Formalization of a Newton Series Representation of Polynomials

Cyril Cohen, Boris Djalal

Formalization of a Newton Series Representation of Polynomials

Cyril Cohen
Inria Sophia Antipolis – Méditerranée, France
cyril.cohen@inria.fr

Boris Djalal
Inria Sophia Antipolis – Méditerranée, France
boris.djalal@inria.fr

Abstract

We formalize an algorithm to change the representation of a polynomial to a Newton power series. This provides a way to compute efficiently polynomials whose roots are the sums or products of roots of other polynomials, and hence provides a base component of efficient computation for algebraic numbers. In order to achieve this, we formalize a notion of truncated power series and develop an abstract theory of poles of fractions.

Categories and Subject Descriptors D.2.4 [*SOFTWARE ENGINEERING*]: Software/Program Verification —Correctness proofs, Formal methods; F.2.1 [*ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY*]: Numerical Algorithms and Problems—Computations on polynomials; I.1.2 [*SYMBOLIC AND ALGEBRAIC MANIPULATION*]: Algorithms—Algebraic algorithms; I.1.1 [*SYMBOLIC AND ALGEBRAIC MANIPULATION*]: Expressions and Their Representation—Representations (general and polynomial)

Keywords formalization of mathematics, algebraic numbers, fractions, polynomials, Newton power series

1. Introduction

Real algebraic geometry studies points and sets defined by polynomial equations and inequations. Algorithms in real algebraic geometry handle these points and sets in an implicit way and use the defining polynomials as a basis for computations. For example, a real algebraic number is represented by a nonzero polynomial with rational coefficients and a way to select a root (e.g. real approximation, rational interval(Cohen 2012b), Thom encoding). In order to compute basic arithmetic operations on algebraic numbers (sum, product, comparison), we perform operations on these polynomials and need to find a polynomial whose roots are the result of the given operation.

Given two polynomials in $\mathbb{K}[X]$, $P = \sum a_i X^i$ with roots $\alpha_1, \ldots, \alpha_m$ and $Q = \sum b_j X^j$ with roots $\beta_1, \ldots, \beta_n$, we wish to compute a polynomial whose roots are $\alpha_i + \beta_j$ for $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$, which we write $P \oplus Q$ and which we call composed sum. Similarly we define the composed product $P \otimes Q$, whose roots are $\alpha_i \beta_j$ for $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$.

If the base field $\mathbb{K}$ is algebraically closed (i.e. in which every non constant polynomial has a root), we can split the polynomials $P = a_n \prod_i (X - \alpha_i)$ and $Q = b_m \prod_j (X - \beta_j)$, and compute the results

$$P \oplus Q = a_m b_n \prod_{(i,j)} (X - (\alpha_i + \beta_j)).$$

$$P \otimes Q = a_m b_n \prod_{(i,j)} (X - (\alpha_i \cdot \beta_j)).$$

Given an arbitrary field $\mathbb{K}$, folklore mathematics textbooks usually take an algebraic closure or splitting fields and performs the construction above. However, in order to take the algebraic closure, one needs to already have a way to perform the composed sums and products, so this does not solve the problem. In fact, these operations may rely on purely algebraic computations on the coefficients, and thus do not require a preexisting algebraic closure to perform the computation.

In prior work by the first author (Cohen 2012b), we provide a way to perform these operations and build the algebraic closure. However the underlying algorithms for composed sums and products use resultants, which is not an efficient implementation and would not allow for a practical implementation of algebraic numbers. An efficient way relies on a morphism between polynomials and Newton power series, as described by A. Bostan in (Bostan 2003), which is a main reference for the computer algebra community. Composed sums and products on polynomials are mapped to low-cost operations on Newton power series. In the case of zero characteristic, the new algorithm improves the complexity by a linear factor (Bostan 2003). If n denotes the degree of two input polynomials and $M(n)$ the costs to multiply two polynomials of degree n, then the traditional algorithm based on bivariate resultant has asymptotic complexity $\mathcal{O}(nM(n)\log(n))$ while the new algorithm has complexity $\mathcal{O}(M(n^2))$ (Bostan 2003). In this work we use our prior knowledge of the existence of an algebraic closure (even though it is not efficient) as a reference implementation to certify our new results. Indeed, all the algorithms we describe use the coefficients in the base field, but all the proofs are made by supposing one can split the polynomials in a field extension. The existence of the algebraic closure provides grounding for this assumption and thus serves as a bootstrap.

In this paper, we describe a Coq formalization of an isomorphism between monic polynomials of a bounded degree and a truncated version of formal power series.

In order to explain and formalize this isomorphism we first introduce the general mathematical notions we had to formalize (Section 2). The truncated formal power series (Section 2.1) is an approximation of traditional power series, better suited for use in Coq with the Mathematical Components library. We also need to build the fraction field of the domain of polynomials in order to compute the isomorphism, and instead we first give an

abstract interface for poles and evaluation of fraction (Section 2.2) which we implement twice.

Throughout Section 3, we describe some algorithms on polynomials on a field and prove them correct with regard to another, more concise definition in an algebraic closure. We explain how to compute the Newton power series in the context of algebraically closed fields (Section 3.1). Then we describe an algorithm to compute the Newton power series without making operations inside an algebraic closure and we prove it computes the same result as in the previous section (Section 3.2). We also provide an explicit inverse, which computes a polynomial from a Newton power series. Finally, we explain how to compute the composed sum and product using a translation to a Newton power series, a simple computation on the formal power series and then a backward translation.

The results described in this paper are entirely formalized in Coq, unless explicitly stated. The formal development is available on the first author webpage:
http://www.cyrilcohen.fr/work/newtonsums/

2. Mathematical background

2.1 Formal power series and truncated formal power series

In this paper, we make statements indirectly involving formal power series (FPS), the common mathematical object noted $\mathbb{K}[[X]]$. FPS is a generalization of polynomials: it is an infinite sequence of coefficients $(a_i)_{i\in\mathbb{N}}$. Unlike polynomials, the set of non-zero coefficients is not necessarily finite. For example, $1+X^2$ is a polynomial (thus a FPS) and $S = 1 + X + X^2 + X^3 + \dots$ is the FPS with general term X^i. In the general case, coefficients are taken in a commutative ring.

The set of formal series $R[[X]]$ over a commutative ring R has a structure of commutative ring. In this work, we study formal series $\mathbb{K}[[X]]$ over a field $\mathbb{K}$ of characteristic zero, which means that for any natural number n:

$$\sum_{i=1}^{n} 1_{\mathbb{K}} \neq 0.$$

In Coq we could implement FPS by functions from nat to the given ring, as it is done by A. Chaieb in (Chaieb 2011). In this case, comparing two FPS amounts to proving that two given functions are equal. This approach has two drawbacks: equality becomes undecidable (more precisely, disequality is semi-decidable) and without the functional extensionality axiom two FPS with equal terms are not provably equal. While we would not mind being in a context where the second axiom is validated (e.g. Homotopy Type Theory (Univalent Foundations Program 2013)) the algebraic library we use extensively in our work requires a decidable equality.

