth-power-mult*:

**assumes**  $\text{is-nth-power } n \ a \ \text{is-nth-power } n \ b$

**shows**  $\text{is-nth-power } n \ (a * b :: 'a :: \text{comm-monoid-mult})$

*<proof>*

**lemma** *is-nth-power-mult-coprime-natD*:

**fixes**  $a \ b :: \text{nat}$

**assumes**  $\text{coprime } a \ b \ \text{is-nth-power } n \ (a * b) \ a > 0 \ b > 0$

**shows**  $\text{is-nth-power } n \ a \ \text{is-nth-power } n \ b$

*<proof>*

**lemma** *is-nth-power-mult-coprime-nat-iff*:

**fixes**  $a \ b :: \text{nat}$

**assumes**  $\text{coprime } a \ b$

**shows**  $\text{is-nth-power } n \ (a * b) \longleftrightarrow \text{is-nth-power } n \ a \ \wedge \text{is-nth-power } n \ b$

*<proof>*

**lemma** *is-nth-power-prime-power-nat-iff*:

**fixes**  $p :: \text{nat}$  **assumes**  $\text{prime } p$

**shows**  $\text{is-nth-power } n \ (p \wedge^k) \longleftrightarrow n \text{ dvd } k$

*<proof>*

**lemma** *is-nth-power-nth-power'*:

**assumes**  $n \text{ dvd } n'$

**shows**  $\text{is-nth-power } n \ (m \wedge^{n'})$

*<proof>*

**definition** *is-nth-power-nat* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$

**where** *[code-abbrev]*:  $\text{is-nth-power-nat} = \text{is-nth-power}$

**lemma** *is-nth-power-nat-code* *[code]*:

$\text{is-nth-power-nat } n \ m =$

$\text{if } n = 0 \text{ then } m = 1$

$\text{else if } m = 0 \text{ then } n > 0$

$\text{else if } n = 1 \text{ then } \text{True}$

$\text{else } (\exists k \in \{1..m\}. k \wedge^n = m)$

*<proof>*

**lemma** *is-nth-power-mult-cancel-left*:

**fixes**  $a \ b :: 'a :: \text{semiring-gcd}$

**assumes**  $\text{is-nth-power } n \ a \ a \neq 0$

**shows**  $\text{is-nth-power } n \ (a * b) \longleftrightarrow \text{is-nth-power } n \ b$

*<proof>*

**lemma** *is-nth-power-mult-cancel-right*:  
**fixes**  $a\ b :: 'a :: \text{semiring-gcd}$   
**assumes**  $\text{is-nth-power } n\ b\ b \neq 0$   
**shows**  $\text{is-nth-power } n\ (a * b) \longleftrightarrow \text{is-nth-power } n\ a$   
 $\langle \text{proof} \rangle$

## 10.2 The $n$ -root of a natural number

**definition**  $\text{nth-root-nat} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$  **where**  
 $\text{nth-root-nat } k\ n = (\text{if } k = 0 \text{ then } 0 \text{ else } \text{Max } \{m. m \wedge k \leq n\})$

**lemma** *zeroth-root-nat* [simp]:  $\text{nth-root-nat } 0\ n = 0$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-aux1*:  
**assumes**  $k > 0$   
**shows**  $\{m :: \text{nat}. m \wedge k \leq n\} \subseteq \{..n\}$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-aux2*:  
**assumes**  $k > 0$   
**shows**  $\text{finite } \{m :: \text{nat}. m \wedge k \leq n\} \ \{m :: \text{nat}. m \wedge k \leq n\} \neq \{\}$   
 $\langle \text{proof} \rangle$

**lemma**  
**assumes**  $k > 0$   
**shows**  $\text{nth-root-nat-power-le: } \text{nth-root-nat } k\ n \wedge k \leq n$   
**and**  $\text{nth-root-nat-ge: } x \wedge k \leq n \implies x \leq \text{nth-root-nat } k\ n$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-less*:  
**assumes**  $k > 0\ x \wedge k > n$   
**shows**  $\text{nth-root-nat } k\ n < x$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-unique*:  
**assumes**  $m \wedge k \leq n\ (m + 1) \wedge k > n$   
**shows**  $\text{nth-root-nat } k\ n = m$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-0* [simp]:  $\text{nth-root-nat } k\ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-1* [simp]:  $k > 0 \implies \text{nth-root-nat } k\ 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *nth-root-nat-Suc-0* [simp]:  $k > 0 \implies \text{nth-root-nat } k\ (\text{Suc } 0) = \text{Suc } 0$   
 $\langle \text{proof} \rangle$

**lemma** *first-root-nat* [simp]:  $\text{nth-root-nat } 1 \ n = n$   
 ⟨proof⟩

**lemma** *first-root-nat'* [simp]:  $\text{nth-root-nat } (\text{Suc } 0) \ n = n$   
 ⟨proof⟩

**lemma** *nth-root-nat-code-naive'*:  
 $\text{nth-root-nat } k \ n = (\text{if } k = 0 \text{ then } 0 \text{ else } \text{Max } (\text{Set.filter } (\lambda m. m \wedge k \leq n) \{..n\}))$   
 ⟨proof⟩

**function** *nth-root-nat-aux* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$  **where**  
 $\text{nth-root-nat-aux } m \ k \ \text{acc } n =$   
 $(\text{let } \text{acc}' = (k + 1) \wedge m$   
 $\text{in if } k \geq n \vee \text{acc}' > n \text{ then } k \text{ else } \text{nth-root-nat-aux } m \ (k+1) \ \text{acc}' \ n)$   
 ⟨proof⟩

