Ultimate complexity for numerical algorithms

Joris van Der Hoeven, Grégoire Lecerf

Ultimate complexity for numerical algorithms

Joris van der Hoeven
CNRS, École polytechnique, Institut Polytechnique de Paris
Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161)
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France
Email: vdhoeven@lix.polytechnique.fr

Grégoire Lecerf
CNRS, École polytechnique, Institut Polytechnique de Paris
Laboratoire d'informatique de l'École polytechnique (LIX, UMR 7161)
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France
Email: lecerf@lix.polytechnique.fr

Abstract

Most numerical algorithms are designed for single or double precision floating point arithmetic, and their complexity is measured in terms of the total number of floating point operations. The resolution of problems with high condition numbers (e.g. when approaching a singularity or degeneracy) may require higher working precisions, in which case it is important to take the precision into account when doing complexity analyses. In this paper, we propose a new "ultimate complexity" model, which focuses on analyzing the cost of numerical algorithms for "sufficiently large" precisions. As an example application we will present an ultimately softly linear time algorithm for modular composition of univariate polynomials.

1 Introduction

The total number of floating point operations is a good measure for the complexity of numerical algorithms that operate at a fixed working precision. However, for certain ill-conditioned problems it may be necessary to use higher working precisions. In order to analyze the complexity to solve such problems, it is crucial to take into account the working precision as an additional parameter.

Multiple precision algorithms are particularly important in the area of symbolic-numeric computation. Two examples of applications are polynomial factorization and lattice reduction. Analyzing

This document has been written using GNU TEXMACS [15].

the bit complexities of multiple precision algorithms is often challenging. Even in the case of a simple operation such as multiplication, there has been recent progress [11]. The complexities of other basic operations such as division, square roots, and the evaluation of elementary functions have also been widely studied; see for instance [5, 25].

For high level algorithms, sharp complexity analyses become tedious. This happens for instance in the area of "numerical algebraic geometry" when analyzing the complexity of homotopy continuation methods: condition numbers are rather intricate and worst case complexity bounds do not well reflect practical timings that can be observed with generic input data. This motivated the introduction of new average complexity models and lead to an active new area of research; see for instance [2, 7].

Studying the complexity of algorithms has various benefits. First, it is useful to predict practical running times of implementations. Secondly, it helps to determine which part of an algorithm consumes most of the time. Thirdly, for algorithms that are harder to implemented, complexity bounds are a way to estimate and compare algorithmic advances.

The present paper is devoted to a new "ultimate complexity" model. It focuses on the asymptotic complexity of a numerical algorithm when the working precision becomes sufficiently large. This means in particular that we are in the asymptotic regime where the working precision becomes arbitrarily large with respect to other parameters for the algorithm, such as the degree of a polynomial or the size of a matrix. The ultimate complexity model allows for rather simple and quick analyses. In particular, it is easy to transfer algebraic complexity results over an abstract effective field $\mathbb{K}$ into ultimate complexity results over the complex numbers. However, it does not provide any information on how large the precision should be in order for the asymptotic bounds to be observable in practice.

As a first example, let us consider the problem of multiplying two $n \times n$ matrices with floating point entries of bit size $< p$. The usual cost is $O(n^\omega)$ floating point operations, for some feasible exponent $\omega > 2$ for matrix multiplication; in [24] it has been shown that one may take $\omega < 2.3729$. Using the above transfer principle, this leads to an ultimate bit complexity bound of $O(n^\omega p \log p)$, where we used the recent result from [11] that p-bit integers can be multiplied in time $O(p \log p)$. In practical applications, one is often interested in the case when $n \gg p$, in which the bit-complexity rather becomes linear in p. But this is not what the ultimate complexity model does: it rather assumes that $p \gg n$, and even $p \gg \left(\exp \circ \overset{k}{\cdots} \circ \exp\right)(n)$ for any k. Under this assumption, we actually have the sharper ultimate bit complexity bound $O(n^2 p \log p)$, which is proved using FFT techniques [10].

As a second example, which will be detailed in the sequel, consider the computation of the complex roots of a separable polynomial f of degree n over the complex numbers: once initial approximations of the roots are known with a sufficiently large precision, the roots can be refined in softly linear time by means of fast multi-point evaluations and Newton iterations. This implies that the ultimate complexity is softly linear, even if a rather slow algorithm is used to compute the initial approximations.

In sections 2 and 3, we recall basic results about *ball arithmetic* [13] and *straight-line programs* [6]. Section 4 is devoted to the new *ultimate complexity model*. In section 5 we illustrate the ultimate complexity model with *modular composition*. Modular composition consists in computing the composition of two univariate polynomials modulo a third one. For polynomials with coefficients in a finite field, Kedlaya and Umans proved in 2008 that the theoretical complexity for performing this task could be made arbitrarily close to linear [21, 22]. Unfortunately, beyond its major theoretical

impact, this result has not led to practically faster implementations yet; see [17]. In this paper, we explore the particular case when the ground field is the field of computable complex numbers and improve previously known results in this special case.

2 Ball arithmetic

2.1 Bit complexity

In the *bit complexity model*, the running time of an algorithm is analyzed in terms of the number of bit operations that need to be performed. In this paper, we always assume that this is done on a Turing machine with a finite and sufficiently large number of tapes [28].

Multiplication is a central operation in both models. We write $\mathsf{I}(p)$ for a function that bounds the bit-cost of an algorithm which multiplies two integers of bit sizes at most p, for the usual binary representation. The best known bound [11] for $\mathsf{I}(p)$ is $O(p \log p) = \tilde{O}(p)$. Here, the *soft-Oh* notation $f(p) = \tilde{O}(g(p))$ means that $f(p) = g(p)(\log g(p))^{O(1)}$; we refer the reader to [9, Chapter 25, Section 7] for technical details. We make the customary assumption that $\mathsf{I}(p)/p$ is non-decreasing. Notice that this assumption implies the *super-additivity* of I, namely $\mathsf{I}(p_1) + \mathsf{I}(p_2) \leqslant \mathsf{I}(p_1 + p_2)$ for all $p_1 \geqslant 0$ and $p_2 \geqslant 0$.

2.2 Fixed point numbers

Let a be a real number, we write $\lfloor a \rfloor$ for the largest integer less or equal to a and $\lfloor a \rceil := \lfloor a + 1/2 \rfloor$ for the closest integer to a.

Given a precision $p \in \mathbb{N}$, we denote by $\mathbb{D}_p = \mathbb{Z}2^{-p}$ the set of *fixed point numbers* with p binary digits after the dot. This set $\mathbb{D}_p$ is clearly stable under addition and subtraction. We can also define approximate multiplication $\times_p$ on $\mathbb{D}_p$ using $x \times_p y = \lfloor 2^p xy \rceil 2^{-p}$, so $|x \times_p y - xy| \leqslant 2^{-p-1}$ for all $x, y \in \mathbb{D}_p$.

For any fixed constant $K > 0$ and $x, y \in \mathbb{D}_p \cap [-K, K]$, we notice that $x + y$ and $x - y$ can be computed in time $O(p)$, whereas $x \times_p y$ can be computed in time $\mathsf{I}(p) + O(p)$. Similarly, one may define an approximate inversion ι_p on $\mathbb{D}_p^{\neq} := \mathbb{D}_p \setminus \{0\}$ by $\iota_p(x) = \lfloor 2^p x^{-1} \rceil 2^{-p}$. For any fixed constant $K > 0$ and $x \in \mathbb{D}_p^{\neq} \cap [-K, K]$, we may compute $\iota_p(x)$ in time $O(\mathsf{I}(p))$.

