

Formal Puiseux Series

Manuel Eberl

December 14, 2021

Abstract

Formal Puiseux series are generalisations of formal power series and formal Laurent series that also allow for fractional exponents. They have the following general form:

$$\sum_{i=N}^{\infty} a_{i/d} X^{i/d}$$

where N is an integer and d is a positive integer.

This entry defines these series including their basic algebraic properties. Furthermore, it proves the Newton–Puiseux Theorem, namely that the Puiseux series over an algebraically closed field of characteristic 0 are also algebraically closed.

Contents

1	Auxiliary material	3
1.1	Facts about polynomials	3
1.2	A typeclass for algebraically closed fields	4
1.3	Facts about Laurent series	6
2	Hensel's lemma for formal power series	9
3	Formal Puiseux Series	11
3.1	Auxiliary facts and definitions	12
3.2	Definition	12
3.3	Basic algebraic typeclass instances	14
3.4	The substitution $X \mapsto X^r$	16
3.5	Multiplication and ring properties	17
3.6	Constant Puiseux series and the series X	18
3.7	More algebraic typeclass instances	19
3.8	Valuation	20
3.9	Powers of X and shifting	22
3.10	The n -th root of a Puiseux series	24
3.11	Algebraic closedness	24
3.12	Metric and topology	25

1 Auxiliary material

1.1 Facts about polynomials

theory *Puiseux-Polynomial-Library*

imports *HOL-Computational-Algebra.Computational-Algebra Polynomial-Interpolation.Ring-Hom-Poly*
begin

lemma *poly-sum-mset*: $\text{poly } (\sum x \in \#A. p\ x) y = (\sum x \in \#A. \text{poly } (p\ x) y)$
<proof>

lemma *poly-prod-mset*: $\text{poly } (\prod x \in \#A. p\ x) y = (\prod x \in \#A. \text{poly } (p\ x) y)$
<proof>

lemma *set-mset-subset-singletonD*:
assumes $\text{set-mset } A \subseteq \{x\}$
shows $A = \text{replicate-mset } (\text{size } A) x$
<proof>

lemma *inj-idom-hom-compose [intro]*:
assumes $\text{inj-idom-hom } f\ \text{inj-idom-hom } g$
shows $\text{inj-idom-hom } (f \circ g)$
<proof>

lemma *coeff-pcompose-linear*:
 $\text{coeff } (p\text{compose } p\ [:0, a :: 'a :: \text{comm-semiring-1}])\ i = a^i * \text{coeff } p\ i$
<proof>

lemma *degree-cong*:
assumes $\bigwedge i. \text{coeff } p\ i = 0 \longleftrightarrow \text{coeff } q\ i = 0$
shows $\text{degree } p = \text{degree } q$
<proof>

lemma *coeff-Abs-poly-If-le*:
 $\text{coeff } (\text{Abs-poly } (\lambda i. \text{if } i \leq n \text{ then } f\ i \text{ else } 0)) = (\lambda i. \text{if } i \leq n \text{ then } f\ i \text{ else } 0)$
<proof>

lemma *coeff-Abs-poly*:
assumes $\bigwedge i. i > n \implies f\ i = 0$
shows $\text{coeff } (\text{Abs-poly } f) = f$
<proof>

lemma (**in** *inj-idom-hom*) *inj-idom-hom-map-poly [intro]*: $\text{inj-idom-hom } (\text{map-poly } \text{hom})$
<proof>

lemma *inj-idom-hom-pcompose [intro]*:
assumes [*simp*]: $\text{degree } (p :: 'a :: \text{idom poly}) \neq 0$

shows *inj-idom-hom* ($\lambda q. pcompose\ q\ p$)
 ⟨*proof*⟩

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

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

lemma *coeff-linear-poly-power*:
fixes $c :: 'a :: semiring-1$
assumes $i \leq n$
shows $coeff\ ([:a, b:] \wedge n)\ i = of_nat\ (n\ choose\ i) * b \wedge i * a \wedge (n - i)$
 ⟨*proof*⟩

lemma *pcompose-altdef*: $pcompose\ p\ q = poly\ (map_poly\ (\lambda x. [:x:])\ p)\ q$
 ⟨*proof*⟩

lemma *coeff-mult-0*: $coeff\ (p * q)\ 0 = coeff\ p\ 0 * coeff\ q\ 0$
 ⟨*proof*⟩

lemma *coeff-pcompose-0 [simp]*:
 $coeff\ (pcompose\ p\ q)\ 0 = poly\ p\ (coeff\ q\ 0)$
 ⟨*proof*⟩

lemma *reducible-polyI*:
fixes $p :: 'a :: field\ poly$
assumes $p = q * r$ $degree\ q > 0$ $degree\ r > 0$
shows $\neg irreducible\ p$
 ⟨*proof*⟩

lemma *root-imp-reducible-poly*:
fixes $x :: 'a :: field$
assumes $poly\ p\ x = 0$ **and** $degree\ p > 1$
shows $\neg irreducible\ p$
 ⟨*proof*⟩

1.2 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 *alg-closed*: $n > 0 \implies f\ n \neq 0 \implies \exists x. (\sum_{k \leq n} f\ k * x \wedge k) = 0$

We can then however easily show the equivalence to the proper definition:

lemma *alg-closed-imp-poly-has-root*:

assumes $\text{degree } (p :: 'a :: \text{alg-closed-field poly}) > 0$

shows $\exists x. \text{poly } p \ x = 0$

<proof>

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 $\text{OFCLASS}('a :: \text{field, alg-closed-field-class})$

<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 $\bigwedge p :: 'a \text{ poly. degree } p > 1 \implies \text{lead-coeff } p = 1 \implies \text{coeff } p \ 0 \neq 0 \implies \neg \text{irreducible } p$

shows $\text{OFCLASS}('a :: \text{field, alg-closed-field-class})$

<proof>

Using a clever Tschirnhausen transformation mentioned e.g. in the article by Nowak [2], 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 $\bigwedge p :: 'a \text{ poly. degree } p > 1 \implies \text{lead-coeff } p = 1 \implies \text{coeff } p \ (\text{degree } p - 1) = 0 \implies$

