Fast computation of the roots of polynomials over the ring of power series

Vincent Neiger
Technical University of Denmark
Kgs. Lyngby, Denmark
vinn@dtu.dk

Johan Rosenkilde
Technical University of Denmark
Kgs. Lyngby, Denmark
jsrn@jsrn.dk

Éric Schost
University of Waterloo
Waterloo ON, Canada
eschost@uwaterloo.ca

ABSTRACT

We give an algorithm for computing all roots of polynomials over a univariate power series ring over an exact field $\mathbb{K}$. More precisely, given a precision d, and a polynomial Q whose coefficients are power series in x, the algorithm computes a representation of all power series $f(x)$ such that $Q(f(x)) = 0 \bmod x^d$. The algorithm works unconditionally, in particular also with multiple roots, where Newton iteration fails. Our main motivation comes from coding theory where instances of this problem arise and multiple roots must be handled.

The cost bound for our algorithm matches the worst-case input and output size $d \deg(Q)$, up to logarithmic factors. This improves upon previous algorithms which were quadratic in at least one of d and $\deg(Q)$. Our algorithm is a refinement of a divide & conquer algorithm by Alekhnovich (2005), where the cost of recursive steps is better controlled via the computation of a factor of Q which has a smaller degree while preserving the roots.

KEYWORDS

Polynomial root finding; power series; list decoding.

1 INTRODUCTION

In what follows, $\mathbb{K}$ is an exact field, and $\mathbb{K}[\![x]\!][y]$ denotes the set of polynomials in y whose coefficients are power series in x over $\mathbb{K}$.

Problem and main result. Given a polynomial in $\mathbb{K}[\![x]\!][y]$, we are interested in computing its power series roots to some precision, as defined below.

Definition 1.1. Let $Q \in \mathbb{K}[\![x]\!][y]$ and $d \in \mathbb{Z}_{>0}$. A power series $f \in \mathbb{K}[\![x]\!]$ is called a *root of Q to precision d* if $Q(f) = 0 \bmod x^d$; the set of all such roots is denoted by $\mathcal{R}(Q, d)$.

Our main problem asks, given Q and d, to compute a finite representation of $\mathcal{R}(Q, d)$; the fact that such a representation exists is explained below (Theorem 2.8). In all the paper, we count operations in $\mathbb{K}$ at unit cost; we use the soft-O notation $O\tilde{~}$ to indicate the omission of polylogarithmic factors.

Problem 1.
Input:

- *a precision $d \in \mathbb{Z}_{>0}$,*
- *a polynomial $Q \in \mathbb{K}[\![x]\!][y]$ known at precision d.*

Output:

- *(finite) list of pairs $(f_i, t_i)_{1 \le i \le \ell} \subset \mathbb{K}[x] \times \mathbb{Z}_{\ge 0}$ such that $\mathcal{R}(Q, d) = \bigcup_{1 \le i \le \ell} (f_i + x^{t_i} \mathbb{K}[\![x]\!])$*

In this paper, we design an algorithm to solve this problem in time quasi-linear in the precision d and the degree of Q. Such an algorithm must necessarily involve finding roots of polynomials in $\mathbb{K}[y]$.

The existence, and complexity, of root-finding algorithms for univariate polynomials over $\mathbb{K}$ depends on the nature of $\mathbb{K}$. In this paper, we assume that $\mathbb{K}$ is such that we can find roots in $\mathbb{K}$ of a degree n polynomial in $\mathbb{K}[y]$ in time $\mathsf{R}_{\mathsf{Y}}(n)$, for some function $\mathsf{R}_{\mathsf{Y}} : \mathbb{Z}_{\ge 0} \to \mathbb{R}$; the underlying algorithm may be deterministic or randomized. For instance, if $\mathbb{K} = \mathbb{F}_q$, we can take $\mathsf{R}_{\mathsf{Y}}(n) \in O\tilde{~}(n)$ using either a Las Vegas algorithm (in which case the runtime can be more precisely stated as $O\tilde{~}(n \log(q))$ [19, Corollary 14.16]), or a deterministic one (with for instance a runtime $O\tilde{~}(nk^2 \sqrt{p})$, where we write $q = p^k$, p prime [17]).

Our main result is as follows: we separate the cost of a deterministic part of the algorithm from that of root-finding, which may be randomized.

Theorem 1.2. *There is an algorithm which solves Problem 1 using $O\tilde{~}(dn)$ deterministic operations in $\mathbb{K}$, together with an extra $d\mathsf{R}_{\mathsf{Y}}(n)$ operations, where $n = \deg(Q)$.*

A cost such as $O\tilde{~}(dn)$ is essentially optimal for Problem 1. Indeed, if $Q = (y - f_1) \cdots (y - f_n)$, for some power series $f_1, \ldots, f_n$, with $f_i - f_j$ a unit for all $i \ne j$, the roots of Q to precision d are all those power series of the form $f_i + x^d \mathbb{K}[\![x]\!]$, for some i; solving Problem 1 then involves writing down all $f_i \bmod x^d$, which amounts to dn elements in $\mathbb{K}$.

Previous work. When the discriminant of $Q \in \mathbb{K}[\![x]\!][y]$ has x-valuation zero, or equivalently, when all y-roots of $Q_{|x=0}$ are simple (as in the example above), our problem admits an obvious solution: first, compute all y-roots of $Q_{|x=0}$ in $\mathbb{K}$, say $y_1, \ldots, y_\ell$, for some $\ell \le n$, where $n = \deg Q$. Then, apply Newton iteration to each of these roots to lift them to power series roots $f_1, \ldots, f_\ell$ of precision d; to go from precision say $d/2$ to d, Newton iteration replaces f_i by

$$f_i - \frac{Q(f_i)}{Q'(f_i)} \bmod x^d ,$$

where $Q' \in \mathbb{K}[\![x]\!][y]$ is the formal derivative of Q. The bottleneck of this approach is the evaluation of all $Q(f_i)$ and $Q'(f_i)$. Using an algorithm for fast multi-point evaluation in the ring of univariate polynomials over $\mathbb{K}[\![x]\!]/x^d$, these evaluations can both be done in $O\tilde{~}(dn)$ operations in $\mathbb{K}$. Taking all steps into account, we obtain the roots $f_1, \ldots, f_\ell$ modulo x^d using $O\tilde{~}(dn)$ operations in $\mathbb{K}$; this is essentially optimal, as we pointed out above. In this case, the total time for root finding is $\mathsf{R}_{\mathsf{Y}}(n)$.

Thus, the non-trivial cases of Problem 1 arise when $Q_{|x=0}$ has multiple roots. In this case, leaving aside the cost of root-finding (which is handled in a non-uniform way in previous work), there is no algorithm of cost similar to ours: the best results known to us are of the form $\tilde{O}(n^2 d)$ from [1], the precise cost estimate we give being due to [13], or $\tilde{O}(nd^2)$ from [4].

When $Q_{|x=0}$ has multiple roots, a natural generalization of our problem consists in computing Puiseux series solutions of Q. It is then customary to consider a two-stage computation: first, compute sufficiently many terms of the power series / Puiseux series solutions in order to be able to *separate* the branches, then switch to another algorithm to compute many terms efficiently.

Most algorithms for the first stage compute the so-called singular parts of rational Puiseux expansions [7] of the solutions. They are inspired by what we will call the *Newton-Puiseux* algorithm, that is, Newton's algorithmic proof that the field of Puiseux series $\mathbb{K}\langle\langle x\rangle\rangle$ is algebraically closed when $\mathbb{K}$ is algebraically closed of characteristic zero [12, 20]. In the case of Puiseux series roots, one starts by reading off the leading exponent γ of a possible solution on the Newton polygon of the input equation $Q \in \mathbb{K}\langle\langle x\rangle\rangle[y]$. The algorithm then considers $Q_i = Q(x^\gamma y)/x^s \in \mathbb{K}\langle\langle x\rangle\rangle[y]$, where s is the valuation at x of $Q(x^\gamma y))$. If $y_1, \ldots, y_\ell$ are the y-roots of $Q_{i\,|x=0}$, then these are the x^γ terms of the Puiseux series roots of Q. For each i we then replace Q with $Q(x^\gamma(y_i + y))/x^{s'}$, where s' is the valuation at x of $Q(x^\gamma(y_i + y))$. This allows us to compute the terms of the solutions one by one. The best algorithms to date [14, 15] use an expected $\tilde{O}(n^2 \nu + n^3 + n^2 \log(q))$ base field operations, if $\mathbb{K} = \mathbb{F}_q$ and where ν is the valuation of the discriminant of Q. The runtime is Las Vegas, since the algorithm relies on Las Vegas root finding in $\mathbb{F}_q[y]$.

