Accelerated tower arithmetic

Joris van der Hoeven, Grégoire Lecerf

Accelerated tower arithmetic

JORIS VAN DER HOEVEN[a], GRÉGOIRE LECERF[b]

CNRS (UMR 7161, LIX)
Laboratoire d'informatique de l'École polytechnique
Campus de l'École polytechnique
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France

a. *Email:* vdhoeven@lix.polytechnique.fr
b. *Email:* lecerf@lix.polytechnique.fr

Preliminary version of December 14, 2018

Nowadays, asymptotically fast algorithms are widely used in computer algebra for computations in towers of algebraic field extensions of small height. Yet it is still unknown how to reach softly linear time for products and inversions in towers of arbitrary height. In this paper we design the first algorithm for general ground fields with a complexity exponent that can be made arbitrarily close to one from the asymptotic point of view. We deduce new faster algorithms for changes of tower representations, including the computation of primitive element representations in subquadratic time.

KEYWORDS: complexity, algorithm, computer algebra, algebraic extension, algebraic tower, triangular set, accelerated tower.

1. INTRODUCTION

1.1. Statement of the problem

Let $\mathbb{A}$ be an effective commutative ring with unity. Here *effective* means that elements of $\mathbb{A}$ can be represented by concrete data structures and that we have algorithms for the ring operations, including the zero test. Effective fields are defined in a similar way.

Given a monic polynomial $\mu_1 \in \mathbb{A}[x]$ of degree d_1, the quotient ring $\mathbb{A}_1 := \mathbb{A}[x]/(\mu_1(x))$ is an effective ring and fast algorithms exist for the arithmetic operations in $\mathbb{A}_1$. More precisely, counting in terms of operations in $\mathbb{A}_0 := \mathbb{A}$, additions in $\mathbb{A}_1$ require d_1 additions in $\mathbb{A}$, whereas multiplications can be done [15] in almost linear time $\mathsf{M}(d_1) = O(d_1 \log d_1 \log\log d_1)$. Here $\mathsf{M}(d_1)$ is a standard notation for the cost of multiplying two univariate polynomials of degree d_1: see section 2.2.

Given another monic polynomial $\mu_2 \in \mathbb{A}_1[x]$, we may build the effective ring $\mathbb{A}_2 := \mathbb{A}_1[x]/(\mu_2(x))$ and fast arithmetic operations in $\mathbb{A}_2$ are available for the same reason. Doing so inductively, we obtain a *tower* $\mathbb{A}_0 \subseteq \mathbb{A}_1 \subseteq \mathbb{A}_2 \subseteq \cdots \subseteq \mathbb{A}_t$ of extensions of $\mathbb{A}$, with $\mathbb{A}_i = \mathbb{A}_{i-1}[x]/(\mu_i(x))$ for $i = 1, \ldots, t$. Such a tower is written $(\mathbb{A}_i)_{i \leqslant t}$ and we call t its *height*. Throughout this paper, we write α_i for the class of x in $\mathbb{A}_i = \mathbb{A}_{i-1}[x]/(\mu_i(x))$, we let $d_i := \deg \mu_i$, and $d := d_1 \cdots d_t$. The μ_i are called the *defining polynomials* of the tower and d its *degree*. Elements of $\mathbb{A}_i$ are naturally represented by univariate polynomials in α_i over $\mathbb{A}_{i-1}$ of degrees $< d_i$. If all $\mathbb{A}_i$ are fields, then we write $\mathbb{K}$ instead of $\mathbb{A}$ and $\mathbb{K}_i$ instead of $\mathbb{A}_i$, for convenience.

Towers of this type naturally arise in several contexts of applied algebra, including cryptography (for instance in [2, 3, 18]), error correcting codes (for instance in [26]), and the resolution of differentially algebraic systems in the more general setting of triangular sets (for instance in [1, 12]).

By induction over t, basic arithmetic operations in $\mathbb{A}_t$ can be naively performed in time $O(d\,(K \log d \log\log d)^t)$, for some constant $K > 1$. But it is well known that, for a sufficiently large constant $C > 1$, these operations can actually be carried out in time $O(C^t d \log d \log\log d)$ by means of Kronecker substitution (see for instance [27, Chapter 8]). However, if many of the individual degrees d_i are small, then t can become as large as $\log d / \log 2$, which leads to an overall complexity bound of the form $d^{1+\log C/\log 2+o(1)}$. The main aim of this paper is to prove the sharper bound $d^{1+o(1)}$ in the case when $\mathbb{A}_0 = \mathbb{K}$ is a field and the μ_i are irreducible and separable. The top level complexity result is stated in Corollary 4.11.

In order to simplify the presentation of complexity bounds, we often use the *soft-Oh* notation: $f(n) \in \tilde{O}(g(n))$ means that $f(n) = g(n) \log^{O(1)}(g(n)+3)$; see [27, Chapter 25, section 7] for technical details. The least integer larger or equal to x is written $\lceil x \rceil$; the largest one smaller or equal to x is written $\lfloor x \rfloor$. The $\mathbb{A}$-module of polynomials of degree $<d$ is written $\mathbb{A}[x]_{<d}$.

1.2. Modular composition and related problems

One essential tool for our new algorithms is modular composition. Before returning to our main problem, let us recall several useful existing results on this problem and various related topics.

Let $\mathbb{A}_1 = \mathbb{A}[x]/(\mu_1(x))$ be as above and assume that $t = 1$, whence $d = d_1$. Given $f, g \in \mathbb{A}[x]_{<d}$ the computation of the remainder $(f \circ g) \operatorname{rem} \mu_1$ of the Euclidean division of $f \circ g$ by μ_1 is called the problem of *modular composition*. It is equivalent to the problem of evaluating f at a point $a \in \mathbb{A}_1$. Currently, the best known solution to this problem is due to Brent and Kung [13, 56] and requires $O(d^\omega)$ operations in $\mathbb{A}$. Here the constant ω, with $3/2 < \omega < 2$, denotes an exponent such that a rectangular $d \times \sqrt{d}$ matrix can be multiplied with a square $\sqrt{d} \times \sqrt{d}$ matrix with $O(d^\omega)$ operations in $\mathbb{A}$. Huang and Pan showed in [40] that one may take $\omega < 1.667$. Brent and Kung's algorithm is based on the baby-step giant-step technique [50]; we will recall and generalize it in our section 3.

For a fixed point $a \in \mathbb{A}_1$, the evaluation map

$$\begin{aligned} \mathbb{A}^d &\to \mathbb{A}_1 \\ (f_0, \ldots, f_{d-1}) &\mapsto f_0 + \cdots + f_{d-1} a^{d-1} \end{aligned}$$

is linear. The dual or transposed map is given by

$$\begin{aligned} \mathbb{A}_1^* &\to \mathbb{A}^d \\ \ell &\mapsto (\ell(1), \ell(a), \ldots, \ell(a^{d-1})) \end{aligned}$$

and sends an $\mathbb{A}$-linear functional $\ell : \mathbb{A}_1 \to \mathbb{A}$ to the vector $(\ell(1), \ell(a), \ldots, \ell(a^{d-1}))$. The computation of this transposed map is called the problem of *modular power projection*. Using the transposition principle (see [10] and historical references therein), the cost of power projection and modular composition are essentially the same. In particular, the computation of the traces $\operatorname{Tr}_{\mathbb{A}_1/\mathbb{A}}(1), \ldots, \operatorname{Tr}_{\mathbb{A}_1/\mathbb{A}}(a^{d-1})$ corresponds to one modular power projection.

Given an element $a \in \mathbb{A}_1$, the *characteristic polynomial* of a is the characteristic polynomial of the multiplication endomorphism by a in $\mathbb{A}_1$. If $\mathbb{A} = \mathbb{K}$ is a field, then this polynomial can be computed in softly quadratic time by means of a resultant (see for instance [45, Corollary 29]). Shoup has also designed a practical randomized algorithm to compute minimal polynomials for the case when $\mathbb{K}$ has sufficiently many elements, with an expected complexity of $O\big(\mathsf{M}(d)\sqrt{d} + d^2\big)$.

The fastest known method to compute the characteristic polynomial χ of an element $a \in \mathbb{A}_1$ over $\mathbb{A}$ proceeds as follows: first compute the *traces* $\mathrm{Tr}_{\mathbb{A}_1/\mathbb{A}}(a^i)$ of the powers of a for all $i = 1, \ldots, d$ using modular power projection, and then recover χ by integrating the differential equation

$$-\frac{\tilde{\chi}'(z)}{\tilde{\chi}(z)} = \sum_{i=0}^{d-1} \mathrm{Tr}_{\mathbb{A}_1/\mathbb{A}}(a^{i+1})\, z^i + O(z^d), \tag{1.1}$$

where $\tilde{\chi}(z) = z^d \chi(1/z)$ represents the *reverse polynomial* of χ. This integration requires to have the inverses of $2, \ldots, d$ in $\mathbb{A}$ at our disposal. Historically, this method goes back to Le Verrier [43], and formula (1.1), often called the Newton–Girard formula, expresses the relationship between symmetric and power sum polynomials. The integration step takes softly linear time: the seminal contributions are due to Brent and Kung [13], generalizations can be found in [8], the first efficient extension to finite fields has been designed in [9], and we refer the reader to [29, section 2] for a concise proof of it.

1.3. Previous work on fast tower arithmetic

Recall that an element $\beta \in \mathbb{K}_t$ is said to be *primitive* over $\mathbb{K}$ if $\mathbb{K}_t = \mathbb{K}[\beta]$. The primitive element theorem states that such an element β always exists: if $\mathbb{K}$ contains sufficiently many elements, then it suffices to take a sufficiently generic $\mathbb{K}$-linear combination of the α_i. In this light, it may seem odd at first sight to work with towers $(\mathbb{K}_i)_{i \leqslant t}$ of height $t \geqslant 2$, since an isomorphic tower of height one always exists. The problem is that both the computation of primitive elements and conversions between representations are usually expensive.

Concerning primitive elements, it is currently not known how to efficiently find one, together with its minimal polynomial ν and polynomials $\varphi_i \in \mathbb{K}[x]_{<d}$ such that $\alpha_i = \varphi_i(\beta)$ for $i = 1, \ldots, t$. In fact, naive algorithms mostly boil down to linear algebra in dimension d, and thus feature at least quadratic costs. One may for instance appeal to the FGLM algorithm to change orderings of Gröbner bases [21, 22].

If a modular composition algorithm (technically, for the multivariate version of section 3.2) of softly linear complexity were known over any field, then primitive element representations of towers would be computable in softly linear expected time by a randomized algorithm; conversions between towers and their primitive element representations would also become possible in softly linear time. These aspects have been studied by Poteaux and Schost in [51, 52], where subquadratic costs are achieved for multivariate modular composition and its transpose, as well as for computing primitive representations of towers.

Let $\varepsilon > 0$ represent a fixed positive rational value. If $\mathbb{K} = \mathbb{F}_q$ is the finite field with q elements, then a major result by Kedlaya and Umans [42] states that modular composition and power projection are possible in time $(d \log q)^{1+\varepsilon}$. Whenever $t = 1$, then this in particular implies that characteristic polynomials can be obtained with expected bit complexity $(d \log q)^{1+\varepsilon}$: see [29, sections 2.3–2.5] for details. Based on these results, Poteaux and

Schost have derived [51] fast algorithms for multiplication, division, traces and primitive element representations for separable towers over finite fields. Unfortunately, very large constant factors are hidden in the complexities of all these algorithms, which currently prevent them from being of any practical relevance.

For some other particular cases of interest, fast algorithms also exist for modular composition and related problems. For fixed irreducible moduli $h \in \mathbb{F}_q[x]$ of sufficiently smooth degree $d_1 \cdots d_t$, practically faster algorithms for modular composition have been proposed in [39]. When $\mathbb{K} \subseteq \mathbb{C}$, we have also shown in [38] that modular composition can be achieved in quasi-linear time when computing with a sufficiently high numeric precision.

More direct approaches for algebraic tower arithmetic are usually based on computations modulo so-called "triangular sets". We may see the preimage of μ_i over $\mathbb{A}$ as a multivariate polynomial T_i in $\mathbb{A}[x_1, ..., x_i]$ such that T_i is monic in x_i, T_i has degree d_i in x_i, degrees $< d_j$ in x_j for all $j = 1, ..., i-1$, and $\mu_i(x) = T_i(\alpha_1, ..., \alpha_{i-1}, x)$. The sequence $(T_i)_{i \leqslant t}$ forms a special case of a *triangular set*. Triangular sets and decompositions have a long history that goes back to Ritt's contributions in differential algebra [53].

In this context of triangular sets, it was shown in [46] that two elements in $\mathbb{A}_t$ can be multiplied in time $O(4^t d \log d \log\log d)$. In [44, Proposition 2], Lebreton has proved the stronger bound $O(3^t d \log d \log\log d)$. This result even applies if the ideal $(T_1, ..., T_t)$ is not radical, under the condition that $T_1, ..., T_t$ form a *regular chain* (in the sense of Kalkbrener [41], see also [1] for generalities about triangular sets). For triangular sets of certain quite special types, we notice that even faster multiplication algorithms have been designed in [7]. When working over finite fields, practically efficient algorithms have also been proposed in [17], based on fast embeddings into towers of a specific type.

Another major occurrence of algebraic towers in the literature concerns "dynamic evaluation". This technique was introduced by Della Dora, Dicrescenzo and Duval [19, 20] as a way to compute with algebraic numbers without requiring algorithms for polynomial factorization. Basically, the idea is to compute in $\mathbb{A}_t$ as if it were a field, and cases are distinguished whenever a zero test of an element a is requested: the "current tower" is then explicitly decomposed into a "direct product of two towers" such that the projections of a in these two towers are respectively zero and invertible. The computations finally resume non-deterministically with each of the two new towers in the role of the current tower. For recent complexity results on this technique we refer the reader to [16].

1.4. Our contributions

The efficient computation with polynomials modulo zero-dimensional ideals defined by triangular sets is an important topic in applied algebra. Obtaining softly linear complexity bounds when working over a general field is an open problem. The previous best known bound due to Lebreton [44, section 3.2] is recalled in section 2.4.

Then, one first technical contribution of us concerns multivariate modular composition: in section 3.2 we design a new baby-step giant-step algorithm to evaluate $f(g_1, ..., g_t) \operatorname{rem} h$ with the same kind of complexity as the Brent–Kung algorithm for the univariate case. It slightly improves upon the incremental method described in [52], that actually relies on the bivariate case. In fact, our method applies the baby-step giant-step paradigm over the whole representation of f. Compared to [52, Lemma 4] used with $\mathsf{C}(\delta) = O(\delta^\omega)$, the complexity bound given in Proposition 3.3 does not have restrictions on the characteristic, and is asymptotically smaller by a factor of t.

The main contribution of this paper is section 4, which is devoted to a new deterministic algorithm to multiply in towers of separable field extensions (all the μ_i are thus irreducible and separable). For the first time we achieve an asymptotic complexity $d^{1+o(1)}$ with an exponent that becomes arbitrarily close to one. From the asymptotically complexity point of view, and in the specific case of towers of separable field extensions, this improves upon [44, 52], and also upon the randomized algorithms for finite fields from [51]. For specific towers over finite fields, some of the algorithms from [17] remain more efficient both in practice and in theory.

The main idea behind our algorithms is as follows: we use the primitive element theorem to replace a general tower of height t by a new "accelerated" tower for which we can control the height s. There is a tradeoff between the cost of tower arithmetic (more expensive for larger t) and primitive element computations (more expensive for small s). By making a suitable compromise, we obtain our main complexity bound.

One shortcoming of our main result is that it only applies to towers of *fields*. Now if an algorithm is available for factoring polynomials in $\mathbb{K}[x]$ into irreducibles, then our results generalize to towers of separable integral extensions over $\mathbb{K}$, by "factoring" them into towers of separable field extensions. This reduction and the corresponding conversion algorithms (that generalize Chinese remaindering) are detailed in section 5. Some particular cases when the cost of factorization is often affordable are discussed in section 5.5.

Our final section 6 contains a general subquadratic Las Vegas complexity bound for computing primitive element representations of towers of separable field extensions, along with the related data for conversions: our algorithms directly benefit from the fast product, and they rely on power projections as in [11, 42, 51]. Compared to [52, Lemma 4] used with $\mathsf{C}(\delta) = O(\delta^\omega)$, section 6 gains a factor of t asymptotically, without restrictions on the characteristic. When working over a finite field, section 6 does not improve upon the asymptotic complexity bounds from [51]. Nevertheless, [51] relies on the Kedlaya–Umans algorithm for modular composition, so we expect our approach to behave better in practice.

