Discovering the roots: Uniform closure results for algebraic classes under factoring *

PRANJAL DUTTA, Chennai Mathematical Institute (& IIT Kanpur)
NITIN SAXENA, Indian Institute of Technology, Kanpur
AMIT SINHABABU, Aalen University, Germany

Newton iteration (NI) is an almost 350 years old recursive formula that approximates a simple root of a polynomial quite rapidly. We generalize it to a matrix recurrence (allRootsNI) that approximates *all* the roots simultaneously. In this form, the process yields a better circuit complexity in the case when the number of roots r is small but, the multiplicities are exponentially large. Our method sets up a linear system in r unknowns and iteratively builds the roots as formal power series. For an algebraic circuit $f(x_1, \ldots, x_n)$ of size s, we prove that *each* factor has size at most a polynomial in: s and the degree of the squarefree part of f. Consequently, if f_1 is a $2^{\Omega(n)}$-hard polynomial then any nonzero multiple $\prod_i f_i^{e_i}$ is equally hard for *arbitrary* positive e_i's, assuming that $\sum_i \deg(f_i)$ is at most $2^{O(n)}$.

It is an old open question whether the class of poly(n)-sized formulas (respectively algebraic branching programs) is closed under factoring. We show that given a polynomial f of degree $n^{O(1)}$ and formula (respectively ABP) size $n^{O(\log n)}$ we can find a similar size formula (respectively ABP) factor in randomized poly($n^{\log n}$)-time. Consequently, if determinant requires $n^{\Omega(\log n)}$ size formula, then the same can be said about any of its nonzero multiples.

In all our proofs, we exploit the following property of multivariate polynomial factorization. Under a random linear transformation τ, the polynomial $f(\tau\overline{x})$ *completely* factors via power series roots. Moreover, the factorization adapts well to circuit complexity analysis. This with allRootsNI are the techniques that help us make progress towards the old open problems; supplementing the vast body of classical results and concepts in algebraic circuit factorization (eg. Zassenhaus, J.NT 1969; Kaltofen, STOC 1985-7 & Bürgisser, FOCS 2001).

CCS Concepts: • **Mathematics of computing** → **Combinatoric problems**; • **Theory of computation** → **Problems, reductions and completeness**; **Algebraic complexity theory**; • **Computing methodologies** → **Algebraic algorithms**; **Hybrid symbolic-numeric methods**.

Additional Key Words and Phrases: circuit factoring, formula, ABP, randomized, hard, VF, VBP, VP, VNP, quasipoly

ACM Reference Format:
Pranjal Dutta, Nitin Saxena, and Amit Sinhababu. 2022. Discovering the roots: Uniform closure results for algebraic classes under factoring . 1, 1 (January 2022), 37 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*"A preliminary version was presented in STOC 2018.

Authors' addresses: Pranjal Dutta, pranjal@cmi.ac.in, Chennai Mathematical Institute (& IIT Kanpur); Nitin Saxena, nitin@cse.iitk.ac.in, Indian Institute of Technology, Kanpur; Amit Sinhababu, amitks@cse.iitk.ac.in, Aalen University, Germany.

1 INTRODUCTION

Algebraic circuits provide a way, alternate to Turing machines, to study computation. Here, the complexity classes contain (multivariate) polynomial families instead of languages. It is a natural question whether an algebraic complexity class is closed under factors. This is also a useful, and hence, a very well-studied problem both from the point of view of practice and theory. We study the following two questions related to multivariate polynomial factorization:

(1) *Closure properties*: Let $\{f_n(x_1, \ldots, x_n)\}_n$ be a polynomial family in an algebraic complexity class C (egs. VP, VF, VBP, VNP or $\overline{\text{VP}}$ etc.). Let g_n be an arbitrary factor of f_n. Can we say that $\{g_n\}_n \in C$? Equivalently, is the class C *closed under factoring*?

(2) *Uniformity*: Can we design an *efficient*, i.e. randomized poly(n)-time, algorithm to output the factor g_n with a representation in C?