In the second stage, given the singular parts of the solutions, it becomes for instance possible to apply Newton iteration, as in [9]. If Q is actually in $\mathbb{K}[x][y]$, one may alternatively derive from it a linear recurrence with polynomial coefficients satisfied by the coefficients of the solutions we are looking for; this allows us to compute them at precision d using $O(dn)$ operations, that is, in time genuinely linear in n, d [5, 6] (but keep in mind that in both cases, we may need to know about ν terms of the solutions before being able to switch to the faster algorithm). We will discuss a similar observation in the context of our algorithm, in the last section of the paper.

Using ideas akin to the Newton-Puiseux algorithm, Berthomieu, Quintin and Lecerf gave in [4] an algorithm that computes roots of polynomials in $L[y]$, for a wide class of local rings L. In the particular case where $L = \mathbb{F}_q[[x]]$, with $q = p^s$, the expected runtime of their algorithm is $\tilde{O}(nd^2 + n\log(q) + nd\log(k)/p)$ operations in $\mathbb{F}_q$.

Let us finally mention algorithms for polynomial factorization in local fields. Using the so-called Montes algorithm [10], reference [3] proves that one can compute a so-called OM-factorization of a degree n polynomial Q in $\mathbb{F}_q\langle\langle x\rangle\rangle[y]$ at precision d using $\tilde{O}(n^2\nu + n\nu^2 + n\nu\log(q))$, where ν is the valuation of the discriminant of Q; the relation to basic sets of roots, as defined below, remains to be elucidated.

Sudan's and Guruswami–Sudan's algorithms for the list-decoding of Reed-Solomon codes [8, 18] have inspired a large body of work, some of which is directly related to Problem 1. These algorithms operate in two stages: the first stage finds a constrained polynomial in $\mathbb{K}[x, y]$; the second one finds its factors of the form $y - f(x)$, for f in $\mathbb{K}[x]$.

The Newton-Puiseux algorithm can easily be adapted to compute such factors; in this context, it becomes essentially what is known as the Roth and Ruckenstein algorithm [16]; its complexity is $O(d^2 n^2)$, neglecting work for univariate root finding.

In the context of Sudan's and Guruswami-Sudan's algorithms, we may actually be able to use Newton iteration directly, by exploiting the fact that we are looking for *polynomial* roots. Instead of computing power series solutions (that is, the Taylor expansions of these polynomial roots at the origin), one can as well start from another expansion point x_0 in $\mathbb{K}$; if the discriminant of Q does not vanish at x_0, Newton iteration applies. If $\mathbb{K}$ is finite, one cannot exclude the possibility that all x_0 in $\mathbb{K}$ are roots of Q; if needed, one may then look for x_0 in an extension of $\mathbb{K}$ of small degree. Augot and Pecquet show in [2] that in the cases appearing in Sudan's algorithm, there is always a suitable $x_0 \in \mathbb{K}$.

However, for e.g. the Wu list decoding algorithm [21], or for list-decoding certain algebraic geometry codes [13], we really seek truncated power series roots. In this case, one may then use Alekhnovich's algorithm [1, Appendix], which is a divide and conquer variant of the Roth and Ruckenstein algorithm (and will form the basis of our algorithm). It solves Problem 1 using $n^{O(1)}\tilde{O}(d)$ base field operations plus calls to univariate root finding; the refined analysis in [13] gives a runtime of the form $\tilde{O}(n^2 d + nd\log q)$.

Outline. We start by giving properties about the structure of the set of roots in Section 2. We will see in particular how $\mathcal{R}(Q, d)$ can recursively be described as the finite union of set of roots at a lower precision for shifts of Q, that is, polynomials of the form $Q(f + x^t y)$. From this, we will be able to derive a divide-and-conquer algorithm which is essentially Alekhnovich's.

The reason why the runtime of this algorithm is quadratic in n is the growth of the (sum of the) degrees of these shifts. Having in mind to control this degree growth, we conclude Section 2 with the definition of so-called *reduced root sets*, for which we establish useful degree properties.

In Section 3, we detail a fast algorithm for the computation of *affine factors*, which are polynomials having the same roots as the shifts but which can be computed more efficiently thanks to the degree properties of our reduced root sets. Finally, in Section 4, we incorporate this into the divide and conquer approach, leading to our fast power series roots algorithm.

2 STRUCTURE OF THE SET OF ROOTS

Recall the notation of Problem 1. In the following analysis, we consider knowing Q to arbitrary precision, i.e. $Q \in \mathbb{K}[[x]][y]$. For convenience, we also define for any $d \leq 0$ that $\mathcal{R}(Q, d) = \mathbb{K}[[x]]$. First, we introduce basic notation.

- $v_x : \mathbb{K}[[x]][y] - \{0\} \to \mathbb{Z}_{\geq 0}$ denotes the valuation at x, that is $v_x(Q)$ is the greatest power of x which divides Q, for any nonzero $Q \in \mathbb{K}[[x]][y]$.
- For $Q \in \mathbb{K}[[x]][y]$, we write $Q_{|x=0}$ for the univariate polynomial in $\mathbb{K}[y]$ obtained by replacing x by 0 in Q.

- $\deg(\cdot)$ always stands for the degree of some polynomial in $\mathbb{K}[y]$, while the degree of polynomials in $\mathbb{K}[x]$ is denoted using $\deg_x(\cdot)$.
- $\mathcal{S}_d = \mathbb{K}[\![x]\!]/(x^d)$ is the ring of power series in x over $\mathbb{K}$ truncated at precision d.

The next lemma is straightforward and shows that we can focus on the case $v_x(Q) = 0$.

Lemma 2.1. *Let $Q \in \mathbb{K}[\![x]\!][y]$ be nonzero and let $d \in \mathbb{Z}_{>0}$. If $Q_{|x=0} = 0$, then $\mathcal{R}(Q,d) = \mathcal{R}(x^{-s}Q,\, d-s)$, where $s = v_x(Q)$.*

In the following we give a characterization of $\mathcal{R}(Q,d)$ which admits a compact representation even though $\mathcal{R}(Q,d)$ is usually infinite. Similar characterizations are also behind the correctness and complexity of Ruckenstein [16] and Berthomieu *et al.* [4, Section 2.2]. To support the divide-and-conquer structure of our algorithm, we further describe how these representations compose. A similar setup is also necessary for the correctness and complexity of Alekhnovich's algorithm [1, Appendix], though it was not explicitly addressed in that reference.

Definition 2.2. For d in $\mathbb{Z}_{>0}$, a *basic root set* of Q to precision d is a set of pairs $(f_i, t_i)_i$, with $(f_i, t_i) \in \mathbb{K}[x] \times \mathbb{Z}_{\geq 0}$ for all i, such that:

- $v_x(Q(f_i + x^{t_i}y)) \geq d$ for each i,
- we have the equality

$$\mathcal{R}(Q,d) = \bigcup_{1 \leq i \leq \ell} \left\{ f_i + x^{t_i}\mathbb{K}[\![x]\!] \right\}.$$

For $d \leq 0$, we define the unique basic root set of Q to precision d as $\{(0,0)\}$; note that it satisfies both conditions above.

We remark that the first restriction on being a basic root set is key: for instance, $Q = y^2 + y$ over $\mathbb{F}_2[x][y]$ has $\mathcal{R}(Q,1) = \mathbb{F}_2[\![x]\!]$. But $\{(0,0)\}$ is *not* a basic root set because it does not satisfy the first property; rather a basic root set is given by expanding the first coefficient: $\{(0,1),(1,1)\}$.

For $d = 1$, one can easily build a finite, small basic root set for Q:

Lemma 2.3. *Let $Q \in \mathbb{K}[\![x]\!][y]$ be such that $Q_{|x=0} \neq 0$. Let $y_1, \ldots, y_\ell$ be the roots of $Q_{|x=0}$. Then, $(y_i, 1)_{1 \leq i \leq \ell}$ is a basic root set for Q to precision 1.*

Proof. Take i in $\{1, \ldots, \ell\}$ and write the Taylor expansion of $Q(y_i + xy)$ as $Q(y_i + xy) = Q(y_i) + xR_i(y)$, for some polynomial R_i. Since both terms in the sum have valuation at least 1, $s_i = v_x(Q(y_i + xy))$ is at least 1.