Let us finally stress once more that we regard our contribution as a uniform way to prove almost optimal complexity bounds for tower arithmetic *in the algebraic complexity model*. It is well known that this model is reasonably close to bit complexity models when working over finite fields. Over other fields such as $\mathbb{Q}$, coefficient growth needs to be taken into account, which leads us beyond the scope of this paper. From a practical perspective, we also recall that efficient tower arithmetic has been developed over various specific kinds of base fields such as finite fields and subfields of $\mathbb{C}$: see [17, 38, 39] and section 5.5. It is true however that actual implementations of our own algorithms have fallen somewhat behind. We intend to address such implementation issues in the near future.

2. BASIC TOWER ARITHMETIC

This section gathers basic prerequisites on complexity models and polynomial arithmetic. In particular we recall Lebreton's results on multiplication in towers [44]. Then we examine the costs of divisions.

2.1. Complexity model

Let $\mathbb{A}$ be an effective ring. Our complexity analyses concern the *algebraic complexity model* [14, Chapter 4] over $\mathbb{A}$, and more precisely straight-line programs and computation trees. In other words, running times of algorithms are measured in terms of the required number of ring operations in $\mathbb{A}$, and constants are thought to be freely at our disposal.

It is convenient to introduce the notations $\mathsf{m}_{\mathbb{A}}$ and $\mathsf{d}_{\mathbb{A}}$ as abstractions for the respective costs of multiplication and division in $\mathbb{A}$ (whenever defined). For simplicity we always assume that multiplication is at least as expensive as addition, subtraction, and zero testing.

For randomized algorithms over a finite effective ring $\mathbb{A}$, we assume a special instruction that uniformly generates random elements in $\mathbb{A}$, with constant cost. For a given input, the cost of a randomized algorithm is thus a random variable. The *expected cost* for input size s is defined as the maximum of the averages of these random variables over all possible inputs of size $\leqslant s$.

Remark 2.1. For arithmetic operations in basic rings and fields, such as $\mathbb{Z}/r\mathbb{Z}$ or $\mathbb{F}_q[x]$, it is common to use Turing or RAM machines to count the total number of "bit operations". The translation of our results in terms of bit complexity is mostly straightforward, although some care is needed for the manipulation of multivariate polynomials on Turing machines. We refer to [36, 37] for more technical details.

2.2. Univariate polynomial multiplication

Let $\mathbb{A}$ be an effective commutative ring with unity. We denote by $\mathsf{M}_{\mathbb{A}}(d)$ a cost function for multiplying two polynomials $f, g \in \mathbb{A}[x]_{<d}$. For general $\mathbb{A}$, it has been shown in [15] that one may take

$$\mathsf{M}_{\mathbb{A}}(d) = O(\mathsf{m}_{\mathbb{A}}\, d \log d \log\log d). \tag{2.1}$$

For rings of positive characteristic, one has

$$\begin{aligned} \mathsf{M}_{\mathbb{A}}(d) &= O(\mathsf{m}_{\mathbb{A}}\, d \log d\, 4^{\log^* d}) \\ \log^* d &= \min\left\{k \in \mathbb{N} : \log \circ \overset{k\times}{\cdots} \circ \log d \leqslant 1\right\}, \end{aligned}$$

by [30]. We make the following assumptions:

- $\mathsf{M}_{\mathbb{A}}(d)/d$ is a non-decreasing function in d.
- $\mathsf{M}_{\mathbb{A}}$ is sufficiently close to linear, in the sense that

$$\frac{\mathsf{M}_{\mathbb{A}}(m\,d)}{m\,d} = O\left(\frac{\mathsf{M}_{\mathbb{A}}(d)}{d}\right) \tag{2.2}$$

  holds whenever $m \leqslant d$.

These assumptions hold for $\mathsf{M}_{\mathbb{A}}(d)$ as in (2.1). Notice that (2.2) is also equivalent to $\mathsf{M}_{\mathbb{A}}(m\,d) = O(m\,\mathsf{M}_{\mathbb{A}}(d))$. Applying equation (2.2) a finite number of times, it follows that $\mathsf{M}_{\mathbb{A}}(m^k d) = O(m^k\,\mathsf{M}_{\mathbb{A}}(d))$ for any fixed positive integer k.

If $\mathbb{B}$ is an effective $\mathbb{A}$-algebra that is also a finite dimensional free $\mathbb{A}$-module, then we denote by $\mathsf{M}_{\mathbb{B}/\mathbb{A}}(d)$ a cost function for multiplying two polynomials in $\mathbb{B}[x]_{<d}$, in terms of the number of operations in $\mathbb{A}$, and we make the same assumptions as for $\mathsf{M}_{\mathbb{A}}$. Notice that we may always take $\mathsf{M}_{\mathbb{B}}(d) = O(\mathsf{m}_{\mathbb{A}}\,\mathsf{M}_{\mathbb{B}/\mathbb{A}}(d))$.

In particular, if μ is a monic polynomial in $\mathbb{A}[x]$ of degree d then $\mathbb{B} = \mathbb{A}[x]/(\mu(x))$ is such an $\mathbb{A}$-algebra of dimension d; elements in $\mathbb{B}$ are represented by polynomials in $\mathbb{A}[x]_{<d}$. In this case, we have $\mathsf{m}_{\mathbb{B}} = O(\mathsf{M}_{\mathbb{A}}(d))$ and $\mathsf{M}_{\mathbb{B}}(n) = O(\mathsf{M}_{\mathbb{A}}(n\,d))$ by means of Kronecker substitution [27, Chapter 8].

2.3. Primitive towers

If $\mathbb{B}$ is a finitely generated $\mathbb{A}$-algebra, an element $\beta \in \mathbb{B}$ is said to be *primitive* if $\mathbb{B} = \mathbb{A}[\beta]$. In this case, we write ν the monic minimal polynomial of β, so the following isomorphism holds:

$$\begin{aligned} \mathbb{A}[x]/(\nu(x)) &\cong \mathbb{B} \\ x &\mapsto \beta. \end{aligned}$$

Notice that such a primitive element representation does not necessarily exist: consider $\mathbb{A} = \mathbb{Q}$ and $\mathbb{B} = \mathbb{A}[x,y]/(x^2, xy, y^2)$.

DEFINITION 2.2. *A **primitive tower representation** of $(\mathbb{A}_i)_{i\leqslant t}$ consists of the following data:*

- *A primitive element β_i of $\mathbb{A}_i$ over $\mathbb{A}$, for $i = 1, \ldots, t$.*
- *The minimal polynomial $\nu_i \in \mathbb{A}[x]$ of β_i over $\mathbb{A}$, for $i = 1, \ldots, t$.*
- *$\phi_{i,j} \in \mathbb{A}[x]_{<d_1 \cdots d_i}$ such that $\alpha_j = \phi_{i,j}(\beta_i)$, for $i = 1, \ldots, t$ and $j = 1, \ldots, i$.*

These data induce the following isomorphisms for $i = 1, \ldots, t$:

$$\begin{aligned} \mathbb{A}_i &\cong \mathbb{A}[x]/(\nu_i(x)) \\ \alpha_j &\mapsto \phi_{i,j}(x), \quad \text{for } j = 1, \ldots, i \\ \beta_i &\leftarrow\!\shortmid\ x. \end{aligned}$$

The polynomials $\phi_{i,j}$ are called *parametrizations* of the α_j in terms of the β_i.

With the triangular set $(T_i)_{i\leqslant t}$ as defined in the introduction, an element $a \in \mathbb{A}_i$ is represented by a polynomial $f(\alpha_1, \ldots, \alpha_i)$ with $f \in \mathbb{A}[x_1, \ldots, x_i]$ defined modulo $(T_1, \ldots, T_i)$. So the conversion of a into an element of $\mathbb{A}[x]/(\nu_i(x))$ boils down to evaluating $f(\phi_{i,1}, \ldots, \phi_{i,i})$ modulo ν_i. The backward conversion consists in evaluating a univariate polynomial $f \in \mathbb{A}[x]_{<d_1 \cdots d_i}$ at β_i. Such conversions are the topic of section 3 below.

Remark 2.3. Special kinds of primitive tower representations have been used before; see [18, 39], for instance. Algorithms therein use special routines for conversions between $\mathbb{A}[\beta_i][\alpha_{i+1}]$ and $\mathbb{A}[\beta_{i+1}]$. In the present paper, we only rely on conversions between $\mathbb{A}_i$ and $\mathbb{A}[\beta_i]$, the natural identity isomorphism $\mathbb{A}_{i+1} = \mathbb{A}_i[\alpha_{i+1}]$, and combinations of these conversions.

2.4. Multiplication in towers

Under the assumptions of section 2.2, recall that there exists a constant $C > 1$ such that $\mathsf{m}_{\mathbb{A}_i} \leqslant C\,\mathsf{M}_{\mathbb{A}_i/\mathbb{A}_{i-1}}(d_i)$ for $i = 1, \ldots, t$, and where $\mathsf{M}_{\mathbb{A}_i/\mathbb{A}_{i-1}}(d)$ represents a cost function for multiplying two polynomials in $\mathbb{A}_i[x]_{<d}$, in terms of the number of operations in $\mathbb{A}_{i-1}$. When using this bound in an iterated fashion, we obtain

$$\mathsf{m}_{\mathbb{A}_i/\mathbb{A}_0} = C^i\,\mathsf{M}_{\mathbb{A}_1/\mathbb{A}_0}(d_1) \cdots \mathsf{M}_{\mathbb{A}_i/\mathbb{A}_{i-1}}(d_i).$$

By taking care of the cost of polynomial divisions in terms of multiplications, one may prove the following sharper bounds:

PROPOSITION 2.4. *If $d_i \geqslant 2$ for $i = 1, \ldots, t$, then there exists a constant $1 < C \leqslant 3$ such that*

$$\begin{aligned} \mathsf{m}_{\mathbb{A}_t} &= O(C^t \mathsf{M}_{\mathbb{A}}(d)) \\ \mathsf{M}_{\mathbb{A}_t}(n) &= O(C^t \mathsf{M}_{\mathbb{A}}(nd)). \end{aligned}$$

Proof. Taking $C = 3$, Lebreton proved that $\mathsf{m}_{\mathbb{A}_t} = O(\mathsf{M}_{\mathbb{A}}(3^t d))$; see [44, section 3.2]. His proof was done for the particular cost function $\mathsf{M}_{\mathbb{A}}(n) = O(n \log n \log\log n)$. For completeness, we briefly repeat this proof, for a general cost function $\mathsf{M}_{\mathbb{A}}(n)$ such that $n \mapsto \mathsf{M}_{\mathbb{A}}(n)/n$ is non-decreasing. We recall that two polynomials $P, Q \in \mathbb{A}[x_1, \ldots, x_t]$ of partial degrees $< d_i$ in the x_i can be multiplied in time $O(\mathsf{M}_{\mathbb{A}}(2^t d))$ using Kronecker substitution recursively; see [27, Chapter 8] and [44, section 3.2].

We let $T_1(x_1), \ldots, T_t(x_1, \ldots, x_t)$ represent the triangular set associated to the tower, as defined in section 1.3. We identify elements in the tower with their canonical representatives in $\mathbb{A}[x_1, \ldots, x_t]$. Let $\tilde{T}_i(x_1, \ldots, x_i) := x_i^{d_i} T_i(x_1, \ldots, x_{i-1}, 1/x_i)$ represent the reverse polynomial of T_i in x_i. Assume for now that we precomputed polynomials $S_i(x_1, \ldots, x_i)$ in $\mathbb{A}[x_1, \ldots, x_i]$, of partial degrees $< d_j$ in the x_j, such that

$$S_i(x_1, \ldots, x_i)\, \tilde{T}_i(x_1, \ldots, x_i) = 1 \bmod (T_1, \ldots, T_{i-1}, x_i^{d_i}).$$

Let $\mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_i)$ represent a cost function for reducing polynomials in $\mathbb{A}[x_1, \ldots, x_i]$ of degrees $\leqslant 2(d_j - 1)$ in x_j for $j = 1, \ldots, i$ modulo $(T_1, \ldots, T_i)$. The reduction of such a polynomial P modulo $(T_1, \ldots, T_t)$ is done recursively, as follows:

1. Let $e := \deg_{x_t} P$. If $e < d_t$, then we reduce P modulo $(T_1, \ldots, T_{t-1})$ and we are done.
2. Otherwise, let $\tilde{P}(x_1, \ldots, x_t) := x_t^e P(x_1, \ldots, x_{t-1}, 1/x_t)$ be the reverse of P.
   a. Reduce $\tilde{P}$ rem $x_t^{e-d_t+1}$ modulo $(T_1, \ldots, T_{t-1})$, in time $\leqslant d_t \mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_{t-1})$.
   b. Compute $\tilde{Q} := S_t \tilde{P}$ rem $x_t^{e-d_t+1}$, in time $O(\mathsf{M}_{\mathbb{A}}(2^t d))$.
   c. Reduce $\tilde{Q}$ modulo $(T_1, \ldots, T_{t-1})$, in time $\leqslant d_t \mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_{t-1})$.
3. Let $Q(x_1, \ldots, x_t) := x_t^{e-d_t} \tilde{Q}(x_1, \ldots, x_{t-1}, 1/x_t)$, compute $R := P - Q\, T_t$ rem $x_t^{d_t}$, and then reduce R modulo $(T_1, \ldots, T_{t-1})$. It turns out that R equals P modulo $(T_1, \ldots, T_t)$; see for instance [27, Chapter 9, section 1].

Altogether, the cost $\mathsf{R}_{\mathbb{A}}$ of this algorithm is therefore bounded by

$$\mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_t) \leqslant 2\, \mathsf{M}_{\mathbb{A}}(2^t d) + c\, 2^t d + 3\, \mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_{t-1})\, d_t$$

for some universal constant $c > 0$. We deduce that $\mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_t) = O(\mathsf{M}_{\mathbb{A}}(3^t d))$.

Now let A and B be the respective representatives of two elements a and b to be multiplied in the tower. The product $P := A\, B$ takes $O(\mathsf{M}_{\mathbb{A}}(2^t d))$, and P is then reduced modulo $(T_1, \ldots, T_t)$, whence

$$\mathsf{m}_{\mathbb{A}_t} = \mathsf{M}_{\mathbb{A}}(2^t d) + \mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_t) = O(\mathsf{M}_{\mathbb{A}}(3^t d)).$$

For the second bound of the proposition, we may multiply two polynomials in $\mathbb{A}_t[y]_{<n}$ as in $\mathbb{A}[x_1, \ldots, x_t][y]$ with $O(\mathsf{M}_{\mathbb{A}}(2^t n d))$ operations in $\mathbb{A}$, and then reduce their product coefficientwise modulo $(T_1, \ldots, T_t)$ with cost $\leqslant 2n \mathsf{R}_{\mathbb{A}}(d_1, \ldots, d_t)$, whence $\mathsf{M}_{\mathbb{A}_t}(n) = O(\mathsf{M}_{\mathbb{A}}(3^t n d))$.

It remains to observe that once $S_1, \ldots, S_{t-1}$ are known, the precomputation of S_t using Newton's method takes $O(\mathsf{M}_{\mathbb{A}_{t-1}}(d_t)) = O(\mathsf{M}_{\mathbb{A}}(3^t d))$ operations in $\mathbb{A}$; see [27, section 9.1], for instance. Therefore, the complexity bounds obtained so far remain valid up to a constant factor when taking the cost for obtaining the S_i into account. Finally, our assumption $d_i \geqslant 2$ for $i = 1, \ldots, t$ implies $3^t = O(d^2)$, so (2.2) leads to the claimed bounds. □

Remark 2.5. The constant C does not play a critical role in the design of the forthcoming algorithms. Of course any improvement of C remains relevant; the dependence on C will be made apparent in Corollary 4.11.

Remark 2.6. The assumption that $d_i \geqslant 2$ for all i is convenient for proving several complexity bounds. For i with $d_i = 1$, we notice that α_i does not occur in the representation of elements in $\mathbb{K}_i$, so extensions of degree one can be suppressed from the tower without loss of generality. The cost of the corresponding rewritings does not intervene in the algebraic complexity model. On a Turing or RAM machine this cost would actually depend on the way how multivariate polynomials are represented, but it is typically at most linear in the size of the representation.