Different classes give rise to new challenges for the closure questions. Before discussing further, we give a brief overview of the algebraic complexity classes relevant for our paper. For more details, see [Mah14, SY10, BCS13].

Algebraic circuits are a natural model to represent polynomials compactly. An *algebraic circuit* has the structure of a directed acyclic graph. It has leaf nodes labelled as input variables $x_1, \ldots, x_n$ and constants from the underlying field $\mathbb{F}$. All the other nodes are labelled as addition and multiplication gates. It has a root node that outputs the polynomial computed by the circuit. Some of the complexity parameters of a circuit are *size* (number of edges and nodes), *depth* (the length of the longest path in the circuit), formal *degree* (the maximum degree polynomial computed by any node), *fan-in* (maximum number of inputs to a node) and *fan-out*. An *algebraic formula* is a circuit whose underlying graph is a *directed tree*. In a formula, the fan-out of the nodes is at most one, i.e. 'reuse' of intermediate computation is not allowed.

The class VP (respectively VF) contains the families of n-variate polynomials of degree $n^{O(1)}$ over $\mathbb{F}$, computed by $n^{O(1)}$-sized circuits (respectively formulas). The class VF is sometimes denoted as VP_e, for it collects 'expressions' which is another name for formulas. Similarly, one can define VQP (respectively VQF) which contains the families of n-variate polynomials of degree $n^{O(1)}$ over $\mathbb{F}$, computed by $2^{\text{poly}(\log n)}$-sized circuits (respectively formulas). If we relax the condition on the degree in the definition of VP, by allowing the degree to be possibly exponential, then we define the class VP_{nb}.

Algebraic branching program (ABP) is another model for computing polynomials, which we define in Section 2.1. The class VBP contains the families of polynomials computed by $n^{O(1)}$-sized ABPs. We have the easy containments: VF $\subseteq$ VBP $\subseteq$ VP $\subseteq$ VQP = VQF; for details we refer [BOC92, VSBR83].

Finally, we give an overview of the class VNP, which can be seen as a non-deterministic analog of the class VP. A family of polynomials $\{f_n\}_n$ over $\mathbb{F}$ is in VNP if there exist polynomials $t(n), s(n)$ and a family $\{g_n\}_n$ in VP such that for every n, $f_n(\overline{x}) = \sum_{w \in \{0,1\}^{t(n)}} g_n(\overline{x}, w_1, \ldots, w_{t(n)})$. Here, the *witness* size is $t(n)$ and the *verifier* circuit g_n has size $s(n)$. VP is contained in VNP and it is believed that this containment is strict (Valiant's Hypothesis [Val79]).

Now, we briefly discuss the state of the art on the closure questions for various algebraic complexity classes. To cover more depth and breadth, see [Kal90, Kal92, FS15].

1.1 Previously known closure results

Famously, Kaltofen [Kal85a, Kal86, Kal87, Kal89] showed that VP is *uniformly* closed under factoring, i.e. for a given d degree n variate polynomial f of circuit size s, there exists a randomized poly(snd)-time algorithm that outputs its factor as a circuit whose size is bounded by poly(snd). This fundamental result has several applications such as 'hardness versus randomness' in algebraic complexity [KI03, AV08, DSY09, AGS19, CKS19b, KST19, DST21, Dut21], derandomization of Noether Normalization Lemma [Mul17], in the problem of circuit reconstruction [KS09, Sin16], and polynomial equivalence testing [Kay11]. In general, multivariate polynomial factoring has several applications, including decoding of Reed-Solomon, Reed-Muller codes [GS98, Sud97], integer factoring [LLMP90], primary decomposition of polynomial ideals [GTZ88] and algebra isomorphism [KS06, IKRS12]. Typically algorithms for multivariate polynomial factorization use univariate factorization as a subroutine. For factoring univariate polynomials over rationals, Lenstra, Lenstra and Lovasz [LLL82] gave a deterministic polynomial time algorithm. For factoring univariate polynomials over finite fields, there are randomized polynomial time algorithms due to Berlekamp [Ber70], Cantor and Zassenhaus [CZ81]. Even over other fields (complex, p-adics, algebraic number fields) and rings (Galois rings etc), several results on univariate factoring are known [Sch82, Chi94, CG00, Lan85, Len83, vZGH96, DMS19].