Remark that $\mathcal{R}(Q,1) = \{f \mid Q(f) = 0 \bmod x\} = \{f \mid Q_{|x=0}(f_0) = 0\}$, where f_0 is the constant coefficient of f. This implies that $\mathcal{R}(Q,1)$ is the set of f whose constant coefficient is in $\{y_1, \ldots, y_\ell\}$, so the proof is complete. □

Proposition 2.4. *Let $Q \in \mathbb{K}[\![x]\!][y]$ be such that $Q_{|x=0} \neq 0$ and let d', d be in $\mathbb{Z}_{\geq 0}$, with $d' \leq d$. Suppose that Q admits a basic root set $(f_i, t_i)_{1 \leq i \leq \ell}$ to precision d'. For $i = 1, \ldots, \ell$, suppose furthermore that $Q(f_i + x^{t_i}y)/x^{s_i}$ admits a basic root set $(f_{i,j}, t_{i,j})_{1 \leq j \leq \ell_i}$ to precision $d - s_i$, where $s_i = v_x(Q(f_i + x^{t_i}y))$. Then a basic root set of Q to precision d is given by*

$$(f_i + f_{i,j}x^{t_i}, t_i + t_{i,j})_{1 \leq j \leq \ell_i, 1 \leq i \leq \ell}\,.$$

Proof. For $1 \leq i \leq \ell$, let $Q_i = Q(f_i + x^{t_i}y)/x^{s_i}$. Then, for all i,j, from the definition of basic root sets, we have

$$v_x\Big(Q(f_i + f_{i,j}x^{t_i} + x^{t_i+t_{i,j}})\Big) = v_x\Big(\big(x^{s_i}Q_i\big)_{|y=f_{i,j}+x^{t_{i,j}}y}\Big) \geq s_i + (d - s_i).$$

This proves that the first property of Definition 2.2 holds.

For the second property, we begin with the inclusion $\mathcal{R}(Q,d) \subseteq \cup_{i,j}\{f_i + x^{t_i}f_{i,j} + x^{t_i+t_{i,j}}\mathbb{K}[\![x]\!]\}$. Consider some $f \in \mathcal{R}(Q,d)$; since $d' \leq d$, f is in $\mathcal{R}(Q,d')$, so we can write $f = f_i + x^{t_i}g$, for some i in $\{1, \ldots, \ell\}$ and g in $\mathbb{K}[\![x]\!]$. Then $Q(f) = x^{s_i}Q_i(g) \equiv 0 \bmod x^d$, so $g \in \mathcal{R}(Q_i, d - s_i)$; this means there is a j such that $g \in f_{i,j} + x^{t_{i,j}}\mathbb{K}[\![x]\!]$ as we sought.

For the reverse inclusion $\supseteq$, consider, for some index i, a power series $g \in \mathcal{R}(Q_i, d - s_i)$. This means that $Q_i(g) \equiv 0 \bmod x^{\max(0, d - s_i)}$, so that $Q(f_i + x^{t_i}g) = x^{s_i}Q_i(g) \equiv 0 \bmod x^d$, and $f_i + x^{t_i}g$ is in $\mathcal{R}(Q,d)$. □

We can now readily deduce, by induction on d, that any $Q \in \mathbb{K}[\![x]\!][y]$ admits a finite basic root set to precision d for any $d \in \mathbb{Z}_{\geq 0}$: By Lemma 2.1 we can reduce the case $v_x(Q) > 0$, and so can assume $Q_{|x=0} \neq 0$. The claim is readily seen to be true for $d \leq 0$ (take $\{(0,0)\}$) and $d = 1$ (Lemma 2.3). Suppose the claim holds for all $d' < d$, for some $d \geq 2$; we can then apply the induction property to $d - 1$, obtaining a basic root set $(f_i, t_i)_{1 \leq i \leq \ell}$ of Q to precision $d - 1$. We know that, with the notation of the proposition, $s_i \geq d - 1$ holds for all i, so in particular $s_i \geq 1$, and thus $d - s_i < d$. We can then apply the induction assumption to to all $Q_i, d - s_i$, and the conclusion of the proposition establishes our claim.

These results can be used algorithmically to build basic root sets: iteratively apply Lemma 2.3, or use Proposition 2.4 in a divide-and-conquer fashion with Lemma 2.3 applied at the leaves. As discussed in Section 1, these algorithms are similar to Newton–Puiseux, and are known in coding theory as the Roth–Ruckenstein algorithm [16] (iterative) and Alekhnovich algorithm [1, Appendix] (divide-and-conquer). The latter algorithm is listed for convenience as Algorithm 1, since our new algorithm, Algorithm 4, runs along the same lines. We will not in further detail discuss the correctness or complexity of Algorithm 1, but see [1, Appendix] or [13].

Algorithm 1 : DnCSeriesRoots [1]

```
Input: $d \in \mathbb{Z}_{>0}$ and $Q \in \mathcal{S}_d[y]$ with $Q_{|x=0} \neq 0$.
Output: A basic root set of $Q$ to precision $d$.
if $d = 1$ then
    $(y_i)_{1 \leq i \leq \ell} \leftarrow$ roots of $Q_{|x=0} \in \mathbb{K}[y]$
    return $(y_i, 1)_{1 \leq i \leq \ell}$
else
    $(f_i, t_i)_{1 \leq i \leq \ell} \leftarrow$ DnCSeriesRoots($Q \bmod x^{\lceil d/2 \rceil}, \lceil d/2 \rceil$)
    $(Q_i)_{1 \leq i \leq \ell} \leftarrow (Q(f_i + x^{t_i}y) \bmod x^d)_{1 \leq i \leq \ell}$
    $(s_i)_{1 \leq i \leq \ell} \leftarrow (v_x(Q_i))_{1 \leq i \leq \ell}$
    for $1 \leq i \leq \ell$ do
        if $s_i \geq d$ then
            $(f_{i,1}, t_{i,1}) \leftarrow (0,0)$ and $\ell_i \leftarrow 1$
        else
            $(f_{ij}, t_{ij})_{1 \leq j \leq \ell_i} \leftarrow$ DnCSeriesRoots($x^{-s_i}Q_i, d - s_i$)
    return $(f_i + x^{t_i}f_{i,j}, t_i + t_{i,j})_{1 \leq j \leq \ell_i, 1 \leq i \leq \ell}$.
```

The next step is to prove that there are special, small basic root sets, and that also these compose in a way similar to Proposition 2.4. In order to formulate this, we first introduce a generalization of root multiplicity to our setting:

Definition 2.5. Let $Q \in \mathbb{K}[\![x]\!][y]$ and $(f, t) \in \mathbb{K}[x] \times \mathbb{Z}_{>0}$ with $0 \leq \deg_x(f) < t$, f being written as $g + f_{t-1}x^{t-1}$, with $\deg_x(g) < t - 1$. The *root multiplicity of* (f, t) *in* Q is the root multiplicity of f_{t-1} in $R_{|x=0} \in \mathbb{K}[y]$, where R is the polynomial of valuation zero defined as $R = Q(g + x^{t-1}y)/v_x(Q(g + x^{t-1}y))$.

Note that if f_{t-1} is not a root of $R_{|x=0}$, the root multiplicity of (f, t) is 0. Also, if $t = 1$, so that $f = f_0$ is in $\mathbb{K}$, and if $Q_{|x=0} \neq 0$, the root multiplicity of $(f_0, 1)$ is simply the multiplicity of f_0 in $Q_{|x=0}$.

Definition 2.6. Let $Q \in \mathbb{K}[\![x]\!][y]$ be such that $Q_{|x=0} \neq 0$ and let d be in $\mathbb{Z}$. Suppose that $(f_i, t_i)_{1 \leq i \leq \ell}$ is a basic root set for Q at precision d. Then, we say that $(f_i, t_i)_{1 \leq i \leq \ell}$ is a *reduced root set*, if the following holds:
Either $d \leq 0$,
Or $d > 0$ and each $f_i \neq 0$, and the following points are all satisfied, where for $1 \leq i \leq \ell$, we write $s_i = v_x(Q(f_i + x^{t_i}y))$, $Q_i = Q(f_i + x^{t_i}y)/x^{s_i}$, and we write m_i for the root multiplicity of (f_i, t_i) in Q:

1) $m_i \geq 1$ for all i,
2) $\deg(Q_{i\,|x=0}) \leq m_i$ for all i, and
3) $\sum_i m_i \leq \deg(Q_{|x=0})$.