Remark 1 In numerical algorithms, floating point arithmetic is often preferred with respect to fixed point arithmetic. From our perspective of ultimate complexity, it is interesting to notice that both formalisms are actually equivalent, except at zero. Indeed, it suffices to take the bit precision large enough so that it exceeds the absolute value of the exponent of any floating point number involved in the computation. For technical reasons, it is more convenient to use fixed point arithmetic in this paper.

2.3 Fixed point ball arithmetic

Ball arithmetic is used for providing reliable error bounds for approximate computations. A *ball* is a set $\mathcal{B}(c, r) = \{z \in \mathbb{R} : |z - c| \leqslant r\}$ with $c \in \mathbb{R}$ and $r \in \mathbb{R}^{\geqslant}$. From the computational point of view, we represent such balls by their centers c and radii r. We denote by $\mathbb{B}_p$ the set of balls with centers in $\mathbb{D}_p$ and radii in $\mathbb{D}_p^{\geqslant}$. Given vectors $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = (\mathcal{B}(c_1, r_1), \ldots, \mathcal{B}(c_n, r_n)) \in \mathbb{B}_p^n$ we write $x \in \boldsymbol{x}$ to mean $x_1 \in \boldsymbol{x}_1 \wedge \cdots \wedge x_n \in \boldsymbol{x}_n$, and we also set $\mathrm{rad}(\boldsymbol{x}) := \max(r_1, \ldots, r_n)$.

Let D be an open subset of $\mathbb{R}^n$. We say that $\boldsymbol{D}_p \subseteq \mathbb{B}_p^n$ is a *domain lift* at precision p if $\boldsymbol{x} \subseteq D$ for all $\boldsymbol{x} \in \boldsymbol{D}_p$. The maximal such lift is given by $\boldsymbol{D}_p = \{\boldsymbol{x} \in \mathbb{B}_p^n : \boldsymbol{x} \subseteq D\}$. Given a function $f : D \to \mathbb{R}^m$, a *ball lift* of f at precision p is a function $\boldsymbol{f}_p : \boldsymbol{D}_p \to \mathbb{B}_p^m$, where $\boldsymbol{D}_p = \operatorname{dom} \boldsymbol{f}_p$ is a domain lift of D at precision p, that satisfies the *inclusion property*: for any $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \boldsymbol{D}_p^n$ and $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, we have

$$x \in \boldsymbol{x} \quad \Longrightarrow \quad f(x) \in \boldsymbol{f}_p(\boldsymbol{x}).$$

A *ball lift* $\boldsymbol{f}$ of f is a computable sequence $(\boldsymbol{f}_p)_{p \in \mathbb{N}}$ of ball lifts at every precision such that for any sequence $(\boldsymbol{x}_p)_{p \in \mathbb{N}}$ with $\boldsymbol{x}_p \in \operatorname{dom} \boldsymbol{f}_p$, we have

$$\lim_{p \to \infty} \operatorname{rad}(\boldsymbol{x}_p) = 0 \quad \wedge \quad \bigcap_{p \in \mathbb{N}} \boldsymbol{x}_p \neq \emptyset \quad \Longrightarrow \quad \lim_{p \to \infty} \operatorname{rad}(\boldsymbol{f}_p(\boldsymbol{x}_p)) = 0.$$

This condition implies the following:

$$\lim_{p \to \infty} \operatorname{rad}(\boldsymbol{x}_p) = 0 \quad \wedge \quad \bigcap_{p \in \mathbb{N}} \boldsymbol{x}_p = \{x\} \quad \Longrightarrow \quad \bigcap_{p \in \mathbb{N}} \boldsymbol{f}_p(\boldsymbol{x}_p) = \{f(x)\}.$$

We say that $\boldsymbol{f}$ is *maximal* if $\operatorname{dom} \boldsymbol{f}_p$ is the maximal domain lift for each p. Notice that a function f must be continuous in order to admit a maximal ball lift.

The following formulas define maximal ball lifts $\oplus_p$, $\ominus_p$ and $\otimes_p$ at precision p for the ring operations $+$, $-$ and $\times$:

$$\begin{aligned}
\mathcal{B}(a, r) \oplus_p \mathcal{B}(b, s) &:= \mathcal{B}(a + b, r + s) \\
\mathcal{B}(a, r) \ominus_p \mathcal{B}(b, s) &:= \mathcal{B}(a - b, r + s) \\
\mathcal{B}(a, r) \otimes_p \mathcal{B}(b, s) &:= \mathcal{B}(a \times_p b, (|a| + r) \times_p s + |b| \times_p r + 2^{1-p}).
\end{aligned}$$

The extra 2^{1-p} in the formula for multiplication is needed in order to counter the effect of rounding errors that might occur in the three multiplications $a \times_p b$, $(|a| + r) \times_p s$ and $|b| \times_p s$. For $\mathcal{B}(a, r) \in \mathbb{B}_p$ with $r < |a|$, the following formula also defines a maximal ball lift $\boldsymbol{\iota}_p$ at precision p for the inversion:

$$\boldsymbol{\iota}_p(\mathcal{B}(a, r)) := \mathcal{B}(\iota_p(a), \iota_p(|a| - r) - \iota_p(|a|) + 2^{1-p}).$$

For any fixed constant $K > 0$ and $a, r, b, s \in \mathbb{D}_p \cap [-K, K]$, we notice that $\mathcal{B}(a, r) \oplus_p \mathcal{B}(b, s)$, $\mathcal{B}(a, r) \ominus_p \mathcal{B}(b, s)$, $\mathcal{B}(a, r) \otimes_p \mathcal{B}(b, s)$ and $\boldsymbol{\iota}_p(\mathcal{B}(a, r))$ can be computed in time $O(\mathsf{I}(p))$.

Let $\boldsymbol{f}$ be the ball lift of a function $f : D \to \mathbb{R}^m$ with $D \subseteq \mathbb{R}^n$. Consider a second ball lift $\boldsymbol{g}$ of a function $g : E \to \mathbb{R}^l$ with $f(D) \subseteq E \subseteq \mathbb{R}^m$. Then we may define a ball lift $\boldsymbol{g} \circ \boldsymbol{f}$ of the composition $g \circ f : D \to \mathbb{R}^l$ as follows. For each precision p, we take $(\boldsymbol{g} \circ \boldsymbol{f})_p = \boldsymbol{g}_p \circ (\boldsymbol{f}_p)_{|D_p}$, where $(\boldsymbol{f}_p)_{|D_P}$ is the restriction of $\boldsymbol{f}_p$ to the set $D_p = \{\boldsymbol{x} \in \operatorname{dom} \boldsymbol{f}_p : \boldsymbol{f}_p(\boldsymbol{x}) \in \operatorname{dom} \boldsymbol{g}_p\}$.

We shall use ball arithmetic for the computation of complex functions $\mathbb{C}^n \to \mathbb{C}^m$ simply through the consideration of real and imaginary parts. This point of view is sufficient for the asymptotic complexity analyses of the present paper. Of course, it would be more efficient to directly compute with complex balls (i.e. balls with a complex center and a real radius), but this would involve approximate square roots and ensuing technicalities.

