Fast multivariate multi-point evaluation revisited

Joris van der Hoeven, Grégoire Lecerf

Fast multivariate multi-point evaluation revisited

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 July 24, 2018

In 2008, Kedlaya and Umans designed the first multivariate multi-point evaluation algorithm over finite fields with an asymptotic complexity that can be made arbitrarily close to linear. However, it remains a major challenge to make their algorithm efficient for practical input sizes. In this paper, we revisit and improve their algorithm, while keeping this ultimate goal in mind. In addition we sharpen the known complexity bounds for modular composition of univariate polynomials over finite fields.

1. INTRODUCTION

Let $\mathbb{A}$ be a commutative ring with unity, and let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be a multivariate polynomial. The *multi-point evaluation problem* consists in evaluating f at several given points $\alpha_1, \ldots, \alpha_N$ in $\mathbb{A}^n$. Let $g_1, \ldots, g_n$ be polynomials in $\mathbb{A}[x]$ of degrees $<d$ and let h be a monic polynomial in $\mathbb{A}[x]$ of degree d. The *modular composition problem* consists in computing $f(g_1, \ldots, g_n)$ modulo h. This is equivalent to the computation of the remainder $f(g_1, \ldots, g_n)$ rem h of the Euclidean division of $f(g_1, \ldots, g_n)$ by h. It turns out that these two problems are related and that they form important building blocks in computer algebra. Theoretically speaking, Kedlaya and Umans have given efficient solutions to both problems when $\mathbb{A}$ is a finite ring of the form $(\mathbb{Z}/r\mathbb{Z})[z]/(\theta(z))$ where θ is a monic polynomial [34]. The design of practically efficient algorithms remains an important challenge. The purpose of this paper is to revisit the algorithms by Kedlaya and Umans in detail, to sharpen their theoretical complexity bounds, and get some insight into the required data size for which this approach outperforms asymptotically slower algorithms.

1.1. Related work

Let $\mathsf{M}(d)$ denote a complexity function that bounds the number of operations in $\mathbb{A}$ required to multiply two polynomials of degree $\leqslant d$ in $\mathbb{A}[x]$. We will 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 [13, chapter 25, section 7] for technical details. The least integer larger or equal to x is written $\lceil x \rceil$. The

largest integer smaller or equal to x is written $\lfloor x \rfloor$. The $\mathbb{A}$-module of polynomials of degree $<d$ is denoted by $\mathbb{A}[x]_{<d} := \{P \in \mathbb{A}[x] : \deg P < d\}$.

Multi-point evaluation

In the univariate case when $n = 1$, the evaluation of $f \in \mathbb{A}[x]_{<d}$ at d points in $\mathbb{A}$ can be achieved with $O(\mathsf{M}(d) \log d)$ operations in $\mathbb{A}$. We refer the reader to [13, chapter 10] for the description of the well known algorithm based on remainder trees. Algorithms with the smallest constant hidden in the "O" may be found in [6]. By allowing precomputations that only depend on the set of points, this evaluation complexity even drops to $O((\mathsf{M}(d) \log d) / \log\log d)$ as shown in [23]. For specific sets of points, such as geometric progressions or TFT points, multi-point evaluation requires only $O(\mathsf{M}(d))$ operations in $\mathbb{A}$; see [4, 7, 21].

The univariate situation does not extend to several variables, unless the set S of evaluation points has good properties. For instance if S has the form $S_1 \times \cdots \times S_n$ with $S_i \subseteq \mathbb{A}$, then fast univariate evaluations may be applied coordinate by coordinate. Fast algorithms also exist for suitable initial segments of such Cartesian products [29]. Other specific families of sets of points are used for fast evaluation and interpolation of multivariate polynomials in sparse representation; see [1, 24] for some recent results.

In the bivariate case when $n = 2$, a smart use of the univariate case leads to a cost $\tilde{O}(\ell^3)$, where $\ell - 1$ bounds the partial degrees of f [36, Theorem 3]. In 2004, Nüsken and Ziegler improved this bound to $\tilde{O}(\ell^{\omega+1})$ [36, Result 4]—here the constant $\omega > 1.5$ is such that a $\sqrt{n} \times \sqrt{n}$ matrix over $\mathbb{A}$ may be multiplied with another $\sqrt{n} \times n$ rectangular matrix with $O(n^\omega)$ operations in $\mathbb{A}$. When $\mathbb{A}$ is a field the best currently known bound $\omega < 1.667$ is due to Huang and Pan [30, Theorem 10.1].

In 2008, Kedlaya and Umans achieved a major breakthrough for the general case [33]. In [34, Corollary 4.3] they showed the following statement (simplified here for conciseness): let $\varepsilon > 0$ be a fixed rational value, given $f(x_1, \ldots, x_n)$ in $(\mathbb{Z}/r\,\mathbb{Z})[x_1, \ldots, x_n]$ with partial degrees in any x_i at most $\ell - 1$, and evaluation points $\alpha_1, \ldots, \alpha_N$ in $(\mathbb{Z}/r\,\mathbb{Z})^n$, then $f(\alpha_1), \ldots, f(\alpha_N)$ can be computed with $((\ell^n + N) \log r)^{1+c\varepsilon}$ bit operations, provided that $n \leqslant \ell^\varepsilon$ and where c is a constant independent of f, n, ℓ, N, r. This result was stated for random access memory machines. In fact, some of the underlying arguments (such as the use of lookup tables) need to be adapted to make them work properly on Turing machines. This is one of our contributions in this paper.

Modular composition

Let us first discuss the standard modular composition problem when $n = 1$. Let f, g and h be polynomials in $\mathbb{A}[x]$ of respective degrees $<d$, $<d$ and d, with h monic. The naive modular composition algorithm takes $O(d\ \mathsf{M}(d))$ operations in $\mathbb{A}$. In 1978, Brent and Kung [9] gave an algorithm with cost $O(d^2 + \sqrt{d}\ \mathsf{M}(d))$, which uses the *baby-step giant-step* technique [39]. Their algorithm even yields a sub-quadratic cost $O(d^\omega + \sqrt{d}\ \mathsf{M}(d))$ when using fast linear algebra; see [32, p. 185].

The major breakthrough for this problem is again due to Kedlaya and Umans [33, 34] in the case when $\mathbb{A}$ is a finite field $\mathbb{F}_q$ (and even more generally a finite ring of the form $(\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$ for any integer r and θ monic). For any fixed real value $\varepsilon > 0$, they have shown that the composition $f \circ g$ could be computed modulo h using $O((d \log q)^{1+\varepsilon})$ bit operations.

The special case of power series composition corresponds to $h(x) = x^d$. The best known complexity bound in the algebraic model when $\mathbb{A}$ is a field, written $\mathbb{K}$ for convenience, is still due to Brent and Kung: in [9], they achieved $O(\sqrt{d}\,\mathsf{M}(d)\log^{1/2} d)$ operations in $\mathbb{K}$, under the condition that $g'(0)$ is invertible and that the characteristic of $\mathbb{K}$ is at least d/l, where $l := \lceil \sqrt{d/\log d} \rceil$. The variant proposed by van der Hoeven [20, section 3.4.3] raises the condition on $g'(0)$. For fields with small characteristic p, Bernstein [3] proposed an algorithm that is softly linear in d but linear in p. These algorithms have been generalized to moduli h of the form $\hbar^m$ in [25]; it is shown therein that such a composition reduces to one power series composition at order m over $\mathbb{K}[z]/(h(z))$, plus m compositions modulo $\hbar$, and one characteristic polynomial computation modulo $\hbar$. Let us finally mention that an optimized variant, in terms of the constant hidden in the "O", of the Brent–Kung algorithm has been proposed recently by Johansson in [31], and that series with integer, rational or floating point coefficients can often be composed in quasi-linear time in suitable bit complexity models, as shown by Ritzmann [40]; see also [22].

Relationship between multi-point evaluation and modular composition

Multi-point evaluation and modular composition are instances of evaluation problems at points lying in different extensions of $\mathbb{A}$. The former case involves several points with coordinates in $\mathbb{A}$. The latter case implies one point in the extension $\mathbb{A}[x]/(h(x))$. In the next paragraphs we summarize known conversions between evaluation problems.

When $n = 1$, several algorithms are known for converting evaluations at any set of points to specific sets of points. For instance evaluating at roots of unity can be done fast thanks to the seminal FFT algorithm, so we usually build fast algorithms upon FFTs. Typically fast polynomial products are reduced to FFTs over synthetic roots of unity lying in suitable extensions of $\mathbb{A}$ by means of the Schönhage–Strassen algorithm. And since fast multi-point evaluation reduces to polynomial products, they thus reduce to FFTs. Such reductions to FFTs are omnipresent in computer algebra.

Let us still assume that $n = 1$. Let $f \in \mathbb{K}[x]_{<d}$, let $\alpha_1, \ldots, \alpha_d$ be given evaluation points in a field $\mathbb{K}$, and let $\beta_1, \ldots, \beta_d$ be pairwise distinct evaluation points in $\mathbb{K}$. Let $h(x) = (x-\beta_1)\cdots(x-\beta_d)$ and let $g \in \mathbb{K}[x]_{<d}$ be such that $g(\beta_i) = \alpha_i$ for $i = 1, \ldots, d$. Setting $\rho = f \circ g \bmod h$ we have $\rho(\beta_i) = f(\alpha_i)$. So the evaluations of f at $\alpha_1, \ldots, \alpha_d$ reduce to evaluations and interpolations in degree $d-1$ at the chosen points plus one modular composition. Conversely given a modulus h, one may benefit from factorizations of h to compose modulo h. We have studied this approach when h has factors with large multiplicities in [25], when it splits into linear factors over $\mathbb{C}$ in [26], and also when it factors over an algebraic extension of $\mathbb{K}$ in [27].

The key idea of Nüsken and Ziegler to speed up multi-point evaluation is a reduction to modular composition; then their aforementioned complexity bound follows from a variant of the Brent–Kung algorithm. Assume $n = 2$. In order to evaluate f at d points $\alpha_1, \ldots, \alpha_d$ they first compute $h(x) = (x-\alpha_{1,1})\cdots(x-\alpha_{d,1})$ and interpolate $g \in \mathbb{K}[x]$ such that $g(\alpha_{i,1}) = \alpha_{i,2}$ for $i = 1, \ldots, d$ (assuming the $\alpha_{i,1}$ being pairwise distinct, which is not restrictive). Then they compute $\rho(x) = f(x, g(x)) \bmod h(x)$ and deduce $f(\alpha_i)$ as $\rho(\alpha_{i,1})$.

Over finite fields, Kedlaya and Umans showed an equivalence between multi-point evaluation and modular composition. Using Kronecker segmentation, Theorem 3.1 from [34] reduces such a composition to multi-point evaluation for an increased number of variables. Kedlaya and Umans' reduction in the opposite direction is close to the one of Nüsken and Ziegler. Let $\beta_1, \ldots, \beta_N$ be pairwise distinct points in $\mathbb{K}$. For each $j = 1, \ldots, n$, they interpolate $g_j \in \mathbb{K}[x]_{<N}$ such that $g_j(\beta_i) = \alpha_{i,j}$ for $i = 1, \ldots, N$. Then they compute $\rho = f(g_1, \ldots, g_n) \bmod h$, so that $f(\alpha_i) = \rho(\beta_i)$.

1.2. Contributions

On machines with random access memory, arbitrary memory accesses admit a constant cost. This does not reflect the actual behavior of real computers, on which memory is organized into different levels, with efficient hardware support for copying contiguous blocks of memory from one level to another. In this paper, we opted for the standard Turing machine model with a finite number of tapes [38], which charges a "maximal penalty" for non contiguous memory accesses. This means in particular that complexity bounds established for this model are likely to hold for any more or less realistic alternative model. Our first contribution in the present paper is to show that Kedlaya and Umans' complexity bounds hold in the Turing machine model.

Our second contribution concerns sharper and more precise bit complexity bounds. For multi-point evaluation over $\mathbb{A} = \mathbb{Z}/r\,\mathbb{Z}$, we achieve softly linear time in the bit size of r and obtain more general explicit bounds in terms of n, N, the partial and total degrees of f, without the assumption $n = d^{o(1)}$. We also put into evidence the advantage of taking N much larger than the dense size of the support of f. In particular, we analyze the threshold for which the average cost per evaluation point stabilizes. This analysis turns out to be important for our refined bounds for univariate modular composition in section 6, but also for our new bit complexity bounds for multivariate modular composition, with the application to polynomial system solving in [28].

Let us now turn to multi-point evaluation over an extension ring $\mathbb{A} = (\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$, with θ monic of degree k. Kedlaya and Umans proposed a reduction to multi-point evaluation over $\mathbb{Z}/R\,\mathbb{Z}$, with R large, based on Kronecker substitution. In section 4, we propose an alternative approach, based on univariate polynomial evaluation, interpolation, and Chinese remaindering, to directly reduce to several compositions over suitable finite prime fields.

Section 7 adresses the special case when $\mathbb{A}$ is a field $\mathbb{K}$ of small positive characteristic p. We revisit the method proposed in [34, section 6], and make the complexity bound more explicit. Again we quantify the number of evaluation points from which the average cost per point stabilizes, and we deduce a sharpened complexity bound for modular composition.

2. COMPLEXITY MODEL AND BASIC OPERATIONS

We consider Turing machines with sufficiently many tapes. In fact seven tapes are usually sufficient to implement all useful complexity bounds for the elementary operations on polynomials, series and matrices involved in the present paper (standard algorithms may be found in [42]). The number of symbols used by the machine is not of the utmost importance, since it only impacts complexity bounds by constant factors. In the sequel, Turing machines will always have two symbols "0" and "1", as well as a few specific additional ones dedicated to data representation.

Some algebraic structures involve a natural bit size for representing their elements (e.g. modular integers, finite fields); others involve a variable size (e.g. arbitrarily large integers, arrays, polynomials). In both cases, elements are seen as sequences of symbols on tapes ended by a specific symbol, written "#" in the sequel. Because heads of the machine can just move one cell left or right at time, algorithms must take care of consuming data in the most contiguous way as possible. In particular, we notice that loop counters cannot be used for free, in general. In this section we gather standard data types and elementary operations needed in the next sections. We freely use well known complexity bounds for polynomials and matrices from [13] and refer to [42] for more details on Turing machine implementations.

Integers
We use binary representation for integers, so that $n \in \mathbb{N}$ has bit size $\mathrm{bs}\, n := \lceil \log_2 (n+1) \rceil$. A modular integer in $\mathbb{Z}/r\mathbb{Z}$ is represented by its natural representative in $\{0, \ldots, r-1\}$. Integers may be added in linear time. The expression $\mathsf{I}(n)$ will represent a nondecreasing cost function for multiplying two integers of bit sizes $\leqslant n$, which satisfies $\mathsf{I}(n_1)/n_1 \leqslant \mathsf{I}(n_2)/n_2$ for all $0 < n_1 \leqslant n_2$. At present time the best known complexity bound is $\mathsf{I}(n) = O(n \log n\, 4^{\log^* n}) = \tilde{O}(n)$, where $\log^* n = \min \{k \in \mathbb{N} : \log \overset{k\times}{\ldots} \log n \leqslant 1\}$; see [16, 17, 18] and historical references therein. The integer division in bit sizes $\leqslant n$ takes time $O(\mathsf{I}(n))$ (see Lemma 2.15 below for instance), and the extended gcd costs $O(\mathsf{I}(n) \log n)$ by [41]. Overall, all arithmetic operations in $\mathbb{Z}/r\mathbb{Z}$ take softly linear time.

Arrays
One dimensional arrays are sequences of elements ended with the symbol "#".

Example 2.1. The vector $(1,0,1) \in \mathbb{F}_2^3$ is stored as 1#0#1##.

For bidimensional arrays we use column-major representation. Precisely an array $(A_{i,j})_{1\leqslant i\leqslant r, 1\leqslant j\leqslant c}$ of size $r \times c$ (r rows and c columns), is stored as the vector of its columns, that is $((A_{1,1}, \ldots, A_{r,1}), (A_{1,2}, \ldots, A_{r,2}), \ldots, (A_{1,c}, \ldots, A_{r,c}))$. Such arrays are allowed to contain elements of different types and sizes.

Example 2.2. The matrix $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ over $\mathbb{F}_2$ is stored as 1#0##1#0###.

In the Turing machine model, it is not known how to perform transpositions of bidimensional arrays in linear time. The following lemma shows how to do transpositions with a logarithmic overhead. The special case when all entries have the same bit size was treated before in [5, appendix]. Notice that transpositions do not preserve the total bit size for non square matrices, due to changes in the number of "#" symbols.

LEMMA 2.3. *Let $A = (A_{i,j})$ be an $r \times c$ matrix. Let $b_{i,j}$ denote the size of $A_{i,j}$ for all i, j, and define $B := \sum_{i,j} b_{i,j}$. Then A can be transposed in time $O((B + rc) \log \min(r, c))$.*

Proof. We first handle the case $r \leqslant c$ using the following "divide and conquer" algorithm. If $r = 1$, then the array $(A_{1,1}, \ldots, A_{1,c})$ is encoded as $A_{1,1}\#\#A_{1,2}\#\#\ldots A_{1,c}\#\#\#$ and we write its transpose $A_{1,1}\#A_{1,2}\#\ldots A_{1,c}\#\#\#$ on the output tape using one linear traversal. Otherwise, we split A into two matrices H and L on separate tapes, where H is made of the $r_1 := \lfloor r/2 \rfloor$ first rows of A, and L of the $r_2 := \lceil r/2 \rceil$ remaining ones. We recursively transpose H and L and glue the results together on the output tape.

Clearly, the case when $r = 1$ can be handled in time $O(B + rc)$, as well as the algorithm for splitting A into H and L, and the algorithm for gluing the transposes of H and L together into the transpose of A. Let C be a constant such that each of these algorithms takes time at most $C(B + rc)$. Let $B_1 := \sum_{i\leqslant r_1, j} b_{i,j}$ and $B_2 := \sum_{r_1 < i, j} b_{i,j}$. Let us show by induction over r that the transposition algorithm takes time $C(B + rc)(4 \log_2 r + 1)$. This is clear for $r = 1$. For $r = r_1 + r_2 > 1$, the computation time is bounded by

$$\begin{aligned} & C(B_1 + r_1 c)(4 \log_2 r_1 + 1) + C(B_2 + r_2 c)(4 \log_2 r_2 + 1) + 2C(B + rc) \\ \leqslant\ & C(B + rc)(4 \log_2 r_2 + 3) \\ \leqslant\ & C(B + rc)(4 \log_2 r + 1). \end{aligned}$$

The case when $r \geqslant c$ is handled in an essentially similar way, by reverting the steps of the algorithm: if $c = 1$, then $A_{1,1}\#A_{2,1}\#\ldots A_{r,1}\#\#\#$ is rewritten into $A_{1,1}\#\#A_{2,1}\#\#\ldots A_{r,1}\#\#\#$ using one linear pass. If $c > 1$, then we recursively apply the algorithm on the first $\lfloor c/2 \rfloor$ and the last $\lceil c/2 \rceil$ columns, and merge the results in a linear pass. The entire computation can be done in time $O((B + rc) \log c))$, by a similar complexity analysis as above. □

Univariate polynomials
For univariate polynomials we use dense representation, which means that a polynomial of degree d is stored as the vector of its $d+1$ coefficients from degrees 0 to d. Additions and subtractions take linear time in d. Let $\mathsf{M}(d)$ denote a cost function that yields an upper bound for the number of operations in $\mathbb{A}$ needed to multiply two polynomials in $\mathbb{A}[x]_{<d}$. For a general ring $\mathbb{A}$ one may take $\mathsf{M}(d) = O(d \log d \log\log d)$ thanks to [10]. For finite fields better bounds exist, and we write $\mathsf{M}_{\mathbb{F}_q}(d)$ for the time taken by a Turing machine to multiply two polynomials in $\mathbb{F}_q[x]_{<d}$.

In what follows, any finite field $\mathbb{F}_q$ with $q = p^k$ and p prime is always assumed to be given as $(\mathbb{Z}/p\,\mathbb{Z})[z]/(\theta(z))$ with θ monic and irreducible of degree k. Elements of $\mathbb{F}_q$ are stored as their natural representatives in $(\mathbb{Z}/p\,\mathbb{Z})[z]_{<k}$. Additions and subtractions in $\mathbb{F}_q$ take linear time, one product takes time $O(\mathsf{M}_{\mathbb{F}_p}(k))$ and one inversion $O(\mathsf{M}_{\mathbb{F}_p}(k)\log k + \mathsf{I}(\log p)\log\log p)$: see [13, part II], for instance. In [15, 19], it was shown that $\mathsf{M}_{\mathbb{F}_q}(d) = O(d\log q \log(d\log q)\, 4^{\log^*(d\log q)})$.