2.5. Division in towers

In the remainder of this section, we assume that we work in a tower of fields $(\mathbb{K}_i)_{i\leqslant t} = (\mathbb{A}_i)_{i\leqslant t}$ over $\mathbb{K} = \mathbb{A}$. Given $f, g \in \mathbb{K}[x]_{<d}$ such that g is monic, it is well known [27, Chapter 9] that the quotient $f \operatorname{quo} g$ and the remainder $f \operatorname{rem} g$ of the Euclidean division of f by g can be computed in time $O(\mathsf{M}_{\mathbb{K}}(d))$.

It is important for us that the gcd algorithm with input $f_1, f_2 \in \mathbb{K}[x]_{\leqslant d}$ returns the monic polynomial $g = \gcd(f_1, f_2)$ along with $u_1, u_2 \in \mathbb{K}[x]_{<d}$ such that $g = u_1 f_1 + u_2 f_2$. In this way, if $g = 1$ then u_1 is the inverse of f_1 modulo f_2 and u_2 is the inverse of f_2 modulo f_1. It is well known [27, Chapter 11, Algorithm 11.6] that this extended gcd can be performed in time $\mathsf{d}_{\mathbb{K}}(d+1) + O(\mathsf{M}_{\mathbb{K}}(d)\log d)$.

In fact, when replacing all polynomial divisions by pseudo-divisions in [27, Chapter 11], we avoid all divisions in $\mathbb{K}$, but the computed gcd g is not necessarily monic. Multiplying g, u_1 and u_2 with the inverse of the leading coefficient of g, we do obtain a monic gcd, at the expense of a single division in $\mathbb{K}$. Summarizing, this shows that the extended monic gcd can actually be computed in time $\mathsf{d}_{\mathbb{K}} + O(\mathsf{M}_{\mathbb{K}}(d)\log d)$.

Now consider an extension $\mathbb{L} := \mathbb{K}[x]/(\mu(x))$ of $\mathbb{K}$, where $\mu \in \mathbb{K}[x]$ is a monic irreducible polynomial of degree d. Then the above bounds for division and gcd computations yield

$$\mathsf{d}_{\mathbb{L}} \;=\; \mathsf{d}_{\mathbb{K}} + O(\mathsf{M}_{\mathbb{K}}(d)\log d).$$

When using the univariate algorithm for inverting non-zero elements in $\mathbb{K}_i$ in a recursive manner, we obtain the following complexity bound:

PROPOSITION 2.7. *Let $\bar{d} := \max(d_1, \ldots, d_t)$ and assume that $d_i \geqslant 2$ for $i = 1, \ldots, t$. With the above assumptions and notations, and with $1 < C \leqslant 3$ as in Proposition 2.4, we have*

$$\mathsf{d}_{\mathbb{K}_t} \;=\; \mathsf{d}_{\mathbb{K}} + O(C^t \mathsf{M}_{\mathbb{K}}(d)\log \bar{d}).$$

Proof. Let $A \geqslant 1$ and $B \geqslant 1$ be universal constants with $\mathsf{M}_{\mathbb{K}_i}(n) \leqslant A C^i \mathsf{M}_{\mathbb{K}}(n d_1 \cdots d_i)$ for all i and $\mathsf{d}_{\mathbb{L}} \leqslant \mathsf{d}_{\mathbb{K}} + B\,\mathsf{M}_{\mathbb{K}}(d)\log d$ for all extensions $\mathbb{L}$ of $\mathbb{K}$ as previously considered. Then we have

$$\begin{aligned} \mathsf{d}_{\mathbb{K}_i} &\leqslant \mathsf{d}_{\mathbb{K}_{i-1}} + B\,\mathsf{M}_{\mathbb{K}_{i-1}}(d_i)\log d_i \\ &\leqslant \mathsf{d}_{\mathbb{K}_{i-1}} + A B C^{i-1} \mathsf{M}_{\mathbb{K}}(d_1 \cdots d_i)\log d_i \end{aligned}$$

and we conclude by induction. □

Let us mention that inversion modulo a general triangular set (that does not determine a prime ideal) is more subtle; see various approaches in [16, 49].

3. EVALUATING POLYNOMIALS AT POINTS IN ALGEBRAS

Throughout this section $\mathbb{A}$ represents an effective ring and $\mathbb{B}$ is an effective $\mathbb{A}$-algebra with a given basis $b_1, \ldots, b_r$. Given a polynomial $f \in \mathbb{A}[x_1, \ldots, x_n]$ and a point $(a_1, \ldots, a_n) \in \mathbb{B}^n$, we study how to compute $f(a_1, \ldots, a_n)$ efficiently. We first recall the well known baby-step giant-step algorithm in the case when $n = 1$. In section 4 below, these evaluation algorithms will be used for conversions between triangular and primitive representations of the same algebra.

3.1. Univariate baby-step giant-step method

Given a polynomial $f = f_0 + \cdots + f_{d-1} x^{d-1} \in \mathbb{A}[x]_{<d}$ and $a \in \mathbb{B}$, how to evaluate f efficiently at a? Horner's method uses time $O(\mathsf{m}_{\mathbb{B}}\, d)$. For convenience of the reader, we now recall a well known faster algorithm from [50] that relies on the baby-step giant-step technique.

Algorithm 3.1
Input: $f \in \mathbb{A}[x]_{<d}$ and $a \in \mathbb{B}$.
Output: $f(a) \in \mathbb{B}$.

1. Let $p := \lfloor \sqrt{d} \rfloor$ and $q := \lceil d/p \rceil$.
2. For $0 \leqslant i < p$ do:
   a. Compute and decompose $a^i = M_{1,i} b_1 + \cdots + M_{r,i} b_r$.
   (This yields an $r \times p$ matrix $M \in \mathbb{A}^{r \times p}$.)
   b. For $0 \leqslant j < q$, let $N_{i,j} := f_{i+pj}$.
   (This yields a $p \times q$ matrix $N \in \mathbb{A}^{p \times q}$.)
3. Compute the matrix product $R := MN$.
4. For $0 \leqslant j < q$, let $v_j := R_{1,j} b_1 + \cdots + R_{r,j} b_r$.
5. Return $\sum_{0 \leqslant j < q} v_j a^{pj}$.

PROPOSITION 3.1. *Algorithm 3.1 is correct. If $d = O(r)$, then it runs in time*

$$O\big(\mathsf{m}_{\mathbb{B}} \sqrt{d} + \mathsf{m}_{\mathbb{A}}\, r\, d^{\omega-1}\big).$$

Proof. By construction, we have $v_j = f_{jp} + f_{jp+1} a + \cdots + f_{jp+p-1} a^{p-1}$ for $j = 0, \ldots, q-1$, whence $f(a) = \sum_{0 \leqslant j < q} v_j a^{pj}$. This proves the correctness of the algorithm. Step 2a requires $O(p) = O(\sqrt{d})$ multiplications in $\mathbb{B}$, when computing the powers using $a^i = a \cdot a^{i-1}$. Step 2b takes $O(d)$ assignments in $\mathbb{A}$, which come for free in our complexity model. The matrix multiplication in step 3 can be done in time $O(\mathsf{m}_{\mathbb{A}}\, r\, d^{\omega-1})$. Step 5 involves $O(q) = O(\sqrt{d})$ multiplications and additions in $\mathbb{B}$, when using Horner's method. Altogether, this leads to the claimed running time. □

Remark 3.2. If $d \gg r$ and if a monic annihilator $\chi \in \mathbb{A}[x]$ with $\chi(a) = 0$ is known, then it is possible to compute $f(a)$ with $O(\mathsf{m}_{\mathbb{B}} \sqrt{r} + \mathsf{m}_{\mathbb{A}}\, r^{\omega} + \mathsf{M}_{\mathbb{A}}(d))$ operations in $\mathbb{A}$. Indeed, we have $f(a) = (f \operatorname{rem} \chi)(a)$, and $f \operatorname{rem} \chi$ is computed in time $O(\mathsf{M}_{\mathbb{A}}(d))$.

3.2. Multivariate baby-step giant-step generalization

Let us now study the evaluation of a multivariate polynomial $f \in \mathbb{A}[x_1, \ldots, x_t]$ of partial degree $< d_i$ in each x_i at a point $(a_1, \ldots, a_t) \in \mathbb{B}^t$. Setting $d = d_1 \cdots d_t$ and writing $f_{e_1,\ldots,e_t}$ for the coefficient of the monomial $x_1^{e_1} \cdots x_t^{e_t}$ in f, we have the following generalization of Algorithm 3.1 to multivariate polynomials.

Algorithm 3.2
Input: $f \in \mathbb{A}[x_1, \ldots, x_t]$ and $a_1, \ldots, a_t \in \mathbb{B}$ with $\deg_{x_i} f < d_i$ for $i = 1, \ldots, t$.
Output: $f(a_1, \ldots, a_t) \in \mathbb{B}$.

1. Let $\ell \in \{1, \ldots, t\}$ be maximal with $d_1 \cdots d_{\ell-1} < \sqrt{d}$.
2. If $d_1 \cdots d_{\ell-1} < \frac{1}{2}\sqrt{d}$ then let $p := \lfloor \sqrt{d} / (d_1 \cdots d_{\ell-1}) \rfloor$ else let $p := 1$.
   Then let $q := \lceil d_\ell / p \rceil$, $P := d_1 \cdots d_{\ell-1} p$, and $Q := q\, d_{\ell+1} \cdots d_t$.
3. For $0 \leqslant i_1 < d_1, \ldots, 0 \leqslant i_{\ell-1} < d_{\ell-1}, 0 \leqslant i_\ell < p$ do:
   a. Let $i := i_1 + d_1 i_2 + \cdots + d_1 \cdots d_{\ell-1} i_\ell$.
   b. Compute and decompose $a_1^{i_1} \cdots a_\ell^{i_\ell} = M_{1,i} b_1 + \cdots + M_{r,i} b_r$.
      (This yields an $r \times P$ matrix $M \in \mathbb{A}^{r \times P}$.)
   c. For $0 \leqslant j_\ell < q, 0 \leqslant j_{\ell+1} < d_{\ell+1}, \ldots, 0 \leqslant j_t < d_t$ do:
      i. Let $j := j_\ell + q j_{\ell+1} + \cdots + q\, d_{\ell+1} \cdots d_{t-1} j_t$.
      ii. Let $N_{i,j} := f_{i_1,\ldots,i_{\ell-1},i_\ell+pj_\ell,j_{\ell+1},\ldots,j_t}$.
      (This yields a $P \times Q$ matrix $N \in \mathbb{A}^{P \times Q}$.)
4. Compute the matrix product $R := MN$.
5. For $0 \leqslant j < Q$, let $v_j := R_{1,j} b_1 + \cdots + R_{r,j} b_r$.
6. Return $\sum_{0 \leqslant j_\ell < q} \sum_{0 \leqslant j_{\ell+1} < d_{\ell+1}} \cdots \sum_{0 \leqslant j_t < d_t} v_{j_\ell + q j_{\ell+1} + \cdots + q d_{\ell+1} \cdots d_{t-1} j_t} a_\ell^{p j_\ell} a_{\ell+1}^{j_{\ell+1}} \cdots a_r^{j_r}$.

PROPOSITION 3.3. *Algorithm 3.2 is correct. If $d = O(r)$ and $d_i \geqslant 2$ for $i = 1, \ldots, t$, then it runs in time*

$$O\big(\mathsf{m}_{\mathbb{B}} \sqrt{d} + \mathsf{m}_{\mathbb{A}}\, r\, d^{\omega-1}\big).$$

Proof. This time, for all $0 \leqslant j_\ell < q, 0 \leqslant j_{\ell+1} < d_{\ell+1}, \ldots, 0 \leqslant j_t < d_t$, we have

$$v_{j_\ell + q j_{\ell+1} + \cdots + q d_{\ell+1} \cdots d_{t-1} j_t} = \sum_{0 \leqslant i_1 < d_1} \cdots \sum_{0 \leqslant i_{\ell-1} < d_{\ell-1}} \sum_{0 \leqslant i_\ell < p} f_{i_1,\ldots,i_{\ell-1},i_\ell+pj_\ell,j_{\ell+1},\ldots,j_t} a_1^{i_1} \cdots a_\ell^{i_\ell}.$$

Plugging this expression into the formula of step 6 shows that the algorithm indeed returns the desired evaluation.

From $d_1 \cdots d_\ell \geqslant \sqrt{d}$ we always have $d_{\ell+1} \cdots d_t \leqslant \sqrt{d}$. If $d_1 \cdots d_{\ell-1} < \frac{1}{2}\sqrt{d}$, then we obtain

$$\frac{\sqrt{d}}{d_1 \cdots d_{\ell-1}} - 1 < p \leqslant \frac{\sqrt{d}}{d_1 \cdots d_{\ell-1}}$$

and then

$$\sqrt{d} - d_1 \cdots d_{\ell-1} < P \leqslant \sqrt{d}.$$

The lower bound on p also implies

$$q < \frac{d_\ell}{\sqrt{d}/(d_1 \cdots d_{\ell-1}) - 1} + 1,$$

whence

$$Q = q\, d_{\ell+1} \cdots d_t < \frac{d}{\sqrt{d} - d_1 \cdots d_{\ell-1}} + d_{\ell+1} \cdots d_t < 3\sqrt{d}.$$

Otherwise $\frac{1}{2}\sqrt{d} \leqslant d_1 \cdots d_{\ell-1}$ straightforwardly implies $P = d_1 \cdots d_{\ell-1} < \sqrt{d}$ and $Q = d_\ell \cdots d_t \leqslant 2\sqrt{d}$. In all cases P and Q are in $O(\sqrt{d})$. Consequently the complexity analysis is the same as for Algorithm 3.1. Of course, one has to carefully evaluate the power products in step 3b and the sum in step 6, in order to use only $O(\sqrt{d})$ multiplications in $\mathbb{B}$. □

Remark 3.4. The constant $1/2$ in step 2 of Algorithm 3.2 may be replaced by any other positive value strictly less than 1.

4. SEPARABLE TOWERS OF FIELDS

Let $(\mathbb{K}_i)_{i \leqslant t}$ be a tower of separable algebraic extensions of an effective base field $\mathbb{K}$. Throughout this section, we assume that the extension degrees d_i satisfy $d_i \geqslant 2$ for all i (which is usually not restrictive as explained in Remark 2.6), and that $1 < C \leqslant 3$ is as in Proposition 2.4. Our main objective is to design a fast algorithm for multiplying elements in the tower.

The basic idea is as follows. Let $\varepsilon > 0$ be an arbitrarily small positive constant. If at least half of the d_i are "sufficiently large", then t automatically becomes "small enough" to ensure that $C^t d^{1+\varepsilon} = d^{1+O(\varepsilon)}$. Otherwise, there exist $k \geqslant 2$ consecutive small degrees $d_{i+1}, \ldots, d_{i+k}$, and we replace the corresponding subtower of extensions $\mathbb{K}_i \subseteq \mathbb{K}_{i+1} \subseteq \cdots \subseteq \mathbb{K}_{i+k}$ by a primitive one $\mathbb{K}_i \subseteq \mathbb{K}_{i+k}$. We keep repeating these replacements until the height of the tower becomes "small enough". Some careful balancing is required here, since the computation of a primitive element for $\mathbb{K}_{i+k}$ over $\mathbb{K}_i$ can become expensive if $d_{i+1} \cdots d_{i+k}$ gets "too large". The precise tradeoff will be made explicit at the end of the section.

In this section, given two effective rings $\mathbb{A}$ and $\mathbb{B}$ along with a natural way to convert elements in $\mathbb{A}$ to elements in $\mathbb{B}$, we denote by $\mathsf{C}(\mathbb{A} \to \mathbb{B})$ the cost of such conversions. If we also have an algorithm for conversions in the opposite direction, then we write $\mathsf{C}(\mathbb{A} \leftrightarrow \mathbb{B}) = \max(\mathsf{C}(\mathbb{A} \to \mathbb{B}), \mathsf{C}(\mathbb{B} \to \mathbb{A}))$.

