

Formal Puiseux Series

Manuel Eberl

February 6, 2026

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	3
2	Hensel's lemma for formal power series	4
3	Formal Puiseux Series	7
3.1	Auxiliary facts and definitions	7
3.2	Definition	8
3.3	Basic algebraic typeclass instances	9
3.4	The substitution $X \mapsto X^r$	12
3.5	Multiplication and ring properties	13
3.6	Constant Puiseux series and the series X	14
3.7	More algebraic typeclass instances	15
3.8	Valuation	15
3.9	Powers of X and shifting	17
3.10	The n -th root of a Puiseux series	19
3.11	Algebraic closedness	20
3.12	Metric and topology	21

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 *inj-idom-hom-compose* [*intro*]:
 assumes *inj-idom-hom f inj-idom-hom g*
 shows *inj-idom-hom (f ∘ g)*
 ⟨*proof*⟩

lemma (**in** *inj-idom-hom*) *inj-idom-hom-map-poly* [*intro*]: *inj-idom-hom (map-poly hom)*
 ⟨*proof*⟩

lemma *inj-idom-hom-pcompose* [*intro*]:
 assumes [*simp*]: *degree (p :: 'a :: idom poly) ≠ 0*
 shows *inj-idom-hom (λq. pcompose q 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 ⇒ f n ≠ 0 ⇒ ∃x. (∑ k≤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 *degree (p :: 'a :: alg-closed-field poly) > 0*
 shows $\exists x. \text{poly } p \ x = 0$
 ⟨*proof*⟩

lemma *alg-closedI* [*Pure.intro*]:
 assumes $\bigwedge p :: 'a \text{ poly. } \text{degree } p > 0 \Rightarrow \text{lead-coeff } p = 1 \Rightarrow \exists x. \text{poly } p \ x = 0$
 shows *OFCLASS('a :: 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 $\exists A. \text{size } A = \text{degree } p \wedge p = \text{smult } (\text{lead-coeff } p) \left(\prod_{x \in \#A. } [:-x, 1:] \right)$
 ⟨*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** $\text{degree } r > 0 \ \text{degree } s > 0 \ \text{coprime } r \ s \ p = r * s$

<proof>

instance *complex* :: *alg-closed-field*

<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 ($\lambda m.$ Abs-poly ($\lambda n.$ fps-nth (coeff p n) m))

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

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

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

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

lemma *degree-fps-poly-swap2-eq*:
 assumes $\bigwedge n.$ degree (fps-nth p n) \leq d
 assumes $d > 0 \vee \text{fps-nth } p \ n \neq 0$
 assumes degree (fps-nth p n) = d
 shows degree (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*: degree (reduce-fps-poly F) = degree F
 and *reduce-fps-poly-monic*: lead-coeff (reduce-fps-poly F) = 1
 <proof>

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

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

fun *hensel-fps-aux* :: nat \Rightarrow 'a poly \times 'a poly **where**

$hensel-fpxs-aux\ n = (if\ n = 0\ then\ (g,\ h)\ else$
 $(let$
 $\quad U = fps-nth\ F\ n -$
 $\quad (\sum\ (i,j)\ |\ i < n \wedge j < n \wedge i + j = n.\ fst\ (hensel-fpxs-aux\ i) * snd$
 $(hensel-fpxs-aux\ j)))$
 $\quad in\ (U * g' + g * ((U * h')\ div\ h),\ (U * h')\ mod\ h)))$

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

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

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

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

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

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

theorem fps-hensel-aux:

defines $f \equiv fps-nth\ F\ 0$

assumes $f = g * h$

assumes $\forall n > 0.\ degree\ (fps-nth\ F\ n) < degree\ f$

defines $G \equiv hensel-fpxs1$ **and** $H \equiv hensel-fpxs2$

shows $F = G * H$ $fps-nth\ G\ 0 = g$ $fps-nth\ H\ 0 = h$

$\forall n > 0.\ degree\ (fps-nth\ G\ n) < degree\ g$

$\forall n > 0.\ degree\ (fps-nth\ H\ n) < degree\ h$

⟨proof⟩

end

end

locale fps-hensel =

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

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

defines $f \equiv reduce-fps-poly\ F$

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

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

begin

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

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 *FPS-Hensel*
begin

3.1 Auxiliary facts and definitions

lemma *div-dvd-self*:
fixes $a b :: 'a :: \{\text{semidom-divide}\}$
shows $b \text{ dvd } a \implies a \text{ div } b \text{ dvd } a$
⟨*proof*⟩