Instead of FPS, we consider an approximation we call truncated formal power series up to X^m (TFPS$_m$ or TFPS if there is no ambiguity), noted $\mathbb{K}_m[X]$. In this case, we deliberately provide only $m+1$ coefficients. The set of TFPS$_m$ is isomorphic to the set of polynomials quotiented by X^{m+1}. Consequently, we can implement a TFPS by a polynomial whose degree is less than m, i.e. by a polynomial together with a proof that its degree is less than m. We use polynomials from the Mathematical Components library, already used for many results (algebraic numbers, Galois theory, Cayley Hamilton theorem, odd order theorem (Gonthier et al. 2013), ...).

```
Record tfps := TFPS
{
  truncation_tfps :> {poly K};
  _ : size truncation_tfps <= n.+1
}.
```

We write `{tfps K n}` for the formal power series over `K` truncated up to precision X^n (included).

We can use the following fact to turn any polynomial into a TFPS$_m$ for any m, and build a smart constructor `Tfpsp` which turns any polynomial to a TFPS$_m$.

```
Fact leq_modpXn m p : size (p %% 'X^m) <= m.

Definition Tfpsp m : {poly K} -> {tfps K m} :=
  fun p => TFPS (leq_modpXn m.+1 p).
```

We also provide a construction to define a TFPS$_m$ from its coefficients:

```
Notation "[tfps s => E]" :=
   Tfpsp m (\poly_(i < m.+1) E)
```

2.1.1 Arithmetic properties

The decision procedure for equality consists in comparing the underlying polynomial representations. The addition of two polynomials whose degrees are less than m produces a polynomial whose degree is less than m. Thus, we define addition of two TFPS as the addition of the underlying polynomials.

In order to multiply two TFPS$_m$, we multiply the underlying polynomials, we can get a polynomial with degree more than $m+1$. We then take the remainder modulo X^{m+1} to guarantee that the degree of the obtained polynomial is less than m. This makes sense since the coefficients from $m+1$ are only partially known (we are missing the information from the rest of the series).

Our definition of multiplication of two TFPS in Coq is as follows:

```
Definition mul_tfps (f1 f2 : {tfps K m})
 : {tfps K m}
 := Tfpsp (f1 * f2).
```

For example in $\mathbb{K}_2[X]$, we have: $(1+X^2)\times X = X$.

Additionally, the Hadamard product of two FPS is the term-wise product of the FPS. Formally we write:

```
Definition hmul_tfps (f1 f2 : {tfps K m})
 : {tfps K m}
 := [tfps s => f1`_s * f2`_s]
```

For example in $\mathbb{K}_2[X]$, we have:

$$(1+X^2)\odot(1+X+2X^2) = 1+2X^2.$$

FPS over an integral domain forms an integral domain. This is not true with TFPS as we defined them. Indeed, in TFPS$_3$

$$X^2\cdot X^2 = 0 \pmod{X^4}$$

This is due to the fixed precision of our representation. We could opt for a "floating truncation" FPS$_m$, where m would actually refer to the number of meaningful terms, *i.e.* the representation of these series would be of the form

$$X^M \sum_{0\leq s\leq m} a_s X^s$$

with a_0 nonzero. Moreover, allowing M to be negative would lead to a representation of Laurent formal series instead.

Derivative

We can translate any statement about FPS into a statement about TFPS. Let's consider the additivity of formal derivation as an example. In terms of FPS, it expresses as: $\forall f,\ g\in\mathbb{K}[[X]]\ (f+g)' = f'+g'$. In terms of TFPS, it expresses as: $\forall m\in\mathbb{N}, \forall f,\ g\in\mathbb{K}_m[X]\ (f+g)' = f'+g'$. Here, we just have replaced occurrences of FPS by TFPS. However, the translation of a statement

dealing with FPS into a statement dealing with TFPS can be trickier. This is the case for the definition of the derivative of a TFPS (see 2.1.1). We formally prove that $\mathbb{K}_m[X]$ is a ring. The relationships between FPS and TFPS is summarized by the following diagram:

$$\mathbb{K}[[X]] \xrightarrow{\lfloor . \rfloor_m} \mathbb{K}_m[X] \twoheadrightarrow \mathbb{K}_n[X]$$

where $n \leq m$, $\twoheadrightarrow$ denotes a surjective ring morphism, and $\lfloor . \rfloor_m$ denotes the function which sends a FPS (or a polynomial) to its truncation in $\mathbb{K}_m[X]$. Please note that in our Coq development we chose only to deal with TFPS, not FPS. One can mathematically define the derivative, primitive, exponential and logarithm of FPS. All these operations are required to formalize the algorithm which changes the representation of a polynomial to a Newton TFPS. We are going to define such operations on TFPS by adapting their equivalent definitions on FPS.

The derivative of a polynomial is defined as follows. For $P \in \mathbb{K}[X]$,

$$P' = \sum_{i=1}^{m} i \cdot a_i \, X^{i-1} = \sum_{i=0}^{m-1} (i+1) \cdot a_{i+1} \, X^i.$$

The notion of derivative on polynomials is easily extended to FPS. For $P \in \mathbb{K}[[X]]$,

$$P' = \sum_{i=1}^{\infty} i \cdot a_i \, X^{i-1} = \sum_{i=0}^{\infty} (i+1) \cdot a_{i+1} \, X^i.$$

In Coq we write:

```
Definition deriv_tfps : {tfps K n} -> {tfps K n.-1}
  := fun f => [tfps s => f`_s.+1 *+ s.+1]
```

Note that derivation must be a map from $\mathbb{K}_m[X]$ to $\mathbb{K}_{m-1}[X]$, not a map from $\mathbb{K}_m[X]$ to $\mathbb{K}_m[X]$. Indeed, when we take derivative, we lose precision. Otherwise, the following commutative diagram would be broken:

$$\begin{array}{ccc} \mathbb{K}[[X]] & \xrightarrow{derivative} & \mathbb{K}[[X]] \\ \downarrow \lfloor . \rfloor_m & & \downarrow \lfloor . \rfloor_{m-1} \\ \mathbb{K}_m[X] & \xrightarrow[derivative]{} & \mathbb{K}_{m.-1}[X]. \end{array}$$

Primitive

We define the canonical primitive of a polynomial as the only primitive such that the constant term is zero.

$$\int P = \int \sum_{i=0}^{m} a_i \, X^i = \sum_{i=0}^{m} \frac{a_i}{i+1} \, X^{i+1}.$$

```
Definition prim (p : {poly K}) :=
  \poly_(i < (size p).+1)
    (p`_i.-1 *+ (i != 0) / (i%:R)).
```

From the canonical primitive, we can derive the family of primitive functions for each possible constant term in the ring by adding the constant a, as in `(a + prim p)`.

Like for the derivative, these definitions naturally extend to a FPS.

$$\int \sum_{s=0}^{\infty} a_s \, X^s = \sum_{s=0}^{\infty} \frac{a_s}{s+1} \, X^{s+1}.$$

We formally define primitive for polynomial and for TFPS.

```
Definition prim_tfps : {tfps K n} -> {tfps K n.+1}
  := fun f => [tfps s => f`_s.-1 *+ (s != 0) / s%:R].
```

Note that the primitive must be a map from $\mathbb{K}_m[X]$ to $\mathbb{K}_{m+1}[X]$, not a map from $\mathbb{K}_m[X]$ to $\mathbb{K}_m[X]$, for exactly the same reason as for derivation.