It follows from restriction 1) and 3) that $\ell \leq \deg(Q_{|x=0})$. Mimicking the structure of the first half of the section, we now first prove the existence of reduced root sets for $d = 1$ and then describe compositionality. The following lemma is inspired by [1, Lemma 1.1].

Lemma 2.7. *Let* $Q \in \mathbb{K}[\![x]\!][y]$ *be such that* $Q_{|x=0} \neq 0$. *The basic root set for* Q *to precision* 1 *defined in Lemma 2.3 is reduced.*

Proof. Let $y_1, \ldots, y_\ell$ be the roots of $Q_{|x=0}$, and, for $i = 1, \ldots, \ell$, let $s_i = v_x(Q(y_i + xy))$, $Q_i = v_x(Q(y_i + xy))/x^{s_i}$, and let m_i be the root multiplicity of y_i in $Q_{|x=0}$.

The inequalities $m_i \geq 1$ and $\sum_i m_i \leq \deg(Q_{|x=0})$ are clear. Consider then a fixed index i. To prove that $\deg(Q_{i\,|x=0}) \leq m_i$, we prove $\deg(Q_{i\,|x=0}) \leq s_i \leq m_i$. There are $P \in \mathbb{K}[y]$ and $R \in \mathbb{K}[\![x]\!][y]$ such that $P(y_i) \neq 0$ and $Q = (y - y_i)^{m_i}P(y) + xR$. Then

$$x^{s_i}Q_i = Q(y_i + xy) = (xy)^{m_i}P(y_i + xy) + xR(y_i + xy)\,.$$

The right-hand side reveals the following:

- Any monomial $x^\alpha y^\beta$ in $x^{s_i}Q_i$ satisfies $\alpha \geq \beta$, and hence $\deg(Q_{i\,|x=0}) \leq s_i$.
- $x^{s_i}Q_i$ contains the term $(xy)^{m_i}P(y_i)$, since this appears in $(xy)^{m_i}P(y_i + xy)$ and it cannot be cancelled by a term in $xR(y_i+xy)$ since all monomials there have greater x-degree than y-degree.

These two points imply $\deg(Q_{i\,|x=0}) \leq s_i \leq m_i$. □

The following theorem is exactly the statement of Proposition 2.4 except that "basic" has been replaced by "reduced":

Theorem 2.8. *Let* $Q \in \mathbb{K}[\![x]\!][y]$ *be such that* $Q_{|x=0} \neq 0$ *and let* d', d *be in* $\mathbb{Z}_{\geq 0}$, *with* $d' \leq d$. *Suppose that* Q *admits a reduced root set* $(f_i, t_i)_{1 \leq i \leq \ell}$ *to precision* d'. *For* $i = 1, \ldots, \ell$, *suppose furthermore that* $Q(f_i + x^{t_i}y)/x^{s_i}$ *admits a reduced root set* $(f_{i,j}, t_{i,j})_{1 \leq j \leq \ell_i}$ *to precision* $d - s_i$, *where* $s_i = v_x(Q(f_i + x^{t_i}y))$. *Then a reduced root set of* Q *to precision* d *is given by*

$$(f_i + f_{i,j}x^{t_i}, t_i + t_{i,j})_{1 \leq j \leq \ell_i, 1 \leq i \leq \ell}\ .$$

Proof. By Proposition 2.4 it is clear that the specified set is a basic root set, and we should verify the additional restrictions of Definition 2.5. Introduce for each i, j

$$Q_{i,j} = Q(f_i + f_{i,j}x^{t_i} + x^{t_i + t_{i,j}}y)/x^{s_{i,j}} = Q_i(f_{i,j} + x^{t_{i,j}}y)/x^{s_{i,j}}\ ,$$

where $s_{i,j} = v_x(Q_i(f_{i,j} + x^{t_{i,j}}y))$.

Consider first for some i the case $d - s_i \leq 0$. Then $\ell_i = 1$ and $(f_{i,1}, t_{i,1}) = (0, 0)$, and so the root multiplicity $m_{i,1}$ of $(f_i + f_{i,1}x^{t_i}, t_i + t_{i,1})$ in Q is m_i which is positive by assumption. Also $Q_{i,j} = Q_i$ so $\deg(Q_{i,j\,|x=0}) = \deg(Q_{i\,|x=0})$ which is at most $m_i = m_{i,1}$ by assumption. Finally, $\sum_j m_{i,j} = m_{i,1} = m_i$. We will collect the latter fact momentarily to prove the third item of the reduced root definition.

Consider next an i where $d - s_i > 0$. In this case $t_{i,j} > 0$ for all $1 \leq j \leq \ell_i$, and the root multiplicity of $(f_i + f_{i,j}x^{t_i}, t_i + t_{i,j})$ in Q equals the root multiplicity $m_{i,j}$ of $(f_{i,j}, t_{i,j})$ in Q_i which is positive by assumption. The assumptions also ensure that $\deg(Q_{i,j\,|x=0}) \leq m_{i,j}$, and $\sum_j m_{i,j} \leq \deg(Q_{i\,|x=0}) \leq m_i$.

Thus, the two first restrictions on being a reduced root set is satisfied for each element. All that remains is the third restriction: but using our previous observations, we have $\sum_i \sum_j m_{i,j} \leq \sum_i m_i$ and this is at most $\deg(Q_{|x=0})$ by assumption. □

To solve Problem 1 we will compute a reduced root set using Lemma 2.7 and Theorem 2.8. Note that it follows that a reduced root set is essentially unique: apart from possible redundant elements among the f_i, non-uniqueness would only be due to unnecessarily expanding a coefficient in a root (f, t), i.e. replace that root by the $|\mathbb{K}|$ roots $(f + ax^t, t + 1)_{a \in \mathbb{K}}$. Of course this could only be an issue if $\mathbb{K}$ is finite and if $\deg(Q_{|x=0})$ is very large. Our algorithm as well as previous ones are computing the "minimal" set of reduced roots. According to Theorem 2.8, the total number of $\mathbb{K}$ elements required to represent this minimal set cannot exceed $d \deg(Q_{|x=0}) \leq d \deg(Q)$.

3 AFFINE FACTORS OF THE SHIFTS

The appendix of [13] gives a careful complexity analysis of Algorithm 1, and proves that it runs in time $\tilde{O}(dn^2 + dn\mathsf{R}_{\mathsf{y}})$. The main reason why the cost is quadratic in $\deg(Q)$ is that all the shifted polynomials $Q_i = x^{-s_i}Q(f_i + x^{t_i}y)$ can have large degree, namely up to $\deg(Q)$. Thus, merely representing the Q_i's may use a number of field elements quadratic in $\deg(Q)$.

Nonetheless, we are actually not interested in these shifts themselves, but only in their reduced root sets. The number of these roots is well controlled: the shifts have altogether a reduced root set of at most $\deg(Q_{|x=0})$ elements. Indeed, by definition, we know that $\deg(Q_{i\,|x=0})$ is at most the multiplicity m_i of the root (f_i, t_i), and the sum of these multiplicities is at most $\deg(Q_{|x=0})$.

The difficulty that we face now is that we want to efficiently compute the reduced root set of the shifts without fully computing these shifts. To achieve this, we compute for each shift Q_i a factor of it which has the same roots and whose degree is $\deg(Q_{i\,|x=0}) \leq m_i$, *without computing* Q_i *itself.* We design a fast algorithm for computing these factors, by using ideas from [11, Algorithm Q], in which we also incorporate fast modular reduction techniques so as to carefully control the quantity of information we process concerning the shifts.

The next result formalizes the factorization we will rely on. It is a direct consequence of the Weierstrass preparation theorem for multivariate power series [22, VII.§1. Cor. 1 of Thm. 5].

Theorem 3.1. *Let $Q \in \mathbb{K}[[x]][y]$ be such that $Q_{|x=0} \neq 0$. Then, there exist unique $A, B \in \mathbb{K}[[x]][y]$ such that $Q = AB$, A is monic and $B_{|x=0} \in \mathbb{K}\backslash\{0\}$.*

In the case at hand, one may as well derive existence and uniqueness of A and B (together with a slow algorithm to compute them) by writing their unknown coefficients as $A = a_0(y) + xa_1(y) + \cdots$ $B = b_0 + xb_1(y) + \cdots$, with b_0 in $\mathbb{K} - \{0\}$ and all a_i's ($i \geq 1$) of degree less than that of a_0. Extracting coefficients of $x^0, x^1, \ldots$, we deduce that the relation $Q = AB$ defines the a_i's and b_i's uniquely.

