Arithmetic Circuit Complexity of Division and Truncation

Pranjal Dutta[*] Gorav Jindal[†] Anurag Pandey[‡] Amit Sinhababu[§]

Abstract

Given polynomials $f, g, h \in \mathbb{F}[x_1, \ldots, x_n]$ such that $f = g/h$, where both g and h are computable by arithmetic circuits of size s, we show that f can be computed by a circuit of size $\mathsf{poly}(s, \deg(h))$. This solves a special case of division elimination for high-degree circuits (Kaltofen'87 & WACT'16). The result is an exponential improvement over Strassen's classic result (Strassen'73) when $\deg(h)$ is $\mathsf{poly}(s)$ and $\deg(f)$ is $\exp(s)$, since the latter gives an upper bound of $\mathsf{poly}(s, \deg(f))$.

Further, we show that any univariate polynomial family $(f_d)_d$, defined by the initial segment of the power series expansion of rational function $g_d(x)/h_d(x)$ up to degree d (i.e. $f_d = g_d/h_d \bmod x^{d+1}$), where circuit size of g is s_d and degree of g_d is at most d, can be computed by a circuit of size $\mathsf{poly}(s_d, \deg(h_d), \log d)$. We also show a hardness result when the degrees of the rational functions are high (i.e. $\Omega(d)$), assuming hardness of the integer factorization problem.

Finally, we extend this conditional hardness to *simple* algebraic functions as well, and show that for every prime p, there is an integral algebraic power series with its minimal polynomial satisfying a degree p polynomial equation, such that its initial segment is hard to compute unless integer factoring is easy, or a multiple of $n!$ is easy to compute. Both, integer factoring and computation of multiple of $n!$, are believed to be notoriously hard. In contrast, we show examples of transcendental power series whose initial segments are easy to compute.

1 Introduction

An arithmetic circuit over an underlying field $\mathbb{F}$ is a natural model that represents a polynomial compactly (for definition see Appendix A). Arithmetic circuit complexity is the study of complexity (in terms of circuit size) of computing polynomial families. In this paper, we study two important questions in arithmetic circuit complexity. The first question is about the power of division in arithmetic circuits. The second question is about arithmetic circuit complexity of univariate polynomial families, defined by initial segments of various power series.

[*]Chennai Mathematical Institute (& CSE, IIT Kanpur), India. Email: `pranjal@cmi.ac.in`

[†]Graduiertenkolleg "Facets of Complexity/Facetten der Komplexität" (GRK 2434) (& Institut für Mathematik, Technische Universität Berlin), Germany. Email: `gorav.jindal@gmail.com`

[‡]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany. Email: `anurag.pandey3113@gmail.com`

[§]Aalen University, Germany. Email: `amitkumarsinhababu@gmail.com`

Complexity of division. In a classic result [Str73], Strassen showed that a polynomial $f(x_1, \ldots, x_n)$ of degree d, computed by an arithmetic circuit of size s using division, can also be computed by a division-free arithmetic circuit (i.e. only using addition and multiplication gates) of size $\mathsf{poly}(s, d)$.

Note that, arithmetic circuits can compute polynomials that have *exponential* degree wrt its size. For example, $g(x) := x^{2^s} - 1$, has $O(s)$-size circuit. Now, if we divide it by $h(x) := x - 1$, we get the polynomial $f(x) := 1 + x + \cdots + x^{2^s - 1}$. Strassen [Str73] gives an $\exp(s)$-size upper bound on the complexity of $f(x)$, whereas it is easy to see that $f(x)$ can be computed by just a $\mathsf{poly}(s)$-size circuit (see Remark 3.1). This leads to the following natural question.

Problem 1.1 ([Kal87, Problem 5])**.** *If a polynomial can be computed by an arithmetic circuit (with division) of size s, can it be computed by a division-free arithmetic circuit of size* $\mathsf{poly}(s)$*?*

This question is still *open* [Wac] and it is unclear whether we should *expect* a positive answer. One can push the division gate at the top and show that if f has a s-size circuit (with division gates) then there exist polynomials g and h such that $f = g/h$, where both g and h have $\mathsf{poly}(s)$-size circuits. However, $\deg(f), \deg(g)$ and $\deg(h)$ can be $\exp(s)$, and it is not clear how to eliminate this division gate at the top without incurring *exponential* blowup. In fact, the division elimination method, due to Strassen [Str73], leads to an exponential blowup in size (see Section 1.3).

Even a special case of eliminating division is open, when $f = g/x^{2^s}$, and $\deg(g)$ and $\deg(f)$ are $\exp(s)$, but g has a s-size circuit. Solving this case would resolve a couple of interesting questions in algebraic complexity. We briefly discuss some of these implications in Section 4.

Complexity of truncated power series. The second part of the paper studies the complexity of families of univariate polynomials, defined by the initial segments (equivalently, *truncation*) of a power series. Power series are ubiquitous in all branches of mathematics. From the perspective of computer science, they are quite crucial because of their pervasiveness in enumeration and combinatorics. Efficient methods to compute truncations of power series allows us to compute number sequences emerging in enumerative combinatorics like Fibonacci numbers, Catalan numbers, and Bell numbers; thanks to the *generating functions* (see [Pak18] for a survey). It also facilitates approximations of several irrational and transcendental numbers of interest, for example: $e, \pi, \sqrt{2}$, and $\zeta(3)$. The relation between truncations of power series and the theory of formal languages and context-free grammars, and the theory of codes is also well studied (see, for instance, [LS78; BR88]). In complexity theory, computing truncations of power series has been crucial in results on polynomial factorization [DSS18], division elimination in circuits [Str73], complexity of symmetric polynomials [BJ19], and complexity of algebraic functions [KT78].

Easy and hard univariate families. A univariate polynomial family $(f_d)_d$, where f_d has degree d, is called *easy to compute*, if there is a $\mathsf{poly}(\log d)$-size circuit computing f_d, otherwise we call it a *hard* family. Some examples of easy families are, $f_d := x^d$, $f_d := \sum_{i \in [d]} i^r \, x^i$, where $r \in \mathbb{N}$ (see [GS80]). A candidate *hard* family is the Pochhammer-Wilkinson polynomial $f_d := \prod_{i \in [d]} (x + i)$, for if it turns out to be easy, it would imply that integer factorization is also easy [Lip94; Bür09].

One of the ultimate goals in algebraic complexity is to characterize "easy" and "hard" polynomial families (by showing explicit bounds). Can we give interesting examples of easy univariate polynomial families that can be defined via truncation of power series? Let us again look at the polynomial family $f_d := 1 + x + \cdots + x^d$; this has a $O(\log d)$-size circuit (Remark 3.1). Interestingly, it is also the initial segment of the power series expansion of $1/(1-x)$. In contrast, [Lip78] showed that there exists a power series with $0-1$ coefficients such that their initial segments are *hard*. In fact, some of the famous candidate hard univariate polynomial families are those corresponding to initial segments of transcendental power series, for instance, $f_d := \sum_{i=0}^{d} x^i/i!$, and $f_d := \sum_{i\in[d]} (-1)^i x^i/i$, the truncations of e^x and $\log(1+x)$ respectively. Their hardness is known to imply that permanent requires superpolynomial size constant-free circuits, which implies the constant-free version of Valiant's hypothesis (the algebraic analog of $\mathsf{P} \neq \mathsf{NP}$ hypothesis) [Bür09].

This motivates our second problem.

Problem 1.2. *Characterize (differentiate "easy" and "hard") polynomial families $(f_d(x))$, defined by the initial segment (upto degree d) of a power series $\sum_{i\geq 0} a_i x^i$.*

Since the truncation of $1/(1-x)$ is easy to compute, as a natural first step towards the above Problem 1.2, we explore the complexity of initial segments of general rational functions $g(x)/h(x)$. Note that, rational function truncation is interesting, as any power series truncation up to some degree matches with a unique rational function (of given numerator and denominator degree) given by *Padé approximation* and this arises in many symbolic computational problems.

Subsequently, we study the complexity of initial segments of algebraic power series (eg. $\sqrt{1+x}$), and its connections to the central problems in algebraic complexity theory. Also, the examples of truncations of e^x and $\log(1+x)$ make us wonder whether all transcendental power series are likely to be hard. Towards this, we study truncations of transcendental power series as well.

Remark 1.1. Very recently, [DST21] introduced the notion of *SOS-hardness* (in the sum-of-squares (SOS) representation). A family $(f_d)_d$ is *SOS-easy* if it can be written as $f_d = \sum_{i\in[s]} c_i g_i^2$, for $c_i \in \mathbb{F}$ such that $\sum_i |g_i|_0 = O(d^{1/2})$, where $|g_i|_0$ denotes the sparsity or the number of monomials in g_i. Otherwise, f_d is a *SOS-hard* family. The *minimal* SOS-representation captures its SOS-complexity. For formal definitions, refer to Section 8. [DST21] showed that the SOS-hard families are innately connected to proving $\mathsf{VP} \neq \mathsf{VNP}$ (for definitions, see Appendix A). Throughout the paper, we will talk about easy/hard families wrt. both the measures (circuit complexity and SOS-complexity[1]).

1.1 Our contributions

In this work, we make progress towards both Problems Problem 1.1 and Problem 1.2. Towards the division problem, we show the following Theorem 1.1. For more details, see Section 3 (Theorem 3.2 and Theorem 3.3).

Theorem 1.1 (Division by low-degree polynomial). *Suppose, f, g, h are polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ such that $f = g/h$. Then, f can be computed by an arithmetic circuit of size*

[1]Although there are polynomial families like $f_d := \sum_{i=0}^{d} x^i$, which are easy wrt. both the measures (see Lemma B.3), in general, connection between these notions is unclear. Eg. $f_d := (x+1)^d$ is a candidate SOS-hard family, but has $O(\log d)$-size circuit. Conversely, a *random* $d^{1/2}$-sparse polynomial is trivially SOS-easy but *requires* $\omega(\log d)$-size circuit.

$\mathsf{poly}(s_1, s_2, d_h)$, *where* s_1 *(respectively,* s_2*) is the circuit complexity of* g *(respectively* h*), and* d_h *is the degree of* h*.*

Remark 1.2.

1. This result also holds when one replaces the circuit-size by *approximative* circuit-size; see Section 3.3 for details.
2. When $s_1, s_2 \leq s$, $\deg(h) = \mathsf{poly}(s)$, and $\deg(f) = \exp(s)$, our result is exponentially better than Strassen's division elimination [Str73] as the latter gives $\exp(s)$ upper bound.

Cofactor of a low-degree factor. If a multivariate polynomial $f = gh$, has size s, with $\gcd(g, h) = 1$, and $\deg(g) = \mathsf{poly}(s)$, then [Kal87] showed that g has a $\mathsf{poly}(s)$-size circuit. Invoking Theorem 1.1, we can now conclude that the cofactor h has a $\mathsf{poly}(s)$-size circuit as well (Kaltofen claimed only a $\mathsf{poly}(s)$-size circuit with division, for computing h; see the last paragraph in [Kal87, Section 4]). If g, h are not co-prime, then we need Factor Conjecture (see Section 4) to claim low complexity of h.

A related problem to division elimination is the truncation problem. Towards that, we initiate a systematic study by considering truncation of rational, algebraic and transcendental functions. For computing the initial segment of rational functions, we first generalize the observation that the initial segment of $1/(1-x)$ is easy to compute, via the *inverse identity*: $1/(1-x) = \sum_{i \geq 0} x^i$. It turns out that as long as the degree of the denominator is small, the degree-d truncation has low complexity (Theorem 1.2). We denote the ring of formal power series as $\mathbb{F}[[x]]$.

Theorem 1.2 (Truncation of low-degree rational function)**.** *Suppose,* g *and* h *are two univariate polynomials in* $\mathbb{F}[x]$ *such that* $\deg(g) \leq d$, $\deg(h) = d_h$*, and* g *can be computed a circuit of size* s*. Let,* $g/h \in \mathbb{F}[[x]]$*. Then, truncation of* g/h *upto degree-*d *can be computed by a circuit of size* $\mathsf{poly}(s, d_h, \log d)$*.*

Remark 1.3.

1. When g and h are both constant-degree polynomials, then the truncation, in fact, has a *small* SOS-complexity. For details, see Theorem 8.1.
2. We complement the above Theorem 1.2 upper bound by a conditional hardness result. In particular, we exhibit rational functions of high degree (e.g. $\Omega(d)$) whose degree-d truncations are hard to compute conditioned on the hardness of integer factorization or computation of $n!$. See Theorem 5.2 for more details.

Continuing the study of the complexity of truncated power series, we move on to algebraic power series. Here we work with constant-free circuits (i.e. constants like 2^n has to be built up from 1, requiring $O(\log n)$ many gates; for formal definition, see Section 5.2). It is not hard to show that the n-th coefficient of the *integral* power series expansion of $\sqrt{1+4x}$ (which has minpoly $y^2 = 1 + 4x$, of degree 2) is hard to compute (implying the truncation must be hard to compute, by a constant-free circuit as well) unless integer factoring is easy [CC86]; this follows from the well-known reductions: integer-factoring $\leq_{\mathsf{P}}$ computing $n!$ $\leq_{\mathsf{P}}$ computing $\binom{2n}{n}$ [2]; for a self-contained proof we refer to Theorem F.1.

Can we show such a result for simple[3] algebraic functions when the minpoly has degree > 2? For instance, for $\sqrt[3]{1+9x}$? Here, 9 is just to make the power series integral. It is

[2]here computation of an integer means, by a straight-line program or a constant-free circuit, see Definition 2.1

[3]here simple means that the degree of the minpoly of the algebraic functions and the degree of the coefficients of minpoly are both bounded by a constant

not at all clear, how the n-th coefficient of $\sqrt[3]{1+9x}$, namely $3^n/n!\prod_{j=0}^{n-1}(1-3j)$, helps in integer factoring (or in efficiently computing a multiple of $n!$). However, it turns out that the product of the n-th coefficients of $\sqrt[3]{1+9x}$ and $\sqrt[3]{(1+9x)^2}$, is a divisor of $3^n(3n)!/(n!)^3$; and computing it efficiently implies both the consequences. Exploiting the product of such binomial coefficients leads us to the following generalization; for details see Theorem 6.2 and Theorem 6.3.

Theorem 1.3 (Truncation of algebraic power series)**.** *Let $k \in \mathbb{N}$. Then, there exists $1 \le i < k$ with $i \in \mathbb{N}$, such that truncation of the integral power series $(1+k^2x)^{i/k}$ cannot have small constant-free circuits unless (i) integer factoring is easy (in the non-uniform setting) (see Algorithm 1), or (ii) some multiple of $n!$ is easy to compute (i.e. by a small straight-line program).*

Remark 1.4.

1. [Sha79] showed that if $n!$ is easy to compute, then integer factoring must be easy as well. However, it is not clear whether such statement can be drawn from some multiple of $n!$. Thus, (i) may not reduce to (ii) (& vice-versa). For details and definitions, see Section 6.

2. We also show that the hardness of the truncation of the above power series implies that permanent *requires* superpolynomial-size constant-free circuits, implying $\mathsf{VP}_0 \ne \mathsf{VNP}_0$; in fact, assuming GRH (Generalized Riemann Hypothesis), it implies $\mathsf{VP}_{\mathbb{C}} \ne \mathsf{VNP}_{\mathbb{C}}$. This is reminiscent of [Bür09]. For details, we refer to Appendix H.

Finally, we move to the truncations of transcendental functions, where we show, to our surprise that there do exist some integral transcendental power series whose initial segments are easy to compute. Thus, transcendental power series *does not necessarily mean hard*. We refer the readers to Section 7.1 for the detailed formal statements.

Theorem 1.4 (Informal)**.** *There are integeral transcendental power series whose truncations are easy.*

Therefore, Theorem 1.2–Theorem 1.4 together help in getting a good picture of the characterization sought in Problem 1.2.

1.2 Limitations of known techniques

We first discuss why standard techniques for division elimination and computing the truncations of power series do not yield the results we discover.

For the division problem, we first discuss why the division elimination method, due to Strassen [Str73], leads to an exponential blowup in size.

Strassen's division elimination. For $g(x_1,\ldots,x_n)/h(x_1,\ldots,x_n)$, wlog, assume that $h(0,\ldots,0) = 1$ (if not, then shift x_i by a *random value* α_i and get $h(\alpha_1,\ldots,\alpha_n)$ as a non-zero constant, which can be made 1, by scaling). Now, $f = g/h = g/(1-(1-h)) = g\sum_{i=0}^{\infty}(1-h)^i$. Here, we use the inverse identity: $1/(1-x) = \sum_{i\ge 0} x^i$. Assume that, f has degree d. Note that, $\tilde{f} := g\left(1+(1-h)+(1-h)^2+\cdots+(1-h)^d\right)$, has a $\mathsf{poly}(s,\log d)$ size circuit. Moreover, as $1-h$ is constant-free, truncation of $\tilde{f}$ upto degree-d (denoted as $\mathrm{Hom}_{\le d}\tilde{f}$), correctly computes f.

Howbeit, computationally, the truncation incurs a $\mathsf{poly}(d)$-size multiplicative blowup. In general, given a polynomial f, computed by a circuit of size s, it is *unlikely* that we can always get $\mathsf{poly}(s,\log d)$-size circuit for the polynomial $\mathrm{Hom}_{\le d} f$, unless, permanent has a small circuit (see Lemma D.1 for a proof of this well-known fact). In fact, every method to eliminate divisions which uses truncation, (for instance, Newton iteration, see [VZGG13], Kaltofen's Hensel-lifting [Kal86; Kal87], or allRootNI-technique via logarithmic-derivative [DSS18]) give polynomial dependence on the degree (or the square-free part) of the quotient polynomial f; both can be large.