$\text{coeff } p \ 0 \neq 0 \implies \neg \text{irreducible } p$

shows $\text{OFCLASS}('a :: \text{field-char-0, alg-closed-field-class})$

<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 $r \ s$ **where** $\text{degree } r > 0 \ \text{degree } s > 0 \ \text{coprime } r \ s \ p = r * s$
 $\langle \text{proof} \rangle$

instance $\text{complex} :: \text{alg-closed-field}$
 $\langle \text{proof} \rangle$

end

1.3 Facts about Laurent series

theory *Puiseux-Laurent-Library*
imports *HOL-Computational-Algebra.Computational-Algebra*
begin

lemma *filterlim-at-top-div-const-nat*:
assumes $c > 0$
shows $\text{filterlim } (\lambda x :: \text{nat}. x \ \text{div } c) \ \text{at-top} \ \text{at-top}$
 $\langle \text{proof} \rangle$

lemma *fls-eq-iff*: $f = g \iff (\forall n. \text{fls-nth } f \ n = \text{fls-nth } g \ n)$
 $\langle \text{proof} \rangle$

lemma *fls-shift-eq-1-iff*: $\text{fls-shift } n \ f = 1 \iff f = \text{fls-shift } (-n) \ 1$
 $\langle \text{proof} \rangle$

lemma *fps-to-fls-eq-iff* [*simp*]: $\text{fps-to-fls } f = \text{fps-to-fls } g \iff f = g$
 $\langle \text{proof} \rangle$

lemma *fps-to-fls-eq-0-iff* [*simp*]: $\text{fps-to-fls } f = 0 \iff f = 0$
 $\langle \text{proof} \rangle$

lemma *fps-to-fls-eq-1-iff* [*simp*]: $\text{fps-to-fls } f = 1 \iff f = 1$
 $\langle \text{proof} \rangle$

lemma *fps-to-fls-power*: $\text{fps-to-fls } (f \wedge n) = \text{fps-to-fls } f \wedge n$
 $\langle \text{proof} \rangle$

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 $\exists f'. f = \text{fls-shift } n \ (\text{fps-to-fls } f')$

<proof>

definition *fls-compose-fps* :: 'a :: field fls \Rightarrow 'a fps \Rightarrow 'a fls **where**

fls-compose-fps f g =
(if f = 0 then 0
else if fls-subdegree f \geq 0 then fps-to-fls (fps-compose (fls-regpart f) g)
else fps-to-fls (fps-compose (fls-base-factor-to-fps f) g) /
fps-to-fls g ^ nat (-fls-subdegree f))

lemma *fls-compose-fps-fps* [simp]:

fls-compose-fps (fps-to-fls f) g = fps-to-fls (fps-compose f g)
<proof>

lemma *fls-const-transfer* [transfer-rule]:

rel-fun (=) (*pcr-fls* (=))
($\lambda c n.$ if n = 0 then c else 0) *fls-const*
<proof>

lemma *fls-shift-transfer* [transfer-rule]:

rel-fun (=) (*rel-fun* (*pcr-fls* (=)) (*pcr-fls* (=)))
($\lambda n f k.$ f (k+n)) *fls-shift*
<proof>

lift-definition *fls-compose-power* :: 'a :: zero fls \Rightarrow nat \Rightarrow 'a fls **is**

$\lambda f d n.$ if d > 0 \wedge int d dvd n then f (n div int d) else 0
<proof>

lemma *fls-nth-compose-power*:

assumes d > 0
shows *fls-nth* (*fls-compose-power* f d) n = (if int d dvd n then *fls-nth* 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 \implies *fls-compose-power* 1 d = 1

<proof>

lemma *fls-compose-power-const-left* [simp]:

d > 0 \implies *fls-compose-power* (*fls-const* c) d = *fls-const* c
<proof>

lemma *fls-compose-power-shift* [simp]:

d > 0 \implies *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]: *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*:

$\text{fps-nth } (f \text{ oo } (\text{fps-X } ^ d)) \ n = (\text{if } d \ \text{dvd } n \ \text{then } \text{fps-nth } f \ (n \ \text{div } d) \ \text{else } 0)$
(proof)

lemma *fls-compose-power-fps-to-fls*:

assumes $d > 0$

shows $\text{fls-compose-power } (\text{fps-to-fls } f) \ d = \text{fps-to-fls } (\text{fps-compose } f \ (\text{fps-X } ^ d))$

(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 $\text{fls-compose-power } (f ^ n :: 'a :: \text{idom } \text{fls}) \ d = \text{fls-compose-power } f \ d ^ n$

(proof)

lemma *fls-nth-compose-power'* [simp]:

$d = 0 \vee \neg d \text{ dvd } n \implies \text{fls-nth } (\text{fls-compose-power } f \ d) \ n = 0$
 $d \text{ dvd } n \implies d > 0 \implies \text{fls-nth } (\text{fls-compose-power } f \ d) \ n = \text{fls-nth } f \ (n \text{ div } d)$
 <proof>

end

2 Hensel's lemma for formal power series

theory *FPS-Hensel*

imports *HOL-Computational-Algebra.Computational-Algebra Puiseux-Polynomial-Library*
begin

The following proof of Hensel's lemma for formal power series follows the book "Algebraic Geometry for Scientists and Engineers" by Abhyankar [1, p. 90–92].

definition *fps-poly-swap1* :: 'a :: zero fps poly \Rightarrow 'a poly fps **where**
fps-poly-swap1 p = Abs-fps ($\lambda m. \text{Abs-poly } (\lambda n. \text{fps-nth } (\text{coeff } p \ n) \ m)$)

lemma *coeff-fps-nth-fps-poly-swap1* [*simp*]:
 $\text{coeff } (\text{fps-nth } (\text{fps-poly-swap1 } p) \ m) \ n = \text{fps-nth } (\text{coeff } p \ n) \ m$
 <proof>

definition *fps-poly-swap2* :: 'a :: zero poly fps \Rightarrow 'a fps poly **where**
fps-poly-swap2 p = Abs-poly ($\lambda m. \text{Abs-fps } (\lambda n. \text{coeff } (\text{fps-nth } p \ n) \ m)$)

lemma *fps-nth-coeff-fps-poly-swap2*:
assumes $\bigwedge n. \text{degree } (\text{fps-nth } p \ n) \leq d$
shows $\text{fps-nth } (\text{coeff } (\text{fps-poly-swap2 } p) \ m) \ n = \text{coeff } (\text{fps-nth } p \ n) \ m$
 <proof>

lemma *degree-fps-poly-swap2-le*:
assumes $\bigwedge n. \text{degree } (\text{fps-nth } p \ n) \leq d$
shows $\text{degree } (\text{fps-poly-swap2 } p) \leq d$
 <proof>