Multivariate polynomials
For a polynomial $f \in \mathbb{A}[x_1, \ldots, x_n]$ in a given number of variables n, we use the *recursive dense representation*, by viewing f as an element of $\mathbb{A}[x_1][x_2]\cdots[x_n]$. In particular, f admits the same representation as its expansion $f = f_0 + \cdots + f_{\ell_n - 1}\, x^{\ell_n - 1} \in \mathbb{A}[x_1, \ldots, x_{n-1}][x_n]$ as a univariate polynomial in x_n. In our algorithms, the number of variables n is not part of the representation of f, so it must be supplied as a separate parameter.

Example 2.4. The univariate polynomial $f(x) = c_0 + c_1 x + \cdots + c_d x^d$ of degree d is represented by $c_0\#c_1\#\ldots c_d\#\#$. The bivariate polynomial $f(x_1, x_2) = c_{0,0} + c_{0,1} x_1 + (c_{1,0} + c_{1,1} x_1) x_2$ is represented by $c_{0,0}\#c_{0,1}\#\#c_{1,0}\#c_{1,1}\#\#\#$.

The *support* $\operatorname{supp} f$ of $f \in \mathbb{A}[x_1, \ldots, x_n]$ is defined as the set of monomials with nonzero coefficients and we write $|\operatorname{supp} f|$ for its cardinality. Assuming that, apart from the mandatory trailing "#" symbol, the representations of coefficients in $\mathbb{A}$ do not involve the "#" symbol (this can always be achieved through suitable renaming $\# \rightsquigarrow \#_{\mathbb{A}}$), we denote the number of "#" symbols involved in the representation of f by $|f|_\#$. We notice that $|\operatorname{supp} f| \leqslant |f|_\#$.

LEMMA 2.5. *Let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$. Then $|f|_\# \leqslant \sum_{i=1}^{n} \ell_i \cdots \ell_n + 1 \leqslant n\,\pi + 1$, where $\pi := \ell_1 \cdots \ell_n \geqslant |\operatorname{supp} f|$.*

Proof. This follows by an easy induction over n: for $n = 0$, we have nothing to do. If $n > 0$ and $f = f_0 + \cdots + f_{\ell_n - 1} x_n^{\ell_n - 1}$, then we get

$$\begin{aligned} |f|_\# &\leqslant \sum_{k < \ell_n} |f_k|_\# + 1 \\ &\leqslant \ell_n \left(\sum_{i=1}^{n-1} \ell_i \cdots \ell_{n-1} + 1 \right) + 1 \\ &\leqslant \sum_{i=1}^{n-1} \ell_i \cdots \ell_n + \ell_n + 1 \;=\; \sum_{i=1}^{n} \ell_i \cdots \ell_n + 1, \end{aligned}$$

which concludes the proof. □

If all the ℓ_i equal 1, then f is the constant polynomial $c \in \mathbb{A}$ and its representation is $c\#\ldots\#$ with $n+1$ symbols "#". If $\ell_i \geqslant 2$ for all i, then the number of # becomes $O(\pi)$.

LEMMA 2.6. *Let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be a nonzero polynomial of total degree $\leqslant d$. Then $|f|_\# \leqslant \sum_{i=0}^{n} \binom{d+i}{i} \leqslant n\rho + 1$, where $\rho := \binom{d+n}{n} \geqslant |\mathrm{supp}\ f|$.*

Proof. We use a similar induction as in the proof of Lemma 2.5:

$$\begin{aligned} |f|_\# &\leqslant \sum_{k<\ell_n} |f_k|_\# + 1 \leqslant \sum_{k\leqslant d} \sum_{i=0}^{n-1} \binom{k+i}{i} + 1 \\ &= \sum_{i=0}^{n-1} \sum_{k\leqslant d} \binom{k+i}{i} + 1 = \sum_{i=0}^{n} \binom{d+i}{i}. \end{aligned}$$

□

If $d = 0$, then we already observed that $|f|_\# = n + 1$. If $d = 1$, then $|f|_\# \leqslant \sum_{i=0}^{n} \binom{i+1}{i} \sim n^2/2$. For the remainder of this subsection, we assume that the size of the elements in $\mathbb{A}$ is bounded by a constant $\mathsf{s}_\mathbb{A}$. In particular, the total size of a multivariate polynomial f is bounded by $|f|_\# + |\mathrm{supp}\ f|\,\mathsf{s}_\mathbb{A}$.

LEMMA 2.7. *The partial degree bounds $\ell_1 = 1 + \deg_{x_1} f$, ..., $\ell_n = 1 + \deg_{x_n} f$ of a nonzero polynomial $f \in \mathbb{A}[x_1, \ldots, x_n]$ can be computed in time $O(|f|_\# (n + \log \pi) + |\mathrm{supp}\ f|\,\mathsf{s}_\mathbb{A})$, where $\pi := \ell_1 \cdots \ell_n$.*

Proof. Recall that both $n \in \mathbb{N}$ and $f \in \mathbb{A}[x_1, \ldots, x_n]$ are considered to be the inputs. We use the following recursive algorithm: if $n = 0$, then we have nothing to do. If $n > 0$, then we write $f = f_0 + \cdots + f_{\ell_n - 1} x_n^{\ell_n - 1}$ and recursively compute partial degree bounds $\deg_{x_i} f_k < \ell_{k,i}$ for the coefficients. We next return $\ell_1 = \max_k \ell_{k,1}$, ..., $\ell_{n-1} = \max_k \ell_{k,n-1}$, ℓ_n. The lemma clearly holds for $n = 1$. By induction, the recursive computations can be done in time

$$O((|f|_\# - \ell_n)\,(n - 1 + \log\,(\ell_1 \cdots \ell_{n-1})) + |\mathrm{supp}\ f|\,\mathsf{s}_\mathbb{A} + \ell_n).$$

The computation of the maxima can be done using one linear pass in time

$$O(\ell_n\,(\log \ell_1 + 1 + \cdots + \log \ell_{n-1} + 1)) = O(\ell_n\,(n - 1 + \log\,(\ell_1 \cdots \ell_{n-1}))).$$

Determining ℓ_n requires an additional time $O(\ell_n \log\ \ell_n)$. Altogether, the computation takes time $O(|f|_\# (n + \log(\ell_1 \cdots \ell_n)) + |\mathrm{supp}\ f|\,\mathsf{s}_\mathbb{A})$. □

LEMMA 2.8. *The total degree $d = \deg f$ of a polynomial $f \in \mathbb{A}[x_1, \ldots, x_n]$ can be computed in time $O(n\,|f|_\# \log(d + 3) + |\mathrm{supp}\ f|\,\mathsf{s}_\mathbb{A})$, under the convention that $\deg 0 := -1$.*

Proof. We use the following recursive algorithm: if $n = 0$, then we have nothing to do. If $n > 0$, then we write $f = f_0 + \cdots + f_{\ell_n - 1} x_n^{\ell_n - 1}$ and recursively compute total degree bounds $\deg f_k \leqslant d_k$ for the coefficients. We next return $d = -1$ if $f = 0$ and $d = \max\{d_k + k : k < \ell_n, d_k \geqslant 0\}$ otherwise. The complexity bound follows using a similar induction argument as in the previous lemma. □

Evaluation and multi-remaindering

In the following paragraphs we recall the costs of integer multi-remaindering and univariate multi-point evaluation together with the inverse problems. The underlying techniques are well known. They are recalled in the context of Turing machines for convenience.

LEMMA 2.9. *Let $r \geqslant 2$ be an integer, let $a = (a_1, \ldots, a_m)$ be an array of integers in $\{0, \ldots, r-1\}$, and let $p_1, \ldots, p_s \geqslant 2$ be integers such that $B := p_1 \cdots p_s \geqslant r$. Then $a \bmod p_1, \ldots, a \bmod p_s$ may be computed in time $m\,\tilde{O}(\log B)$.*

Proof. We first compute the bidimensional array

$$\begin{matrix} a_1 \operatorname{rem} p_1 & \dots & a_m \operatorname{rem} p_1 \\ \vdots & & \vdots \\ a_1 \operatorname{rem} p_s & \dots & a_m \operatorname{rem} p_s \end{matrix}$$

in time $m\,\tilde{O}(\log B)$ by using fast multi-remaindering [13, chapter 10]. Then we appeal to Lemma 2.3 to obtain $a \operatorname{rem} p_1, \dots, a \operatorname{rem} p_s$ in time $O(m \log B \log s) = m\,\tilde{O}(\log B)$. □

LEMMA 2.10. *Let $r \geqslant 2$ be an integer, let $f \in \mathbb{Z}[x_1, \dots, x_n]$ have its coefficients in $\{0, \dots, r-1\}$, and let $p_1, \dots, p_s \geqslant 2$ be integers such that $B := p_1 \cdots p_s \geqslant r$. Then $f \operatorname{rem} p_1, \dots, f \operatorname{rem} p_s$ can be computed in time $\rho\,\tilde{O}(\log B) + O(s\,|f|_{\#})$, where $\rho := |\operatorname{supp} f|$.*

Proof. We first extract the vector v of all the nonzero coefficients of f in time $O(\rho \log r + |f|_{\#})$. We use the latter lemma on v, which incurs time $\rho\,\tilde{O}(\log B)$. Then we recover $f \operatorname{rem} p_i$ from $v \operatorname{rem} p_i$ and f, for $i = 1, \dots, s$, in time $O(\rho \log p_i + |f|_{\#})$. □

LEMMA 2.11. *Let $p_1, \dots, p_s \geqslant 2$ be pairwise coprime integers, let $B := p_1 \cdots p_s$, and let $a_1 \in (\mathbb{Z}/p_1\mathbb{Z})^m, \dots, a_s \in (\mathbb{Z}/p_s\mathbb{Z})^m$. Then the unique vector $a \in \mathbb{Z}^m$ with entries in $\{0, \dots, B-1\}$ such that $a \equiv a_i \operatorname{rem} p_i$ for $i = 1, \dots, s$ can be computed in time $m\,\tilde{O}(\log B)$.*

Proof. First we transpose the bidimensional array $a_1, \dots, a_s$ of size $m \times s$ and then use Chinese remaindering m times. By Lemma 2.3 and [13, chapter 10], this can be done in time $O(m \log B \log s) + m\,\tilde{O}(\log B) = m\,\tilde{O}(\log B)$. □

LEMMA 2.12. *Let $r \geqslant 2$ be an integer and let $a = (a_1, \dots, a_m)$ be an array of polynomials in $(\mathbb{Z}/r\mathbb{Z})[x]_{<k}$. For any set of points $\xi_1, \dots, \xi_K$ we may compute $a_1(\xi_1), \dots, a_m(\xi_1), \dots, a_1(\xi_K), \dots, a_m(\xi_K)$ in time $m\,(k+K)\,(\tilde{O}(\log^2 k) + \min(\log m, \log K))\,\tilde{O}(\log r)$.*

Proof. We first compute the bidimensional array

$$\begin{matrix} a_1(\xi_1) & \cdots & a_m(\xi_1) \\ \vdots & & \vdots \\ a_1(\xi_K) & \cdots & a_m(\xi_K) \end{matrix}$$

in time $m\,(k+K)\,\tilde{O}(\log^2 k)\,\tilde{O}(\log r)$, using fast univariate multi-point evaluation [13, chapter 12]. We next transpose the array in time $O(m\,K \log r \min(\log m, \log K))$, using Lemma 2.3. □

LEMMA 2.13. *Let $r \geqslant 2$ be an integer, let $f \in (\mathbb{Z}/r\mathbb{Z})[z]_{<k}[x_1, \dots, x_n]$, and let $\zeta_1, \dots, \zeta_K$ be elements in $\mathbb{Z}/r\mathbb{Z}$. Then $f(\zeta_1, x_1, \dots, x_n), \dots, f(\zeta_K, x_1, \dots, x_n)$ can be computed in time*

$$\rho\,(k+K)\,(\tilde{O}(\log^2 k) + \min(\log \rho, \log K))\,\tilde{O}(\log r) + O(K\,|f|_{\#}),$$

where ρ is the cardinality of the support of f in the variables $x_1, \dots, x_n$.

Proof. We first extract the vector $v \in ((\mathbb{Z}/r\mathbb{Z})[z]_{<k})^{\rho}$ of all the nonzero coefficients of f in time $O(\rho\,k \log r + |f|_{\#})$ together with $f^{\#} \in \mathbb{F}_2[x_1, \dots, x_n]$ of the same support as f. We use the latter lemma on v, which incurs time

$$\rho\,(k+K)\,(\tilde{O}(\log^2 k) + \min(\log \rho, \log K))\,\tilde{O}(\log r).$$

Then for $i = 1, \dots, K$ we recover the evaluation of f at $z = \zeta_i$ from $v(\zeta_i)$ and $f^{\#}$ in time $O(\rho \log r + |f|_{\#})$. □

LEMMA 2.14. *Let r be a prime number, let $\zeta_1, \ldots, \zeta_K$ be distinct elements of $\mathbb{Z}/r\mathbb{Z}$ and let $b_1, \ldots, b_K$ be vectors in $(\mathbb{Z}/r\mathbb{Z})^m$. Then the unique vector of polynomials $a = (a_1, \ldots, a_m) \in ((\mathbb{Z}/r\mathbb{Z})[z]_{<K})^m$ with $a_j(\zeta_i) = b_i$ for $i = 1, \ldots, K$ and $j = 1, \ldots, m$ can be computed in time $m\tilde{O}(K \log r)$.*

Proof. We first transpose the $m \times K$ array made of $b_1, \ldots, b_m$ to obtain

$$\begin{matrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & & \vdots \\ b_{K,1} & \cdots & b_{K,m} \end{matrix}$$

in time $O(mK \log r \log K)$ by Lemma 2.3. We next obtain the result through m interpolations, in time $m\tilde{O}(K \log r)$ by [13, chapter 12]. □

Lexicographic orders

We will have to use the lexicographic order on $\mathbb{N}^n$, written $<_{\text{lex}}$, defined by

$$\alpha <_{\text{lex}} \beta \Leftrightarrow (\exists j \in \{1, \ldots, n\},\ \alpha_n = \beta_n \wedge \cdots \wedge \alpha_{j+1} = \beta_{j+1} \wedge \alpha_j < \beta_j).$$

Notice that in the dense polynomial representation used here, coefficients are stored accordingly to the lexicographic order on the exponent vectors; this corresponds to the usual lexicographic monomial order induced by $x_n >_{\text{lex}} x_{n-1} >_{\text{lex}} \cdots >_{\text{lex}} x_1$.

Fixed point numbers

We use fixed point representation to approximate real numbers. The number $a = \sum_{-n \leqslant i \leqslant m} a_i\, 2^i$, with $a_i \in \{0, 1\}$ and $m, n \geqslant 0$, is represented by $a_m a_{m-1} \cdots a_0 . a_{-1} a_{-2} \cdots a_{-n}\#$, where "." is a specific symbol of the machine. A negative number starts with the symbol "-". The bit size of a is $O(m + n)$. The integer n is called the (absolute) precision. Additions, subtractions and reducing the precision take linear time. The product of two fixed point numbers of bit size l takes time $O(\mathsf{I}(l)) = \tilde{O}(l)$. We gather well known results from [8].

LEMMA 2.15. *Let $a_1 := \sum_{-n_1 \leqslant i \leqslant m_1} a_{1,i}\, 2^i$ and $a_2 := \sum_{-n_2 \leqslant i \leqslant m_2} a_{2,i}\, 2^i \neq 0$. An approximation χ of a_1/a_2 to precision $\tau \geqslant 0$ such that $\left|\frac{a_1}{a_2} - \chi\right| < 2^{-\tau}$ can be computed in time $O(\mathsf{I}(m_1 + n_1 + m_2 + n_2 + \tau))$.*

Proof. Without loss of generality we may assume $\tau \geqslant \max(n_1 - n_2, 0)$. We perform the following long integer division in time $O(\mathsf{I}(m_1 + n_1 + m_2 + n_2 + \tau))$:

$$2^{n_2+\tau} a_1 = q\, 2^{n_2} a_2 + r, \text{ with } 0 \leqslant r < 2^{n_2} a_2.$$

Then we have

$$\frac{a_1}{a_2} = 2^{-\tau} q + 2^{-n_2-t} \frac{r}{a_2}, \text{ with } 0 \leqslant 2^{-n_2-\tau} \frac{r}{a_2} < 2^{-\tau}.$$

Consequently we may take $\chi = 2^{-\tau} q$. □

LEMMA 2.16. *Consider a fixed point number $a := \sum_{-n \leqslant i \leqslant m} a_i 2^i > 0$. An approximation β of $\log a$ with $|\log a - \beta| < 2^{-\tau}$ for $\tau \geqslant 0$ can be computed in time $O(\mathsf{I}(m + n + \tau) \log(m + n + \tau))$.*

Proof. We write $a = 2^k b$ with $k \in \mathbb{Z}$ and $1 < b \leqslant 2$ so $\log a = k \log 2 + \log b$. By using [8, Theorem 6.1], $\log 2$ and $\log b$ may be computed to precisions $\lceil \log_2(\max(|k|, 1)) \rceil + \tau + 1$ and $\tau + 1$ respectively in time $O(\mathsf{I}(m + n + \tau) \log(m + n + \tau))$. Let us write λ and β the respective approximations, which satisfy

$$|\log 2 - \lambda| < 2^{-(\lceil \log_2(\max(|k|,1)) \rceil + \tau + 1)} = 2^{-(\tau+1)} / (\max(|k|, 1))$$

and $|\log b - \beta| < 2^{-(\tau+1)}$. Then $|\log a - k\lambda - \beta| < 2^{-\tau}$. □

LEMMA 2.17. *Consider a fixed point number* $a := \sum_{-n \leqslant i \leqslant m} a_i 2^i > 0$. *An approximation* β *of* $\sqrt{a}$ *to precision* $\tau \geqslant 0$ *such that* $|\sqrt{a} - \beta| < 2^{-\tau}$ *may be computed in time* $O(\mathsf{I}(m+n+\tau)\log(m+n+\tau))$.

Proof. We write $a = 2^k\, b$ with $k \in 2\,\mathbb{Z}$ and $1 < b \leqslant 4$, so we have $\sqrt{a} = 2^{k/2}\,\sqrt{b}$. By using [8, Lemma 2.3], $\sqrt{b}$ may be approximated to precision $k/2 + \tau + 1$ in time $O(\mathsf{I}(m+n+\tau)\log(m+n+\tau))$. □

3. FAST MULTI-POINT EVALUATION

In this section r is an integer with $r \geqslant 2$ and f is a polynomial in $(\mathbb{Z}/r\,\mathbb{Z})[x_1, \ldots, x_n]$ with partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$, and total degree $\leqslant d$. We assume that f is given in recursive dense representation for $(\mathbb{Z}/r\,\mathbb{Z})[x_1][x_2]\cdots[x_n]$, as described in the previous section, so the size of its representation is $O(\pi \log r + |f|_{\#})$ where $\pi := \ell_1 \cdots \ell_n$ and $|f|_{\#}$ represents the number of # in the representation of f. Throughout the section we assume that $\ell_i \geqslant 1$ for all i, that $d \geqslant 1$ and that the bounds on partial and total degrees are naturally correlated as follows:

$$\bar{\ell} := \max(\ell_1, \ldots, \ell_n) \leqslant d+1 \text{ and } d+n \leqslant \ell_1 + \cdots + \ell_n. \tag{3.1}$$

We wish to evaluate f at N points in $(\mathbb{Z}/r\,\mathbb{Z})^n$, written $\alpha_1, \ldots, \alpha_N$.

3.1. Overview of the multi-modular approach

In order to evaluate f at a point $\alpha \in (\mathbb{Z}/r\,\mathbb{Z})^n$ the initial idea consists in performing the evaluation over $\mathbb{Z}$, that is to say by discarding the modulus r. We write $\bar{f} \in \mathbb{Z}[x]$ for the natural preimage of f with coefficients in $\{0, \ldots, r-1\}$, and $\bar{\alpha} \in \mathbb{Z}^n$ for the natural preimage of α with entries in $\{0, \ldots, r-1\}$. In order to compute $\bar{f}(\bar{\alpha})$ we construct an *ad hoc* sequence of primes $p_1, \ldots, p_s$ such that $p_1 \cdots p_s > \bar{f}(\bar{\alpha})$. In this way, $\bar{f}(\bar{\alpha})$ may be recovered by Chinese remaindering from $\bar{f}(\bar{\alpha}) \operatorname{rem} p_1, \ldots, \bar{f}(\bar{\alpha}) \operatorname{rem} p_s$.