For computing the truncation of power series of rational functions, Kung and Treib [KT78] used Newton iteration which also works, more generally, for all algebraically functions. However, the problem with Newton iteration is that even though the precision doubles with each iteration, there is always an error term as well (see [KT78] for details). So, if we want to exactly compute the polynomial up to degree d, we need to truncate in order to get rid of the error terms. This again, due to the reasons described above, incurs a $\mathsf{poly}(d)$-size multiplicative blowup, and is unlikely to be possible with an overhead bounded by $\mathsf{poly}(\log d)$.

1.3 Proof idea

Our proofs are simple and use natural ideas combined with some subtle observations and careful maneuvering. We denote $\mathbf{x} = (x_1, \ldots, x_n)$.

Division by low-degree polynomial: Proof idea of Theorem 1.1. As a warm up, we first show a similar theorem for univariate polynomials which is a much simpler case, yet it constitutes the fundamental idea.

Division by a low-degree polynomial for univariates. Let g be a univariate polynomial in $\mathbb{F}[x]$, computable by an arithmetic circuit C, and we want to divide it by degree-d univariate polynomial h. We do this by splitting each gate of C into two parts – one computing the quotient and the other computing the remainder when divided by h (denoted by div h, and mod h respectively); they are computed corresponding to each gate of the circuit, in the bottom-up manner.

In case of a '$+$' gate, the corresponding quotient and the remainder are precisely the sum of the quotients and the remainders corresponding to its children gates. While for a '$\times$' gate with its children computing polynomials $p_1 = q_1 h + r_1$ and $p_2 = q_2 h + r_2$, we have $p_1 p_2 \bmod h = r_1 r_2 \bmod h$, and $p_1 p_2 \text{ div } h = q_1 q_2 h + q_2 r_1 + q_1 r_2 + r_1 r_2 \text{ div } h$. Thus, apart from combining the outputs of the children gates, we also need to compute the quotient and the remainder of the product of the remainders of the two children ($r_1 r_2$ div h and $r_1 r_2$ mod h), which is unclear. However, if we are in the regime where the degree of h is low, then both $r_1 r_2$ div h and $r_1 r_2$ mod h will have low degree. So, we can use a simple fact that every univariate polynomial of degree at most d is trivially computable by an arithmetic circuit of size $O(d)$. This is sufficient to complete the proof (see Section 3.1 for details).

Going from univariates to multivariates. Here, the strategy is to somehow exploit the core idea used in the univariate setting. The very first step is to view the polynomials $g(\mathbf{x})$ and $h(\mathbf{x})$ as univariates in x_n, and also see $h(\mathbf{x})$ as a monic polynomial in x_n (wlog) where the coefficients are polynomials in the variables $x_1, \ldots, x_{n-1}$,. This step is fairly standard and is achieved via an invertible linear transformation (see Appendix C).

Now, the obvious idea of splitting each gate in the circuit of g into two gates computing the quotient and remainder simultaneously, *fails* directly, as a polynomial whose degree with respect to x_n is bounded by d, *may not* be computable by a $\mathsf{poly}(d)$-size circuit.

To overcome this, we need a subtler observation from the univariate case. Recall that apart from combining the output from children gates, the *only* extra quotient and remainder computation that need to be done locally for a '$\times$' gate are $r_1r_2 \bmod h$ and $r_1r_2 \operatorname{div} h$. Since, $\deg(r_1), \deg(r_2) \leq d-1$, we need to compute the quotient and remainder of a polynomial of degree at most $2d-2$. We show that when we divide a polynomial of degree d_1 by a polynomial of degree d_2, then there exists a circuit of size $O(d_1d_2)$ which takes as input the coefficients of both the polynomials and outputs the coefficients of the quotient and remainder polynomials (see Lemma 3.1). In the univariate case, this gives a multiplicative blowup of $O(d^2)$ which is *worse* than plugging in the trivial circuits of the quotient and the remainder (trivial circuit has size $O(d)$). However, the advantage this offers is that it also extends to the multivariate case (see Lemma 3.1). There, the degree refers to the degree wrt x_n, and instead of coefficients of the polynomials r_1r_2 and h as the inputs, we have the circuits for their coefficients (viewed as univariates in x_n) as inputs.

This also suggests the right structure to maintain in the circuit throughout. Since we also need the circuits for the coefficients of the remainder, we split each gates in the circuit of $g(\mathbf{x})$ into $d+1$ gates: d gates to maintain the remainder, and the $(d+1)$-th gate to maintain the quotient. Note that, since the degree in x_n is bounded by d, hence the degree (wrt x_n) of the remainder $\leq d-1$, and the d remainder gates compute the corresponding coefficients (which will be polynomials in $x_1, \ldots, x_{n-1}$). We also need the coefficients of $h(\mathbf{x})$, when viewed as a univariate in x_n; this can be efficiently done with a small blowup using standard techniques (see Lemma A.1). It turns out that the above idea suffices in the multivariate setting, see Section 3.2 for details.

Going to border. It turns out that our proof technique is *robust* to taking approximations, in the sense of border (or approximative) complexity, used in algebraic and geometric complexity theory (see Section 3.3 for details). The only subtle difference from the non-border case is that here the degree of the approximate circuit for h can be *large* (over $\mathbb{F}(\epsilon)[\mathbf{x}]$), but thanks to *homogenization* (Lemma A.2) which would keep the degree (in $\mathbf{x}$) low throughout.

Truncation of rational function: Proof idea of Theorem 1.2. Here, the core idea is to use *partial fraction decomposition* of rational functions. Over an algebraically closed field ($\mathbb{F} = \overline{\mathbb{F}}$), this allows us to decompose an arbitrary rational function $g(x)/h(x)$ (with $\deg(g) < \deg(h)$) as a sum of rational functions, each of the form $b/(x-a)^i$, where $a, b \in \mathbb{F}$ (see Lemma 5.1); this basically follows from factoring of h over $\mathbb{F}[x]$ (and thus the a's are roots of h).

When, $\deg(h)$ is small, number of such $b/(x-a)^i$ is also small. Moreover, the truncations of the $1/(x-a)^i$, for $a \neq 0$, is easy to compute (see Section 5.1). But there are two subtle issues to be handled: (i) what to do when $a = 0$? and (ii) what happens when $\deg(g) > \deg(h)$?

Theorem 1.1 along with some basic analysis turns out to be the savior for both the cases.

For the first issue, note that $a = 0$ implies x^m divides h for some $m \geq 1$. However, as $g/h \in \mathbb{F}[[x]]$, it turns out that x^m must also divide g, for such power series to exist (Lemma B.1). Thus, it suffices to work with $g/h = g_1/h_1$, where $g_1 := g/x^m$ and $h_1 := h/x^m$, both being polynomials in $\mathbb{F}[x]$. But what happens to the size of g_1 ? Well, thanks to Theorem 1.1: as, m is small (because $m \leq \deg(h)$), it turns out that the circuit complexity of g_1 is also small.

For the second issue, note that, $\deg(g) > \deg(h)$ implies $\deg(g_1) > \deg(h_1)$. But thanks to Theorem 1.1 again. Of course, $g_1/h_1 = g_1 \operatorname{div} h_1 + (g_1 \bmod h_1)/h_1$. Thus, $g_1 \operatorname{div} h_1$ and

$g_1 \bmod h_1$ have small complexity and moreover $\deg(g_1 \bmod h_1) < \deg(h_1)$. Additionally, g_1 div h_1 has degree $< d$ (as $\deg(g) \le d$). Thus, combining all these, the conclusion follows.

Extending to SOS-complexity. We remark that, similar proof works wrt SOS-complexity when both g and h have constant-degrees. This is mainly because $1/(1-x)^i$ has small SOS-complexity as SOS-model is *closed* under small derivatives (Lemma 8.2). For details, see Theorem 8.1.

Truncation of algebraic functions: Proof idea of Theorem 1.3. There are two parts of the proof. But before delving into that, it is not hard to show that $(1+k^2x)^{1/k}$ is an integral power series; this can be proved by some basic number-theoretic tools, for details see Theorem G.2.

For the first part, we show that easiness of the truncation of each $(1+k^2x)^{i/k}$, for all $i \in [k-1]$, leads to an efficient integer factoring algorithm (Algorithm 1). This algorithm is a subtle generalization of the algorithm of [LR09]. Note that from binomial expansion, coefficient of x^d in $(1+k^2x)^{i/k}$ is $C_{d,i} := k^d/d! \cdot \prod_{j=0}^{d-1}(i-kj)$. Moreover, when the truncations are easy, the coefficients are also easily computable, just by subtracting two consecutive truncatations and substituting $x = 1$. For a fixed i and $k \ge 3$, it is not clear how $C_{d,i}$ behaves (when $k = 2$, it is $= \binom{2d}{d}/(2d-1)$). However, if we take product of all the d-degree coefficients (i.e. $\prod_{i\in[k-1} C_{d,i}$), it turns out to be a 'nicer' quantity. In particular, one can show that this product is a divisor of the integer $N(d,k) := k^{(k-2)d}\,(dk)!/(d!)^k$. Moreover, $N(d,k)$ turns out to be easily computable as well.

Can we exploit any property of $N(d,k)$ which could help us factor an integer n? Well, as $N(d,k)$ is easy, computing gcd of $N(d,k)$ and n is also easy. If we can figure-out a d such that $\gcd(N(d,k),n) \ne 1, n$, we have already found a factor! So the aim is to somehow reduce the search space cleverly and find a suitable d. Wlog, one can assume that all the factors of n are greater than k (otherwise we can remove them by brute-force, as k is constant). Now, we try to find the smallest prime p dividing n. Of course, there must exist $t \in S := \{k, k^2, \ldots, k^\ell\}$, where $k^\ell \le n/k$, such that $p \in [t+1, tk]$ (as these disjoint intervals cover $[n]$). Note that $|S| = \log n$. Also, trivially $p \mid N(t,k)$, as p divides the numerator but cannot divide the denominator. So, if the $\gcd(N(t,k),n) \ne n$, we are done. But, if the gcd becomes n, it simply implies all the prime factors of n must lie in the interval $[t+1, tk]$.

Unfortunately, this interval size is still huge and we cannot brute-force over it. But, we can further reduce our search space by binary search. This idea is similar to [LR09]; each time we halve the search interval to reduce the search space for candidate d such that $\gcd(N(d,k),n) \ne 1, n$. At first, we have two integers a, b with $a = 1$ and $b = t$ such that the prime factors are in $[ak+1, bk]$. Fix $c = (a+b)/2$ and compute $\gcd(N(c,k),n)$. If the gcd is $\ne 1, n$, we are done, otherwise we branch accordingly into the first half or the second. When the gcd is 1, it must happen that the factors are in the second half i.e.$[ck+1, bk]$. When gcd $= n$, the factors are in the first half $[ak+1, ck]$. After at most $\log n$ steps, we must have either found a factor and if not, we have found a *small* interval $[sk, (s+1)k]$ of length k where all the prime factors lie. We can now brute-force to find the factors. For details, see Section 6.1 and Algorithm 1.

The second part eventually exploits and recurse on the fact that $(dk)!/(d!)^k$ is easy to compute and $(d!)^k$ is easy when $(d!)$ is easy, implying a clear pattern of recurrence from dk to d (Section 6.2).

Truncation of transcendental power series. Finally, for showing Theorem 1.4 about transcendental power series, we discover some explicit integral power series whose initial segments are *non-sparse* yet easy to compute. For this purpose, we use *stern sequences* (Section 7.1) and power series whose coefficients are multiplicative, and exploit their recursive structures. Conversely, we show hardness for the truncation of an integral transcendental power series defined via *holonomic sequences* (Section 7.2).

2 Preliminaries

Notation. We denote $\mathbf{x} = (x_1, \ldots, x_n)$. $[n]$ denotes the set $\{1, \ldots, n\}$. For a polynomial $f \in \mathbb{F}[\mathbf{x}]$, we denote up to degree-d part as $\mathrm{Hom}_{\leq d} f$ and $|f|_0$ as the sparsity or the number of monomials in f. For a differentiable function $f(x)$, we denote $f^{(k)}(x) := d^k f / dx^k$, as the k-th derivative of f. We also recall the definition of gcd of two polynomials f, g in the ring $\mathbb{F}[\mathbf{x}]$: $\gcd(f, g) =: h \Leftrightarrow h \mid f, h \mid g$, and $h' \mid f, g \implies h' \mid h$. It is unique up to constant multiples.

Field. We denote the underlying field as $\mathbb{F}$ and assume that it is algebraically closed. All our results hold when the characteristic is large or not algebraically closed, as we can go to polynomial extensions and work with it.

Binomial series. For rational n, $(x+a)^n = \sum_{k \geq 0} \binom{n}{k} x^k a^{n-k}$, where $\binom{n}{k} = n \cdot (n-1) \cdot \cdots \cdot (n-k+1)/k!$.

div **and** mod **operations.** For polynomials f and $g \in \mathbb{F}[x]$, if $f = g \cdot h + r$, where $h, r \in \mathbb{F}[x]$ such that $\deg(r) < \deg(g)$, then h is called the quotient, denoted f div g, and r is called the remainder, denoted f mod g. Operation mod may not be well-defined in the multivariate settings, however, if one assumes g to be monic in a variable say x_n, it is always well-defined (by thinking g to be a univariate in x_n). A polynomial g is monic in x_n if the leading coefficient (the nonzero coefficient of highest degree) of x_n is a non-zero constant in $\mathbb{F}$. Of course, if $g \mid f$, then f div $g = h$ and f mod $g = 0$, irrespective of monic-ness.

Power series and truncation. A formal power series is a generalization of a polynomial, where the number of terms can be infinite. Formally, $A = \sum_{i \geq 0} A_i x^i$ with $A_i \in \mathbb{F}$, is a power series in the power series ring $\mathbb{F}[[x]]$. We define the degree d truncation $\mathrm{trunc}(A, d)$ of A to be $\mathrm{trunc}(A, d) := \sum_{0 \leq i \leq d} A_i x^i$. So, $\mathrm{trunc}(A, d)$ is always a polynomial of degree at most d.

Definition 2.1 (Straight Line Program)**.** An SLP (straight line program) P (for computing an integer) of length (or size) n is a sequence of integers $a_0, \ldots, a_n$ with $a_0 = 1$ and $a_k = a_i \circ a_j$ with $i, j < k$ for $\circ \in \{+, -, \times\}$. We say that the SLP P computes the integer a_n. For an integer N, we define the straight line complexity $\tau(N)$ of N to be the length of the smallest SLP computing N.

Definition 2.2 (Algebraic and Transcendental Power Series)**.** A formal power series $f \in \mathbb{C}[[x]]$ is said to be algebraic if there exists a polynomial $g \in \mathbb{C}[x][t]$ such that $g(f) = 0$. Otherwise f is said to be transcendental.

With abuse of notation, for integers, we will sometime use complexity of the integer (implying $\tau(\cdot)$ only). Sometimes we also allow division as a operation in straight line program (each time we mention if so). For a polynomial $f \in \mathbb{F}[\mathbf{x}]$, we define the complexity $L_{\mathbb{F}}(f)$ of f to be the length of the smallest division-free arithmetic circuit (with only $\{+, -, \times\}$ gates) computing f. We also define, the complexity $\tau_{\mathbb{F}}(f)$ of f to be the length of the smallest division and constant-free arithmetic circuit computing f (all the constants are made from

1), for formal definition see Section 5.2. We will remove subscript $\mathbb{F}$ when the underlying field is clear from the context.

3 Division elimination in high-degree circuits

This section deals with Problem 1.1, where the divisor has small degree and proves Theorem 1.1. Section 3.1 shows it in the univariate setting while Section 3.2 deals with the multivariate setting, and finally, Section 3.3 shows an analogous theorem in the border complexity setting. Here, we remark that formally, one should use $f_d = g_d/h_d$, with d as an index, however with abuse of notation, we use g/h throughout the paper.

3.1 Division of Univariate Polynomials

The following theorem deals with Problem 1.1 in the univariate setup.

Theorem 3.1. *Let g, h be polynomials in $\mathbb{F}[x]$. If $L(g) = s$ and $\deg(h) = d$, then both $L(g \text{ div } h)$ and $L(g \bmod h)$ have complexity $O(sd)$.*

Proof. Suppose C is a circuit of size s which computes g. We split every gate Φ in C into two gates Φ_1 and Φ_2, to make a new circuit C', which computes both g div h and g mod h. If Φ is computing some polynomial ϕ in C, then Φ_1 computes the polynomial ϕ mod h and Φ_2 computes the polynomial ϕ div h.

The proof is inductive and traverses from bottom to the top. The base case is trivial. At some step, say that we are at a gate Φ. The children gate of ϕ are computing polynomials α and β. Let, $\alpha = q_1h + r_1$, $\beta = q_2h + r_2$ and $\phi = qh + r$, where the degrees of r, r_1, r_2 are smaller than d. So in the new circuit C', we have already computed r_1, q_1, r_2, q_2. If Φ is a $\pm$ gate then it is clear that $r = r_1 \pm r_2$ and $q = q_1 \pm q_2$. If Φ is a $\times$ gate then we have:

$$r = (r_1r_2) \bmod h\,, \text{ and } q = q_1q_2h + r_1q_2 + r_2q_1 + (r_1r_2) \text{ div } h.$$

We know that r is a polynomial of degree at most $d-1$. Since, $\deg(r_1r_2) \le 2d-2$, we get that $\deg((r_1r_2) \text{ div } h) \le d-2$. Therefore, we trivially have that: $L(r) = O(d)$ and $L((r_1r_2) \text{ div } h)) = O(d)$. Hence we can compute r, q using additional $O(d)$ many gates. Thus, C' has at most $O(sd)$ many gates. Hence $L(g \text{ div } h) = O(sd)$ (same for g mod h). $\square$

Corollary 3.1. *For $f, g, h \in \mathbb{F}[x]$, if $f = g/h$ with $L(g) = s$ and $\deg(h) = d$ then $L(f) = O(sd)$.*

Remark 3.1. The polynomial $f_d := 1 + \cdots + x^d = (x^{d+1} - 1)/(x-1)$ has $O(\log d)$ size circuit. This can also be shown via a recursive computation argument.