2.4 The Lipschitz property

Assume that we are given the ball lift $\boldsymbol{f}$ of a function $f : D \to \mathbb{R}^m$ with $D \subseteq \mathbb{R}^n$. Given a subset $U \subseteq D$ and constants $\lambda \geqslant 0, \mu \geqslant 0$, we say that the ball lift $\boldsymbol{f}$ is (λ, μ)-*Lipschitz* on U if

$$\begin{aligned}&\exists p_0 \in \mathbb{N}, \quad \exists \varrho > 0, \quad \forall p \geqslant p_0, \quad \forall \boldsymbol{x} \in \mathbb{B}_p^n,\\ &\quad \boldsymbol{x} \subseteq U \quad \wedge \quad \operatorname{rad}(\boldsymbol{x}) \leqslant \varrho \quad \Longrightarrow \quad \boldsymbol{x} \in \operatorname{dom} \boldsymbol{f}_p \quad \wedge \quad \operatorname{rad}(\boldsymbol{f}_p(\boldsymbol{x})) \leqslant \lambda \operatorname{rad}(\boldsymbol{x}) + \mu 2^{-p}.\end{aligned}$$

For instance, the ball lifts $\oplus$ and $\ominus$ of addition and subtraction are $(2, 0)$-Lipschitz on $\mathbb{R}^2$. Similarly, the ball lift $\otimes$ of multiplication is $(3\lambda, 3)$-Lipschitz on $U = \{(x, y) \in \mathbb{R}^2 : |x| \leqslant \lambda, |y| \leqslant \lambda\}$ (by taking $\varrho = \lambda$), whereas the ball lift $\boldsymbol{\iota}$ of ι is $(\lambda, 3)$-Lipschitz on $U = \{x \in \mathbb{R} : \lambda^{-1/2} \leqslant |x|\}$.

Given $\boldsymbol{f}$ and $\lambda > 0, \mu \geqslant 0$ as above, we say that $\boldsymbol{f}$ is *locally* (λ, μ)-*Lipschitz* on U if $\boldsymbol{f}$ is (λ, μ)-Lipschitz on each compact subset of U. We define $\boldsymbol{f}$ to be λ-Lipschitz (resp. locally λ-Lipschitz) on U if there exists a constant $\mu > 0$ for which $\boldsymbol{f}$ is (λ, μ)-Lipschitz (resp. locally (λ, μ)-Lipschitz). If $\boldsymbol{f}$ is locally λ-Lipschitz on U, then it is not hard to see that f is necessarily locally Lipschitz on U, with Lipschitz constant λ. That is,

$$\forall x \in U, \quad \exists \eta > 0, \quad \forall a, b \in \mathcal{B}(x, \eta) \cap U, \qquad \|f(b) - f(a)\|_\infty \leqslant \lambda \|b - a\|_\infty.$$

In fact, the requirement that a computable ball lift $\boldsymbol{f}$ is λ-Lipschitz implies that we have a way to compute high quality error bounds. We finally define $\boldsymbol{f}$ to be Lipschitz (resp. locally Lipschitz) on U if there exists a constant $\lambda > 0$ for which $\boldsymbol{f}$ is λ-Lipschitz (resp. locally λ-Lipschitz).

Lemma 1 *Let $\boldsymbol{f}$ be a locally (λ, μ)-Lipschitz ball lift of $f : D \to \mathbb{R}^m$ on an open set U. Let $\boldsymbol{g}$ be a locally (λ', μ')-Lipschitz ball lift of $g : E \to \mathbb{R}^l$ on an open set V. If $f(D) \subseteq E$ and $f(U) \subseteq V$, then $\boldsymbol{g} \circ \boldsymbol{f}$ is a locally $(\lambda\lambda', \mu\lambda' + \mu')$-Lipschitz ball lift of $g \circ f$ on U.*

Proof Consider a compact subset $C \subseteq U$. Since this implies $f(C)$ to be a compact subset of $f(U) \subseteq V$, it follows that there exists an $\varepsilon > 0$ such that $f(C) + \mathcal{B}(0, \varepsilon) \subseteq V$. Let $p_0 \in \mathbb{N}$, $0 < \varrho < (\varepsilon - \mu 2^{-p})/\lambda$ and $0 < \varrho'$ be such that for any $p \geqslant p_0$, $\boldsymbol{x} \in \mathbb{B}_p^n$ and $\boldsymbol{y} \in \mathbb{B}_p^m$, we have

$$\begin{aligned}\boldsymbol{x} \subseteq C \wedge \operatorname{rad}(\boldsymbol{x}) \leqslant \varrho \quad &\Longrightarrow \quad \boldsymbol{x} \in \operatorname{dom} \boldsymbol{f}_p \wedge \operatorname{rad}(\boldsymbol{f}_p(\boldsymbol{x})) \leqslant \lambda \operatorname{rad}(\boldsymbol{x}) + \mu 2^{-p} < \varepsilon\\ (\boldsymbol{y} \subseteq f(C) + \mathcal{B}(0, \varepsilon)) \wedge \operatorname{rad}(\boldsymbol{y}) \leqslant \varrho' \quad &\Longrightarrow \quad \boldsymbol{y} \in \operatorname{dom} \boldsymbol{g}_p \wedge \operatorname{rad}(\boldsymbol{g}_p(\boldsymbol{y})) \leqslant \lambda' \operatorname{rad}(\boldsymbol{y}) + \mu' 2^{-p}.\end{aligned}$$

Given $x \in \mathbb{B}_p^n$ with $\boldsymbol{x} \subseteq C$ and $\operatorname{rad}(\boldsymbol{x}) \leqslant \varrho$, it follows that $\boldsymbol{y} := \boldsymbol{f}_p(\boldsymbol{x})$ satisfies $\operatorname{rad}(\boldsymbol{y}) < \varepsilon$, whence $\boldsymbol{y} \subseteq f(C) + \mathcal{B}(0, \varepsilon)$. If we also assume that $\operatorname{rad}(\boldsymbol{x}) \leqslant (\varrho' - \mu 2^{-p})/\lambda$, then it also follows that $\operatorname{rad}(\boldsymbol{y}) \leqslant \varrho'$, whence $\boldsymbol{y} \in \operatorname{dom} \boldsymbol{g}_p$ and $\operatorname{rad}(\boldsymbol{g}_p(\boldsymbol{y})) \leqslant \lambda'(\lambda \operatorname{rad}(\boldsymbol{x}) + \mu 2^{-p}) + \mu' 2^{-p} = \lambda\lambda' \operatorname{rad}(\boldsymbol{x}) + (\mu\lambda' + \mu') 2^{-p}$. In other words, if $\boldsymbol{x} \subseteq C$ and $\operatorname{rad}(\boldsymbol{x}) \leqslant \min(\varrho, (\varrho' - \mu 2^{-p})/\lambda)$, then $\boldsymbol{x} \in \operatorname{dom}(\boldsymbol{g}_p \circ \boldsymbol{f}_p)$ and $\operatorname{rad}((\boldsymbol{g}_p \circ \boldsymbol{f}_p)(\boldsymbol{x})) \leqslant \lambda\lambda' \operatorname{rad}(\boldsymbol{x}) + (\mu\lambda' + \mu') 2^{-p}$. □

3 Straight-line programs

Informally speaking, a straight-line program is a sequence of programing instructions that does not contain any loop or branching. Each instruction applies a single elementary operation to constants and values stored in variables. The result of the instruction is then stored into a variable. A detailed presentation of straight-line programs can be found in the book [6]. From the mathematical point

of view, a straight-line program can be regarded as a data structure that encodes of a composition of elementary operations.

For straight-line programs over rings (and fields), elementary operations are usually the arithmetic ones: addition, subtraction, products (and inversions). Our application to modular composition in section 5 only requires these operations. However in the present numerical context, it is relevant to allow us for wider sets of elementary operations such as exponentials, logarithms, trigonometric functions, etc. The following paragraphs thus formalize straight-line programs for a wide class of numerical algorithms. We take care of definition domains and allow for changes of evaluation domains.