lemma *degree-fps-poly-swap2-eq*:
assumes $\bigwedge n. \text{degree } (\text{fps-nth } p \ n) \leq d$
assumes $d > 0 \vee \text{fps-nth } p \ n \neq 0$
assumes $\text{degree } (\text{fps-nth } p \ n) = d$
shows $\text{degree } (\text{fps-poly-swap2 } p) = d$
 <proof>

definition *reduce-fps-poly* :: 'a :: zero fps poly \Rightarrow 'a poly **where**
reduce-fps-poly F = *fps-nth* (*fps-poly-swap1* F) 0

lemma
fixes F :: 'a :: field fps poly
assumes *lead-coeff* F = 1

shows *degree-reduce-fps-poly-monic*: $\text{degree} (\text{reduce-fps-poly } F) = \text{degree } F$
and *reduce-fps-poly-monic*: $\text{lead-coeff} (\text{reduce-fps-poly } F) = 1$
⟨*proof*⟩

locale *fps-hensel-aux* =
fixes $F :: 'a :: \text{field-gcd poly fps}$
fixes $g\ h :: 'a \text{ poly}$
assumes *coprime*: $\text{coprime } g\ h$ **and** *deg-g*: $\text{degree } g > 0$ **and** *deg-h*: $\text{degree } h > 0$
begin

context
fixes $g'\ h' :: 'a \text{ poly}$
defines $h' \equiv \text{fst} (\text{bezout-coefficients } g\ h)$ **and** $g' \equiv \text{snd} (\text{bezout-coefficients } g\ h)$
begin

fun *hensel-fpxs-aux* :: $\text{nat} \Rightarrow 'a \text{ poly} \times 'a \text{ poly}$ **where**
hensel-fpxs-aux $n = (\text{if } n = 0 \text{ then } (g, h) \text{ else}$
 (let
 $U = \text{fps-nth } F\ n -$
 $(\sum (i,j) \mid i < n \wedge j < n \wedge i + j = n. \text{fst} (\text{hensel-fpxs-aux } i) * \text{snd}$
*hensel-fpxs-aux } j))
 in $(U * g' + g * ((U * h') \text{ div } h), (U * h') \text{ mod } h))$)*

lemmas [*simp del*] = *hensel-fpxs-aux.simps*

lemma *hensel-fpxs-aux-0* [*simp*]: $\text{hensel-fpxs-aux } 0 = (g, h)$
⟨*proof*⟩

definition *hensel-fpxs1* :: $'a \text{ poly fps}$
where *hensel-fpxs1* = $\text{Abs-fps} (\text{fst} \circ \text{hensel-fpxs-aux})$

definition *hensel-fpxs2* :: $'a \text{ poly fps}$
where *hensel-fpxs2* = $\text{Abs-fps} (\text{snd} \circ \text{hensel-fpxs-aux})$

lemma *hensel-fpxs1-0* [*simp*]: $\text{hensel-fpxs1 } \$ 0 = g$
⟨*proof*⟩

lemma *hensel-fpxs2-0* [*simp*]: $\text{hensel-fpxs2 } \$ 0 = h$
⟨*proof*⟩

theorem *fps-hensel-aux*:
defines $f \equiv \text{fps-nth } F\ 0$
assumes $f = g * h$
assumes $\forall n > 0. \text{degree} (\text{fps-nth } F\ n) < \text{degree } f$
defines $G \equiv \text{hensel-fpxs1}$ **and** $H \equiv \text{hensel-fpxs2}$
shows $F = G * H$ $\text{fps-nth } G\ 0 = g$ $\text{fps-nth } H\ 0 = h$
 $\forall n > 0. \text{degree} (\text{fps-nth } G\ n) < \text{degree } g$
 $\forall n > 0. \text{degree} (\text{fps-nth } H\ n) < \text{degree } h$
⟨*proof*⟩

end

end

locale *fps-hensel* =

fixes $F :: 'a :: \text{field-gcd fps poly}$ **and** $f\ g\ h :: 'a\ \text{poly}$

assumes *monic*: $\text{lead-coeff } F = 1$

defines $f \equiv \text{reduce-fps-poly } F$

assumes *f-splits*: $f = g * h$

assumes *coprime*: $\text{coprime } g\ h$ **and** *deg-g*: $\text{degree } g > 0$ **and** *deg-h*: $\text{degree } h > 0$

begin

definition F' **where** $F' = \text{fps-poly-swap1 } F$

sublocale *fps-hensel-aux* $F' g h$

<proof>

definition G **where**

$G = \text{fps-poly-swap2 hensel-fpxs1}$

definition H **where**

$H = \text{fps-poly-swap2 hensel-fpxs2}$

lemma *deg-f*: $\text{degree } f = \text{degree } F$

<proof>

lemma

F-splits: $F = G * H$ **and**

reduce-G: $\text{reduce-fps-poly } G = g$ **and**

reduce-H: $\text{reduce-fps-poly } H = h$ **and**

deg-G: $\text{degree } G = \text{degree } g$ **and**

deg-H: $\text{degree } H = \text{degree } h$ **and**

lead-coeff-G: $\text{lead-coeff } G = \text{fps-const } (\text{lead-coeff } g)$ **and**

lead-coeff-H: $\text{lead-coeff } H = \text{fps-const } (\text{lead-coeff } h)$

<proof>

end

end

3 Formal Puiseux Series

theory *Formal-Puiseux-Series*

imports *Puiseux-Laurent-Library FPS-Hensel*

begin

3.1 Auxiliary facts and definitions

lemma *div-dvd-self*:

fixes $a\ b :: 'a :: \{\text{semidom-divide}\}$
shows $b\ \text{dvd}\ a \implies a\ \text{div}\ b\ \text{dvd}\ a$
 $\langle \text{proof} \rangle$

lemma *quotient-of-int [simp]*: $\text{quotient-of}\ (\text{of-int}\ n) = (n, 1)$

$\langle \text{proof} \rangle$

lemma *of-int-div-of-int-in-Ints-iff*:

$(\text{of-int}\ n\ /\ \text{of-int}\ m :: 'a :: \text{field-char-0}) \in \mathbb{Z} \longleftrightarrow m = 0 \vee m\ \text{dvd}\ n$
 $\langle \text{proof} \rangle$

lemma *rat-eq-quotientD*:

assumes $r = \text{rat-of-int}\ a\ /\ \text{rat-of-int}\ b\ b \neq 0$
shows $\text{fst}\ (\text{quotient-of}\ r)\ \text{dvd}\ a\ \text{snd}\ (\text{quotient-of}\ r)\ \text{dvd}\ b$
 $\langle \text{proof} \rangle$

lemma *quotient-of-denom-add-dvd*:

$\text{snd}\ (\text{quotient-of}\ (x + y))\ \text{dvd}\ \text{snd}\ (\text{quotient-of}\ x) * \text{snd}\ (\text{quotient-of}\ y)$
 $\langle \text{proof} \rangle$

lemma *quotient-of-denom-diff-dvd*:

$\text{snd}\ (\text{quotient-of}\ (x - y))\ \text{dvd}\ \text{snd}\ (\text{quotient-of}\ x) * \text{snd}\ (\text{quotient-of}\ y)$
 $\langle \text{proof} \rangle$

definition *supp* :: $('a \Rightarrow ('b :: \text{zero})) \Rightarrow 'a\ \text{set}$ **where**

$\text{supp}\ f = f\ -' (-\{0\})$

lemma *supp-0 [simp]*: $\text{supp}\ (\lambda_.\ 0) = \{\}$

and *supp-const*: $\text{supp}\ (\lambda_.\ c) = (\text{if}\ c = 0\ \text{then}\ \{\}\ \text{else}\ \text{UNIV})$

and *supp-singleton [simp]*: $c \neq 0 \implies \text{supp}\ (\lambda x.\ \text{if}\ x = d\ \text{then}\ c\ \text{else}\ 0) = \{d\}$

$\langle \text{proof} \rangle$

lemma *supp-uminus [simp]*: $\text{supp}\ (\lambda x.\ -f\ x :: 'a :: \text{group-add}) = \text{supp}\ f$

$\langle \text{proof} \rangle$

3.2 Definition

Similarly to formal power series $R[[X]]$ and formal Laurent series $R((X))$, we define the ring of formal Puiseux series $R\{\{X\}\}$ as functions from the rationals into a ring such that

1. the support is bounded from below, and
2. the denominators of the numbers in the support have a common multiple other than 0

One can also think of a formal Puiseux series in the parameter X as a formal Laurent series in the parameter $X^{1/d}$ for some positive integer d . This is often written in the following suggestive notation:

$$R\{\{X\}\} = \bigcup_{d \geq 1} R((X^{1/d}))$$

Many operations will be defined in terms of this correspondence between Puiseux and Laurent series, and many of the simple properties proven that way.

definition *is-fpxs* :: (rat \Rightarrow 'a :: zero) \Rightarrow bool **where**
is-fpxs f \longleftrightarrow bdd-below (supp f) \wedge (LCM r \in supp f. snd (quotient-of r)) \neq 0

typedef (overloaded) 'a fpxs = {f::rat \Rightarrow 'a :: zero. *is-fpxs* f}
morphisms fpxs-nth Abs-fpxs
 <proof>

setup-lifting type-definition-fpxs

lemma *fpxs-ext*: (\bigwedge r. fpxs-nth f r = fpxs-nth g r) \implies f = g
 <proof>

lemma *fpxs-eq-iff*: f = g \longleftrightarrow (\forall r. fpxs-nth f r = fpxs-nth g r)
 <proof>

lift-definition *fpxs-supp* :: 'a :: zero fpxs \Rightarrow rat set **is** supp <proof>

lemma *fpxs-supp-altdef*: *fpxs-supp* f = {x. fpxs-nth f x \neq 0}
 <proof>

The following gives us the “root order” of f , i.e. the smallest positive integer d such that f is in $R((X^{1/d}))$.

lift-definition *fpxs-root-order* :: 'a :: zero fpxs \Rightarrow nat **is**
 λ f. nat (LCM r \in supp f. snd (quotient-of r)) <proof>

lemma *fpxs-root-order-pos* [simp]: *fpxs-root-order* f > 0
 <proof>

lemma *fpxs-root-order-nonzero* [simp]: *fpxs-root-order* f \neq 0
 <proof>

Let d denote the root order of a Puiseux series f , i.e. the smallest number d such that all monomials with non-zero coefficients can be written in the form $X^{n/d}$ for some n . Then f can be written as a Laurent series in $X^{1/d}$. The following operation gives us this Laurent series.

lift-definition *fls-of-fpxs* :: 'a :: zero fpxs \Rightarrow 'a fls **is**
 λ f n. f (of-int n / of-int (LCM r \in supp f. snd (quotient-of r)))

<proof>

lemma *fls-nth-of-fpxs*:

fls-nth (fls-of-fpxs f) n = fpxs-nth f (of-int n / of-nat (fpxs-root-order f))

<proof>

3.3 Basic algebraic typeclass instances

instantiation *fpxs* :: (*zero*) *zero*

begin

lift-definition *zero-fpxs* :: 'a *fpxs* **is** $\lambda r::\text{rat. } 0 :: 'a$

<proof>

instance *<proof>*

end

instantiation *fpxs* :: (*{one, zero}*) *one*

begin

lift-definition *one-fpxs* :: 'a *fpxs* **is** $\lambda r::\text{rat. if } r = 0 \text{ then } 1 \text{ else } 0 :: 'a$

<proof>

instance *<proof>*

end

lemma *fls-of-fpxs-0* [*simp*]: *fls-of-fpxs 0 = 0*

<proof>

lemma *fpxs-nth-0* [*simp*]: *fpxs-nth 0 r = 0*

<proof>

lemma *fpxs-nth-1*: *fpxs-nth 1 r = (if r = 0 then 1 else 0)*

<proof>

lemma *fpxs-nth-1'*: *fpxs-nth 1 0 = 1 r ≠ 0 ⇒ fpxs-nth 1 r = 0*

<proof>

instantiation *fpxs* :: (*monoid-add*) *monoid-add*

begin

lift-definition *plus-fpxs* :: 'a *fpxs* ⇒ 'a *fpxs* ⇒ 'a *fpxs* **is**

$\lambda f g x. f x + g x$

<proof>

instance

<proof>

end

instance $fpxs :: (comm-monoid-add) comm-monoid-add$
 $\langle proof \rangle$

lemma $fpxs-nth-add [simp]: fpxs-nth (f + g) r = fpxs-nth f r + fpxs-nth g r$
 $\langle proof \rangle$

lift-definition $fpxs-of-fls :: 'a :: zero fls \Rightarrow 'a fpxs$ **is**
 $\lambda f r. \text{if } r \in \mathbb{Z} \text{ then } f [r] \text{ else } 0$
 $\langle proof \rangle$

instantiation $fpxs :: (group-add) group-add$
begin

lift-definition $uminus-fpxs :: 'a fpxs \Rightarrow 'a fpxs$ **is** $\lambda f x. -f x$
 $\langle proof \rangle$