Can we expect both div and mod to have $\mathsf{poly}(s, \log d)$-size circuits? We show that it is highly unlikely unless factoring is *easy*, see Theorem 9.1 for details.

3.2 Division of Multivariate Polynomials

This section deals with division in the multivariate setting. But before that, we solve a particular case (by folklore techniques) which will play a crucial role to prove the main Theorem 3.2. For a proof of the following Lemma 3.1, see Appendix E.

Lemma 3.1. *Suppose $g = \sum_{i \le d_1} g_i x^i$ and $h = x^{d_2} + \sum_{i<d_2} h_i\, x^i$, in $\mathbb{F}[\mathbf{x}]$. Suppose $g = hq + r$, with $r = \sum_{i<d_2} r_i x^i$ and $q = \sum_{i \le d_1 - d_2} q_i x^i$. Then, there is a circuit of size $O(d_1\, d_2)$, whose inputs are all h_i, g_i and outputs are all r_i, q_i.*

Now we prove the following Lemma 3.2 which shows that both div and mod have low complexity when the divisor has low-degree and monic (in fact, constant leading-coefficient suffices).

Lemma 3.2 (Main Lemma)**.** *Let the polynomials $g, h \in \mathbb{F}[\mathbf{x}]$ such that h is monic in x_n, $L(g) = s_1, L(h) = s_2$, and $\deg_{x_n}(h) = d$. Then, both $L(g \text{ div } h), L(g \bmod h) \le O((s_1 + s_2)\, d^2)$.*

Proof. Suppose C is a circuit of size s_2 which computes h and C_g is the circuit of size s_1 which computes g. By using Lemma A.1, there is a circuit of size $O(s_2\, d^2)$, which computes $h_0, \cdots, h_{d-1}$.

Now, we split every gate F in C into $d+1$ gates $F_0, F_1, \ldots, F_d$. Suppose, the gate F is computing a polynomial P_F. Let $P_F \bmod h = \sum_{i<d} p_i x_n^i$. Then we want the property that F_i computes p_i for $i < d$. And if $i = d$ then F_i computes P_F div h.

Suppose F is a $+$ gate in C with children gates computing the polynomials a and b. Again express $a \bmod h = \sum_{i<d} a_i x_n^i$ and $b \bmod h = \sum_{i<d} b_i x_n^i$. It is clear that

$$(a+b) \bmod h \;=\; a \bmod h \;+\; b \bmod h.$$

Therefore $p_i = a_i + b_i$. It is also clear that $P_F \text{ div } h \;=\; a \text{ div } h \;+\; b \text{ div } h$.

Suppose F is a $\times$ gate in C with children gates computing the polynomials a and b. Again express $a \bmod h = \sum_{i<d} a_i x_n^i$ and $b \bmod h = \sum_{i<d} b_i x_n^i$. It is clear that:

$$(a \cdot b) \bmod h \;=\; (a \bmod h \cdot b \bmod h) \bmod h.$$

For div , we have that:

$$P_F \text{ div } h \;=\; a \text{ div } h \cdot b \text{ div } h \cdot h + b \text{ div } h \cdot a \bmod h + a \text{ div } h \cdot b \bmod h + (a \bmod h \cdot b \bmod h) \text{ div } h.$$

We have already computed $a \bmod h, b \bmod h, a \text{ div } h, b \text{ div } h$. So, we only need to compute $(a \bmod h \cdot b \bmod h) \bmod h$ and $(a \bmod h \cdot b \bmod h) \text{ div } h$. Since we have already computed a_i, b_i for all $i < d$, by using Lemma 3.1, we can compute all the p_i and $(a \bmod h \cdot b \bmod h) \text{ div } h$ in $O((2d-2)d) = O(d^2)$ many gates. Therefore the new circuit has $O(s_1 d^2)$ has many gates. Also we used $O(s_2 d^2)$ gates to computes $h_0, \cdots, h_{d-1}$. Hence,

$$L(g \text{ div } h) = O((s_1 + s_2)d^2), \text{ and } L(g \bmod h) = O((s_1 + s_2)d^2)\,.$$

□

The following theorem *settles* Problem 1.1, when the divisor has small degree (proving Theorem 1.1).

Theorem 3.2 (Division elimination for low-degree divisor)**.** *Let the polynomials $f, g, h \in \mathbb{F}[\mathbf{x}]$ such that $f = g/h$, with $L(g) = s_1, L(h) = s_2$, and $\deg(h) = d$. Then, $L(f) \le O((s_1 + s_2)\, d^2)$.*

Proof. The above Lemma 3.2 shows that when h is monic in x_n, the upper bound holds. Let $\tau : \mathbb{F}[\mathbf{x}] \longrightarrow \mathbb{F}[\mathbf{x}]$, be an *invertible* monic transformation (sends $x_i \mapsto \alpha_i \cdot x_n + x_i$, where $\alpha_i \in \mathbb{F}$) s.t. $\tau(h)$ is *monic* wrt x_n, such transformation exists (Lemma C.1). Note that, $L(\tau(g)) \leq s + n = O(s_1)$, and $L(\tau(h)) \leq s_2 + n = O(s_2)$. Moreover, as τ is degree-preserving, $\deg_{x_n}(\tau(h)) = d$.

So, apply Lemma 3.2 to conclude that $\tau(f) = \tau(g)$ div $\tau(h)$, has a circuit of size $O((s_1 + s_2)d^2)$. We apply τ^{-1} again (which is just a additive n-blowup) to finally deduce that

$$L(f) \;\leq\; O((s_1 + s_2)\, d^2)\,.$$

□

Remark 3.2. This proof holds when one replaces L by τ, i.e. the constant-free circuit complexity (for definition, see Section 5.2). Note that, neither div nor mod introduce any new constant in the process. Moreover, one can choose the α_i to be explicit and $\mathsf{poly}(\log d)$-computable so that τ is a monic invertible map. This establishes the claim.

3.3 Division in border complexity

The notion of border (equivalently, approximative) complexity is important in computer science. This concept popped up from early works on matrix multiplication and border rank of tensors, see [BCS13]). Whether *approximation* of polynomials provides any additional computational power is a natural question which fundamentally motivated the foundation of Geometric Complexity theory (GCT). The notion of border complexity can be motivated through two ways: *topological* and *algebraic*, and both the perspectives are known to be *equivalent* [Ald84]. For further details, we refer to [GMQ16; Mul12].

In this paper, we only work with algebraic approximation upper bounds. In the algebraic definition, one can talk about the *convergence* $\epsilon \to 0$. Here, one can see ϵ as a formal variable and $\mathbb{F}(\epsilon)$ as the function field. For an algebraic complexity class C, the approximation is defined as follows [BIZ18, Definition 2.1].

Definition 3.1 (Approximative closure of a class [BIZ18])**.** Let C be an algebraic complexity class over field $\mathbb{F}$. A family (f_n) of polynomials from $\mathbb{F}[\mathbf{x}]$ is in the *class* $\overline{C}(\mathbb{F})$ if there are polynomials $f_{n;i}$ and a function $t : \mathbb{N} \mapsto \mathbb{N}$ such that g_n is in the class C over the field $\mathbb{F}(\epsilon)$ with $g_n(\mathbf{x}) = f_n(\mathbf{x}) + \epsilon f_{n,1}(\mathbf{x}) + \epsilon^2 f_{n,2}(x) + \cdots + \epsilon^{t(n)} f_{n,t(n)}(\mathbf{x})$.

Definition 3.2. [Bür04, Defn.3.1] Let $f \in \mathbb{F}[\mathbf{x}]$. The *border complexity* $\underline{L}(f)$ is the smallest number r, such that there exists F in $\mathbb{F}(\epsilon)[\mathbf{x}]$ satisfying $F|_{\epsilon=0} = f$ and $L_{\mathbb{F}(\epsilon)}(F) \leq r$.

Note that, the circuit of F may be using $1/\epsilon$ in an intermediate step. So, we cannot merely assign $\epsilon = 0$ and get a ϵ-free circuit. Also, the ϵ-degree can be exponential in its size (and thus cannot be interpolated), see [Bür04, Theorem 5.7]). Thus, potentially $\underline{L}(f)$ can be significantly smaller than $L(f)$.

The above definition can be used to define closures of complexity class, e.g., $\overline{\mathsf{VP}}$. In this case, one can assume wlog that the degrees of g_n and $f_{n,i}$ are $\mathsf{poly}(n)$. It is known to be *closed under factoring* [Bür04, Theorem 4.1]. However, the usual method of Hensel-lifting *does not* work when the given circuit class computes polynomials of super-polynomial degree. Also, Strassen's method would have a dependency on the degree of the final polynomial. However, we can prove Theorem 3.2 analogously, in the border sense.

Theorem 3.3 (Division elimination in border complexity)**.** *Let $f, g, h \in \mathbb{F}[\mathbf{x}]$, such that $f = g/h$, with $\underline{L}(g) = s_1$, $\underline{L}(h) = s_2$, and $\deg(h) = d$. Then, $\underline{L}(f) \le O(s_1 d^2 + s_2 d^4)$.*

Proof. By definition, there exists $G, H \in \mathbb{F}(\epsilon)[\mathbf{x}]$, of size at most s_1 and s_2, respectively, such that $G := g + \epsilon \cdot \tilde{g}(\mathbf{x}, \epsilon)$, and $H := h + \epsilon \cdot \overline{h}(\mathbf{x}, \epsilon)$, where $\tilde{g}, \overline{h} \in \mathbb{F}[\epsilon, \mathbf{x}]$. We note that, $\deg_{\mathbf{x}}(H)$ can be *larger* than d. However, using Lemma A.2, we know that $L_{\mathbb{F}(\epsilon)}(\mathrm{Hom}_{\le d} H) \le O(s_2 d^2) := s_2'$.

We denote $\tilde{H} := \mathrm{Hom}_{\le d} H$. It is important to observe that $\tilde{H}|_{\epsilon=0} = h$. By definition, there exists m (could be $\exp(s_2')$) such that

$$\tilde{H} := h + \epsilon \cdot \tilde{h}(\mathbf{x}, \epsilon) \; = \; h + \sum_{j \in [m]} \epsilon^j \cdot h_j(\mathbf{x}) \,, \text{ where } h_j \in \mathbb{F}[\mathbf{x}] \,.$$

Let $\tau : \mathbb{F}[\mathbf{x}] \longrightarrow \mathbb{F}[\mathbf{x}]$, be an *invertible* monic transformation (sends $x_i \mapsto \alpha_i \cdot x_n + x_i$, where $\alpha_i \in \mathbb{F}$) s.t. $\tau(h)$ and each $\tau(h_j)$, for $j \in [m]$ is *monic* wrt x_n; such transformation exists (Lemma C.1). Note that, $L_{\mathbb{F}(\epsilon)}(\tau(G)) \le O(s_1)$ and $L_{\mathbb{F}(\epsilon)}(\tau(\tilde{H})) \le O(s_2')$. Further, $\deg_{\mathbf{x}}(\tau(\tilde{H})) = d$, as τ is a degree-preserving map. We also have the following identities:

$$\tau(\tilde{H}) = \tau(h) + \epsilon \cdot \tau(\tilde{h}) \text{ and } \tau(G) = \tau(f) \cdot \tau(h) + \epsilon \cdot \tau(\tilde{g}) \,.$$

By assumption, the leading coefficient of x_n in $\tau(\tilde{H})$ (call it α) is in $\mathbb{F}[\epsilon]$ (in fact, $\alpha \not\equiv 0 \mod \epsilon$). This basically makes $\tau(\tilde{H})$ a monic polynomial over $\mathbb{F}(\epsilon)[\mathbf{x}]$. Therefore, $\text{div}\ \tau(\tilde{H})$ and $\bmod \tau(\tilde{H})$ now make sense over $\mathbb{F}(\epsilon)[\mathbf{x}]$. By simple division, we have

$$\tau(G) \ \text{div} \ \tau(\tilde{H}) \; = \; \tau(f) + \epsilon \cdot \left(\left(\tau(\tilde{g}) - \tau(f) \cdot \tau(\tilde{h}) \right) \ \text{div} \ \tau(\tilde{H}) \right) \,. \tag{1}$$

Note that, Lemma 3.2 implies $L_{\mathbb{F}(\epsilon)} \left(\tau(G) \ \text{div} \ \tau(\tilde{H}) \right) = O((s_1 + s_2') d^2)$. By definition of $\underline{L}$ and Equation (1), it is trivial to conclude that $\underline{L}(\tau(f)) \le O((s_1 + s_2')d^2) = O(s_1 d^2 + s_2 d^4)$. As τ is invertible, we can get back f by applying τ^{-1} (incurring n-additive blowup). This finally shows

$$\underline{L}(f) \; \le \; O(s_1 d^2 + s_2 d^4) \,.$$

□

4 Implications of division elimination in algebraic complexity

An affirmative solution to Problem 1.1 would have nontrivial applications in algebraic complexity. We briefly discuss some of them in the next few paragraphs.

Division elimination in border complexity. It is not clear whether a positive solution to Problem 1.1 would resolute to solving $\mathrm{VP} = \overline{\mathrm{VP}}$ (the converse direction is also not clear). Note that, an approximative circuit can use arbitrary scalars from the field $\mathbb{F}(\epsilon)$. So it is not clear if the polynomial computed by an approximative circuit of size s can be expressed as g/h, where $g, h \in \mathbb{F}[\epsilon, \mathbf{x}]$ can be computed by circuits (using constants from $\mathbb{F}$) of size $\mathsf{poly}(s)$. However, a special case of Problem 1.1, when the denominator is as simple as x^d, is open, and it has interesting implications as we discuss. The following example is from Bürgisser [Bür04], which relates the complexity of *trailing coefficient* of a polynomial to the complexity of the polynomial itself.

Let us take a polynomial $f(\mathbf{x}, \epsilon) \in \mathbb{F}[\mathbf{x}, \epsilon]$ computed by an arithmetic circuit of size s (over $\mathbb{F}$). Suppose, $f := \sum_{i=d}^{D} C_i(\mathbf{x})\, \epsilon^i$ where C_i are polynomials in $\mathbb{F}[\mathbf{x}]$. The trailing coefficient of f wrt ϵ, which is the polynomial C_d can be computed by a circuit of size $\mathsf{poly}(s, d)$, by homogenization. Note that d can be $\exp(s)$. In contrast, it can be computed by an *approximative* circuit of size just s. The approximative circuit C' computes the polynomial f/ϵ^d (as $\lim_{\epsilon \to 0} f/\epsilon^d = C_d$). Note that, ϵ^d has $O(\log d)$-size circuit. Now, a positive solution to Problem 1.1 would imply that f/ϵ^d has a division-free circuit C of size $\mathsf{poly}(s, \log d)$. We can simply put $\epsilon = 0$ in C and compute C_d.

Division elimination in polynomial factoring. Another interesting consequence of the above mentioned case of Problem 1.1 would be the proof of Factor conjecture [Kal87; Bür04]: Any factor g of a given polynomial f can be computed by $\mathsf{poly}(s, \deg(g))$-size circuit. Bürgisser [Bür04] gave an approximative circuit of $\mathsf{poly}(s, \deg(g))$ that involves division by ϵ^d where ϵ can be seen as a formal variable. See [Bür04; Gro+20] for various consequences of Factor conjecture.

Division elimination and gcd. It turns out that the existence of small circuits for gcd and division elimination can resolve the *radical conjecture* [DSS18]: the squarefree-part or the radical of a multivariate polynomial f of size s, has size $\mathsf{poly}(s)$.

The gcd question [Kal87, Problem 4] asks whether given polynomials $f_1, \ldots, f_m$, computed by a circuit size s, their gcd $g := \gcd(f_1, \ldots, f_m)$ has size $\mathsf{poly}(s)$. Currently, the best known bound (due to Kaltofen [Kal87]) is $\mathsf{poly}(s, \deg(g))$. It is not hard to show that a positive resolution to both Problem 1.1 and gcd would also resolve the aforementioned radical conjecture.

In fact, it would also lead to $\mathsf{poly}(s)$ bound for computing the *reduced rational function*. Given a rational function p/q computed by a circuit (with division gates) of size s, compute the numerator and denominator in the reduced form ($g/h = p/q$, where g and h are coprime) in $\mathsf{poly}(s)$. Kaltofen [Kal87, Problem 4] showed a bound of $\mathsf{poly}(s, \deg(g), \deg(h))$. Note that getting numerator and denominator of reduced rational function in $\mathsf{poly}(s)$ directly implies solution to both high degree division and gcd questions.

Remark 4.1. It is known that given a polynomial f, computed by $\mathsf{poly}(s)$, all its factors *cannot* be computed by $\mathsf{poly}(s)$-size circuits. For eg. $x^{2^s} - 1$; it has factors of size $\exp(s)$ [LS78]. However, this does not give a counterexample for Problem 1.1 (as the cofactor of a hard factor is also expected to be hard).

5 Circuit complexity of rational function truncation

First, we deal with rational functions. We show both upper bound and conditional lower bound results (relating to integer factoring).

5.1 Upper bounds for rational function truncation

We show that complexity of truncation of rational functions where the degrees are small, has low complexity. For simplicity, we work with $\mathbb{F} = \overline{\mathbb{F}}$, an algebraically closed field. We first recall the following folklore decomposition.