Minimizing the bit size of $p_1 \cdots p_s$ is of the utmost importance for efficiency. For this purpose we introduce the following quantity σ which bounds the cardinality of the support of f both in terms of the partial and total degrees:

$$\sigma := \min\left(\pi, \binom{d+n}{n}\right).$$

On the other hand the quantity

$$\varphi := \min\left(\ell_1 + \cdots + \ell_n, d+1+\left\lceil \frac{\log\binom{d+n-1}{n-1}}{\log 2}\right\rceil\right) \in \mathbb{N} \tag{3.2}$$

is used as a bound for $\log_r \bar{f}(\bar{\alpha})$. It satisfies the following inequality to be used several times in the proofs when $n \geqslant 2$:

$$\varphi \geqslant \min\left(d+n, d+1+\frac{\log(n-1)}{\log 2}\right) \geqslant d+1 \geqslant \bar{\ell}. \tag{3.3}$$

LEMMA 3.1. *Let* $\bar{f} \in \mathbb{Z}[x_1, \ldots, x_n]$ *have coefficients in* $\{0, \ldots, r-1\}$ *and let* $\bar{\alpha} \in \{0, \ldots, r-1\}^n$. *Then*

$$0 \leqslant \bar{f}(\bar{\alpha}) < r^{\varphi}.$$

Proof. We first prove that

$$\bar{f}(\bar{\alpha}) < r^{\ell_1 + \cdots + \ell_n}. \tag{3.4}$$

This inequality trivially holds when $n=1$. Suppose by induction that it holds up to $n-1$ variables. Then $\bar{f}(\bar{\alpha}) < \sum_{j=0}^{\ell_n-1} r^{\ell_1\cdots\ell_{n-1}} (r-1)^j < r^{\ell_1+\cdots+\ell_n}$ again holds for n variables.

On the other hand we have

$$\begin{aligned}
\bar{f}(\bar{\alpha}) &\leqslant \sum_{j=0}^{d} \binom{j+n-1}{n-1} (r-1)^{j+1} \\
&\leqslant \sum_{j=0}^{d} \binom{j+n-1}{n-1} (r-1)\, r^j \\
&\leqslant \binom{d+n-1}{n-1} \sum_{j=0}^{d} (r-1)\, r^j \;<\; \binom{d+n-1}{n-1} r^{d+1}. \qquad (3.5)
\end{aligned}$$

The conclusion simply follows from the assumption $r \geqslant 2$. □

Given $f \in (\mathbb{Z}/r\,\mathbb{Z})[x_1,\ldots,x_n]$ and N evaluation points $\alpha_1,\ldots,\alpha_N$ in $(\mathbb{Z}/r\,\mathbb{Z})^n$, the multi-point evaluation algorithm of this section works as follows:

1. If r is "sufficiently small", then we evaluate f at all points in $(\mathbb{Z}/r\,\mathbb{Z})^n$ and read off the needed values. This task is detailed in the next subsection.
2. Otherwise we compute prime numbers $p_1,\ldots,p_s$ such that $p_1\cdots p_s \geqslant r^\varphi$. This is adressed in sections 3.3 and 3.4.
3. We evaluate $\bar{f}$ at all $\bar{\alpha}_1,\ldots,\bar{\alpha}_N$ modulo p_i for $i=1,\ldots,s$ by calling the algorithm recursively.
4. We reconstruct the values of $\bar{f}$ at all $\bar{\alpha}_1,\ldots,\bar{\alpha}_N$ by Chinese remaindering and perform the final divisions by r.

We will be able to take all the p_i of the order of $\varphi \log r$. Therefore, the bit size of the modulus when entering the first recursive call is of the order $\varphi \log r$, then $\varphi \log(\varphi \log r)$ at depth 2, then $\varphi \log(\varphi \log(\varphi \log r))$ at depth 3, etc. The total bit size of all recursive problems to be solved at depth t turns out to grow with φ^t. In section 3.5 we study the complexity of the algorithm in terms of the depth t. Section 3.6 is devoted to finding a suitable value for t to end the recursive calls. Roughly speaking, the property "sufficiently small" from step 1 becomes "$\log r$ is of the order $\varphi \log \varphi$", so the time spent in the exhaustive evaluation of step 1 is of the order $(\varphi \log \varphi)^n$.

3.2. Exhaustive evaluation

We begin with studying the base case of the multi-modular algorithm, i.e. the exhaustive evaluation at all points of $(\mathbb{Z}/r\,\mathbb{Z})^n$. We recall that this algorithm is used for sufficiently small values of r. We regard the evaluation of $f \in (\mathbb{Z}/r\,\mathbb{Z})[x_1, \ldots, x_n]$ at all points in $(\mathbb{Z}/r\,\mathbb{Z})^n$ as a vector in $(\mathbb{Z}/r\,\mathbb{Z})^{r^n}$.

LEMMA 3.2. *Let $f \in (\mathbb{Z}/r\,\mathbb{Z})[x_1,\ldots,x_n]$ be of partial degree $<\ell_i$ in x_i for $i=1,\ldots,n$. All the values of f at $(\mathbb{Z}/r\,\mathbb{Z})^n$ can be computed in time*

$$n\,(\bar{\ell}^n + r^n)\,\tilde{O}(\log^2 \bar{\ell} \log r).$$

Proof. Detecting if f is the constant polynomial c takes time $O(n + \log r)$. If so, then it suffices to copy c onto the destination tapes r^n times. This costs $\tilde{O}(n \log r) + O(r^n \log r)$. From now we assume that f is not a constant, whence $n \geqslant 1$, $d \geqslant 1$ and $\bar{\ell} \geqslant 2$.

We interpret $f \in (\mathbb{Z}/r\mathbb{Z})[x_1,\ldots,x_n]$ as a univariate polynomial $f = f_0 + \cdots + f_{\ell_n-1} x_n^{\ell_n-1}$ and recursively evaluate the coefficients f_0, ..., f_{ℓ_n-1} at all points in $(\mathbb{Z}/r\mathbb{Z})^{n-1}$. After one $r^{n-1} \times \ell_n$ matrix transposition of cost $O(r^{n-1}\,\bar{\ell}\log\bar{\ell}\log r)$, this yields a vector of r^{n-1} univariate polynomials

$$f(a_1,\ldots,a_{n-1},x_n) = f_0(a_1,\ldots,a_{n-1}) + \cdots + f_{\ell_n-1}(a_1,\ldots,a_{n-1})\,x_n^{\ell_n-1} \in (\mathbb{Z}/r\mathbb{Z})[x_n]_{<\ell_n},$$

where $(a_1,\ldots,a_{n-1})$ ranges over $(\mathbb{Z}/r\mathbb{Z})^{n-1}$. Using r^{n-1} multi-point evaluations of these polynomials at all $a_n \in \{0,\ldots,r-1\}$ of cost $r^{n-1}\max(r,\bar{\ell})\,\tilde{O}(\log^2\bar{\ell}\log r)$, we finally obtain the vector of all $f(a_1,\ldots,a_n)$ with $(a_1,\ldots,a_n) \in (\mathbb{Z}/r\mathbb{Z})^n$. Denoting by $\mathsf{T}(n,\bar{\ell})$ the cost of the algorithm, we thus obtain

$$\mathsf{T}(n,\bar{\ell}) \leqslant \bar{\ell}\,\mathsf{T}(n-1,\bar{\ell}) + r^{n-1}\max(r,\bar{\ell})\,\tilde{O}(\log^2\bar{\ell}\log r).$$

By induction over n, it follows that

$$\mathsf{T}(n,\bar{\ell}) \leqslant n\max(r,\bar{\ell})^n\,\tilde{O}(\log^2\bar{\ell}\log r),$$

which implies the claimed bound. □

In order to evaluate f at a specific sequence $\alpha_1,\ldots,\alpha_N$ of points in $(\mathbb{Z}/r\mathbb{Z})^n$, we next wrap the latter lemma in the following algorithm that simply reads off the requested values once the exhaustive evaluation is done. This task is immediate in the RAM model, but induces a logarithmic overhead on Turing machines.

Algorithm 3.1
Input. $f \in (\mathbb{Z}/r\mathbb{Z})[x_1,\ldots,x_n]$; a sequence $\alpha_1,\ldots,\alpha_N$ of points in $(\mathbb{Z}/r\mathbb{Z})^n$.
Output. $f(\alpha_1),\ldots,f(\alpha_N)$.

1. Evaluate f at all points of $(\mathbb{Z}/r\mathbb{Z})^n$ sorted lexicographically.
2. For j from 1 to $\lceil N/r^n \rceil$ do
   a. Set $a := (j-1)\,r^n + 1$ and $b := \min(jr^n, N)$.
   b. Sort the vectors of pairs $(i-a,\alpha_i)_{a\leqslant i\leqslant b}$ into $(\tau(i-a),\alpha_{\tau(i-a)+a})_{a\leqslant i\leqslant b}$ accordingly to the second coordinate (where τ denotes a permutation of $\{0,\ldots,b-a\}$).
   c. Deduce the vector $(\tau(i-a), f(\alpha_{\tau(i-a)+a}))_{a\leqslant i\leqslant b}$ by using the exhaustive evaluation of step 1.
   d. Sort the latter vector according to the first coordinate in order to obtain $(i-a, f(\alpha_i))_{a\leqslant i\leqslant b}$.
3. Return $f(\alpha_1),\ldots,f(\alpha_N)$.

PROPOSITION 3.3. *Algorithm 3.1 is correct and takes time*

$$n\,(\bar{\ell}^n + r^n)\,\tilde{O}(\log^2\bar{\ell}\log r) + O(N n \log r \min(\log N, n\log r)).$$

Proof. The cost of step 1 is given in Lemma 3.2. In the Turing machine model the loop counter j and the bounds a and b do not need to be explicitly computed in step 2. Instead it suffices to allocate an array of r^n bits once on an auxiliary tape and use it to split the sequence of evaluation points into subsequences of r^n elements—except the last one which has cardinality $N \operatorname{rem} r^n$.

With this point of view in mind, each step 2.b and 2.d requires time

$$O((b-a)\,(n\log r)\log(b-a)),$$

so the sum over all the values of j is

$$O(N n \log r \min(\log N, n \log r)).$$

Each step 2.c takes time $O(r^n n \log r)$. The total cost of all steps 2.c is therefore bounded by $O((N + r^n) n \log r)$. □

3.3. The first Chebyshev function

Multi-modular techniques classically involve bounds on the first Chebyshev function

$$\vartheta(\beta) := \sum_{2 \leqslant p \leqslant \beta, p \text{prime}} \log p.$$

For complexity analyses, we rely on the estimate

$$\vartheta(\beta) = \beta + O\left(\frac{\beta}{\log \beta}\right) \tag{3.6}$$

due to Barkley Rosser and Schoenfeld in the sixties. More precisely they showed in [2, Theorem 31] that, for all $\beta \geqslant 569$,

$$\beta\left(1 - \frac{0.47}{\log \beta}\right) < \vartheta(\beta) < \beta\left(1 + \frac{0.47}{\log \beta}\right).$$

Sharper bounds may be found in [12] but they will not be necessary here. From now on γ represents a constant in $\mathbb{R}^>$ such that

$$|\vartheta(\beta) - \beta| \leqslant \gamma \frac{\beta}{\log \beta} \text{ for all } \beta > 1. \tag{3.7}$$

LEMMA 3.4. *There exists $\bar{\gamma} \in \mathbb{R}^>$ such that*

$$\beta = x + \bar{\gamma}\frac{x}{\log x} \implies \beta - 2\gamma\frac{\beta}{\log \beta} \geqslant x$$

holds for all $x \geqslant 2$.

Proof. For fixed $\bar{\gamma}$ and large x, one has

$$2\gamma\frac{\beta}{\log \beta} = 2\gamma\frac{x\left(1 + \frac{\bar{\gamma}}{\log x}\right)}{\log x + \log\left(1 + \frac{\bar{\gamma}}{\log x}\right)} \sim 2\gamma\frac{x}{\log x}.$$

Taking $\bar{\gamma} > 2\gamma$ and x sufficiently large (say $x > A$), it follows that

$$\beta - 2\gamma\frac{\beta}{\log \beta} = x + \bar{\gamma}\frac{x}{\log x} - 2\gamma\frac{\beta}{\log \beta} \geqslant x.$$

Then it suffices to further increase $\bar{\gamma}$ so that the implication also holds on the interval $[2, A]$. □

In the rest of this section the constant $\bar{\gamma}$ of the lemma will be used via the following function:

$$\mu(r) := \frac{1}{\log r} + \bar{\gamma}\frac{1 + \frac{1}{\log r}}{\log(r \log r)} = O\left(\frac{1}{\log r}\right). \tag{3.8}$$

It is also convenient to introduce the function

$$\mathsf{B}(r) \;:=\; (1+\mu(\varphi))\,\varphi\log r \tag{3.9}$$

that will bound the inflation of the modulus at successive recursive calls of our main algorithm. We will write $\mathsf{B}^{\circ t} := \mathsf{B}\circ\overset{t\times}{\ldots}\circ\mathsf{B}$ for the t-th iterate of this function.

3.4. Computing prime numbers

Generating prime numbers is a standard task. In the next paragraphs we include the needed results for completeness.

LEMMA 3.5. *Given a positive integer β, we may generate all the primes $\leqslant\beta$ in time $O(\beta\log^3\beta)$.*

Proof. We use the well known Eratosthenes sieve. On the same tape we generate all the integer multiples of 2 not larger than β, followed by all the multiples of 3 not larger than β, then all the multiples of 4 not larger than β, etc. The total number of multiples generated in this way is $O(\beta/2+\beta/3+\beta/4+\cdots+\beta/\beta)=O(\beta\log\beta)$. These multiples can all be generated in time $O(\beta\log^2\beta)$. Then we sort these integers in increasing order and remove duplicates in time $O(\beta\log^3\beta)$. The integers $\leqslant\beta$ which are not listed in this way are exactly the requested prime numbers, which can thus be deduced with further $O(\beta\log\beta)$ bit operations. □

The following algorithm computes consecutive prime numbers larger than a given integer $\underline{\beta}$, such that their product exceeds a given threshold $\bar{\beta}$.

Algorithm 3.2
Input. Positive integers $\underline{\beta}$ and $\bar{\beta}\geqslant 2$.
Output. The shortest sequence of consecutive primes $p_1,\ldots,p_s$ with $\underline{\beta}<p_1$ and $p_1\cdots p_s\geqslant\bar{\beta}$.

1. Set $\tau := 2$.
2. Compute all the consecutive prime numbers $\pi_1<\cdots<\pi_t$ less or equal to 2^τ.
3. If $\prod_{1\leqslant i\leqslant t,\ \underline{\beta}<\pi_i}\pi_i<\bar{\beta}$, then increase τ by 1 and go back to step 2.
4. Compute the first index a such that $\pi_a>\underline{\beta}$, and set $b_0:=a$ and $b_1:=t$.
5. If $\pi_a\geqslant\bar{\beta}$, then return π_a.
6. While $b_0<b_1$ do
   a. Compute $c:=\lfloor(b_0+b_1)/2\rfloor$.
   b. If $\pi_a\cdots\pi_c<\bar{\beta}$, then $b_0:=c+1$, else $b_1:=c$.
7. Return $\pi_a,\ldots,\pi_{b_1}$.

PROPOSITION 3.6. *Algorithm 3.2 is correct and takes time $\tilde{O}(\underline{\beta}+\log\bar{\beta})$. In addition we have $p_s=O(\underline{\beta}+\log\bar{\beta})$.*

Proof. After step 3 we have $\prod_{1\leqslant i\leqslant t,\ \underline{\beta}<\pi_i}\pi_i\geqslant\bar{\beta}$, so the rest of the algorithm corresponds to a binary search to obtain the minimal index b_1 such that $\prod_{a\leqslant i\leqslant b_1}\pi_i\geqslant\bar{\beta}$. During the "while" loop of step 6 we have $\prod_{a\leqslant i<b_0}\pi_i<\bar{\beta}$ and $\prod_{a\leqslant i\leqslant b_1}\pi_i\geqslant\bar{\beta}$. So when $b_0=b_1$ the minimal sequence is actually found. Since $b_0\leqslant c\leqslant b_1-1$ at each step of the "while" loop, either b_0 increases or b_1 decreases strictly. Consequently the algorithm returns the correct result.

We exit step 3 once $\vartheta(2^\tau) - \vartheta(\underline{\beta}) \geqslant \log \bar{\beta}$. Thanks to (3.6), this condition is met for $2^\tau = O(\underline{\beta} + \log \bar{\beta})$, after time $\tilde{O}(\underline{\beta} + \log \bar{\beta})$, by Lemma 3.5. The binary search also takes time $\tilde{O}(\underline{\beta} + \log \bar{\beta})$. □

3.5. The main recursion

We are now ready to present the multi-modular evaluation algorithm. The parameter t indicates the allowed depth for the recursive calls.

Algorithm 3.3

Input. $f \in (\mathbb{Z}/r\mathbb{Z})[x_1, \ldots, x_n]$; a sequence $\alpha_1, \ldots, \alpha_N$ of points in $(\mathbb{Z}/r\mathbb{Z})^n$; a nonnegative integer t; φ as defined in (3.2).

Output. $f(\alpha_1), \ldots, f(\alpha_N)$.

Assumption. $r > \varphi$.

1. If $t = 0$ or if f is a constant polynomial, then use Algorithm 3.1.
2. Call Algorithm 3.2 with φ and r^φ to compute the shortest sequence of consecutive prime numbers $p_1 < \cdots < p_s$ with $p_1 > \varphi$ and $p_1 \cdots p_s \geqslant r^\varphi$.
3. Let $\bar{f}$ be the canonical preimage of f in $\mathbb{Z}[x_1, \ldots, x_n]$, with coefficients in $\{0, \ldots, r-1\}$. Let $\bar{\alpha}_i$ represent the canonical preimage of α_i in $\{0, \ldots, r-1\}^n$, for $i = 1, \ldots, N$. Compute $\bar{f}$ rem p_j and $(\bar{\alpha}_i)_{1 \leqslant i \leqslant N}$ rem p_j, for $j = 1, \ldots, s$.
4. For j from 1 to s, compute $\bar{f}(\bar{\alpha}_1), \ldots, \bar{f}(\bar{\alpha}_N)$ modulo p_j by calling the algorithm recursively with $t - 1$.
5. Use Chinese remaindering to recover $\bar{f}(\bar{\alpha}_1), \ldots, \bar{f}(\bar{\alpha}_N)$.
6. Compute the remainders by r of these values to obtain and return $f(\alpha_1), \ldots, f(\alpha_N)$.

PROPOSITION 3.7. *Algorithm 3.3 is correct. Assume $n \geqslant 2$, that (3.1) holds, and that f has partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$ and total degree $\leqslant d$. When $t = 0$ the algorithm takes time*

$$(N + r^n)\, \tilde{O}(n\, (\log^2 d + \log N) \log r).$$

For a fixed value of $t \geqslant 1$ the algorithm takes time

$$(N + \mathsf{B}^{\circ t}(r)^n)\, \tilde{O}(n\, \varphi^t \log N \log r),$$

where B *is defined in* (3.9).