It is natural to ask whether Kaltofen's VP factoring result can be extended to VP_{nb} which allows the degree of the polynomials to be exponentially high [1]. It is known that *not every* factor of a high degree polynomial has a small sized circuit. For example, the polynomial $x^{2^s} - 1$ can be computed in size s, but it has factors (of high degree) over $\mathbb{C}$ that require circuit size $\Omega\left(2^{s/2}/\sqrt{s}\right)$ [LS78, Sch77]. It is conjectured [Bür13, Conj.8.3] that *low* degree factors of high degree small-sized circuits have *small* circuits. Formally, any factor g of a polynomial f computed by an arithmetic circuit of size s, can be computed by an arithmetic circuit of size $\mathrm{poly}(s, d_g)$, where d_g is the degree of factor g. Kaltofen asked this as an open question [Kal86, Kal87] and Bürgisser posed this as "Factor Conjecture" [Bür04]. The Factor Conjecture has several interesting consequences in algebraic complexity. For more exposure, see [Bür04], [Bür13, Remark 8.15] and the recent survey [Gro20].

Partial results towards the Factor Conjecture are known. Kaltofen [Kal87] showed the following result. If a polynomial f, given by a circuit of size s factors as $g^e h$, where g and h are coprime, then g can be computed by a circuit of size $\mathrm{poly}(e, \deg(g), s)$. The question left open is to remove the dependency on e. In the special case where $f = g^e$, it was proved in [Kal87] that g has circuit size $\mathrm{poly}(\deg(g), \mathrm{size}(f))$. Kaltofen also observed that if $f = g^e h$ and degree of h is $\mathrm{poly}(s)$, then g can be computed by a small circuit.

In the high degree regime, we do not expect uniform closure results, as several algorithmic problems related to factoring are NP-hard there, eg. computing the degree of the squarefree part, gcd, or lcm. Even in the special case of *supersparse* (or lacunary) univariate polynomials (represented as a list of nonzero coefficients and the binary encoding of exponents of corresponding terms), the above mentioned problems are NP-hard [Pla77]. Nevertheless, efficient algorithms are known for computing *bounded* degree factors of supersparse polynomials ([Gre16] and references therein).

Now, we discuss the closure results for classes more restrictive than VP (such as VF, VBP, etc.). Unfortunately, Kaltofen's technique [Kal89] for VF will give a superpolynomial-sized factor formula; as it heavily *reuses* intermediate computations in the steps of Hensel lifting, linear system solving, and gcd computation. The same holds for the class VBP. In contrast, extending the idea of [DSY09], Oliveira [Oli16] showed that an n-variate polynomial with *bounded individual degree* and computed by a formula of size s, has factors of formula size $\mathrm{poly}(n, s)$. Furthermore, it was established [Oli16] that for a given n-variate individual-degree-r polynomial, computed by a circuit (respectively formula) of size s and depth Δ, there exists a $\mathrm{poly}(n^r, s)$-time randomized algorithm that outputs any factor of f computed by a circuit (respectively formula) of depth $\Delta + 5$ and size $\mathrm{poly}(n^r, s)$. A special case of factoring algebraic branching programs was

[1] Throughout the article, we will refer to *high degree* circuit which basically means that the degree of the polynomial computed by the circuit is exponential wrt. size.

considered in [KK08]—it dealt with the elimination of a *single* division gate from skew circuits (also see Section 2.1 & Lemma 5), and another special case was solved in [Jan11]. [2]

Going beyond VP we can ask about the closure of VNP. Bürgisser conjectured [Bür13, Conj.2.1] that VNP is closed under factoring. Kaltofen's technique [Kal89] for factoring VP circuits does not yield the closure of VNP. [3]