Lemma 5.1 (Partial fraction decomposition). *Let $g(x)/h(x)$ be a rational function with $\deg(g) < \deg(h)$. If $h(x) = \prod_{i\in[k]}(x-a_i)^{d_i}$ is the factorization of $h(x)$ over $\mathbb{F}[x]$, then, there exist $b_{ij} \in \mathbb{F}$ s.t. :*

$$g(x)/h(x) \;=\; \sum_{i\in[k]}\sum_{j\in[d_i]} b_{ij}/(x-a_i)^j \,.$$

Here is an important lemma which plays a crucial role in the size upper bound of truncation.

Lemma 5.2. *For any non zero $a \in \mathbb{F}$, we have $L\left(\text{trunc}\left(1/(x-a), d\right)\right) = O(\log d)$.*

Proof. This follows from the inverse identity: $1/(a-x) = 1/a\sum_{i\geq 0}(x/a)^i$ and the fact that $L(\sum_{0\leq i\leq d}(x/a)^i) = O(\log d)$ (By using Remark 3.1). □

Now, we prove Theorem 1.2. For brevity, we state it again.

Theorem 5.1 (Truncation of low-degree rational function). *Suppose, g and h are two univariate polynomials in $\mathbb{F}[x]$ such that $\deg(g) \leq d$, $\deg(h) = d_h$, and g can be computed a circuit of size s. Let, $g/h \in \mathbb{F}[[x]]$. Then, truncation of g/h upto degree-d can be computed by a circuit of size $\mathsf{poly}(s, d_h, \log d)$.*

Proof. The main idea is to use Lemma 5.1 and the low complexity of the truncation of inverse identity (Lemma 5.2). However, the given polynomial h may be divisible by x (i.e. $h(0) = 0$). In that case, let m be the highest power of x such that $x^m \mid h$ (i.e. $x^{m+1} \nmid h$). Note that, as $g/h \in \mathbb{F}[[x]]$, $x^m \mid g$ as well (Lemma B.1). As $\deg(h) \leq d_h$, thus $m \leq d_h$.

By using Theorem 3.2, we know that $g_1 := g/x^m$ has a cicuit of size $O((s+\log d_h)\,d_h^2) =: s_1$. Trivially, $h_1 := h/x^m$ has degree $\leq d_h$, and $g/h = g_1/h_1$. Denote, $g_2 := g_1 \bmod h_1$. Obviously, $\deg(g_2) < \deg(h_1)$ and $g_1/h_1 = g_1 \text{ div } h_1 + g_2/h_1$. Invoking Theorem 3.1, one concludes that $L(g_1 \text{ div } h_1) = O(s_1\,d_h)$. Therefore, $L(\text{trunc}(g_1 \text{ div } h_1, d)) = O(s_1\,d_h)$, as $\deg(g_1) < d$.

Let h_1 factors over $\mathbb{F}[x]$ as $h_1 := \prod_{i\in[k]}(x-a_i)^{d_i}$. Trivially, $\sum d_i \leq d_h$. By using Lemma 5.1 on g_2/h_1, we know that there are constants $a_i, b_{ij} \in \mathbb{F}$ such that:

$$g_2(x)/h_1(x) \;=\; \sum_{i\in[k]}\sum_{j\in[d_i]} b_{ij}/(x-a_i)^j \,.$$

Note that, for any $a \in \mathbb{F}$ and $t \in \mathbb{N}$, $d^t/dx^t\,(1/(x-a)) = (-1)^t\,t!\cdot\left(1/(x-a)^{t+1}\right)$, and thus,

$$\text{trunc}(1/(x-a)^{t+1}, d) \;=\; (-1)^t/t!\cdot d^t/dx^t\;(\text{trunc}(1/(x-a), d)) + \sum_{i=d-t+1}^{d}\gamma_i\,x^i\,,\;\text{ where } \gamma_i \in \mathbb{F}\,.$$

Using the above identity and Lemma A.3, we can show that

$$L\left(\text{trunc}\left(\sum_{j\in[d_i]} b_{ij}/(x-a_i)^j, d\right)\right) \;=\; O(\log d\cdot d_i^2)\,.$$

To show this, note that $L(\text{trunc}(1/(x-a_i), d)) = O(\log d)$, and using Lemma A.3, we compute all its derivative till the d_i-th one which has a circuit of size $O(\log d\cdot d_i^2)$. Using the above identity, we can add $d_i - 1$ many monomials of the form cx^ℓ with $d - d_i + 2 \leq \ell \leq d$ (each monomial has trivial size of $O(\log d)$) to the circuit to obtain a circuit for $\text{trunc}\left(\sum_{j\in[d_i]} b_{ij}/(x-a_i)^j, d\right)$, which still has size $O(\log d \cdot d_i^2)$. Thus, doing it for each a_i for $i \in [k]$, one obtains that

$$\begin{aligned}
L\left(\text{trunc}\left(g(x)/h(x), d\right)\right) &= L\left(\text{trunc}\left(g_1(x)/h_1(x), d\right)\right) \\
&= L\left(g_1 \text{ div } h_1, d\right) + L\left(g_2/h_1, d\right) \\
&= O(s_1\, d_h) + L\left(\text{trunc}\left(\sum_{i\in[k]}\sum_{j\in[d_i]} b_{ij}/(x-a_i)^j, d\right)\right) \\
&= O((s + \log d_h)\, d_h^3) + O(\log d \cdot \sum_{i\in[k]} d_i^2) \\
&= O(s\, d_h^3 \log d)
\end{aligned}$$

□

Remark 5.1. Eventually, we can replace $g \in \mathbb{F}[[x]]$ with the given complexity $\text{trunc}(g, d) = s$ and show that the exact same proof as above, works.

5.2 Hardness results for rational function truncation.

Now, we give some evidence that we cannot expect logarithmic dependence on d_h in Theorem 5.1, unless integer factoring is *easy*. Before going into technicalities, we define *easy* sequence and constant-free complexity.

Definition 5.1 (Easy sequence)**.** A sequence $(a_n)_n$ of integers is said to be "easy to compute" if there exists a polynomial p such that straight line complexity of a_n, i.e. $\tau(a_n) \leq p(\log n)$, for $n \geq 1$.

If a sequence is not easy to compute, it is said to be hard. In fact, for most numbers N, one can show that $\tau(N) \geq \log N / \log\log N$ ("close" to the trivial upper bound) [DMS96; Mor97]. It is believed that $(d!)$ is hard to compute. In fact, its hardness is deeply connected to the infamous integer factoring problem. [Sha79] showed that $d!$ being easy to compute will imply factoring is easy in the non-uniform setting [4].

Constant-free circuit complexity. In the same spirit, one can define constant-free circuit complexity of polynomials where the given constants belong to the set $\{-1, 0, 1\}$[5]. We denote, $\tau(f)$ as the size of the minimal constant-free circuit computing f. Trivially, $L(f) \leq \tau(f)$.

It was shown in [And20] that $(a_n)_{n\in\mathbb{N}}$, where $a_n := \binom{2n}{n}$, is easy implies $(n!)_{n\in\mathbb{N}}$ is easy. This proof is similar to [Sha79]. This lemma will be crucial to prove the hardness result for truncations.

[4]However, this result does not imply that natural numbers can be factored in polynomial time in the Turing-Machine model, as the numbers used can be $\text{poly}(n)$-bits.

[5]To use 2^n in the circuit, one has to build up a circuit for 2^n, of size $\log n$, from 1; whereas in the usual sense of circuit size, constants are *free*. Thus, $f_d := 2^{2^d} x^d$ has $O(\log d)$-size circuit but *requires* $\Omega(d)$-size constant-free circuit.

Lemma 5.3 (Lemma 6.3 in [And20]). *If $a_n := \binom{2n}{n}$ has complexity $O(\log^c n)$, for some $c \in \mathbb{N}$, then $(n!)$ has complexity $O(\log^{c+1} n)$.*

In the following theorem, we show that constant-free complexity of the truncation of a power series with the denominator degree being *high*, is expected to be large, otherwise $n!$ is easy.

Theorem 5.2. *If $\tau\left(\text{trunc}\left(1/(1+x)^{d+1}, m\right)\right) = O(\log^c d)$, for some constant $c \in \mathbb{N}$ and $m \in \{d-1, d\}$, then $(n!)$ is easy. In fact, $\tau(n!) = O(\log^{c+1} n)$.*

Proof. From the power series expansion (Section 2), it is easy to see that,

$$\text{trunc}\left(1/(1+x)^{d+1}, m\right) = \sum_{i=0}^{m} \binom{-d-1}{i} x^i .$$

Let us notice $\binom{-d-1}{i} = (-d-1)(-d-2)\dots(-d-i)/i! = (-1)^i (d+i)!/i!\,d! = (-1)^i \binom{d+i}{i}$. Therefore,

$$\text{trunc}\left(1/(1+x)^{d+1}, d\right) - \text{trunc}\left(1/(1+x)^{d+1}, d-1\right) = (-1)^d \binom{2d}{d} x^d .$$

By assumption, $\tau\left((-1)^d \binom{2d}{d} x^d\right) = O(\log^c d)$. Therefore $\binom{2d}{d}$ has complexity $O(\log^c d)$, as desired (just by substituting $x = 1$, which gives an SLP). Invoking Lemma 5.3, we conclude. □

6 Hardness of Truncation of algebraic functions

In this section, we show conditional hardness of truncation of power series of algebraic functions with degree of its minpoly ≥ 3. In the first part, we show connection with integer factoring. In the second part, we show connection with computation of multiple of $(n!)$.

Throughout the section, we will be working with algebraic functions of the form $(1 + k^2 x)^{i/k}$, for $i, k \in \mathbb{N}$ with $i < k$. Here is a crucial claim. For a proof, we refer to Theorem G.2.

Theorem 6.1. *Fix $i, k \in \mathbb{N}$ with $i < k$. Then, $(1 + k^2 x)^{i/k} \in \mathbb{Z}[[x]]$, i.e. it is an integral power series.*

6.1 Hardness of truncation of algebraic functions and integer factoring

Here, we show that if the truncation of each $(1 + k^2 x)^{i/k}$, for $i \in [k-1]$, has small constant-free circuit, then one can factor n in $\text{poly}(\log n)$ time, in the non-uniform setting. This would readily imply the first part of Theorem 1.3.

Theorem 6.2. *Let $k \in \mathbb{N}$. If $\tau(\text{trunc}((1 + k^2 x)^{\frac{i}{k}}, d)) = O(\log^c d)$ (for some constant c) for all $i \in [k-1]$ then integer factorization (in the non-uniform setting) can be performed in polynomial time.*

Proof. Let, $(1+k^2x)^{\frac{i}{k}} = \sum_{d\geq 0} C_{d,i}\, x^d \in \mathbb{Z}[[x]]$, where the coefficient $C_{d,i}$ of x^d is equal to $\pm k^d(-i)\cdot(k-i)\cdot(2k-i)\cdots((d-1)k-i)/d!$. We see that the product of all $C_{d,i}$ is equal to:

$$\prod_{i\in[k-1]} C_{d,i} \;=\; \pm\frac{k^{(k-1)d}(k-1)!(dk)!}{(d!)^k k^d (kd-1)(kd-2)\cdots(kd-(k-1))}.$$

The assumption $\tau(\mathrm{trunc}((1+k^2x)^{\frac{i}{k}}, d)) = O(\log^c d)$ implies that $\tau(C_{d,i}) = O(\log^c d)$ (just by subtracting two consecutive truncations and substituting $x = 1$). This further implies that $\tau(\prod_{i\in[k-1]} C_{d,i}) = O(\log^c d)$, Let us define, for any $d \geq 1$,

$$N(d,k) \;:=\; \frac{k^{(k-2)d}(dk)!}{(d!)^k}\,.$$

We first argue that $N(d,k) \in \mathbb{N}$. This follows from the fact that $N(d,k) = \prod_{i\in[k-1]} C_{d,i}\cdot (kd-1)\cdots(kd-(k-1))/(k-1)!$, and $(k-1)!$ must divide $(kd-1)\cdots(kd-(k-1))$, by Fact G.1.

Further, since k is constant, it implies that $\tau(N(d,k)) = O(\log^c d)$ (because the extra term has trivial $O(\log d)$-complexity).

Now, we describe how to find a non-trivial factor of a given integer n. We assume that all the primes dividing n are larger than k; otherwise we can remove all the prime factors smaller than $k+1$ (since k is a constant).

The idea is to first find a positive integer t such that all the primes dividing n are in the interval $[t+1, tk]$, by using an iterative algorithm; if such a t does not exist we would have already found a non-trivial factor of n (by the algorithm). As an *invariant*, we maintain an integer m such that all the prime divisors of n are greater than m. We start with $m = k$ and compute $\gcd(N(m,k), n)$ at each iteration. Since all the primes dividing n are greater than m (by assumption), we get that $\gcd(N(m,k), n) = \gcd((mk)!, n)$. If the $\gcd((mk)!, n) \neq 1, n$, we must have already found a non-trivial factor of n and we are done. Otherwise, we can have two cases: either (i) $\gcd((mk)!, n) = 1$, or (ii) $\gcd((mk)!, n) = n$.

If $\gcd((mk)!, n) = 1$ then we set $m \leftarrow mk$ and continue (because in this case all the primes dividing n must be greater than mk). Otherwise we have $\gcd((mk)!, n) = n$, and hence, all the primes dividing n are in the interval $[m+1, mk]$ and we stop with $t \leftarrow m$. We know that $t \leq \lceil n/k \rceil$ and this uses at most $\log_k n = \log n$ iterations. So, this step has given us an integer t such that all the primes dividing n are in the interval $[t+1, tk]$, and the time taken is $\mathsf{poly}(\log n)$, due to only $\log n$ many iterations and each step takes $\mathsf{poly}(\log n)$-time due to the fact that $\tau(N(d,k)) = O(\log^c d)$ implies gcd computation can be done in $\mathsf{poly}(\log n)$ (by euclidean algorithm).

Once, we know that all the primes are in an interval of the form $[t+1, tk]$, we now try to reduce the length of it to k so that, we can simply *brute force* to get a factor of n, otherwise of course our algorithm would already find a factor. The length reduction part is similar to binary search algorithm that we describe below.

To find a positive integer s such that all the primes dividing n are in the interval $[sk+1, (s+1)k]$ (Or we find a non-trivial factor of n), again we use an iterative algorithm. As an invariant, we maintain two positive integers a, b such that all the prime divisors of n are in the interval $[ak+1, bk]$. We start with $a = 1, b = t$. Our invariant is trivially true at the start. At each iteration, we set $c = \lceil (a+b)/2 \rceil$ and compute $\gcd(N(c,k), n)$. Since $c \leq t$ and all the prime divisors of n are larger than t, we get that $\gcd(N(c,k),n) = \gcd((ck)!,n)$. Again, we argue in the same way as before. If the gcd is $\neq 1, n$, we have already found a non-trivial factor of n and we are done. Otherwise, we have two cases: either (i) $\gcd((ck)!,n) = 1$, or (ii) $\gcd((ck)!,n) = n$.

If $\gcd((ck)!,n) = 1$ then it is clear that all the primes dividing n are in the interval $[ck+1, bk]$ and hence we set $a \leftarrow c, b \leftarrow b$. If $\gcd((ck)!,n) = n$ then they all the primes dividing n are in the interval $[ak+1, ck]$ and hence we set $a \leftarrow a, b \leftarrow c$. This will terminate when $b - a \leq 1$. Hence we find the desired positive integer s. This uses at most $\log t = \log n$ iterations.

Now we just need to search for the prime divisors of n in the interval $[sk+1,(s+1)k]$ (an interval of constant length). Now, we brute force to finally find a non-trivial factor of n.

Similarly, this step also takes $\mathsf{poly}(\log n)$ as each gcd computation takes $\mathsf{poly}(\log n)$ time. So, we have successfully found a non-trivial factor of n by the end of this process, repeating this, we can get all the factors in $\mathsf{poly}(\log n)$-time and we are done. □

We also refer to Algorithm 1 in Appendix I.

6.2 Hardness of truncation of algebraic functions and complexity of multiple of $(n!)$

In this section, we show that easiness of truncation of $(1+k^2x)^{i/k}$ shows that a multiple of $n!$ must be easy. Note that, this may not imply that $n!$ is easy, however, from complexity-theoretic point-of-view, it is believed to be hard because of non-trivial implications. Shub & Smale [SS95] proved: *If $n!$ is ultimately hard to compute, then $\mathsf{P} \neq \mathsf{NP}$ over the field of complex numbers.*. Here, the computation is over Blum-Shub-Smale (BSS) model and can use complex numbers in the algorithm. In fact, a *stronger* version (known as τ-conjecture) connects $z(f)$, distinct integer roots of f with $\tau(f)$. Recently, [Dut21] showed that a similar conjecture, in the SOS-model, would in fact imply explicit constructions of rigid matrices & $\mathsf{VP} \neq \mathsf{VNP}$. For similar related works, we refer to [Koi11; Koi+15].

Before discussing and stating the formal result, we need an important notion of complexity, which is closely related to τ-complexity.

Definition 6.1 (Ultimately easy)**.** A sequence of integers (a_n) is *ultimately easy* if there exists another sequence (b_n) such that $\tau(a_n\, b_n) \leq \mathsf{poly}(\log n)$ for all large enough n.

Definition 6.2 (Ultimate complexity)**.** Define the *ultimate complexity* of an integer n as the minimum τ-complexity of its multiple, i.e. $\tau_1(n) = \min_{b \in \mathbb{Z}\setminus\{0\}} \tau(b \cdot n)$.

It is clear that Definition 6.1 can be stated wrt τ_1. We remark that $\tau_1(n_1 \cdot n_2) \leq \tau_1(n_1) + \tau_1(n_2) + 1$, for any $n_1, n_2 \in \mathbb{Z}$.

Following the same spirit as above, we prove the second part of Theorem 1.3.