A *signature* is a finite or countable set of function symbols $\mathcal{F}$ together with an arity $r_f \in \mathbb{N}$ for each $f \in \mathcal{F}$. A *model* for $\mathcal{F}$ is a set K together with a function $f_K : U_f \to K$ with $U_f \subseteq K^{r_f}$ for each $k \in \mathcal{F}$. If K is a topological space, then U_f is required to be an open subset of K^{r_f}. Let $\mathcal{V}$ be a countable and ordered set of variable symbols.

A *straight-line program* (SLP) Γ with signature $\mathcal{F}$ is a sequence $\Gamma_1, \ldots, \Gamma_\ell$ of instructions of the form

$$\Gamma_k \quad \equiv \quad X_k := f_k \left(Y_{k,1}, \ldots, Y_{k,r_{f_k}} \right),$$

where $f_k \in \mathcal{F}$ and $X_k, Y_{k,1}, \ldots, Y_{k,r_{f_k}} \in \mathcal{V}$, together with a subset $\mathcal{O}_\Gamma \subseteq \{X_1, \ldots, X_\ell\}$ of *output variables*. Variables that appear for the first time in the sequence in the right-hand side of an instruction are called *input variables*. We denote by $\mathcal{I}_\Gamma$ the set of input variables. The number ℓ is called the *length* of Γ.

There exist unique sequences $I_1 < \cdots < I_n$ and $O_1 < \cdots < O_m$ with $\mathcal{I}_\Gamma = \{I_1, \ldots, I_n\}$ and $\mathcal{O}_\Gamma = \{O_1, \ldots, O_m\}$. Given a model K of $\mathcal{F}$ we can run Γ for inputs in K, provided that the arguments $Y_{k,1}, \ldots, Y_{k,r_{f_k}}$ are always in the domain of f_k when executing the instruction Γ_k. Let $D_{\Gamma,K}$ be the set of tuples $I = (I_1, \ldots, I_n) \in K^n$ on which Γ can be run. Given $I \in K^n$, let $\Gamma_K(I) \in K^m$ denote the value of $(O_1, \ldots, O_m)$ at the end of the program. Hence Γ gives rise to a function $\Gamma_K : D_{\Gamma,K} \to K^m$.

Now assume that $(\mathbb{R}, (f_\mathbb{R})_{f \in \mathcal{F}})$ is a model for $\mathcal{F}$ and that we are given a ball lift $\boldsymbol{f}$ of $f_\mathbb{R}$ for each $f \in \mathcal{F}$. Then $\mathbb{B}_p$ is also a model for $\mathcal{F}$ at each precision p, by taking $f_{\mathbb{B}_p} = \boldsymbol{f}_p$ for each $f \in \mathcal{F}$. Consequently, any SLP Γ as above gives rise to both a function $\Gamma_\mathbb{R} : D_{\Gamma,\mathbb{R}} \to \mathbb{R}^m$ and a ball lift $\Gamma_{\mathbb{B}_p} : D_{\Gamma,\mathbb{B}_p} \to \mathbb{B}_p^m$ at each precision p. The sequence $(\Gamma_{\mathbb{B}_p})_p$ thus provides us with a ball lift $\boldsymbol{\Gamma}$ for $\Gamma_\mathbb{R}$.

Proposition 1 *If the ball lift $\boldsymbol{f}$ of $f_\mathbb{R}$ is Lipschitz for each $f \in \mathcal{F}$, then $\boldsymbol{\Gamma}$ is again Lipschitz.*

Proof For each model K of $\mathcal{F}$, for each variable $v \in \mathcal{V}$ and each input $I = (I_1, \ldots, I_n) \in D_{\Gamma,K}$, let $v_{K,k}(I)$ denote the value of v after step k. We may regard $v_{K,k}$ as a function from $D_{\Gamma,K}$ to K. In particular, we obtain a computable sequence of functions $v_{\mathbb{B}_p,k}$ that give rise to a ball lift $\boldsymbol{v}^{(k)}$ of $v_{\mathbb{R},k}$. Let us show by induction over k that $\boldsymbol{v}^{(k)}$ is Lipschitz for every $v \in \mathcal{V}$. This is clear for $k = 0$, so let $k > 0$. If $v \neq X_k$, then we have $\boldsymbol{v}^{(k)} = \boldsymbol{v}^{(k-1)}$; otherwise, we have

$$\boldsymbol{v}^{(k)} = \boldsymbol{f}_k \left(\boldsymbol{Y}_{k,1}^{(k-1)}, \ldots, \boldsymbol{Y}_{k,r_{f_k}}^{(k-1)} \right).$$

In both cases, it follows from Lemma 1 that $\boldsymbol{v}^{(k)}$ is again a Lipschitz ball lift. We conclude by noticing that $\boldsymbol{\Gamma} = (\boldsymbol{O}_1^{(\ell)}, \ldots, \boldsymbol{O}_n^{(\ell)})$. □

4 Computable numbers and ultimate complexity

A real number $x \in \mathbb{R}$ is said to be *computable* if there exists an *approximation algorithm* $\check{x}$ that takes $p \in \mathbb{N}$ on input and produces $\check{x}(p) \in \mathbb{D}_p$ on output with $|x - \check{x}(p)| \leqslant 2^{-p}$ (we say that $\check{x}(p)$ is a 2^{-p}-*approximation* of x). We denote by $\mathbb{R}^{\mathrm{com}}$ the field of computable real numbers.

Let $T(p)$ be a nondecreasing function. We say that a computable real number $x \in \mathbb{R}^{\mathrm{com}}$ has *ultimate complexity* $T(p)$ if it admits an approximation algorithm $\check{x}$ that computes $\check{x}(p)$ in time $T(p+\delta)$ for some fixed constant $\delta \in \mathbb{N}$. The fact that we allow $\check{x}(p)$ to be computed in time $T(p+\delta)$ and not $T(p)$ is justified by the observation that the position of the "binary dot" is somewhat arbitrary in the approximation process of a computable number.

The notion of approximation algorithm generalizes to vectors with real coefficients: given $v \in (\mathbb{R}^{\mathrm{com}})^n$, an approximation algorithm for v as a whole is an algorithm $\check{v}$ that takes $p \in \mathbb{N}$ on input and returns $\check{v}(p) \in \mathbb{D}_p^r$ on output with $|\check{v}(p)_i - v_i| \leqslant 2^{-p}$ for $i = 1, \ldots, n$. This definition naturally extends to any other mathematical objects that can be encoded by vectors of real numbers: complex numbers (by their real and complex parts), polynomials and matrices (by their vectors of coefficients), etc. The notion of ultimate complexity also extends to any of these objects.

A ball lift $\boldsymbol{f}$ is said to be *computable* if there exists an algorithm for computing $\boldsymbol{f}_p$ for all $p \in \mathbb{N}$. A computable ball lift $\boldsymbol{f}$ of a function $f : D \to \mathbb{R}^m$ with $D \subseteq \mathbb{R}^n$ allows us to compute the restriction of f to $D \cap (\mathbb{R}^{\mathrm{com}})^n$: given $x \in D \cap (\mathbb{R}^{\mathrm{com}})^n$ with approximation algorithm $\check{x}$, by taking $\boldsymbol{x}_p = \mathcal{B}(\check{x}(p), 2^{-p}) \in \mathbb{B}_p^n$, we have $\bigcap_{p\in\mathbb{N}} \boldsymbol{x}_p = \{x\}$, $\bigcap_{p\in\mathbb{N}} \boldsymbol{f}_p(\boldsymbol{x}_p) = \{f(x)\}$, and $\lim_{p\to\infty} \mathrm{rad}(\boldsymbol{f}_p(\boldsymbol{x}_p)) = 0$.