Looking at the Border. Recently, *approximative* algebraic complexity classes like $\overline{\text{VP}}$ [GMQ16] have become objects of interest, especially in the context of the geometric complexity program [Mul12a, Mul12b, Gro15]. A polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ has approximative (or border) complexity $\leq s$ if there is an infinite sequence of circuits (using arbitrary elements from the field $\mathbb{F}(\epsilon)$ as scalars) of size $\leq s$ computing $f_n \in \mathbb{F}(\epsilon)[x_1, \ldots, x_n]$, where ϵ is a new variable such that $\lim_{\epsilon \to 0} f_n = f$. For example, if a polynomial $f(\overline{x}, \epsilon) = \epsilon^d g(\overline{x}) + \epsilon^{d+1} h(\overline{x}, \epsilon)$ can be computed by an arithmetic circuit of size s over $\mathbb{F}(\epsilon)$, then g has an *approximative* circuit of size s. Thus, *approximative complexity* (size) of g must be $\leq s$.

The class $\overline{\text{VP}}$ (the closure of VP) contains families of polynomials of degree poly(n) that can be approximated (infinitesimally closely) by poly(n)-sized circuits. Bürgisser [Bür04, Bür01] discusses approximative complexity of factors, proving that low degree factors of high degree circuits have small approximative complexity. Thus, the Factor Conjecture is true in the setting of approximative complexity. Also, $\overline{\text{VP}}$ is closed under factoring [Bür04, Theorem 4.1]. For the standard algebraic complexity classes, we can ask whether their approximative closure is closed under factors.

We conclude by stating a few reasons why closure results under factoring are interesting and non-trivial. First, some classes are *not* closed under factors. For example, the family of sparse polynomials (the number of monomials with nonzero coefficients in f_n is bounded by $n^{O(1)}$); this is because a factor's sparsity may blowup super-polynomially [vzGK85]. However, the recent breakthrough work of Bhargava, Saraf and Volkovich [BSV20] gave $s^{\text{poly}(d) \cdot \log n}$ sparsity upper bound for factors of n-variate polynomials of *individual* degree d and sparsity s. In particular, for constant individual degree sparse polynomials, the factors can have at most *quasi*polynomial blowup in sparsity.

Closure under factoring indicates the robustness of an algebraic complexity class, as it proves that all nonzero multiples of a *hard* polynomial remain hard. For this reason, closure results are useful for proving lower bounds on the power of some algebraic proof systems [FSTW16].

Finally, factoring of arithmetic circuits is crucially used in many hardness versus randomness results in algebraic complexity. Kabanets and Impagliazzo showed that the famous problem of black-box derandomization of polynomial identity testing (PIT) for the class VP can be solved if we can prove strong enough arithmetic circuit lower bounds (get an explicit hard polynomial family); for details see [KI03, Theorem 4.1].

Suppose a polynomial $f(\overline{y})$ is such that for a nonzero size-s circuit C, $C(\overline{y}, f(\overline{y})) = 0$. Using factoring results for low degree C, one deduces that f also has circuit size poly(s). This gives us the connection: *If we picked a "hard" polynomial family $\{f_n\}_n$ then $(\overline{y}, f_n(\overline{y}))$ would be a hitting-set generator (hsg) for the circuit family $\{C_n\}_n$* [KI03, Theorem 7.7].

1.2 Our results

Before stating the results, we describe some of the assumptions and notations used throughout the paper. Set $[n]$ refers to $\{1, 2, \ldots, n\}$ while $[a, b]$ for $a < b$ and $a, b \in \mathbb{N}$ means for all integers $a \leq i \leq b$. Logarithms are wrt base 2. For a polynomial f, size(f) refers to the smallest size of circuits computing f; it is the *algebraic circuit complexity* of f.

Field. We denote the underlying field as $\mathbb{F}$ and assume that it is of characteristic 0 and algebraically closed. For eg. complex $\mathbb{C}$, algebraic numbers $\overline{\mathbb{Q}}$ or algebraic p-adics $\overline{\mathbb{Q}}_p$. All the results partially hold for other fields (such as

[2] Recently, in [ST20] Sinhababu and Thierauf proved that VBP is closed under factors.
[3] Recently, in [CKS19b] Chou,Kumar and Solomon confirmed that VNP is indeed closed under factors.