Proof. Lemma 3.1 ensures that the multi-modular approach works well, whence the correctness of the algorithm. From now assume $n \geqslant 2$ and $d \geqslant 1$. Inequality (3.1), combined to the definition of φ, implies $\varphi \geqslant 3$. If φ is bounded, then so are n, d, and σ. Consequently we may freely assume that r is sufficiently large in the cost analysis. From (3.7), for all $\beta \geqslant \varphi$ we obtain

$$|\vartheta(\beta) - \vartheta(\varphi) - (\beta - \varphi)| \leqslant \gamma \left(\frac{\beta}{\log \beta} + \frac{\varphi}{\log \varphi} \right).$$

Since $u \mapsto u / \log u$ is increasing for all $u \geqslant \exp 1$, we deduce

$$|\vartheta(\beta) - \vartheta(\varphi) - (\beta - \varphi)| \leqslant \frac{2\gamma\beta}{\log \beta}.$$

The condition $\vartheta(\beta) - \vartheta(\varphi) \geqslant \varphi \log r$ is therefore satisfied whenever

$$\beta - \frac{2\gamma\beta}{\log \beta} \geqslant \left(1 + \frac{1}{\log r}\right) \varphi \log r. \tag{3.10}$$

By Lemma 3.4, there exists $\bar{\gamma}$ such that (3.10) is satisfied when β is larger than

$$\begin{aligned}
&\varphi \log r \left(1 + \frac{1}{\log r} + \bar{\gamma} \frac{1 + \frac{1}{\log r}}{\log\left(\left(1 + \frac{1}{\log r} \right) \varphi \log r \right)} \right) \\
\leqslant\ &\varphi \log r \left(1 + \frac{1}{\log \varphi} + \bar{\gamma} \frac{1 + \frac{1}{\log \varphi}}{\log(\varphi \log \varphi)} \right) = (1 + \mu(\varphi))\, \varphi \log r.
\end{aligned}$$

It follows that

$$p_i \leqslant (1 + \mu(\varphi))\, \varphi \log r. \tag{3.11}$$

From $p_1 \cdots p_{s-1} < r^{\varphi}$, we deduce that

$$p_1 \cdots p_s < (1 + \mu(\varphi))\, r^{\varphi}\, \varphi \log r =: B, \quad \text{so} \quad \log B = O(\varphi \log r).$$

By Proposition 3.6, step 2 takes time $\tilde{O}(\varphi \log r)$. The number of "#" in the representation of f is $O((d+1)^n)$ by Lemma 2.5. Consequently the multi-remaindering of f in step 3 takes time $\sigma\, \tilde{O}(\log B) + O(s\,(d+1)^n) = (d+1)^n\, \tilde{O}(\log B)$ by Lemma 2.10. By Lemma 2.9 the multi-remaindering of $\bar{\alpha}_1, \ldots, \bar{\alpha}_N$ takes time $n N \tilde{O}(\log B)$. In total step 3 contributes to

$$(n N + (d+1)^n)\, \tilde{O}(\varphi \log r).$$

Step 5 costs $N \tilde{O}(\log B)$ by Lemma 2.11. The cost of step 6 is also dominated by $N \tilde{O}(\log B)$.

Let $\mathsf{C}_t(\ell, d, r, N)$ denote the cost function of Algorithm 3.3, for t being fixed. In other words, the constants hidden in the "O" of the rest of the proof do not depend on $\ell := (\ell_1, \ldots, \ell_n), d, r, N$ but on t. Since $\varphi \geqslant d+1$ by (3.3), we have $r > \bar{\ell}$. Proposition 3.3 yields the cost of step 1:

$$\begin{aligned}
\mathsf{C}_0(\ell, d, r, N) &= n\,(\bar{\ell}^n + r^n)\, \tilde{O}(\log^2 \bar{\ell} \log r) + n N O(\log N \log r) \\
&= n N O(\log N \log r) + n r^n \tilde{O}(\log^2 d \log r) \\
&= (N + r^n)\, \tilde{O}(n\,(\log^2 d + \log N) \log r).
\end{aligned} \tag{3.12}$$

By summing the costs of steps 2 to 6, we deduce that

$$\mathsf{C}_t(\ell, d, r, N) = (n N + (d+1)^n)\, \tilde{O}(\varphi \log r) + \sum_{i=1}^{s} \mathsf{C}_{t-1}(\ell, d, p_i, N). \tag{3.13}$$

Consequently, if $t = 1$, using the bounds (3.11), (3.12), and

$$\sum_{i=1}^{s} \tilde{O}(\log p_i) = \tilde{O}(\log B) = \tilde{O}(\varphi \log r),$$

the cost of Algorithm 3.3 simplifies as follows:

$$\begin{aligned}
\mathsf{C}_1(\ell, d, r, N) &= (n N + (d+1)^n)\, \tilde{O}(\varphi \log r) + \sum_{i=1}^{s} (N + p_i^n)\, \tilde{O}(n\,(\log^2 d + \log N) \log p_i) \\
&\leqslant (n N + (d+1)^n)\, \tilde{O}(\varphi \log r) \\
&\quad + (N + ((1 + \mu(\varphi))\, \varphi \log r)^n)\, \tilde{O}(n\,(\log^2 d + \log N)\, \varphi \log r).
\end{aligned}$$

Using (3.3) again gives $(d+1)^n \leqslant \varphi^n$, whence

$$\mathsf{C}_1(\ell, d, r, N) = (N + \mathsf{B}(r)^n)\, \tilde{O}(n\, \varphi \log N \log r).$$

Now assume by induction that $\mathsf{C}_{t-1}(\ell, d, r, N) = (N + \mathsf{B}^{\circ(t-1)}(r)^n)\, \tilde{O}(n\, \varphi^{t-1} \log N \log r)$ holds for some $t \geqslant 2$. Combining (3.11), (3.12) and (3.13) we deduce that:

$$\begin{aligned}
\mathsf{C}_t(\ell,d,r,N) &= (nN + (d+1)^n)\, \tilde{O}(\varphi \log r) + \sum_{i=1}^{s} (N + \mathsf{B}^{\circ(t-1)}(p_i)^n)\, \tilde{O}(n\, \varphi^{t-1} \log N \log p_i) \\
&= (nN + (d+1)^n)\, \tilde{O}(\varphi \log r) + (N + \mathsf{B}^{\circ t}(r)^n) \sum_{i=1}^{s} \tilde{O}(n\, \varphi^{t-1} \log N \log p_i) \\
&= (nN + (d+1)^n)\, \tilde{O}(\varphi \log r) + (N + \mathsf{B}^{\circ t}(r)^n)\, \tilde{O}(n\, \varphi^{t} \log N \log r).
\end{aligned}$$

We claim that $\mathsf{B}^{\circ t}(r) \geqslant (1 + \mu(\varphi))\ \varphi \log \varphi$ for all $t \geqslant 1$ which implies $\mathsf{C}_t(\ell, d, r, N) = (N + \mathsf{B}^{\circ t}(r)^n)\, \tilde{O}(n\, \varphi^t \log N \log r)$ and concludes the proof. The latter claim is proved by induction on t. It clearly holds for $t = 1$. Assume it holds for $t - 1$. Then, using $\varphi \geqslant 3$, we verify that

$$\begin{aligned}
\mathsf{B}^{\circ t}(r) = (1+\mu(\varphi))\, \varphi \log(\mathsf{B}^{\circ(t-1)}(r)) &\geqslant (1+\mu(\varphi))\, \varphi \log((1+\mu(\varphi))\, \varphi \log \varphi) \\
&\geqslant (1+\mu(\varphi))\, \varphi \log \varphi,
\end{aligned}$$

which concludes the proof of the claim. □

3.6. The main complexity bound

In order to complete Algorithm 3.3, we still have to specify how to set the parameter t in terms of ℓ, d, r. It is natural to let t increase as a function of r. Yet we cannot take t arbitrarily large because the complexity in Proposition 3.7 involves a factor φ^t. The key idea here is to observe that, if r is very large, namely when $\sigma = O(\log \log r)$, then we may use the naive algorithm to evaluate f independently at each α_i. This leads to the following complexity bound.

LEMMA 3.8. *We may evaluate $f \in (\mathbb{Z}/r\mathbb{Z})[x_1, \ldots, x_n]$ at a point $(a_1, \ldots, a_n)$ in $(\mathbb{Z}/r\mathbb{Z})^n$ in time $|f|_\# \tilde{O}(\log r) = (n\sigma + 1)\, \tilde{O}(\log r)$.*

Proof. If $n = 0$, then f is constant and we just copy the input. If $n > 0$, then we expand $f = f_0 + \cdots + f_{\ell_n - 1}\, x_n^{\ell_n - 1}$ as a univariate polynomial in x_n and recursively evaluate $f_0, \ldots, f_{\ell_n - 1}$ at the point $(a_1, \ldots, a_{n-1})$. This yields a univariate polynomial $f(a_1, \ldots, a_{n-1}, x_n) = f_0(a_1, \ldots, a_{n-1}) + \cdots + f_{\ell_n - 1}(a_1, \ldots, a_{n-1})\, x_n^{\ell_n - 1} \in (\mathbb{Z}/r\mathbb{Z})[x_n]$ in x_n that we evaluate at a_n using Horner's method. Using induction over n, it is not hard to show that the algorithm essentially performs $O(|f|_\#)$ ring operations in $\mathbb{Z}/r\mathbb{Z}$, which can be done in time $|f|_\# \tilde{O}(\log r)$. We finally recall that $|f|_\# \leqslant n\sigma + 1$, by Lemmas 2.5 and 2.6. □

We are now ready to present the top level multi-point evaluation procedure. Recall that the bit size of an integer $n \in \mathbb{N}$ is given by $\operatorname{bs} n := \lceil \log_2 (n+1) \rceil$.

Algorithm 3.4

Input. $f \in (\mathbb{Z}/r\mathbb{Z})[x_1, \ldots, x_n]$; a sequence $\alpha_1, \ldots, \alpha_N$ of points in $(\mathbb{Z}/r\mathbb{Z})^n$.

Output. $f(\alpha_1), \ldots, f(\alpha_N)$.

1. If $n = 1$, then compute the output $f(\alpha_1), \ldots, f(\alpha_N)$ using the univariate multi-point evaluation algorithm.
2. Compute $\ell_1, \ldots, \ell_n, d, \sigma, \varphi$.
3. Compute the bit size $\operatorname{bs} r$ of r and then the bit size $\operatorname{bs}(\operatorname{bs} r)$ of $\operatorname{bs} r$.

If $\sigma \leqslant \mathrm{bs}(\mathrm{bs}\, r) + 3$, then compute the output $f(\alpha_1), \ldots, f(\alpha_N)$ using the naive algorithm.

4. Compute the bit size $\mathrm{bs}\, \varphi$ of φ.
   If $\varphi < 8$ or $4r \leqslant \varphi\, \mathrm{bs}\, \varphi$, then compute the output $f(\alpha_1), \ldots, f(\alpha_N)$ using Algorithm 3.1.
5. Compute the output $f(\alpha_1), \ldots, f(\alpha_N)$ using Algorithm 3.3 with parameter $t := 5$.

PROPOSITION 3.9. *Algorithm 3.4 is correct and takes time*

$$(1 + \varepsilon(\ell, d, r))^n (N + (\varphi \log \varphi)^n)\, \tilde{O}(n \varphi^5 \log N \log r),$$

where $\varepsilon(\ell, d, r)$ is a function which tends to 0 when $\max(n, \ell_1, \ldots, \ell_n, d, r)$ tends to infinity.

Proof. When we arrive at step 5 with $\varphi \geqslant 8$ and $4r > \varphi\, \mathrm{bs}\, \varphi$, the inequality $\mathrm{bs}\, \varphi \geqslant 4$ holds, whence $r > \varphi$. Consequently the assumption of Algorithm 3.3 is satisfied. This proves the correctness of the algorithm thanks to Propositions 3.3 and 3.7.

If $n = 1$, then multi-point evaluation costs

$$O(\lceil N/d \rceil\, \mathsf{M}_{\mathbb{Z}/r\mathbb{Z}}(d) \log d) = \lceil N/d \rceil\, \tilde{O}(\varphi \log r),$$

which is below the bound of the proposition. From now on we assume that $n \geqslant 2$. Recall that $\varphi \geqslant d + 1$ by (3.3).

The quantities $\ell_1, \ldots, \ell_n, d$ may be obtained in time

$$O(\sigma \log r + n\, (d+1)^n \log(d+1)) = \varphi^n\, \tilde{O}(n \log \varphi + \log r),$$

by combining Lemmas 2.5, 2.7 and 2.8.

By the subproduct tree technique, we may compute π in time $\tilde{O}(\log\ \pi) = \tilde{O}(n\, \log(d+1)) = \tilde{O}(n\, \log\ \varphi)$, and $\binom{d+n}{n}$ in time $\tilde{O}(n\, \log(d\, +\, n)) = \tilde{O}(n\, \log\ \varphi)$. The cost for summing $\ell_1 + \cdots + \ell_n$ is $\tilde{O}(n\, \log(d\, +\, n)) = \tilde{O}(n\, \log\ \varphi)$. We may also compute $\binom{d+n-1}{n-1}$ in time $\tilde{O}(n \log(d+n)) = \tilde{O}(n \log \varphi)$ and then easily deduce $\left\lceil \log \binom{d+n-1}{n-1} / \log 2 \right\rceil$ as the bit size of $\binom{d+n-1}{n-1} - 1$. Overall the cost of step 2 is negligible.

Let $\log_2$ denote the logarithm function in base 2. The bit size $\mathrm{bs}\, r = \lceil \log_2(r+1) \rceil$ of r and then $\mathrm{bs}(\mathrm{bs}\, r)$ may be obtained in time $\tilde{O}(\log r)$. We have $0 \leqslant \mathrm{bs}\, r - \log_2(r+1) < 1$ and $0 \leqslant \mathrm{bs}(\mathrm{bs}\, r) - \log_2(\mathrm{bs}\, r + 1) < 1$, whence

$$\begin{aligned} |\log_2 \log_2(r+1) - \mathrm{bs}(\mathrm{bs}\, r)| &\leqslant |\log_2(\mathrm{bs}\, r + 1) - \log_2 \log_2(r+1)| + |\mathrm{bs}(\mathrm{bs}\, r) - \log_2(\mathrm{bs}\, r + 1)| \\ &\leqslant \left| \log_2 \left(\frac{\mathrm{bs}\, r + 1}{\log_2(r+1)} \right) \right| + 1 \\ &\leqslant \log_2 \left(1 + \frac{2}{\mathrm{bs}\, r - 1} \right) + 1 \ < \ 3. \end{aligned} \tag{3.14}$$

The naive evaluation in step 3 costs $n\, \sigma\, N\, \tilde{O}(\log r)$ by Lemma 3.8. So when $\sigma \leqslant \mathrm{bs}(\mathrm{bs}\, r) + 3$ this cost drops to $N \tilde{O}(n \log r)$.

From now we may assume that $\mathrm{bs}(\mathrm{bs}\, r) + 3 < \sigma$. If φ is bounded, then so are all other parameters n, d, σ by (3.3) and r, whence the execution takes time $O(1)$. If $\varphi \geqslant 8$ and $4r \leqslant \varphi\, \mathrm{bs}\, \varphi$, then we have

$$\begin{aligned} r \ &\leqslant \ \frac{1}{4} \varphi\, (\log_2(\varphi + 1) + 1) \ = \ \frac{1}{4} \varphi \left(\frac{\log(\varphi+1)}{\log 2} + 1 \right) \\ &< \ \varphi \log \varphi\, \frac{2 \log(\varphi+1) + 1}{4 \log \varphi} \ = \ \varphi \log \varphi \left(\frac{1}{2} + \frac{2 \log(1 + 1/\varphi) + 1}{4 \log \varphi} \right) \\ &\leqslant \ \varphi \log \varphi, \end{aligned}$$

so we may use Proposition 3.3 to bound the time of step 4 by

$$\begin{aligned} & n\,(\bar{\ell}^n + r^n)\,\tilde{O}(\log^2 \bar{\ell} \log r) + O(N n \log r \min(\log N, n \log r)) \\ = \;& O(n N \log N \log r) + (\varphi \log \varphi)^n \tilde{O}(n \log^2 \varphi \log r) \\ = \;& (N + (\varphi \log \varphi)^n)\,\tilde{O}(n \log^2 \varphi \log N \log r). \end{aligned}$$

Let us now consider step 5, where we have $\varphi \geqslant 8$ and $4r > \varphi \operatorname{bs} \varphi$, whence $r > \varphi$, as previously mentioned. For our complexity analysis, we may freely assume that r is sufficiently large. In particular, by using (3.14), inequality $\operatorname{bs}(\operatorname{bs} r) + 3 < \sigma$ implies

$$\log \log r \leqslant \log \log (r+1) \leqslant \log_2 \log_2 (r+1) \leqslant \operatorname{bs}(\operatorname{bs} r) + 3 < \sigma.$$

On the one hand $\sigma \leqslant \pi$ implies $\log \sigma \leqslant \log \ell_1 + \cdots + \log \ell_n \leqslant \ell_1 + \cdots + \ell_n$. On the other hand $\sigma \leqslant \binom{d+n}{n} = \frac{d+n}{n} \binom{d+n-1}{n-1}$ implies $\log \sigma \leqslant \log(d+1) + \log \binom{d+n-1}{n-1} / \log 2$. Consequently we have $\log^{\circ 3} r < \varphi$ and deduce:

$$\begin{aligned} \mathsf{B}(r) &= (1+\mu(\varphi))\,\varphi \log r = O(\varphi \log r) \\ \mathsf{B}^{\circ 2}(r) &= (1+\mu(\varphi))\,\varphi \log \mathsf{B}(r) = O(\varphi\,(\log \log r + \log \varphi)) \\ \mathsf{B}^{\circ 3}(r) &= (1+\mu(\varphi))\,\varphi \log \mathsf{B}^{\circ 2}(r) = O(\varphi\,(\log \log \log r + \log \varphi)) = O(\varphi^2) \\ \mathsf{B}^{\circ 4}(r) &= (1+\mu(\varphi))\,\varphi \log \mathsf{B}^{\circ 3}(r) = O(\varphi \log \varphi) \\ \mathsf{B}^{\circ 5}(r) &= (1+\mu(\varphi))\,\varphi \log \mathsf{B}^{\circ 4}(r) = (1+\mu(\varphi)) \left(1 + O\left(\frac{\log \log \varphi}{\log \varphi}\right)\right) \varphi \log \varphi. \end{aligned}$$

Therefore the cost of Algorithm 3.3 with parameter $t = 5$ is

$$\left(1 + O\left(\frac{\log \log \varphi}{\log \varphi}\right)\right)^n (N + (\varphi \log \varphi)^n)\,\tilde{O}(n \varphi^5 \log N \log r),$$

by Proposition 3.7 and equation (3.8).

By gathering costs of each step we thus obtain that Algorithm 3.4 takes time

$$\left(1 + c \frac{\log \log \varphi}{\log \varphi}\right)^n (N + (\varphi \log \varphi)^n)\,\tilde{O}(n \varphi^5 \log N \log r),$$

for some universal constant $c > 0$. Finally we set

$$\varepsilon(\ell, d, r) := \begin{cases} c \dfrac{\log \log \varphi}{\log \varphi} & \text{if } \operatorname{bs}(\operatorname{bs} r) + 3 < \sigma \text{ and } \varphi \geqslant 8 \text{ and } 4r > \varphi \operatorname{bs} \varphi \\ 0 & \text{otherwise.} \end{cases} \tag{3.15}$$

to conclude the proof. □

THEOREM 3.10. *There exists a function $\varepsilon(\ell, d, r)$ which tends to 0 when $\max(n, \ell_1, \ldots, \ell_n, d, r)$ tends to infinity, such that the following assertion holds. Let $f \in (\mathbb{Z}/r\mathbb{Z})[x_1, \ldots, x_n]$ be of partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$ and of total degree d, and let $\alpha_1, \ldots, \alpha_N$ be a sequence of points in $(\mathbb{Z}/r\mathbb{Z})^n$. Then, we may compute $f(\alpha_1), \ldots, f(\alpha_N)$ in time*

$$(1 + \varepsilon(\ell, d, r))^n\,(N + (\varphi \log \varphi)^n)\,\tilde{O}(n^2 \varphi^5 \log r), \text{ with } \varphi \text{ defined in (3.2).}$$

Proof. We may compute $\kappa, \nu \in \mathbb{N}$ with $|\kappa - \log_2 N| < 1$ and $|\nu - n \log_2(\varphi \log \varphi)| < 1$ in time $\tilde{O}(\log N + \log n + \log \varphi)$ thanks to the lemmas from the end of section 2. If $\kappa \leqslant \nu$, then $N = O((\varphi \log \varphi)^n)$ and the result directly follows from Proposition 3.9. Otherwise we apply the evaluation algorithm several times with sets of evaluation points of cardinality at most $2^\nu \asymp (\varphi \log \varphi)^n = O(N)$. □

4. EXTENSION RINGS

In this section $r \geqslant 2$ and $k \geqslant 2$ represent integers and we study multi-point multivariate evaluation over $\mathbb{A} := (\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$, where θ is a monic polynomial in $(\mathbb{Z}/r\,\mathbb{Z})[z]$ of degree k. The approach developed in the next paragraphs lifts this problem to an evaluation problem over $(\mathbb{Z}/r\,\mathbb{Z})[z]$, so that fast univariate evaluation/interpolation in z may be used.