**termination** ⟨proof⟩

**lemma** *nth-root-nat-aux-le*:  
**assumes**  $k \wedge m \leq n \ m > 0$   
**shows**  $\text{nth-root-nat-aux } m \ k \ (k \wedge m) \ n \wedge m \leq n$   
 ⟨proof⟩

**lemma** *nth-root-nat-aux-gt*:  
**assumes**  $m > 0$   
**shows**  $(\text{nth-root-nat-aux } m \ k \ (k \wedge m) \ n + 1) \wedge m > n$   
 ⟨proof⟩

**lemma** *nth-root-nat-aux-correct*:  
**assumes**  $k \wedge m \leq n \ m > 0$   
**shows**  $\text{nth-root-nat-aux } m \ k \ (k \wedge m) \ n = \text{nth-root-nat } m \ n$   
 ⟨proof⟩

**lemma** *nth-root-nat-naive-code* [code]:  
 $\text{nth-root-nat } m \ n = (\text{if } m = 0 \vee n = 0 \text{ then } 0 \text{ else if } m = 1 \vee n = 1 \text{ then } n \text{ else}$   
 $\text{nth-root-nat-aux } m \ 1 \ 1 \ n)$   
 ⟨proof⟩

**lemma** *nth-root-nat-nth-power* [simp]:  $k > 0 \implies \text{nth-root-nat } k \ (n \wedge k) = n$   
 ⟨proof⟩

**lemma** *nth-root-nat-nth-power'*:  
**assumes**  $k > 0 \ k \ \text{dvd } m$   
**shows**  $\text{nth-root-nat } k \ (n \wedge m) = n \wedge (m \ \text{div } k)$   
 ⟨proof⟩

**lemma** *nth-root-nat-mono*:  
**assumes**  $m \leq n$

**shows**  $\text{nth-root-nat } k \ m \leq \text{nth-root-nat } k \ n$   
 $\langle \text{proof} \rangle$

**end**

## 11 Polynomials, fractions and rings

**theory** *Polynomial-Factorial*

**imports**

*Complex-Main*

*Polynomial*

*Normalized-Fraction*

**begin**

### 11.1 Lifting elements into the field of fractions

**definition**  $\text{to-fract} :: 'a :: \text{idom} \Rightarrow 'a \text{ fract}$

**where**  $\text{to-fract } x = \text{Fract } x \ 1$

— FIXME: more idiomatic name, abbreviation

**lemma**  $\text{to-fract-0} \ [\text{simp}]: \text{to-fract } 0 = 0$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-1} \ [\text{simp}]: \text{to-fract } 1 = 1$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-add} \ [\text{simp}]: \text{to-fract } (x + y) = \text{to-fract } x + \text{to-fract } y$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-diff} \ [\text{simp}]: \text{to-fract } (x - y) = \text{to-fract } x - \text{to-fract } y$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-uminus} \ [\text{simp}]: \text{to-fract } (-x) = -\text{to-fract } x$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-mult} \ [\text{simp}]: \text{to-fract } (x * y) = \text{to-fract } x * \text{to-fract } y$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-eq-iff} \ [\text{simp}]: \text{to-fract } x = \text{to-fract } y \longleftrightarrow x = y$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-eq-0-iff} \ [\text{simp}]: \text{to-fract } x = 0 \longleftrightarrow x = 0$

$\langle \text{proof} \rangle$

**lemma**  $\text{to-fract-quot-of-fract}:$

**assumes**  $\text{snd } (\text{quot-of-fract } x) = 1$

**shows**  $\text{to-fract } (\text{fst } (\text{quot-of-fract } x)) = x$

$\langle \text{proof} \rangle$

**lemma** *Fract-conv-to-fract*:  $\text{Fract } a \ b = \text{to-fract } a \ / \ \text{to-fract } b$   
 $\langle \text{proof} \rangle$

**lemma** *quot-of-fract-to-fract* [simp]:  $\text{quot-of-fract } (\text{to-fract } x) = (x, 1)$   
 $\langle \text{proof} \rangle$

**lemma** *snd-quot-of-fract-to-fract* [simp]:  $\text{snd } (\text{quot-of-fract } (\text{to-fract } x)) = 1$   
 $\langle \text{proof} \rangle$

## 11.2 Lifting polynomial coefficients to the field of fractions

**abbreviation** (*input*) *fract-poly* ::  $\langle 'a :: \text{idom } \text{poly} \Rightarrow 'a \text{ fract poly} \rangle$   
**where** *fract-poly*  $\equiv \text{map-poly } \text{to-fract}$

**abbreviation** (*input*) *unfract-poly* ::  $\langle 'a :: \{\text{ring-gcd}, \text{semiring-gcd-mult-normalize}, \text{idom-divide}\} \text{ fract poly} \Rightarrow 'a \text{ poly} \rangle$   
**where** *unfract-poly*  $\equiv \text{map-poly } (\text{fst} \circ \text{quot-of-fract})$

**lemma** *fract-poly-smult* [simp]:  $\text{fract-poly } (\text{smult } c \ p) = \text{smult } (\text{to-fract } c) \ (\text{fract-poly } p)$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-0* [simp]:  $\text{fract-poly } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-1* [simp]:  $\text{fract-poly } 1 = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-add* [simp]:  
 $\text{fract-poly } (p + q) = \text{fract-poly } p + \text{fract-poly } q$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-diff* [simp]:  
 $\text{fract-poly } (p - q) = \text{fract-poly } p - \text{fract-poly } q$   
 $\langle \text{proof} \rangle$