In what follows, A is called the *affine factor* of Q. Remark that if we start from Q in $\mathcal{S}_d[y]$, we can still define its affine factor as a polynomial in $\mathcal{S}_d[y]$, by reducing modulo x^d the affine factor of an arbitrary lift of Q to $\mathbb{K}[[x]][y]$ (the construction above shows that the result is independent of the lift).

Our algorithm will compute the affine factors $(A_i)_{1\leq i\leq \ell}$ of the shifts $(Q_i)_{1\leq i\leq \ell}$ at some prescribed precision d in x, having as input Q and the shifting elements $(f_i + x^{t_i}y)_{1\leq i\leq \ell}$. A factorization $Q_i = A_iB_i$ can be computed modulo any power x^d of x from the knowledge of Q_i by means of Hensel lifting [11, Algorithm Q], doubling the precision at each iteration. However, the above-mentioned degree bounds indicate that neither the shifts $(Q_i)_i$ nor the cofactors $(B_i)_i$ may be computed modulo x^d in time quasi-linear in $\deg(Q)$ and d: the key of our algorithm is to show how to compute the affine factors A_i at precision d directly from Q within the prescribed time bounds. (Hensel lifting factorization techniques were also used in [4], but in a context without the degree constraints that prevent us from computing the shifts Q_i). Hereafter, $A \operatorname{quo} B$ and $A \operatorname{rem} B$ denote the quotient and the remainder in the division of the polynomial A by the monic polynomial B.

The input of the algorithm is the polynomial Q known modulo x^d, as output, we compute the affine factors A_i of the shifts at respective precisions $d - s_i$, together with the valuation s_i; if $s_i \geq d$, we detect it and return $(0, d)$. The initialization consists in computing the affine factors of the x-constant polynomials $(Q_{i\,|x=0})_{1\leq i\leq \ell}$. If these polynomials are known, this is straightforward: the affine factor of $Q_{i\,|x=0}$ is itself divided by its leading coefficient, which is a nonzero constant from $\mathbb{K}$. It turns out that computing these polynomials is not an issue, thanks to the sum of their degrees being at most $m_1 + \cdots + m_\ell \leq \deg(Q)$. Explicitly, we compute the remainders $(Q(f_i + x^{t_i}y) \operatorname{rem} y^{m_i+1})_i$ via fast modular reduction techniques; then, we can both retrieve the valuations $(s_i)_i = (v_x(Q(f_i + x^{t_i}y)))_i$ (or, more precisely, $s_i^* = \min(s_i, d)$), and, when $s_i < d$, the x-constant terms of $Q_i = x^{-s_i}Q(f_i + x^{t_i}y)$ to carry out the initialization step (Line 1 to Line 11 in Algorithm 3).

Before continuing to describe the algorithm, we detail one of its main building blocks (Algorithm 2): the fast computation of simultaneous shifted remainders via multiple modular reduction.

Algorithm 2 : ShiftedRem

```
Input: a commutative ring $\mathcal{A}$, a polynomial $Q \in \mathcal{A}[y]$, and triples
    $(A_i, f_i, r_i)_{1\leq i\leq \ell} \in \mathcal{A}[y] \times \mathcal{A} \times \mathcal{A}$, with the $A_i$'s monic.
Output: the remainders $Q(f_i + r_iy) \operatorname{rem} A_i$ for $1 \leq i \leq \ell$.
$(\bar{A}_i)_{1\leq i\leq \ell} \leftarrow (\sum_{0\leq j\leq \delta_i} r_i^{\delta_i - j} a_{i,j}y^j)_{1\leq i\leq \ell}$
        where $(A_i)_{1\leq i\leq \ell} = (\sum_{0\leq j\leq \delta_i} a_{i,j}y^j)_{1\leq i\leq \ell}$ with $a_{i,\delta_i} = 1$
$(\hat{A}_i)_{1\leq i\leq \ell} \leftarrow (\bar{A}_i(y - f_i))_{1\leq i\leq \ell}$
$(\hat{R}_i)_{1\leq i\leq \ell} \leftarrow (Q \operatorname{rem} \hat{A}_i)_{1\leq i\leq \ell}$
$(R_i)_{1\leq i\leq \ell} \leftarrow (\hat{R}_i(f_i + r_iy))_{1\leq i\leq \ell}$
return $(R_i)_{1\leq i\leq \ell}$
```

Proposition 3.2. *Algorithm 2 is correct and uses*

$$O\tilde{~}(\deg(Q) + \deg(A_1 \cdots A_\ell))$$

operations in $\mathcal{A}$.

Proof. Let $i \in \{1, \ldots, \ell\}$. Since $\hat{A}_i$ is monic, the remainder $\hat{R}_i = Q \operatorname{rem} \hat{A}_i$ is well-defined, and $Q = P_i\hat{A}_i + \hat{R}_i$ with $\deg(\hat{R}_i) < \deg(Q)$ and $P_i \in \mathcal{A}[y]$. Then, we have

$$\begin{aligned} Q(f_i + r_iy) &= P_i(f_i + r_iy)\hat{A}_i(f_i + r_iy) + \hat{R}_i(f_i + r_iy) \\ &= P_i(f_i + r_iy)\bar{A}_i(r_iy) + R_i(y) \\ &= P_i(f_i + r_iy)r_i^{\delta}A_i(y) + R_i(y), \end{aligned}$$

which ensures $R_i = Q(f_i + r_iy) \operatorname{rem} A_i(y)$, hence the correctness.

Concerning the cost bound, the polynomial $\bar{A}_i$ is computed using at most $2\delta_i$ multiplications in $\mathcal{A}$, where $\delta_i = \deg(A_i)$, and then $\hat{A}_i$ is computed by fast shifting using $O\tilde{~}(\delta_i)$ operations in $\mathcal{A}$ [19, Theorem 9.15]. The conclusion follows, since fast remaindering can be used to compute all remainders $(\hat{R}_1, \ldots, \hat{R}_\ell)$ simultaneously in $O\tilde{~}(\deg(Q) + \delta_1 + \cdots + \delta_\ell)$ operations in $\mathcal{A}$. Indeed, we start by computing the subproduct tree in $O\tilde{~}(\delta_1 + \cdots + \delta_\ell)$ operations [19, Lemma 10.4], which gives us in particular the product $\hat{A}_1 \cdots \hat{A}_\ell$. Then, we compute the remainder $\hat{R} = Q \operatorname{rem} \hat{A}_1 \cdots \hat{A}_\ell$, which can be done in $O\tilde{~}(\deg(Q) + \delta_1 + \cdots + \delta_\ell)$ operations in $\mathcal{A}$ using fast division [19, Theorem 9.6]. Finally, the sought $\hat{R}_i = \hat{R} \bmod \hat{A}_i$ are computed by going down the subproduct tree, which costs $O\tilde{~}(\delta_1 + \cdots + \delta_\ell)$ operations in $\mathcal{A}$ [19, Corollary 10.17]. □

Now, let us describe how we implement the Hensel lifting strategy to manage to compute the sought affine factors without fully computing the shifts. In addition to the affine factors, we will make use of partial information on the inverse of the cofactor: we compute this inverse modulo the affine factor. Let $1 \leq i \leq \ell$ and assume that we have computed, at precision K,

- the affine factor $A_i \in \mathcal{S}_K[y]$ of $Q_i \bmod x^K$,
- $C_i = B_i^{-1} \operatorname{rem} A_i \in \mathcal{S}_K[y]$, where $B_i \in \mathcal{S}_K[y]$ denotes the cofactor such that $A_iB_i = Q_i \bmod x^K$.

Note that B_i is invertible as a polynomial of $\mathcal{S}_K[y]$ since by definition $B_{i\,|x=0} \in \mathbb{K}\backslash\{0\}$. Thus, our requirement is that the inverse of B_i coincides with C_i when working modulo A_i.

Now, we want to find similar polynomials when we increase the precision to $2K$. The main point concerning efficiency is that we will be able to do this by only considering computations modulo the affine factors A_i and their squares; remember that we control the sum of their degrees. In the algorithm, we increase for each

Algorithm 3 : AffineFacOfShifts

Input: a precision $d \in \mathbb{Z}_{>0}$, a polynomial $Q \in \mathcal{S}_d[y]$ such that $Q_{|x=0} \neq 0$, and triples $(f_i, t_i, m_i)_{1\le i\le \ell} \subset \mathcal{S}_d \times \mathbb{Z}_{>0} \times \mathbb{Z}_{>0}$.