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

lemma *of-int-div-of-int-in-Ints-iff*:
 $(\text{of-int } n / \text{of-int } m :: 'a :: \text{field-char-0}) \in \mathbb{Z} \iff m = 0 \vee m \text{ dvd } n$
⟨*proof*⟩

lemma *rat-eq-quotientD*:
assumes $r = \text{rat-of-int } a / \text{rat-of-int } b \text{ } b \neq 0$

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

$\text{is-fpxs } f \iff \text{bdd-below}(\text{supp } f) \wedge (\text{LCM } r \in \text{supp } f. \text{snd}(\text{quotient-of } r)) \neq 0$

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: ($\bigwedge r. \text{fpxs-nth } f \ r = \text{fpxs-nth } g \ r$) $\implies f = g$
 ⟨proof⟩

lemma fpxs-eq-iff: $f = g \iff (\forall r. \text{fpxs-nth } f \ r = \text{fpxs-nth } g \ r)$
 ⟨proof⟩

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

lemma fpxs-supp-altdef: $\text{fpxs-supp } f = \{x. \text{fpxs-nth } f \ x \neq 0\}$
 ⟨proof⟩

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

lift-definition fpxs-root-order :: 'a :: zero fpxs \Rightarrow nat **is**
 $\lambda f. \text{nat } (\text{LCM } r \in \text{supp } f. \text{snd } (\text{quotient-of } r))$ ⟨proof⟩

lemma fpxs-root-order-pos [simp]: $\text{fpxs-root-order } f > 0$
 ⟨proof⟩

lemma fpxs-root-order-nonzero [simp]: $\text{fpxs-root-order } f \neq 0$
 ⟨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**
 $\lambda f \ n. f \ (\text{of-int } n \ / \ \text{of-int } (\text{LCM } r \in \text{supp } f. \text{snd } (\text{quotient-of } r)))$
 ⟨proof⟩

lemma fls-nth-of-fpxs:
 $\text{fls-nth } (\text{fls-of-fpxs } f) \ n = \text{fpxs-nth } f \ (\text{of-int } n \ / \ \text{of-nat } (\text{fpxs-root-order } f))$
 ⟨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  $\lambda r::rat. \text{if } r = 0 \text{ then } 1 \text{ else } 0$  :: '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
  <proof>

lemma fpxs-nth-add [simp]: fpxs-nth (f + g) r = fpxs-nth f r + fpxs-nth g r
  <proof>

lift-definition fpxs-of-fls :: 'a :: zero fls ⇒ 'a fpxs is
   $\lambda f r. \text{if } r \in \mathbb{Z} \text{ then } f \lfloor r \rfloor \text{ else } 0$ 
  <proof>

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 $fpxs-nth-of-fls$:
 $fpxs-nth\ (fpxs-of-fls\ f)\ r = (if\ r \in \mathbb{Z}\ then\ fls-nth\ f\ \lfloor r \rfloor\ else\ 0)$
 $\langle proof \rangle$

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

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

lemma $fpxs-root-order-of-fls$ [simp]: $fpxs-root-order\ (fpxs-of-fls\ f) = 1$
 $\langle proof \rangle$

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 \Rightarrow rat \Rightarrow 'a fpxs is
 $\lambda f r x.$ 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]: $r > 0 \implies fpxs-compose-power 1 r = 1$
 <proof>

lemma *fls-of-fpxs-eq-0-iff* [simp]: $fls-of-fpxs x = 0 \iff x = 0$
 <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*:
 $r1 > 0 \vee r2 > 0 \implies$
 $fpxs-compose-power (fpxs-compose-power f r1) r2 = fpxs-compose-power f (r1 * r2)$
 <proof>

lemma *fpxs-compose-power-divide-right*:
 $r1 > 0 \implies r2 > 0 \implies$
 $fpxs-compose-power f (r1 / r2) = fpxs-compose-power (fpxs-compose-power f r1) (inverse r2)$
 <proof>

lemma *fpxs-compose-power-1-right* [simp]: $fpxs-compose-power f 1 = f$
 <proof>

lemma *fpxs-compose-power-eq-iff* [simp]:
 assumes $r > 0$
 shows $fpxs-compose-power f r = fpxs-compose-power g r \iff f = g$