Let F be a nondecreasing function and assume that D is open. We say that $\boldsymbol{f}$ has *ultimate complexity* $F(p)$ if for every compact set $C \subseteq D$, there exist constants $p_0 \in \mathbb{N}$, $\varrho > 0$ and $\delta \in \mathbb{N}$ such that for any $p \geqslant p_0$ and $\boldsymbol{x}_p \in \mathrm{dom}\, \boldsymbol{f}_p$ with $\boldsymbol{x}_p \subseteq C$ and $\mathrm{rad}(\boldsymbol{x}_p) \leqslant \varrho$, we can compute $\boldsymbol{f}_p(\boldsymbol{x}_p)$ in time $F(p+\delta)$. For instance, $\oplus$ and $\ominus$ have ultimate complexity $O(p)$, whereas $\otimes$ and $\boldsymbol{\iota}$ have ultimate complexity $O(\mathsf{l}(p))$.

Proposition 2 *Assume that $\boldsymbol{f}$ is locally Lipschitz. If $\boldsymbol{f}$ has ultimate complexity $F(p)$ and $x \in D \cap (\mathbb{R}^{\mathrm{com}})^n$ has ultimate complexity $T(p)$, then $f(x)$ has ultimate complexity $T(p) + F(p)$.*

Proof Let $\check{x}$ be an approximation algorithm for x of complexity $T(p+\delta)$, where $\delta \in \mathbb{N}$. There exist $p_0 \in \mathbb{N}$ and a compact ball C around x with $C \subseteq \mathrm{dom}\, f$ and such that $\boldsymbol{x}_p = \mathcal{B}(\check{x}(p), 2^{-p}) \in \mathbb{B}_p^n$ is included in C for all $p \geqslant p_0$. There also exists a constant $\delta' \in \mathbb{N}$ such that $\boldsymbol{f}_p(\boldsymbol{x}_p)$ can be computed in time $F(p+\delta')$ for all $p \geqslant p_0$. Since $\boldsymbol{f}$ is locally Lipschitz, there exists yet another constant $\delta'' \in \mathbb{N}$ such that $\mathrm{rad}(\boldsymbol{f}_p(\boldsymbol{x}_p)) \leqslant 2^{\delta''-p}$ for $p \geqslant p_0$. For $q = p - \delta'' \geqslant \max(p_0 - \delta'', 0)$ and $\delta''' = \max(\delta, \delta')$, this shows that we may compute a 2^{-q}-approximation of $f(x)$ in time $T(q+\delta''') + F(q+\delta''')$. □

Proposition 3 *Assume that $\boldsymbol{f}$ and $\boldsymbol{g}$ are two locally Lipschitz ball lifts of f and g that can be composed. If $\boldsymbol{f}$ and $\boldsymbol{g}$ have respective ultimate complexities $F(p)$ and $G(p)$, then $\boldsymbol{g} \circ \boldsymbol{f}$ has ultimate complexity $F(p) + G(p)$.*

Proof In a similar way as in the proof of Lemma 1, the evaluation of $(\boldsymbol{g} \circ \boldsymbol{f})_p(\boldsymbol{x}_p)$ for $\boldsymbol{x}_p \in \mathrm{dom}\, \boldsymbol{f}_p$ with $\boldsymbol{x}_p \subseteq C$ and $\mathrm{rad}(\boldsymbol{x}_p) \leqslant \varrho$ boils down to the evaluation of $\boldsymbol{f}_p$ at $\boldsymbol{x}_p$ and the evaluation of $\boldsymbol{g}_p$ at $\boldsymbol{y}_p := \boldsymbol{f}_p(\boldsymbol{x}_p) \subseteq C' := f(C) + \mathcal{B}(0, \varepsilon)$ with $\mathrm{rad}(\boldsymbol{y}_p) \leqslant \varrho'$. Modulo a further lowering of ϱ and ϱ' if necessary, these evaluations can be done in time $F(p+\delta)$ and $G(p+\delta')$ for suitable $\delta, \delta' \in \mathbb{N}$ and sufficiently large p. □

Theorem 1 *Assume that $\mathbb{R}$ is a model for the function symbols $\mathcal{F}$, and that we are given a computable ball lift $\boldsymbol{f}$ of $f_{\mathbb{R}}$ for each $f \in \mathcal{F}$. For each $f \in \mathcal{F}$, assume in addition that $\boldsymbol{f}$ is locally Lipschitz, and let F_f be a nondecreasing function such that $\boldsymbol{f}$ has ultimate complexity $F_f(p)$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_\ell$ be an SLP over $\mathcal{F}$ whose k-th instruction Γ_k writes $X_k := f_k(Y_{k,1}, \ldots, Y_{k,r_f})$. Then, the ball lift $\boldsymbol{\Gamma}$ of $\Gamma_{\mathbb{R}}$ has ultimate complexity*

$$F_\Gamma(p) \;:=\; F_{f_1}(p) + \cdots + F_{f_\ell}(p).$$

Proof This is a direct consequence of Proposition 3. □

Corollary 1 *Let Γ be an SLP of length ℓ over $\mathcal{F} = \{0, 1, +, -, \times, \iota\}$ (where 0 and 1 are naturally seen as constant functions of arity zero). Then, there exists a ball lift $\boldsymbol{\Gamma}$ of $\Gamma_{\mathbb{R}}$ with ultimate complexity $O(\mathsf{I}(p)\ell)$.*

Proof We use the ball lifts of section 2 for each $f \in \{+, -, \times, \iota\}$: they are locally Lipschitz and computable with ultimate complexity $O(\mathsf{I}(p))$. We may thus apply the Theorem 1 to obtain $F_\Gamma(p) = O(\mathsf{I}(p)\ell)$. □

5 Application to modular composition

Let $\mathbb{K}$ be an effective field, and let f, g, h be polynomials in $\mathbb{K}[x]$. The problem of *modular composition* is to compute $g \circ f$ modulo h. Modular composition is an important problem in complexity theory because of its applications to polynomial factorization [21, 22, 20]. It also occurs very naturally whenever one wishes to perform polynomial computations over $\mathbb{K}$ inside an algebraic extension of $\mathbb{K}$. In addition, given two different representations $\mathbb{K}[x]/(h(x)) \cong \mathbb{K}[\tilde{x}]/(\tilde{h}(\tilde{x}))$ of an algebraic extension of $\mathbb{K}$, the implementation of an explicit isomorphism actually boils down to modular composition.

5.1 Algebraic complexity

Besides the bit complexity model from section 2.1, we will also need the *algebraic complexity model*, in which running times are analyzed in terms of the number of operations in some abstract ground ring or field [6, Chapter 4].

We write $\mathsf{M} : \mathbb{N} \to \mathbb{R}^>$ for a cost function such that for any effective field $\mathbb{K}$ and any two polynomials of degree $< n$ in $\mathbb{K}[x]$, we can compute their product using at most $\mathsf{M}(n)$ arithmetic operations in $\mathbb{K}$. The fastest currently known algorithm [8] allows us to take $\mathsf{M}(n) = O(n \log n \log\log n) = \tilde{O}(n)$. As in the case of integer multiplication, we assume that $\mathsf{M}(n)/n$ is non-decreasing; this again implies that M is super-additive.