Output: $(A_i, s_i)_{1\le i\le\ell}$ with $(A_i, s_i) = (0, d)$ if $Q(f_i + x^{t_i}y) = 0$ in $\mathcal{S}_d[y]$, and otherwise $s_i = v_x(Q(f_i + x^{t_i}y)) < d$ and $A_i \in \mathcal{S}_{d-s_i}[y]$ is the affine factor of $Q_i = x^{-s_i}Q(f_i + x^{t_i}y)$ at precision $d - s_i$.

Assumption: $A_i = 0$ or $\deg(A_i) \le m_i$ for $1 \le i \le \ell$.

```
1  $\mathcal{I} \leftarrow (1, \ldots, \ell)$        /* list of not yet computed factors */
2  $(R_i)_{1\le i\le\ell} \leftarrow$ ShiftedRem$(\mathcal{S}_d, Q, (y^{m_i+1}, f_i, x^{t_i})_{1\le i\le\ell})$
3  /* Process trivial affine factors */
4  for $1 \le i \le \ell$ such that $R_i = 0$ do
5      $(A_i, s_i) \leftarrow (0, d)$, and remove $i$ from $\mathcal{I}$
6  /* Set valuations and compute affine factors mod $x$ */
7  for $i \in \mathcal{I}$ do
8      $s_i \leftarrow v_x(R_i)$
9      $\bar{R}_i \in \mathbb{K}[y] \leftarrow (x^{-s_i}R_i)_{|x=0}$
10     $C_i \in \mathbb{K}\setminus\{0\} \leftarrow$ inverse of the leading coefficient of $\bar{R}_i$
11     $A_i \in \mathcal{S}_1[y] \leftarrow C_i\bar{R}_i$
12 /* Each iteration doubles the precision */
13 for $1 \le k \le \lceil \log_2(d) \rceil$ do
14     for $i \in \mathcal{I}$ such that $d - s_i \le 2^{k-1}$ do
15         remove $i$ from $\mathcal{I}$
16     $K \leftarrow 2^{k-1}$;  $(\delta_i)_{i\in\mathcal{I}} \leftarrow (\min(K, d - s_i - K))_{i\in\mathcal{I}}$
17     /* Lift the affine factors $(A_i)_i$ to precisions $\delta_i + K$ */
18     $(R_i)_{i\in\mathcal{I}} \leftarrow$ ShiftedRem$(\mathcal{S}_d, Q, (\bar{A}_i, f_i, x^{t_i})_{i\in\mathcal{I}})$,
19         where $\bar{A}_i$ is $A_i$ lifted into $\mathcal{S}_d[y]$
20     $(A_{i\top} \in \mathcal{S}_{\delta_i}[y])_{i\in\mathcal{I}} \leftarrow ((x^{-s_i-K}R_iC_i) \operatorname{rem} A_i)_{i\in\mathcal{I}}$,
21         with $x^{-s_i-K}R_i$, $C_i$, and $A_i$ truncated at precision $\delta_i$
22     $(A_i \in \mathcal{S}_{\delta_i+K}[y])_{i\in\mathcal{I}} \leftarrow (A_i + x^K A_{i\top})_{i\in\mathcal{I}}$
23     /* Find the cofactor inverses $(C_i)_i$ at precisions $\delta_i + K$ */
24     $(S_i)_{i\in\mathcal{I}} \leftarrow$ ShiftedRem$(\mathcal{S}_d, Q, (\bar{A}_i^2, f_i, x^{t_i})_{i\in\mathcal{I}})$,
25         where $\bar{A}_i$ is $A_i$ lifted in $\mathcal{S}_d[y]$
26     $(C_i \in \mathcal{S}_{\delta_i+K}[y])_{i\in\mathcal{I}} \leftarrow (((x^{-s_i}S_i) \operatorname{quo} A_i)^{-1} \operatorname{rem} A_i)_{i\in\mathcal{I}}$,
27         with $x^{-s_i}S_i$ and $A_i$ truncated at precision $\delta_i + K$
28 return $(A_i, s_i)_{1\le i\le\ell}$
```

i the precision from K to $K + \delta_i$, which is taken as the minimum of $2K$ and $d - s_i$: in the latter case, this is the last iteration which affects A_i, since it will be known at the wanted precision $d - s_i$.

First, we use fast remaindering to get $R_i = Q(f_i + x^{t_i}y) \operatorname{rem} A_i$ at precision d in x, simultaneously for all i (see Line 18); this gives us $Q_i \operatorname{rem} A_i = x^{-s_i}R_i \operatorname{rem} A_i$ at precision $d - s_i$, and thus $K + \delta_i$. Since A_i is the affine factor of Q_i at precision K, $Q_i \operatorname{rem} A_i$ is divisible by x^K.

We then look for $A_{i\top} \in \mathcal{S}_{\delta_i}[y]$ such that $\hat{A}_i = A_i + x^K A_{i\top}$ is the affine factor of Q_i at precision $K + \delta_i$; to ensure that $\hat{A}_i$ is still monic, we require that $\deg(A_{i\top}) < \deg(A_i)$. Thus, we can determine $A_{i\top}$ by working modulo A_i: having

$$(A_i + x^K A_{i\top})(B_i + x^K B_{i\top}) = Q_i,$$

at precision $K + \delta_i$, for some $B_{i\top} \in \mathcal{S}_{\delta_i}[y]$, implies that the identity

$$A_{i\top}B_i = x^{-K}Q_i$$

holds modulo A_i and at precision δ_i. Multiplying by $C_i = B_i^{-1}$ on both sides yields

$$A_{i\top} = (x^{-K}Q_iC_i) \operatorname{rem} A_i = (x^{-K-s_i}R_iC_i) \operatorname{rem} A_i \ .$$

Therefore, Line 20 and Line 22 correctly lift the affine factor of Q_i from precision K to precision $K + \delta_i$.

From now on, we work at precision $K + \delta_i$, and, as in the pseudo-code, we denote by A_i the affine factor obtained through the lifting step above (that is, $A_i \leftarrow \hat{A}_i$). Besides, let C_i now denote the cofactor inverse at precision $K + \delta_i$: $C_i = B_i^{-1} \operatorname{rem} A_i$, where $B_i \in \mathcal{S}_{K+\delta_i}[y]$ is the cofactor such that $Q_i = A_iB_i$. Our goal is to compute C_i, without computing B_i.

We remark that the remainder $S_i = Q(f_i + x^{t_i}y) \operatorname{rem} A_i^2$ (as in Line 24) is such that $x^{-s_i}S_i = Q_i \operatorname{rem} A_i^2 = A_i(B_i \operatorname{rem} A_i)$; $x^{-s_i}S_i$ is known at precision $d - s_i \ge K + \delta_i$. Thus,

$$(x^{-s_i}S_i) \operatorname{quo} A_i = B_i \operatorname{rem} A_i \ ,$$

and therefore C_i can be obtained as

$$C_i = B_i^{-1} \operatorname{rem} A_i = ((x^{-s_i}S_i) \operatorname{quo} A_i)^{-1} \operatorname{rem} A_i.$$

This shows that Line 26 correctly computes C_i at precision $K + \delta_i$.

Proposition 3.3. *Algorithm 3 is correct and uses*

$$\tilde{O}\big(d(\deg(Q) + m_1 + \cdots + m_\ell)\big)$$

operations in $\mathbb{K}$.

Proof. The correctness follows from the above discussion. Concerning the cost bound, we will use the following degree properties. Since A_i is monic, we have the degree bound $\deg(A_i) = \deg(A_{i\,|x=0}) \le m_i$ for all i and at any iteration of the loop; and since C_i is always computed modulo A_i, we also have $\deg(C_i) < m_i$.

The cost of the initialization (Line 1 to Line 11) is dominated by the computation of shifted remainders at Line 2, which costs $\tilde{O}(d(\deg(Q) + m_1 + \cdots + m_\ell))$ operations in $\mathbb{K}$ according to Proposition 3.2. The same cost bound holds for each call to ShiftedRem at Line 18 or Line 24, since we have $\deg(A_i) \le m_i$ and $\deg(A_i^2) \le 2m_i$.

At both Line 20 and Line 26, the degrees of R_i, C_i, and A_i are at most m_i, and $\delta_i \le \delta_i + K \le 2d$; thus the quotient and remainder computations use $\tilde{O}(d(m_1 + \cdots + m_\ell))$ operations in $\mathbb{K}$ according to [19, Theorem 9.6].