$\mathbb{R}, \mathbb{Q}, \mathbb{Q}_p$ or finite fields of characteristic > degree of the input polynomial). For a brief discussion on this issue, see Section 6.

Ideal. We denote the variables $(x_1, \ldots, x_n)$ as $\overline{x}$. The *ideal* $I := \langle \overline{x} \rangle$ of the polynomial ring will be of special interest, and its power ideal I^d, whose generators are all degree d monomials in n variables. Often we will reduce the polynomial ring modulo I^d (inspired from *Taylor series of an analytic function around* $\overline{0}$ [Tay15]).

Radical. For a polynomial $f = \prod_i f_i^{e_i}$, with f_i's being coprime irreducible nonconstant factors of multiplicity $e_i > 0$, the squarefree part $\prod_i f_i$ is called as the *radical* of f, denoted as $\mathrm{rad}(f)$. The radical of a polynomial is unique (up to scalar).

What can we say about the size of the radical of f, if f has a circuit of size s? We prove that the size of the radical is $\mathrm{poly}(s, d_r)$, where d_r is the degree of the radical. As a direct corollary, we get that every factor of a given circuit C has size bounded by a polynomial in $\mathrm{size}(C)$ and the degree of the radical of C. Thus, our main result gives a good circuit size bound for factors when $\mathrm{rad}(f)$ has small degree.

A more general formulation is:

Theorem 1. *If $f = u_0 u_1$ is a nonzero product in the polynomial ring $\mathbb{F}[\overline{x}]$, with $size(f) + size(u_0) \leq s$, then every factor of u_1 has a circuit of size $poly(s + deg(rad(u_1)))$.*

Note that Kaltofen's proof technique in [Kal89] does not extend to the *exponential* degree regime (even when the degree of $\mathrm{rad}(f)$ is small) because it requires solving equations with $\deg_{x_i}(f)$ many unknowns for some x_i, where $\deg_{x_i}(f)$ denotes *individual degree* of x_i in f, which can be very high. We do not know how to extend the proof technique in Kaltofen's single factor Hensel lifting paper [Kal87, Theorem 2] that works for the perfect-power case of $f = g^e$. It can be seen that $\mathrm{rad}(f)$ equals $f/\gcd(f, \partial_{x_i}(f))$, but the gcd itself can be of exponential-degree and so one cannot hope to use [Kal87, Theorem 4] to compute the gcd either. It is an open question whether the gcd of two polynomials (computed by small circuits of high degree) can be computed by a small circuit [Kal87].

Remarks. (1) Interestingly, our result when combined with [Kal87, Theorem 3] implies that every factor g of f has a circuit of size polynomial in: $\mathrm{size}(f)$, $\deg(g)$ and $\min\{\deg(\mathrm{rad}(f)), \mathrm{size}(\mathrm{rad}(f))\}$. We leave it as an open question whether the latter expression is polynomially related to $\mathrm{size}(f)$.

(2) Theorem 1 shows an interesting way to create *hard* polynomials. In the theorem statement let the size concluded be $(s + \deg(\mathrm{rad}(u_1)))^e$, for some constant e. If one has a polynomial $f_1(x_1, \ldots, x_n)$ that is 2^{cn}-hard, then any nonzero $f := \prod_i f_i^{e_i}$ is also $2^{\Omega(n)}$-hard for *arbitrary* positive e_i's, as long as $\sum_i \deg(f_i) \leq 2^{\frac{cn}{e} - 1}$.

1.2.1 A detour into numerical analysis (via arithmetic circuits). Root approximation of univariate polynomials has been an interesting problem in mathematics & engineering. Interestingly, it has also found applications in various other problems such as computing the largest eigenvalue, and checking whether a matrix is approximately PSD (positive semi-definite) [LV16].