4.1. Reduction to prime fields

We endow $\mathbb{Z}[z]$ with the usual norm $\|.\|_\infty$:

$$\left\| \sum_{i \geqslant 0} c_i z^i \right\|_\infty := \max(|c_0|, |c_1|, |c_2|, \ldots).$$

LEMMA 4.1. *For all $i \geqslant 1$ we have $\|(1 + z + \cdots + z^{k-1})^i\|_\infty \leqslant k^{i-1}$.*

Proof. The proof is done by induction on i. The inequality is an equality for $i = 1$. Then we verify that $\|(1 + z + \cdots + z^{k-1})^{i+1}\|_\infty \leqslant k \|(1 + z + \cdots + z^{k-1})^i\|_\infty$ holds for $i \geqslant 1$. □

LEMMA 4.2. *Consider $\bar{f} \in \mathbb{Z}[z][x_1, \ldots, x_n]$ of total degree d in $x_1, \ldots, x_n$ and $\bar{\alpha} \in \mathbb{Z}[z]^n$. Assume that $\bar{f}$ and $\bar{\alpha}$ have their coefficients in $\{0, \ldots, r-1\}$ and degrees $\leqslant k-1$ in z. Then we have $\deg \bar{f}(\bar{\alpha}) \leqslant K$ and $\|\bar{f}(\bar{\alpha})\|_\infty < R$, where*

$$\begin{aligned} K &:= (d+1)(k-1), \\ R &:= k^d \min\left(r^{\ell_1 + \cdots + \ell_n}, \binom{d+n-1}{n-1} r^{d+1} \right). \end{aligned}$$

Proof. The degree bound is clear. Now consider the polynomials

$$\begin{aligned} F &:= \sum_{\substack{0 \leqslant e_1 < \ell_1, \ldots, 0 \leqslant e_n < \ell_n \\ e_1 + \cdots + e_n \leqslant d}} x_1^{e_1} \cdots x_n^{e_n} \\ A &:= (r-1) + \cdots + (r-1) z^{k-1}. \end{aligned}$$

From Lemma 4.1 we obtain

$$\begin{aligned} \|\bar{f}(\bar{\alpha})\|_\infty &\leqslant \|(AF)(A, \ldots, A)\|_\infty \\ &\leqslant \sum_{\substack{0 \leqslant e_1 < \ell_1, \ldots, 0 \leqslant e_n < \ell_n \\ e_1 + \cdots + e_n \leqslant d}} (r-1)^{1 + e_1 + \cdots + e_n} k^{e_1 + \cdots + e_n} \\ &\leqslant k^d \sum_{\substack{0 \leqslant e_1 < \ell_1, \ldots, 0 \leqslant e_n < \ell_n \\ e_1 + \cdots + e_n \leqslant d}} (r-1)^{1 + e_1 + \cdots + e_n}. \end{aligned}$$

The conclusion follows by applying (3.4) and (3.5) to the polynomial $(r-1)F$. □

Algorithm 4.1

Input. $\mathbb{A} := (\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$; $f \in \mathbb{A}[x_1, \ldots, x_n]$; a sequence $\alpha_1, \ldots, \alpha_N$ of points in $\mathbb{A}^n$.

Output. $f(\alpha_1), \ldots, f(\alpha_N)$.

Assumption. θ is monic of degree k.

1. Compute $\ell_1, \ldots, \ell_n, d, \sigma, \varphi, K, R$, the bit size $\mathrm{bs}\, r$ of r, and the bit size $\mathrm{bs}(k \,\mathrm{bs}\, r)$.

2. If $\sigma \leqslant \mathrm{bs}(k \mathrm{bs}\ r)$, then use the naive evaluation algorithm to compute and return $f(\alpha_1), \ldots, f(\alpha_N)$.
3. Call Algorithm 3.2 with $\max(K, \log R)$ and R to compute the minimal sequence of the smallest prime numbers $p_1 < \cdots < p_s$ such that $p_1 > \max(K, \log R)$ and $p_1 \cdots p_s \geqslant R$.
4. Let $\bar{f}$ be the canonical preimage of f in $\mathbb{Z}[z][x_1, \ldots, x_n]$ with degrees $< k$ in z and with integer coefficients in $\{0, \ldots, r-1\}$.
   Let $\bar{\alpha}_i$ represent the similar preimage of α_i in $\mathbb{Z}[z]^n$, for $i = 1, \ldots, N$.
   Compute $\bar{f} \operatorname{rem} p_j$ and $(\bar{\alpha}_i)_{1 \leqslant i \leqslant N} \operatorname{rem} p_j$, for $j = 1, \ldots, s$.
5. For j from 1 to s do
   a. Specialize $\bar{f} \operatorname{rem} p_j$ for $z = 0, \ldots, K$.
   b. Specialize $\bar{\alpha}_1 \operatorname{rem} p_j, \ldots, \bar{\alpha}_N \operatorname{rem} p_j$ for $z = 0, \ldots, K$.
   c. For $z = 0, \ldots, K$, evaluate $\bar{f} \operatorname{rem} p_j$ at $\bar{\alpha}_1 \operatorname{rem} p_j, \ldots, \bar{\alpha}_N \operatorname{rem} p_j$ by using the algorithm from Theorem 3.10.
   d. Interpolate $\bar{f}\ (\bar{\alpha}_1) \operatorname{rem} p_j, \ldots, \bar{f}\ (\bar{\alpha}_N) \operatorname{rem} p_j$.
6. Use Chinese remaindering to recover $\bar{f}(\bar{\alpha}_1), \ldots, \bar{f}(\bar{\alpha}_N)$.
7. Compute the remainders by r and θ of these values to obtain and return $f(\alpha_1), \ldots, f(\alpha_N)$.

PROPOSITION 4.3. *Algorithm 4.1 is correct. Assume $n \geqslant 2$ and that f has partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$ and total degree $\leqslant d$. Then there exists a function $\eta(\ell, d, r, k)$ which tends to 0 when* $\max(n, \ell_1, \ldots, \ell_n, d, r, k)$ *tends to infinity, such that the cost of Algorithm 4.1 is*

$$(1 + \eta(\ell, d, r, k))^n (N + (\varphi \log \varphi)^n)\, \tilde{O}(n^2 \varphi^5 K \log R).$$

Proof. The correctness follows from Lemma 4.2. The quantities $\ell_1, \ldots, \ell_n, d$ may be obtained in time

$$O(\sigma k \log r + n (d+1)^n (1 + \log(d+1))) = \varphi^n \tilde{O}(n \log \varphi k \log r),$$

by Lemmas 2.5, 2.7 and 2.8. As in the beginning of the proof of Proposition 3.9, the cost for deducing σ and φ is negligible. The degree k of θ may be obtained in time $\tilde{O}(k \log r)$. Then, computing K requires time $\tilde{O}(\log(k d))$. To obtain R we first evaluate $r^{\ell_1 + \cdots + \ell_n}$ in time $\tilde{O}((\ell_1 + \cdots + \ell_n) \log r) = \tilde{O}(n d \log r)$ and then $\binom{d+n-1}{n-1} r^{d+1}$ in time $\tilde{O}(n \log(d+1) + d \log r) = \tilde{O}(n d \log r)$. Overall step 1 takes negligible time

$$\varphi^n \tilde{O}(n \log \varphi k \log r). \tag{4.1}$$

If $\sigma \leqslant \mathrm{bs}(k \mathrm{bs}\ r)$, then the naive algorithm in step 2 runs in time

$$n N \sigma \tilde{O}(k \log r) = n N \tilde{O}(k \log r), \tag{4.2}$$

by adapting Lemma 3.8. Proposition 3.6 implies that step 3 takes time

$$\tilde{O}(K + \log R), \tag{4.3}$$

and we have $p_i = O(K + \log R)$ whence $p_1 \cdots p_s \leqslant B := c' R (K + \log R)$, for a universal constant c'.

The cost of step 4 is obtained by adapting Lemmas 2.9 and 2.10 to $\mathbb{A}$:

$$k (n N + (d+1)^n)\, \tilde{O}(\log B) = k (n N + (d+1)^n)\, \tilde{O}(\log R + \log K). \tag{4.4}$$

Let us now examine the cost of step 5. For fixed j, the specializations of $\bar{f}$ rem p_j and the $\bar{\alpha}_i$ rem p_j for $z = 0, \ldots, K$ in steps 5.a and 5.b require time

$$(nN + (d+1)^n)\,\tilde{O}(K \log p_j),$$

by Lemmas 2.12 and 2.13. By Theorem 3.10, the evaluations in step 5.c take time

$$K\,(1 + \varepsilon(\ell, d, p_j))^n \left(N + (\varphi \log \varphi)^n\right) \tilde{O}(n^2 \varphi^5 \log p_j),$$

where (3.15) implies $\varepsilon(\ell,\, d,\, p_j) \leqslant c\, \frac{\log\log\varphi}{\log\varphi}$. The cost of step 5.d is $N\ \tilde{O}(K\ \log\ p_j)$ by Lemma 2.14. It follows that the total cost of step 5 is

$$\begin{aligned}
&\sum_{j=1}^{s} (nN + (d+1)^n)\,\tilde{O}(K \log p_j) \\
&\qquad + K \left(1 + c\,\frac{\log\log\varphi}{\log\varphi}\right)^n (N + (\varphi\log\varphi)^n) \sum_{j=1}^{s} \tilde{O}(n^2\varphi^5 \log p_j) \\
&= (nN + (d+1)^n)\,\tilde{O}(K \log B) + \left(1 + c\,\frac{\log\log\varphi}{\log\varphi}\right)^n (N + (\varphi\log\varphi)^n)\,\tilde{O}(n^2\varphi^5 K \log B) \\
&= \left(1 + c\,\frac{\log\log\varphi}{\log\varphi}\right)^n (N + (\varphi\log\varphi)^n)\,\tilde{O}(n^2\varphi^5 K \log R). \qquad (4.5)
\end{aligned}$$

The cost of step 6 is provided by Lemma 2.11, that is

$$NK\tilde{O}(\log B) = N\tilde{O}(K \log R). \qquad (4.6)$$

Finally the cost of step 7 is

$$N\tilde{O}(K \log B) = N\tilde{O}(K \log R). \qquad (4.7)$$

Summing all costs from (4.1)–(4.7), we obtain the total cost of the algorithm

$$\left(1 + c\,\frac{\log\log\varphi}{\log\varphi}\right)^n (N + (\varphi\log\varphi)^n)\,\tilde{O}(n^2\varphi^5 K \log R).$$

We conclude that the function

$$\eta(\ell, d, r, k) := \begin{cases} c\,\dfrac{\log\log\varphi}{\log\varphi} & \text{if } \mathrm{bs}(k \,\mathrm{bs}\, r) < \sigma \\ 0 & \text{otherwise} \end{cases}$$

satisfies the requirement of the proposition. □

THEOREM 4.4. *There exists a function $\eta(\ell, d, r, k)$ which tends to 0 when $\max(n, \ell_1, \ldots, \ell_n, d, r, k)$ tends to infinity, such that the following holds. Let $\mathbb{A} := (\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$ with θ monic of degree k, let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$ and of total degree d, and let $\alpha_1, \ldots, \alpha_N$ be a sequence of points in $\mathbb{A}^n$. Then, we may compute $f(\alpha_1), \ldots, f(\alpha_N)$ in time*

$$(1 + \eta(\ell, d, r, k))^n\,(N + (\varphi\log\varphi)^n)\,\tilde{O}(n^2 d\,\varphi^6 k \log r).$$

Proof. If $n = 1$, then we use fast univariate multi-point evaluation. Otherwise we use Proposition 4.3 in combination with $\log R \leqslant d \log k + \varphi \log r$. □

4.2. Corollaries in terms of partial and total degrees

The first corollary is a complexity bound in terms of the partial degrees, while the second one concerns the total degree.

COROLLARY 4.5. *Let $\varepsilon > 0$ be a fixed real value. Let $\mathbb{A} := (\mathbb{Z}/r\mathbb{Z})[z]/(\theta(z))$ with θ monic of degree k, let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of partial degree $< \bar{\ell}$ in x_i for $i = 1, \ldots, n$, and let $\alpha_1, \ldots, \alpha_N$ be a sequence of points in $\mathbb{A}^n$. Then we may compute $f(\alpha_1), \ldots, f(\alpha_N)$ in time*

$$(1+\varepsilon)^n (N + (n\bar{\ell}\log(n\bar{\ell}))^n)\, \tilde{O}(n^9 \bar{\ell}^7 k \log r).$$

Proof. We apply Theorem 4.4 with $d, \varphi \leqslant n\bar{\ell}$. □

COROLLARY 4.6. *Let $\varepsilon > 0$ be a fixed real value. Let $\mathbb{A} := (\mathbb{Z}/r\mathbb{Z})[z]/(\theta(z))$ with θ monic of degree k, let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of total degree $\leqslant d$, and let $\alpha_1, \ldots, \alpha_N$ be a sequence of points in $\mathbb{A}^n$. Then we may compute $f(\alpha_1), \ldots, f(\alpha_N)$ in time*

$$(1+\varepsilon)^n (N + ((3d+2n)\log(3d+2n))^n)\, \tilde{O}(n^2 d\,(d+n)^6 k \log r).$$

Proof. We apply Theorem 4.4 with $\ell \leqslant d+1$ and make use of the well known inequality $\binom{2n}{n} \leqslant 4^n$. If $d \leqslant n-1$, then

$$\varphi \leqslant d+1+\left\lceil \frac{\log \binom{d+n-1}{n-1}}{\log 2} \right\rceil \leqslant d+1+\frac{\log \binom{2(n-1)}{n-1}}{\log 2}+1 \leqslant d+2n.$$

Otherwise,

$$\varphi \leqslant d+1+\left\lceil \frac{\log \binom{d+n-1}{d}}{\log 2} \right\rceil \leqslant d+1+\frac{\log \binom{2d}{d}}{\log 2}+1 \leqslant d+2+2d = 3d+2.$$

In both cases we thus have $\varphi \leqslant 3d+2n$. □

5. KRONECKER SEGMENTATION

If the partial degrees are large with respect to the number of the variables, then we may use Kronecker segmentation on f in order to decrease the dependency in φ in the complexity bounds from the two previous sections. We first analyze the cost of Kronecker segmentation on Turing machines and then show how to reduce the complexity of multi-point evaluation. Throughout this section, $\mathbb{A}$ is an effective ring whose elements occupy at most $\mathsf{s}_{\mathbb{A}}$ cells on tapes and whose arithmetic operations take softly linear time.

5.1. Univariate case

Let $\hat{\ell}_1, \ldots, \hat{\ell}_m$ be integers $\geqslant 2$. The Kronecker substitution map is the unique $\mathbb{A}$-algebra morphism determined by

$$\begin{aligned} K_{\hat{\ell}_1,\ldots,\hat{\ell}_m} \colon \mathbb{A}[x_1,\ldots,x_m] &\to \mathbb{A}[x] \\ x_i &\mapsto x^{\hat{\ell}_1 \cdots \hat{\ell}_{i-1}}. \end{aligned}$$

When restricted to the space of polynomials of partial degree $< \hat{\ell}_i$ in x_i, it becomes an $\mathbb{A}$-linear isomorphism onto the space of polynomials in $\mathbb{A}[x]$ of degree $< \hat{\ell}_1 \cdots \hat{\ell}_m$. The Kronecker segmentation associated to $\hat{\ell}_1, \ldots, \hat{\ell}_m$ transforms the univariate polynomial $f \in \mathbb{A}[x]$ of degree $< \hat{\ell}_1 \cdots \hat{\ell}_m$ into the multivariate polynomial $\hat{f} = K^{-1}_{\hat{\ell}_1,\ldots,\hat{\ell}_m}(f)$, so that

$$f(x) = \hat{f}\left(x, x^{\hat{\ell}_1}, x^{\hat{\ell}_1\hat{\ell}_2}, \ldots, x^{\hat{\ell}_1 \cdots \hat{\ell}_{m-1}}\right).$$

Algorithm 5.1
Input. $f = \sum_{0 \leqslant i < \ell} f_i x^i \in \mathbb{A}[x]$; a sequence $\hat{\ell}_1, \ldots, \hat{\ell}_m$ of integers $\geqslant 2$.
Output. $K^{-1}_{\hat{\ell}_1, \ldots, \hat{\ell}_m}(f)$.
Assumption. $\ell \leqslant \hat{\ell}_1 \cdots \hat{\ell}_m$.

1. If $m = 1$, then return $f(x_1)$.
2. For $i = 0, \ldots, \hat{\ell}_m - 1$, call the algorithm recursively on $F_i(x) := \sum_{0 \leqslant j < \hat{\ell}_1 \cdots \hat{\ell}_{m-1}} f_{j + \hat{\ell}_1 \cdots \hat{\ell}_{m-1} i} x^j$ and integers $\hat{\ell}_1, \ldots, \hat{\ell}_{m-1}$ to obtain $\hat{F}_i := K^{-1}_{\hat{\ell}_1, \ldots, \hat{\ell}_{m-1}}(F_i)$.
3. Return $\sum_{0 \leqslant i < \ell_m} \hat{F}_i(x_1, \ldots, x_{m-1})\, x_m^i$.

PROPOSITION 5.1. *Algorithm 5.1 is correct and takes time* $O\big(m\, \hat{\ell}_1 \cdots \hat{\ell}_m\, \mathsf{s}_{\mathbb{A}}\big)$.

Proof. The correctness is clear. Let $\mathsf{K}\big(\hat{\ell}_1, \ldots, \hat{\ell}_m\big)$ represent the cost function of the algorithm. Step 1 takes linear time in the size of f, that is $\mathsf{K}\big(\hat{\ell}_1\big) = O\big(\hat{\ell}_1\, \mathsf{s}_{\mathbb{A}}\big)$. Step 2 requires one linear traversal and $\hat{\ell}_m$ recursive calls, whence

$$\mathsf{K}\big(\hat{\ell}_1, \ldots, \hat{\ell}_m\big) = O\big(\hat{\ell}_1 \cdots \hat{\ell}_m\, \mathsf{s}_{\mathbb{A}}\big) + \hat{\ell}_m\, \mathsf{K}\big(\hat{\ell}_1, \ldots, \hat{\ell}_{m-1}\big).$$

By induction over m, it follows that $\mathsf{K}\big(\hat{\ell}_1, \ldots, \hat{\ell}_m\big) = O\big(m\, \hat{\ell}_1 \cdots \hat{\ell}_m\, \mathsf{s}_{\mathbb{A}}\big)$. □

The cost of the Kronecker substitution, stated in the next proposition, will be needed in section 7 only.

PROPOSITION 5.2. *Let* $f \in \mathbb{A}[x_1, \ldots, x_m]$ *be of partial degree* $< \ell_i$ *in* x_i *for* $i = 1, \ldots, n$. *The Kronecker substitution* $K_{\hat{\ell}_1, \ldots, \hat{\ell}_m}(f)$ *of* f *may be computed in time* $O\big(m\, \hat{\ell}_1 \cdots \hat{\ell}_m\, \mathsf{s}_{\mathbb{A}}\big)$.

Proof. The proof is done by induction on m. We will require in addition that $K_{\hat{\ell}_1, \ldots, \hat{\ell}_m}(f)$ is zero padded up to degree $\hat{\ell}_1 \cdots \hat{\ell}_m - 1$ (at the end, we may clearly remove trailing zeros in linear time). The case $m = 1$ corresponds to a simple zero padding up to degree $\ell_1 - 1$, which takes time $O(\ell_1\, \mathsf{s}_{\mathbb{A}})$. If $m \geqslant 2$, then we write $f = f_0 + \cdots + f_{\hat{\ell}_m - 1} x_m^{\hat{\ell}_m - 1}$, we recursively compute $K_{\hat{\ell}_1, \ldots, \hat{\ell}_{m-1}}(f_i)$ for $i = 0, \ldots, \hat{\ell}_m - 1$, and concatenate the results onto the output tape. The complexity bound is straightforward. □

Only the univariate Kronecker segmentation is actually needed for the modular composition algorithm of the next section. In the rest of this section we introduce the multivariate segmentation and make use of it in order to speed up multi-point evaluation.

5.2. Multivariate case

Now consider a multivariate polynomial $f \in \mathbb{A}[x_1, \ldots, x_n]$ of partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$. For $i = 1, \ldots, n$, let $\hat{\ell}_{i,1}, \ldots, \hat{\ell}_{i,m_i}$ be integers such that

$$\ell_i \leqslant \hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i},\ m_i = 1 \text{ and } \hat{\ell}_{i,1} = 1 \text{ whenever } \ell_i = 1, \text{ and } \hat{\ell}_{i,j} \geqslant 2 \text{ otherwise.} \tag{5.1}$$

We introduce the multivariate Kronecker substitution map

$$\begin{aligned} K_{\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n,m_n}} \colon \mathbb{A}[x_{1,1}, \ldots, x_{1,m_1}, \ldots, x_{n,1}, \ldots, x_{n,m_n}] &\to \mathbb{A}[x_1, \ldots, x_n] \\ x_{i,j} &\mapsto x_i^{\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,j-1}} \qquad (1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m_i). \end{aligned} \tag{5.2}$$

This map restricts to an $\mathbb{A}$-linear isomorphism between polynomials of partial degree $<\hat{\ell}_{i,j}$ in $x_{i,j}$ and polynomials in $\mathbb{A}[x_1, \ldots, x_n]$ of partial degree $<\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i}$ in x_i. In this context, the Kronecker segmentation of f is defined as $K^{-1}_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(f)$.