**lemma** *to-fract-sum* [simp]:  $\text{to-fract } (\text{sum } f \ A) = \text{sum } (\lambda x. \text{to-fract } (f \ x)) \ A$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-mult* [simp]:  
 $\text{fract-poly } (p * q) = \text{fract-poly } p * \text{fract-poly } q$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-eq-iff* [simp]:  $\text{fract-poly } p = \text{fract-poly } q \longleftrightarrow p = q$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-eq-0-iff* [simp]:  $\text{fract-poly } p = 0 \longleftrightarrow p = 0$   
 $\langle \text{proof} \rangle$



**lemma** *fract-poly-dvd*:  $p \text{ dvd } q \implies \text{fract-poly } p \text{ dvd } \text{fract-poly } q$   
 <proof>

**lemma** *prod-mset-fract-poly*:  
 $(\prod_{x \in \#A. \text{map-poly to-fract } (f \ x)}) = \text{fract-poly } (\text{prod-mset } (\text{image-mset } f \ A))$   
 <proof>

**lemma** *is-unit-fract-poly-iff*:  
 $p \text{ dvd } 1 \iff \text{fract-poly } p \text{ dvd } 1 \wedge \text{content } p = 1$   
 <proof>

**lemma** *fract-poly-is-unit*:  $p \text{ dvd } 1 \implies \text{fract-poly } p \text{ dvd } 1$   
 <proof>

**lemma** *fract-poly-smult-eqE*:  
**fixes**  $c :: 'a :: \{\text{idom-divide}, \text{ring-gcd}, \text{semiring-gcd-mult-normalize}\}$  *fract*  
**assumes**  $\text{fract-poly } p = \text{smult } c \ (\text{fract-poly } q)$   
**obtains**  $a \ b$   
**where**  $c = \text{to-fract } b \ / \ \text{to-fract } a \ \text{smult } a \ p = \text{smult } b \ q \ \text{coprime } a \ b \ \text{normalize}$   
 $a = a$   
 <proof>

### 11.3 Fractional content

**abbreviation** *(input) Lcm-coeff-denoms*  
 $:: 'a :: \{\text{semiring-Gcd}, \text{idom-divide}, \text{ring-gcd}, \text{semiring-gcd-mult-normalize}\}$  *fract*  
 $\text{poly} \Rightarrow 'a$   
**where**  $\text{Lcm-coeff-denoms } p \equiv \text{Lcm } (\text{snd } \text{'quot-of-fract' set (coeffs } p))$

**definition** *fract-content* ::  
 $'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{fract poly} \Rightarrow 'a \text{ fract}$  **where**  
 $\text{fract-content } p =$   
 $(\text{let } d = \text{Lcm-coeff-denoms } p \text{ in } \text{Fract } (\text{content } (\text{unfract-poly } (\text{smult } (\text{to-fract } d) \ p))) \ d)$

**definition** *primitive-part-fract* ::  
 $'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{fract poly} \Rightarrow 'a \text{ poly}$  **where**  
 $\text{primitive-part-fract } p =$   
 $\text{primitive-part } (\text{unfract-poly } (\text{smult } (\text{to-fract } (\text{Lcm-coeff-denoms } p)) \ p))$

**lemma** *primitive-part-fract-0* [simp]:  $\text{primitive-part-fract } 0 = 0$   
 <proof>

**lemma** *fract-content-eq-0-iff* [simp]:  
 $\text{fract-content } p = 0 \iff p = 0$   
 <proof>

**lemma** *content-primitive-part-fract* [simp]:  
**fixes**  $p :: 'a :: \{\text{semiring-gcd-mult-normalize,}$   
 $\text{factorial-semiring, ring-gcd, semiring-Gcd, idom-divide}\}$  *fract poly*  
**shows**  $p \neq 0 \implies \text{content} (\text{primitive-part-fract } p) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *content-times-primitive-part-fract*:  
 $\text{smult} (\text{fract-content } p) (\text{fract-poly} (\text{primitive-part-fract } p)) = p$   
 $\langle \text{proof} \rangle$

**lemma** *fract-content-fract-poly* [simp]:  $\text{fract-content} (\text{fract-poly } p) = \text{to-fract} (\text{content } p)$   
 $\langle \text{proof} \rangle$

**lemma** *content-decompose-fract*:  
**fixes**  $p :: 'a :: \{\text{factorial-semiring, semiring-Gcd, ring-gcd, idom-divide,}$   
 $\text{semiring-gcd-mult-normalize}\}$  *fract poly*  
**obtains**  $c \ p'$  **where**  $p = \text{smult } c (\text{map-poly to-fract } p')$   $\text{content } p' = 1$   
 $\langle \text{proof} \rangle$

**lemma** *fract-poly-dvdD*:  
**fixes**  $p :: 'a :: \{\text{factorial-semiring, semiring-Gcd, ring-gcd, idom-divide,}$   
 $\text{semiring-gcd-mult-normalize}\}$  *poly*  
**assumes**  $\text{fract-poly } p \text{ dvd } \text{fract-poly } q$   $\text{content } p = 1$   
**shows**  $p \text{ dvd } q$   
 $\langle \text{proof} \rangle$