We can quantify the same question about 'approximating roots' of a univariate polynomial and analyze the bit-complexity of the root; where the measure is the *bitsize* of the best circuit. For an $f(x) \in \mathbb{R}[x]$ we define $\mathrm{bitsize}(f) = s$ if we can compute $f(x)$ by a circuit using $\{+, -, \times, \div\}$ gates and of overall bitsize s. Bit-size of a constant c is the number of bits (binary) required to represent c (which is $\log_2(c)$). Note that, this is in contrast with the usual VP notion where constants are free. Here, the degree of f can be exponential (eg. 2^s) and the coefficients may have exponential bit-length (eg. 2^s bits, or values 2^{-2^s} to 2^{2^s}).

For this complexity notion, we can show a surprising fact for the roots.

Claim 1. For each root $a \in (0,1)$ of $f(x)$, there is some 2^m-bit approximation a' such that $\text{bitsize}(a') \leq O((s+m)\cdot\log(\frac{1}{\epsilon}))$, where $\text{bitsize}(f) =: s$, and $\epsilon \in (0,1)$ lower bounds the gap between a and the other roots of $f(x)$.

A proof is sketched in the appendix (Section A).

Note that the estimate is nontrivial, as it is essentially expressing 2^m bits of the root using 'only' bitsize m; so, roots of small circuits are rather special, in contrast to generic strings that are incompressible [VL97]. It is relevant here to recall Shub-Smale's tau conjecture that states that 'small' circuits have 'few' *integral* roots! A proof of tau conjecture would imply P≠NP over $\mathbb{C}$ [BCSS98]. This motivates well the study of the roots of circuits.

Our Theorem 1 is an algebraic analog of the above; there $m\log(\frac{1}{\epsilon})$ gets replaced by the degree of the radical (or, the number of roots). Also, the algebraic result is better (than Theorem 1) in the sense that we do not need ÷ gates.

1.2.2 Back to multivariate algebraic models. In general, for a high degree circuit f, $\text{rad}(f)$ can be of high degree (exponential in the size of the circuit). Ideally, we would like to show that every degree d factor of f has $\text{poly}(\text{size}(f), d)$-size circuit. The next theorem reduces the above question to a special kind of modular division, where the denominator polynomial may *not* be invertible but the quotient is well-defined (eg. $x^2/x \mod x$). All that remains is to eliminate somehow this kind of *non-unit division* operator (which we leave as an open question). Consider *random* [4] elements $\alpha_i, \beta_i \in_r \mathbb{F}$ and the corresponding *random linear map* $\tau : x_i \mapsto \alpha_i y + x_i + \beta_i$, $i \in [n]$, where y is a new variable apart from $x_1, \ldots, x_n$. In the following theorem, the notation $A/B \mod \langle \overline{x} \rangle^{d+1}$ means we are truncating the power series $C = A/B$ upto total degree d (the lower part).

Theorem 2. *If nonzero $f \in \mathbb{F}[\overline{x}]$ can be computed by a circuit of size s, then any degree d factor of $f(\tau\overline{x})$ is of the form $A/B \mod \langle \overline{x} \rangle^{d+1}$ where polynomials A, B have circuits of size poly(sd).*

Note that in Theorem 2, B may be non-invertible in $\mathbb{F}[\overline{x}]/\langle \overline{x} \rangle^{d+1}$ and may have a high degree (eg. 2^s). So, we cannot use the famous trick of Strassen to do division elimination here [Str73].

We prove uniform closure results, under factoring, for the algebraic complexity classes defined below. Let $s : \mathbb{N} \to \mathbb{N}$ be a function. Define the class VF(s) to contain families $\{f_n\}_n$ such that n-variate f_n can be computed by an algebraic formula of size $\text{poly}(s(n))$ and has degree $\text{poly}(n)$. Similarly, VBP(s) contains families $\{f_n\}_n$ such that f_n can be computed by an ABP of size $\text{poly}(s(n))$ and has degree $\text{poly}(n)$. Finally, VNP(s) denotes the class of families $\{f_n\}_n$ such that f_n has witness size $\text{poly}(s(n))$, verifier circuit size $\text{poly}(s(n))$, and has degree $\text{poly}(n)$.