Theorem 6.3. *Fix $k \in \mathbb{N}$. Suppose, for each $i \in [k-1]$, there exists some constant c such that $\tau(\mathrm{trunc}\left((1+k^2 \cdot x)^{i/k}, d\right) = O(\log^c d)$, for large enough d. Then, $(n!)_{n\in\mathbb{N}}$ is ultimately easy.*

Proof. Let, $(1+k^2x)^{\frac{i}{k}} = \sum_{d\geq 0} C_{d,i}\, x^d \in \mathbb{Z}[[x]]$. From the hypothesis, it follows that there exists c such that $\tau(C_{d,i}) \leq \log^c d$, for each $i \in [k-1]$ (subtract two consecutive terms and substitute $x = 1$). Further, from the proof in Section 6.1 (and following the same notation), we know that

$$\prod_{i\in[k-1]} C_{d,i} = \pm\frac{k^{(k-2)d}(k-1)!(dk)!}{(d!)^k(kd-1)(kd-2)\cdots(kd-(k-1))}.$$

Let us define, $a(d,k) := (dk)!/(d!)^k$. Note that, $a(d,k) \in \mathbb{N}$ (it is the multinomial coefficient $\binom{dk}{d,\ldots,d}$). Further, $k^{(k-2)d} \cdot a(d,k) = \prod_{i\in[k-1]} C_{d,i} \cdot (kd-1)\cdots(kd-(k-1))/(k-1)!$, and $(k-1)!$ must divide $(kd-1)\cdots(kd-(k-1))$, by Fact G.1. As k is constant, each $kd-i$ can be computed in $O(\log d)$-time trivially. Further, $\tau(\prod_{i\in[k-1]} C_{d,i}) \le O(\log^c d)$. As τ is additive over multiplication, it follows that

$$\tau(k^{(k-2)d} \cdot a(d,k)) \le O(\log^c d) \Rightarrow \tau_1(a(d,k)) \le O(\log^c d)\,.$$

Now we recurse by noticing the following trivial identity that $n! = n!/\left(\lfloor n/k\rfloor)!\right)^k \cdot \left((\lfloor n/k\rfloor)!\right)^k$.

We know by the above relation on $a(d,k)$ (and replacing $d := \lfloor n/k\rfloor$ for some integer n) that

$$\tau_1\left(\frac{(k\cdot\lfloor n/k\rfloor)!}{(\lfloor n/k\rfloor)!^k}\right) \le O(\log^c n)\,.$$

Further, any integer n can be written as $n = k\cdot\lfloor n/k\rfloor + j$ for some $j \le k-1$. Note that $k\cdot\lfloor n/k\rfloor + j$ has complexity at most $\log n$ for each $j \in [k-1]$. So, multiplying $k\cdot\lfloor n/k\rfloor + j$ for $j \in [k-1]$, it is straightforward to deduce that

$$\tau_1\left(\frac{n!}{(\lfloor n/k\rfloor!)^k}\right) \le O(\log^c n)\,. \tag{2}$$

As, $n! = n!/\left(\lfloor n/k\rfloor)!\right)^k \cdot \left((\lfloor n/k\rfloor)!\right)^k$, and $\tau_1\left((\lfloor n/k\rfloor!)^k\right) \le \tau_1\left(\lfloor n/k\rfloor!\right) + O(1)$; use Equation (2):

$$\begin{aligned}\tau_1(n!) &\le \tau_1\left(\lfloor n/k\rfloor!\right) + O(\log^c n) + O(1)\\ &\le \tau_1\left(\lfloor n/k^2\rfloor!\right) + O(\log^c n) + O(\log^c n) + O(1)\\ &\vdots\\ &\le \log_k n\cdot O(\log^c n) = O(\log^{c+1} n)\,.\end{aligned}$$

Therefore, $(n!)$ is ultimately easy to compute, as we wanted.

□

7 Complexity of the truncation of transcendental power series

In this section, we show examples where the truncation of transcendental power series is easy. We also complement this by showing the existence of *integral* transcendental power series which is conditionally hard.

7.1 The truncation of transcendental power series can be easy

In this section, we show two examples of integral transcendental power series whose truncations are easy.

Transcendental series corresponding to the Stern Sequence is easy

Definition 7.1 (The Stern sequence)**.** The sequence $(a_n)_{n\geq 0}$ given by $a_0 = 0, a_1 = 1$, and when $n \geq 1$, by $a_{2n} = a_n$ and $a_{2n+1} = a_n + a_{n+1}$, is called the Stern sequence.

The generating function $A(x) \overset{\text{def}}{=\!=} \sum a_n x^n$ of the Stern sequence has the following properties.

Theorem 7.1 (Lemma 2.1 and Theorem 2.2 in [Coo11])**.** *If $A(z)$ is the generating function of the Stern sequence, then*

1. $A(x^2) = A(x)\left(\frac{x}{x^2+x+1}\right)$.
2. *The function $A(x)$ is transcendental.*

Now we prove the following Theorem 7.2 which shows that its truncation has small circuit.

Theorem 7.2. *For the generating function $A(x)$ of the Stern sequence, we have*

$$L\left(\operatorname{trunc}\left(A(x), d\right)\right) \;=\; O(\log^2 d)\,.$$

Proof. By using Theorem 7.1, we obtain that:

$$A(x) = (x^2+1)A(x^2) + \frac{A(x^2)}{x}. \tag{3}$$

Suppose $B_d(x) \overset{\text{def}}{=\!=} \operatorname{trunc}\left(A(x), \lfloor \frac{d}{2} \rfloor + 1\right)$. Notice that the degree of $C_d(x) \overset{\text{def}}{=\!=} (x^2 + 1)B_d(x^2) + B_d(x^2)/x$ is at most $2\lfloor d/2 \rfloor + 4$ and $\operatorname{trunc}(C_d(x), d) = \operatorname{trunc}\left(A(x), d\right)$. Hence we can compute $\operatorname{trunc}\left(A(x), d\right)$ from $C_d(x)$ by subtracting at most 4 monomials, which can be done using $O(\log d)$ gates. Also $B_d(x)$ can be computed from $\operatorname{trunc}\left(A(x), \lfloor d/2 \rfloor\right)$ using $O(\log d)$ gates. Hence we obtain the following recurrence:

$$L\left(\operatorname{trunc}\left(A(x), d\right)\right) \;\leq\; L\left(\operatorname{trunc}\left(A(x), \lfloor d/2 \rfloor\right)\right) \;+\; O(\log d)\,.$$

This implies, $L\left(\operatorname{trunc}\left(A(x), d\right)\right) = O(\log^2 d)$. ☐

Transcendental power series whose coefficients are multiplicative

The sequence $(f_n)_{n\geq 0}$ is defined as: $f_0 = 1, f_1 = 1, f_2 = -1, f_p = 1$ for all odd primes p and $f_{ab} = f_a f_b$. We look at the corresponding generating function $F(x) \overset{\text{def}}{=\!=} \sum f_n x^n$.

Theorem 7.3 ([CB08, Theorem 2])**.** *The power series $F(x)$ is transcendental.*

Now we prove the following Theorem 7.4 which shows that truncation of $F(x)$ is easy.

Theorem 7.4. *For $F(x)$, we have $L\left(\text{trunc}\left(F(x),d\right)\right) = O(\log^2 d)$.*

Proof. We use the notation $\nu_2(m)$ to denote the highest power of 2 which divides $m \in \mathbb{N}$. We partition the set $[d]$ into $\lfloor \log d \rfloor$ sets $S_0, S_1, S_2, \ldots, S_{\lfloor \log d \rfloor}$ such that $k \in S_i$ iff $\nu_2(k) = i$. We define the set $O_m \stackrel{\text{def}}{=\!=} \{k \mid k \leq m \text{ and } k \text{ is odd}\}$. Now, notice that $S_i = \{2^i k \mid k \in O_{\lfloor d/2^i \rfloor}\}$. For a set $S \in \mathbb{N}$, we define the polynomial $g_S \stackrel{\text{def}}{=\!=} \sum_{i \in S} x^i$. Observe that:

$$\text{trunc}\left(F(x),d\right) \;=\; 1 \;+\; \sum_{i=1}^{\lfloor \log d \rfloor} (-1)^i g_{S_i}\,.$$

Trivially, $g_{S_i} = g_{O_{\lfloor d/2^i \rfloor}}(x^{2^i})$. Also notice that $g_{O_m} = g_{[m]} - g_{\lfloor \frac{m}{2} \rfloor}(x^2)$. Therefore, $L(g_{O_m}) = (\log m)$, which implies that $g_{S_i} = O(\log d)$. Hence, $L\left(\text{trunc}\left(F(x),d\right)\right) \;=\; O(\log^2 d)$. $\square$

Remark 7.1. Note that, there are power series like $\sum_{i \geq 0} x^{i!}$ which are transcendental and their truncations up to degree d are easy to compute. However, the series is highly *sparse* and degree-d truncations has only $\mathsf{poly}(\log d)$ monomials, hence the easiness is trivial. The examples we discover in this work are of dense power series.

7.2 The truncation of Transcendental power series can be hard

A sequence $(h_n)_{n \geq 0}$ is called holonomic if it satisfies the recurrence of the form:

$$a_r(n)\, h_{n+r} \;+\; a_{r-1}(n)\, h_{n+r-1} \;+\; \cdots \;+\; a_0(n) h_n \;=\; 0\,,$$

where a_i are polynomials in n. The corresponding generating function, $H(x) \stackrel{\text{def}}{=\!=} \sum h_n x^n$, is said to be a *holonomic function*.

Consider the holonomic sequence $f_n = (n!)$ defined by $f_0 = 1$ and $f_{n+1} - (n+1) f_n = 0$. Also consider the corresponding generating function $F(x) = \sum_{n \geq 0} n! x^n$. We now show that $F(x)$ is transcendental and that truncation of $F(x)$ is (conditionally) hard to compute. To this end, we need the following Lemma 7.1, which follows directly from Proposition 2 in [Kuh96].

Lemma 7.1 ([Kuh96]). *If $F(x) = \sum_{n \geq 0} f_n x^n$ is a power series in $\mathbb{C}[[x]]$ and the radius of convergence of $F(x)$ is zero then $F(x)$ is transcendental.*

Corollary 7.1. *The power series $F(x) = \sum_{n \geq 0} n! x^n$ is transcendental.*

Proof. It is clear that the radius of convergence of $F(x)$ is zero (follows from the ratio test). Hence Lemma 7.1 implies that $F(x)$ is transcendental. $\square$

Theorem 7.5. *If $\tau(\text{trunc}(F(x),d)) = \mathsf{poly}(\log d)$ then $(d!)$ has complexity $\mathsf{poly}(\log d)$.*

Proof. We know that $d! x^d = \text{trunc}(F(x),d) - \text{trunc}(F(x),d-1)$. Setting $x = 1$, we conclude. $\square$

8 SOS-complexity of truncation

A univariate polynomial $f(x) \in \mathbb{F}[x]$ over a field $\mathbb{F}$ is computed as a *sum-of-squares* (SOS) if

$$f = \sum_{i=1}^{s} c_i f_i^2 \,, \tag{4}$$

for some *top-fanin* s, where $f_i(x) \in \mathbb{F}[x]$ and $c_i \in \mathbb{F}$.

Remark 8.1. In real analysis, the SOS representation of a polynomial $f(x) \in \mathbb{R}[x]$, is defined where the coefficients $c_i > 0$ (in fact, we can take $c_i = 1$, by taking $\sqrt{c_i}$ inside f_i); thus the definition makes sense only for non-negative polynomials f. In this sense, (Equation (4)) is a *weighted* SOS. However, we will skip the term "weighted" (also because $\mathbb{F}$ can be $= \mathbb{C}$ here).

Definition 8.1 (Support-sum size $S_{\mathbb{F}}(f)$, [DST21])**.** The *size* of the representation of f in Equation (4) is the *support-sum*, the sum of the support size (or sparsity) of the polynomials f_i. The *support-sum size* of f, denoted by $S_{\mathbb{F}}(f)$, is defined as the minimum support-sum of f.

We will often refer to $S_{\mathbb{F}}(f)$ as the SOS-complexity of f. Note that, it is *sub-additive*, i.e. for two polynomials $f, g \in \mathbb{F}[x]$, we have $S_{\mathbb{F}}(f+g) \le S_{\mathbb{F}}(f) + S_{\mathbb{F}}(g)$.

Let $|f|_0$ denote the sparsity of f. For any field $\mathbb{F}$ of characteristic $\neq 2$, we have $|f|_0^{1/2} \le S_{\mathbb{F}}(f) \le \; 2\,|f|_0 + 2$. The lower bound can be shown by counting monomials. The upper bound is because $f = (f+1)^2/4 - (f-1)^2/4$. In particular, the SOS-model is *complete* when $\mathrm{char}(\mathbb{F}) \neq 2$. We will drop the subscript $\mathbb{F}$ when it is clear or unnecessary in the context.

Definition 8.2 (SOS-hardness, [DST21])**.** An "explicit" univariate $(f_d(x))_d$, where f_d is of degree d in $\mathbb{F}[x]$, is SOS-hard if $S(f_d) = \omega(d^{1/2})$.

Remark 8.2. If $S(f_d) = O(d^{1/2})$, we call (f_d) *SOS-easy*. Eg. $f_d = \sum_{i=0}^{d} x^i$ is SOS-easy (Lemma B.3).

It was shown in [DST21] that an SOS-hard family, with $S(f_d) \ge d^{1/2+\epsilon}$, for $\epsilon = \omega\left(\sqrt{\frac{\log\log d}{\log d}}\right)$, implies VP $\neq$ VNP. We want to characterize the SOS-easy and SOS-hard families, via natural operations like division and truncation. Towards that, we show the following Theorem 8.1. We assume $\mathbb{F} = \overline{\mathbb{F}}$ (otherwise we can go to small extensions).

Theorem 8.1 (Truncation is SOS-easy)**.** *Let $g, h \in \mathbb{F}[x]$ are both constant-degree polynomials s.t. $g/h \in \mathbb{F}[[x]]$. Then, truncation of g/h upto degree-d is SOS-easy,i.e. $S(\mathrm{trunc}(g/h, d)) = O(d^{1/2})$.*

Before proving this, we need a few important lemmas.

Lemma 8.1. *Let $f \in \mathbb{F}[x]$. Then, $S(f^{(k)}) \le O(k\,S(f))$.*

Proof. Let $f = \sum_{i=1}^{s} c_i f_i^2$ be the *minimal* SOS representation with $|f_i|_0 = t_i$, i.e. $\sum_{i\in[s]} t_i = S(f)$. Trivially, $f^{(k)} = \sum_{i\in[s]} f_i^{2^{(k)}}$. Using the Leibniz rule (Lemma B.2), we have

$$f_i^{2^{(k)}} = \begin{cases} 2\displaystyle\sum_{j=0}^{\frac{k}{2}-1} \binom{k}{j} \cdot f_i^{(j)} \cdot f_i^{(k-j)} + \binom{k}{k/2} \left(f_i^{(k/2)}\right)^2 & \text{if } k \equiv 0 \bmod 2 \\ 2\displaystyle\sum_{j=0}^{\frac{k-1}{2}} \binom{k}{j} \cdot f_i^{(j)} \cdot f_i^{(k-j)} & \text{if } k \equiv 1 \bmod 2 \end{cases}$$

Write each $f_i^{(j)} \cdot f_i^{(k-j)}$ as

$$f_i^{(j)} \cdot f_i^{(k-j)} \;=\; 1/4 \cdot (f_i^{(j)} + f_i^{(k-j)})^2 \;-\; 1/4 \cdot (f_i^{(j)} - f_i^{(k-j)})^2 \,.$$

Note that, $|f_i^{(j)}|_0 \le t_i$, for each $i \in [s]$ and $j \in [0,k]$. Thus, $f_i^{2(k)}$ has a representation with support-sum at most $\lceil \frac{k+1}{2} \rceil \cdot 4 \cdot t_i \le O(k\,t_i)$. Applying this to each $i \in [s]$ shows that $f^{(k)}$ has a SOS representation with support-sum at most $O\left(k \cdot \sum_i t_i\right) = O(k\,S(f))$; and the conclusion follows. $\square$

Lemma 8.2. *$S\left(\text{trunc}\ \left(1/(x-a)^j\,,\,d\right)\right) \le O\left(j \cdot \sqrt{d+j}\right)$, for any $j \in \mathbb{Z}_{\ge 0}$.*

Proof. Let $g_d(x) := \text{trunc}(1/x-a, d) = -1/a \cdot \left(\sum_{i=0}^{d} (x/a)^i\right)$. By differentiation, it follows that $(1/(x-a))^{(j-1)} = (-1)^{j-1} \cdot (j-1)! \cdot \left(1/(x-a)^j\right)$. Thus, one can conclude that

$$\text{trunc}\ \left(1/(x-a)^j\,,\,d\right) \;=\; (-1)^{j-1}/(j-1)! \cdot g_{d+j-1}^{(j-1)}(x)\,.$$

Note that, $S_{\mathbb{F}}(g_{d+j-1}(x)) = O\left(\sqrt{d+j-1}\right)$ (Lemma B.3). Using Lemma 8.1, the conclusion follows.

$\square$

Now, we are well-equipped to prove Theorem 8.1.

Proof of Theorem 8.1. This proof is very similar to that of Theorem 5.1. Let m be the highest power of x such that $x^m \mid h$ (i.e. $x^{m+1} \nmid h$). Note that, as $g/h \in \mathbb{F}[[x]]$, $x^m \mid g$ as well (Lemma B.1). Suppose, $\deg(h) =: d_h$. Thus $m \le d_h$. As d_h is a constant, so is m. Note that, $g_1 := g/x^m$ and $h_1 := h/x^m$ are both constant degree polynomials.