We prove basic theorems linking derivative with primitive:

$$\left(\int P \right)' = P$$

$$\int (P') = P - a_0$$

Our corresponding Coq pieces of code of these two statements are:

```
Lemma prim_tfpsK (n : nat) :
  cancel (@prim_tfps n) (@deriv_tfps K n.+1).

Lemma deriv_tfpsK n (f : {tfps K n.+1}) :
  {in @coef0_is_0 K n.+1,
     cancel (@deriv_tfps _ _) (@prim_tfps _)}.
```

Composition

The composition of FPS is not always well defined: given two $P = \sum_{i=0}^{\infty} a_i \, X^i$ and $Q = \sum_{j=0}^{\infty} b_j \, X^j$, for the composition to be well-defined it is sufficient that $b_0 = 0$ (cf (Wikipedia)). We can then write the composition of P et Q in the following way

$$P \circ Q = \sum_{i=0}^{\infty} a_i \, (\sum_{j=0}^{\infty} b_j X^j)^i.$$

We could also define the composition with explicit coefficients:

$$P \circ Q = \sum_{k=0}^{\infty} c_k X^k$$

with

$$c_k = \sum_{\substack{i \in \mathbb{N},\, s \in \mathbb{N}^i \\ \sum_{j \in I_i} s_j = k}} \left(a_i \prod_{j \in I_i} b_{s_j} \right)$$

where $I_i = \{1, \ldots, i\}$.

The coefficient c_k is well-defined whenever $b_0 = 0$, since the sum and each product is finite. Indeed, for any k, if there is a j such that $s_j = 0$ then $\prod_{j \in I_i} b_j = 0$, so we can sum over only the sequences $s \in (\mathbb{N}^*)^i$.

Since $\sum_{j \in I_i} s_j = k$ and $\forall j \in I_i, s_j > 0$, we have $i \leq k$. Hence the sum indexed by $i \in \mathbb{N}$ and $s \in \mathbb{N}^i$ such that $\sum_{j \in I_i} s_j = k$ is indeed finite. This proves that c_k is a finite sum and is well-defined.

The formula above works also for $k = 0$. In this case the only possible i is 0. It leads to:

$$c_0 = a_0 \prod_{j \in \emptyset} b_{s_j} = a_0 \times 1 = a_0$$

which is the expected result. Formally proving the general formula for the coefficient of the composition of two TFPS is the subject of an ongoing work.

Composition for polynomials is already defined in the MATHEMATICAL COMPONENTS library. We formally define this notion when formal power series are truncated, by directly defining the composition in terms of polynomials. We use this opportunity in our COQ code:

```
Definition comp_tfps m (q p : {tfps K m}) :=
  if q \in (coef0_is_0 K m)
    then Tfpsp m (comp_poly q p) else 0.

Notation "p \So q" := (comp_tfps q p).
```

We did not yet prove in COQ the formula with explicit coefficients, nor did we prove the existence of the compositional inverse when the coefficient of X^0 is 0 and the coefficient of X^1 is 1. Indeed, this could have eased the definition of exponential and logarithm, but it is not used in our developments yet (it is not required to define Newton Power Series).

Exponential

There are two main ways to define the exponential of a FPS.

We can see the exponential as the element of $\mathbb{K}[[X]]$ defined by

$$\exp = \sum_{i=0}^{\infty} \frac{X^i}{i!}.$$

Note that here we use the hypothesis that $\mathbb{K}$ has zero characteristic to guarantee $i! \neq 0$. This is commonly expressed in MATHEMATICAL COMPONENTS by the following hypothesis.

```
Hypothesis char_K_is_zero : [char K] =i pred0.
```

We can also view this exponential as a function from FPS with zero constant to FPS with constant 1, defined by:

$$f_{\exp}(P) = \sum_{i=0}^{\infty} \frac{P^i}{i!}.$$

This definition is related to the first one by:

$$f_{\exp}(P) = \exp \circ P.$$

In our development, we use the former definition adapted to TFPS:

```
Definition exp (p : {tfps K n}) : {tfps K n} :=
  if p \notin coef0_is_0 then 0 else
  Tfpsp (\sum_(i < n.+1) ((i`!%:R)^-1) *: (p ^+ i))).
```

We formally prove that the formal exponential is a morphism: for all P and Q such that $P(0) = Q(0) = 0$, $\exp(P+Q) = \exp(P)\exp(Q)$. The corresponding COQ code statement is:

```
Lemma exp_is_morphism :
  {in (@coef0_is_0 K m) &,
  {morph (@exp _ _) : p q / p + q >-> p * q}}.
```

We formally prove the formula linking the exponential and its derivative, which states, for any formal power series P:

$$(\exp P)' = P' \exp(P).$$

In our COQ code, this formula is expressed as:

```
Lemma deriv_exp (m : nat) (p : tfps K m) :
  (exp p)^` () = (p^` ()) * (Tfpsp m.-1 (exp p)).
```

It would be simplier if exp were defined as a TFPS, as we could write $\exp' = \exp$ and prove the previous theorem as a trivial application of the derivation of composition theorem. Moreover, injectivity could be obtained by the existence of compositional inverses under the right conditions.

Logarithm

As for exponential, logarithm can be defined both as a function or as a FPS. For any P such that $P_0 = 1$:

$$\log P = -\sum_{i=1}^{\infty} \frac{(1-P)^i}{i}.$$

This series is well-defined whenever $P_0 = 1$ for the same reason as for the composition. In our development, we use the latter definition adapted to TFPS:

```
Definition log (p : {tfps K n}) :=
  if p \notin coef0_is_1 then 0
  else Tfpsp n
  (- \sum_(1 <= i < n.+1)
    ((i %:R) ^-1) *: ((1 - val p) ^+i)).
```

We formally prove:

$$(\log P)' = \frac{P'}{P}.$$

We expressed the lemma about the derivative of the formal logarithm on TFPS with:

```
Lemma deriv_log (m : nat) (p : tfps K m) :
    p \in (coef0_is_1 K m) ->
   (log p) ^` () = (p )^` () / (Tfpsp m.-1 p).
```

We prove that $\log(\exp P) = P$ on TFPS formally by derivating the expression and using the lemma about the derivative of log. It is possible to obtain this results from logarithm injectivity. But in our development, we use the derivative of the logarithm formula to prove the injectivity of the logarithm function. The injectivity of the formal logarithm on TFPS expresses as:

```
Lemma log_inj :
  {in coef0_is_1 K n &, injective (@log K n)}.
```

We prove the injectivity of the logarithm as follows. First, we notice that the derivative formula of a division holds for TFPS. For any TFPS P and Q such that $Q \neq 0$, we have:

$$\left(\frac{P}{Q}\right)' = \frac{P'Q - QP'}{Q^2}.$$

Let be TFPS P et Q such that $P(0) = 1$ and $Q(0) = 1$, and let's suppose that we have $\log(P) = \log(Q)$. Since $Q(0) = 1$, we have $Q \neq 0$. We are going to prove that $P = Q$ and thus the injectivity of the logarithm. By derivating the hypothesis, we get: $\frac{P'}{P} = \frac{Q'}{Q}$ which rewrites as $P'Q - QP' = 0$ then $\frac{P'Q-QP'}{Q^2} = 0$ (since $Q \neq 0$). Using our first remark, we thus get $\left(\frac{P}{Q}\right)' = 0$. It implies that $\frac{P}{Q}$ is a constant. In other words, the underlying polynomials of P and Q are associate polynomials. Since $P(0) = Q(0)$, we finally get $P = Q$. This achieves the proof of the injectivity of the formal logarithm on TFPS.