Finally, at Line 26 we are performing the inversion of the polynomial $((x^{-s_i}R_i) \operatorname{quo} A_i)$ modulo A_i; it is invertible in $\mathcal{S}_{\delta_i+K}[y]/(A_i)$ since its x-constant coefficient is a nonzero field element. As a consequence, this this inversion can be done in $\tilde{O}((\delta_i + K)\deg(A_i))$ field operations using Newton iteration [19, Theorem 9.4], and altogether Line 26 costs $\tilde{O}(d(m_1 + \cdots + m_\ell))$ operations in $\mathbb{K}$.

Summing these cost bounds over the $\lceil \log_2(d) \rceil$ iterations yields the announced total cost bound. □

4 FAST SERIES ROOTS ALGORITHM

In this section, we describe our fast algorithm for solving Problem 1. As explained above, this follows the divide and conquer strategy of Algorithm 1 while incorporating the fast computation of the affine factors of the shifts (Algorithm 3) in order to control the degrees of the polynomials that are passed as arguments to the recursive calls. Besides, we propagate the information of the multiplicities of the

Algorithm 4 : SeriesRoots∞

```
Input: $d \in \mathbb{Z}_{>0}$ and $Q \in \mathbb{K}[\![x]\!][y]$ such that $Q_{|x=0} \neq 0$.
Output: List of triples $(f_i, t_i, m_i)_{1 \le i \le \ell} \subset \mathbb{K}[x] \times \mathbb{Z}_{\ge 0} \times \mathbb{Z}_{>0}$ formed
    by a reduced root set of $Q$ to precision $d$ with their multiplicities.
if $d = 1$ then
    $(y_i, m_i)_{1 \le i \le \ell} \leftarrow$ roots w. multiplicity of $Q_{|x=0} \in \mathbb{K}[y]$
    return $(y_i, 1, m_i)_{1 \le i \le \ell}$
else
    $(f_i, t_i, m_i)_{1 \le i \le \ell} \leftarrow$ SeriesRoots∞$(Q, \lceil d/2 \rceil)$
    $(s_i)_{1 \le i \le \ell} \leftarrow (v_x(Q(f_i + x^{t_i} y)))_{1 \le i \le \ell}$
    $(A_i)_{1 \le i \le \ell} \leftarrow (\text{AffineFactor}(Q(f_i + x^{t_i} y)/x^{s_i}))_{1 \le i \le \ell}$
    for $1 \le i \le \ell$ do
        if $s_i \ge d$ then
            $(f_{i,1}, t_{i,1}, m_{i,1}) \leftarrow (0, 0, m_i)$ and $\ell_i \leftarrow 1$
        else
            $(f_{i,j}, t_{i,j}, m_{i,j})_{1 \le j \le \ell_i} \leftarrow$ SeriesRoots∞$(A_i, d - s_i)$
    return $(f_i + x^{t_i} f_{i,j}, t_i + t_{i,j}, m_{i,j})_{1 \le j \le \ell_i, 1 \le i \le \ell}$.
```

roots, which is then used as an input of Algorithm 3 to specify the list of degree upper bounds for the affine factors.

We start with a lemma, which states that taking affine factors preserves reduced root sets.

Lemma 4.1. *Let Q be in $\mathbb{K}[\![x]\!][y]$, with $Q_{|x=0} \neq 0$, and let $A \in \mathbb{K}[\![x]\!][y]$ be its affine factor. Then, any reduced root set of A at precision d is a reduced root set of Q at precision d.*

Proof. The claim follows from the factorization $Q = AB$, with $B_{|x=0} \in \mathbb{K} - \{0\}$. Indeed, as a result, $B(P)$ is a unit in $\mathbb{K}[[x]][y]$ for any P in $\mathbb{K}[[x]][y]$, whence $\mathcal{R}(Q,d) = \mathcal{R}(A,d)$ for any d; similarly, for any (f,t), $Q(f + x^t y)$ and $A(f + x^t y)$ have the same valuation, say s, and $Q(f + x^t y)/x^s$ and $A(f + x^t y)/x^s$ differ by a constant factor. In particular, if $\{(f_i, t_i)\}_i$ is a basic root set for A, it is a basic root set for Q, and the multiplicities of (f_i, t_i) in A and Q are the same. This implies that if $\{(f_i, t_i)\}_i$ is in fact a reduced root set for A, it remains so for Q. □

We continue with a procedure that operates on polynomials in $\mathbb{K}[\![x]\!][y]$, without applying any truncation with respect to x: as such, this is not an algorithm over $\mathbb{K}$, as it defines objects that are power series in x, but it is straightforward to prove that it outputs a reduced root set (remark that the procedure uses affine factors at "full precision", that is, in $\mathbb{K}[\![x]\!][y]$, so we do not use Algorithm 3 yet). In a second time, we will consider a truncated version of the same algorithm.

Proposition 4.2. *Algorithm 4 outputs a reduced root set of Q to precision d with their multiplicities.*

Proof. Correctness by induction on $d \ge 1$. By Lemma 2.7, the algorithm is correct for the induction base case $d = 1$. Take $d > 1$, and assume that the claim holds for all $d' < d$. Then, we obtain a reduced root set (f_i, t_i) from the first recursive calls, so in particular the valuations s_i are at least equal to $d' \ge 1$. This shows that $d - s_i < d$, so the second recursive call is made at a lower precision, and the procedure terminates.

By induction, in all cases, $(f_{i,j}, t_{i,j})_{1 \le j \le \ell_i}$ is a reduced root set of Q_i to precision $d - s_i$: this is obvious when $s_i \ge d$, and follows from Lemma 4.1 when $s_i < d$. Theorem 2.8 implies that $(f_i + x^{t_i} f_{i,j}, t_i + t_{i,j})_{1 \le j \le \ell_i, 1 \le i \le \ell}$ is a reduced root set of Q to precision d. We verify that the integers $m_{i,j}$ are the associated multiplicities as we did in the proof of that theorem. □

Algorithm 5 : SeriesRootsTrc

```
Input: $d \in \mathbb{Z}_{>0}$ and $Q^* \in \mathbb{K}[\![x]\!][y]$ reduced modulo $x^d$ such that
    $Q^*_{|x=0} \neq 0$.
Output: List of triples $(f_i, t_i, m_i)_{1 \le i \le \ell} \subset \mathbb{K}[x] \times \mathbb{Z}_{\ge 0} \times \mathbb{Z}_{>0}$ formed
    by a basic root set of $Q^*$ to precision $d$.
if $d = 1$ then
    $(y_i, m_i)_{1 \le i \le \ell} \leftarrow$ roots w. multiplicity of $Q^*_{|x=0} \in \mathbb{K}[y]$
    return $(y_i, 1, m_i)_{1 \le i \le \ell}$
else
    $(f_i, t_i, m_i)_{1 \le i \le \ell} \leftarrow$ SeriesRootsTrc$(Q^* \operatorname{rem} x^{\lceil d/2 \rceil}, \lceil d/2 \rceil)$
    $(A_i^*, s_i^*)_{1 \le i \le \ell} \leftarrow$ AffineFacOfShifts$(Q^*, d, (f_i, t_i, m_i)_{1 \le i \le \ell})$
    for $1 \le i \le \ell$ do
        if $s_i^* = d$ then
            $(f_{i,1}, t_{i,1}, m_{i,1}) \leftarrow (0, 0, m_i)$ and $\ell_i \leftarrow 1$
        else
            $(f_{i,j}, t_{i,j}, m_{i,j})_{1 \le j \le \ell_i} \leftarrow$ SeriesRootsTrc$(A_i^*, d - s_i^*)$

    return $(f_i + x^{t_i} f_{i,j}, t_i + t_{i,j}, m_{i,j})_{1 \le j \le \ell_i, 1 \le i \le \ell}$.
```

Next, we describe a similar algorithm, where we maintain the input polynomial with degree less than d in x (when it is the case, we say that it is *reduced modulo x^d*). To differentiate this version from the previous one and facilitate proof of correctness, we add a superscript * to the objects handled here, when they differ from their counterpart in Algorithm 4. Remark that we do not claim that the output forms a reduced root set of Q^*, merely a basic root set; we also do not claim that the m_i's in the output are the corresponding multiplicities.