By definition, $g/h = g_1/h_1$. Let $g_2 := g_1 \bmod h_1$. Hence, $g_1/h_1 = g_1 \text{ div } h_1 + g_2/h_1$ and $\deg(g_2) < \deg(h_1)$. Finally, $\text{trunc}(g_1/h_1, d) = g_1 \text{ div } h_1 + \text{trunc}(g_2/h_1, d)$. However, $S(g_1 \text{ div } h_1) = O(1)$, as it has constant degree. Thus, it suffices to bound $S(\text{trunc}(g_2/h_1, d))$.

Suppose, h_1 factors over $\mathbb{F}[x]$, as $h_1 := \prod_{i\in[k]} (x-a_i)^{d_i}$. Moreover, using Lemma 5.1, we know that there are constants $a_i, b_{ij} \in \mathbb{F}$ such that

$$g_2(x)/h_1(x) \;=\; \sum_{i\in[k]} \sum_{j\in[d_i]} b_{ij}/(x-a_i)^j\,.$$

Therefore,

$$\text{trunc}(g_2/h_1, d) \;=\; \sum_{i\in[k]} \sum_{j\in[d_i]} b_{ij} \cdot \text{trunc}\left(1/(x-a_i)^j, d\right)\,.$$

Note that, d_i and k are constants. Using Lemma 8.2 and sub-additivity property of S, the conclusion follows. $\square$

Remark 8.3.

1. It is unclear how to extend this proof to non-constant degree polynomials g and h.

2. It is unclear whether $S(g/h)$ is small, when $h \mid g$ and $S(g)$ is small and $\deg(h)$ is small.

9 Constant-free complexity of $\bmod x^d$ and PosSLP

In this section, we investigate constant-free complexity of computing $\bmod x^d$ and its intrinsic connection with the positivity questions (i.e. PosSLP, for definition, see Problem 9.3).

Problem 9.1 (Modular complexity). *If we have $L(f) = s$ for some $f \in \mathbb{C}[x]$, what is complexity of $f \bmod x^d$?*

We prove a conditional lower bounds on the constant-free complexity of $f \mod x^d$.

Theorem 9.1. *If $\tau(f) = s$ implies $\tau(f \mod x^d) = \mathsf{poly}(s, \log d)$ for all $f \in \mathbb{Z}[x]$ then $\binom{2n}{n}_{n\in\mathbb{N}}$ has complexity $\mathsf{poly}(\log n)$.*

Proof. Suppose $m = 2^{\lceil \log d \rceil}$. Consider $\sqrt{1+4x}$, by Lemma F.1, we know that $\sqrt{1+4x} \in \mathbb{Z}[[x]]$. By using Newton's iteration, we can compute a polynomial $g \in \mathbb{Z}[x]$ such that $g \mod x^m = \sqrt{1+4x} \mod x^m$ and $\tau(g) = O(m) = (\log d)$ (Using Newton's iteration, see Theorem 6.5 in [Jin19], also [KT78]). Now $g \mod x^d = \mathrm{trunc}(\sqrt{1+4x}, d)$. Our assumption implies that $L(\mathrm{trunc}(\sqrt{1+4x}, d) = \mathsf{poly}(\log d)$. By a similar argument as in the proof of Theorem F.1, we get that $\binom{2n}{n}_{n\in\mathbb{N}}$ has complexity $\mathsf{poly}(\log n)$.

An alternative proof: we know $\tau((x+1)^{2n}) = O(\log n)$. Now see that $((x+1)^{2n}) \mod x^{n+1} - ((x+1)^{2n}) \mod x^n = x^n\binom{2n}{n}$. Therefore the assumption in the statement of the theorem implies that $\binom{2n}{n}_{n\in\mathbb{N}}$ has complexity $\mathsf{poly}(\log n)$. □

Theorem 9.1 demonstrates that computing remainders $\bmod x^d$ should be hard. Now we pose the following simpler problem.

Problem 9.2 (Special divisibility question). *If we have $\tau(f) = s$ for some $f \in \mathbb{C} = \mathbb{Z}[x]$, what is complexity of deciding if $f \mod x^d = 0$, i.e., decide if x^d divides f? Here the input is a circuit C of size s which computes f.*

It turns out that the question essentially reduces to decide the positivity of a number, computed by an SLP (Theorem 9.2).

Problem 9.3 (PosSLP [All+06]). *Given an SLP P (without divisions), decide if the integer computed by P is positive?*

Remark 9.1. [All+06] proved that that the Generic Task of Numerical Computation is polynomial-time equivalent to PosSLP and also showed that PosSLP lies in the counting hierarchy CH.

Proposition 1 (Folklore). *Given an an SLP P (with divisions) of length n computing a rational number $\frac{p}{q}$, there exist a division free SLP $Q = (q_0, q_1, \dots, q_{6n})$ such that $q_{6n-1} = p$ and $q_{6n} = q$.*

Proof. Suppose $P = (a_0, a_1, \dots, a_n)$. We split every gate a_i in P to two gates b_i and c_i such that $a_i = \frac{b_i}{c_i}$. Now notice that:

$$\frac{b_1}{c_1} + \frac{b_2}{c_2} = \frac{b_1c_2 + b_2c_1}{c_1c_2}.$$

$$\frac{b_1}{c_1} \cdot \frac{b_2}{c_2} = \frac{b_1b_2}{c_1c_2}.$$

This implies the claimed SLP Q. □

Lemma 9.1. *Given two SLP P_1, P_2 (with divisions) of length n computing the rational numbers $\frac{a}{b}$ and $\frac{p}{q}$ respectively, problem of deciding $\left|\frac{a}{b}\right| > \left|\frac{p}{q}\right|$ is in* $\mathsf{P}^{\text{PosSLP}}$.

Proof. By using Theorem 1, we first obtain SLPs $Q = (q_0, q_1, \dots, q_{6n})$ and $R = (r_0, r_1, \dots, r_{6n})$ such that $q_{6n-1} = a, q_{6n} = b$ and $r_{6n-1} = p, r_{6n} = q$. Using the PosSLP oracle, we find the signs of $\frac{a}{b}$ and $\frac{p}{q}$. After finding the signs, we can find SLPs (of length $6n+1$) which compute $|a|, |b|, |p|, |q|$. This implies an SLP of length $24n + 7$ which computes $|a|\,|q| - |p|\,|b|$. And deciding $|a|\,|q| - |p|\,|b| > 0$ also decides $\left|\frac{a}{b}\right| > \left|\frac{p}{q}\right|$. □

Theorem 9.2. *Problem 9.2 is in* $\mathsf{P}^{\text{PosSLP}}$.

Proof. We are given a constant free circuit C of size s which computes f. It is easy to see that $\deg(f) \leq 2^s$. We define $\|f\|_\infty$ to be the largest absolute value of coefficients of f. By induction, it is easy to see that $\|f\|_\infty \leq 2^{2^{2s}}$. Let M be any positive integer such that $M > 4 \cdot 2^s \cdot \|f\|_\infty$. Now we claim:

$$x^d \mid f \iff \left| f\left(\frac{1}{M}\right) \right| < \frac{1}{4M^{d-1}}.$$

Suppose $x^d \mid f$. Then we have $f = f_d x^d + f_{d+1} x^{d+1} + \cdots + f_n x^n$. In this case:

$$f\left(\frac{1}{M}\right) = \frac{1}{M^{d-1}} \left(\frac{f_d}{M} + \frac{f_{d+1}}{M^2} + \cdots + \frac{f_i}{M^{i-d+1}} + \cdots + \frac{f_n}{M^{n-d+1}} \right). \tag{5}$$

In Equation (5), the absolute value of each term $\frac{f_i}{M^{i-d+1}}$ is less than $\frac{1}{4 \cdot 2^s}$. Therefore $\left|f\left(\frac{1}{M}\right)\right| < \frac{1}{4M^{d-1}}$.

Now consider the case when $x^d \nmid g$. Let $m < d$ be the least positive integer such that x^m has non-zero coefficient in f. So $f = f_m x^m + g$ with $f_m \neq 0$ and $g = f_{m+1} x^{m+1} + \cdots + f_n x^n$. By using the argument above, we obtain $\left|g\left(\frac{1}{M}\right)\right| < \frac{1}{4M^m}$. Also, $|f_m x^m| \geq \frac{1}{M^m}$. Therefore $\left|f\left(\frac{1}{M}\right)\right| > \frac{3}{4}\frac{1}{M^m} \geq \frac{3}{4}\frac{1}{M^{d-1}} > \frac{1}{4M^{d-1}}$. Hence our claim is true.

Now notice that M has straight complexity at most $3s$. Therefore$f\left(\frac{1}{M}\right)$ has straight complexity (with divisions) at most $4s + 1$. Also, $\frac{1}{4M^{d-1}}$ has straight complexity (with divisions) at most $3s + 2 + 2\log d$. Therefore, by using Lemma 9.1 we can check $\left|f\left(\frac{1}{M}\right)\right| < \frac{1}{4M^{d-1}}$ in $\mathsf{P}^{\text{PosSLP}}$. Therefore Problem 9.2 is in $\mathsf{P}^{\text{PosSLP}}$. □

Theorem 9.2 and Remark 9.1 imply that Problem 9.2 lies in the counting hierarchy CH.

10 Conclusion

Our result on division elimination can be seen as evidence towards the possibility of a positive solution of Problem 1.1. Though the current techniques may not solve Problem 1.1, it is interesting to know division elimination (in circuits) is possible without using power series.

It is known that the decision problem of divisibility testing in the high degree regime: whether g (of size s and degree $\exp(s)$) is divisible by a polynomial h (of size s and degree $\exp(s)$) is NP-hard, even when h is a supersparse polynomial [Pla84]. However, its NP-hardness does not rule out the possibility of positive solution of Problem 1.1.

There are several avenues for extending our study of truncations of power series. Here, we remark that, Theorem 1.3 implies that, for any prime p, there is a simple algebraic function with degree of its minpoly $= p$, such that the truncation is conditionally hard. But it is not clear whether it is true for composite (because i/k can reduce, when $k \neq p$).

One can also investigate truncation of algebraic power series over characteristic p. [BCD16] showed that n-th coefficient of an algebraic power series over characteristic p can be computed in $O(\log n, p)$-time. One can study truncations of power series with $0-1$ coefficients and relate their hardness with classical assumptions in complexity, eg. truncated Θ-functions [NP18].

Here are some immediate questions of interest which require rigorous investigation.

1. Can we remove the degree condition on g in Theorem 1.2?
2. Does Theorem 1.2 hold in the border sense? Note that, the degree of the approximate circuit can have degree $> d$ and thus homogenization seems necessarily blowing the complexity in d.
3. Can we show that the truncation of *any* "simple" algebraic function (satisfying a minpoly of degree > 2 with bounded coefficients) must be conditionally hard in Theorem 1.3? In particular, can we show that $(1+9x)^{1/3}$ is conditionally hard?
4. Does Theorem 1.1 hold in the SOS-complexity regime?

Acknowledgments

We thank Himanshu Shukla for several discussions on the complexity of truncated power series, and for bringing the reference [CB08] to our attention. P. D. would like to thank CSE, IIT Kanpur for the hospitality. A. S. would like to thank the Institute of Theoretical Computer Science at Ulm University for the hospitality. We thank Thomas Thierauf and Nitin Saxena for discussions and feedback on the draft.

References

[Ald84] Alexander Alder. “Grenzrang und Grenzkomplexität aus algebraischer und topologischer Sicht.” PhD thesis. Zentralstelle der Studentenschaft, 1984.

[All+06] Eric Allender et al. “On the Complexity of Numerical Analysis.” In: *SIAM Journal on Computing* 38 (Jan. 2006). Preliminary version in the 21st Annual IEEE Conference on Computational Complexity (CCC’06).

[And20] Robert Andrews. “Algebraic Hardness Versus Randomness in Low Characteristic.” In: *35th Computational Complexity Conference (CCC 2020)*. Vol. 169. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 37:1–37:32. ISBN: 978-3-95977-156-6.

[BR88] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Berlin, Heidelberg: Springer-Verlag, 1988. ISBN: 0387186263.

[BJ19] Markus Bläser and Gorav Jindal. “On the Complexity of Symmetric Polynomials.” In: 10th *Innovations in Theoretical Computer Science Conference (ITCS'19)*. Vol. 124. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 47:1–47:14.

[BCD16] Alin Bostan, Gilles Christol, and Philippe Dumas. “Fast computation of the Nth term of an algebraic series over a finite prime field.” In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation (ISSAC'16)*. 2016, pp. 119–126.

[BIZ18] Karl Bringmann, Christian Ikenmeyer, and Jeroen Zuiddam. “On Algebraic Branching Programs of Small Width.” In: *J. ACM* 65.5 (2018). (Preliminary version in the 32nd Computational Complexity Conference (CCC'17), pp. 1–29.

[Bür04] Peter Bürgisser. “The complexity of factors of multivariate polynomials.” In: *Foundations of Computational Mathematics* 4.4 (2004), pp. 369–396.

[Bür09] Peter Bürgisser. “On defining integers and proving arithmetic circuit lower bounds.” In: *Computational Complexity* 18.1 (2009), pp. 81–103.

[BCS13] Peter Bürgisser, Michael Clausen, and Amin Shokrollahi. *Algebraic complexity theory*. Vol. 315. Springer Science & Business Media, 2013.

[CC86] David V Chudnovsky and Gregory V Chudnovsky. “On expansion of algebraic functions in power and Puiseux series, I.” In: *Journal of Complexity* 2.4 (1986), pp. 271–294.

[Coo11] Michael Coons. “THE TRANSCENDENCE OF SERIES RELATED TO STERN'S DIATOMIC SEQUENCE.” In: *International Journal of Number Theory* 06 (Nov. 2011). DOI: 10.1142/S1793042110002958.

[CB08] Michael Coons and Peter Borwein. “Transcendence of Power Series for Some Number Theoretic Functions.” In: *Proceedings of the American Mathematical Society* 137 (July 2008). DOI: 10.1090/S0002-9939-08-09737-2.

[DMS96] Wellington De Melo and Benar Fux Svaiter. “The cost of computing integers.” In: *Proceedings-American Mathematical Society* 124 (1996), pp. 1377–1378.

[DL78] Richard A. Demillo and Richard J. Lipton. “A probabilistic remark on algebraic program testing.” In: *Information Processing Letters* 7.4 (1978), pp. 193 –195. ISSN: 0020-0190.

[Dut21] Pranjal Dutta. “Real tau-Conjecture for sum-of-squares: A unified approach to lower bound and derandomization.” In: *16th International Computer Science Symposium in Russia (CSR 2021)*. 2021.

[DSS18] Pranjal Dutta, Nitin Saxena, and Amit Sinhababu. “Discovering the roots: Uniform closure results for algebraic classes under factoring.” In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 2018, pp. 1152–1165.

[DST21] Pranjal Dutta, Nitin Saxena, and Thomas Thierauf. "A Largish Sum-Of-Squares Implies Circuit Hardness and Derandomization." In: *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Vol. 185. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, 23:1–23:21.

[GS80] J.von zur Gathen and V. Strassen. "Some polynomials that are hard to compute." In: *Theoretical Computer Science* 11.3 (1980), pp. 331 –335. ISSN: 0304-3975.

[GMQ16] Joshua A. Grochow, Ketan D. Mulmuley, and Youming Qiao. "Boundaries of VP and VNP." In: *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Vol. 55. 2016, 34:1–34:14.

[Gro+20] Joshua A Grochow et al. "Complexity in ideals of polynomials: questions on algebraic complexity of circuits and proofs." In: *Bulletin of EATCS* 2.130 (2020).

[Jin19] Gorav Jindal. "On approximate polynomial identity testing and real root finding." PhD thesis. Saarland University, 2019. DOI: `http://dx.doi.org/10.22028/D291-29880`.

[Kal86] Erich Kaltofen. "Uniform closure properties of p-computable functions." In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 1986, pp. 330–337.

[Kal87] Erich Kaltofen. "Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials." In: *Proceedings of the* 19^{th} *annual ACM symposium on Theory of computing (STOC'87)*. 1987, pp. 443–452.

[Koi05] Pascal Koiran. "Valiant's model and the cost of computing integers." In: *computational complexity* 13.3 (2005), pp. 131–146.

[Koi11] Pascal Koiran. "Shallow circuits with high-powered inputs." In: *Innovations in Computer Science (ICS)* (2011).

[KP11] Pascal Koiran and Sylvain Perifel. "Interpolation in Valiant's theory." In: *Computational Complexity* 20.1 (2011), pp. 1–20.

[Koi+15] Pascal Koiran et al. "A τ-Conjecture for Newton Polygons." In: *Foundations of computational mathematics* 15.1 (2015), pp. 185–197.

[Kuh96] FV Kuhlmann. *On convergent power series*. 1996.

[KT78] Hsiang Kung and Joseph Traub. "All Algebraic Functions Can Be Computed Fast." In: *J. ACM* 25 (Apr. 1978), pp. 245–260.

[LR09] Dick Lipton and Ken Regan. *Factoring and Factorials*. Feb. 2009. URL: `https://rjlipton.wordpress.com/2009/02/23/factoring-and-factorials/`.

[Lip78] Richard J Lipton. "Polynomials with 0-1 coefficients that are hard to evaluate." In: *SIAM Journal on Computing* 7.1 (1978). Preliminary version in the 16^{th} Annual Symposium on Foundations of Computer Science (FOCS 1975), pp. 61–69.

[Lip94] Richard J Lipton. "Straight-line complexity and integer factorization." In: *International Algorithmic Number Theory Symposium (ANTS 94)*. Springer. 1994, pp. 71–79.

[LS78] Richard J Lipton and Larry J Stockmeyer. "Evaluation of polynomials with super-preconditioning." In: *Journal of Computer and System Sciences* 16.2 (1978), pp. 124–139.

[Mah14] Meena Mahajan. "Algebraic Complexity Classes." In: *Perspectives in Computational Complexity*. Springer, 2014, pp. 51–75.

[Mor97] Carlos Moreira. "On asymptotic estimates for arithmetic cost functions." In: *Proceedings of the American Mathematical Society* 125.2 (1997), pp. 347–353.