PROPOSITION 5.3. *Let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of partial degree $<\ell_i$ in x_i for $i = 1, \ldots, n$, and let $\hat{\ell}_{i,1}, \ldots, \hat{\ell}_{i,m_i}$ be integers satisfying (5.1). The Kronecker segmentation $K^{-1}_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(f)$ of f may be computed in time*

$$O(\hat{n}\,\hat{\pi}\,(\mathsf{s}_{\mathbb{A}} + n)),$$

where $\hat{n} := m_1 + \cdots + m_n$ and $\hat{\pi} := \hat{\ell}_{1,1} \cdots \hat{\ell}_{1,m_1} \cdots \hat{\ell}_{n,1}, \ldots, \hat{\ell}_{n,m_n}$.

Proof. If $n = 1$, then we may use Proposition 5.1. Otherwise we consider f in the form $f = \sum_{0 \leqslant i < \ell_m} f_i(x_1, \ldots, x_{n-1})\, x_n^i$. Let $\mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n,m_n}\big)$ represent the cost of the segmentation of f. By Lemma 2.5 the representation size of each f_i is $O(\ell_1 \cdots \ell_{n-1}\,(\mathsf{s}_{\mathbb{A}} + n))$. If $\ell_n \geqslant 2$, then Proposition 5.1 gives

$$\mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n,m_n}\big) \leqslant O(m_n\,\hat{\pi}\,(\mathsf{s}_{\mathbb{A}} + n)) + \hat{\ell}_{n,1} \cdots \hat{\ell}_{n,m_n}\, \mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n-1,m_{n-1}}\big).$$

Otherwise, we simply have $\mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n,m_n}\big) = O(\pi\,(\mathsf{s}_{\mathbb{A}} + n)) + \mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n-1,m_{n-1}}\big)$. It follows that $\mathsf{K}\big(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{n,m_n}\big) = O((m_n + m_{n-1} + \cdots + m_1)\,\hat{\pi}\,(\mathsf{s}_{\mathbb{A}} + n))$. □

5.3. Application to multi-point evaluation

In the rest of this section we explain how to decrease the cost of multi-point evaluation using Kronecker segmentation of f.

Recall that $\bar{\ell} := \max(\ell_1, \ldots, \ell_n)$. Let $\bar{m} \geqslant 1$ be an integer such that $\bar{m} = O(\log \bar{\ell})$. For $i = 1, \ldots, n$, we let

$$m_i := \left\lceil \frac{\log \ell_i}{\log \bar{\ell}}\, \bar{m} \right\rceil = O(\log \ell_i).$$

We thus have

$$\frac{\log \ell_i}{\log \bar{\ell}}\, \bar{m} \leqslant m_i < \frac{\log \ell_i}{\log \bar{\ell}}\, \bar{m} + 1$$

which implies

$$\frac{\log \bar{\ell}}{\bar{m} + \frac{\log \bar{\ell}}{\log \ell_i}} < \frac{\log \ell_i}{m_i} \leqslant \frac{\log \bar{\ell}}{\bar{m}},$$

whence

$$\ell_i^{1/m_i} \leqslant \bar{\ell}^{1/\bar{m}}. \tag{5.3}$$

If $\ell_i = 1$, then we set $m_i := 1$ so that inequality (5.3) still holds. In addition, if $m_i \geqslant 2$, then $\ell_i > \bar{\ell}^{1/\bar{m}}$, whence $\frac{\log \bar{\ell}}{\log \ell_i} < \bar{m}$, and

$$\bar{\ell}^{1/(2\bar{m})} < \ell_i^{1/m_i}. \tag{5.4}$$

Notice that we have $\bar{m} = \max(m_1, \ldots, m_n)$. For all $1 \leqslant i \leqslant n$ we introduce

$$\hat{\ell}_{i,j} := \lfloor \ell_i^{1/m_i} \rfloor \text{ for } j = 1, \ldots, m_i - 1, \text{ and } \hat{\ell}_{i,m_i} := \left\lceil \ell_i / \big(\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i-1}\big) \right\rceil,$$

so $\hat{\ell}_{i,1} = \cdots = \hat{\ell}_{i,m_i-1} \leqslant \hat{\ell}_{i,m_i}$ and $\ell_i \leqslant \hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i}$ hold. From

$$\ell_i^{1/m_i} - 1 < \hat{\ell}_{i,j} \leqslant \ell_i^{1/m_i}, \tag{5.5}$$

for $j = 1, \ldots, m_i - 1$, and

$$\ell_i / \left(\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i-1}\right) \leqslant \hat{\ell}_{m_i} < \ell_i / \left(\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i-1}\right) + 1 \tag{5.6}$$

we deduce that

$$\hat{\ell}_{i,1} \cdots \hat{\ell}_{i,m_i} < \ell_i + \ell_i^{(m_i-1)/m_i} = \ell_i \left(1 + \ell_i^{-1/m_i}\right) \leqslant 2\,\ell_i. \tag{5.7}$$

In a dual manner to the Kronecker substitution map (5.1) associated to the $\hat{\ell}_{i,j}$ we introduce the map

$$\begin{aligned} E_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}} \colon \mathbb{A}^n &\to \mathbb{A}^{\hat{n}} \\ (x_1, \ldots, x_n) &\mapsto \left(x_1, x_1^{\hat{\ell}_{1,1}}, \ldots, x_1^{\hat{\ell}_{1,1} \cdots \hat{\ell}_{1,m_1-1}}, \ldots, x_n, x_n^{\hat{\ell}_{n,1}}, \ldots, x_n^{\hat{\ell}_{n,1} \cdots \hat{\ell}_{n,m_n-1}}\right), \end{aligned}$$

where $\hat{n} := m_1 + \cdots + m_n$. Letting $\hat{f} = K^{-1}_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(f)$ we thus have

$$f = \hat{f} \circ E_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}.$$

In this way we reduce the multi-point evaluation in n variables and partial degree bounds $(\ell_1, \ldots, \ell_n)$ to evaluation in $\hat{n}$ variables and partial degree bounds $\left(\hat{\ell}_{1,1}, \ldots, \hat{\ell}_{1,m_1}, \ldots, \hat{\ell}_{n,1}, \ldots, \hat{\ell}_{n,m_n}\right)$. Notice that this segmentation generically builds a polynomial $\hat{f}$ of total degree close to the sum of its partial degrees. The cardinality of the support of $\hat{f}$ is the same as of the support of f, but its number of "#" symbols in the representation is larger. From (5.7) we deduce that

$$\hat{\pi} := \hat{\ell}_{1,1} \cdots \hat{\ell}_{1,m_1} \cdots \hat{\ell}_{n,1} \cdots \hat{\ell}_{n,m_n} \leqslant 2^n \pi. \tag{5.8}$$

The latter 2^n may be replaced by a smaller value c^n with $c > 1$ arbitrarily close to 1 whenever $\min(\ell_1, \ldots, \ell_n)$ is sufficiently large. The reduction process is summarized in the following algorithm.

Algorithm 5.2

Input. $f \in \mathbb{A}[x_1, \ldots, x_n]$; a sequence $\alpha_1, \ldots, \alpha_N$ of points in $\mathbb{A}^n$; an integer $\bar{m} \geqslant 1$.

Output. $K^{-1}_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(f)$; $E_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(\alpha_1), \ldots, E_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(\alpha_N)$.

1. Compute $\ell_1, \ldots, \ell_n, \bar{\ell}$ and then m_i and $\hat{\ell}_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$.
2. Build $\hat{f} := K^{-1}_{\hat{\ell}_{1,1},\ldots,\hat{\ell}_{n,m_n}}(f)$.
3. For all $i = 1, \ldots, N$ compute $\hat{\alpha}_i := \left(\alpha_{i,1}, \alpha_{i,1}^{\hat{\ell}_{1,1}}, \ldots, \alpha_{i,1}^{\hat{\ell}_{1,1} \cdots \hat{\ell}_{1,m_1-1}}, \ldots, \alpha_{i,n}, \alpha_{i,n}^{\hat{\ell}_{n,1}}, \ldots, \alpha_{i,n}^{\hat{\ell}_{n,1} \cdots \hat{\ell}_{n,m_n-1}}\right)$.
4. Return $\hat{f}$ and $\hat{\alpha}_1, \ldots, \hat{\alpha}_N$.

PROPOSITION 5.4. *Algorithm 5.2 is correct and takes time*

$$O(n\,\pi \log \pi) + n\,\tilde{O}(\log^2 \bar{\ell}) + O(\hat{n}\,\hat{\pi}\,(\mathsf{s}_{\mathbb{A}} + n)) + N\,(\hat{n} + \log \hat{\pi})\,\tilde{O}(\mathsf{s}_{\mathbb{A}}).$$

Proof. The correctness is clear from the definitions. The quantities $\ell_1, \ldots, \ell_n, \bar{\ell}$ may be obtained in time $O(\pi\,\mathsf{s}_{\mathbb{A}} + n\,\pi\,(n + \log\,\pi)) = \pi\,O(\mathsf{s}_{\mathbb{A}} + n^2 + n \log\,\pi)$ by Lemmas 2.5 and 2.7. Then we use a binary search to determine m_i as the first integer such that $\ell_i^{\bar{m}} \leqslant \bar{\ell}^{m_i}$ in time $\tilde{O}(\bar{m} \log \bar{\ell}) = \tilde{O}(\log^2 \bar{\ell})$. By Proposition 5.3 the segmentation in step 2 takes time $O(\hat{n}\,\hat{\pi}\,(\mathsf{s}_{\mathbb{A}} + n))$. Using binary powering, step 3 involves $O(\hat{n} + \log \hat{\pi})$ operations in $\mathbb{A}$ for each point α_i. □

THEOREM 5.5. *Let $\varepsilon > 0$ be a fixed real value and let $\bar{m} \geqslant 2$ be a fixed integer. Let $\mathbb{A} := (\mathbb{Z}/r\mathbb{Z})[z]/(\theta(z))$ with θ monic of degree k, let $f \in \mathbb{A}[x_1, \ldots, x_n]$ be of partial degree $< \ell_i$ in x_i for $i = 1, \ldots, n$, and let $\alpha_1, \ldots, \alpha_N$ be a sequence of points in $\mathbb{A}^n$. Then, we may compute $f(\alpha_1), \ldots, f(\alpha_N)$ in time*

$$(1+\varepsilon)^{2\bar{m}n} (N + (\bar{m}n)^{\bar{m}n} \bar{\ell}^n \log^{\bar{m}n}((1+\varepsilon)\bar{m}n\bar{\ell}^{1/\bar{m}}))\, \tilde{O}(n^{10}\bar{\ell}^{7/\bar{m}} k \log r),$$

where $\bar{\ell} := \max(\ell_1, \ldots, \ell_n)$.

Proof. We may freely assume that $\max(n, \bar{\ell}, k, \log r)$ is sufficiently large, so that the cost of multi-modular evaluation is

$$\begin{aligned} & (1+\varepsilon)^n (N + (\varphi \log \varphi)^n)\, \tilde{O}(n^2 d \varphi^6 k \log r) \\ = \; & (1+\varepsilon)^n (N + (n\bar{\ell}\log(n\bar{\ell}))^n)\, \tilde{O}(n^9 \bar{\ell}^7 k \log r), \end{aligned}$$

by Theorem 4.4. If $\bar{\ell} \leqslant n^{1/7}$, then we deduce the complexity bound

$$(1+\varepsilon)^n (N + n^{2n}\bar{\ell}^n)\, \tilde{O}(n^{10} k \log r),$$

so we are done.

From now we assume that $n^{1/7} \leqslant \bar{\ell}$. If $\bar{\ell}$ is bounded, then so is n, and we may appeal to the naive evaluation algorithm; the conclusion follows by adapting Lemma 3.8 to $\mathbb{A}$. We may thus further assume that $\bar{\ell}$ is sufficiently large to satisfy

$$\frac{1}{(1-\bar{\ell}^{-2/\bar{m}})^{\bar{m}-1}} + \bar{\ell}^{-1/\bar{m}} \leqslant 1 + \varepsilon.$$

If $m_i = 1$, then $\hat{\ell}_{i,m_i} = \ell_i^{1/m_i} \leqslant \bar{\ell}^{1/\bar{m}}$. Otherwise, (5.3), (5.4), (5.5), and (6.6) imply

$$\hat{\ell}_{i,m_i} < \frac{\ell_i}{(\ell_i^{1/m_i} - 1)^{m_i - 1}} + 1 = \frac{\ell_i^{1/m_i}}{(1-\ell_i^{-1/m_i})^{m_i-1}} + 1 \leqslant \frac{\bar{\ell}^{1/\bar{m}}}{(1-\bar{\ell}^{-2/\bar{m}})^{\bar{m}-1}} + 1,$$

whence

$$\hat{\ell}_{i,m_i} < \left(\frac{1}{(1-\bar{\ell}^{-2/\bar{m}})^{\bar{m}-1}} + \bar{\ell}^{-1/\bar{m}} \right) \bar{\ell}^{1/\bar{m}}.$$

For all $1 \leqslant i \leqslant n$ we thus have $\hat{\ell}_{i,m_i} \leqslant (1+\varepsilon)\,\bar{\ell}^{1/\bar{m}}$. It follows that

$$\hat{d} := \hat{\ell}_{1,1} + \cdots + \hat{\ell}_{n,m_n} - \hat{n} \leqslant (1+\varepsilon)\,\bar{m}n\bar{\ell}^{1/\bar{m}}$$

and

$$\hat{\varphi} := \min\left(\hat{\ell}_{1,1} + \cdots + \hat{\ell}_{n,m_n}, \hat{d} + 1 + \left\lceil \frac{\log \binom{\hat{d}+\hat{n}-1}{\hat{n}-1}}{\log 2} \right\rceil \right) \leqslant (1+\varepsilon)\,\bar{m}n\bar{\ell}^{1/\bar{m}}.$$

Using Proposition 5.4, we compute $\hat{f}$ and the $\hat{\alpha}_i$ by Algorithm 5.2 in time

$$\begin{aligned} & O(\hat{n}\hat{\pi}(k\log r + n\log^3\bar{\ell})) + N(\hat{n} + \log\hat{\pi})\,\tilde{O}(k\log r) \\ = \; & (1+\varepsilon)^{n\bar{m}} n\bar{m}\bar{\ell}^n \tilde{O}(n\bar{\ell}^{7/\bar{m}} k\log r) + \hat{n}N(\hat{n} + \log\bar{\ell})\,\tilde{O}(k\log r) \\ = \; & (1+\varepsilon)^{n\bar{m}} (N + n\bar{m}\bar{\ell}^n)\,\tilde{O}(n^2\bar{\ell}^{\varepsilon} k\log r). \end{aligned}$$

Then Theorem 4.4 ensures that the evaluation of $\hat{f}$ at all the $\hat{\alpha}_i$ takes time

$$(1 + \eta(\hat{\ell}, \hat{d}, r, k))^{\bar{m}n} (N + (\hat{\varphi}\log\hat{\varphi})^{\hat{n}})\, \tilde{O}(\hat{n}^2 \hat{d}\hat{\varphi}^6 k \log r). \tag{5.9}$$

Now we further assume that $\max(n, \bar{\ell}, r, k)$ is sufficiently large such that

$$\eta(\hat{\ell}, \hat{d}, r, k) \leqslant \varepsilon.$$

Then the cost (5.9) rewrites into

$$(1+\varepsilon)^{\bar{m}n} (N + (1+\varepsilon)^{\bar{m}n} (\bar{m}n)^{\bar{m}n} \bar{\ell}^n \log^{\bar{m}n}((1+\varepsilon)\bar{m}n\bar{\ell}^{1/\bar{m}}))\, \tilde{O}(n^9\bar{\ell}^{7/\bar{m}} k\log r). \qquad \square$$

5.4. Consequence in terms of total degree

In the univariate case it is well known that a polynomial of degree d may be evaluated at d points in softly linear time. In the multivariate setting we wish to reach softly linear time for the evaluation in degree d, with n variables, at $\binom{d+n}{n}$ points. Although such a complexity bound seems out of reach for the present techniques, the aim of this section is to prove a slightly weaker bound. We start with a simple lemma.

LEMMA 5.6. *For all positive integers n and d we have*

$$\log\binom{d+n}{n} \leqslant n\log\left(1+\frac{d}{n}\right) + d\log\left(1+\frac{n}{d}\right).$$

Proof. The bound is proved as follows:

$$\begin{aligned}\log\binom{n+d}{n} &= \sum_{i=1}^{n} \log\left(1+\frac{d}{i}\right)\\ &\leqslant \sum_{i=1}^{n} \int_{i-1}^{i} \log\left(1+\frac{d}{t}\right)\mathrm{d}t\\ &= \int_{0}^{n} \log\left(1+\frac{d}{t}\right)\mathrm{d}t = n\log\left(1+\frac{d}{n}\right) + d\log\left(1+\frac{n}{d}\right).\end{aligned}$$

□

PROPOSITION 5.7. *Let $\varepsilon > 0$ be a fixed rational value. Let $\mathbb{A} := (\mathbb{Z}/r\,\mathbb{Z})[z]/(\theta(z))$ with θ monic of degree k, let $f \in \mathbb{A}[x_1,\ldots,x_n]$ be of total degree $\leqslant d$ and let $\alpha_1,\ldots,\alpha_N$ be in $\mathbb{A}^n$ with $N \leqslant d^n$. Then we may compute $f(\alpha_1),\ldots,f(\alpha_N)$ in time $\tilde{O}(\max(n^2, d^{(1+\varepsilon)n})\,k\log r)$.*

Proof. If $d = 1$, then Lemma 3.8 ensures evaluation time $n^2\,\tilde{O}(d^n k\log r)$. From now on we may assume $d \geqslant 2$.

First we examine the case when $d \leqslant \gamma n$ for a constant $\gamma > 0$ to be fixed. Lemma 5.6 combined to the fact that the function $\gamma \mapsto \log(1+\gamma) + \gamma\log(1+1/\gamma)$ is nondecreasing yields

$$\log\binom{d+n}{n} \leqslant n\left(\log\left(1+\frac{d}{n}\right) + \frac{d}{n}\log\left(1+\frac{n}{d}\right)\right) \leqslant (\log(1+\gamma) + \gamma\log(1+1/\gamma))\,n.$$

We fix the constant γ sufficiently small such that $\binom{d+n}{n} \leqslant (1+\varepsilon)^n$. By Lemma 3.8, the evaluations can be achieved in time $(1+\varepsilon)^n\,\tilde{O}(d^n k\log r)$.

From now we may assume that $\gamma\, n < d$ holds. If d is bounded, then so is n and Lemma 3.8 again leads to an evaluation time $\tilde{O}(d^n\, k\log\, r)$. Consequently we may further assume that d is sufficiently large to satisfy

$$d \geqslant 3 \quad\text{and}\quad (3+2/\gamma)\log((3+2/\gamma)\,d) \leqslant d^{\varepsilon}. \tag{5.10}$$

Setting $\bar{m} := \lceil 7/\varepsilon \rceil$, Theorem 5.5 provides us with the time complexity bound

$$(1+\varepsilon)^{2\bar{m}n}\,(N + (\bar{m}\,n)^{\bar{m}n}\,(d+1)^n \log^{\bar{m}n}((1+\varepsilon)\,\bar{m}\,n\,(d+1)^{1/\bar{m}}))\,\tilde{O}(n^{10}\,(d+1)^{7/\bar{m}}\,k\log r).$$

For a universal constant $c \geqslant 1$, this bound simplifies to

$$(1+\varepsilon)^{O(n)}\,n^{cn}\log^{cn} d\,\tilde{O}(d^{n+\varepsilon}\,k\log r).$$

Whenever $n^{cn}\log^{cn} d \leqslant d^{\varepsilon n}$ the latter cost further simplifies to

$$(1+\varepsilon)^{O(n)}\,\tilde{O}(d^{(1+\varepsilon)n+\varepsilon}\,k\log r) = (1+\varepsilon)^{O(n)}\,\tilde{O}(d^{(1+2\varepsilon)n}\,k\log r).$$

We now consider the other case when $d^{\varepsilon n} < n^{cn} \log^{cn} d$, so we have $d^{\varepsilon} < n^c \log^c d$, hence $d = \log^{O(1)}(d^n)$. In this case, Corollary 4.6 gives the cost

$$(1+\varepsilon)^n (d^n + ((3d+2n)\log(3d+2n))^n)\, \tilde{O}(n^2 d\,(d+n)^6 k \log r),$$

which is bounded from above by

$$\begin{aligned} & (1+\varepsilon)^n (d^n + ((3d+2n)\log(3d+2n))^n)\, \tilde{O}(d^{O(1)} k \log r) \\ \leqslant\ & (1+\varepsilon)^n ((3+2/\gamma)\log((3+2/\gamma)\,d))^n \tilde{O}(d^n k \log r) \\ \leqslant\ & (1+\varepsilon)^n \tilde{O}(d^{(1+\varepsilon)n} k \log r), \qquad \text{by (5.10).} \end{aligned}$$

In all cases with $d \geqslant 2$, we have thus proved the complexity bound

$$(1+\varepsilon)^n \tilde{O}(d^{(1+2\varepsilon)n} k \log r).$$

Since $d \geqslant 2$, we have $(1+\varepsilon)^n = d^{n\log(1+\varepsilon)/\log d} \leqslant d^{2\varepsilon n}$, so the total running time is bounded by $\tilde{O}(d^{(1+4\varepsilon)n} k \log r)$. We conclude by applying this for $\varepsilon/4$ instead of ε. □

6. MODULAR COMPOSITION

In this section, $\mathbb{K}$ is an effective field, h is a monic polynomial in $\mathbb{K}[x]$ of degree d, and f, g are two polynomials in $\mathbb{K}[x]_{<d}$. We want to compute $f \circ g$ rem h. We first describe and analyze the algorithm in the algebraic complexity model and then return to Turing machines in the case when $\mathbb{K}$ is a finite field.