4.1. Primitive tower representations

Assuming that $\mathbb{K}$ contains sufficiently many elements, the aim of this subsection is to construct a primitive element representation in the sense of Definition 2.2. We thus have to compute primitive elements $\beta_1 \in \mathbb{K}_1, \ldots, \beta_t \in \mathbb{K}_t$ over $\mathbb{K}$ such that

$$\begin{aligned}
\mathbb{K}_0 &= \mathbb{K} \\
\mathbb{K}_1 &= \mathbb{K}_0[\alpha_1] \cong \mathbb{K}[\beta_1] \\
\mathbb{K}_2 &= \mathbb{K}_1[\alpha_2] \cong \mathbb{K}[\beta_2] \\
&\vdots \\
\mathbb{K}_t &= \mathbb{K}_{t-1}[\alpha_{t-1}] \cong \mathbb{K}[\beta_t],
\end{aligned}$$

together with the minimal polynomials $\nu_i \in \mathbb{K}[x]$ of β_i over $\mathbb{K}$ for $i = 1, \ldots, t$. We also show how to convert efficiently between each of the two representations of $\mathbb{K}_i$.

PROPOSITION 4.1. *Assume that we are given a primitive tower representation of $(\mathbb{K}_i)_{i \leqslant t}$ such that $\beta_1 = \alpha_1$ and $\beta_i = \alpha_i + \lambda_i \beta_{i-1}$ for some $\lambda_i \in \mathbb{K}$, for $i = 2, \ldots, t$. Then we have*

$$\mathsf{C}(\mathbb{K}_t \leftrightarrow \mathbb{K}[\beta_t]) = O(\mathsf{m}_{\mathbb{K}}\, d^{\omega}).$$

Proof. The bound on the cost of conversions from $\mathbb{K}_t$ to $\mathbb{K}[\beta_t]$ is a direct consequence of Proposition 3.3 by taking the assumption $\omega > 3/2$ into account. For $i = 1, \ldots, t$, the minimal polynomial of α_i over $\mathbb{K}_{i-1}$ may thus be converted into a minimal polynomial ρ_i over $\mathbb{K}[\beta_{i-1}]$ in time $O(\mathsf{m}_{\mathbb{K}}\, (d_1 \cdots d_{i-1})^{\omega} d_i)$. The computation of $\rho_1, \ldots, \rho_t$ requires $O(\mathsf{m}_{\mathbb{K}}\, d^{\omega})$ extra operations since $d_i \geqslant 2$ for all i.

For conversions from $\mathbb{K}[\beta_t]$ to $\mathbb{K}_t$, we consider a polynomial $f_t \in \mathbb{K}[x]_{<d}$. We evaluate f_t at $\alpha_t + \lambda_t \beta_{t-1}$ in $\mathbb{K}[\beta_{t-1}][\alpha_t]$ seen as $\mathbb{K}[\beta_{t-1}][x] / (\rho_t(x))$ with cost $O(\mathsf{m}_{\mathbb{K}}\, d^{\omega})$ by Proposition 3.1. We thus obtain a polynomial $f_{t-1} \in \mathbb{K}[\beta_{t-1}][x]_{<d_t}$ such that $f_t(\beta_t) = f_{t-1}(\alpha_t)$.

We next need to convert the d_t coefficients of f_{t-1} from $\mathbb{K}[\beta_{t-1}]$ to $\mathbb{K}_{t-1}$. Using a straightforward induction this leads to the following cost:

$$\begin{aligned} \mathsf{C}(\mathbb{K}[\beta_t] \to \mathbb{K}_t) &= O(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + d_t\, \mathsf{m}_{\mathbb{K}}\, (d/d_t)^{\omega} + \cdots + d_2 \cdots d_t\, \mathsf{m}_{\mathbb{K}}\, (d/(d_2 \cdots d_t))^{\omega}) \\ &= O(\mathsf{m}_{\mathbb{K}}\, d^{\omega}\, (1 + d_t^{1-\omega} + \cdots + (d_2 \cdots d_t)^{1-\omega})) \;=\; O(\mathsf{m}_{\mathbb{K}}\, d^{\omega}). \end{aligned}$$

Here we again use of the assumptions that $\omega > 3/2$ and $d_i \geqslant 2$ for all i. □

Remark 4.2. In [39, Corollary 7.3], we have introduced an alternative approach to conversions that is more efficient when $\bar{d} := \max(d_1, \ldots, d_t)$ is "sufficiently small". More precisely, modulo suitable precomputations, we have shown that conversions between elements of $\mathbb{K}[\beta_t]$ and $\mathbb{K}[\alpha_1, \ldots, \alpha_t]$ can be done in time $O(2^t\, \bar{d}\, \mathsf{M}_{\mathbb{K}}(d))$. In various special cases, the factor 2^t can be further reduced.

The next proposition provides us with complexity bounds for computing primitive tower representations by induction. We denote the cardinality of $\mathbb{K}$ by $\operatorname{card} \mathbb{K}$.

PROPOSITION 4.3. *Assume that $\operatorname{card} \mathbb{K} > \binom{d}{2}$ and that a primitive tower representation of $(\mathbb{K}_i)_{i \leqslant t-1}$ has already been computed. Then*

a) We can compute β_t of the form $\alpha_t + \lambda_t \beta_{t-1}$ with $\lambda_t \in \mathbb{K}$, and ν_t in time

$$O(\mathsf{d}_{\mathbb{K}}\, d^2 + d\, \mathsf{M}_{\mathbb{K}}(d^2) \log d).$$

b) Given β_t and ν_t, we can compute $\phi_{t,i} \in \mathbb{K}[x]$ with $\alpha_i = \phi_{t,i}(\beta_t)$ for $i = 1, \ldots, t$ in time

$$O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2) \log d).$$

In addition, if $\operatorname{card} \mathbb{K} \geqslant 2\binom{d}{2}$ then the computations in part a can be achieved with expected cost $O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2) \log d)$ by means of a randomized Las Vegas algorithm.

Proof. Thanks to Proposition 4.1, modulo conversions of cost $O(\mathsf{m}_{\mathbb{K}}\, d_t\, (d/d_t)^{\omega}) = O(\mathsf{m}_{\mathbb{K}}\, d^{\omega}\, d_t^{1-\omega})$ between $\mathbb{K}[\beta_{t-1}]$ and $\mathbb{K}_{t-1}$, we rewrite $\mu_t(y)$ into $\rho_t(\beta_{t-1}, y)$ such that $\rho_t \in \mathbb{K}[x, y]$, $\deg_x \rho_t < d/d_t$, $\deg_y \rho_t = d_t$. We thus have

$$\mathbb{K}[\beta_{t-1}, \alpha_t] \cong \mathbb{K}[x, y] / (\nu_{t-1}(x), \rho_t(x, y)).$$

Now let $\lambda_t \in \mathbb{K}$ be a parameter that will be specified later, and consider $\beta_t = \alpha_t + \lambda_t \beta_{t-1}$. The characteristic polynomial of β_t over $\mathbb{K}[\beta_{t-1}]$ equals to $\rho_t(\beta_{t-1}, y - \lambda_t \beta_{t-1})$. Consequently the characteristic polynomial ν_t of β_t over $\mathbb{K}$ is $\mathbb{K}$-proportional to

$$R(\lambda_t, y) := \mathrm{Res}_x(\nu_{t-1}(x), \rho_t(x, y - \lambda_t x)).$$

The polynomial $\rho_t(x, y - \lambda_t x)$ rem $\nu_{t-1}(x)$ can be obtained as the preimage of $\rho_t(\beta_{t-1}, y - \lambda_t \beta_{t-1})$ whose computation costs $O(\mathsf{M}_{\mathbb{K}}(d) \log d_t)$ by a standard "divide and conquer" approach; see [5, Lemma 7], for instance. Then the resultant may be computed in time $O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2/d_t) \log d)$ by [45, Corollary 26].

Regarding λ_t as an indeterminate, the polynomial $R(\lambda_t, y)$ has degree $\leqslant d$. Therefore R may be interpolated from $d+1$ values of λ_t. The total cost to obtain R is thus

$$O(\mathsf{d}_{\mathbb{K}}\, d^2 + d\, \mathsf{M}_{\mathbb{K}}(d^2/d_t) \log d + d\, \mathsf{M}_{\mathbb{K}}(d) \log d) = O(\mathsf{d}_{\mathbb{K}}\, d^2 + d\, \mathsf{M}_{\mathbb{K}}(d^2/d_t) \log d).$$

Geometrically speaking, the system $\nu_{t-1}(x) = \rho_t(x, y) = 0$ admits the d pairwise distinct solutions $(x_1, y_1), \ldots, (x_d, y_d)$ in $(\mathbb{K}^{\mathrm{alg}})^2$, where $\mathbb{K}^{\mathrm{alg}}$ denotes the algebraic closure of $\mathbb{K}$. The polynomial $\nu_t(y)$ is separable if, and only if, its roots $y_1 + \lambda_t x_1, \ldots, y_d + \lambda_t x_d$ are pairwise distinct. There are at most $\binom{d}{2}$ values of λ_t for which this is not the case, so we simply need to try $\binom{d}{2} + 1$ distinct values of λ_t until ν_t becomes separable. Testing the separability of ν_t for d values $\lambda_{t,1}, \ldots, \lambda_{t,d}$ of λ_t is achieved as follows.

We first evaluate $R(\lambda_{t,i}, y)$ for $i = 1, \ldots, d$. Using fast multi-point evaluation, this takes time $O(d\, \mathsf{M}_{\mathbb{K}}(d) \log d)$. We next test whether the discriminant of $R(\lambda_{t,i}, y)$ in y vanishes, for $i = 1, \ldots, d$; this can be done in time $O(d\, \mathsf{M}_{\mathbb{K}}(d) \log d)$. Doing this for $O(d)$ packets of d values, the overall computation can be done in time $O(d^2\, \mathsf{M}_{\mathbb{K}}(d) \log d)$. This completes the proof of part *a*.

As to part *b*, for any root ζ of ν_t the gcd of $\nu_{t-1}(x)$ and $\rho_t(x, \zeta - \lambda_t x)$ has degree 1, so the specialization property of subresultants ensures that the subresultant in x of degree 1 of $\nu_{t-1}(x)$ and $\rho_t(x, y - \lambda_t x)$ has the form $A(y)\, x + B(y)$ with $A(y)$ coprime to $\nu_t(y)$. This subresultant can be computed in time $O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2/d_t) \log d)$ again by [45, Corollary 26]. Since A and B have degrees $< d$, we obtain the polynomial $\psi_t(y) = -A(y)^{-1} B(y)$ modulo $\nu_t(y)$ in $\mathbb{K}[y]_{<d}$ with additional cost $O(\mathsf{d}_{\mathbb{K}} + \mathsf{M}_{\mathbb{K}}(d) \log d)$. Then from $A(\beta_t)\, \beta_{t-1} + B(\beta_t) = 0$ we deduce that $\beta_{t-1} = \psi_t(\beta_t)$. For $i < t$, we then take $\phi_{t,i} = (\phi_{t-1,i} \circ \psi_t)$ rem ν_t. Proposition 3.1 implies that the cost of these modular compositions is bounded by $\mathsf{m}_{\mathbb{K}}\, \tilde{O}(d^{3/2}) + O(t\, \mathsf{m}_{\mathbb{K}}\, d^{\omega}) = O(\mathsf{m}_{\mathbb{K}}\, d^{\omega} \log d)$. This completes the proof of part *b*.

If $\mathrm{card}\, \mathbb{K} \geqslant 2\binom{d}{2}$, then we use a Las Vegas algorithm for part *a*: we pick λ_t at random in a set of size $2\binom{d}{2}$ until ν_t is separable. Each pick is successful with probability at least $1/2$, so the expected cost is $O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2) \log d)$. □

COROLLARY 4.4. *Assume that* $\mathrm{card}\, \mathbb{K} > \binom{d}{2}$. *Then a primitive tower representation of* $(\mathbb{K}_i)_{i \leqslant t}$ *can be computed in time* $O(\mathsf{d}_{\mathbb{K}}\, d^2 + d\, \mathsf{M}_{\mathbb{K}}(d^2) \log d)$. *Using a randomized Las Vegas algorithm, the computation can even be done with expected cost* $O(\mathsf{d}_{\mathbb{K}}\, d + \mathsf{M}_{\mathbb{K}}(d^2) \log d)$.

Proof. This directly follows from the latter proposition, using that $d_i \geqslant 2$ for all i. □

Remark 4.5. In a similar way as for the computation of gcds in section 2.5, it is possible to avoid divisions during the computation of resultants and subresultants. However, the required adaptations of the algorithms from [45] are a bit more technical than the mere replacement of Euclidean divisions by pseudo-divisions. For this reason, we have not further optimized the number of divisions in our complexity bounds.

4.2. Accelerated tower arithmetic

The main problem with the complexity bounds of Proposition 2.4 is that the height t of the tower may get as large as $\lfloor \log_2 d \rfloor$. Now if t indeed gets large, then many of the d_i are necessarily small. Furthermore, as long as a product of the form $d_{i+1} \cdots d_j$ is reasonably small, the results from the previous subsection allow us to change the given representation of $\mathbb{K}_j = \mathbb{K}_i[\alpha_{i+1}, \ldots, \alpha_j]$ into a more efficient primitive element representation $\mathbb{K}_j \cong \mathbb{K}_i[\beta_j]$. Repeating this operation as many times as necessary, we reduce the height of the tower and guarantee that all defining polynomials have sufficiently large degrees. This process is detailed in the next paragraphs.

DEFINITION 4.6. *Let $\delta \leqslant d$ be a positive integer. We define a **δ-accelerated tower representation** of $(\mathbb{K}_i)_{i \leqslant t}$ to consist of the following data:*

- *A sequence of integers $0 = i_0 < i_1 < \cdots < i_s = t$.*
- *A tower $(\mathbb{L}_j)_{j \leqslant s}$ such that $\mathbb{L}_j \cong \mathbb{K}_{i_j}$ for $j = 0, \ldots, s$, represented by a sequence of defining polynomials $\rho_1, \ldots, \rho_s$, where $\mathbb{L}_j := \mathbb{L}_{j-1}[\beta_j] = \mathbb{L}_{j-1}[x] / (\rho_i(x))$ and $\rho_j \in \mathbb{L}_{j-1}[x]$.*
- *For $j = 1, \ldots, s$, a primitive tower representation of $\mathbb{K}_{i_{j-1}+1}, \ldots, \mathbb{K}_{i_j}$ seen as a tower over $\mathbb{L}_{j-1}$ and ending with $\mathbb{L}_{j-1}[\beta_j]$.*
- *Denote $e_j := d_{i_{j-1}+1} \cdots d_{i_j}$. If $j < s$ and $i_j = i_{j-1} + 1$, then $e_j e_{j+1} \geqslant \delta$. If $i_j > i_{j-1} + 1$, then $e_j < \delta$.*

LEMMA 4.7. *Let the notations be as in Definition 4.6 and assume that* $\operatorname{card} \mathbb{K} > \binom{d}{2}$. *There exists a δ-accelerated tower representation of height $s \leqslant 3 \frac{\log d}{\log \delta} + 1$.*

Proof. The construction of the sequence $0 = i_0 < i_1 < \cdots < i_s = t$ is done by induction. For $j \geqslant 1$ with $i_{j-1} < t$, assume that we have completed the construction up to height $j-1$. We distinguish the following cases:

- If $d_{i_{j-1}+1} \geqslant \delta$ then we set $i_j := i_{j-1} + 1$.
- Otherwise, we take $k \leqslant t$ maximal such that $d_{i_{j-1}+1} \cdots d_k < \delta$. So either $k = t$ or $d_{i_{j-1}+1} \cdots d_{k+1} \geqslant \delta$.

Whenever $e_j < \delta$ we must have either $j = s$ or $e_j e_{j+1} \geqslant \delta$. Consequently the number m of such indices j is constrained by $\delta^{m-1} \leqslant d^2$. It follows that $m \leqslant 2 \frac{\log d}{\log \delta} + 1$. On the other hand the number of indices j with $e_j \geqslant \delta$ is necessarily $\leqslant \frac{\log d}{\log \delta}$. It follows that $s \leqslant 3 \frac{\log d}{\log \delta} + 1$. Once the indices $i_1, \ldots, i_s$ are known, the existence of a δ-accelerated tower representation follows from Corollary 4.4. □

Algorithm 4.1
Input. A separable tower of fields $(\mathbb{K}_i)_{i \leqslant t}$ over $\mathbb{K}$ and $\delta \in \mathbb{N}$ with $0 < \delta < d$.
Output. A δ-accelerated tower representation of $(\mathbb{K}_i)_{i \leqslant t}$.