## 11.4 Polynomials over a field are a Euclidean ring

**context**  
**begin**

**interpretation** *field-poly*:  
 $\text{normalization-euclidean-semiring-multiplicative}$  **where**  $\text{zero} = 0 :: 'a :: \text{field poly}$   
**and**  $\text{one} = 1$  **and**  $\text{plus} = \text{plus}$  **and**  $\text{minus} = \text{minus}$   
**and**  $\text{times} = \text{times}$   
**and**  $\text{normalize} = \lambda p. \text{smult} (\text{inverse} (\text{lead-coeff } p)) \ p$   
**and**  $\text{unit-factor} = \lambda p. [\text{lead-coeff } p]$   
**and**  $\text{euclidean-size} = \lambda p. \text{if } p = 0 \text{ then } 0 \text{ else } 2 \wedge \text{degree } p$   
**and**  $\text{divide} = \text{divide}$  **and**  $\text{modulo} = \text{modulo}$   
**rewrites**  $\text{dvd.dvd} (\text{times} :: 'a \text{ poly} \Rightarrow -) = \text{Rings.dvd}$   
**and**  $\text{comm-monoid-mult.prod-mset times } 1 = \text{prod-mset}$   
**and**  $\text{comm-semiring-1.irreducible times } 1 \ 0 = \text{irreducible}$   
**and**  $\text{comm-semiring-1.prime-elem times } 1 \ 0 = \text{prime-elem}$   
 $\langle \text{proof} \rangle$

**lemma** *field-poly-irreducible-imp-prime*:  
 $\text{prime-elem } p$  **if**  $\text{irreducible } p$  **for**  $p :: 'a :: \text{field poly}$   
 $\langle \text{proof} \rangle$

**lemma** *field-poly-prod-mset-prime-factorization*:  
 $\text{prod-mset } (\text{field-poly.prime-factorization } p) = \text{smult } (\text{inverse } (\text{lead-coeff } p)) \ p$   
**if**  $p \neq 0$  **for**  $p :: 'a :: \text{field poly}$   
 $\langle \text{proof} \rangle$

**lemma** *field-poly-in-prime-factorization-imp-prime*:  
 $\text{prime-elem } p$  **if**  $p \in \# \text{ field-poly.prime-factorization } x$   
**for**  $p :: 'a :: \text{field poly}$   
 $\langle \text{proof} \rangle$

## 11.5 Primality and irreducibility in polynomial rings

**lemma** *nonconst-poly-irreducible-iff*:  
**fixes**  $p :: 'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{poly}$   
**assumes**  $\text{degree } p \neq 0$   
**shows**  $\text{irreducible } p \longleftrightarrow \text{irreducible } (\text{fract-poly } p) \wedge \text{content } p = 1$   
 $\langle \text{proof} \rangle$

**lemma** *irreducible-imp-prime-poly*:  
**fixes**  $p :: 'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{poly}$   
**assumes**  $\text{irreducible } p$   
**shows**  $\text{prime-elem } p$   
 $\langle \text{proof} \rangle$

**lemma** *degree-primitive-part-fract [simp]*:  
 $\text{degree } (\text{primitive-part-fract } p) = \text{degree } p$   
 $\langle \text{proof} \rangle$

**lemma** *irreducible-primitive-part-fract*:  
**fixes**  $p :: 'a :: \{\text{idom-divide}, \text{ring-gcd}, \text{factorial-semiring}, \text{semiring-Gcd}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{fract poly}$   
**assumes**  $\text{irreducible } p$   
**shows**  $\text{irreducible } (\text{primitive-part-fract } p)$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-primitive-part-fract*:  
**fixes**  $p :: 'a :: \{\text{idom-divide}, \text{ring-gcd}, \text{factorial-semiring}, \text{semiring-Gcd}, \text{semiring-gcd-mult-normalize}\}$   
 $\text{fract poly}$   
**shows**  $\text{irreducible } p \implies \text{prime-elem } (\text{primitive-part-fract } p)$   
 $\langle \text{proof} \rangle$

**lemma** *irreducible-linear-field-poly*:  
**fixes**  $a \ b :: 'a :: \text{field}$   
**assumes**  $b \neq 0$   
**shows**  $\text{irreducible } [:a, b:]$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-linear-field-poly*:

$(b :: 'a :: \text{field}) \neq 0 \implies \text{prime-elem } [:a,b:]$   
 $\langle \text{proof} \rangle$

**lemma** *irreducible-linear-poly*:

**fixes**  $a\ b :: 'a :: \{\text{idom-divide}, \text{ring-gcd}, \text{factorial-semiring}, \text{semiring-Gcd}, \text{semiring-gcd-mult-normalize}\}$   
**shows**  $b \neq 0 \implies \text{coprime } a\ b \implies \text{irreducible } [:a,b:]$   
 $\langle \text{proof} \rangle$

**lemma** *prime-elem-linear-poly*:

**fixes**  $a\ b :: 'a :: \{\text{idom-divide}, \text{ring-gcd}, \text{factorial-semiring}, \text{semiring-Gcd}, \text{semiring-gcd-mult-normalize}\}$   
**shows**  $b \neq 0 \implies \text{coprime } a\ b \implies \text{prime-elem } [:a,b:]$   
 $\langle \text{proof} \rangle$

## 11.6 Prime factorisation of polynomials

**lemma** *poly-prime-factorization-exists-content-1*:

**fixes**  $p :: 'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
*poly*  
**assumes**  $p \neq 0$  *content*  $p = 1$   
**shows**  $\exists A. (\forall p. p \in \# A \longrightarrow \text{prime-elem } p) \wedge \text{prod-mset } A = \text{normalize } p$   
 $\langle \text{proof} \rangle$

**lemma** *poly-prime-factorization-exists*:

**fixes**  $p :: 'a :: \{\text{factorial-semiring}, \text{semiring-Gcd}, \text{ring-gcd}, \text{idom-divide}, \text{semiring-gcd-mult-normalize}\}$   
*poly*  
**assumes**  $p \neq 0$   
**shows**  $\exists A. (\forall p. p \in \# A \longrightarrow \text{prime-elem } p) \wedge \text{normalize } (\text{prod-mset } A) = \text{normalize } p$   
 $\langle \text{proof} \rangle$