Let $\delta := \sqrt{7}$. This precise choice of δ will be motivated below. We let n be an integer such that

$$\left| n - \delta \left(\frac{\log(d+1)}{\log\log(d+1)} \right)^{1/2} \right| < 1. \tag{6.1}$$

In particular, we have

$$n = \delta \left(\frac{\log(d+1)}{\log\log(d+1)} \right)^{1/2} \left(1 + O\left(\left(\frac{\log\log(d+1)}{\log(d+1)} \right)^{1/2} \right) \right)$$

and

$$\frac{1}{n} = \frac{1}{\delta} \left(\frac{\log\log(d+1)}{\log(d+1)} \right)^{1/2} \left(1 + O\left(\left(\frac{\log\log(d+1)}{\log(d+1)} \right)^{1/2} \right) \right),$$

so that

$$\begin{aligned} (d+1)^{1/n} &> (d+1)^{\frac{1}{\delta}\left(\frac{\log\log(d+1)}{\log(d+1)}\right)^{1/2}(1+o(1))} \\ &\geqslant \exp\left(\frac{1}{\delta} (\log(d+1)\log\log(d+1))^{1/2} (1+o(1)) \right) \end{aligned} \tag{6.2}$$

tends to $+\infty$ for large values of d. Now we define

$$\begin{aligned} \ell_i &:= \lfloor (d+1)^{1/n} \rfloor && (1 \leqslant i \leqslant n-1) \\ \ell_n &:= \lceil (d+1)/(\ell_1 \cdots \ell_{n-1}) \rceil, \end{aligned}$$

so that $d+1 \leqslant \ell_1 \cdots \ell_n$. If d is sufficiently large, then $\ell_i \geqslant 2$ for all i. In addition, from

$$(d+1)^{1/n} - 1 < \ell_i \leqslant (d+1)^{1/n} \qquad (1 \leqslant i \leqslant n-1)$$

and

$$(d+1)/(\ell_1 \cdots \ell_{n-1}) \leqslant \ell_n < (d+1)/(\ell_1 \cdots \ell_{n-1}) + 1,$$

we deduce

$$\ell_1 \cdots \ell_n < (d+1) + (d+1)^{(n-1)/n} = (d+1)\,(1 + (d+1)^{-1/n}) \leqslant 2\,(d+1). \tag{6.3}$$

We are now ready to state the modular composition algorithm.

Algorithm 6.1

Input. $f, g, h \in \mathbb{K}[x]$; N distinct points $\gamma_1, \ldots, \gamma_N$ in $\mathbb{K}$.

Output. $f \circ g \bmod h$.

Assumptions. $d := \deg h$, $\deg f < d$, $\deg g < d$, $D := \ell_1 + \cdots + \ell_n - n$, and $N := D(d-1)+1$, where n is as in (6.1).

1. Compute $n, \ell_1, \ldots, \ell_n$.
   If one of the ℓ_i is less than 2, then use the naive modular composition algorithm.
2. Build $F := K^{-1}_{\ell_1,\ldots,\ell_n}(f)$.
3. Compute $a_1 := x$, $a_2 := g^{\ell_1} \bmod h, \ldots, a_n := g^{\ell_1 \cdots \ell_{n-1}} \bmod h$.
4. Compute $\alpha_i := (a_1(\gamma_i), \ldots, a_n(\gamma_i))$ for $i = 1, \ldots, N$.
5. Evaluate F at α_i for $i = 1, \ldots, N$.
6. Interpolate $\rho \in \mathbb{A}[x]_{<N}$ such that $\rho(\gamma_i) = F(\alpha_i)$ for $i = 1, \ldots, N$.
7. Return $\rho \bmod h$.

The following proposition summarizes the cost in terms of arithmetic operations in $\mathbb{K}$.

PROPOSITION 6.1. *Algorithm 6.1 is correct and takes $O(n\,D\,\mathsf{M}(d)\log d + \mathsf{M}(D\,d)\log(D\,d))$ operations in $\mathbb{K}$ plus the multi-point evaluation of step 5.*

Proof. If d is sufficiently large, then $\ell_i \geqslant 2$ for all i. Step 3 performs

$$O(\mathsf{M}(d)\log(\ell_1 \cdots \ell_{n-1})) = O(\mathsf{M}(d)\log d)$$

operations in $\mathbb{K}$ in view of (6.3). Steps 4 and 6 respectively take time $O(n\,D\,\mathsf{M}(d)\log d)$ and $O(\mathsf{M}(D\,d)\log(D\,d))$. Step 7 takes $O(\mathsf{M}(D\,d))$ additional operations in $\mathbb{K}$. □

From now we return to the Turing machine model.

THEOREM 6.2. *Let f, g, h be polynomials in $\mathbb{F}_q[x]$ such that h is monic of degree d and f, g have degrees $<d$. We assume that $\mathbb{F}_q$ is given as $\mathbb{F}_p[z]/(\theta(z))$ with p prime and θ monic of degree k. Then $f \circ g \bmod h$ may be computed in time*

$$(d+1)^{\left(28\frac{\log\log(d+1)}{\log(d+1)}\right)^{1/2}\left(1+O\left(\frac{1}{\log\log(d+1)}\right)\right)}\,\tilde{O}(d\log q).$$

Proof. The integer d can be computed in time $\tilde{O}(d \log q)$. Without loss of generality we may suppose $d \geqslant 2$, so that $\log\log(d+1) > 0$. Since $\log n = O(\log\log d)$, Lemmas 2.15, 2.16 and 2.17 allow us to routinely compute n in time $\tilde{O}(\log\log d)$. We compute ℓ_1 as the largest integer such that $\ell_1^n \leqslant d$, in time $\tilde{O}(\log d)$, and deduce ℓ_n with additional cost $\tilde{O}(\log d)$.

The sum $L := \ell_1 + \cdots + \ell_n$ may be bounded from above as follows:

$$\begin{aligned}
L &\leqslant (n-1)(d+1)^{1/n} + \frac{d+1}{\ell_1 \cdots \ell_{n-1}} + 1 \\
&\leqslant (n-1)(d+1)^{1/n} + \frac{d+1}{((d+1)^{1/n}-1)^{n-1}} + 1 \\
&\leqslant (n-1)(d+1)^{1/n} + \frac{(d+1)^{1/n}}{(1-(d+1)^{-1/n})^{n-1}} + 1 \\
&\leqslant (n-1)(d+1)^{1/n}\left(1 + \frac{1}{(n-1)(1-(d+1)^{-1/n})^{n-1}} + \frac{1}{(n-1)(d+1)^{1/n}}\right).
\end{aligned}$$

From (6.2) we obtain

$$\begin{aligned}\log\left(\frac{1}{(1-(d+1)^{-1/n})^{n-1}}\right) &= -(n-1)\log(1-(d+1)^{-1/n})\\ &= (n-1)\,(d+1)^{-1/n}\,(1+O((d+1)^{-1/n})).\end{aligned}$$

Since $(n-1)\,(d+1)^{-1/n} = o(1)$ we deduce that

$$\frac{1}{(1-(d+1)^{-1/n})^{n-1}} = \exp((n-1)\,(d+1)^{-1/n}\,(1+O((d+1)^{-1/n}))) = 1+o(1).$$

It follows that

$$L = (n-1)\,(d+1)^{1/n}\left(1+O\left(\frac{1}{n}\right)\right), \tag{6.4}$$

whence

$$L^n = O((n-1)^n\,(d+1)). \tag{6.5}$$

We then obtain

$$\log L = \frac{\log(d+1)}{n} + \log(n-1) + O\left(\frac{1}{n}\right) = \frac{\log(d+1)}{n}\left(1+O\left(\frac{n\log n}{\log(d+1)}\right)\right)$$

and

$$\begin{aligned}n\log\left(1+O\left(\frac{n\log n}{\log(d+1)}\right)\right) &= \frac{n^2\log n}{\log(d+1)}\left(1+O\left(\frac{n\log n}{\log(d+1)}\right)\right)\\ &= O\left(\frac{\left(\delta^2\frac{\log(d+1)}{\log\log(d+1)}\right)\log\log(d+1)}{\log(d+1)}\right)\\ &= O(1),\end{aligned}$$

which imply that $(\log L)^n = O\left(\left(\frac{\log(d+1)}{n}\right)^n\right)$. Combined with (6.5), this yields

$$\begin{aligned}(L\log L)^n &= O\left((n-1)^n\,(d+1)\left(\frac{\log(d+1)}{n}\right)^n\right)\\ &= O((d+1)\,(\log(d+1))^n)\\ &= O\left((d+1)\exp\left(\log\log(d+1)\left(\delta\left(\frac{\log(d+1)}{\log\log(d+1)}\right)^{1/2}+1\right)\right)\right)\\ &= O\left((d+1)^{1+\frac{\log\log(d+1)}{\log(d+1)}\left(\delta\left(\frac{\log(d+1)}{(\log\log(d+1))}\right)^{1/2}+1\right)}\right)\\ &= O\left((d+1)^{1+\frac{\delta^2}{n}+O\left(\frac{\log\log(d+1)}{\log(d+1)}\right)}\right).\end{aligned}$$

On the other hand, thanks to (6.4), we have

$$N = D\,(d-1)+1 \leqslant L\,(d+1)+1 = O(n\,(d+1)^{1+1/n}). \tag{6.6}$$

First we handle the case when $q \geqslant N$. We may generate N pairwise distinct $\gamma_1,\ldots,\gamma_N \in \mathbb{F}_q$ in time $N\tilde{O}(\log q) = (d+1)^{1/n}\tilde{O}(d\log q)$ and use Algorithm 6.1. Step 5 takes time

$$(1+\eta(\ell,L-n,p,k))^n\,(N+(L\log L)^n)\,\tilde{O}(L^7\log q),$$

by Theorem 4.4. This bound simplifies into

$$(1+c)^n\,(d+1)^{\frac{\delta^2+7}{n}+O\left(\frac{\log\log(d+1)}{\log(d+1)}\right)}\,\tilde{O}(d\log q),$$

for some constant $c>0$. Notice that

$$\frac{\delta^2+7}{n}=\left(\delta+\frac{7}{\delta}\right)\left(\frac{\log\log(d+1)}{\log(d+1)}\right)^{1/2}\left(1+O\left(\left(\frac{\log\log(d+1)}{\log(d+1)}\right)^{1/2}\right)\right),$$

so $\delta=\sqrt{7}$ minimizes $\delta+\frac{7}{\delta}$. Now

$$(1+c)^n=(d+1)^{\frac{n\log(1+c)}{\log(d+1)}}=(d+1)^{\left(\left(\frac{\log(d+1)}{\log\log(d+1)}\right)^{1/2}+O(1)\right)\frac{\delta\log(1+c)}{\log(d+1)}}=(d+1)^{O\left(\frac{1}{(\log(d+1)\log\log(d+1))^{1/2}}\right)},$$

so step 5 takes

$$(d+1)^{2\sqrt{7}\left(\frac{\log\log(d+1)}{\log(d+1)}\right)^{1/2}\left(1+O\left(\frac{1}{\log\log(d+1)}\right)\right)}\tilde{O}(d\log q).$$

Step 2 of Algorithm 6.1 takes time $O(n\,\ell_1\cdots\ell_n\log q)=\tilde{O}(d\log q)$ by Proposition 5.1 and (6.3). The cost for other steps, as analyzed by Proposition 6.1, amounts to

$$O(nL\mathsf{M}(d)\log d+\mathsf{M}(Ld)\log(Ld))$$

operations in $\mathbb{F}_q$, which simplifies into $\tilde{O}(d^{1+1/n}\log q)$. We are done with the case $q\geqslant N$.

It remains to deal with the case $q<N$. Basically we construct a suitable algebraic extension $\mathbb{F}_{q^e}$ of $\mathbb{F}_q$ and run Algorithm 6.1 over $\mathbb{F}_{q^e}$ instead of $\mathbb{F}_q$. In fact we need $q^e\geqslant N$ to hold. We compute the first integer $\check{e}$ such that $q^{\check{e}}\geqslant N$, in time $\tilde{O}(\log N)=\tilde{O}(\log d)$, so we have $\check{e}=O(\log N)$. We next compute the smallest integer $e\geqslant\check{e}$ that its coprime with k. Since $e=\check{e}+O(k)=O(\log N)$, this takes time $k\,\tilde{O}(\log(\check{e}+k))=\tilde{O}(\log N)=\tilde{O}(\log d)$. We proceed with the construction of an irreducible polynomial $\eta\in\mathbb{F}_p[y]$ of degree e. Using [43, Theorem 3.2], this can be done in time

$$\sqrt{p}\,\tilde{O}(e^4\log^4 p)=N^{1/(2k)}\,\tilde{O}(\log^4 N)=N\tilde{O}(\log^4 d).$$

This includes the computation of the irreducible factorization of e: the primes $<e$ can be computed in time $\tilde{O}(e)$ by Lemma 3.5, so the prime factors of e may be deduced in time $\tilde{O}(e)=\tilde{O}(\log N)$.

We let $\mu(u)$ represent the monic part of the resultant $\mathrm{Res}_z(\theta(z),\eta(u-z))$ in z and we write $A(u)\,z-B(u)$ for the corresponding subresultant of degree 1 in z. It is well known that μ is irreducible and that A is invertible modulo μ (see for instance [43, Lemma 2.4]). Setting $\nu(u)=A(u)^{-1}B(u)\bmod\mu(u)$, we then have the following isomorphism:

$$\begin{aligned}\mathbb{F}_p[z,y]/(\theta(z),\eta(y)) &\to \mathbb{F}_p[u]/(\mu(u))\\ z &\mapsto \nu(u)\\ y &\mapsto u-\nu(u).\end{aligned}\tag{6.7}$$

We identify $\mathbb{F}_p[u]/(\mu(u))$ to $\mathbb{F}_{p^{ke}}\equiv\mathbb{F}_{q^e}$. We may obtain μ and ν in time $\tilde{O}(e\,k^2\log p)=\tilde{O}(\log^3 N)$ (see [35, Corollary 31] for fast algorithms, but it would be sufficient here to appeal to naive methods). An element of $\mathbb{F}_q$ represented by $a(z)\bmod\theta(z)$ may be sent into $\mathbb{F}_p[u]/(\mu(u))$ in time $\tilde{O}(e\,k^2\log p)$. For the backward conversion we simply replace u by $y+z$ and reduce modulo $\theta(z)$ and $\eta(y)$, which takes time $\tilde{O}(e^2k^2\log p)=\tilde{O}(\log^4 N)$. Consequently applying Algorithm 6.1 over $\mathbb{F}_{q^e}$ instead of $\mathbb{F}_q$ only involves and additional overhead of $\log^{O(1)} d$ in the total complexity. □

Remark 6.3. Instead of relying on Theorem 4.4 in step 5, one might wonder whether it is interesting to consider Theorem 5.5. It turns out that the same complexity analysis applies with $n\approx\delta\left(\frac{\log(d+1)}{\bar{m}\log\log(d+1)}\right)^{1/2}$, with a similar complexity result.

Remark 6.4. In practice, the case when $q < N$ at the end of the proof can be handled more efficiently by constructing irreducible polynomials by means of a faster, although probabilistic Las Vegas algorithm; see [13, chapter 14] and [43] for instance. It is also worth it to build an extension of $\mathbb{F}_q$ of smooth degree e with $\gcd(e,k)=1$, which allows the isomorphism (6.7) to be computed more efficiently [27].

7. FIELDS OF SMALL POSITIVE CHARACTERISTIC

For the case when $\mathbb{K}$ is a field of small characteristic p, Kedlaya and Umans also designed an algebraic algorithm for fast multi-point evaluation [34, Theorem 6.3]. This algorithm turns out to be somewhat more efficient than those from the previous sections. In the present section, we adapt their techniques and prove a complexity bound in terms of the total degree instead of the partial degrees. We also refine their complexity estimates for multi-point evaluation and modular composition.

The base field is written $\mathbb{F}_q$ with $q = p^k$ and p prime; we assume it is explicitly given as $\mathbb{F}_p[z]/(\theta(z))$ with θ monic and irreducible of degree k. We let $2 < \omega \leqslant 3$ represent a constant such that two $n \times n$ matrices can be multiplied with $O(n^\omega)$ ring operations.

7.1. Multi-point evaluation

We begin with p-th root extractions in $\mathbb{F}_q$. It is well known that such root extractions reduce to linear algebra *via* the so-called Pietr-Berlekamp matrix of the isomorphism $\alpha \mapsto \alpha^p$ of $\mathbb{F}_q$ in the canonical basis $1, \ldots, z^{k-1}$. Once the inverse of this matrix is known, each root extraction takes $O(k^2)$ ring operations in $\mathbb{F}_p$, and k root extractions take $O(k^\omega)$ ring operations in $\mathbb{F}_p$. We could use this strategy in this section but it would involve an extra factor $k^{\omega-1}$ in our complexity estimates. Instead, since we focus on the case when p remains small, it is worth using the following algorithm, borrowed from [34, Theorem 6.1] and originating from [37].

Algorithm 7.1

Input. $\alpha \in \mathbb{F}_p[z]_{<k}$, $\theta \in \mathbb{F}_p[z]$ monic and irreducible of degree k.

Output. $\beta \in \mathbb{F}_p[z]_{<k}$ such that $\beta^p = \alpha \bmod \theta$.

1. Expand $\theta^{p-1}(z)$ into $a_0(z^p) + a_1(z^p)\, z + \cdots + a_{p-1}(z^p)\, z^{p-1}$, where $a_j \in \mathbb{F}_p[z]_{<k}$.
2. Expand $\alpha(z)\, \theta^{p-1}(z)$ into $b_0(z^p) + b_1(z^p)\, z + \cdots + b_{p-1}(z^p)\, z^{p-1}$, where $b_j \in \mathbb{F}_p[z]_{<k}$.
3. Select i such that $a_i \neq 0$, compute $\beta := a_i^{-1} b_i \bmod \theta$, and return β.

PROPOSITION 7.1. *Algorithm 7.1 is correct and takes $O(\mathsf{M}(p\,k) + \mathsf{M}(k) \log k)$ ring operations in $\mathbb{F}_p$ plus $O(k)$ inversions in $\mathbb{F}_p$.*

Proof. The identity $\beta^p = \alpha \bmod \theta$ is equivalent to $\beta(z^p)\, \theta^{p-1}(z) = \alpha(z)\, \theta^{p-1}(z) \bmod \theta(z^p)$. For all $0 \leqslant j \leqslant p-1$ we have $\beta(z^p)\, a_j(z^p) = b_j(z^p) \bmod \theta(z^p)$ and thus $\beta\, a_i = b_i \bmod \theta$. Since θ is irreducible, a_i is invertible modulo θ. We are done with the correctness.

The computations in steps 1 and 2 take $O(\mathsf{M}(p\,k))$ ring operations in $\mathbb{F}_p$. Step 3 requires $O(p\,k + \mathsf{M}(k) \log k)$ ring operations in $\mathbb{F}_p$ plus $O(k)$ inversions in $\mathbb{F}_p$ by using fast extended gcds. □

The next algorithm, borrowed from [34, section 6], performs multivariate multi-point evaluations by reduction to the univariate case.

Algorithm 7.2

Input. $\mathbb{F}_q$ explicitly given as $\mathbb{F}_p[z]/(\theta(z))$, where θ is a monic irreducible polynomial of degree k in $\mathbb{F}_p[z]$; $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ of total degree $\leqslant d$; a sequence $\alpha_1, \ldots, \alpha_N$ of points in $\mathbb{F}_q^n$.