1. Determine the integer sequence $0 = i_0 < i_1 < \cdots < i_s = t$ as described in the proof of Lemma 4.7. Set $i_0 := 0$ and $\mathbb{L}_0 := \mathbb{K}$.
2. For $j = 1, \ldots, s$ do:
   a. Compute a primitive tower representation of $\mathbb{K}_{i_{j-1}+1}, \ldots, \mathbb{K}_{i_j}$ over $\mathbb{L}_{j-1}$.

b. Let β_j and ρ_j respectively represent the top level primitive element found for $\mathbb{K}_{i_j}$ over $\mathbb{L}_{j-1}$ and its minimal polynomial, and set $\mathbb{L}_j := \mathbb{L}_{j-1}[\beta_j] = \mathbb{L}_{j-1}[x]/(\rho_j(x))$.

3. Return $(i_j)_{j\leqslant s}$, $(\mathbb{L}_j)_{j\leqslant s}$, together with the primitive tower representations from step 2a.

In order to bound the execution time of Algorithm 4.1, we need to carefully analyze the cost of conversions between $\mathbb{L}_j$ and $\mathbb{K}_{i_j}$ for $j=1,\ldots,s$.

LEMMA 4.8. *With the notations of Algorithm 4.1, there exists a universal constant K such that for $j=0,\ldots,s$, we have*

$$\mathsf{C}(\mathbb{L}_j \leftrightarrow \mathbb{K}_{i_j}) \leqslant K C^j \mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,\delta^{\omega-1}.$$

Proof. By Proposition 2.4, there exists a universal constant A with

$$\mathsf{m}_{\mathbb{L}_j} \leqslant A C^j \mathsf{M}_{\mathbb{K}}(e_1\cdots e_j).$$

By Proposition 4.1, there also exists a universal constant B such that conversions between $\mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}]$ and $\mathbb{L}_j$ can be performed in time

$$\begin{aligned}\mathsf{C}(\mathbb{L}_j \leftrightarrow \mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}]) &\leqslant B\,\mathsf{m}_{\mathbb{L}_{j-1}} e_j^{\omega}\\ &\leqslant A B C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_{j-1})\,e_j^{\omega}\\ &\leqslant A B C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,e_j^{\omega-1},\end{aligned}$$

for $j=1,\ldots,s$, and where we used the assumption that $\mathsf{M}_{\mathbb{K}}(n)/n$ is non-decreasing in n. If $i_j = i_{j-1}+1$ then such conversions are actually for free. Otherwise we have $e_j < \delta$, whence

$$\begin{aligned}\mathsf{C}(\mathbb{L}_j \leftrightarrow \mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}]) &\leqslant A B C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,e_j^{\omega-1}\\ &\leqslant A B C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,\delta^{\omega-1}.\end{aligned}$$

Now take $K \geqslant A B/(C-1)$ and let us prove the lemma by induction over j. For $j=0$ we have nothing to do, so we assume that $j>0$. Using the induction hypothesis, conversions between $\mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}]$ and $\mathbb{K}_{i_j}$ are performed in time

$$\begin{aligned}\mathsf{C}(\mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}] \leftrightarrow \mathbb{K}_{i_j}) &\leqslant \mathsf{C}(\mathbb{L}_{j-1}\leftrightarrow\mathbb{K}_{i_{j-1}})\,e_j\\ &\leqslant K C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_{j-1})\,e_j\,\delta^{\omega-1}\\ &\leqslant K C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,\delta^{\omega-1}.\end{aligned}$$

Consequently,

$$\begin{aligned}\mathsf{C}(\mathbb{L}_j \leftrightarrow \mathbb{K}_{i_j}) &\leqslant \mathsf{C}(\mathbb{L}_j \leftrightarrow \mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}]) + \mathsf{C}(\mathbb{L}_{j-1}[\alpha_{i_{j-1}+1},\ldots,\alpha_{i_j}] \leftrightarrow \mathbb{K}_{i_j})\\ &\leqslant (A B + K)\,C^{j-1}\mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,\delta^{\omega-1}\\ &\leqslant K C^j \mathsf{M}_{\mathbb{K}}(e_1\cdots e_j)\,\delta^{\omega-1}.\end{aligned}$$

The lemma follows by induction. □

PROPOSITION 4.9. *Assume that* $\operatorname{card}\mathbb{K} > \binom{d}{2}$. *Then Algorithm 4.1 is correct and runs in time*

$$O(\mathsf{d}_{\mathbb{K}}\,\delta^2 \log d + C^s \mathsf{M}_{\mathbb{K}}(d)\,\delta^2 \log d).$$

In addition, if $\operatorname{card}\mathbb{K} \geqslant 2\binom{d}{2}$, *then a randomized Las Vegas version of Algorithm 4.1 has expected cost* $O(\mathsf{d}_{\mathbb{K}}\,\delta \log d + C^s \mathsf{M}_{\mathbb{K}}(d)\,\delta \log d)$.

Proof. The correctness of Algorithm 4.1 is ensured by Lemma 4.7. Let us analyze the cost of step 2a for a given $j \in \{1, \ldots, s\}$. The necessary conversions for rewriting the defining polynomials of $\alpha_{i_{j-1}+1}, \ldots, \alpha_{i_j}$ over $\mathbb{L}_{j-1}$ can be done in time

$$\begin{aligned} O(\mathsf{C}(\mathbb{L}_{j-1} \leftrightarrow \mathbb{K}_{i_{j-1}})\, e_j) &= O(C^{j-1}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_{j-1})\, e_j\, \delta^{\omega-1}) \\ &= O(C^{j}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_{j})\, \delta^{\omega-1}), \end{aligned}$$

by Lemma 4.8. If $i_j = i_{j-1} + 1$, then there is nothing to be done since $\beta_j = \alpha_{i_{j-1}+1}$. So assume that $i_j > i_{j-1} + 1$, whence $e_j < \delta$. Then Proposition 2.7 gives

$$\mathsf{d}_{\mathbb{L}_{j-1}} = \mathsf{d}_{\mathbb{K}} + O(C^{j-1}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_{j-1}) \log \max(\bar{d}, \delta)).$$

Consequently, in view of Corollary 4.4 and using assumption (2.2), the remainder of step 2a takes time

$$\begin{aligned} & O(\mathsf{d}_{\mathbb{L}_{j-1}} e_j^2 + e_j\, \mathsf{M}_{\mathbb{L}_{j-1}}(e_j^2) \log e_j) \\ = \; & O((\mathsf{d}_{\mathbb{K}} + C^{j-1}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_{j-1}) \log \max(\bar{d}, \delta))\, e_j^2 + C^{j-1}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_j)\, e_j^2 \log e_j) \\ = \; & O(\mathsf{d}_{\mathbb{K}}\, \delta^2 + C^{j-1}\, \mathsf{M}_{\mathbb{K}}(e_1 \cdots e_j)\, \delta^2 \log \max(\bar{d}, \delta)), \end{aligned}$$

which dominates the cost of conversions since $\omega < 2$. The total cost for all $j = 1, \ldots, s$ is thus $O(\mathsf{d}_{\mathbb{K}}\, \delta^2 \log d + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^2 \log d)$.

Finally, if $\operatorname{card} \mathbb{K} > 2\binom{d}{2}$, then the randomized variant from Corollary 4.4 leads to the claimed expected cost. □

THEOREM 4.10. *If $1 \leqslant \delta \leqslant d$ and $\operatorname{card} \mathbb{K} > \binom{d}{2}$, then*

$$\begin{aligned} \mathsf{m}_{\mathbb{K}_t} &= O(\mathsf{d}_{\mathbb{K}}\, \delta^2 \log d + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^2 \log d) \\ \mathsf{M}_{\mathbb{K}_t}(n) &= O(\mathsf{d}_{\mathbb{K}}\, \delta^2 \log d + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^2 \log d + C^s\, (n\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1} + \mathsf{M}_{\mathbb{K}}(n d))). \end{aligned}$$

In addition, if a δ-accelerated tower representation of $(\mathbb{K}_i)_{i \leqslant t}$ is known, then we have

$$\begin{aligned} \mathsf{m}_{\mathbb{K}_t} &= O(C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1}) \\ \mathsf{M}_{\mathbb{K}_t}(n) &= O(C^s\, (n\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1} + \mathsf{M}_{\mathbb{K}}(n d))). \end{aligned}$$

Proof. The cost to obtain a δ-accelerated tower representation of $(\mathbb{K}_i)_{i \leqslant t}$ is given in Proposition 4.9, namely $O(\mathsf{d}_{\mathbb{K}}\, \delta^2 \log d + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^2 \log d)$.

Then, in order to multiply two elements in $(\mathbb{K}_i)_{i \leqslant t}$, we convert them into the accelerated representation, and multiply them with cost

$$\begin{aligned} \mathsf{m}_{\mathbb{L}_s} &= O(C^s\, \mathsf{M}_{\mathbb{K}}(d)) \\ \mathsf{M}_{\mathbb{L}_s}(n) &= O(C^s\, \mathsf{M}_{\mathbb{K}}(n d)), \text{ for all } n \geqslant 0 \end{aligned}$$

by Proposition 2.4. We finally convert the product back into $\mathbb{K}_t$. By Lemma 4.8, each of these conversions can be done in time $O(C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1})$. □

COROLLARY 4.11. *If $\operatorname{card} \mathbb{K} > \binom{d}{2}$, then we have*

$$\mathsf{m}_{\mathbb{K}_t} = O(\mathsf{d}_{\mathbb{K}}\, d^{2\epsilon(d)} \log d + \mathsf{M}_{\mathbb{K}}(d)\, d^{4\epsilon(d)} \log d),$$

where $\epsilon(d) = \sqrt{\frac{3 \log C}{2 \log d}}$. In particular, $\mathsf{m}_{\mathbb{K}_t} = O((\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}}\, d)\, d^{o(1)})$.

Proof. It is important to notice that constants hidden in the "O" of Theorem 4.10 are independent of the value for δ, so we may freely set δ in terms of d. Now taking $\delta = d^{\epsilon(d)}$ balances the contributions of C^s and δ^2. Indeed, since $s \leqslant 3\frac{\log d}{\log \delta} + 1$ by Lemma 4.7, we have $C^{s-1} \leqslant C^{3\epsilon(d)^{-1}} = d^{2\epsilon(d)}$. We conclude by the latter theorem. □

Remark 4.12. For the same reasons as in Remark 4.5, we have not attempted to further optimize the number of divisions in our complexity bounds. Again, we expect that most divisions in $\mathbb{K}$ can actually be avoided.

5. SEPARABLE TOWERS AND FACTORIZATION

Up to now we have focused on towers of field extensions. In this section, we turn our attention to more general towers of separable integral ring extensions. Each extension ring is still assumed to be a product of separable field extensions over a common ground field $\mathbb{K}$, which will allow us to apply the theory from the previous sections.

We start with a description of the precise types of towers of rings that we will be able to handle. We next explain how such towers of rings can be "factored" into towers of fields and we analyze the cost of the corresponding conversions. Finally, we give a complexity bound for the computation of "tower factorizations". We do not claim that our approach is efficient in all cases. Nevertheless it is expected to be so whenever the factorization process behaves as a pretreatment with negligible cost (or when factoring polynomials over $\mathbb{K}$ is reasonably affordable; see section 5.5). See the conclusion of this paper for a discussion of alternative approaches.

5.1. Separable towers of rings

Throughout this section, we still assume that the ground ring $\mathbb{A}$ is a field, written $\mathbb{K}$, and we consider a tower $(\mathbb{A}_i)_{i\leqslant t}$ of integral ring extensions of $\mathbb{K}$. For $i = 1, \ldots, t$, we still let μ_i denote the defining polynomial of $\mathbb{A}_i$ over $\mathbb{A}_{i-1}$, define d_i, α_i, etc. as before, and assume that $d_i \geqslant 2$. We say that $(\mathbb{A}_i)_{i\leqslant t}$ is a tower of separable integral ring extensions, or just *separable* in short, if

- $\mathbb{A} = \mathbb{A}_0$ is the field $\mathbb{K}$.
- There exist polynomials u_i and v_i in $\mathbb{A}_{i-1}[x]$ such that $u_i\mu_i + v_i\mu_i' = 1$, for all $i = 1, \ldots, t$, where μ_i' denotes the derivative of μ_i.

By induction over i, one may check that each $\mathbb{A}_i$ is isomorphic to a direct product $\mathbb{K}_{i,1} \times \cdots \times \mathbb{K}_{i,s_i}$ of separable algebraic extensions of $\mathbb{K}$. The projection $\mu_{i,j}$ of μ_i into $\mathbb{K}_{i-1,j}[x]$ is separable in the usual sense, which means that $\mu_{i,j}$ and $\mu_{i,j}'$ are coprime for all $i = 1, \ldots, t$ and $j = 1, \ldots, s_{i-1}$.

In terms of the triangular set $(T_i)_{i\leqslant t}$ defined in the introduction, a separable tower corresponds to the situation where the ideal $(T_1, \ldots, T_t)$ is radical over the algebraic closure $\mathbb{K}^{\mathrm{alg}}$ of $\mathbb{K}$. In particular, this means that $(T_i)_{i\leqslant t}$ and therefore $(\mathbb{A}_i)_{i\leqslant t}$ are uniquely determined by the variety

$$V_{(\mathbb{A}_i)_{i\leqslant t}} := \mathcal{V}(T_1, \ldots, T_t) = \{(\zeta_1, \ldots, \zeta_t) \in (\mathbb{K}^{\mathrm{alg}})^t : T_1(\zeta_1) = \cdots = T_t(\zeta_1, \ldots, \zeta_t) = 0\}.$$

For each $k \leqslant t$, we notice that $V_{(\mathbb{A}_i)_{i\leqslant k}}$ is the projection of $V_{(\mathbb{A}_i)_{i\leqslant t}}$ onto the first k coordinates.

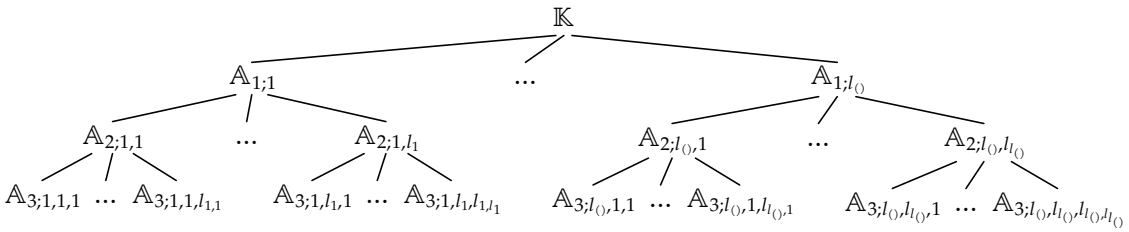

Figure 5.1. Example of a representation of a tree factorization of a tower of height $t=3$.

From a geometric point of view, one may regard computations with elements in $\mathbb{A}_t$ as computations with all zeros $(\alpha_1,\ldots,\alpha_t)$ in $V_{(\mathbb{A}_i)_{i\leqslant t}}$ in parallel. Alternatively, we may regard them as computations with parameters $\alpha_1,\ldots,\alpha_t$ subject to the polynomial constraints $T_1(\alpha_1)=\cdots=T_t(\alpha_1,\ldots,\alpha_t)=0$.

5.2. Tree factorizations of towers

Consider two separable towers $(\mathbb{A}_i)_{i\leqslant t}$ and $(\mathbb{B}_i)_{i\leqslant t}$ of the same height t and over the same base field $\mathbb{K}$. Let $(\mu_i)_{i\leqslant t}$ and $(\nu_i)_{i\leqslant t}$ denote the respective defining polynomials for $(\mathbb{A}_i)_{i\leqslant t}$ and $(\mathbb{B}_i)_{i\leqslant t}$. We say that $(\mathbb{B}_i)_{i\leqslant t}$ is a *factor* of $(\mathbb{A}_i)_{i\leqslant t}$ if $V_{(\mathbb{B}_i)_{i\leqslant t}}\subseteq V_{(\mathbb{A}_i)_{i\leqslant t}}$. This is the case if and only if there exist natural projections $\pi_i\colon \mathbb{A}_i\to\mathbb{B}_i$ (that naturally extend to projections $\pi_i\colon \mathbb{A}_i[x]\to\mathbb{B}_i[x]$ in a coefficientwise manner) such that:

- ν_i divides $\pi_{i-1}(\mu_i)$ for $i=1,\ldots,t$.
- π_i sends an element $a\in\mathbb{A}_i$ represented by $f=\sum_{i=0}^{d_{i-1}} f_i x^i\in\mathbb{A}_{i-1}[x]_{<d_i}$ to $\pi_{i-1}(f)=\sum_{i=0}^{d_{i-1}}\pi_{i-1}(f_i)\,x^i \bmod \nu_i$, for $i=1,\ldots,t$.