definition $minus-fpxs :: 'a fpxs \Rightarrow 'a fpxs \Rightarrow 'a fpxs$ **where**
 $minus-fpxs f g = f + (-g)$

instance $\langle proof \rangle$

end

lemma $fpxs-nth-uminus [simp]: fpxs-nth (-f) r = -fpxs-nth f r$
 $\langle proof \rangle$

lemma $fpxs-nth-minus [simp]: fpxs-nth (f - g) r = fpxs-nth f r - fpxs-nth g r$
 $\langle proof \rangle$

lemma $fpxs-of-fls-eq-iff [simp]: fpxs-of-fls f = fpxs-of-fls g \iff f = g$
 $\langle proof \rangle$

lemma $fpxs-of-fls-0 [simp]: fpxs-of-fls 0 = 0$
 $\langle proof \rangle$

lemma $fpxs-of-fls-1 [simp]: fpxs-of-fls 1 = 1$
 $\langle proof \rangle$

lemma $fpxs-of-fls-add [simp]: fpxs-of-fls (f + g) = fpxs-of-fls f + fpxs-of-fls g$
 $\langle proof \rangle$

lemma $fpxs-to-fls-sum [simp]: fpxs-to-fls (sum f A) = (\sum x \in A. fpxs-to-fls (f x))$
 $\langle proof \rangle$

lemma $fpxs-of-fls-sum [simp]: fpxs-of-fls (sum f A) = (\sum x \in A. fpxs-of-fls (f x))$
 $\langle proof \rangle$

lemma *fpxs-nth-of-fls*:

fpxs-nth (fpxs-of-fls f) r = (if r ∈ ℤ then fls-nth f [r] else 0)
 ⟨proof⟩

lemma *fpxs-of-fls-eq-0-iff [simp]*: *fpxs-of-fls f = 0 ⟷ f = 0*
 ⟨proof⟩

lemma *fpxs-of-fls-eq-1-iff [simp]*: *fpxs-of-fls f = 1 ⟷ f = 1*
 ⟨proof⟩

lemma *fpxs-root-order-of-fls [simp]*: *fpxs-root-order (fpxs-of-fls f) = 1*
 ⟨proof⟩

3.4 The substitution $X \mapsto X^r$

This operation turns a formal Puiseux series $f(X)$ into $f(X^r)$, where r can be any positive rational number:

lift-definition *fpxs-compose-power* :: 'a :: zero fpxs ⇒ rat ⇒ 'a fpxs is
λf r x. if r > 0 then f (x / r) else 0
 ⟨proof⟩

lemma *fpxs-as-fls*:

fpxs-compose-power (fpxs-of-fls (fls-of-fpxs f)) (1 / of-nat (fpxs-root-order f)) = f
 ⟨proof⟩

lemma *fpxs-compose-power-0 [simp]*: *fpxs-compose-power 0 r = 0*
 ⟨proof⟩

lemma *fpxs-compose-power-1 [simp]*: *r > 0 ⟹ fpxs-compose-power 1 r = 1*
 ⟨proof⟩

lemma *fls-of-fpxs-eq-0-iff [simp]*: *fls-of-fpxs x = 0 ⟷ x = 0*
 ⟨proof⟩

lemma *fpxs-of-fls-compose-power [simp]*:

fpxs-of-fls (fls-compose-power f d) = fpxs-compose-power (fpxs-of-fls f) (of-nat d)
 ⟨proof⟩

lemma *fpxs-compose-power-add [simp]*:

fpxs-compose-power (f + g) r = fpxs-compose-power f r + fpxs-compose-power g r
 ⟨proof⟩

lemma *fpxs-compose-power-distrib*:

r1 > 0 ∨ r2 > 0 ⟹
*fpxs-compose-power (fpxs-compose-power f r1) r2 = fpxs-compose-power f (r1 * r2)*

<proof>

lemma *fpxs-compose-power-divide-right*:

$r1 > 0 \implies r2 > 0 \implies$

$fpxs\text{-compose-power } f (r1 / r2) = fpxs\text{-compose-power } (fpxs\text{-compose-power } f$
 $r1) (inverse\ r2)$

<proof>

lemma *fpxs-compose-power-1-right* [*simp*]: $fpxs\text{-compose-power } f\ 1 = f$

<proof>

lemma *fpxs-compose-power-eq-iff* [*simp*]:

assumes $r > 0$

shows $fpxs\text{-compose-power } f\ r = fpxs\text{-compose-power } g\ r \iff f = g$

<proof>

lemma *fpxs-compose-power-eq-1-iff* [*simp*]:

assumes $l > 0$

shows $fpxs\text{-compose-power } p\ l = 1 \iff p = 1$

<proof>

lemma *fpxs-compose-power-eq-0-iff* [*simp*]:

assumes $r > 0$

shows $fpxs\text{-compose-power } f\ r = 0 \iff f = 0$

<proof>

lemma *fls-of-fpxs-of-fls* [*simp*]: $fls\text{-of-fpxs } (fpxs\text{-of-fls } f) = f$

<proof>

lemma *fpxs-as-fls'*:

assumes $fpxs\text{-root-order } f\ dvd\ d\ d > 0$

obtains f' **where** $f = fpxs\text{-compose-power } (fpxs\text{-of-fls } f') (1 / of\text{-nat } d)$

<proof>

3.5 Multiplication and ring properties

instantiation $fpxs :: (comm\text{-semiring-1})\ comm\text{-semiring-1}$

begin

lift-definition $times\text{-fpxs} :: 'a\ fpxs \Rightarrow 'a\ fpxs \Rightarrow 'a\ fpxs$ **is**

$\lambda f\ g\ x. (\sum (y,z) \mid y \in supp\ f \wedge z \in supp\ g \wedge x = y + z. f\ y * g\ z)$

<proof>

lemma *fpxs-nth-mult*:

$fpxs\text{-nth } (f * g)\ r =$

$(\sum (y,z) \mid y \in fpxs\text{-supp } f \wedge z \in fpxs\text{-supp } g \wedge r = y + z. fpxs\text{-nth } f\ y * fpxs\text{-nth } g\ z)$