Output. $f(\alpha_1), \ldots, f(\alpha_N)$.

1. Compute the total degree d of f.
2. Let $h := p^c$ be minimal such that $c \in \mathbb{N}$ and $h \geqslant \max(d(n-1)+2, n+1)$.
3. Build a monic irreducible polynomial $\lambda \in \mathbb{F}_p[z]$ of degree c, and find a primitive root η of unity of maximal order $h-1$ in $\mathbb{F}_p[z]/(\lambda(z))$.
4. Reinterpret λ as an element of $\mathbb{F}_q[u]$, let $\mathbb{L} := \mathbb{F}_q[u]/(\lambda(u))$, and reinterpret η as an element of $\mathbb{L}$.
5. For i from 1 to N do
   a. Compute $\beta_i := \left(\alpha_{i,1}, \alpha_{i,2}^{1/p}, \alpha_{i,3}^{1/p^2}, \ldots, \alpha_{i,n}^{1/p^{n-1}}\right) \in \mathbb{F}_q^n$.
   b. Interpolate $g_i \in \mathbb{L}[y]_{<n}$ such that $g_i(\eta^{j-1}) = \beta_{i,j}$ for $j = 1, \ldots, n$.
6. Let $f^*(y) := f\left(y, y^h, \ldots, y^{h^{n-1}}\right)$, compute $\gamma_i(y) := f^*(g_i(y))$ modulo $E(y) := y^{h-1} - \eta$ for $i = 1, \ldots, N$.
7. Return $(\gamma_1(1), \ldots, \gamma_N(1))$.

PROPOSITION 7.2. *Algorithm 7.2 is correct. If $n \geqslant 2$, then it takes time*

$$(N + p^{n-1} n^{n-1} d^n)\, \tilde{O}(p\, n^3\, d \log q).$$

Proof. For $i = 1, \ldots, N$ and $j = 1, \ldots, n$ we write $\bar{g}_{i,j}(y) := g_i(y)^{h^{j-1}} \operatorname{rem} E(y)$. Let σ^j represent the endomorphism of $\mathbb{L}[y]$ that rises coefficients to the h^j-th power. Then

$$\bar{g}_{i,j}(y) \equiv \sigma^{j-1}(g_i)\left(y^{h^{j-1}}\right) \equiv \sigma^{j-1}(g_i)(\eta^{j-1} y) \bmod E(y).$$

In particular $\bar{g}_{i,j}(y)$ has degree $\leqslant n-1$ whence $\bar{g}_{i,j}(1) = \alpha_{i,j}$, by the assumption $h \geqslant n+1$. Now

$$\gamma_i(y) \equiv f\left(g_i(y), g_i^h(y), \ldots, g_i^{h^{n-1}}(y)\right) \equiv f(\bar{g}_{i,1}(y), \ldots, \bar{g}_{i,n}(y)) \bmod E(y).$$

Since $f(\bar{g}_{i,1}(y), \ldots, \bar{g}_{i,n}(y))$ has degree $\leqslant d(n-1) < h-1 = \deg E$, we deduce that it coincides with $\gamma_i(y)$, whence $\gamma_i(1) = f(\bar{g}_{i,1}(1), \ldots, \bar{g}_{i,n}(1)) = f(\alpha_{i,1}, \ldots, \alpha_{i,n})$. We are done with the correctness.

Step 1 takes time

$$\begin{aligned} O\left(\binom{d+n}{n} \log q + n^2 \binom{d+n}{n} \log(d+1)\right) &= O(n^2 (d+1)^n \log(d+1) \log q) \\ &= O(n^2 2^n d^n \log d \log q) \\ &= O(n^{n+2} d^n \log d \log q). \end{aligned}$$

By Lemmas 2.6 and 2.8. Steps 2 and 4 take negligible time.

By construction, we have

$$p^{c-1} \leqslant \max(d(n-1)+1, n) \leqslant n\, d,$$

$$c = O(\log(n\, d)), \quad \text{and} \quad h \leqslant p\, n\, d. \tag{7.1}$$

In step 3, the construction of λ may be done deterministically in time

$$\tilde{O}(p^{1/2} c^5) = \tilde{O}(p^{1/2} \log^5(d\, n)) = O(p\, d\, n)$$

by means of [43, Theorem 3.2] (this includes the computation of the irreducible factorization of c).

Now [44, Theorem 1.1] provides us with a universal constant γ such that there exists a monic polynomial $\eta(z)$ of degree l with $p^l < \gamma\, p\, c^7$, for which $\eta(z) \bmod \lambda(z)$ is a primitive element. It thus suffices to enumerate all the monic polynomials of increasing degrees $l = 1, 2, 3, \ldots$ and to stop as soon as one discovers a primitive one. The loop terminates after testing $O(p\,c^7)$ candidates.

Verifying that a candidate polynomial η is a primitive element for the multiplicative group of $\mathbb{F}_p[z]/(\lambda(z))$ can be done as follows. We first compute the irreducible factorization of $p^c - 1$ in time $\tilde{O}(p^{c/4})$ (see for instance [11, Theorem 1.1]). We next verify that

$$\eta^{(p^c-1)/\pi} \neq 1 \bmod \lambda$$

for each prime divisor π of $p^c - 1$ in time $\tilde{O}(c \log^2 p^c)$. Altogether, this allows us to compute η deterministically in time

$$\tilde{O}(p^{c/4}) + O(p\,c^8 \log^2 p^c) = O(p\,n\,d).$$

Now each step 5.a requires $n-1$ extractions of p-th roots in $\mathbb{F}_q$. In total, they take time

$$n\,N\,\tilde{O}(p \log q),$$

by Proposition 7.1. The total cost of step 5.b is $N\tilde{O}(n\,c \log q)$.

Since the partial degrees of f are $\leqslant d$ and since $h \geqslant d+1$, the polynomial f^* is the Kronecker substitution $K_{h,\ldots,h}(f)$, following the notation of section 5.1. The univariate polynomial f^* in step 6 may be obtained in time

$$O(n\,h^n \log q) = O(n\,(p\,n\,d)^n \log q)$$

by Proposition 5.2, thanks to (7.1) and the assumption that $n \geqslant 2$.

Then the simultaneous evaluation of f^* at N points in $\mathbb{E} = \mathbb{L}[y]/(E(y))$ takes

$$(N + \deg f^*)\, \tilde{O}(\log^2(\deg f^*)),$$

operations in $\mathbb{E}$ which corresponds to time

$$\begin{aligned} & (N + d\,h^{n-1})\, \tilde{O}(\log^2(d\,h^{n-1})\, c\,h \log q) && (\text{since } \deg f^* \leqslant d\,h^{n-1}) \\ = \; & (N + p^{n-1} n^{n-1} d^n)\, \tilde{O}(p\,n^3\,d \log q) && \text{by (7.1).} \end{aligned}$$

The final evaluations in step 7 take time $N\tilde{O}(h\,c \log q) = N\tilde{O}(p\,n\,d \log q)$. □

Remark 7.3. Notice that $\mathbb{L}$ is not necessarily a field. As an optimization, it is worth taking $e = c / \gcd(k, c)$ and build the extension $\mathbb{L} = \mathbb{F}_q[u]/(\theta(u))$ of degree e of $\mathbb{F}_q$. Then we compute a primitive root η in $\mathbb{L}$ of order $h-1$. In practice, it is also worth using faster probabilistic algorithms in step 3.

7.2. Modular composition

Let us now reanalyze the complexity of Algorithm 6.1 when using Algorithm 7.2 for multi-point evaluations. For simplicity we assume that p is a fixed prime number, so the constant hidden in the "O" below actually depend on p. This time, we let n be an integer such that

$$\left| n - \left(\frac{\log(d+1)}{\log(p \log(d+1))} \right)^{1/2} \right| < 1. \tag{7.2}$$

THEOREM 7.4. *Let p be a fixed prime number. Let f, g, h be polynomials in $\mathbb{F}_q[x]$ such that h is monic of degree d and f, g have degrees $<d$. We assume that $\mathbb{F}_q$ is given as $\mathbb{F}_p[z]/(\theta(z))$ with θ monic irreducible of degree k. Then $f \circ g \bmod h$ may be computed in time*

$$(d+1)^{2\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right)}\tilde{O}(d\log q).$$

Proof. We adapt the proof of Theorem 6.2 and use (7.2) for the value of n instead of (6.1). Here, it is important for p to be fixed, in order to benefit from the same kind of asymptotic expansions as in the case of Theorem 6.2:

$$n=\left(\frac{\log(d+1)}{\log(p\log(d+1))}\right)^{1/2}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right)$$

and

$$\frac{1}{n}=\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right).$$

In particular $(d+1)^{1/n}$ still tends to $+\infty$ for large values of d.

The main change is the use of Proposition 7.2 to obtain the following cost for step 5 of Algorithm 6.1:

$$(N+p^{n-1}n^{n-1}L^n)\,\tilde{O}(n^3 L\log q).$$

By using (6.4), (6.5), (6.6), and the fact that $n=O(\log^{1/2} d)$, the latter cost is bounded by

$$(n\,(d+1)^{2/n}+p^{n-1}n^{2n-1}\,(d+1)^{1/n})\,\tilde{O}(d\log q).$$

We then need to upper bound the term

$$\begin{aligned} p^{n-1}n^{2n-1}\,(d+1)^{1/n} &\leqslant (d+1)^{\frac{1}{n}+\frac{n\log p}{\log(d+1)}+\frac{2n\log n}{\log(d+1)}} \\ &\leqslant (d+1)^{\frac{1}{n}\left(1+\frac{n^2\log(pn^2)}{\log(d+1)}\right)}. \end{aligned}$$

Now for sufficiently large d, we have

$$\begin{aligned} n^2\log(pn^2) &= \frac{\log(d+1)}{\log(p\log(d+1))}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right) \\ &\quad\times\log\left(\frac{p\log(d+1)}{\log(p\log(d+1))}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right)\right) \\ &\leqslant \log(d+1)\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right). \end{aligned}$$

It follows that

$$p^{n-1}n^{2n-1}\,(d+1)^{1/n}\leqslant(d+1)^{2\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right)}.$$

On the other hand, we have

$$(d+1)^{2/n}=(d+1)^{2\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\left(1+O\left(\left(\frac{\log(p\log(d+1))}{\log(d+1)}\right)^{1/2}\right)\right)},$$

which concludes the proof. □

n	6	7	8	9	10	11	12	13	14	15	16	18	21	26	43
$\bar{\ell} = \lceil n^{7/(n-5)} \rceil$	279936	908	129	47	26	17	12	10	8	7	6	5	4	3	2
$\lfloor \log_{10} \bar{\ell}^n \rceil$	32	21	17	15	14	14	13	13	13	13	12	13	13	12	13

Table 8.1. Efficiency threshold orders for the complexity bound of Theorem 3.10.

For small fixed values of p, Theorem 7.4 therefore improves upon Theorem 6.2. This happens roughly whenever $4 \log(p \log(d+1)) \ll 28 \log\log(d+1)$, which rewrites into $p \ll \log^6(d+1)$. A more precise complexity analysis in terms of the parameter p could be developed under the latter condition.

8. CONCLUSION

An important application of the present results concerns polynomial system solving, for which we prove new complexity bounds in [28]: the key algorithms are the Kronecker solver [14] and fast multivariate modular composition. For the latter problem, we mostly follow the strategy deployed in the proof of Proposition 5.7.

Besides technical adaptations to Turing machines and various refinements within the asymptotic complexity bounds, our main improvements upon Kedlaya and Umans' algorithms concern complexity analyses in terms of the total degree, and the way we appeal to the naive evaluation algorithm as a fallback in Theorem 3.10. In particular, our complexity bounds are quasi-optimal with respect to the bit size of the elements in the ground ring.

Another major motivation behind our work is to understand how relevant the new complexity bounds are for practical implementations. Unfortunately, the input sizes for which our optimized variants of Kedlaya and Umans' algorithms become faster than previous approaches still seem to be extremely large. It is instructive to discuss, even in very informal terms, the orders of magnitude of the cross-over points.

In the univariate case $n = 1$ over $\mathbb{Z}/r\,\mathbb{Z}$, fast algorithms for multi-point evaluation allow for an average evaluation cost per point of $\tilde{O}(\log^2 n \log r)$, by means of the classical subproduct tree technique [13, chapter 12], and as soon as the number of points is of the order of the degree. In the same spirit, in order to minimize the average cost per evaluation point, Theorem 3.10 indicates that it is favorable to take N of the order of $(\varphi \log \varphi)^n$ (that is larger than the cardinality of the support of f).

To simplify the discussion, we discard the factor $(1 + \varepsilon(\ell, d, r))^n$ occurring in Theorem 3.10, and we focus on the case when $\ell_1 = \cdots = \ell_n = \bar{\ell}$ and $d = n\,(\bar{\ell} - 1)$. So when $N = (\varphi \log \varphi)^n$, the average cost per point roughly becomes $\tilde{O}(n^2 \varphi^5 \log r)$. Recall that this average cost is $\bar{\ell}^n \tilde{O}(\log r)$ with the naive approach. The ratio between both bounds is therefore of the order $\tilde{O}(n^7 \bar{\ell}^5)\ (\log\log r)^{O(1)} / \bar{\ell}^n$. In Table 8.1 we report on the first values of $\bar{\ell}$ such that $(n^7 \bar{\ell}^5) / \bar{\ell}^n \leqslant 1$, namely $\bar{\ell} = \lceil n^{7/(n-5)} \rceil$, along with the closest integer to $\log_{10} \bar{\ell}^n$. Whenever the same value of $\bar{\ell}$ is encountered for different values of n, we only display the case with smallest $\bar{\ell}^n$. We observe that the sizes of the corresponding input polynomials are not realistic for a common workstation.

The above factor $n^2 \varphi^5$ corresponds to the value $t = 5$ in Algorithm 3.3. In practice it turns out to be more interesting to use smaller values for t. In fact, it is worth using Algorithm 3.3 with $t = 3$ whenever $\log r < \varphi < r$, since

$$\begin{aligned} \mathsf{B}^{\circ 3}(r) &= (1 + \mu(\varphi))\, \varphi \log \mathsf{B}^{\circ 2}(r) \\ &= (1 + O(\mu(\varphi)))\, \varphi \log(O(\varphi \log \varphi)) \\ &= \left(1 + O\left(\frac{\log\log \varphi}{\log \varphi}\right)\right) \varphi \log \varphi, \end{aligned}$$

n	6	7	8	9	10	11	12	16	27
$\bar{\ell} = \lceil n^{5/(n-3)} \rceil$	20	12	8	7	6	5	4	3	2
$\lfloor \log_{10} \bar{\ell}^n \rfloor$	8	8	7	8	8	8	7	8	8

Table 8.2. Efficiency threshold orders for the complexity bound of Proposition 3.7 with $t = 3$.

and the cost given in Proposition 3.7 drops to $(N + \mathsf{B}^{\circ 3}(r)^n)\ \tilde{O}(n^2\ \varphi^3 \log r)$. Table 8.2 displays the first resulting values of $\bar{\ell}$ for which $(n^5\, \bar{\ell}^3) / \bar{\ell}^n \leqslant 1$. Considering for instance that r is a 64 bit integer, we may thus use Algorithm 3.3 with $t = 3$ whenever $\log r \simeq 44 < \varphi < 2^{64}$. Consequently, for input polynomial sizes of about 100 MB, the fast algorithm might start to be of interest. Yet, the orders of magnitude considered here are quite optimistic and we expect the actual thresholds to be larger.

In small positive characteristic $p > 0$, Proposition 7.2 seems more promising for practical purposes. For $N \geqslant p^{n-1}\ n^{n-1}\ d^n$, the average cost per evaluation point drops to $\tilde{O}(p\ n^3\ d \log q)$. The efficiency ratio with respect to the naive algorithm is thus of the order $p\, n^3 d / \binom{d+n}{n}$. For instance, with $p = 2$ and $n = 2$, this ratio rewrites into $32\ d\ /\ ((d+2)\ (d+1))$. Consequently, Algorithm 7.2 might be relevant in practice for input data of a few kilobytes. However we are not aware of such an efficient implementation.

For the above reasons, faster software implementations of multivariate multi-point evaluation and modular composition remain major challenges. At least, we hope that our new detailed complexity bounds will stimulate more theoretical and practical research in this area. For example, is it possible to decrease the contribution of $(\varphi \log \varphi)^n$ in Theorem 3.10? Or, still in Theorem 3.10, could one decrease the exponent 5 of φ? Is it possible to improve upon the constant 28 in Theorem 6.2?

BIBLIOGRAPHY

[1] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse multivariate polynomial interpolation of straight-line programs. *J. Symbolic Comput.*, 75:4–24, 2016.

[2] J. Barkley Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math*, 6:64–94, 1962.

[3] D. J. Bernstein. Composing power series over a finite ring in essentially linear time. *J. Symbolic Comput.*, 26(3):339–341, 1998.

[4] L. I. Bluestein. A linear filtering approach to the computation of discrete Fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.

[5] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier–Manin operator. *SIAM J. Comput.*, 36:1777–1806, 2007.

[6] 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.

[7] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

[8] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. ACM*, 23(2):242–251, 1976.

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

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

[11] E. Costa and D. Harvey. Faster deterministic integer factorization. *Math. Comp.*, 83(285):339–345, 2014.

[12] P. Dusart. Estimates of some functions over primes without R.H. Technical report, ArXiv, 2010. https://arxiv.org/abs/1002.0442.

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

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

[15] 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.

[16] D. Harvey and J. van der Hoeven. Faster integer multiplication using plain vanilla FFT primes. *Math. Comp.*, 2018. https://doi.org/10.1090/mcom/3328.

[17] D. Harvey and J. van der Hoeven. Faster integer multiplication using short lattice vectors. Technical report, ArXiv, 2018. http://arxiv.org/abs/1802.07932.

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

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

[20] J. van der Hoeven. Relax, but don't be too lazy. *J. Symbolic Comput.*, 34(6):479–542, 2002.

[21] J. van der Hoeven. The truncated Fourier transform and applications. In J. Gutierrez, editor, *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, ISSAC '00, pages 290–296, New York, NY, USA, 2004. ACM.

[22] J. van der Hoeven. Fast composition of numeric power series. Technical Report 2008-09, Université Paris-Sud, Orsay, France, 2008.

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

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

[25] J. van der Hoeven and G. Lecerf. Composition modulo powers of polynomials. In M. Blurr, editor, *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 445–452, New York, NY, USA, 2017. ACM.

[26] 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.

[27] J. van der Hoeven and G. Lecerf. Modular composition via factorization. *J. Complexity*, 2018. https://doi.org/10.1016/j.jco.2018.05.002.

[28] J. van der Hoeven and G. Lecerf. On the complexity exponent of polynomial system solving. 2018.

[29] J. van der Hoeven and É. Schost. Multi-point evaluation in higher dimensions. *Appl. Alg. Eng. Comm. Comp.*, 24(1):37–52, 2013.

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

[31] F. Johansson. A fast algorithm for reversion of power series. *Math. Comp.*, 84:475–484, 2015.

[32] E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 184–188, New York, NY, USA, 1997. ACM.

[33] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *FOCS'08: IEEE Conference on Foundations of Computer Science*, pages 146–155, Washington, DC, USA, 2008. IEEE Computer Society.

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

[35] G. Lecerf. On the complexity of the Lickteig–Roy subresultant algorithm. *J. Symbolic Comput.*, 2018. https://doi.org/10.1016/j.jsc.2018.04.017.

[36] M. Nüsken and M. Ziegler. Fast multipoint evaluation of bivariate polynomials. In S. Albers and T. Radzik, editors, *Algorithms – ESA 2004. 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004*, volume 3221 of *Lect. Notes Comput. Sci.*, pages 544–555. Springer Berlin Heidelberg, 2004.

[37] D. Panario and D. Thomson. Efficient pth root computations in finite fields of characteristic p. *Des. Codes Cryptogr.*, 50(3):351–358, 2009.

[38] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[39] 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.

[40] P. Ritzmann. A fast numerical algorithm for the composition of power series with complex coefficients. *Theoret. Comput. Sci.*, 44:1–16, 1986.

[41] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1(2):139–144, 1971.

[42] A. Schönhage, A. F. W. Grotefeld, and E. Vetter. *Fast algorithms: A multitape Turing machine implementation*. B. I. Wissenschaftsverlag, Mannheim, 1994.

[43] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54(189):435–447, 1990.

[44] V. Shoup. Searching for primitive roots in finite fields. *Math. Comp.*, 58:369–380, 1992.