⟨proof⟩

lemma *fpxs-compose-power-eq-1-iff* [simp]:
 assumes $l > 0$
 shows $\text{fpxs-compose-power } p \ l = 1 \longleftrightarrow p = 1$
⟨proof⟩

lemma *fpxs-compose-power-eq-0-iff* [simp]:
 assumes $r > 0$
 shows $\text{fpxs-compose-power } f \ r = 0 \longleftrightarrow f = 0$
⟨proof⟩

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

lemma *fpxs-as-fls'*:
 assumes *fpxs-root-order* $f \ \text{dvd} \ d \ d > 0$
 obtains f' **where** $f = \text{fpxs-compose-power } (\text{fpxs-of-fls } f') \ (1 / \text{of-nat } d)$
⟨proof⟩

3.5 Multiplication and ring properties

instantiation *fpxs* :: (comm-semiring-1) comm-semiring-1
begin

lift-definition *times-fpxs* :: 'a fpxs \Rightarrow 'a fpxs \Rightarrow 'a fpxs **is**
 $\lambda f \ g \ x. (\sum (y,z) \mid y \in \text{supp } f \wedge z \in \text{supp } g \wedge x = y + z. f \ y * g \ z)$
⟨proof⟩

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

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

lemma *fpxs-supp-of-fls*: $\text{fpxs-supp } (\text{fpxs-of-fls } f) = \text{of-int } ' \ \text{supp } (\text{fls-nth } f)$
⟨proof⟩

lemma *fpxs-of-fls-mult* [simp]: $\text{fpxs-of-fls } (f * g) = \text{fpxs-of-fls } f * \text{fpxs-of-fls } g$
⟨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* ($f \wedge n$) = *fpxs-of-fls* $f \wedge n$
⟨*proof*⟩

lemma *fpxs-compose-power-power* [*simp*]:
 $r > 0 \implies$ *fpxs-compose-power* ($f \wedge n$) r = *fpxs-compose-power* $f r \wedge n$
⟨*proof*⟩

3.6 Constant Puiseux series and the series X

lift-definition *fpxs-const* :: 'a :: zero \implies 'a *fpxs* is
 $\lambda c n.$ 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
 $\lambda x.$ if $x = 1$ then ($1 :: 'a$) else 0
⟨*proof*⟩

lemma *fpxs-const-altdef*: *fpxs-const* $x =$ *fpxs-of-fls* (*fls-const* x)
⟨*proof*⟩

lemma *fpxs-const-add* [*simp*]: *fpxs-const* ($x + y$) = *fpxs-const* $x +$ *fpxs-const* y
⟨*proof*⟩

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

lemma *fpxs-const-eq-iff* [*simp*]:
 $fpxs-const\ x = fpxs-const\ y \longleftrightarrow x = y$
 ⟨*proof*⟩

lemma *of-nat-fpxs-eq*: $of-nat\ n = fpxs-const\ (of-nat\ n)$
 ⟨*proof*⟩

lemma *fpxs-const-uminus* [*simp*]: $fpxs-const\ (-x) = -fpxs-const\ x$
 ⟨*proof*⟩

lemma *fpxs-const-diff* [*simp*]: $fpxs-const\ (x - y) = fpxs-const\ x - fpxs-const\ y$
 ⟨*proof*⟩

lemma *of-int-fpxs-eq*: $of-int\ n = fpxs-const\ (of-int\ n)$
 ⟨*proof*⟩

3.7 More algebraic typeclass instances

instance *fpxs* :: (*comm-semiring-1*, *semiring-char-0*) *semiring-char-0*
 ⟨*proof*⟩

instance *fpxs* :: (*comm-ring-1*, *ring-char-0*) *ring-char-0* ⟨*proof*⟩

instance *fpxs* :: (*idom*) *idom* ⟨*proof*⟩

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

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

definition *divide-fpxs* :: 'a *fpxs* ⇒ 'a *fpxs* ⇒ 'a *fpxs* **where**
divide-fpxs f g = f * *inverse* g

instance ⟨*proof*⟩

end

instance *fpxs* :: (*field-char-0*) *field-char-0* ⟨*proof*⟩

3.8 Valuation

definition *fpxs-val* :: 'a :: *zero fpxs* ⇒ *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*]: $fpxs-val\ (fpxs-of-fls\ f) = of-int\ (fls-subdegree\ f)$
 ⟨*proof*⟩