[Mul12] Ketan D Mulmuley. "The GCT program toward the P vs. NP problem." In: *Communications of the ACM* 55.6 (2012), pp. 98–107.

[NP18] Danny Nguyen and Igor Pak. "Complexity of short generating functions." In: *Forum of Mathematics, Sigma*. Vol. 6. Cambridge University Press. 2018.

[Ore22] Øystein Ore. "Über höhere kongruenzen." In: *Norsk Mat. Forenings Skrifter* 1.7 (1922), p. 15.

[Pak18] Igor Pak. "Complexity problems in enumerative combinatorics." In: *Proceedings of the International Congress of Mathematicians—Rio de Janeiro 2018. Vol. IV. Invited lectures*. World Sci. Publ., Hackensack, NJ, 2018, pp. 3153–3180.

[Pla84] David A Plaisted. "New NP-hard and NP-complete polynomial and integer divisibility problems." In: *Theoretical Computer Science* 31.1-2 (1984). Preliminary in the 17th Annual Symposium on Foundations of Computer Science (FOCS 1976), pp. 125–138.

[Sch80] Jacob T Schwartz. "Fast probabilistic algorithms for verification of polynomial identities." In: *Journal of the ACM (JACM)* 27.4 (1980), pp. 701–717.

[Sha79] Adi Shamir. "Factoring numbers in O (logn) arithmetic steps." In: *Information Processing Letters* 8.1 (1979), pp. 28–31.

[SY10] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions." In: *Foundations and Trends® in Theoretical Computer Science* 5.3–4 (2010), pp. 207–388.

[SS95] Michael Shub and Steve Smale. "On the intractability of Hilbert's Nullstellensatz and an algebraic version of "NP $\neq$ P?"" In: *Duke Math. J.* 81.1 (1995). A celebration of John F. Nash, Jr., 47–54 (1996). ISSN: 0012-7094. DOI: `10.1215/S0012-7094-95-08105-8`. URL: `https://doi.org/10.1215/S0012-7094-95-08105-8`.

[Str73] Volker Strassen. "Vermeidung von Divisionen." In: *Journal für die reine und angewandte Mathematik* 264 (1973), pp. 184–202.

[Val82] L Valiant. "Reducibility by algebraic projections in: Logic and Algorithmic." In: *Symposium in honour of Ernst Specker*. 1982, pp. 365–380.

[Val79] Leslie G Valiant. "Completeness classes in algebra." In: *Proceedings of the 11th Annual ACM symposium on Theory of computing*. ACM. 1979, pp. 249–261.

[VZGG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.

[Wac] Wact. *Some Accessible Open Problems*. Workshop on Algebraic Complexity Theory (WACT 2016).

[Wag86] Klaus W Wagner. "The complexity of combinatorial problems with succinct input representation." In: *Acta informatica* 23.3 (1986), pp. 325–356.

[Zip79] Richard Zippel. "Probabilistic Algorithms for Sparse Polynomials." In: *Proceedings of the International Symposium on Symbolic and Algebraic Computation*. EUROSAM '79. 1979, pp. 216–226. ISBN: 3-540-09519-5.

A Basics in Arithmetic circuit complexity

An *arithmetic circuit* over a field $\mathbb{F}$ is a layered directed acyclic graph that uses field operations $\{+, \times\}$ and computes a polynomial. It can be thought of as an algebraic analog of Boolean circuits. The leaf nodes are labeled with the input variables $x_1, \ldots, x_n$ and constants from $\mathbb{F}$. Other nodes are labeled as addition and multiplication *gates*. The root node outputs the polynomial computed by the circuit. At times, we also use $\div$ gate in the circuit.

For a polynomial f, the size of the smallest circuit computing f is denoted by $L(f)$, it is the *arithmetic circuit complexity* of f. Here, size of an arithmetic circuit is assumed to be the number of nodes (variables included).

In *complexity classes*, we specify an upper bound on these parameters. Valiant's class VP contains the families of n-variate polynomials of degree $\mathsf{poly}(n)$ over $\mathbb{F}$, computed by circuits of $\mathsf{poly}(n)$-size. The class VNP can be seen as a non-deterministic analog of the class VP. A family of n-variate polynomials $(f_n)_n$ over $\mathbb{F}$ is in VNP if there exists a family of polynomials $(g_n)_n$ in VP such that for every $\mathbf{x} = (x_1, \ldots, x_n)$ one can write $f_n(\mathbf{x}) = \sum_{w \in \{0,1\}^{t(n)}} g_n(\mathbf{x}, w)$, for some polynomial $t(n)$ which is called the *witness size*. It is straightforward to see that VP $\subseteq$ VNP and *conjectured* to be different (Valiant's Hypothesis [Val79]). *Equivalently*, symbolic permanent$_{n\times n}$ requires $n^{\omega(1)}$ size circuit.

One can define the class VP_0 (respectively, VNP_0) as the analogue of VP (respectively, VNP) in the constant-free regime. For more details see [Koi05; KP11; Mah14; SY10; BCS13].

Coefficient-extraction in arithmetic circuits is easy using interpolation, see the folklore lemma below, for a proof see [SY10].

Lemma A.1 (Coefficient-Extraction). *Let $L(f) = s$ with $f \in \mathbb{F}[\mathbf{x}]$ and $f = \sum_{0\le i\le d} f_i x_n^i$ with $f_i \in \mathbb{F}[x_1, x_2, \ldots, x_{n-1}]$. Then there is a circuit C of size $O(sd^2)$ computing $f_0, f_1, \ldots, f_d$.*

The next lemma is a homogenization trick, used in [Str73]. For a proof, see [SY10, Theorem 2.2].

Lemma A.2 (Homogenization). *If f has an arithmetic circuit of size s, then for any d, there is a circuit of size $O(sd^2)$ computing $Hom_{\le d} f$.*

Lemma A.3. *Let f be a polynomial $\mathbb{F}[x]$, computed by a size s circuit C. Then, there exists a circuit C' of size $O(sm^2)$ which computes $f, f^{(1)}, f^{(2)}, \ldots, f^{(m)}$.*

Proof. We split every G gate in C to $n+1$ gates $G_0, \ldots, G_m$ in C'. The property we want is that if the gate G is computing the polynomial g in C then G_k computes the polynomial $g^{(k)}$ in C'. Suppose G is a $+$ gate in C with children gates computing the polynomials g_1 and g_2. Now we know that $g^{(k)} = g_1^{(k)} + g_2^{(k)}$. Thus we can easily propagate the derivatives on addition/subtraction gates. If G is a $\times$ gate then using Lemma B.2, we know that:

$$(g_1 g_2)^{(k)} = \sum_{i=0}^{k} \binom{k}{i} g_1^{(k-i)} g_2^{(i)}$$

Thus we can computes $g, g^{(1)}, g^{(2)}, \ldots, g^{(m)}$ using additional $O(m^2)$ gates. Therefore C' has $O(sm^2)$ gates. $\square$

Polynomial Identity Testing (PIT) is a fundamental question in algebraic complexity. It asks for an algorithm to test the zeroness of a given algebraic circuit via mere query access. It is known that efficient evaluation at random points lead to a randomized polynomial time algorithm for PIT. This is known as *Polynomial Identity Lemma* [Ore22; DL78; Zip79; Sch80].

Lemma A.4 (Polynomial Identity Lemma). *Let $p(\mathbf{x})$ be an n-variate nonzero polynomial of degree d. Let $S \subseteq \mathbb{F}$ be a finite set. Then,*

$$\Pr_{\boldsymbol{\alpha} \sim S^n}[p(\boldsymbol{\alpha}) = 0] \;\leq\; d/|S|\,.$$

Here, $\boldsymbol{\alpha} \in S^n$ is picked independently and uniformly at random.

B Basic mathematical tools

Lemma B.1 (Power series valuation). *Let $g, h \in \mathbb{F}[x]$ such that $g/h \in \mathbb{F}[[x]]$. Let m (respec. n) be the highest power dividing g (respec. h) i.e. $x^m \mid g$ and $x^{m+1} \nmid g$ (respec. for h). Then, $m \geq n$.*

Proof. Suppose, $m < n$. Note that, there exists $0 \neq \alpha \in \mathbb{F}$, such that $h = \alpha\, x^n \cdot (1 + x\,\tilde{h})$, for some $\tilde{h} \in \mathbb{F}[x]$. Similarly, let $g = \beta\, x^m \cdot (1 + x\,\tilde{g})$, for some $\tilde{g} \in \mathbb{F}[x]$ and $\beta \in \mathbb{F}$. Thus,

$$\begin{aligned}\frac{g}{h} &= \frac{\beta}{\alpha} \cdot x^{m-n} \cdot \frac{1 + x\,\tilde{g}}{1 + x\,\tilde{h}} \\ &= \frac{\beta}{\alpha} \cdot x^{m-n} \cdot (1 + x\,\tilde{g}) \cdot (1 + x\,\tilde{h} + (x\,\tilde{h})^2 + \cdots) \\ &\notin \mathbb{F}[[x]]\,, \text{ a contradiction}\,.\end{aligned}$$

$\square$

Lemma B.2 (General Leibniz rule). *If f and g are k-time differentiable functions, then*

$$(fg)^{(k)} \;=\; \sum_{i=0}^{k} \binom{k}{i} f^{(k-i)}\, g^{(i)}\,.$$

Lemma B.3. *Define $f_d := \sum_{i=0}^{d} x^i$. Then, $S_{\mathbb{F}}(f_d) \;\leq\; 9 \cdot d^{1/2}$, over any field $\mathbb{F}$.*

Proof of Lemma B.3. Fix some $n \in \mathbb{N}$. Note that,

$$f_{n^2-1}(x) \;=\; \left(1 \,+\, x \,+\, \ldots \,+\, x^{n-1}\right) \cdot \left(1 \,+\, x^n \,+\, \ldots \,+\, x^{n(n-1)}\right)\;.$$

As each factor has n terms, we can write the product as sum of two squares with each polynomial having at most $2n$ terms. Therefore,

$$S_{\mathbb{F}}(f_{n^2-1}(x)) \;\leq\; 4\,n\,. \tag{6}$$

For general d, let $n \in \mathbb{N}$ be such that $n^2 - 1 \leq d < (n+1)^2 - 1$. By definition,

$$f_d(x) \;=\; f_{n^2-1}(x) + x^{n^2} \cdot f_{d-n^2}(x)\,.$$

Note that, $|f_{d-n^2}(x)|_0 \;\leq\; d + 1 - n^2 \leq 2n$. Thus, using the trivial upper bound on $S(f)$, we must have

$$S_{\mathbb{F}}(x^{n^2} \cdot f_{d-n^2}(x)) \;\leq\; 2 \cdot (2n+1)\,. \tag{7}$$

Combining Equation (6) and Equation (7), we get that $S_{\mathbb{F}}(f_d(x)) \leq 8 \cdot \lceil \sqrt{d+1} \rceil \,+\, 2$, and the conclusion follows. $\square$

C Monic transformation

Given any polynomial $p(\mathbf{x})$ in variables $\mathbf{x} = (x_1, \ldots, x_n)$, there is a standard trick to make it monic in x_n by applying a linear transformation on the variables: for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{n-1}) \in \mathbb{F}^{n-1}$, let

$$\tau_{\boldsymbol{\alpha}} : x_i \mapsto \alpha_i x_n + x_i \,,$$

for $i \in [n-1]$, and $x_n \mapsto x_n$. Note that $\deg(\tau_{\boldsymbol{\alpha}}(p)) \le \deg(p)$ [it may *decrease* because of non-trivial cancellations]. It is easy to see that $\tau_{\boldsymbol{\alpha}}$ is an *invertible* map. We show that $\tau_{\boldsymbol{\alpha}}(p)$ is monic in x_n, for a *random* transformation $\tau_{\boldsymbol{\alpha}}$ i.e. when $\boldsymbol{\alpha} \in \mathbb{F}^{n-1}$ chosen randomly. In fact, we show that this map can simultaneously make polynomials monic given that the field $\mathbb{F}$ is sufficiently large.

Lemma C.1 (Monic Transformation)**.** *Let $p_1(\mathbf{x}), \ldots, p_m(\mathbf{x})$ be m-many polynomial of degree d. Let $S \subseteq \mathbb{F}$ be a finite set. For $\boldsymbol{\alpha} \in S^{n-1}$, picked independently and uniformly at random,*

$$\Pr\left[\bigwedge_{i=1}^{m} \tau_{\boldsymbol{\alpha}}(p_i(\mathbf{x})) \text{ is monic in } x_n\right] \ge 1 - \frac{dm}{|S|}\,.$$

Proof. Consider the terms of degree d of a non-zero polynomial $p \in \mathbb{F}[\mathbf{x}]$. Define the set

$$T := \{\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \;\mid\; |\boldsymbol{\beta}|_0 = \sum_i \beta_i = d, \text{ and } \mathrm{coef}_{\mathbf{x}^{\boldsymbol{\beta}}}(p) \neq 0\}\,.$$

We also denote $\boldsymbol{\beta}' = (\beta_1, \ldots, \beta_{n-1})$, the first $n-1$-coordinates of $\boldsymbol{\beta}$, and similarly $\mathbf{x}' = (x_1, \ldots, x_{n-1})$. Note that, $\tau_{\boldsymbol{\alpha}}(\mathbf{x}^{\boldsymbol{\beta}}) = \boldsymbol{\alpha}^{\boldsymbol{\beta}'} \cdot x_n^d + (\text{lower terms in } x_n)$.

Observe that the homogeneous component of degree d in $\tau_{\boldsymbol{\alpha}}(p)$ can be written as $a_{d,p}(\mathbf{x}) = \sum_{\boldsymbol{\beta} \in T} c_{\boldsymbol{\beta}} \cdot \tau_{\boldsymbol{\alpha}}(\mathbf{x}^{\boldsymbol{\beta}})$, for some constants $c_{\boldsymbol{\beta}}$. Trivially, $a_{d,p}$ is a nonzero polynomial, and moreover,

$$a_{d,p}(\boldsymbol{\alpha}) = \Big(\sum_{\boldsymbol{\beta} \in T} c_{\boldsymbol{\beta}} \boldsymbol{\alpha}^{\boldsymbol{\beta}'}\Big) \cdot x_n^d + (\text{lower terms in } x_n).$$

In order to make $\tau_{\boldsymbol{\alpha}}(p)$ monic in x_n, we want $(\sum_{\boldsymbol{\beta} \in T} c_{\boldsymbol{\beta}} \boldsymbol{\alpha}^{\boldsymbol{\beta}'}) \neq 0$. So, define, another polynomial $b_{d,p}(\mathbf{x}') = (\sum_{\boldsymbol{\beta} \in T} c_{\boldsymbol{\beta}} \mathbf{x}'^{\boldsymbol{\beta}'})$. It can have degree atmost d.

As we want each $\tau_{\boldsymbol{\alpha}}(p)$ monic where $p = p_m(\mathbf{x})$, it suffices to find $\boldsymbol{\alpha}$ such that $\prod_{i \in [m]} b_{d,p_i}(\boldsymbol{\alpha}) \neq 0$. Note that, $\deg\left(\prod_{i \in [m]} b_{d,p_i}(\mathbf{x})\right) \le d \cdot m$. Thus, when we pick $\boldsymbol{\alpha}$ at random, the probability that $\prod_{i \in [m]} b_{d,p_i}(\boldsymbol{\alpha}) = 0$, is at most $\le dm/|S|$, from Lemma A.4. Hence, the conclusion follows. $\square$

D Truncation is hard

One can show that truncation (or cost of mod) cannot be *expected* to be logarithmically dependent on the precision (unless permanent is *easy*), reminiscent to [Val82]. We sketch the proof for the sake of completeness.

Lemma D.1 (Folkore)**.** *Suppose, for any polynomial $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$ of size s, $Hom_{\le d} f(\mathbf{x})$ can be computed by circuit of size $\mathsf{poly}(s, \log d)$, then $\mathsf{VP} = \mathsf{VNP}$.*

Proof. Consider the following polynomial of n^2+n variables, where we denote $\mathbf{y} = (y_1, \ldots, y_n)$, and $\mathbf{z} = (z_{1,1}, \ldots, z_{n,n})$:

$$g(\mathbf{y}, \mathbf{z}) \;:=\; \prod_{i \in [n]} \left(\sum_{j \in [n]} y_j z_{i,j} \right)$$

Observe that coefficient of $y_1 \ldots y_n$ in g is nothing but $\mathrm{perm}(z_{1,1}, \ldots, z_{n,n})$, the permanent polynomial on variables $\mathbf{z}$. Further, each $\mathrm{coef}_{\mathbf{y}^{\alpha}}(g)$ is a *multilinear* polynomial in $\mathbf{z}$, of degree n. Consider a new polynomial f by substituting $y_i = x^{(n+1)^{i-1}}$ (Kronecker substitution). In particular, let

$$f(x, \mathbf{z}) \;:=\; g(x, x^{n+1}, x^{(n+1)^2}, \ldots, x^{(n+1)^{n-1}}, \mathbf{z}).$$

As Kronecker substitution gives different weights to different monomials and the maximum degree can be $n \cdot (n+1)^{n-1}$ (i.e. when y_n^n gets substituted), it is easy to deduce that

$$f \;=\; \sum_{k=0}^{n \cdot (n+1)^{n-1}} c_k(\mathbf{z}) \cdot x^k \,.$$

Here, each $c_k(\mathbf{z})$ is a multilinear polynomial of degree n. Moreover, from the above discussion,

$$c_j(z_{1,1}, \ldots, z_{n,n}) \;=\; \mathrm{perm}(z_{1,1}, \ldots, z_{n,n}) \text{ ,where } j \;:=\; 1 + (n+1) + \ldots + (n+1)^{n-1} \,.$$

In that case, the degree of $c_j(\mathbf{z}) \cdot x^j$ is $m := j + n = n^{O(n)}$. Thus, we can conclude that $\mathrm{Hom}_{=m}(f) = c_j(\mathbf{z}) \cdot x^j = \mathrm{perm}(\mathbf{z}) \cdot x^j$.