**end**

## 11.7 Typeclass instances

**instance** *poly* ::  $(\{\text{factorial-ring-gcd}, \text{semiring-gcd-mult-normalize}\}) \text{ factorial-semiring}$   
 $\langle \text{proof} \rangle$

**instantiation** *poly* ::  $(\{\text{factorial-ring-gcd}, \text{semiring-gcd-mult-normalize}\}) \text{ factorial-ring-gcd}$   
**begin**

**definition** *gcd-poly* ::  $'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly}$  **where**  
 $\text{[code del]: gcd-poly = gcd-factorial}$

**definition** *lcm-poly* ::  $'a \text{ poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly}$  **where**  
 $\text{[code del]: lcm-poly = lcm-factorial}$

**definition** *Gcd-poly* ::  $'a \text{ poly set} \Rightarrow 'a \text{ poly}$  **where**  
 $\text{[code del]: Gcd-poly = Gcd-factorial}$

**definition** *Lcm-poly* :: 'a poly set  $\Rightarrow$  'a poly **where**

[code del]: *Lcm-poly* = *Lcm-factorial*

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

**end**

**instance** *poly* :: (*factorial-ring-gcd*, *semiring-gcd-mult-normalize*) *semiring-gcd-mult-normalize*  
 $\langle proof \rangle$

**instance** *poly* :: (*field*, *factorial-ring-gcd*, *semiring-gcd-mult-normalize*)  
*normalization-euclidean-semiring*  $\langle proof \rangle$

**instance** *poly* :: (*field*, *normalization-euclidean-semiring*, *factorial-ring-gcd*,  
*semiring-gcd-mult-normalize*) *euclidean-ring-gcd*  
 $\langle proof \rangle$

**instance** *poly* :: (*field*, *normalization-euclidean-semiring*, *factorial-ring-gcd*,  
*semiring-gcd-mult-normalize*) *factorial-semiring-multiplicative*  
 $\langle proof \rangle$

## 11.8 Polynomial GCD

**lemma** *gcd-poly-decompose*:

**fixes** *p q* :: 'a :: {*factorial-ring-gcd*, *semiring-gcd-mult-normalize*} *poly*  
**shows** *gcd p q* =  
*smult (gcd (content p) (content q)) (gcd (primitive-part p) (primitive-part q))*  
 $\langle proof \rangle$

**lemma** *gcd-poly-pseudo-mod*:

**fixes** *p q* :: 'a :: {*factorial-ring-gcd*, *semiring-gcd-mult-normalize*} *poly*  
**assumes** *nz*: *q*  $\neq 0$  **and** *prim*: *content p* = 1 *content q* = 1  
**shows** *gcd p q* = *gcd q (primitive-part (pseudo-mod p q))*  
 $\langle proof \rangle$

**lemma** *degree-pseudo-mod-less*:

**assumes** *q*  $\neq 0$  *pseudo-mod p q*  $\neq 0$   
**shows** *degree (pseudo-mod p q)* < *degree q*  
 $\langle proof \rangle$

**function** *gcd-poly-code-aux* :: 'a :: *factorial-ring-gcd poly*  $\Rightarrow$  'a *poly*  $\Rightarrow$  'a *poly*  
**where**

*gcd-poly-code-aux p q* =  
*(if q = 0 then normalize p else gcd-poly-code-aux q (primitive-part (pseudo-mod p q)))*  
 $\langle proof \rangle$

**termination**

*<proof>*

**declare** *gcd-poly-code-aux.simps* [*simp del*]

**lemma** *gcd-poly-code-aux-correct*:

**assumes** *content p = 1 q = 0*  $\vee$  *content q = 1*

**shows** *gcd-poly-code-aux p q = gcd p q*

*<proof>*

**definition** *gcd-poly-code*

$:: 'a :: \text{factorial-ring-gcd poly} \Rightarrow 'a \text{ poly} \Rightarrow 'a \text{ poly}$

**where** *gcd-poly-code p q =*

*(if p = 0 then normalize q else if q = 0 then normalize p else*

*smult (gcd (content p) (content q))*

*(gcd-poly-code-aux (primitive-part p) (primitive-part q)))*

**lemma** *gcd-poly-code [code]: gcd p q = gcd-poly-code p q*

*<proof>*

**lemma** *lcm-poly-code [code]:*

**fixes** *p q :: 'a :: {factorial-ring-gcd, semiring-gcd-mult-normalize} poly*

**shows** *lcm p q = normalize (p \* q div gcd p q)*

*<proof>*

**lemmas** *Gcd-poly-set-eq-fold [code] =*

*Gcd-set-eq-fold [where ?'a = 'a :: {factorial-ring-gcd, semiring-gcd-mult-normalize} poly]*

**lemmas** *Lcm-poly-set-eq-fold [code] =*

*Lcm-set-eq-fold [where ?'a = 'a :: {factorial-ring-gcd, semiring-gcd-mult-normalize} poly]*

**end**

## 12 Squarefreeness

**theory** *Squarefree*

**imports** *Primes*

**begin**

**definition** *squarefree :: 'a :: comm-monoid-mult  $\Rightarrow$  bool* **where**

*squarefree n  $\longleftrightarrow (\forall x. x^2 \text{ dvd } n \longrightarrow x \text{ dvd } 1)$*

**lemma** *squarefreeI:  $(\bigwedge x. x^2 \text{ dvd } n \Longrightarrow x \text{ dvd } 1) \Longrightarrow \text{squarefree } n$*