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

assumes $r > 0$

shows $\text{fpxs-nth } (\text{fpxs-compose-power } f \ r) \ n = \text{fpxs-nth } f \ (n / r)$

<proof>

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

<proof>

lemma *fpxs-root-order-uminus* [simp]: $\text{fpxs-root-order } (-f) = \text{fpxs-root-order } f$

<proof>

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

<proof>

lemma *fpxs-val-minus-commute*: $\text{fpxs-val } (f - g) = \text{fpxs-val } (g - f)$

<proof>

lemma *fpxs-val-const* [simp]: $\text{fpxs-val } (\text{fpxs-const } c) = 0$

<proof>

lemma *fpxs-val-1* [simp]: $\text{fpxs-val } 1 = 0$

<proof>

lemma *of-int-fls-subdegree-of-fpxs*:

$\text{rat-of-int } (\text{fls-subdegree } (\text{fls-of-fpxs } f)) = \text{fpxs-val } f * \text{of-nat } (\text{fpxs-root-order } f)$

<proof>

lemma *fpxs-nth-val-nonzero*:

assumes $f \neq 0$

shows $\text{fpxs-nth } f \ (\text{fpxs-val } f) \neq 0$

<proof>

lemma *fpxs-nth-below-val*:

assumes $n: n < \text{fpxs-val } f$

shows $\text{fpxs-nth } f \ n = 0$

<proof>

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

<proof>

lemma *fpxs-val-0* [simp]: $\text{fpxs-val } 0 = 0$

<proof>

lemma *fpxs-val-geI*:

assumes $f \neq 0 \wedge r. r < r' \implies \text{fpxs-nth } f \ r = 0$

shows $\text{fpxs-val } f \geq r'$

<proof>

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

assumes $r > 0$

shows $\text{fpxs-val } (\text{fpxs-compose-power } f \ r) = \text{fpxs-val } f * r$
(proof)

lemma *fpxs-val-add-ge*:

assumes $f + g \neq 0$

shows $\text{fpxs-val } (f + g) \geq \min (\text{fpxs-val } f) (\text{fpxs-val } g)$
(proof)

lemma *fpxs-val-diff-ge*:

assumes $f \neq g$

shows $\text{fpxs-val } (f - g) \geq \min (\text{fpxs-val } f) (\text{fpxs-val } g)$
(proof)

lemma *fpxs-nth-mult-val*:

$\text{fpxs-nth } (f * g) (\text{fpxs-val } f + \text{fpxs-val } g) = \text{fpxs-nth } f (\text{fpxs-val } f) * \text{fpxs-nth } g$
($\text{fpxs-val } g$)
(proof)

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 $\text{fpxs-val } (f * g) = \text{fpxs-val } f + \text{fpxs-val } g$
(proof)

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 $\text{fpxs-val } (f \wedge n) = \text{of-nat } n * \text{fpxs-val } f$
(proof)

lemma *fpxs-nth-power-val* [simp]:

fixes $f :: 'a :: \{\text{comm-semiring-1}, \text{semiring-no-zero-divisors}\}$ *fpxs*

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

3.9 Powers of X and shifting

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

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

(proof)

lemma *fpxs-X-power-0* [simp]: *fpxs-X-power* $0 = 1$

(proof)

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

(proof)

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

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

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

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

lift-definition *fpxs-shift* :: $rat \Rightarrow 'a :: zero \ fpxs \Rightarrow 'a \ fpxs$ **is**
 $\lambda r f n. f (n + r)$
 ⟨proof⟩

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

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

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

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

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

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

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

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

lemma *fpxs-shift-shift-uminus' [simp]*: $fpxs\text{-shift} (-r) (fpxs\text{-shift } r f) = f$
 ⟨proof⟩

lemma *fpxs-shift-diff [simp]*: $fpxs\text{-shift } r (f - g) = fpxs\text{-shift } r f - fpxs\text{-shift } r g$