<proof>

lemma *fpxs-compose-power-mult* [simp]:
fpxs-compose-power (f * g) r = *fpxs-compose-power* f r * *fpxs-compose-power* g r
 ⟨proof⟩

lemma *fpxs-supp-of-fls*: *fpxs-supp* (*fpxs-of-fls* f) = *of-int* 'supp (*fls-nth* f)
 ⟨proof⟩

lemma *fpxs-of-fls-mult* [simp]: *fpxs-of-fls* (f * g) = *fpxs-of-fls* f * *fpxs-of-fls* g
 ⟨proof⟩

instance ⟨proof⟩

end

instance *fpxs* :: (comm-ring-1) comm-ring-1
 ⟨proof⟩

instance *fpxs* :: ({comm-semiring-1, semiring-no-zero-divisors}) semiring-no-zero-divisors
 ⟨proof⟩

lemma *fpxs-of-fls-power* [simp]: *fpxs-of-fls* (f ^ n) = *fpxs-of-fls* f ^ n
 ⟨proof⟩

lemma *fpxs-compose-power-power* [simp]:
 r > 0 ⇒ *fpxs-compose-power* (f ^ n) r = *fpxs-compose-power* f r ^ n
 ⟨proof⟩

3.6 Constant Puiseux series and the series X

lift-definition *fpxs-const* :: 'a :: zero ⇒ 'a *fpxs* is
 λc n. if n = 0 then c else 0
 ⟨proof⟩

lemma *fpxs-const-0* [simp]: *fpxs-const* 0 = 0
 ⟨proof⟩

lemma *fpxs-const-1* [simp]: *fpxs-const* 1 = 1
 ⟨proof⟩

lemma *fpxs-of-fls-const* [simp]: *fpxs-of-fls* (*fls-const* c) = *fpxs-const* c
 ⟨proof⟩

lemma *fls-of-fpxs-const* [simp]: *fls-of-fpxs* (*fpxs-const* c) = *fls-const* c
 ⟨proof⟩

lemma *fls-of-fpxs-1* [simp]: *fls-of-fpxs* 1 = 1
 ⟨proof⟩

lift-definition *fpxs-X* :: 'a :: {one, zero} *fpxs* is

$\lambda x. \text{if } x = 1 \text{ then } (1 :: 'a) \text{ else } 0$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-altdef*: $\text{fpxs-const } x = \text{fpxs-of-fls } (\text{fls-const } x)$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-add* [*simp*]: $\text{fpxs-const } (x + y) = \text{fpxs-const } x + \text{fpxs-const } y$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-mult* [*simp*]:
fixes $x y :: 'a :: \{\text{comm-semiring-1}\}$
shows $\text{fpxs-const } (x * y) = \text{fpxs-const } x * \text{fpxs-const } y$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-eq-iff* [*simp*]:
 $\text{fpxs-const } x = \text{fpxs-const } y \longleftrightarrow x = y$
 $\langle \text{proof} \rangle$

lemma *of-nat-fpxs-eq*: $\text{of-nat } n = \text{fpxs-const } (\text{of-nat } n)$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-uminus* [*simp*]: $\text{fpxs-const } (-x) = -\text{fpxs-const } x$
 $\langle \text{proof} \rangle$

lemma *fpxs-const-diff* [*simp*]: $\text{fpxs-const } (x - y) = \text{fpxs-const } x - \text{fpxs-const } y$
 $\langle \text{proof} \rangle$

lemma *of-int-fpxs-eq*: $\text{of-int } n = \text{fpxs-const } (\text{of-int } n)$
 $\langle \text{proof} \rangle$

3.7 More algebraic typeclass instances

instance *fpxs* :: ($\{\text{comm-semiring-1}, \text{semiring-char-0}\}$) *semiring-char-0*
 $\langle \text{proof} \rangle$

instance *fpxs* :: ($\{\text{comm-ring-1}, \text{ring-char-0}\}$) *ring-char-0* $\langle \text{proof} \rangle$

instance *fpxs* :: (*idom*) *idom* $\langle \text{proof} \rangle$

instantiation *fpxs* :: (*field*) *field*
begin

definition *inverse-fpxs* :: '*a* *fpxs* \Rightarrow '*a* *fpxs* **where**
inverse-fpxs $f =$
 $\text{fpxs-compose-power } (\text{fpxs-of-fls } (\text{inverse } (\text{fls-of-fpxs } f))) (1 / \text{of-nat } (\text{fpxs-root-order } f))$

definition *divide-fpxs* :: '*a* *fpxs* \Rightarrow '*a* *fpxs* \Rightarrow '*a* *fpxs* **where**
divide-fpxs $f g = f * \text{inverse } g$

instance $\langle proof \rangle$

end

instance $fpxs :: (field-char-0) field-char-0 \langle proof \rangle$

3.8 Valuation

definition $fpxs-val :: 'a :: zero fpxs \Rightarrow rat$ **where**

$fpxs-val f =$
 $of-int (fls-subdegree (fls-of-fpxs f)) / rat-of-nat (fpxs-root-order f)$

lemma $fpxs-val-of-fls [simp]: fpxs-val (fpxs-of-fls f) = of-int (fls-subdegree f)$
 $\langle proof \rangle$

lemma $fpxs-nth-compose-power [simp]:$

assumes $r > 0$

shows $fpxs-nth (fpxs-compose-power f r) n = fpxs-nth f (n / r)$
 $\langle proof \rangle$

lemma $fls-of-fpxs-uminus [simp]: fls-of-fpxs (-f) = -fls-of-fpxs f$
 $\langle proof \rangle$

lemma $fpxs-root-order-uminus [simp]: fpxs-root-order (-f) = fpxs-root-order f$
 $\langle proof \rangle$

lemma $fpxs-val-uminus [simp]: fpxs-val (-f) = fpxs-val f$
 $\langle proof \rangle$

lemma $fpxs-val-minus-commute: fpxs-val (f - g) = fpxs-val (g - f)$
 $\langle proof \rangle$

lemma $fpxs-val-const [simp]: fpxs-val (fpxs-const c) = 0$
 $\langle proof \rangle$

lemma $fpxs-val-1 [simp]: fpxs-val 1 = 0$
 $\langle proof \rangle$

lemma $of-int-fls-subdegree-of-fpxs:$
 $rat-of-int (fls-subdegree (fls-of-fpxs f)) = fpxs-val f * of-nat (fpxs-root-order f)$
 $\langle proof \rangle$

lemma $fpxs-nth-val-nonzero:$

assumes $f \neq 0$

shows $fpxs-nth f (fpxs-val f) \neq 0$
 $\langle proof \rangle$

lemma $fpxs-nth-below-val:$

assumes $n: n < \text{fpxs-val } f$
shows $\text{fpxs-nth } f \ n = 0$
 $\langle \text{proof} \rangle$

lemma $\text{fpxs-val-leI}: \text{fpxs-nth } f \ r \neq 0 \implies \text{fpxs-val } f \leq r$
 $\langle \text{proof} \rangle$