Given two polynomials $g, h \in \mathbb{K}[x]$, we define $g \operatorname{quo} h$ and $g \operatorname{rem} h$ to be the *quotient* and the *remainder* of the Euclidean division of g by h. Both the quotient and the remainder can be computed using $O(\mathsf{M}(n))$ operations in $\mathbb{K}$, if g and h have degrees $\leqslant n$. We recall that the gcd of two polynomials of degrees at most n over $\mathbb{K}$ can be computed using $O(\mathsf{M}(n) \log n)$ operations in $\mathbb{K}$ [9, Algorithm 11.4]. Given polynomials f and $g_1, \ldots, g_l$ over $\mathbb{K}$ with $\deg f = n$ and $\deg g_1 + \cdots + \deg g_l = O(n)$, all the remainders $f \operatorname{rem} g_i$ may be computed simultaneously in cost $O(\mathsf{M}(n) \log l)$ using a *subproduct tree* [9, Chapter 10]. The inverse problem, called *Chinese remaindering*, can be solved with a similar cost $O(\mathsf{M}(n) \log l)$, assuming that the g_i are pairwise coprime. The fastest known algorithms for these tasks can be found in [3, 1, 14].

5.2 Related work

Let f, g and h be polynomials in $\mathbb{K}[x]$ of respective degrees $< n$, $< n$ and n. The naive modular composition algorithm takes $O(n\mathsf{M}(n))$ operations in $\mathbb{K}$. In 1978, Brent and Kung [4] gave an algorithm with cost $O(\sqrt{n}\mathsf{M}(n) + n^2)$. It uses the *baby-step giant-step* technique due to Paterson and Stockmeyer [29], and even yields a sub-quadratic cost $O(n^{\varpi} + \sqrt{n}\mathsf{M}_{\mathbb{K}}(n))$ when using fast linear algebra; see [19, p. 185]. The constant $\varpi > 1.5$ is such that a $\sqrt{n} \times \sqrt{n}$ matrix over $\mathbb{K}$ can be multiplied with another $\sqrt{n} \times n$ rectangular matrix in time $O(n^{\varpi})$. The best current bound $\varpi < 1.667$ is due to Huang and Pan [18, Theorem 10.1].

A major breakthrough has been achieved by Kedlaya and Umans [21, 22] in the case when $\mathbb{K}$ is the finite field $\mathbb{F}_q$. For any positive $\varepsilon > 0$, they showed that the composition $g \circ f$ modulo h could be computed with bit complexity $O((n \log q)^{1+\varepsilon})$. Unfortunately, it remains a major open problem to turn this theoretical complexity bound into practically useful implementations.

Quite surprisingly, the existing literature on modular composition does not exploit the simple observation that composition modulo a separable polynomial $h \in \mathbb{K}[x]$ that splits over $\mathbb{K}$ can be reduced to the well known problems of multi-point evaluation and interpolation [9, Chapter 10]. More precisely, assume that $h = (x - \sigma_1) \cdots (x - \sigma_n)$ is separable, which means that $\gcd(h, h') = 1$. If $f, g \in \mathbb{K}[x]$ are of degree $< n$, then $g \circ f \bmod h$ can be computed by evaluating f at $\sigma_1, \ldots, \sigma_n$, by evaluating g at $f(\sigma_1), \ldots, f(\sigma_n)$, and by interpolating the evaluations $(g \circ f)(\sigma_1), \ldots, (g \circ f)(\sigma_n)$ to yield $g \circ f \bmod h$.

Whenever $\mathbb{K}$ is algebraically closed and a factorization of h is known, the latter observation leads to a softly-optimal algorithm for composition modulo h. More generally, if the computation of a factorization of h has a negligible or acceptable cost, then this approach leads to an efficient method for modular composition. In this paper, we prove a precise complexity result in the case when $\mathbb{K}$ is the field of computable complex numbers. In a separate paper [16], we also consider the case when $\mathbb{K}$ is a finite field and h has composite degree; in that case, h can be factored over suitable field extensions, and similar ideas lead to improved complexity bounds.

In the special case of power series composition (i.e. composition modulo $h = x^n$), our approach is similar in spirit to the analytic algorithm designed by Ritzmann [30]; see also [12]. In order to keep the exposition as simple as possible in this paper, we only study composition modulo separable polynomials. By handling multiplicities with Ritzmann's algorithm, we expect our algorithm to extend to the general case.

5.3 Abstract modular composition in the separable case

For any field $\mathbb{K}$ and $n \in \mathbb{N}$, we denote

$$\mathbb{K}[x]_{<n} \quad := \quad \{P \in \mathbb{K}[x] : \deg P < n\}.$$

In this section, $\mathbb{K}$ represents an abstract algebraically closed field of constants. Let $h = x^n + h_{n-1}x^{n-1} + \cdots + h_0 \in \mathbb{K}[x]$ be a separable monic polynomial, so h admits n pairwise distinct roots $\sigma_1, \ldots, \sigma_n$ in $\mathbb{K}$. Then we may use the following algorithm for composition modulo h:

Algorithm 1

Input Polynomials $f, g \in \mathbb{K}[x]_{<n}$ and pairwise distinct $\sigma_1, \ldots, \sigma_n \in \mathbb{K}$.

Output $g \circ f \operatorname{rem} h$, where $h = (x - \sigma_1) \cdots (x - \sigma_n)$.

1. Compute $v_1 = f(\sigma_1), \ldots, v_n = f(\sigma_n)$ using fast multi-point evaluation.
2. Compute $w_1 = g(v_1), \ldots, w_n = g(v_n)$ using fast multi-point evaluation.
3. Retrieve $\varrho \in \mathbb{K}[x]_{<n}$ with $\varrho(\sigma_1) = v_1, \ldots, \varrho(\sigma_n) = v_n$ using fast interpolation.
4. Return ϱ.

Theorem 2 *Algorithm 1 is correct and requires* $O(\mathsf{M}(n) \log n)$ *operations in* $\mathbb{K}$.

Proof By construction, $\varrho(\sigma_i) = (g \circ f)(\sigma_i) = (g \circ f \operatorname{rem} h)(\sigma_i)$ for $i = 1, \ldots, n$. Since $\deg \varrho < n$ and the σ_i are pairwise distinct, it follows that $\varrho = g \circ f \operatorname{rem} h$. This proves the correctness of the algorithm. The complexity bound follows from the fact that steps 1, 2 and 3 take $O(\mathsf{M}(n) \log n)$ operations in $\mathbb{K}$. □

We wish to apply the theorem in the case when $\mathbb{K} = \mathbb{C}$. Of course, on a Turing machine, we can only approximate complex numbers with arbitrarily high precision, and likewise for the field operations in $\mathbb{C}$. For given numbers x and y, approximations at precision p for $x+y$, $x-y$, $x \times y$ and x/y (whenever $y \neq 0$) can all be computed in time $O(\mathsf{I}(p))$. In view of Theorem 2, it is therefore natural to ask whether p-bit approximations of the coefficients of $g \circ f \operatorname{rem} h$ may be computed in time $O(\mathsf{I}(p)\mathsf{M}(n) \log n)$.