<proof>

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

$$\text{fps-shift } r \ (\text{fps-compose-power } f \ s) = \text{fps-compose-power } (\text{fps-shift } (r / s) \ f)$$

s

<proof>

lemma *rat-of-int-div-dvd*: $d \ \text{dvd} \ n \implies \text{rat-of-int } (n \ \text{div} \ d) = \text{rat-of-int } n / \text{rat-of-int } d$

d

<proof>

lemma *fps-of-fls-shift* [*simp*]:

$$\text{fps-of-fls } (\text{fls-shift } n \ f) = \text{fps-shift } (\text{of-int } n) \ (\text{fps-of-fls } f)$$

<proof>

lemma *fps-shift-mult*: $f * \text{fps-shift } r \ g = \text{fps-shift } r \ (f * g)$

$$\text{fps-shift } r \ f * g = \text{fps-shift } r \ (f * g)$$

<proof>

lemma *fps-shift-1*: $\text{fps-shift } r \ 1 = \text{fps-X-power } (-r)$

<proof>

lemma *fps-X-power-conv-shift*: $\text{fps-X-power } r = \text{fps-shift } (-r) \ 1$

<proof>

lemma *fps-shift-power* [*simp*]: $\text{fps-shift } n \ x^{\wedge} m = \text{fps-shift } (\text{of-nat } m * n) \ (x^{\wedge} m)$

m

<proof>

lemma *fps-compose-power-X-power* [*simp*]:

$$s > 0 \implies \text{fps-compose-power } (\text{fps-X-power } r) \ s = \text{fps-X-power } (r * s)$$

<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 *fps-radical* :: $(\text{nat} \Rightarrow 'a :: \text{field-char-0} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a \ \text{fps} \Rightarrow 'a \ \text{fps}$ where

fps-radical *rt* *r* *f* = (if *f* = 0 then 0 else

(let *f'* = *fls-base-factor-to-fps* (*fls-of-fps* *f*);

f'' = *fps-of-fls* (*fps-to-fls* (*fps-radical* *rt* *r* *f'*))

in *fps-shift* ($-\text{fps-val } f / \text{rat-of-nat } r$)

(*fps-compose-power* *f''* ($1 / \text{rat-of-nat } (\text{fps-root-order } f)$))))

lemma *fps-radical-0* [*simp*]: $\text{fps-radical } rt \ r \ 0 = 0$

<proof>

lemma

fixes $r :: \text{nat}$

assumes $r: r > 0$

shows *fps-power-radical*:

$rt\ r\ (fps\text{-}nth\ f\ (fps\text{-}val\ f))\ \hat{\ }^r = fps\text{-}nth\ f\ (fps\text{-}val\ f) \implies fps\text{-}radical\ rt\ r\ f\ \hat{\ }^r = f$

and *fps-radical-lead-coeff*:

$f \neq 0 \implies fps\text{-}nth\ (fps\text{-}radical\ rt\ r\ f)\ (fps\text{-}val\ f\ / \text{rat-of-nat}\ r) = rt\ r\ (fps\text{-}nth\ f\ (fps\text{-}val\ f))$

<proof>

lemma *fls-base-factor-power*:

fixes $f :: 'a::\{\text{semiring-1}, \text{semiring-no-zero-divisors}\}$ *fls*

shows *fls-base-factor* $(f\ \hat{\ }^n) = fls\text{-}base\text{-}factor\ f\ \hat{\ }^n$

<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 *fpxs* :: (*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 *fpxs* :: (*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 *<proof>*

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 *<proof>*

end

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

definition *uniformity-fpxs-def* [code del]:

(*uniformity* :: ('a fpxs × 'a fpxs) filter) = (INF e ∈ {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 <proof>

end

instance *fpxs* :: (group-add) dist-norm
<proof>

lemma *fpxs-const-eq-0-iff* [simp]: *fpxs-const* x = 0 \longleftrightarrow x = 0
<proof>

lemma *semiring-char-fpxs* [simp]: CHAR('a :: comm-semiring-1 fpxs) = CHAR('a)
<proof>

instance *fpxs* :: ({semiring-prime-char, comm-semiring-1}) semiring-prime-char
<proof>

instance *fpxs* :: ({comm-semiring-prime-char, comm-semiring-1}) comm-semiring-prime-char
<proof>

instance *fpxs* :: ({comm-ring-prime-char, comm-semiring-1}) comm-ring-prime-char
<proof>

instance *fpxs* :: ({idom-prime-char, comm-semiring-1}) idom-prime-char
<proof>

instance *fpxs* :: (field-prime-char) field-prime-char
<proof>

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.