We could simplify some proof by getting the logarithm of a series as a composition for series. Indeed we could define $\log(1-X)$ as a series:

$$\mathcal{L} = -\sum_{i=1}^{\infty} \frac{X^i}{i}.$$

So that $\log P = \mathcal{L} \circ (1-P)$, which would be well-defined whenever the constant coefficient of P is 1. Moreover, injectivity could be obtained by the existence of compositional inverses under the right conditions.

2.2 Fractions

The main theorem about Newton power series requires polynomial fractions. We use a more general result about the field of fractions of an integral domain and $\mathbb{K}(X)$ is constructed as the field of fractions of $\mathbb{K}[X]$. The construction of polynomial is correct because $\mathbb{K}[X]$ is an integral domain whenever $\mathbb{K}$ is a field. Let R be an integral domain and $\iota : R \hookrightarrow \mathcal{F}(R)$ the (canonical) injection to its field of fractions.

We not only want to manipulate fractions, we also want to define functions from $\mathcal{F}(R)$ to another ring without having to go back to the implementation of $\mathcal{F}(R)$ as $R \times R$ quotiented by an equivalence relation.

In this section we show an interface for an abstract notion of poles of fraction. We first show two use-cases of this interface. We then establish the relationship between this interface and the universal property of the field of fractions.

Here we present the first use-case, which is the evaluation of polynomial fractions. Let's consider the following examples:

- the evaluation of $\dfrac{X^2-2}{X+5}$ in 3 is $\dfrac{7}{8}$,
- the evaluation of $\dfrac{X^2-2}{X-3}$ in 3 is not defined because 3 is a pole,
- the evaluation of $\dfrac{X^2-3X}{X^2-X-6} = \dfrac{X}{X+2}$ in 3 is $\dfrac{3}{5}$.

An evaluation algorithm proceeds by first deciding whether there is a good representation for our fraction. Then, there are two mutually exclusive possibilities:

- evaluation is not defined
- it boils down to evaluating two polynomials and performing a division.

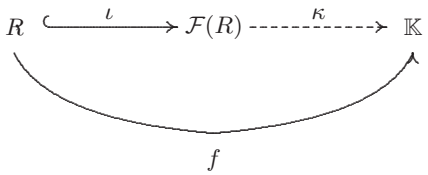
A unifying interface via abstract poles of fractions

Let R be an integral domain and $\iota : R \hookrightarrow \mathcal{F}(R)$ the canonical injection to its field of fractions. Let K be a field and $f : R \to K$ a morphism. For any a in K, we say x has a regular representation in a if x can be written as $\frac{u}{v}$ with $f(v) \neq a$. We use this definition of regular representation for $a = 0$.

We define

$$\kappa(x) = \begin{cases} \frac{f(u)}{f(v)} & \text{if there is a regular representation } \frac{u}{v} \text{ of x in 0} \\ \text{undefined} & \text{otherwise (we return 0 in Coq) .} \end{cases}$$

where we say x has a regular representation in 0 if x can be written as $\frac{u}{v}$ with $f(v) \neq 0$.



(1)

Our κ is mathematically well-defined, but not always computable. The computability of κ is guaranted when these three points are satisfied: f is computable, for all x, it is decidable whether there is a regular representation of x and we can compute it when it exists.

Application to the evaluation of polynomial fractions. Let $\mathbb{K}$ be a field.

- R is the ring of polynomials $\mathbb{K}[X]$
- $\mathbb{K}(X)$ is the field of fractions of R, noted $\mathcal{F}(R)$
- f: $R \longrightarrow \mathbb{K}$ is the evaluation of polynomials in a

The evaluation of polynomial fractions in a is the map:

$$\kappa : \mathcal{F}(R) \longrightarrow \mathbb{K}$$

$$\kappa(x) = \begin{cases} \frac{f(u)}{f(v)} & \text{if } x \text{ can be written as } \frac{u}{v} \text{ with } f(v) \neq 0 \\ undefined & \text{otherwise.} \end{cases}$$

Note that f is parameterized by an element $a \in \mathbb{K}$.

Application to the lifting of a the field of coefficients. Let $\mathbb{F}$ and $\mathbb{L}$ denote fields and $\mathbb{L}/\mathbb{F}$ be a field extension, where $\iota : \mathbb{F} \hookrightarrow \mathbb{L}$ is the canonical injection. We assume that we know how to lift from $\mathbb{F}[X]$ to $\mathbb{L}(X)$. We want to lift any element of $F \in \mathbb{F}(X)$ to $\mathbb{L}(X)$.

We reuse or interface where

- $\mathbb{F}[X]$ is the integral domain R
- $\mathbb{F}(X)$ is the field of fractions of R, noted $\mathcal{F}(R)$
- $\mathbb{L}(X)$ is a field $\mathbb{K}$
- f: $R \longrightarrow \mathbb{K}$ is the lifting from $\mathbb{F}[X]$ to $\mathbb{L}(X)$
- our lifting function from $\mathbb{F}(X)$ to $\mathbb{L}(X)$ is the map: κ : $\mathcal{F}(R) \longrightarrow \mathbb{K}$

$$\kappa(x) = \begin{cases} \frac{f(u)}{f(v)} & \text{if } x \text{ can be written as } \frac{u}{v} \text{ with } v \neq 0 \\ undefined & \text{otherwise.} \end{cases}$$

Note that since here f is injective, $v \neq 0$ implies $f(v) \neq 0$. Thus, κ is always defined.

Relation to the universal property of fractions

When f is injective, it is basically the universal property of the field of fractions of an integral domain. This universal property states: for any field $\mathbb{K}$ and injective ring morphism f from R to $\mathbb{K}$, there is a unique ring morphism κ from the field of fractions $\mathcal{F}(R)$ to $\mathbb{K}$ such that our diagram (1) commutes.

However the function κ is defined even when f is not injective — we call this broader definition of κ an abstract evaluation. When f is not injective, κ fails to be a ring morphism. We develop abstract evaluation theory in the general case i.e. even when f is not injective. We define the notion of abstract pole by:

$$x \in \mathcal{F}(R) \text{ has a } f\text{-pole if: } \forall u\, v \in R \text{ such that } x = \frac{\iota(u)}{\iota(v)}, \\ f(v) = 0.$$

The presence of an abstract pole, or f-pole, in x means that there is no regular representation of x. It is expressed as follows in our Coq code:

```
Definition has_pole x := forall y : R * R,
    x = y.1%:F / y.2%:F -> f y.2 = 0.
```

We then suppose that any $x \in \mathcal{F}(R)$ either has a f-pole or can be written $x = \frac{\iota(u)}{\iota(v)}$ for some $u\, v \in R$ such that $f(v) \neq 0$.

We derive formally the following results:

- $\kappa(0) = 0$
- $\kappa(1) = 1$
- $\forall x \in \mathcal{F}(R)$, $\kappa(-x) = -\kappa(x)$
- $\forall x, y \in \mathcal{F}(R)$, $\kappa(y) \neq 0 \implies \kappa(\frac{x}{y}) = \frac{\kappa(x)}{\kappa(y)}$

The latter result is expressed as:

```
Lemma kappa_div (y z : {fraction R}) :
    kappa z != 0 ->
    kappa (y / z) = (kappa y) / (kappa z).
```

Extra results are provided in the interface when ι is injective. When ι is injective, the main and summarizing result is that abstract evaluation is a ring morphism.