Proposition 4.3. *Algorithm 5 outputs a basic root set of Q^* to precision d.*

Proof. We prove that for any Q and Q^* in $\mathbb{K}[\![x]\!][y]$, and $d > 0$, with $Q^* = Q \operatorname{rem} x^d$, the outputs of SeriesRoots∞ (Q,d) and SeriesRootsTrc (Q^*,d) are the same. Before giving the proof, remark that this will imply the claim in the proposition: we know that this output is a reduced, and thus basic, root set for Q to precision d. Since Q and Q^* are equal modulo x^d, one verifies easily that this output is thus a basic root set for Q^* to precision d as well.

The claim is proved by induction on d. If $d = 1$, the result is clear, as we compute the same thing on both sides.

For $d > 1$, since $Q^* \operatorname{rem} x^{\lceil d/2 \rceil} = Q \operatorname{rem} x^{\lceil d/2 \rceil}$, the induction assumption shows that $(f_i, t_i, m_i)_{1 \le i \le \ell}$ as computed in either SeriesRoots∞ or SeriesRootsTrc are the same.

The affine factors of the shifts of Q and Q^* differ, but they coincide at the precision we need. Indeed, the equality $Q = Q^* \bmod x^d$ implies that for all i, $Q(f_i + x^{t_i} y) = Q^*(f_i + x^{t_i} y) \bmod x^d$. In particular, if $s_i < d$, these two polynomials have the same valuation s_i, and $Q(f_i + x^{t_i} y)/x^{s_i} = Q^*(f_i + x^{t_i} y)/x^{s_i} \bmod x^{d - s_i}$, these implies that their affine factors are the same modulo $x^{d - s_i}$. If $s_i \ge d$, then $Q^*(f_i + x^{t_i} y)$ vanishes modulo x^d.

Remark that the assumption of Algorithm 3 is satisfied: for all i, m_i is the multiplicity of (f_i, t_i) in Q; the definition of a reduced root set then implies that $\deg(Q_{i\,|x=0}) \leq m_i$, so that the same degree bounds holds for the affine factors of $Q^*(f_i + x^{t_i}y)/x^{s_i}$. As a result, for i such that $s_i \geq d$, Algorithm 3 returns $(0, s_i^*) = (0, d)$, whereas if $s_i < d$, it returns (A_i^*, s_i), where A_i^* is the truncation modulo x^{d-s_i} of the affine factor A_i of Q_i. In the first case, the polynomials $(f_{i,1}, t_{i,1}, m_{i,1})$ are the same in both algorithms; in the second case, this is also true, by induction assumption. Our claim follows. □

To conclude the proof of Theorem 1.2, it remains to estimate the cost of Algorithm 5. Let $T(n, d)$ denote the cost of Algorithm 5 for input d and $n := \deg(Q)$. If $d = 1$, then clearly $T(n, 1) = \mathsf{R}_\mathsf{Y}(n)$. Otherwise, the cost is given by the following recursion:

$$T(n,d) = T(n,d/2) + S(n,d,(n_1,\ldots,n_\ell)) + \sum_{i=1}^{\ell} T(n_i, d - s_i)\ ,$$

where $S\Big(n,d,(n_1,\ldots,n_\ell)\Big)$ is the cost of $\mathsf{AffineFactorsOfShifts}$ and $n_i := \deg(A_i^*)$. The degrees of the polynomials A_i^*, in Algorithm 5, and A_i, in Algorithm 4, are the same, except for those cases where $s_i \geq d$ and A_i^* is actually zero. By definition of a reduced root set, we have

$$\sum_i \deg(A_i) \leq \deg(Q_{|x=0}) \leq n,$$

which thus implies $\sum_i n_i \leq n$, and $S(n,d,(n_1,\ldots,n_\ell)) \in \tilde{O}(dn)$. Note also that $s_i \geq d/2$ by the correctness of $\mathsf{SeriesRootsTrc}$. Since T is clearly at least linear in n, we then get $\sum_i T(n_i, d - s_i) \leq T(n, d/2)$ due to $\sum n_i \leq n$. This gives the relation $T(n,d) \leq 2T(n,d/2) + \tilde{O}(nd)$; we deduce that $T(n,d) = \tilde{O}(nd) + d\mathsf{R}_\mathsf{Y}(n)$, and our main theorem is proved.

Finally, we point out an optimization, which is not necessary to establish our main result, but useful in practice: once the affine factor of a shift has degree 1, there is no need to continue the recursion (the affine factor being monic, we can just read off its root from its constant coefficient). This is the analogue of the situation described in the introduction, when we know enough terms of the solution to make it possible to apply Newton iteration without further branching.

REFERENCES

[1] M. Alekhnovich. 2005. Linear Diophantine equations over polynomials and soft decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory* 51, 7 (July 2005), 2257–2265. DOI: http://dx.doi.org/10.1109/TIT.2005.850097

[2] D. Augot and L. Pecquet. 2000. A Hensel lifting to replace factorization in list-decoding of algebraic-geometric and Reed-Solomon codes. *IEEE Transactions on Information Theory* 46, 7 (2000), 2605–2614. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=887868

[3] J.-D. Bauch, E. Nart, and H. D. Stainsby. 2013. Complexity of OM factorizations of polynomials over local fields. *LMS Journal of Computation and Mathematics* 16 (2013), 139–171.

[4] J. Berthomieu, G. Lecerf, and G. Quintin. 2013. Polynomial root finding over local rings and application to error correcting codes. *Applicable Algebra in Engineering, Communication and Computing* 24, 6 (July 2013), 413–443. DOI: http://dx.doi.org/10.1007/s00200-013-0200-5

[5] D. V. Chudnovsky and G. V. Chudnovsky. 1986. On expansion of algebraic functions in power and Puiseux series, I. *J. Complexity* 2, 4 (1986), 271 – 294.

[6] D. V. Chudnovsky and G. V. Chudnovsky. 1987. On expansion of algebraic functions in power and Puiseux series, II. *J. Complexity* 3, 1 (1987), 1 – 25.

[7] D. Duval. 1989. Rational Puiseux expansions. *Composit. Math.* 70, 2 (1989), 119–154.

[8] V. Guruswami and M Sudan. 1999. Improved decoding of Reed–Solomon codes and algebraic-geometric codes. *IEEE Transactions on Information Theory* 45, 6 (1999), 1757–1767.

[9] H. T. Kung and J. F. Traub. 1978. All algebraic functions can be computed fast. *J. ACM* 25, 2 (1978), 245–260.

[10] J. Montes. 1999. *Polígonos de Newton de orden superior y aplicaciones aritméticas.* Ph.D. Dissertation. Universitat de Barcelona.

[11] D. R. Musser. 1975. Multivariate polynomial factorization. *J. ACM* 22, 2 (April 1975), 291–308. DOI: http://dx.doi.org/10.1145/321879.321890

[12] I. Newton. 1736. *The Method of Fluxions and Infinite Series.* Henry Woodfall.

[13] J. S. R. Nielsen and P. Beelen. 2015. Sub-quadratic decoding of one-point Hermitian codes. *IEEE Transactions on Information Theory* 61, 6 (June 2015), 3225–3240. DOI: http://dx.doi.org/10.1109/TIT.2015.2424415

[14] A. Poteaux and M. Rybowicz. 2011. Complexity bounds for the rational Newton-Puiseux algorithm over finite fields. *Appl. Algebra Engrg. Comm. Comput.* 22, 3 (2011), 187–217.

[15] A. Poteaux and M. Rybowicz. 2015. Improving complexity bounds for the computation of Puiseux series over finite fields. In *ISSAC'15*. ACM, 299–306.

[16] R.M. Roth and G. Ruckenstein. 2000. Efficient cecoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory* 46, 1 (2000), 246–257.

[17] V. Shoup. 1991. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *ISSAC'91*. ACM Press, 14–21.

[18] Madhu Sudan. 1997. Decoding of Reed–Solomon codes beyond the error-correction bound. *Journal of Complexity* 13, 1 (1997), 180–193.

[19] J. von zur Gathen and J. Gerhard. 2013. *Modern Computer Algebra* (3rd ed.). Cambridge University Press.

[20] R. J. Walker. 1978. *Algebraic Curves.* Springer-Verlag, New York. x+201 pages. Reprint of the 1950 edition.

[21] Y. Wu. 2008. New list decoding algorithms for Reed-Solomon and BCH codes. *IEEE Transactions on Information Theory* 54, 8 (2008), 3611–3630.

[22] O. Zariski and P. Samuel. 1960. *Commutative Algebra II.* Springer.