In the remainder of this paper we give a positive answer to a carefully formulated version of this question. First we will prove a complexity bound for modular composition that holds for a fixed modulus h with known roots $\sigma_1, \ldots, \sigma_n$ and for sufficiently large working precisions p. Then we will show that the assumption that the roots $\sigma_1, \ldots, \sigma_n$ of h are known is actually quite harmless for ultimate complexity for the following reason: as soon as approximations for $\sigma_1, \ldots, \sigma_n$ are known at a sufficiently high precision, the computation of even better approximations can be done fast using Newton's method combined with multi-point evaluation. Since we are only interested in the complexity for "sufficiently large working precisions", the computation of the initial approximations of $\sigma_1, \ldots, \sigma_n$ can therefore be regarded as a precomputation of negligible cost.

5.4 Ultimate modular composition for separable moduli

Lemma 2 *There exists a constant* $\kappa > 0$ *such that the following assertion holds. Let* $f, g \in \mathbb{C}^{\mathrm{com}}[x]_{<n}$*, let* $\sigma_1, \ldots, \sigma_n$ *be pairwise distinct elements of* $\mathbb{C}^{\mathrm{com}}$*, and let* $h = (x - \sigma_1) \cdots (x - \sigma_n)$*. Assume that* $(f_0, \ldots, f_{n-1}, g_0, \ldots, g_{n-1}, \sigma_1, \ldots, \sigma_n)$ *has ultimate complexity* $T(n, p)$*. Then* $\varrho = g \circ f \operatorname{rem} h$ *has ultimate complexity* $T(n, p) + \kappa \mathsf{I}(p)\mathsf{M}(n) \log n$*.*

Proof The algorithm for fast multi-point evaluation of a polynomial $P = \sum_{i=0}^{n-1} P_i x^i \in \mathbb{K}[x]_{<n}$ at $\xi_1, \ldots, \xi_n \in \mathbb{K}$ can be regarded as an SLP over $\mathcal{F} = \{0, 1, +, -, \times, \iota\}$ of length $O(\mathsf{M}(n) \log n)$ that takes $(P_0, \ldots, P_{n-1}, \xi_1, \ldots, \xi_n) \in \mathbb{K}^{2n}$ on input and that produces $(P(\xi_1), \ldots, P(\xi_n)) \in \mathbb{K}^n$ on output. Similarly, the algorithm for interpolation can be regarded as an SLP over $\mathcal{F}$ of length $O(\mathsf{M}(n) \log n)$ that takes $(\xi_1, \ldots, \xi_n, v_1, \ldots, v_n) \in \mathbb{K}^{2n}$ on input and that produces $(P_0, \ldots, P_{n-1}) \in \mathbb{K}^n$ on output with $v_1 = P(\xi_1), \ldots, v_n = P(\xi_n)$. Altogether, we may regard the entire Algorithm 1 as an SLP Γ over $\mathcal{F}$ of length $O(\mathsf{M}(n) \log n)$ that takes $(f_0, \ldots, f_{n-1}, g_0, \ldots, g_{n-1}, \sigma_0, \ldots, \sigma_{n-1}) \in \mathbb{K}^{3n}$

on input and that produces $(\varrho_0, \ldots, \varrho_{n-1}) \in \mathbb{K}^n$ on output with $\rho = g \circ f \operatorname{rem} h = \sum_{i=0}^{n-1} \rho_i x^i \in \mathbb{K}[x]_{<n}$. It follows from Corollary 1 that $\Gamma_{\mathbb{R}}$ admits a ball lift $\mathbf{\Gamma}$ of ultimate complexity

$$O\left(\mathsf{I}(p)\mathsf{M}(n)\log n\right).$$

The conclusion now follows from Proposition 2. □

According to the above lemma, we notice that the time complexity for computing $\varrho = g \circ f \operatorname{rem} h$ is $T(n, p+\delta) + \kappa \mathsf{I}(p+\delta)\mathsf{M}(n)\log n$ for some constant δ that depends on n, f, g, and the σ_i.

Lemma 3 *There exists a constant $\kappa > 0$ such that the following assertion holds. Let $h \in \mathbb{C}^{\mathrm{com}}[x]$ be separable and monic of degree n, and denote the roots of h by $\sigma = (\sigma_1, \ldots, \sigma_n)$. If h has ultimate complexity $T(n,p)$, then σ has ultimate complexity $T(n,p) + \kappa \mathsf{I}(p)\mathsf{M}(n)\log n$.*

Proof There are many algorithms for the certified computation of the roots of a separable complex polynomial. We may use any of these algorithms as a "fall back" algorithm in the case that we only need a 2^{-p}-approximation of σ at a low precision p determined by h only.

For general precisions p, we use the following strategy in order to compute a ball $\boldsymbol{\sigma} \in \mathbb{B}_p^n$ with $\sigma \in \boldsymbol{\sigma}$ and $\operatorname{rad}(\boldsymbol{\sigma}) \leqslant 2^{-\alpha p}$ for some fixed threshold $1/2 < \alpha < 1$. For some suitable $p_0 \in \mathbb{N}$ and $p \leqslant p_0$, we use the fall back algorithm. For $p > p_0$ and for a second fixed constant $1/2 < \beta < 1$, we first compute a ball enclosure $\boldsymbol{\tau} \in \mathbb{B}_q^n$ at the lower precision $q = \lceil \beta p \rceil$ using a recursive application of the method. We next compute $\boldsymbol{\sigma}$ using a ball version of the Newton iteration, as explained below. If this yields a ball $\boldsymbol{\sigma}$ with acceptable radius $\operatorname{rad}(\boldsymbol{\sigma}) \leqslant 2^{-\alpha p}$, then we are done. Otherwise, we resort to our fall-back method. Such calls of the fall-back method only occur if the default threshold precision p_0 was chosen too low. Nevertheless, we will show that there exists a threshold p_1 such that the computed $\boldsymbol{\sigma}$ by the Newton iteration always satisfies $\operatorname{rad}(\boldsymbol{\sigma}) \leqslant 2^{-\alpha p}$ for $p \geqslant p_1$.

Let us detail how we perform our ball version of the Newton iteration. Recall that $\boldsymbol{\tau} \in \mathbb{B}_q^n$ with $\sigma \in \boldsymbol{\tau}$ and $\operatorname{rad}(\boldsymbol{\tau}) \leqslant 2^{-\alpha\beta p}$ is given. We also assume that we computed once and for all a 2^{-p}-approximation of h, in the form of a ball polynomial $\boldsymbol{h}_p \in \mathbb{B}_p[\mathrm{i}][x]$ of radius 2^{-p} that contains h. Now we evaluate $\boldsymbol{h}_p$ and $\boldsymbol{h}_p'$ at each of the points $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n$ using fast multi-point evaluation. Let us denote the results by $\boldsymbol{v} = \boldsymbol{h}_p(\boldsymbol{\sigma})$ and $\boldsymbol{w} = \boldsymbol{h}_p'(\boldsymbol{\sigma})$. Let τ, v and w denote the balls with radius zero and whose centers are the same as for $\boldsymbol{\tau}$, $\boldsymbol{v}$ and $\boldsymbol{w}$. Using vector notation, the Newton iteration now becomes:

$$\boldsymbol{\sigma} \;=\; (\tau \ominus_p \boldsymbol{\iota}_p(w) \otimes_p v) \oplus_p (1 \ominus_p \boldsymbol{\iota}_p(w) \otimes_p \boldsymbol{w}) \otimes_p (\boldsymbol{\tau} \ominus_p \tau).$$