We use the universal property of fractions — lifting from $\mathbb{F}(X)$ to $\mathbb{L}(X)$ — in the particular situation where $\mathbb{F} = \mathbb{L}(Y)$ for a second formal variable Y. We use it to define lifting from $\mathbb{F}(X)$ to $(\mathbb{F}(Y))(X)$. Here, R is $\mathbb{F}[X]$, $\mathbb{K}$ is $\mathbb{F}(X)$ and f is the lifting function from $\mathbb{F}[X]$ to $(\mathbb{F}(Y))[X]$. Then, we derive lifting of polynomial fractions automatically : this implementation helps defining composition of fractions of polynomials. Let U and $V \in \mathbb{F}(X)$. The composition of U and V is defined by: $U \circ V = (\kappa(U))(V)$. That is, we lift U. Then we swap variables X and Y and evaluate the resulting polynomial in V.

Taylor series of polynomial fractions

In Section 3.2 we use the concept of Taylor series of polynomial rational, which is well defined for the set of elements of $\mathbb{K}(X)$ such that the denominator of an irreducible representation has got a nonzero constant. For example, $\frac{1}{X}$ has no Taylor series and $\frac{1}{1-X}$ has the following Taylor series

$$\sum_{i=0}^{\infty} X^i = 1 + X + X^2 + X^3 + \dots.$$

Note that $\frac{X^2}{X-X^2}$ has a Taylor series despite the fact that the constant term of $X - X^2$ is zero. If 0 is a pole of the fraction, we know there is no Taylor series. Otherwise, there is a representation $\frac{U}{V}$ of the fraction where the constant coefficient of the denominator V is nonzero. Then we find the Taylor series S of $\frac{U}{V}$ by solving the system deriving from the equation $S \cdot V = U$ (in this equation U and V cast to FPS). Sum and product of fractions preserve the existence of a Taylor series, hence fractions which have a Taylor series form a sub-ring of the field of polynomial fractions. We use the very same concept of Taylor series, adapted to TFPS. Instead, we output a TFPS with the desired precision.

We could also reuse the generalized universal property of the field of fractions, in two possible different ways. The first way would require a generalization of the interface for the generalized universal property of the field of fractions with a morphism which target is in an integral domain instead of a field. Then we consider the morphism that casts a polynomial inside the integral domain of "floating truncation" FPS. A second way would be to reuse our interface with a morphism which target is truncated Laurent formal series, and show that when the source fraction has no pole, the Laurent formal series has a nonnegative order:

$$X^M \sum_{0 \le s \le m} a_s X^s$$

with $M \ge 0$.

3. Newton Power Series

The mathematical theory of this section is entirely based on (Bostan 2003). A Newton power series is a FPS which represents a monic polynomial (polynomial with leading coefficient equal to one) without loss of information. In pratice, it is enought to use truncated Newton power series, because only n coefficients of the formal series are used to recover the n coefficients of a monic polynomial of degree n.

Throughout this section, we take the running example of two polynomials $P = X^2 - 2$ and $Q = X + 3$ we compute the composed sum and product of.

3.1 Newton series in an algebraic closed field

Let $\mathbb{K}$ be a field and $\mathbb{L}$ an algebraically closed extension of $\mathbb{K}$, where $\iota : K \hookrightarrow L$ is the injective morphism from $\mathbb{K}$ to $\mathbb{L}$. Let $P = \sum_{i \le m} a_i X^i$ be a polynomial of $\mathbb{K}[X]$. We write abusively $\iota(P) = \sum_{i \le m} \iota(a_i) X^i$ the lifting of P to $\mathbb{L}[X]$ and we write `p ^ iota` in Coq.

Since $\mathbb{L}$ is algebraically closed, $\iota(P)$ has roots $\alpha_i \in L$ for $i \in \{1, \dots, m\}$, such that $\iota(P) = \iota(a_m) \prod_i (X - \alpha_i)$. Note that, in common mathematical practice, we would simply write $P = a_m \prod_i (X - \alpha_i)$, the injection ι being implicit.

With our example, we have $X^2 - 2 = (X - \sqrt{2})(X + \sqrt{2})$ and $X + 3 = X - (-3)$. And the results of the composed sum and products should be

$$\begin{aligned}(X^2 - 2) \oplus (X + 3) &= \left(X + \left(3 - \sqrt{2}\right)\right)\left(X + \left(3 + \sqrt{2}\right)\right) \\ &= X^2 + 6X + 7 \\ (X^2 - 2) \otimes (X + 3) &= \left(X + 3\sqrt{2}\right)\left((X - 3\sqrt{2}\right) \\ &= X^2 - 18\end{aligned}$$

We can define

$$N_s(P) = \sum_{i \le m} \alpha_i^s \tag{2}$$

By convention, the sum is zero if P has no root. This can be formalized using the library *bigop* (Bertot et al. 2008) of Mathematical Components libraries, which provides a theory of finite iterations of an operation on a monoid.

With our example, we have:

$$\begin{aligned}N_s(X^2 - 2) &= \sqrt{2}^s + \left(-\sqrt{2}\right)^s \\ &= \begin{cases} 2^{k+1} & \text{if } s = 2k \\ 0 & \text{otherwise} \end{cases} \\ \text{and} \quad N_s(X + 3) &= (-3)^s\end{aligned}$$

The Newton power series of P is the following FPS :

$$\mathcal{N}(P) = \sum_{s=0}^{\infty} N_s(P)\, X^s \tag{3}$$

We call forward transformation the process to obtain $\mathcal{N}(P)$ from P. With our example, we have:

$$\mathcal{N}(X^2 - 2) = 2 + 4X^2 + 8X^4 + \sum_{k>2} 2^{k+1} X^{2k}$$

$$\mathcal{N}(X + 3) = 1 - 3X + 9X^2 + \sum_{k>2} (-3)^k X^k$$

The Newton power series $\mathcal{N}(P)$ and the Newton sums $N_s(P)$ are respectively in $\mathbb{L}[[X]]$ and $\mathbb{L}$. However, each $N_s(P)$ is a symmetric function of the roots of P: each $N_s(P)$ remains unchanged when we apply a permutation to the roots of P. Thus, by the fundamental theorem of symmetric polynomials, $N_s(P)$ can be expressed as a polynomial expression of elementary symmetric functions of the roots of P, which are the coefficients of P (up to a sign). This means $N_s(P)$ can be expressed as a polynomial in the coefficients of P and is hence in $\mathbb{K}$ instead of $\mathbb{L}$. Hence $\mathcal{N}(P)$ can be written as a formal power series over $\mathbb{K}$. Whereas common mathematical practice does not observe the distinction, in Coq we have to be specific and we cannot write the definition of Newton sums as in (2) and Newton power series as in (3). There are two options: using the theory of symmetric polynomials we could have proved that the Newton series are in the sub-field $\mathbb{K}$ of $\mathbb{L}$ and extracted the

corresponding elements, or we can provide an alternative definition of Newton series that makes it obvious they belong to $\mathbb{K}$. Since this alternative definition is also the one used for efficient computation, we chose to define Newton sums using the latter.

The remainder of this section gives a concrete direct representation of Newton sums as a value in $\mathbb{K}$ and hence the truncated power series `newton_tfps` representing the series $\mathcal{N}(P)$, but in $\mathbb{K}[[X]]$ instead of $\mathbb{L}[[X]]$.