We say that $(\mathbb{B}_i)_{i\leqslant t}$ is *irreducible* if the $\mathbb{B}_i$ are all fields or, equivalently, if the ν_i are all irreducible.

The notation "()" stands for the empty tuple. Let Σ_k be index sets of k-tuples of integers with $\Sigma_0=\{()\}$ and

$$\Sigma_k=\{(\sigma_1,\ldots,\sigma_{k-1},i):(\sigma_1,\ldots,\sigma_{k-1})\in\Sigma_{k-1},i\in\{1,\ldots,l_{\sigma_1,\ldots,\sigma_{k-1}}\}\},$$

for suitable integers $l_{\sigma_1,\ldots,\sigma_{k-1}}$ and $k=1,\ldots,t$. Consider a family $((\mathbb{A}_{i;\sigma})_{i\leqslant t})_{\sigma\in\Sigma_t}$ of factor towers of $(\mathbb{A}_i)_{i\leqslant t}$ with the property that $\mathbb{A}_{k;\sigma_1,\ldots,\sigma_t}$ only depends on $\sigma_1,\ldots,\sigma_k$ for all $(\sigma_1,\ldots,\sigma_t)\in\Sigma_t$, and write $\mathbb{A}_{k;\sigma_1,\ldots,\sigma_k}:=\mathbb{A}_{k;\sigma_1,\ldots,\sigma_t}$. We say that such a family of factors forms a *tree factorization* of $(\mathbb{A}_i)_{i\leqslant t}$ if the variety $V_{(\mathbb{A}_i)_{i\leqslant t}}$ is partitioned into $V_{(\mathbb{A}_i)_{i\leqslant t}}=\coprod_{\sigma\in\Sigma_t} V_{(\mathbb{A}_{i;\sigma})_{i\leqslant t}}$. We say that the factorization is *irreducible*, if all its factor towers are irreducible. If $k>0$ and $\sigma\in\Sigma_k$, then we also write $\mu_{k;\sigma}\in\mathbb{A}_{k;\sigma_1,\ldots,\sigma_{k-1}}[x]$ for the defining polynomial of $\mathbb{A}_{k;\sigma}$ over $\mathbb{A}_{k-1;\sigma}$.

It is convenient to represent such a factorization by a labeled tree (see Figure 5.1): the nodes are identified with the index set $\Sigma_0\amalg\Sigma_1\amalg\cdots\amalg\Sigma_t$, and each node $\sigma\in\Sigma_k$ is labeled with the algebraic extension $\mathbb{A}_{k;\sigma}$. The parent of the node $\sigma\in\Sigma_k$ with $k>0$ is simply the node $(\sigma_1,\ldots,\sigma_{k-1})\in\Sigma_{k-1}$. Each individual factor $(\mathbb{A}_{i;\sigma})_{i\leqslant t}$ corresponds to a path from the root to a leaf.

Given $k\leqslant t$ and $\sigma\in\Sigma_k$, let

$$\Sigma_{t;\sigma}:=\{\tau:(\sigma,\tau)\in\Sigma_t\}.$$

Projecting the equality

$$V_{(\mathbb{A}_i)_{i\leqslant t}}=\coprod_{\sigma\in\Sigma_k}\coprod_{\tau\in\Sigma_{t;\sigma}} V_{(\mathbb{A}_{i;\sigma,\tau})_{i\leqslant t}}$$

on the first k coordinates, we observe that the projection of $V_{(\mathbb{A}_{i;\sigma,\tau})_{i\leqslant t}}$, for given $\sigma\in\Sigma_k$, is the same for all $\tau\in\Sigma_{t;\sigma}$, and equals $V_{(\mathbb{A}_{i;\sigma_1,\ldots,\sigma_i})_{i\leqslant k}}$, whence $V_{(\mathbb{A}_i)_{i\leqslant k}} = \coprod_{\sigma\in\Sigma_k} V_{(\mathbb{A}_{i;\sigma_1,\ldots,\sigma_i})_{i\leqslant k}}$. Consequently, $((\mathbb{A}_{i;\sigma_1,\ldots,\sigma_i})_{i\leqslant k})_{\sigma\in\Sigma_k}$ forms a tree factorization of the subtower $(\mathbb{A}_i)_{i\leqslant k}$. From an algebraic point of view, this means that we have a natural isomorphism

$$\mathbb{A}_k \;\cong\; \bigoplus_{\sigma\in\Sigma_k} \mathbb{A}_{k;\sigma}. \tag{5.1}$$

For any $\sigma\in\Sigma_k$, we denote by $\pi_{k;\sigma}$ the natural projection of $\mathbb{A}_k$ onto $\mathbb{A}_{k;\sigma}$. Dually, the family $((\mathbb{A}_{i;\sigma,\tau})_{i\leqslant t})_{\tau\in\Sigma_{t;\sigma}}$ forms a tree factorization of the tower $\mathbb{A}_0\subseteq\mathbb{A}_{1;\sigma_1}\subseteq\cdots\subseteq\mathbb{A}_{k;\sigma_1,\ldots,\sigma_k}\subseteq\mathbb{B}_{k+1;\sigma}\subseteq\cdots\subseteq\mathbb{B}_{t;\sigma}$, where

$$\mathbb{B}_{m;\sigma} \;:=\; \bigoplus_{\tau\in\Sigma_{k;\sigma}} \mathbb{A}_{m;\sigma,\tau}$$

for $m=k+1,\ldots,t$. In particular, if $m=k+1\leqslant t$, then this relation becomes

$$\mathbb{B}_{k+1;\sigma} \;\cong\; \mathbb{A}_{k+1;\sigma,1}\oplus\cdots\oplus\mathbb{A}_{k+1;\sigma,l_\sigma}. \tag{5.2}$$

This corresponds to the factorization

$$\pi_{k;\sigma}(\mu_{k+1}) \;=\; \mu_{k+1;\sigma,1}\cdots\mu_{k+1;\sigma,l_\sigma}, \tag{5.3}$$

where we recall that $\mu_{k+1;\sigma,i}\in\mathbb{A}_{k;\sigma}[x]$ stands for the defining polynomial of $\mathbb{A}_{k+1;\sigma,i}$ for $i=1,\ldots,l_\sigma$, whereas $\pi_{k;\sigma}(\mu_{k+1})\in\mathbb{A}_{k;\sigma}[x]$ is the defining polynomial of $\mathbb{B}_{k+1;\sigma}$.

5.3. Multi-modular representations

If $t=1$, then (5.1) reduces into the well known isomorphism

$$\mathbb{K}[x]/(\mu_1) \;\cong\; \mathbb{K}[x]/(\mu_{1;1})\oplus\cdots\oplus\mathbb{K}[x]/(\mu_{1;l_()}) \tag{5.4}$$

from the Chinese remainder theorem. We may thus view the isomorphism (5.1) as a generalized Chinese remainder theorem. The *multi-modular representation* of an element in $\mathbb{A}_t$ is simply its image under this isomorphism. The aim of this section is to present efficient algorithms for conversions between the usual and the multi-modular representations.

For the usual Chinese remainder theorem, efficient algorithms are well known for carrying out the corresponding conversions [6, 23, 48]. These algorithms are based on the technique of so-called *remainder trees*, for which recent improvements can be found in [4, 10, 35]. In particular, if $t=1$ then the conversions from (5.4) can be carried out in time $O(\mathsf{M}_{\mathbb{K}}(d)\log d)$; see for instance [27, Chapter 10]. These fast algorithms actually work in our context. This means that the isomorphism

$$\Phi_{k;\sigma}:\ \mathbb{A}_{k;\sigma}[x]/(\pi_{k;\sigma}(\mu_{k+1})) \;\cong\; \mathbb{A}_{k;\sigma}[x]/(\mu_{k+1;\sigma,1})\oplus\cdots\oplus\mathbb{A}_{k;\sigma}[x]/(\mu_{k+1;\sigma,l_\sigma})$$

from (5.2) and (5.3) can be computed with complexity $O(\mathsf{M}_{\mathbb{A}_{k;\sigma}}(d_{k+1})\log d_{k+1})$, whenever $\pi_{k;\sigma}(\mu'_{k+1})^{-1}$ modulo $\mu_{k+1;\sigma,i}$ are precomputed for all $i=1,\ldots,l_\sigma$. In fact, if a_i are elements of $\mathbb{A}_{k;\sigma}[x]/(\mu_{k+1;\sigma,i})$ with natural preimages $f_i\in\mathbb{A}_{k;\sigma}[x]$ for $i=1,\ldots,l_\sigma$, then the natural preimage of $\Phi^{-1}_{k;\sigma}(a_1,\ldots,a_{l_\sigma})$ can be computed as

$$\sum_{i=1}^{l_\sigma} (f_i\,\mu'_{k+1;\sigma,i}\,\pi_{k;\sigma}(\mu'_{k+1})^{-1} \bmod \mu_{k+1;\sigma,i})\,\frac{\pi_{k;\sigma}(\mu_{k+1})}{\mu_{k+1;\sigma,i}}, \tag{5.5}$$

using fast "linear combination for linear moduli" [27, Algorithm 10.9].

The idea is to combine these "isomorphisms at nodes Σ_k" in order to compute the global isomorphism from (5.1). For the complexity analysis, it is convenient to introduce the following maximal normalized cost of multiplication in any of the factors of the tree factorization:

$$\mathsf{H} := \max_{0 \leqslant k < t} \max_{\sigma \in \Sigma_k} \frac{\mathsf{M}_{\mathbb{A}_{k;\sigma}}(d_{k+1})}{d_{k+1} \dim_{\mathbb{K}} \mathbb{A}_{k;\sigma}}. \tag{5.6}$$

For the time being, we may use Proposition 2.4 as a complexity bound for $\mathsf{M}_{\mathbb{A}_{k;\sigma}}(d_{k+1})$. The conversion of elements in $\mathbb{A}_t$ into elements in $\bigoplus_{\sigma \in \Sigma_t} \mathbb{A}_{t;\sigma}$ is called *multi-modular reduction* and we may use the following algorithm for this operation:

Algorithm 5.1
Input: a tree factorization $((\mathbb{A}_{i;\sigma})_{i \leqslant t})_{\sigma \in \Sigma_t}$ of a tower $(\mathbb{A}_i)_{i \leqslant t}$, and $a \in \mathbb{A}_t$.
Output: $(\pi_{t;\sigma}(a))_{\sigma \in \Sigma_t}$.

1. If $t = 0$, then return $(a)_{\sigma \in \Sigma_0}$.
2. Expand $a = a_0 + \cdots + a_{d_t - 1} \alpha_t^{d_t - 1}$ with $a_0, \ldots, a_{d_t - 1} \in \mathbb{A}_{t-1}$.
3. For $i = 0, \ldots, d_{t-1} - 1$, recursively compute $(\pi_{t-1;\sigma}(a_i))_{\sigma \in \Sigma_{t-1}}$.
4. For each $\sigma \in \Sigma_{t-1}$ do:
   a. Set $b := \pi_{t-1;\sigma}(a_0) + \cdots + \pi_{t-1;\sigma}(a_{d_t - 1})\, x^{d_t - 1} \in \mathbb{A}_{t-1;\sigma}[x]$.
   b. Compute $b_i := b \operatorname{rem} \mu_{t;\sigma,i}$ for $i = 1, \ldots, l_\sigma$, so that $\pi_{t;\sigma,i}(a) = b_i \bmod \mu_{t;\sigma,i}$.
5. Collect and return the family $(\pi_{t;\sigma,i}(a))_{(\sigma,i) \in \Sigma_t}$.

PROPOSITION 5.1. *Algorithm 5.1 is correct and runs in time $O(\mathsf{H} d \log d)$.*

Proof. The correctness is straightforward from the definitions. We perform step 4b using fast univariate multi-modular reduction. Let A be a universal constant such that multi-modular reduction of a polynomial of degree n over an arbitrary effective ring $\mathbb{A}$ can be performed in time $A\, \mathsf{M}_{\mathbb{A}}(n) \log n$.

Let us show by induction over t that the algorithm runs in time $A\, \mathsf{H} d \log d$. For $t = 0$, the result is obvious, so assume that $t > 0$. By the induction hypothesis, step 3 runs in time $d_t A\, \mathsf{H} (d/d_t) \log (d/d_t) = A\, \mathsf{H} d \log (d/d_t)$. By the definition of A, the contribution of step 4b to the complexity is bounded by

$$\begin{aligned} \sum_{\sigma \in \Sigma_{t-1}} A\, \mathsf{M}_{\mathbb{A}_{t-1;\sigma}}(d_t) \log d_t &\leqslant A\, \mathsf{H} d_t \log d_t \sum_{\sigma \in \Sigma_{t-1}} \dim_{\mathbb{K}} \mathbb{A}_{t-1;\sigma} \\ &= A\, \mathsf{H} d \log d_t. \end{aligned}$$

Adding up, it follows that the algorithm runs in time $A\, \mathsf{H} d \log d$. □

The opposite conversion of elements in $\bigoplus_{\sigma \in \Sigma_t} \mathbb{A}_{t;\sigma}$ into elements in $\mathbb{A}_t$ is called *Chinese remaindering*; we may use the following algorithm for this operation:

Algorithm 5.2
Input: a tree factorization $((\mathbb{A}_{i;\sigma})_{i \leqslant t})_{\sigma \in \Sigma_t}$ of $(\mathbb{A}_i)_{i \leqslant t}$ and a family $(a_\sigma)_{\sigma \in \Sigma_t}$ with $a_\sigma \in \mathbb{A}_{t;\sigma}$.
Output: $a \in \mathbb{A}_t$ such that $\pi_{t;\sigma}(a) = a_\sigma$ for all $\sigma \in \Sigma_t$.

Assumption: $\pi_{k;\sigma}(\mu'_{k+1})^{-1}$ modulo $\mu_{k+1;\sigma,i}$ are precomputed for all $\sigma\in\Sigma_k$, $k=0,\ldots,t-1$, and $i=1,\ldots,l_\sigma$.

1. If $t=0$, then Σ_0 is the singleton $\{()\}$ made of the empty tuple, so return $a_{()}$.
2. For each $\sigma\in\Sigma_{t-1}$ do:

   Compute $b_\sigma = b_{\sigma;0}+\cdots+b_{\sigma;d_t-1}x^{d_t-1}$ such that $a_{\sigma;i}=b_\sigma \bmod \mu_{t;\sigma,i}$ for $i=1,\ldots,l_\sigma$.
3. For $i=0,\ldots,d_t-1$ do:

   Recursively compute $a_i\in\mathbb{A}_{t-1}$ such that $\pi_{t-1;\sigma}(a_i)=b_{\sigma;i}$ for all $\sigma\in\Sigma_{t-1}$.
4. Return $a_0+\cdots+a_{d_t-1}\alpha_{t-1}^{d_t-1}$.

PROPOSITION 5.2. *Algorithm 5.2 is correct and costs* $O(\mathsf{H}d\log d)$.

Proof. The proof is very similar to the one of Proposition 5.1. The polynomials $\pi_{k;\sigma}(\mu'_{k+1})^{-1}$ modulo $\mu_{k+1;\sigma,i}$ are actually used in the Chinese remaindering of step 2 *via* formula (5.5). □

5.4. Factorization

Factoring univariate polynomials over an arbitrary effective field $\mathbb{K}$ is generally expensive or even impossible [24, 25]. Still, if an algorithm for this task is given, then it is interesting to study how to exploit it for the computation of an irreducible tree factorization $((\mathbb{A}_{i;\sigma})_{i\leqslant t})_{\sigma\in\Sigma_t}$ of a given separable tower $(\mathbb{A}_i)_{i\leqslant t}$. Once such an irreducible tree factorization is known, arithmetic and other operations can potentially be sped up by switching to the multi-modular representation thanks to Algorithms 5.1 and 5.2.