Observe that $L(g) \le \mathsf{poly}(n)$. After Kronecker substitution, the blowup in size in still poly i.e. $L(f) \le \mathsf{poly}(n)$. Hence, assuming the hypothesis, we would get that

$$\mathrm{perm}(\mathbf{z}) \cdot x^j \;=\; \mathrm{Hom}_{=m}(f) \;=\; \mathrm{Hom}_{\le m}(f) \;-\; \mathrm{Hom}_{\le m-1}(f) \,,$$

has $\mathsf{poly}(n)$ size circuit. This implies $\mathrm{perm}(\mathbf{z})$ has $\mathsf{poly}(n)$ size circuit (by substituting $x = 1$), i.e. $\mathsf{VP} = \mathsf{VNP}$. $\square$

E Details for Section 3

Here we prove Lemma 3.1. For completeness, we again state the lemma.

Lemma E.1. *Suppose $g = \sum_{i \le d_1} g_i x^i$ and $h = x^{d_2} + \sum_{i < d_2} h_i x^i$, in $\mathbb{F}[\mathbf{x}]$. Suppose $g = hq + r$, with $r = \sum_{i < d_2} r_i x^i$ and $q = \sum_{i \le d_1 - d_2} q_i x^i$. Then, there is a circuit of size $O(d_1 d_2)$, whose inputs are all h_i, g_i and outputs are all r_i, q_i.*

Proof. We shall denote the desired circuit by C_{d_1,d_2}. So we want:

$$C_{d_1,d_2}(g_0, g_1, \ldots, g_{d_1}, h_0, h_1, \ldots, h_{d_2}) = (r_1, r_2, \ldots, r_{d_2-1}, q_0, q_1, \ldots, q_{d_1-d_2}).$$

If $d_1 < d_2$, we know that $q = 0$. Hence:

$$C_{d_2-1,d_2}(g_0, g_1, \ldots, g_{d_1-1}, h_0, h_1, \ldots, h_{d_1}) = (g_1, g_2, \ldots, g_{d_1-1}).$$

If $d_1 > d_2$, we perform a long division step:

$$g \leftarrow g - h \cdot x^{d_1-d_2} \cdot g_{d_1} = \sum_{i \le d_1-d_2-1} g_i\, x^i \;+\; \sum_{i \ge d_1-d_2}^{d_1-1} \left(g_i - h_{i-(d_1-d_2)}\, g_{d_1} \right) x^i.$$

Note that, we can set $q_{d_1-d_2} = g_{d_1}$. Define:

$$\mathbf{g} \overset{\text{def}}{=\!=} (g_0, g_1, \ldots, g_{d_1-d_2-1}, g_{d_1-d_2} - h_0 g_{d_1}, \ldots, g_{d_1-1} - h_{d_2-1} g_{d_1}) .$$

Then we have:

$$C_{d_1,d_2}(g_0, g_1, \ldots, g_{d_1}, h_0, h_1, \ldots, h_{d_2}) = (C_{d_1-1,d_2}(\mathbf{g}, h_0, h_1, \ldots, h_{d_2}), g_{d_1}). \tag{8}$$

Hence if $S(d_1, d_2)$ is the size of C_{d_1,d_2} then Equation (8) implies that $S(d_1, d_2) = S(d_1 - 1, d_2) + 2d_2$ and $S(d_2 - 1, d_2) = 2d_2 - 1$. Therefore $S(d_1, d_2) \leq 2\, d_1\, d_2$. □

F Conditional hardness of $\sqrt{1+4x}$

We first show that $\sqrt{1+4x} \in \mathbb{Z}[[x]]$.

Lemma F.1 (Folklore). *We have* $\sqrt{1+4x} = \sum_{i\geq 0} \binom{2i}{i}/(2i-1)\, x^i \in \mathbb{Z}[[x]]$.

Proof. We know that, $\sqrt{1+4x} = \sum_{i\geq 0} \binom{1/2}{i}(4x)^i$. Now, it is easy to see that:

$$\binom{\frac{1}{2}}{d} = \frac{\frac{1}{2}\cdot\left(\frac{1}{2}-1\right)\cdot\left(\frac{1}{2}-2\right)\cdot\cdots\cdot\left(\frac{1}{2}-d+1\right)}{d!} = (-1)^{d-1}\cdot\frac{\binom{2d}{d}}{4^d(2d-1)}.$$

This implies that $\sqrt{1+4x} = \sum_{i\geq 0} \binom{2i}{i}/(2i-1)\, x^i$. Further, it is also easy to verify that

$$\binom{2d}{d} = \left(4\binom{2d-2}{d-1} - \binom{2d}{d}\right)\cdot(2d-1) \implies \binom{2d}{d}/(2d-1) \in \mathbb{N}\,.$$

Therefore, $\sqrt{1+4x} \in \mathbb{Z}[[x]]$, as desired. □

Lemma F.1 implies that all the truncations of $\sqrt{1+4x}$ can be computed by division-free circuits.

Theorem F.1. *If* $\tau(\text{trunc}\left(\sqrt{1+4x}, d\right) = O(\log^c d)$*, for some constant* $c \in \mathbb{N}$*, then* $(d!)$ *is easy. In fact,* $\tau(d!) = O(\log^{c+1} d)$*.*

Proof. By assumption, we know that $\tau(\text{trunc}\left(\sqrt{1+4x}, d-1\right) = O(\log^c d)$ and $\tau(\text{trunc}\left(\sqrt{1+4x}, d\right) = O(\log^c d)$. By using Lemma F.1, we see that:

$$\text{trunc}\left(\sqrt{1+4x}, d\right) - \text{trunc}\left(\sqrt{1+4x}, d-1\right) = (-1)^{d-1} x^d \frac{\binom{2d}{d}}{2d-1}.$$

Hence, $\tau((-1)^{d-1}x^d \cdot \binom{2d}{d}/(2d-1) = O(\log^c d)$. Therefore $\left((-1)^{n-1}\binom{2d}{d}/(2d-1)\right)$ has complexity $O(\log^c d)$ by substituting $x = 1$. This also implies that $\binom{2d}{d}$ has complexity $O(\log^c d)$. Invoking Lemma 5.3, we conclude. □

Corollary F.1. *If* $(d!)$ *has complexity* $\omega(\mathsf{poly}(\log d))$*, then* $\tau(\text{trunc}\left(\sqrt{1+4x}, d\right) = \omega(\mathsf{poly}(\log d))$*.*

G Integral power series: Details for Section 6

We will use some number-theoretic tool to show that the candidate power series is integral. So, before delving into that, we go through some preliminary tools being used.

Fact G.1 (Folklore). *Product of any* k *consecutive positive integers is divisible by* $k!$*.*

Definition G.1 (p-adic valuation)**.** Let p be a prime and $n \in \mathbb{Z}$. We denote p-adic valuation of n as $\nu_p(n)$ to be the *highest exponent* such that $p^{\nu_p(n)} \mid n$. Formally, $\nu_p : \mathbb{Z} \longrightarrow \mathbb{N}$ defined by

$$\nu_p(n) = \max\{v \in \mathbb{N} : p^v \mid n\}.$$

Note that, by definition, $\nu_p(\mathrm{rad}(n)) = 1$ if $p \mid n$, and 0 otherwise.

Theorem G.1 (Legendre's formula)**.** *For a prime p and $n \in \mathbb{N}$, $\nu_p(n) = \sum_{j=1}^{\infty} \lfloor n/p^j \rfloor$.*

Now, we prove integrality of a power series which is our candidate algebraic function for Theorem 1.3. It suffices to prove the integrality of $(1 + k^2 x)^{1/k}$, which we prove below.

Theorem G.2 (Restatement of Theorem 6.1, Integral power series)**.** *Let $k \in \mathbb{N}$. Define $f_k(x) := \left(1 + k^2 \cdot x\right)^{1/k}$. Then, $f_k(x) \in \mathbb{Z}[[x]]$.*

Proof. By binomial expansion, $f_k \in \mathbb{Q}[[x]]$. Let $f_k(x) = \sum_{d \geq 0} a_d \cdot x^d$. We'll prove by *strong induction* that indeed the coefficients are integers.

Obviously $a_0 = 1$, and assume that for $m \in \mathbb{N}$ we have proved that $a_\ell \in \mathbb{Z}$ for $0 \leq \ell < m$. The coefficient at x^m in $\left(\sum_{d=0}^{\infty} a_d x^d\right)^k = \left(1 + \sum_{d=1}^{\infty} a_d x^d\right)^k$ is equal to $k \cdot a_m$ plus a bunch of terms that we know are integer by the induction hypothesis; hence $k \cdot a_m = b \in \mathbb{Z}$. But by the binomial series formula we have

$$a_m = k^{2m} \cdot \binom{1/k}{m} = \frac{k^{2m} \cdot \prod_{j=0}^{m-1}(1/k - j)}{m!} = \frac{k^m \cdot \prod_{j=0}^{m-1}(1 - kj)}{m!}.$$

It suffices to prove that $k \mid b$. If we can show that $\nu_p(b) \geq \nu_p(k)$ for every prime p dividing k, this would certainly imply that $k \mid b$. So, fix a prime $p \mid k$. Note that

$$b = k \cdot a_m = X/m!\,, \text{ where } X := k^{m+1} \cdot \prod_{j=0}^{m-1}(1 - kj).$$

As, $p \mid k$, we must have $\prod_{j=0}^{m-1}(1 - kj) \equiv 1 \bmod p$. Thus, $\nu_p(X) = \nu_p\left(k^{m+1}\right) = (m+1)\nu_p(k)$. And by Theorem G.1, $\nu_p(m!) = \sum_{j=1}^{\infty} \lfloor m/p^j \rfloor < \sum_{j=1}^{\infty} m/p^j = m/p - 1 \leq m$. Thus,

$$\nu_p(b) = \nu_p(X) - \nu_p(m!) \geq (m+1)\nu_p(k) - m \geq \nu_p(k),$$

as we wanted. Putting it together gives $a_m \in \mathbb{Z}$ proving the inductive step. Hence, the conclusion follows.

□

H From hardness of algebraic functions to hardness of permanent in constant-free regime

Here, we sketch why one of the truncations being hard implies permanent does not have small constant-free circuits (implying $\mathsf{VP}_0 \neq \mathsf{VNP}_0$). The proof is reminiscent to [Bür09]. We point out the main components. We denote Perm_n as the permanent polynomial of a $n \times n$ symbolic matrix.

Theorem H.1 (Hardness of permanent)**.** *Let us fix $i, k \in \mathbb{N}$ such that $i < k$. Further, assume that, $L\left(\mathsf{trunc}\left((1+k^2\,x)^{i/k}\right), d\right) = \omega(\mathsf{poly}(\log d))$, then $\tau(Perm_n) = \omega(\mathsf{poly}(\log n))$.*

Remark H.1. One can also prove a conditional implication referring to the original Valiant hypothesis $\mathsf{VP}_{\mathbb{C}} \neq \mathsf{VNP}_{\mathbb{C}}$, assuming GRH (Generalized Riemann Hypothesis). This has also been pointed out in [Bür09, Corollary 4.2]. This basically follows from the fact that under GRH and assuming $\mathsf{VP} = \mathsf{VNP}$, then $\mathsf{CH} \subseteq \mathsf{P}/\mathsf{poly}$.

Before going into the proof sketch, we define CH-*definable* sequences. The *counting hierarchy* is denoted by CH [Wag86]. The class of poly-size circuits can be expressed by the nonuniform *advice class* P/poly.

Let $q(n)$ be a polynomial. Let $a = (a(n,\ell))_{n\in\mathbb{N},\, \ell\le q(n)}$ be a sequence of integers such that $a(n,\ell)$ has *exponential bitsize*, i.e., $|a(n,\ell)| \le 2^{n^c}$ for all k and some constant c. We think of n, ℓ as being represented in binary using $O(\log n)$ bits.

With the sequence, we associate a language that determines the bits of $a(n,\ell)$ in binary,

$$\begin{aligned} \mathrm{Sgn}(a) &= \{(n,\ell) \mid a(n,\ell) \ge 0\}, \\ \mathrm{Bit}(a) &= \{(n,\ell,j,b) \mid \text{ the } j\text{-th bit of } a(n,\ell) \text{ equals } b\}\,. \end{aligned}$$

Definition H.1 ([Bür09, Definition 3.2])**.** The sequence $a = (a(n,\ell))_{n,\ell}$ of integers of exponential bitsize is CH-*definable* if $\mathrm{Sgn}(a) \in \mathsf{CH}$ and $\mathrm{Bit}(a) \in \mathsf{CH}$.

The sequences of integers that are definable in CH are *closed* under iterated addition, iterated multiplication, and integer division [Bür09, Theorem 3.10]. Koiran et al. [KP11, Theorem 2.14] used the binary version of the same theorem.

Theorem H.2. [Bür09; KP11] *(i) Let $q(n)$ be a polynomial and suppose $(a(n,\ell))_{n\in\mathbb{N},\ell\le q(n)}$ is CH-definable. Then the sum- and product-sequences $b(n)$ and $c(n)$ are CH-definable, where*

$$b(n) = \sum_{\ell=0}^{q(n)} a(n,\ell) \qquad \textit{and} \qquad c(n) = \prod_{\ell=0}^{q(n)} a(n,\ell)\,.$$

(ii) Suppose $(s(n))_{n\in\mathbb{N}}$ and $(t(n))_{n\in\mathbb{N}}$ are definable in CH and $t(n) > 0$ for all n. Then the sequence of quotients $\lfloor s(n)/t(n)\rfloor_{n\in\mathbb{N}}$ is definable in CH.

Now, we state the most important theorem proven in [Bür09, Theorem 4.1] from which Theorem H.1 will follow almost trivially.

Theorem H.3. *Let q be a polynomially bounded function and $(b(n,\ell))_{n\in\mathbb{N},\ell\le q(n)}$ and $(d(n))_{n\in\mathbb{N}}$ are definable in CH. Let*

$$f_n = \sum_{\ell=0}^{q(n)} b(n,\ell)x^\ell \in \mathbb{Z}[x]\,,\ g_n = f_n/d(n) \in \mathbb{Q}[x]\,.$$

If $\tau(Perm_n) = \mathsf{poly}(\log n)$, then $L_{\mathbb{Q}}(g_n) = \mathsf{poly}(\log n)$.

Now, we are ready to prove Theorem H.1.

Proof sketch of Theorem H.1. Let, $(1+k^2\cdot x)^{i/k} := \sum_{j\ge 0} a_{i,j}\,x^j \in \mathbb{Z}[[x]]$. By binomial expansion, we have

$$a_{i,j} = k^{2j}\cdot\binom{i/k}{j} = k^j/j!\cdot\prod_{\ell=0}^{j-1}(i-k\ell)\,.$$

As k is a constant, $\prod_{\ell=0}^{j-1}(i-k\ell), j!, k^j$ are all trivially definable in CH, by Theorem H.2. Further, by Theorem G.2, $a_{i,j} \in \mathbb{Z}$ implying $(a_{i,j})$ CH-definable, again by Theorem H.2.

The rest directly follows from Theorem H.3. Note that, if $\tau(\text{Perm}_n) = \mathsf{poly}(\log n)$, then from the above argument, truncation of the power series upto n i.e. $f_n = \sum_{j=0}^{n} a_{i,j}x^j$ must be *easy*, as the coeffecients are CH-definable. This directly contradicts our assumption that the truncation is hard. Hence, permanent cannot have polynomial size constant-free circuits. $\square$

I Algorithm

Here, we write the algorithm for the first part of Theorem 1.3.

Algorithm 1 Integer factorization assuming the truncations of $(1+k^2x)^{i/k}$ being easy for each i

```
Input: A composite positive integer $n$.
Output: A non-trivial factor of $n$.
1: Define $N(d,k) := \frac{k^{(k-2)d}(dk)!}{(d!)^k}$.
2: $m \leftarrow k$.
3: while true do
4:     Compute $\gcd(N(m,k),n)$.
5:     if $\gcd(N(m,k),n) = 1$ then
6:         $m \leftarrow mk$.
7:     else if $\gcd(N(m,k),n) = n$ then
8:         $t \leftarrow m$.                              ▷ This $m$ is the desired $t$.
9:         break
10:    else
11:        return $\gcd(N(m,k),n)$        ▷ Here $\gcd(N(m,k),n)$ is a non-trivial factor of $n$.
12:    end if
13: end while
            ▷ At this step, all the primes dividing $n$ are in the interval $[t+1, tk]$.
14: $a \leftarrow 1$.
15: $b \leftarrow t$.
16: while true do
17:     if $b - a \leq 1$ then
18:         $s \leftarrow a$.                              ▷ This $a$ is the desired $s$.
19:         break
20:     end if
21:     $c \leftarrow \lceil (a+b)/2 \rceil$.
22:     Compute $\gcd(N(c,k),n)$.
23:     if $\gcd(N(c,k),n) = 1$ then
24:         $a \leftarrow c$.
25:     else if $\gcd(N(c,k),n) = n$ then
26:         $b \leftarrow c$.
27:     else
28:         return $\gcd(N(c,k),n)$        ▷ Here $\gcd(N(c,k),n)$ is a non-trivial factor of $n$.
29:     end if
30: end while
            ▷ At this step, all the primes dividing $n$ are in the interval $[sk+1, (s+1)k]$.
31: for $i = sk+1$ to $(s+1)k$ do
32:     if $i$ divides $n$ then
33:         return $i$                        ▷ Here $i$ is a non-trivial factor of $n$.
34:     end if
35: end for
```