*<proof>*

**lemma** *squarefreeD: squarefree n  $\Longrightarrow x^2 \text{ dvd } n \Longrightarrow x \text{ dvd } 1$*

*<proof>*

**lemma** *not-squarefreeI*:  $x \wedge 2 \text{ dvd } n \implies \neg x \text{ dvd } 1 \implies \neg \text{squarefree } n$   
*<proof>*

**lemma** *not-squarefreeE* [case-names square-dvd]:  
 $\neg \text{squarefree } n \implies (\bigwedge x. x \wedge 2 \text{ dvd } n \implies \neg x \text{ dvd } 1 \implies P) \implies P$   
*<proof>*

**lemma** *not-squarefree-0* [simp]:  $\neg \text{squarefree } (0 :: 'a :: \text{comm-semiring-1})$   
*<proof>*

**lemma** *squarefree-factorial-semiring*:  
assumes  $n \neq 0$   
shows  $\text{squarefree } (n :: 'a :: \text{factorial-semiring}) \longleftrightarrow (\forall p. \text{prime } p \longrightarrow \neg p \wedge 2 \text{ dvd } n)$   
*<proof>*

**lemma** *squarefree-factorial-semiring'*:  
assumes  $n \neq 0$   
shows  $\text{squarefree } (n :: 'a :: \text{factorial-semiring}) \longleftrightarrow (\forall p \in \text{prime-factors } n. \text{multiplicity } p \ n = 1)$   
*<proof>*

**lemma** *squarefree-factorial-semiring''*:  
assumes  $n \neq 0$   
shows  $\text{squarefree } (n :: 'a :: \text{factorial-semiring}) \longleftrightarrow (\forall p. \text{prime } p \longrightarrow \text{multiplicity } p \ n \leq 1)$   
*<proof>*

**lemma** *squarefree-unit* [simp]:  $\text{is-unit } n \implies \text{squarefree } n$   
*<proof>*

**lemma** *squarefree-1* [simp]:  $\text{squarefree } (1 :: 'a :: \text{algebraic-semidom})$   
*<proof>*

**lemma** *squarefree-minus* [simp]:  $\text{squarefree } (-n :: 'a :: \text{comm-ring-1}) \longleftrightarrow \text{squarefree } n$   
*<proof>*

**lemma** *squarefree-mono*:  $a \text{ dvd } b \implies \text{squarefree } b \implies \text{squarefree } a$   
*<proof>*

**lemma** *squarefree-multD*:  
assumes  $\text{squarefree } (a * b)$   
shows  $\text{squarefree } a \ \text{squarefree } b$   
*<proof>*

**lemma** *squarefree-prime-elem*:

**assumes** *prime-elem* ( $p :: 'a :: \text{factorial-semiring}$ )  
**shows** *squarefree*  $p$   
 <proof>

**lemma** *squarefree-prime*:  
**assumes** *prime* ( $p :: 'a :: \text{factorial-semiring}$ )  
**shows** *squarefree*  $p$   
 <proof>

**lemma** *squarefree-mult-coprime*:  
**fixes**  $a\ b :: 'a :: \text{factorial-semiring-gcd}$   
**assumes** *coprime*  $a\ b$  *squarefree*  $a$  *squarefree*  $b$   
**shows** *squarefree* ( $a * b$ )  
 <proof>

**lemma** *squarefree-prod-coprime*:  
**fixes**  $f :: 'a \Rightarrow 'b :: \text{factorial-semiring-gcd}$   
**assumes**  $\bigwedge a\ b. a \in A \implies b \in A \implies a \neq b \implies \text{coprime } (f\ a)\ (f\ b)$   
**assumes**  $\bigwedge a. a \in A \implies \text{squarefree } (f\ a)$   
**shows** *squarefree* ( $\text{prod } f\ A$ )  
 <proof>

**lemma** *squarefree-powerD*:  $m > 0 \implies \text{squarefree } (n \wedge m) \implies \text{squarefree } n$   
 <proof>

**lemma** *squarefree-power-iff*:  
 $\text{squarefree } (n \wedge m) \longleftrightarrow m = 0 \vee \text{is-unit } n \vee (\text{squarefree } n \wedge m = 1)$   
 <proof>

**definition** *squarefree-nat* ::  $\text{nat} \Rightarrow \text{bool}$  **where**  
 [code-abbrev]: *squarefree-nat* = *squarefree*

**lemma** *squarefree-nat-code-naive* [code]:  
 $\text{squarefree-nat } n \longleftrightarrow n \neq 0 \wedge (\forall k \in \{2..n\}. \neg k \wedge 2 \text{ dvd } n)$   
 <proof>

**definition** *square-part* ::  $'a :: \text{factorial-semiring} \Rightarrow 'a$  **where**  
 $\text{square-part } n = (\text{if } n = 0 \text{ then } 0 \text{ else}$   
 $\text{normalize } (\prod_{p \in \text{prime-factors } n}. p \wedge (\text{multiplicity } p\ n \text{ div } 2)))$

**lemma** *square-part-nonzero*:  
 $n \neq 0 \implies \text{square-part } n = \text{normalize } (\prod_{p \in \text{prime-factors } n}. p \wedge (\text{multiplicity } p\ n \text{ div } 2))$   
 <proof>

**lemma** *square-part-0* [simp]: *square-part*  $0 = 0$   
 <proof>



**lemma** *square-part-unit* [simp]:  $\text{is-unit } x \implies \text{square-part } x = 1$   
 ⟨proof⟩

**lemma** *square-part-1* [simp]:  $\text{square-part } 1 = 1$   
 ⟨proof⟩

**lemma** *square-part-0-iff* [simp]:  $\text{square-part } n = 0 \iff n = 0$   
 ⟨proof⟩

**lemma** *normalize-uminus* [simp]:  
 $\text{normalize } (-x :: 'a :: \{\text{normalization-semidom, comm-ring-1}\}) = \text{normalize } x$   
 ⟨proof⟩