For the complexity analysis, the function $\mathsf{F}_{\mathbb{L}}(n)$ represents the time necessary to factor a polynomial in $\mathbb{L}[x]_{\leqslant n}$, and it is again convenient to introduce the following maximal normalized cost of factoring univariate polynomials over any of the fields in the factor towers:

$$\Phi := \max_{0\leqslant k<t}\max_{\sigma\in\Sigma_k}\frac{\mathsf{F}_{\mathbb{A}_{k;\sigma}}(d_{k+1})}{d_{k+1}\dim_{\mathbb{K}}\mathbb{A}_{k;\sigma}}. \tag{5.7}$$

Here we notice that each of the irreducible factors $(\mathbb{A}_{i;\sigma})_{i\leqslant t}$ is a tower of separable fields, so we may directly use the accelerated arithmetic from section 4 for computations over any of the $\mathbb{A}_{i;\sigma}$.

Algorithm 5.3
Input: a separable tower $(\mathbb{A}_i)_{i\leqslant t}$.
Output: the irreducible tree factorization $((\mathbb{A}_{i;\sigma})_{i\leqslant t})_{\sigma\in\Sigma_t}$ of $(\mathbb{A}_i)_{i\leqslant t}$.

1. If $t=0$, then return $((\mathbb{A}_{i;\sigma})_{i\leqslant t})_{\sigma\in\Sigma_0}$.
2. Recursively compute the irreducible tree factorization $((\mathbb{A}_{i;\sigma})_{i\leqslant t-1})_{\sigma\in\Sigma_{t-1}}$ of $(\mathbb{A}_i)_{i\leqslant t-1}$.
3. Compute $(\pi_{t-1;\sigma}(\mu_t))_{\sigma\in\Sigma_{t-1}}$ using Algorithm 5.1.
4. For each $\sigma\in\Sigma_{t-1}$ do:
   a. Compute the irreducible factors $\mu_{t;\sigma,1},\ldots,\mu_{t;\sigma,l_\sigma}\in\mathbb{A}_{t-1;\sigma}[x]$ of $\pi_{t-1;\sigma}(\mu_t)$.
   b. Let $\mathbb{A}_{t;\sigma,i} := \mathbb{A}_{t-1;\sigma}[x]/(\mu_{t;\sigma,i})$ for $i=1,\ldots,l_\sigma$.
5. Return $((\mathbb{A}_{i;\sigma})_{i\leqslant t})_{\sigma\in\Sigma_t}$, where $\Sigma_t=\{(\sigma,i):\sigma\in\Sigma_{t-1},i\in\{1,\ldots,l_\sigma\}\}$.

PROPOSITION 5.3. *Algorithm 5.3 is correct and takes time at most* $2\Phi d + O(\mathsf{H} d \log d)$ *whenever* $d_i \geqslant 2$ *for* $i = 1, \ldots, t$.

Proof. The cost of all factorizations in step 4a is bounded by

$$\sum_{\sigma \in \Sigma_{t-1}} \mathsf{F}_{\mathbb{A}_{t-1;\sigma}}(d_t) \leqslant \Phi d_t \sum_{\sigma \in \Sigma_{t-1}} \dim_{\mathbb{K}} \mathbb{A}_{t-1,\sigma} = \Phi d.$$

Together with recursive calls, the total cost of all factorizations is bounded by

$$\Phi (d_1 \cdots d_t + d_1 \cdots d_{t-1} + \cdots + d_1) \leqslant 2\Phi d.$$

Step 3 takes time $O(\mathsf{H} d \log d)$, by Proposition 5.1, and the same step for the recursive calls amounts to a similar complexity. We conclude by adding up these contributions. □

We are now ready to present a major corollary of this result; in combination with Algorithms 5.1 and 5.2, it reduces arithmetic in general separable towers to arithmetic in accelerated towers of fields.

COROLLARY 5.4. *The irreducible tree factorization of a separable tower* $(\mathbb{A}_i)_{i \leqslant t}$ *can be computed in time* $2\Phi d + (\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}}) d^{1+o(1)}$.

Proof. This follows from Proposition 5.3 combined to Corollary 4.11, which yields $\mathsf{H} = (\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}}) d^{o(1)}$. □

5.5. Remarks about special coefficient fields

For certain specific fields of coefficients $\mathbb{K}$, factorization of univariate polynomials over $\mathbb{K}$ can be reasonably cheap. Let us briefly study two particular such cases.

Complex numbers Let us first consider the case when $\mathbb{K}$ is a subfield of $\mathbb{C}$. In practice, this usually means that elements of $\mathbb{K}$ are represented approximately by complex floating point numbers of fixed bit precision p. The bit complexity of the field operations in $\mathbb{A}$ is then bounded by $O(\mathsf{I}(p))$, where $\mathsf{I}(p)$ bounds the cost of p-bit integer multiplication. It has recently been shown that $\mathsf{I}(p) = O(p \log p 4^{\log^* p}) = \tilde{O}(p)$; see [31, 32, 33].

Even if $\mathbb{K} = \mathbb{Q}$ or if $\mathbb{K}$ is an algebraic number field, then it may be useful to temporarily convert coefficients in $\mathbb{K}$ into p bit complex floating point numbers and convert the results of computations back using rational reconstruction techniques [27, Chapter 5].

A convenient framework for analyzing the "ultimate complexity" of numeric algorithms was introduced in [38]. Concerning the factorization of complex polynomials of degree d, it was shown therein that all roots could be computed at precision p in time $O(\mathsf{I}(dp) \log d)$ for "sufficiently large precisions" p. The actual precision from which this bound becomes relevant highly depends on the location of the roots. As a rule of thumb, a precision $p \geqslant d$ is often required and sufficient, but we refer the reader to the seminal works of Schönhage, Pan and others for details [47, 54].

Since $\mathbb{C}$ is algebraically closed, towers of separable algebraic extensions can be factored into towers of degree one. From the "ultimate bit complexity" point of view from [38], it follows that the ultimate bit complexity of the conversions in Propositions 5.1 and 5.2 becomes $O(\mathsf{I}(dp) \log d)$ and so does the cost of factoring itself in Proposition 5.3. Using our rule of thumb, we expect that a precision p of the order of $\bar{d} := \max(d_1, \ldots, d_t)$ should be sufficient in practice in order to observe these complexity bounds. Small values of $\bar{d}$ are thus expected to be favourable for this type of coefficients (notice that this was the least favourable case for general coefficients!). A refined comparison between the bit complexities of this approach and the building of accelerated towers would require efforts that are beyond the scope of the present paper.

Finite fields Let us next consider the case when $\mathbb{K} = \mathbb{F}_q$ is a finite field of characteristic $p > 0$, with $q = p^\kappa$ elements. The bit complexity of multiplying two polynomials in $\mathbb{F}_q[x]$ is bounded by $\mathsf{M}_{\mathbb{F}_q}(n) = O(n \log q \log(n \log q)\, 4^{\log^*(n \log q)})$, uniformly in q; see [30, 34]. Fast computations in towers of finite field extensions $\mathbb{F}_q \subseteq \mathbb{F}_{q^{d_1}} \subseteq \mathbb{F}_{q^{d_1 d_2}} \subseteq \cdots \subseteq \mathbb{F}_{q^{d_1 \cdots d_t}}$ were treated quite extensively in [39]; see also [17] for alternative approaches. The case when $\bar{d} := \max(d_1, \ldots, d_t)$ is small is most favourable. The reason is that the defining polynomial μ_t of $\mathbb{F}_{q^{d_1 \cdots d_t}}$ can be factored over each of the subfields $\mathbb{F}_{q^{d_1 \cdots d_i}}$ efficiently. It is also worth to mention that [39] contains various fast algorithms for the computation of primitive elements in towers of finite fields. Once again, these algorithms are most efficient whenever $\bar{d}$ remains small. It would be interesting to exploit these techniques to further accelerate arithmetic in towers of finite fields.

6. PRIMITIVE ELEMENT REPRESENTATIONS

This section revisits the computation of traces and characteristic polynomials in towers of field extensions over a general field by using fast products. Our algorithms are rather different from those of [52]: we compute traces faster, but for characteristic polynomials and primitive elements the bottleneck lies in the baby-step giant-step method, so the complexity exponents in terms of ω turn out to be the same as in [52] (notice that we gain a logarithmic factor t). Until the end of this section, $(\mathbb{K}_i)_{i \leqslant t}$ is a tower of separable field extensions of degrees $d_i \geqslant 2$ and $1 < C \leqslant 3$ is as in Proposition 2.4.

6.1. Trace computations in towers

Let us recall how to compute traces in a tower $(\mathbb{K}_i)_{i \leqslant t}$ of separable field extensions. We let $\tilde{\mu}_i(z) = z^{d_i} \mu_i(1/z)$ represent the *reverse polynomial* of μ_i. Its constant coefficient is 1 and the trace function $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$ of $\mathbb{K}_i$ over $\mathbb{K}_{i-1}$ may be computed as the first d_i terms of the power series

$$-\frac{\tilde{\mu}_i'(z)}{\tilde{\mu}_i(z)} = \mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}(\alpha_i) + \mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}(\alpha_i^2)\, z + \cdots + \mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}(\alpha_i^{d_i})\, z^{d_i - 1} + O(z^{d_i}).$$

In other words, the trace map $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$ can be regarded as a vector in $\mathbb{K}_{i-1}^{d_i}$ that can be computed in time $O(\mathsf{M}_{\mathbb{K}_{i-1}}(d_i))$: indeed this vector represents the $1 \times d_i$ matrix of $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$ with respect to the basis $1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{d_i - 1}$. Given such a vector representation, the individual evaluation of $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$ at an element in $\mathbb{K}_i$ takes time $O(\mathsf{m}_{\mathbb{K}_{i-1}} d_i)$.

More generally, $\mathbb{K}_i$ is a $\mathbb{K}_j$-algebra for all $i > j$, and the relative trace function satisfies

$$\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_j} = \mathrm{Tr}_{\mathbb{K}_{j+1}/\mathbb{K}_j} \circ \mathrm{Tr}_{\mathbb{K}_{j+2}/\mathbb{K}_{j+1}} \circ \cdots \circ \mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}.$$

LEMMA 6.1. *Assume that products in $\mathbb{K}_i$ are performed by linear algorithms over $\mathbb{K}$ in the sense of [10] (and as is always the case in this paper). Then the vector representation of the trace map $\mathrm{Tr}_{\mathbb{K}_t/\mathbb{K}}$ can be computed in time*

$$O\left(\sum_{i=1}^{t} \mathsf{M}_{\mathbb{K}_{i-1}}(d_i)\right).$$

Proof. First we compute $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$, as a vector in $\mathbb{K}_{i-1}^{d_i}$, for $i=1,\ldots,t$. This computation takes time $O(\sum_{i=1}^{t} \mathsf{M}_{\mathbb{K}_{i-1}}(d_i))$. We regard each $\mathrm{Tr}_{\mathbb{K}_i/\mathbb{K}_{i-1}}$ as a $\mathbb{K}$-linear map ℓ_i from $\mathbb{K}^{d_1\cdots d_i}$ to $\mathbb{K}^{d_1\cdots d_{i-1}}$, whose evaluation takes time $O(\mathsf{m}_{\mathbb{K}_{i-1}} d_i)$. Regarding $\mathrm{Tr}_{\mathbb{K}_1/\mathbb{K}}$ as a vector $v_1 \in \mathbb{K}^{d_1}$, we apply the transpose of ℓ_2 to v_1 in order to obtain a new vector $v_2 \in \mathbb{K}^{d_1 d_2}$ that represents $\mathrm{Tr}_{\mathbb{K}_2/\mathbb{K}}$. This computation takes time $O(\mathsf{m}_{\mathbb{K}_1} d_2)$ thanks to the transposition principle (as explained in [10]). We next apply the transpose of ℓ_3 to v_2 and obtain a vector $v_3 \in \mathbb{K}^{d_1 d_2 d_3}$ that represents $\mathrm{Tr}_{\mathbb{K}_3/\mathbb{K}}$. Continuing this way, we obtain $\mathrm{Tr}_{\mathbb{K}_t/\mathbb{K}}$ as a vector $v_t \in \mathbb{K}^d$ in time $O(\sum_{i=1}^{t} \mathsf{m}_{\mathbb{K}_{i-1}} d_i) = O(\sum_{i=1}^{t} \mathsf{M}_{\mathbb{K}_{i-1}}(d_i))$. □

6.2. Characteristic polynomials

We may take advantage of accelerated towers to compute a primitive element $\beta \in \mathbb{K}_t$ over $\mathbb{K}$, together with its minimal polynomial ν and the parametrizations $\varphi_i \in \mathbb{K}[x]_{<d}$ with $\alpha_i = \varphi_i(\beta)$ for $i=1,\ldots,t$. We first consider the computation of characteristic polynomials.

THEOREM 6.2. *Assume that a δ-accelerated tower $(\mathbb{L}_j)_{j\leqslant s}$ for $(\mathbb{K}_i)_{i\leqslant t}$ is given, with $\delta^{\omega-1} = O(\sqrt{d})$, and that the inverses of $2,\ldots,d$ are available in $\mathbb{K}$. Then the characteristic polynomial of $a \in \mathbb{K}_t$ over $\mathbb{K}$ can be computed in time $O\big(\mathsf{m}_{\mathbb{K}} d^{\omega} + C^s \mathsf{M}_{\mathbb{K}}(d)\sqrt{d}\big)$.*

Proof. Lemma 6.1 and Proposition 2.4 allow us to compute the representation of $\mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}$ as a vector in $\mathbb{K}^d$ in time

$$O\left(\sum_{j=1}^{s} \mathsf{M}_{\mathbb{L}_{j-1}}(e_j)\right) \;=\; O(C^s \mathsf{M}_{\mathbb{K}}(d)).$$

Using Lemma 4.8, we convert a into an element b of $\mathbb{L}_s$ in time $O(C^s \mathsf{M}_{\mathbb{K}}(d)\,\delta^{\omega-1})$. By using the transposed product in $\mathbb{L}_s$ the vector representation of the linear form $\ell : z \mapsto \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(bz)$ can be computed in time $O(C^s \mathsf{M}_{\mathbb{K}}(d))$. Then, as discussed in section 1.2, the computation of

$$\ell(1) = \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(b), \ldots, \ell(b^{d-1}) = \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(b^d)$$

is achieved by transposing Algorithm 3.1, so Proposition 3.1 yields the cost

$$O\big(\mathsf{m}_{\mathbb{L}_s}\sqrt{d} + \mathsf{m}_{\mathbb{K}} d^{\omega}\big) = O\big(\mathsf{m}_{\mathbb{K}} d^{\omega} + C^s \mathsf{M}_{\mathbb{K}}(d)\sqrt{d}\big).$$

We finally use the Newton–Girard identity (1.1) to recover the characteristic polynomial of b with further cost $O(\mathsf{M}_{\mathbb{K}}(d))$; see for instance [29, section 2.4]. The conclusion follows. □

By taking $\delta = d^{\epsilon(d)}$ as in Corollary 4.11, the expected cost of Theorem 6.2 simplifies to $O(\mathsf{m}_{\mathbb{K}} d^{\omega})$, thanks to the assumption $\omega > 3/2$.

6.3. Primitive elements

We are now in a position to design a randomized algorithm with subquadratic expected cost for computing a primitive element representation of $\mathbb{K}_t$ over $\mathbb{K}$.

THEOREM 6.3. *Assume that* $\mathrm{card}\,\mathbb{K} \geqslant 2d(2d-1)$, *that the inverses of $2,\ldots,d$ are available in $\mathbb{K}$, and that a δ-accelerated tower $(\mathbb{L}_j)_{j\leqslant s}$ for $(\mathbb{K}_i)_{i\leqslant t}$ is given, with $\delta^{\omega-1} = O(\sqrt{d})$. Then a primitive element representation $\beta, \nu, \varphi_1, \ldots, \varphi_t$ of $\mathbb{K}_t$ over $\mathbb{K}$ can be computed by a randomized Las Vegas algorithm with expected cost*

$$O\big(\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}} d^{\omega} t + C^s \mathsf{M}_{\mathbb{K}}(d)\sqrt{d}\, t\big).$$

Taking $\delta = d^{\epsilon(d)}$ as in Corollary 4.11, this cost further simplifies into $O(\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}}\, d^{\omega} t)$.