lemma fpxs-val-0 [simp]: $\text{fpxs-val } 0 = 0$
 $\langle \text{proof} \rangle$

lemma fpxs-val-geI :
assumes $f \neq 0 \wedge r. r < r' \implies \text{fpxs-nth } f \ r = 0$
shows $\text{fpxs-val } f \geq r'$
 $\langle \text{proof} \rangle$

lemma $\text{fpxs-val-compose-power}$ [simp]:
assumes $r > 0$
shows $\text{fpxs-val } (\text{fpxs-compose-power } f \ r) = \text{fpxs-val } f * r$
 $\langle \text{proof} \rangle$

lemma fpxs-val-add-ge :
assumes $f + g \neq 0$
shows $\text{fpxs-val } (f + g) \geq \min (\text{fpxs-val } f) (\text{fpxs-val } g)$
 $\langle \text{proof} \rangle$

lemma fpxs-val-diff-ge :
assumes $f \neq g$
shows $\text{fpxs-val } (f - g) \geq \min (\text{fpxs-val } f) (\text{fpxs-val } g)$
 $\langle \text{proof} \rangle$

lemma fpxs-nth-mult-val :
 $\text{fpxs-nth } (f * g) (\text{fpxs-val } f + \text{fpxs-val } g) = \text{fpxs-nth } f (\text{fpxs-val } f) * \text{fpxs-nth } g$
 $(\text{fpxs-val } g)$
 $\langle \text{proof} \rangle$

lemma fpxs-val-mult [simp]:
fixes $f \ g :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$ fpxs
assumes $f \neq 0 \ g \neq 0$
shows $\text{fpxs-val } (f * g) = \text{fpxs-val } f + \text{fpxs-val } g$
 $\langle \text{proof} \rangle$

lemma fpxs-val-power [simp]:
fixes $f :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$ fpxs
assumes $f \neq 0 \vee n > 0$
shows $\text{fpxs-val } (f \wedge n) = \text{of-nat } n * \text{fpxs-val } f$
 $\langle \text{proof} \rangle$

lemma $\text{fpxs-nth-power-val}$ [simp]:
fixes $f :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$ fpxs

shows $\text{fpxs-nth } (f \hat{\ } r) (\text{rat-of-nat } r * \text{fpxs-val } f) = \text{fpxs-nth } f (\text{fpxs-val } f) \hat{\ } r$
 ⟨proof⟩

3.9 Powers of X and shifting

lift-definition $\text{fpxs-}X\text{-power} :: \text{rat} \Rightarrow 'a :: \{\text{zero}, \text{one}\} \text{fpxs}$ **is**

$\lambda r n :: \text{rat. if } n = r \text{ then } 1 \text{ else } (0 :: 'a)$
 ⟨proof⟩

lemma $\text{fpxs-}X\text{-power-0}$ [simp]: $\text{fpxs-}X\text{-power } 0 = 1$
 ⟨proof⟩

lemma $\text{fpxs-}X\text{-power-add}$: $\text{fpxs-}X\text{-power } (a + b) = \text{fpxs-}X\text{-power } a * \text{fpxs-}X\text{-power } b$
 ⟨proof⟩

lemma $\text{fpxs-}X\text{-power-mult}$: $\text{fpxs-}X\text{-power } (\text{rat-of-nat } n * m) = \text{fpxs-}X\text{-power } m \hat{\ } n$
 ⟨proof⟩

lemma $\text{fpxs-of-fls-}X\text{-power}$ [simp]: $\text{fpxs-of-fls } (\text{fls-shift } n 1) = \text{fpxs-}X\text{-power } (-\text{rat-of-int } n)$
 ⟨proof⟩

lemma $\text{fpxs-}X\text{-power-neq-0}$ [simp]: $\text{fpxs-}X\text{-power } r \neq (0 :: 'a :: \text{zero-neq-one fpxs})$
 ⟨proof⟩

lemma $\text{fpxs-}X\text{-power-eq-1-iff}$ [simp]: $\text{fpxs-}X\text{-power } r = (1 :: 'a :: \text{zero-neq-one fpxs})$
 $\longleftrightarrow r = 0$
 ⟨proof⟩

lift-definition $\text{fpxs-shift} :: \text{rat} \Rightarrow 'a :: \text{zero fpxs} \Rightarrow 'a \text{fpxs}$ **is**

$\lambda r f n. f (n + r)$
 ⟨proof⟩

lemma fpxs-nth-shift [simp]: $\text{fpxs-nth } (\text{fpxs-shift } r f) n = \text{fpxs-nth } f (n + r)$
 ⟨proof⟩

lemma fpxs-shift-0-left [simp]: $\text{fpxs-shift } 0 f = f$
 ⟨proof⟩

lemma $\text{fpxs-shift-add-left}$: $\text{fpxs-shift } (m + n) f = \text{fpxs-shift } m (\text{fpxs-shift } n f)$
 ⟨proof⟩

lemma $\text{fpxs-shift-diff-left}$: $\text{fpxs-shift } (m - n) f = \text{fpxs-shift } m (\text{fpxs-shift } (-n) f)$
 ⟨proof⟩

lemma fpxs-shift-0 [simp]: $\text{fpxs-shift } r 0 = 0$

<proof>

lemma *fpxs-shift-add* [simp]: $fpxs-shift\ r\ (f + g) = fpxs-shift\ r\ f + fpxs-shift\ r\ g$
<proof>

lemma *fpxs-shift-uminus* [simp]: $fpxs-shift\ r\ (-f) = -fpxs-shift\ r\ f$
<proof>

lemma *fpxs-shift-shift-uminus* [simp]: $fpxs-shift\ r\ (fpxs-shift\ (-r)\ f) = f$
<proof>

lemma *fpxs-shift-shift-uminus'* [simp]: $fpxs-shift\ (-r)\ (fpxs-shift\ r\ f) = f$
<proof>

lemma *fpxs-shift-diff* [simp]: $fpxs-shift\ r\ (f - g) = fpxs-shift\ r\ f - fpxs-shift\ r\ g$
<proof>

lemma *fpxs-shift-compose-power* [simp]:
 $fpxs-shift\ r\ (fpxs-compose-power\ f\ s) = fpxs-compose-power\ (fpxs-shift\ (r / s)\ f)$
 s
<proof>

lemma *rat-of-int-div-dvd*: $d\ dvd\ n \implies rat-of-int\ (n\ div\ d) = rat-of-int\ n / rat-of-int\ d$
<proof>