**lemma** *multiplicity-uminus-right* [simp]:  
 $\text{multiplicity } (x :: 'a :: \{\text{factorial-semiring, comm-ring-1}\}) (-y) = \text{multiplicity } x y$   
 ⟨proof⟩

**lemma** *multiplicity-uminus-left* [simp]:  
 $\text{multiplicity } (-x :: 'a :: \{\text{factorial-semiring, comm-ring-1}\}) y = \text{multiplicity } x y$   
 ⟨proof⟩

**lemma** *prime-factorization-uminus* [simp]:  
 $\text{prime-factorization } (-x :: 'a :: \{\text{factorial-semiring, comm-ring-1}\}) = \text{prime-factorization } x$   
 ⟨proof⟩

**lemma** *square-part-uminus* [simp]:  
 $\text{square-part } (-x :: 'a :: \{\text{factorial-semiring, comm-ring-1}\}) = \text{square-part } x$   
 ⟨proof⟩

**lemma** *prime-multiplicity-square-part*:  
 assumes  $\text{prime } p$   
 shows  $\text{multiplicity } p (\text{square-part } n) = \text{multiplicity } p n \text{ div } 2$   
 ⟨proof⟩

**lemma** *square-part-square-dvd* [simp, intro]:  $\text{square-part } n \wedge^2 \text{ dvd } n$   
 ⟨proof⟩

**lemma** *prime-multiplicity-le-imp-dvd*:  
 assumes  $x \neq 0 \ y \neq 0$   
 shows  $x \text{ dvd } y \iff (\forall p. \text{prime } p \longrightarrow \text{multiplicity } p x \leq \text{multiplicity } p y)$   
 ⟨proof⟩

**lemma** *dvd-square-part-iff*:  $x \text{ dvd square-part } n \iff x \wedge^2 \text{ dvd } n$   
 ⟨proof⟩

**definition** *squarefree-part* ::  $'a :: \text{factorial-semiring} \Rightarrow 'a$  **where**

$\text{squarefree-part } n = (\text{if } n = 0 \text{ then } 1 \text{ else } n \text{ div square-part } n \wedge 2)$

**lemma** *squarefree-part-0* [simp]:  $\text{squarefree-part } 0 = 1$   
 <proof>

**lemma** *squarefree-part-unit* [simp]:  $\text{is-unit } n \implies \text{squarefree-part } n = n$   
 <proof>

**lemma** *squarefree-part-1* [simp]:  $\text{squarefree-part } 1 = 1$   
 <proof>

**lemma** *squarefree-decompose*:  $n = \text{squarefree-part } n * \text{square-part } n \wedge 2$   
 <proof>

**lemma** *squarefree-part-uminus* [simp]:  
 assumes  $x \neq 0$   
 shows  $\text{squarefree-part } (-x :: 'a :: \{\text{factorial-semiring, comm-ring-1}\}) = -\text{squarefree-part } x$   
 <proof>

**lemma** *squarefree-part-nonzero* [simp]:  $\text{squarefree-part } n \neq 0$   
 <proof>

**lemma** *prime-multiplicity-squarefree-part*:  
 assumes  $\text{prime } p$   
 shows  $\text{multiplicity } p (\text{squarefree-part } n) = \text{multiplicity } p \ n \bmod 2$   
 <proof>

**lemma** *prime-multiplicity-squarefree-part-le-Suc-0* [intro]:  
 assumes  $\text{prime } p$   
 shows  $\text{multiplicity } p (\text{squarefree-part } n) \leq \text{Suc } 0$   
 <proof>

**lemma** *squarefree-squarefree-part* [simp, intro]:  $\text{squarefree } (\text{squarefree-part } n)$   
 <proof>

**lemma** *squarefree-decomposition-unique*:  
 assumes  $\text{square-part } m = \text{square-part } n$   
 assumes  $\text{squarefree-part } m = \text{squarefree-part } n$   
 shows  $m = n$   
 <proof>

**lemma** *normalize-square-part* [simp]:  $\text{normalize } (\text{square-part } x) = \text{square-part } x$   
 <proof>

**lemma** *square-part-even-power'*:  $\text{square-part } (x \wedge (2 * n)) = \text{normalize } (x \wedge n)$   
 <proof>

**lemma** *square-part-even-power*:  $\text{even } n \implies \text{square-part } (x \wedge n) = \text{normalize } (x \wedge$

(*n div 2*)  
 ⟨*proof*⟩

**lemma** *square-part-odd-power'*: *square-part* ( $x \wedge (\text{Suc } (2 * n))$ ) = *normalize* ( $x \wedge n * \text{square-part } x$ )  
 ⟨*proof*⟩

**lemma** *square-part-odd-power*:  
*odd n*  $\implies$  *square-part* ( $x \wedge n$ ) = *normalize* ( $x \wedge (n \text{ div } 2) * \text{square-part } x$ )  
 ⟨*proof*⟩

**end**

## 13 Pieces of computational Algebra

**theory** *Computational-Algebra*

**imports**

*Euclidean-Algorithm*  
*Factorial-Ring*  
*Formal-Laurent-Series*  
*Fraction-Field*  
*Fundamental-Theorem-Algebra*  
*Group-Closure*  
*Normalized-Fraction*  
*Nth-Powers*  
*Polynomial-FPS*  
*Polynomial*  
*Polynomial-Factorial*  
*Primes*  
*Squarefree*