Proof. The characteristic polynomial ν of $\beta = \lambda_1\beta_1 + \cdots + \lambda_s\beta_s$ has total degree d in $\lambda_1, \ldots, \lambda_t$. So its discriminant is a polynomial in $\mathbb{K}[\lambda_1, \ldots, \lambda_t]$ of total degree $d\,(2\,d-1)$. By the Schwartz–Zippel lemma [55, 57], picking β at random with the λ_i in a set of size $2\,d\,(2\,d-1)$ leads to a primitive element with probability $\geqslant 1/2$. Computing the characteristic polynomial of β takes time $O\big(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d}\big)$ by Theorem 6.2. We can verify that β is a primitive element by checking that its characteristic polynomial is separable with further cost $O(\mathsf{M}_{\mathbb{K}}(d) \log d)$. Consequently the expected time to find a primitive element is $O\big(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d}\big)$.

Now assume that a primitive element β has been found, and let us consider the computation of polynomials $\psi_j \in \mathbb{K}[x]_{<d}$ such that $\beta_j = \psi_j(\beta)$ for $j = 1, \ldots, s$. We follow Kronecker's classical deformation technique of the primitive element; see for instance [28, section 3.3]. For this purpose, we first introduce a new formal indeterminate τ with $\tau^2 = 0$, so that $\mathbb{K}[\tau] = \mathbb{K} \oplus \mathbb{K}\,\tau$ is the ring of tangent numbers over $\mathbb{K}$. We aim at computing the characteristic polynomial of $\beta + \beta_j\,\tau \in \mathbb{L}_s[\tau]$ over $\mathbb{K}[\tau]$. Then

$$\mathrm{Tr}_{\mathbb{L}_s[\tau]/\mathbb{K}[\tau]}((\beta + \beta_j\,\tau)^k) = \mathrm{Tr}_{\mathbb{L}_s[\tau]/\mathbb{K}[\tau]}(\beta^k + k\,\beta_j\,\beta^{k-1}\,\tau) = \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(\beta^k) + k\,\mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(\beta_j\,\beta^{k-1})\,\tau.$$

Notice that $\mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}$ is already available at this point. The computation of the vector representation of the linear form $b \mapsto \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(\beta_j b)$ can be done in time $\mathsf{m}_{\mathbb{L}_s} = O(C^s\, \mathsf{M}_{\mathbb{K}}(d))$ by the transposition principle. Therefore $\mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(\beta_j), \ldots, \mathrm{Tr}_{\mathbb{L}_s/\mathbb{K}}(\beta_j\,\beta^{d-1})$ is obtained by transposing Algorithm 3.1. In view of Proposition 3.1, this computation takes time

$$O\big(\mathsf{m}_{\mathbb{L}_s}\, \sqrt{d} + \mathsf{m}_{\mathbb{K}}\, d^{\omega}\big) = O\big(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d}\big).$$

At this point, we have computed the traces $\mathrm{Tr}_{\mathbb{L}_s[\tau]/\mathbb{K}[\tau]}((\beta + \beta_j\,\tau)^k)$ for all $k = 1, \ldots, d$. We deduce the characteristic polynomial $\nu(x) + \tilde{\nu}_j(x)\,\tau$ of $\beta + \beta_i\,\tau$ using Newton–Girard's formula (1.1), so that

$$\nu(\beta + \beta_j\,\tau) + \tilde{\nu}_j(\beta + \beta_j\,\tau)\,\tau \;=\; \nu(\beta) + (\nu'(\beta)\,\beta_j + \tilde{\nu}_j(\beta))\,\tau \;=\; 0.$$

It follows that $\beta_j = -\tilde{\nu}_j(\beta)\,/\,\nu'(\beta) =: \psi_j(\beta)$. The total cost to obtain all the ψ_j for $j = 1, \ldots, s$ is

$$O\big(\mathsf{d}_{\mathbb{K}} + \mathsf{m}_{\mathbb{K}}\, s\, d^{\omega} + s\, C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d}\big).$$

Now let a be an element of $\mathbb{K}_t$. It can be converted into an element $f(\beta_1, \ldots, \beta_s)$ of $\mathbb{L}_s$ in time $\mathrm{O}(C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1})$ by Lemma 4.8, where $f(x_1, \ldots, x_s)$ has partial degree $<e_j$ in x_j for $i = 1, \ldots, s$. The expression of a in terms of β is obtained as $f(\psi_1, \ldots, \psi_s)$ rem ν in time $O\big(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d}\big)$ by Proposition 3.3. By successively taking $\alpha_1, \ldots, \alpha_t$ for a, the polynomials $\varphi_1, \ldots, \varphi_t$ can all be obtained in time

$$O\big(\big(\mathsf{m}_{\mathbb{K}}\, d^{\omega} + \mathsf{M}_{\mathbb{K}}(d)\, \sqrt{d} + C^s\, \mathsf{M}_{\mathbb{K}}(d)\, \delta^{\omega-1}\big)\, t\big).$$

Finally, thanks to $\delta = d^{\epsilon(d)}$, the assumption $\omega > 3/2$ implies the simplified complexity bound. □

CONCLUSION

We have proved nearly optimal complexity bounds for basic arithmetic operations in towers of fields of arbitrary height. Our results are based on the newly introduced concept of "accelerated" towers. Two major problems remain open:

- *Do there exist algorithms of quasi-linear complexity $\tilde{O}(d)$ instead of $d^{1+o(1)}$?*

It is well known that this is indeed the case for towers of bounded height $t = O(1)$: see Propositions 2.4 and 2.7. In section 5.5, we have singled out a few other situations when quasi-linear algorithms do exist. We refer to [7] for yet a few other examples.

- *Can one efficiently construct accelerated towers for general separable towers, without relying on univariate polynomial factorization as in section 5?*

  One obvious strategy is to use the dynamic evaluation paradigm [19, 20] that we already discussed in the introduction, while controlling complexity issues using techniques from [16]. We do not see any serious obstacle to a positive answer, but the technical details are beyond the scope of this paper. We intend to come back to this problem in a future work.

Another major challenge concerns efficient implementations. Asymptotically fast algorithms based on Proposition 2.4 obviously only become interesting when d is in the range where evaluation-interpolation style algorithms are used for univariate arithmetic. For instance, FFT-based algorithms are typically used for $d \geqslant 100$.

For such sufficiently large d, we expect our acceleration techniques to be useful. For instance, assume that d_1 and d_2 are both small, say $d_1 = d_2 = 2$. Then conversions between $\mathbb{K}[\alpha_1, \alpha_2]$ and an equivalent primitive representation $\mathbb{K}[\beta]$ can be done very fast. Conversions between $\mathbb{K}[\alpha_1, \ldots, \alpha_t]$ and $\mathbb{K}[\beta, \alpha_3, \ldots, \alpha_t]$ can be done coefficientwise, so they are also fast. Without much cost, this allows us to diminish the height t by one and gain a constant factor C when applying Proposition 2.4.

This discussion indicates that our techniques should be of practical interest for sufficiently large d and generic base fields $\mathbb{K}$. For specific base fields $\mathbb{K}$ and specific types of towers, various alternative approaches have been proposed [17, 38, 39], and it would require a more significant implementation effort to figure out which approaches are best; we intend to address to this issue in the near future.

Acknowledgments We thank the anonymous referees for their helpful comments.

BIBLIOGRAPHY

[1] Ph. Aubry, D. Lazard, and M. Moreno Maza. On the theories of triangular sets. *J. Symbolic Comput.*, 28(1–2):105–124, 1999.

[2] S. Baktir, J. Pelzl, T. Wollinger, B. Sunar, and C. Paar. Optimal tower fields for hyperelliptic curve cryptosystems. In *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004*, volume 1, pages 522–526. IEEE, 2004.

[3] N. Benger and M. Scott. Constructing tower extensions of finite fields for implementation of pairing-based cryptography. In M. Anwar Hasan and T. Helleseth, editors, *Arithmetic of Finite Fields. Third International Workshop, WAIFI 2010. Istanbul, Turkey, June 27-30, 2010. Proceedings*, volume 6087 of *Lecture Notes in Comput. Sci.*, pages 180–195. Springer-Verlag Berlin Heidelberg, 2010.

[4] D. Bernstein. Scaled remainder trees. Available from `https://cr.yp.to/arith/scaledmod-20040820.pdf`, 2004.

[5] J. Berthomieu, G. Lecerf, and G. Quintin. Polynomial root finding over local rings and application to error correcting codes. *Appl. Algebra Engrg. Comm. Comput.*, 24(6):413–443, 2013.

[6] A. Borodin and R. T. Moenck. Fast modular transforms. *J. Comput. System Sci.*, 8:366–386, 1974.

[7] A. Bostan, M. F. I. Chowdhury, J. van der Hoeven, and É. Schost. Homotopy methods for multiplication modulo triangular sets. *J. Symbolic Comput.*, 46(12):1378–1402, 2011.

[8] A. Bostan, F. Chyzak, F. Ollivier, B. Salvy, É. Schost, and A. Sedoglavic. Fast computation of power series solutions of systems of differential equations. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1012–1021, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[9] A. Bostan, L. Gonzales-Vega, H. Perdry, and É. Schost. Complexity issues on Newton sums of polynomials. Distributed in the digital proceedings of MEGA'05, 2005.

[10] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In Hoon Hong, editor, *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, ISSAC '03, pages 37–44, New York, NY, USA, 2003. ACM.

[11] A. Bostan, B. Salvy, and É. Schost. Fast algorithms for zero-dimensional polynomial systems using duality. *Appl. Algebra Engrg. Comm. Comput.*, 14(4):239–272, 2003.

[12] F. Boulier, D. Lazard, F. Ollivier, and M. Petitot. Computing representations for radicals of finitely generated differential ideals. *Appl. Algebra Engrg. Comm. Comput.*, 20(1):73–121, 2009.

[13] R. P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25(4):581–595, 1978.

[14] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften*. Springer-Verlag, 1997.

[15] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28:693–701, 1991.

[16] X. Dahan, M. Moreno Maza, É. Schost, and Yuzhen Xie. On the complexity of the D5 principle. In J.-G. Dumas, editor, *Proceedings of Transgressive Computing 2006: a conference in honor of Jean Della Dora*, pages 149–168. U. J. Fourier, Grenoble, France, 2006.

[17] L. De Feo, J. Doliskani, and É. Schost. Fast algorithms for l-adic towers over finite fields. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, ISSAC '13, pages 165–172, New York, NY, USA, 2013. ACM.

[18] L. De Feo and É. Schost. Fast arithmetics in Artin–Schreier towers over finite fields. *J. Symbolic Comput.*, 47(7):771–792, 2012.

[19] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In B. F. Caviness, editor, *EUROCAL '85: European Conference on Computer Algebra Linz, Austria, April 1–3 1985 Proceedings Vol. 2: Research Contributions*, pages 289–290. Springer Berlin Heidelberg, 1985.

[20] D. Duval. Algebraic numbers: An example of dynamic evaluation. *J. Symbolic Comput.*, 18(5):429–445, 1994.

[21] J.-C. Faugère, P. Gaudry, L. Huot, and G. Renault. Sub-cubic change of ordering for Gröbner basis: a probabilistic approach. In K. Nabeshima, editor, *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 170–177, New York, NY, USA, 2014. ACM.

[22] J.-C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *J. Symbolic Comput.*, 16(4):329–344, 1993.

[23] C. M. Fiduccia. Polynomial evaluation via the division algorithm: the fast Fourier transform revisited. In A. L. Rosenberg, editor, *Fourth annual ACM symposium on theory of computing*, pages 88–93, 1972.

[24] A. Fröhlich and J. C. Shepherdson. On the factorisation of polynomials in a finite number of steps. *Math. Z.*, 62:331–334, 1955.

[25] A. Fröhlich and J. C. Shepherdson. Effective procedures in field theory. *Philos. Trans. Roy. Soc. London. Ser. A.*, 248:407–432, 1956.

[26] A. Garcia and H. Stichtenoth. A tower of Artin–Schreier extensions of function fields attaining the Drinfeld–Vladut bound. *Invent. Math.*, 121(1):211–222, 1995.

[27] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 3rd edition, 2013.

[28] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *J. Complexity*, 17(1):154–211, 2001.

[29] B. Grenet, J. van der Hoeven, and G. Lecerf. Deterministic root finding over finite fields using Graeffe transforms. *Appl. Algebra Engrg. Comm. Comput.*, 27(3):237–257, 2016.

[30] D. Harvey and J. van der Hoeven. Faster integer and polynomial multiplication using cyclotomic coefficient rings. Technical report, ArXiv, 2017. http://arxiv.org/abs/1712.03693.

[31] D. Harvey and J. van der Hoeven. Faster integer multiplication using short lattice vectors. Technical report, ArXiv, 2018. http://arxiv.org/abs/1802.07932, to appear in *Proc. ANTS 2018*.

[32] D. Harvey and J. van der Hoeven. Faster integer multiplication using plain vanilla FFT primes. *Math. Comp.*, 88(315):501–514, 2019.

[33] D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. *J. Complexity*, 36:1–30, 2016.

[34] D. Harvey, J. van der Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. *J. ACM*, 63(6), 2017. Article 52.

[35] J. van der Hoeven. Faster Chinese remaindering. Technical report, CNRS & École polytechnique, 2016. http://hal.archives-ouvertes.fr/hal-01403810.

[36] J. van der Hoeven and G. Lecerf. On the complexity of multivariate blockwise polynomial multiplication. In J. van der Hoeven and M. van Hoeij, editors, *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, ISSAC '12, pages 211–218. ACM, 2012.

[37] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Comput.*, 50:227–254, 2013.

[38] J. van der Hoeven and G. Lecerf. Modular composition via complex roots. Technical report, CNRS & École polytechnique, 2017. http://hal.archives-ouvertes.fr/hal-01455731.

[39] J. van der Hoeven and G. Lecerf. Modular composition via factorization. *J. Complexity*, 48:36–68, 2018.

[40] Xiaohan Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[41] M. Kalkbrener. A generalized Euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symbolic Comput.*, 15(2):143–167, 1993.

[42] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[43] U. J. J. Le Verrier. Sur les variations séculaires des éléments elliptiques des sept planètes principales : Mercure, Venus, La Terre, Mars, Jupiter, Saturne et Uranus. *J. Math. Pures Appli.*, 1:220–254, 1840.

[44] R. Lebreton. Relaxed Hensel lifting of triangular sets. *J. Symbolic Comput.*, 68(2):230–258, 2015.

[45] G. Lecerf. On the complexity of the Lickteig–Roy subresultant algorithm. *J. Symbolic Comput.*, 92:243–268, 2019.

[46] X. Li, M. Moreno Maza, and É Schost. Fast arithmetic for triangular sets: from theory to practice. *J. Symbolic Comput.*, 44(7):891–907, 2009.

[47] J. M. McNamee and V. Y. Pan. *Numerical Methods for Roots of Polynomials, Part II*, volume 16 of *Studies in Computational Mathematics*. Elsevier, 2013.

[48] R. T. Moenck and A. Borodin. Fast modular transforms via division. In *Thirteenth annual IEEE symposium on switching and automata theory*, pages 90–96, Univ. Maryland, College Park, Md., 1972.

[49] M. Moreno Maza, É. Schost, and P. Vrbik. Inversion modulo zero-dimensional regular chains. In V. P. Gerdt, W. Koepf, E. W. Mayr, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing. 14th International Workshop, CASC 2012. Maribor, Slovenia, September 2012. Proceedings*, volume 7442 of *Lect. Notes Comput. Sci.*, pages 224–235. Springer Berlin Heidelberg, 2012.

[50] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

[51] A. Poteaux and É. Schost. Modular composition modulo triangular sets and applications. *Comput. Complex.*, 22(3):463–516, 2013.

[52] A. Poteaux and É. Schost. On the complexity of computing with zero-dimensional triangular sets. *J. Symbolic Comput.*, 50:110–138, 2013.

[53] J. F. Ritt. *Differential algebra*. Amer. Math. Soc., New York, 1950.

[54] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Math. Inst. Univ. of Tübingen, 1982.

[55] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[56] V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

[57] R. Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation. EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation, Marseille, France, June 1979*, number 72 in Lect. Notes Comput. Sci., pages 216–226. Springer Berlin Heidelberg, 1979.