With the definition in Section 3.2, we eventually prove that

```
forall (p : {poly K}) (n : nat),
  (newton_tfps n p) ^ iota =
  [tfps j => \sum_(r <- roots (p ^ iota)) r ^+ j].
```

Note that in fact, $\mathbb{L}$ need not be algebraically closed, but a splitting field of P. However, it is sufficient for our development to consider an algebraically closed extension, because we can actually provide the algebraic closure of the fields we consider (Cohen 2012b).

We now explain how to define `newton_tfps` as a series with coefficients in $\mathbb{K}$.

3.2 Newton power series in a general setting

We now show how to compute $\mathcal{N}(P)$ directly from the coefficients of P, without the need for factoring P. Then, we will present a procedure to recover P from $\mathcal{N}(P)$, where P is monic (*i.e.* with a leading coefficient equal to 1, which implies $P \neq 0$). For any polynomial P, the forward transformation theorem holds:

$$\mathcal{N}(P) = \frac{\mathrm{rev}(P')}{\mathrm{rev}(P)} \tag{4}$$

where rev is the reverse operator on polynomials, which outputs a polynomial by reversing the list of coefficients of the input polynomial. For instance: $\mathrm{rev}(1 + 2X + 3X^2) = 3 + 2X + X^2$, $\mathrm{rev}(X^3 - X^5) = -1 + X^2$, $\mathrm{rev}(X^2 - 2) = 1 - 2X^2$, $\mathrm{rev}(2X) = 2$, $\mathrm{rev}(X + 3) = 1 + 3X$ and $\mathrm{rev}(1) = 1$.

In the equation above P is in $\mathbb{K}[X]$ and $\mathcal{N}(P)$ is in $\mathbb{K}(X)$. The constant coefficient of $\mathrm{rev}(P)$ is nonzero because $P \neq 0$, thus $\frac{\mathrm{rev}(P')}{\mathrm{rev}(P)}$ has a Taylor series, which is a FPS in $\mathbb{K}[[X]]$.

With our example we have

$$\frac{\mathrm{rev}(2X)}{\mathrm{rev}(X^2 - 2)} = \frac{2}{1 - 2X^2} = 2\sum_{s\geq 0}(2X^2)^s = \sum_{s\geq 0} 2^{s+1}X^{2s}$$

$$\frac{\mathrm{rev}(1)}{\mathrm{rev}(X + 3)} = \frac{1}{1 + 3X} = \sum_{s\geq 0}(-3X)^s = \sum_{s\geq 0}(-3)^s X^s$$

Hence we define `newton` as a rational fraction in $\mathbb{K}(X)$ and `newton_tfps` as a TFPS with coefficients in $\mathbb{K}$.

```
Definition newton (p : {poly K}) : {fracpoly K} :=
  rev (deriv p) // rev p.

Definition newton_tfps (m : nat) (p : {poly K}) :=
  Tfpsfp m (newton p).
```

The proof of the equivalence of the definitions (3) and (4) relies on the following lemma about the reverse function.

$$\mathrm{rev}(P) = P\left(\frac{1}{X}\right) \cdot X^m.$$

Proof.

$$\begin{aligned}
\mathrm{rev}(P) &= \sum_{i=0}^{m} a_{m-i}X^i \\
&= \left(\sum_{i=0}^{n} a_{m-i}X^{i-m}\right)X^m \\
&= \left(\sum_{i=0}^{m} a_i X^{-i}\right)X^m \\
&= P\left(\frac{1}{X}\right)X^m.
\end{aligned}$$

□

We now prove the equivalence of the definitions (3) and (4).

Proof. The polynomial P factorizes in $\mathbb{L}[X]$ because $\mathbb{L}$ is an algebraic closed field:

$$P = a_m \prod_{i\leq m}(X - \alpha_i).$$

The definition (3) gives

$$\begin{aligned}
\mathcal{N}(P) &= \sum_{s=0}^{\infty}\left(\sum_{i\leq m}\alpha_i^s\right)X^s \\
&= \sum_{i\leq m}\left(\sum_{s=0}^{\infty}(\alpha_i X^s)\right) \\
&= \sum_{i\leq m}\frac{1}{1 - \alpha_i X}.
\end{aligned}$$

The definition (4) uses

$$\mathrm{rev}(P) = P\left(\frac{1}{X}\right)X^m$$

$$\text{and } \mathrm{rev}(P') = P'\left(\frac{1}{X}\right)X^{m-1}.$$

Then:

$$\frac{\mathrm{rev}(P')}{\mathrm{rev}(P)} = \frac{1}{X}\frac{P'\left(\frac{1}{X}\right)}{P\left(\frac{1}{X}\right)} = \frac{1}{X}\sum_i \frac{1}{\frac{1}{X} - \alpha_i} = \sum_i \frac{1}{1 - \alpha_i X}.$$

□

The formal proof follows the same reasoning steps. In the reference (Bostan 2003), (3) is a definition and (4) is a theorem, whereas in our Coq formalization, we exchange definition and theorem of Newton representation. The formal proof relies on TFPS, on fractions and on the development of a fraction of polynomials into a TFPS.

In section 3.3, we use this transformation to compute the Newton power series associated with two polynomials P and Q, on which we perform the operation corresponding to the composed sums and products. In order to get the result, we need to transform back a power series into a polynomial.

We prove the backward transformation theorem

$$\mathrm{rev}(P) = \exp\left(\int \frac{1}{X}(m - \mathcal{N}(P))\right) \tag{5}$$

First, remark that the constant term of $\mathcal{N}(P)$ is

$$N_0(P) = \sum_{i\leq m}\alpha_i^0 = \sum_{i\leq m} 1 = m$$

(the number of roots of P, which is the degree of P). So, X divides $m - \mathcal{N}(P)$. It is then possible to define $\frac{m-\mathcal{N}(P)}{X}$ as an element of $\mathbb{K}[[X]]$.

Whenever we encounter an imprecise equality between a polynomial and an infinite FPS, we explicitly truncate the FPS to the right precision. This formula (5) establishes a relation between $\mathrm{rev}(P)$ and $\mathcal{N}(P)$. It is possible to recover $\mathrm{rev}(P)$ from $\mathcal{N}(P)$. Besides, it is possible to recover P, except the multiplicity of 0 in P, from $\mathrm{rev}(P)$. Thus, it is possible to recover P, except the multiplicity of 0 in P, from $\mathcal{N}(P)$.

When $P(0) \neq 0$, which is a decidable property, $\mathrm{rev}(\mathrm{rev}(P)) = P$ holds and thus our backward conversion formula rewrites as:

$$P = \mathrm{rev}(\exp(\int \frac{1}{X}(m - \mathcal{N}(P)))).$$

If $P(0) = 0$, we can recover the multiplicity of X by making the difference between the degree of the result and m.