**begin**

**end**

**theory** *Field-as-Ring*

**imports**

*Complex-Main*  
*Euclidean-Algorithm*

**begin**

**context** *field*

**begin**

**subclass** *idom-divide* ⟨*proof*⟩

**definition** *normalize-field* :: '*a*  $\Rightarrow$  '*a*

**where** [*simp*]: *normalize-field* *x* = (if *x* = 0 then 0 else 1)

**definition** *unit-factor-field* :: '*a*  $\Rightarrow$  '*a*

```

    where [simp]: unit-factor-field x = x
definition euclidean-size-field :: 'a  $\Rightarrow$  nat
    where [simp]: euclidean-size-field x = (if x = 0 then 0 else 1)
definition mod-field :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
    where [simp]: mod-field x y = (if y = 0 then x else 0)

end

instantiation real ::
  {unique-euclidean-ring, normalization-euclidean-semiring, normalization-semidom-multiplicative}
begin

definition [simp]: normalize-real = (normalize-field :: real  $\Rightarrow$  -)
definition [simp]: unit-factor-real = (unit-factor-field :: real  $\Rightarrow$  -)
definition [simp]: modulo-real = (mod-field :: real  $\Rightarrow$  -)
definition [simp]: euclidean-size-real = (euclidean-size-field :: real  $\Rightarrow$  -)
definition [simp]: division-segment (x :: real) = 1

instance
  ⟨proof⟩

end

instantiation real :: euclidean-ring-gcd
begin

definition gcd-real :: real  $\Rightarrow$  real  $\Rightarrow$  real where
  gcd-real = Euclidean-Algorithm.gcd
definition lcm-real :: real  $\Rightarrow$  real  $\Rightarrow$  real where
  lcm-real = Euclidean-Algorithm.lcm
definition Gcd-real :: real set  $\Rightarrow$  real where
  Gcd-real = Euclidean-Algorithm.Gcd
definition Lcm-real :: real set  $\Rightarrow$  real where
  Lcm-real = Euclidean-Algorithm.Lcm

instance ⟨proof⟩

end

instance real :: field-gcd ⟨proof⟩

instantiation rat ::
  {unique-euclidean-ring, normalization-euclidean-semiring, normalization-semidom-multiplicative}
begin

definition [simp]: normalize-rat = (normalize-field :: rat  $\Rightarrow$  -)
definition [simp]: unit-factor-rat = (unit-factor-field :: rat  $\Rightarrow$  -)
definition [simp]: modulo-rat = (mod-field :: rat  $\Rightarrow$  -)

```

```

definition [simp]: euclidean-size-rat = (euclidean-size-field :: rat  $\Rightarrow$  -)
definition [simp]: division-segment (x :: rat) = 1

instance
  ⟨proof⟩

end

instantiation rat :: euclidean-ring-gcd
begin

definition gcd-rat :: rat  $\Rightarrow$  rat  $\Rightarrow$  rat where
  gcd-rat = Euclidean-Algorithm.gcd
definition lcm-rat :: rat  $\Rightarrow$  rat  $\Rightarrow$  rat where
  lcm-rat = Euclidean-Algorithm.lcm
definition Gcd-rat :: rat set  $\Rightarrow$  rat where
  Gcd-rat = Euclidean-Algorithm.Gcd
definition Lcm-rat :: rat set  $\Rightarrow$  rat where
  Lcm-rat = Euclidean-Algorithm.Lcm

instance ⟨proof⟩

end

instance rat :: field-gcd ⟨proof⟩

instantiation complex ::
  {unique-euclidean-ring, normalization-euclidean-semiring, normalization-semidom-multiplicative}
begin

definition [simp]: normalize-complex = (normalize-field :: complex  $\Rightarrow$  -)
definition [simp]: unit-factor-complex = (unit-factor-field :: complex  $\Rightarrow$  -)
definition [simp]: modulo-complex = (mod-field :: complex  $\Rightarrow$  -)
definition [simp]: euclidean-size-complex = (euclidean-size-field :: complex  $\Rightarrow$  -)
definition [simp]: division-segment (x :: complex) = 1

instance
  ⟨proof⟩

end

instantiation complex :: euclidean-ring-gcd
begin

definition gcd-complex :: complex  $\Rightarrow$  complex  $\Rightarrow$  complex where
  gcd-complex = Euclidean-Algorithm.gcd
definition lcm-complex :: complex  $\Rightarrow$  complex  $\Rightarrow$  complex where
  lcm-complex = Euclidean-Algorithm.lcm

```

**definition** *Gcd-complex* :: *complex set*  $\Rightarrow$  *complex* **where**

*Gcd-complex* = *Euclidean-Algorithm.Gcd*

**definition** *Lcm-complex* :: *complex set*  $\Rightarrow$  *complex* **where**

*Lcm-complex* = *Euclidean-Algorithm.Lcm*

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

**end**

**instance** *complex* :: *field-gcd*  $\langle$ *proof* $\rangle$

**end**

## 14 Computation checks

**theory** *Computation-Checks*

**imports** *Primes Polynomial-Factorial HOL-Library.Discrete-Functions HOL-Library.Code-Target-Numeral*

**begin**

*floor-sqrt 16476148165462159 = 128359449*

*prime 97*

*prime 97*

*prime 9973*

*prime 9973*

*Gcd*  $\{[:1, 2, 3:], [:2, 3, 4:]\} = 1$

*Lcm*  $\{[:1, 2, 3:], [:2, 3, 4:]\} = [[:2:], [:7:], [:16:], [:17:], [:12:]]$

**end**

## References

- [1] K. J. Nowak. Some elementary proofs of Puiseuxs theorems. *Univ. Iagel. Acta Math*, 38:279–282, 2000.