lemma *fpxs-of-fls-shift* [simp]:
 $fpxs-of-fls\ (fls-shift\ n\ f) = fpxs-shift\ (of-int\ n)\ (fpxs-of-fls\ f)$
<proof>

lemma *fpxs-shift-mult*: $f * fpxs-shift\ r\ g = fpxs-shift\ r\ (f * g)$
 $fpxs-shift\ r\ f * g = fpxs-shift\ r\ (f * g)$
<proof>

lemma *fpxs-shift-1*: $fpxs-shift\ r\ 1 = fpxs-X-power\ (-r)$
<proof>

lemma *fpxs-X-power-conv-shift*: $fpxs-X-power\ r = fpxs-shift\ (-r)\ 1$
<proof>

lemma *fpxs-shift-power* [simp]: $fpxs-shift\ n\ x^{\wedge} m = fpxs-shift\ (of-nat\ m * n)\ (x^{\wedge} m)$
<proof>

lemma *fpxs-compose-power-X-power* [simp]:
 $s > 0 \implies fpxs-compose-power\ (fpxs-X-power\ r)\ s = fpxs-X-power\ (r * s)$
<proof>

3.10 The n -th root of a Puiseux series

In this section, we define the formal root of a Puiseux series. This is done using the same concept for formal power series. There is still one interesting theorem that is missing here, e.g. the uniqueness (which could probably be lifted over from FPSs) somehow.

definition *fpxs-radical* :: (nat \Rightarrow 'a :: field-char-0 \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a fpxs \Rightarrow 'a fpxs **where**

fpxs-radical rt r f = (if f = 0 then 0 else
 (let f' = fls-base-factor-to-fps (fls-of-fpxs f);
 f'' = fpxs-of-fls (fps-to-fls (fps-radical rt r f'))
 in fpxs-shift (-fpxs-val f / rat-of-nat r)
 (fpxs-compose-power f'' (1 / rat-of-nat (fpxs-root-order f))))))

lemma *fpxs-radical-0* [simp]: *fpxs-radical* rt r 0 = 0
 <proof>

lemma

fixes r :: nat

assumes r: r > 0

shows *fpxs-power-radical*:

rt r (fpxs-nth f (fpxs-val f)) \wedge r = fpxs-nth f (fpxs-val f) \implies *fpxs-radical* rt r f \wedge r = f

and *fpxs-radical-lead-coeff*:

f \neq 0 \implies fpxs-nth (fpxs-radical rt r f) (fpxs-val f / rat-of-nat r) =
 rt r (fpxs-nth f (fpxs-val f))

<proof>

lemma *fls-base-factor-power*:

fixes f :: 'a::{semiring-1, semiring-no-zero-divisors} fls

shows *fls-base-factor* (f \wedge n) = *fls-base-factor* f \wedge n

<proof>

hide-const (open) *supp*

3.11 Algebraic closedness

We will now show that the field of formal Puiseux series over an algebraically closed field of characteristic 0 is again algebraically closed.

The typeclass constraint *field-gcd* is a technical constraint that mandates that the field has a (trivial) GCD operation defined on it. It comes from some peculiarities of Isabelle's typeclass system and can be considered unimportant, since any concrete type of class *field* can easily be made an instance of *field-gcd*.

It would be possible to get rid of this constraint entirely here, but it is not

worth the effort.

The proof is a fairly standard one that uses Hensel's lemma. Some preliminary tricks are required to be able to use it, however, namely a number of non-obvious changes of variables to turn the polynomial with Puiseux coefficients into one with formal power series coefficients. The overall approach was taken from an article by Nowak [2].

Basically, what we need to show is this: Let

$$p(X, Z) = a_n(Z)X^n + a_{n-1}(Z)X^{n-1} + \dots + a_0(Z)$$

be a polynomial in X of degree at least 2 with coefficients that are formal Puiseux series in Z . Then p is reducible, i.e. it splits into two non-constant factors.

Due to work we have already done elsewhere, we may assume here that $a_n = 1$, $a_{n-1} = 0$, and $a_0 \neq 0$, all of which will come in very useful.

instance *fpzs* :: (*{alg-closed-field, field-char-0, field-gcd}*) *alg-closed-field*
<proof>

We do not actually show that this is the algebraic closure since this cannot be stated idiomatically in the typeclass setting and is probably not very useful either, but it can be motivated like this:

Suppose we have an algebraically closed extension L of the field of Laurent series. Clearly, $X^{a/b} \in L$ for any integer a and any positive integer b since $(X^{a/b})^b - X^a = 0$. But any Puiseux series $F(X)$ with root order b can be written as

$$F(X) = \sum_{k=0}^{b-1} X^{k/b} F_k(X)$$

where the Laurent series $F_k(X)$ are defined as follows:

$$F_k(X) := \sum_{n=n_{0,k}}^{\infty} [X^{n+k/b}]F(X)X^n$$

Thus, $F(X)$ can be written as a finite sum of products of elements in L and must therefore also be in L . Thus, the Puiseux series are all contained in L .

3.12 Metric and topology

Formal Puiseux series form a metric space with the usual metric for formal series: Two series are “close” to one another if they have many initial coefficients in common.

instantiation *fpzs* :: (*zero*) *norm*
begin

definition *norm-fpxs* :: 'a fpxs \Rightarrow real **where**
norm f = (if f = 0 then 0 else 2 powr (-of-rat (fpxs-val f)))

instance \langle proof \rangle

end

instantiation *fpxs* :: (group-add) dist
begin

definition *dist-fpxs* :: 'a fpxs \Rightarrow 'a fpxs \Rightarrow real **where**
dist f g = (if f = g then 0 else 2 powr (-of-rat (fpxs-val (f - g))))

instance \langle proof \rangle

end

instantiation *fpxs* :: (group-add) metric-space
begin

definition *uniformity-fpxs-def* [code del]:
(uniformity :: ('a fpxs \times 'a fpxs) filter) = (INF e \in {0 <.. $\}$. principal {(x, y). dist x y < e})

definition *open-fpxs-def* [code del]:
open (U :: 'a fpxs set) \longleftrightarrow ($\forall x \in U$. eventually ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) uniformity)

instance \langle proof \rangle

end

instance *fpxs* :: (group-add) dist-norm
 \langle proof \rangle

end

References

- [1] S. S. Abhyankar. *Algebraic Geometry for Scientists and Engineers*. Mathematical surveys and monographs. American Mathematical Society, 1990.
- [2] K. J. Nowak. Some elementary proofs of Puiseuxs theorems. *Univ. Iagel. Acta Math*, 38:279–282, 2000.