We write the inverse of `newton_tfps`:

```
Definition newton_inv (p : {tfps K m}) : {poly K}
:= revp (exp (prim_tfps (divfX ((p^_0)%:S - p)))).
```

We now prove (5). First we check that for any polynomials U, V such that $U_0 = 1$:

$$\frac{U'}{U} = V \implies U = \exp\left(\int V\right). \tag{6}$$

Proof.

$$\begin{aligned}
\frac{U'}{U} = V \quad &\iff \quad \int \frac{U'}{U} = \int V \\
&\iff \quad \int (\log(U))' = \int V \\
&\iff \quad \log U - (\log U)_0 = \log U - 0 = \int V \\
&\iff \quad \exp(\log U) = \exp\left(\int V\right) \\
&\qquad \text{(injectivity of the exponential)} \\
&\iff \quad U = \exp\left(\int V\right).
\end{aligned}$$

We proved (6). So, by applying (6) to the desired theorem, it is sufficient to prove:

$$\frac{\mathrm{rev}(P)'}{\mathrm{rev}(P)} = \frac{m - \mathcal{N}(P)}{X} \tag{7}$$

We prove (7) by considering both sides as FPS. On one side:

$$n - \mathcal{N}(P) = m - \sum_{s=0}^{\infty} N_s(P) X^s = -\sum_{s=1}^{\infty} N_s(P)\, X^s$$

$$\frac{n - \mathcal{N}(P)}{X} = -\sum_{s=0}^{\infty} N_{s+1}(P)\, X^i = -\sum_{s=0}^{\infty} \left(\sum_i \alpha_i^{s+1}\right) X^s$$

On the other side:

$$\frac{\mathrm{rev}(P)'}{\mathrm{rev}(P)} = \sum_i \frac{1}{X - \frac{1}{\alpha_i}} = \sum_i \frac{\alpha_i}{\alpha_i X - 1} = -\sum_i \frac{\alpha_i}{1 - \alpha_i X}$$

$$\frac{\mathrm{rev}(P)'}{\mathrm{rev}(P)} = -\sum_i \alpha_i \left(\sum_{s=0}^{\infty} (\alpha_i X)^s\right) = -\sum_{i=0}^{\infty} \left(\sum_i \alpha_i^{s+1}\right) X^s.$$

□

We proved:

$$\frac{\mathrm{rev}(P)'}{\mathrm{rev}(P)} = \frac{n - \mathcal{N}(P)}{X}.$$

The result follows by applying (6) to (7). In our Coq code, the backward transformation theorem is expressed as:

```
Lemma newton_tfpsK (p : {poly K}) :
  size p <= m.+1 -> ~~ (root p 0) -> p \is monic ->
  newton_inv (newton_tfps m p) = p.
```

As one could see in the proof, we use a result about formal geometric series. We use this version:

$$\frac{1}{1 - aX} = \sum_{i=0}^{\infty} a^i X^i$$

In our Coq code, it is expressed as:

```
Lemma geometric_series (a : K) (m : nat) :
  Tfpsp m (((1 - a *: 'X)%:F) ^-1)
  = [tfps j => a ^+ j].
```

3.3 Composed sums and products

Given two monic polynomials in $\mathbb{K}[X]$, $P = \sum a_i X^i$ with roots $\alpha_1, \ldots, \alpha_m$ in $\mathbb{L}$ and $Q = \sum b_j X^j$ with roots $\beta_1, \ldots, \beta_n$ in $\mathbb{L}$, we wish to compute a polynomial whose roots are $\alpha_i + \beta_j$ for $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$, which we write $P \oplus Q$ and which we call composed sum. Similarly we define the composed product $P \otimes Q$, whose roots are $\alpha_i \beta_j$ for $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Since P and Q are supposed monic, a_m and b_n are equal to 1, and we can split the polynomials in $\mathbb{L}[X]$. We write abusively $P = \prod_i (X - \alpha_i)$ and $Q = \prod_j (X - \beta_j)$, and we want to compute the results

$$P \oplus Q = a_m b_n \prod_{(i,j)} (X - (\alpha_i + \beta_j))$$

$$P \otimes Q = a_m b_n \prod_{(i,j)} (X - (\alpha_i \cdot \beta_j)).$$

Since, both $P \oplus Q$ and $P \otimes Q$ are symmetric in the α_i and β_i, the fundamental theorem of symmetric polynomials concludes that their coefficients are polynomials in the coefficients of P and Q. We can compute $P \oplus Q$ and $P \otimes Q$ without the need for factoring P or Q. One solution is to use the resultant (which is already implemented for the construction of algebraic numbers(Cohen 2012b)) thanks to the relations:

$$(P \oplus Q)(X) = Res_Y\left(P(X - Y), Q(Y)\right)$$

$$(P \otimes Q)(X) = Res_Y\left(Y^m P(\frac{X}{Y}), Y\right)$$

where m is the degree of P. Note that $P(\frac{X}{Y}) \in \mathbb{K}(X, Y)$ but $Y^m P(\frac{X}{Y}) \in \mathbb{K}[X, Y]$.

The Newton power series enables us to compute $(P \oplus Q)(X)$ and $(P \otimes Q)(X)$ faster for any characteristic of $\mathbb{K}$. The Newton representation of a polynomial is a FPS. If this FPS is truncated far enough, there is no loss of information about the input polynomial. Then, the composed sum and composed product are done in the space of TFPS.

We formally prove the following equation for the composed product

$$\mathcal{N}(P \otimes Q) = \mathcal{N}(P) \odot \mathcal{N}(Q) \tag{8}$$

where $\odot$ denotes the Hadamard product, *i.e.* the term-wise product of TFPS.

In Coq:

```
Definition cmul (p q : {poly K}) :=
  if (p == 0) || (q == 0) then 0 else
  \prod_(r <- [seq s * t | s <- roots p ^ iota,
               t <- roots q ^ iota]) ('X - r%:P)
Lemma newton_cmul m p q :
  newton_tfps m (cmul iota p q) =
  (hmul_tfps (newton_tfps m p) (newton_tfps m q))
  ^ iota.
```

The proof of (8) boils down to proving the equality of the coefficients of two FPS.

Proof.

$$\begin{aligned} N_s(P \otimes Q) &= \sum_{\substack{i \le m \\ j \le n}} (\alpha_i \beta_j)^s \\ &= (\sum_{i \le m} \alpha_i^s)(\sum_{j \le n} \beta^s) \\ &= N_s(P) N_s(Q). \end{aligned}$$

□

With our example we have

$$\begin{aligned} \mathcal{N}(P \otimes Q) &= \mathcal{N}(P) \odot \mathcal{N}(Q) \\ &= \sum_{s \ge 0} 2^{s+1} X^{2s} \odot \sum_{s \ge 0} (-3)^s X^s \\ &= 2 + 36X^2 + \dots \end{aligned}$$

We then compute

$$\begin{aligned} \mathrm{rev}(P \otimes Q) &= \exp\left(\int \frac{1}{X}\left(2 - \mathcal{N}(P \otimes Q)\right)\right) \\ &= \exp\left(\int (-36X + \dots)\right) \\ &= \exp\left(-18X^2 + \dots\right) \\ &= \sum_{s \ge 0} \frac{(-18X^2 + \dots)^s}{s!} \\ &= 1 - 18X^2 + \dots \end{aligned}$$

which we truncate at precision 2. Hence $P \otimes Q = X^2 - 18$, which is the expected result.

We formally prove the similar equation for the composed sum

$$\mathcal{N}(P \oplus Q) \odot E = (\mathcal{N}(P) \odot E) \cdot (\mathcal{N}(Q) \odot E) \qquad (9)$$

where

$$E = \exp(X) = \sum_{s=0}^{\infty} \frac{X^s}{s!}.$$

In Coq:

```
Definition cadd_poly (p q : {poly K}) :=
  if (p == 0) || (q == 0) then 0 else
  \prod_(r <- [seq s + t | s <- iroots p,
                t <- iroots q]) ('X - r%:P).

Lemma cadd_newton m p q :
  hmul_tfps (newton_tfps m (cadd_poly iota p q)) E
   =
  (hmul_tfps (newton_tfps m p) E
  * hmul_tfps (newton_tfps m q) E) ^ iota.
```

The proof of (9) is also almost immediate.