If $\sigma \in \boldsymbol{\tau}$, then it is well-known [23, 31] that $\sigma \in \boldsymbol{\sigma}$. Since $\operatorname{rad}(\boldsymbol{\tau}) \leqslant 2^{-\alpha\beta p}$, the fact that multi-point ball evaluation (used for $\boldsymbol{h}_p$ and $\boldsymbol{h}_p'$) is locally Lipschitz implies the existence of a constant $\delta > 0$ with $\operatorname{rad}(\boldsymbol{\nu}) \leqslant 2^{\delta - \alpha\beta p}$ and $\operatorname{rad}(\boldsymbol{w}) \leqslant 2^{\delta - \alpha\beta p}$. Since $h'(\sigma_i) \neq 0$ for $i = 1, \ldots, n$, there also exists a constant $\delta' > 0$ with $1 - \boldsymbol{\iota}_p(w)\boldsymbol{w} \subseteq \mathcal{B}(0, 2^{\delta' - \alpha\beta p})$. Altogether, this means that there exists a constant $\delta'' > 0$ with $\operatorname{rad}(\boldsymbol{\sigma}) \leqslant 2^{\delta'' - 2\alpha\beta p}$. Let $p_1 = \lceil \delta''/(\alpha(2\beta - 1)) \rceil$. Then for any $p \geqslant p_1$, the Newton iteration provides us with a $\boldsymbol{\sigma}$ with $\operatorname{rad}(\boldsymbol{\sigma}) \leqslant 2^{-\alpha p}$.

Let us now analyze the ultimate complexity $C(n,p)$ of our algorithm. For large $p \geqslant p_1$, the algorithm essentially performs two multi-point evaluations of ultimate cost $\kappa' \mathsf{I}(p)\mathsf{M}(n)\log n$ for some constant κ' that does not depend on p, and a recursive call. Consequently,

$$C(n,p) \;\leqslant\; \kappa' \mathsf{I}(p)\mathsf{M}(n)\log n + C(n, \lceil \beta p \rceil).$$

We finally obtain an other constant $\kappa \geqslant \kappa'$ such that

$$C(n,p) \quad \leqslant \quad \kappa \mathsf{I}(p)\mathsf{M}(n)\log n,$$

by summing up the geometric progression and using the fact that $\mathsf{I}(p)/p$ is nondecreasing. The conclusion now follows from Lemma 2. □

Remark 2 A remarkable feature of the above proof is that the precision p_1 at which the Newton iteration can safely be used does not need to be known in advance. In particular, the proof does not require any *a priori* knowledge about the Lipschitz constants.

Theorem 3 *There exists a constant $\kappa > 0$ such that the following assertion holds. Let $f, g \in \mathbb{C}^{\mathrm{com}}[x]_{<n}$ and let $h \in \mathbb{C}^{\mathrm{com}}[x]$ be separable and monic of degree n. Assume that (f, g, h) has ultimate complexity $T(n,p)$. Then $\varrho = g \circ f \operatorname{rem} h$ has ultimate complexity $T(n,p) + \kappa \mathsf{I}(p)\mathsf{M}(n)\log n$.*

Proof This is an immediate consequence of the combination of the two above lemmas. □

One disadvantage of ultimate complexity is that it does not provide us with any information about the precision from which the ultimate complexity is reached. In practical applications, the input polynomials f, g and h often admit integer or rational coefficients. In these cases, the required bit precision is expected to be of order $n(l+n)$ in the worst case, where $n = \deg h$ and l is the largest bit size of the coefficients: in fact, this precision allows to compute all the complex roots of h efficiently using algorithms from [26, 27, 32]. This precision should also be sufficient to perform the multi-point polynomial evaluations of g and f by asymptotically fast algorithms.

6 Conclusion and final remarks

With some more work, we expect that all above bounds of the form $O\left(\mathsf{I}(p)\mathsf{M}(n)\log n\right)$ can be lowered to $O\left(\mathsf{I}(np)\log n\right)$. Notice that $\mathsf{I}(np)\log n = O\left(\mathsf{I}(p)n\log n\right)$ for $p \geqslant n$, when taking $\mathsf{I}(p) = \Theta(n\log n)$. In order to prove this stronger bound using our framework, one might add an auxiliary operation $\times^{[n]}$ for the product of two polynomials of degrees $< n$ to the set of signatures $\mathcal{F}$. Polynomial products of this kind can be implemented for coefficients in $\mathbb{D}_p[\mathrm{i}]$ with $p \geqslant n$ using Kronecker substitution. For bounded coefficients, this technique allows for the computation of one such product in time $O\left(\mathsf{I}(np)\right)$. By using Theorem 1, a standard complexity analysis should show that multi-point evaluation and interpolation have ultimate complexity $O\left(\mathsf{I}(np)\log n\right)$.

By Theorem 3, the actual bit complexity of modular composition is of the form $T(n, p+\delta) + \kappa\mathsf{I}(p+\delta)\mathsf{M}(n)\log n$ for some value of δ that depends on f, g, h (hence on n). An interesting problem is to get a better grip on this value δ, which mainly depends on the geometric proximity of the roots of h.

If f, g, h belong to $\mathbb{Q}[x]$, then $T(n,p) = O\left(n\mathsf{I}(p)\right)$ and we may wish to bound δ as a function of n and the maximum bit size l of the coefficients of f, g and h. This would involve bit complexity results for root isolation [26, 27, 32], for multi-point evaluation, and for interpolation. The overall complexity should then be compared with the maximal size of the output, namely $g \circ f \operatorname{rem} h$, which is in general much larger than the input size. Here, the ultimate complexity model therefore offers a convenient trade-off between a fine asymptotic complexity bound and a long technical complexity analysis.

References

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

[2] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer-Verlag New York, 1998.

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

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

[5] R. P. Brent and P. Zimmermann. *Modern computer arithmetic*, volume 18 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, 2010.

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

[7] P. Bürgisser and F. Cucker. *Condition. The Geometry of Numerical Algorithms*, volume 349 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag Berlin Heidelberg, 2013.

[8] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Infor.*, 28(7):693–701, 1991.

[9] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 3rd edition, 2013.

[10] D. Harvey and J. van der Hoeven. On the complexity of integer matrix multiplication. *J. Symbolic Comput.*, 89:1–8, 2018.

[11] D. Harvey and J. van der Hoeven. Integer multiplication in time $O(n \log n)$. Technical report, HAL, 2019. `http://hal.archives-ouvertes.fr/hal-02070778`.

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

[13] J. van der Hoeven. Ball arithmetic. Technical report, CNRS & École polytechnique, 2011. `https://hal.archives-ouvertes.fr/hal-00432152/`.

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

[15] J. van der Hoeven et al. GNU TeXmacs. `http://www.texmacs.org`, 1998.

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

[17] J. van der Hoeven and G. Lecerf. Fast multivariate multi-point evaluation revisited. *J. Complexity*, 56:101405, 2020.

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

[19] 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.

[20] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp.*, 67(223):1179–1197, 1998.

[21] 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.

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

[23] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehler-schranken. *Computing*, 4:187–201, 1969.

[24] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

[25] J.-M. Muller, N. Brunie, F. De Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres. *Handbook of floating-point arithmetic*. Birkhäuser Basel, 2nd edition, 2018.

[26] C. A. Neff and J. H. Reif. An efficient algorithm for the complex roots problem. *J. Complexity*, 12(2):81–115, 1996.

[27] V. Y. Pan. Univariate polynomials: nearly optimal algorithms for numerical factorization and root-finding. *J. Symbolic Comput.*, 33(5):701–733, 2002.

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

[29] 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.

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

[31] S. M. Rump. *Kleine Fehlerschranken bei Matrixproblemen.* PhD thesis, Universität Karlsruhe, 1980.

[32] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Preliminary Report of Mathematisches Institut der Universität Tübingen, Germany, 1982.