Proof. First we have:

$$\mathcal{N}(P \oplus Q) = \sum_{s=0}^{\infty} \left(\sum_{i,j} (\alpha_i + \beta_j)^s\right) X^s.$$

Thus (9) can be rewritten as:

$$\sum_{s=0}^{\infty} \frac{\sum_{i,j}(\alpha_i + \beta_j)^s}{s!} X^s = \left(\sum_{s=0}^{\infty}\left(\sum_i \frac{\alpha_i^s}{s!}\right)X^s\right)\left(\sum_{s=0}^{\infty}\left(\sum_j \frac{\beta_j^s}{s!}\right)X^s\right)$$

which can be re-expressed in the following way:

$$\sum_{i,j} \exp\left((\alpha_i + \beta_j)X\right) = \left(\sum_i \exp(\alpha_i X)\right) \cdot \left(\sum_j \exp(\beta_j X)\right).$$

The latter is a direct consequence of the morphism property of the exponential. This completes the proof of (9). □

With our example we have

$$\begin{aligned} \mathcal{N}(P \oplus Q) \odot E &= (\mathcal{N}(P) \odot E) \cdot (\mathcal{N}(Q) \odot E) \\ &= \sum_{s \ge 0} \frac{2^{s+1}}{(2s)!} X^{2s} \cdot \sum_{s \ge 0} \frac{(-3)^s}{s!} X^s \\ &= \left(2 + 2X^2 + \dots\right)\left(1 - 3X + \frac{9}{2}X^2 + \dots\right) \\ &= 2 - 6X + 11X^2 + \dots \\ \mathcal{N}(P \oplus Q) &= 2 - 6X + 22X^2 + \dots \end{aligned}$$

We then compute the backward transformation.

$$\begin{aligned} \mathrm{rev}(P \oplus Q) &= \exp\left(\int \frac{1}{X}\left(2 - \mathcal{N}(P \oplus Q)\right)\right) \\ &= \exp\left(\int (6 - 22X + \dots)\right) \\ &= \exp\left(6X - 11X^2 + \dots\right) \\ &= \sum_{s \ge 0} \frac{(6X - 11X^2 + \dots)^s}{s!} \\ &= 1 + 6X + 7X^2 + \dots \end{aligned}$$

which we truncate at precision 2. Hence $P \oplus Q = X^2 + 6X + 7$, which is the expected result.

4. Related Work

This work is mostly based on Alin Bostan's PhD thesis (Bostan 2003) for the mathematical results, that he develops with care in order to provide algorithms for computer algebra. In our paper we only study the theory for characteristic zero, while he does it for arbitrary characteristic.

In (Chaieb 2011), the author describes a formalization of formal power series in ISABELLE/HOL as functions from natural numbers. He already provides formal derivative, division, composition, inverse, logarithm. He provides additional constructions such as arbitrary nth roots, compositional inverse, binomial FPS, trigonometric FPS. These constructions are done for arbitrary domain. Contrary to our work, (Chaieb 2011) does not require polynomials and defines them as FPS. We define TFPS by exploiting an existing library about polynomials. In (Alasdair Armstrong 2014), the authors generalizes FPS by implementing them as functions from free monoids into dioids.

We rely on the code of P.-Y. Strub (Strub 2014) for the theory of polynomials on a decidable field, in order to get the list of roots of a polynomial in an algebraically closed field.

For the fractions, we follow up on a work by the first author (Cohen 2013), where he describes a construction of the fraction field of an integral domain using a quotient construction. We expand this work in a slightly different way than (Strub 2014) does, by generalizing the notion of pole and evaluation to an arbitrary integral domain.

We also use the existence of the algebraic closure to conclude that the resulting polynomial has the desired properties (Cohen 2012b,a).

Conclusion and future work

In this paper, we describe a formalization of truncated formal power series. We equip them with a commutative ring structure and define some common operations: Hadamard product, derivative, primitive, composition, exponential, logarithm and prove formulas linking these operations. The theory of truncated power series is both an artifice to avoid handling FPS as infinite objects and a feature as it explicits the precision that the operations have. We hope that the theory of TFPS can be reused for a theory of Taylor series.

We also develop a theory of abstract poles of fractions to factorize code and improve readability, which generalizes the universal property of the field of fractions. The theory of abstract poles is used to develop the theory of evaluation of polynomial fractions in a modular way. It is used twice: once for the evaluation of polynomials, once for lifting of injective morphisms from polynomials to a field, to a morphism from polynomial fractions to a field.

We then use these components to define the Newton power series and prove the main results: forward transformation theorem, backward transformation, theorem linking composed product with the Hadamard product, formula linking the composed sum with the product.

Whatever implementation of algebraic numbers we pick, in order to implement the sum and product on algebraic numbers, one step is to compute the composed sum and product of two polynomials. In this paper we describe an efficient way to do so. We should now investigate efficient ways to select a root of the composed sum and products. Once we have more efficient pieces than in (Cohen 2012b) we may use the CoqEAL library to provide computable versions of the algorithms we describe and formalized here and perform computations on algebraic numbers inside Coq.

Acknowledgments

Pretty Coq code listing was done thanks to Assia Mahboubi's file (Mahboubi). Commutative diagrams where drawn thanks to Pedro Quaresma's package (Quaresma). We thank our colleagues for their input on this work: Enrico Tassi, José Grimm, Laurence Rideau, Laurent Théry and Yves Bertot.

References

T. W. Alasdair Armstrong, Georg Struth. Programming and Automating Mathematics in the Tarski-Kleene Hierarchy. 2014.

Y. Bertot, G. Gonthier, S. Ould Biha, and I. Pasca. Canonical big operators. In *Theorem Proving in Higher-Order Logics*, volume 5170 of *LNCS*, pages 86–101, 2008.

A. Bostan. *Algorithmique efficace pour des opérations de base en Calcul formel*. 2003.

A. Chaieb. Formal power series. *Journal of Automated Reasoning*, 47:291–318, October 2011.

C. Cohen. *Formalized algebraic numbers: construction and first-order theory*. PhD thesis, École polytechnique, Nov 2012a.

C. Cohen. Construction of real algebraic numbers in Coq. 2012b.

C. Cohen. Pragmatic quotient types in coq. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 213–228. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39633-5. doi: 10.1007/978-3-642-39634-2_17. URL http://dx.doi.org/10.1007/978-3-642-39634-2_17.

G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the odd order theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39633-5. doi: 10.1007/978-3-642-39634-2_14. URL http://dx.doi.org/10.1007/978-3-642-39634-2_14.

A. Mahboubi. lstcoq.sty file which defines a Coq - SSReflect style for listings in Latex. https://hal.inria.fr/file/index/docid/611757/filename/lstcoq.sty. Accessed: 25/06/2015.

P. Quaresma. DCpic package to draw diagrams in Latex. https://www.ctan.org/pkg/dcpic. Accessed: 25/06/2015.

P.-Y. Strub. A Coq/Ssreflect Library for Elliptic Curves. In R. G. Gerwin Klein, editor, *Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2014. ISBN 978-3-319-08970-6. doi: 10.1007/978-3-319-08970-6_6. URL https://github.com/strub/elliptic-curves-ssr. Accessed: 3/07/2015.

T. Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

Wikipedia. Composition of series. https://en.wikipedia.org/wiki/Formal_power_series#Composition_of_ser Accessed: 29/06/2015.