Theorem 3. *The classes VF$(n^{\log n})$, VBP$(n^{\log n})$, VNP$(n^{\log n})$ are all closed under factoring.*

Moreover, there exists a randomized poly$(n^{\log n})$-time algorithm that: for a given $n^{O(\log n)}$ sized formula (respectively ABP) f of poly(n)-degree, outputs $n^{O(\log n)}$ sized formula (respectively ABP) of a nontrivial factor of f (if one exists).

Remark. The "time-complexity" in the algorithmic part makes sense only in certain cases. For example, when $\mathbb{F} \in \{\mathbb{Q}, \mathbb{Q}_p, \mathbb{F}_q\}$, or when one allows computation in the BSS-model [BSS89]. In the former case, our algorithm takes $\text{poly}(n^{\log n})$ bit operations (assuming that the characteristic is zero or larger than the degree; see Theorem 26 in Section 6.2).

It is important to note that Theorem 3 does not follow by invoking Kaltofen's circuit factoring [Kal89] and VSBR transformation [VSBR83] from circuit to log-depth formula. Formally, if we are given a formula (respectively ABP) of

[4] By a random choice $\alpha \in_r \mathbb{F}$ we will mean that choose randomly from a fixed finite set $S \subseteq \mathbb{F}$ of appropriate size. This will be in the spirit of the *Polynomial Identity Lemma* (Lemma 8).

size $n^{O(\log n)}$ and degree poly(n), then it has factors which can be computed by a circuit of size $n^{O(\log n)}$ and depth $O(\log n)$. If one converts the factor circuit to a formula (respectively ABP), one will get the size upper bound of the factor formula to be a much larger $(n^{O(\log n)})^{\log n} = n^{O(\log^2 n)}$. Moreover, Kaltofen's methods crucially rely on the circuit representation to do linear algebra, division with remainder, and Euclid gcd efficiently; an excellent overview of the implementation level details to keep in mind is [KSS15, Section 3].

Our proof methods extend to the approximative versions $C(n^{\log n})$ for $C \in \{\overline{\mathrm{VF}}, \overline{\mathrm{VBP}}, \overline{\mathrm{VNP}}\}$ as well (Theorem 25).

As before, Theorem 3 has an interesting lower bound consequence: If f has VF (respectively VBP respectively VNP) complexity $n^{\omega(\log n)}$, then any nonzero fg has similar hardness (for $\deg(g) \leq \mathrm{poly}(n)$). In fact, the method of Theorem 3 yields a formula factor of size $s^e d^{2\log d}$ for a given degree-d size-s formula (e is a constant). This means— If determinant $\det_n$ requires $n^{a \log n}$ size formula, for $a > 2$, then *any* nonzero degree-$O(n)$ multiple of $\det_n$ requires $n^{\Omega(\log n)}$ size formula.

1.3 Polynomial factoring and power series roots

All our results use a reduction of polynomial factoring to approximating the power series roots of a polynomial. Here, first, we try to sketch why approximating the power series roots suffice to compute the factors. Subsequently, we sketch a new method called *allRootsNI* which approximates the roots *simultaneously*.

Power series complete split: We are interested in the *complete* factorization pattern of a polynomial $f(x_1, \ldots, x_n)$. We can view f as a univariate polynomial in one variable, say x_n, with coefficients coming from $\mathbb{F}[x_1, \ldots, x_{n-1}]$. It is easy to connect linear factors with the roots: $x_n - g$ is a factor of f iff $f(x_1, \ldots, x_{n-1}, g(x_1, \ldots, x_{n-1})) = 0$.

Of course, one should not expect that a polynomial always has a linear factor in one variable. But, if one works with an algebraically closed field, then a univariate polynomial completely splits into linear factors (also see the *fundamental theorem of algebra* [CRS96, §2.5.4]). So, if we go to the algebraic closure of $\mathbb{F}(x_1, \ldots, x_{n-1})$, any multivariate polynomial which is monic in x_n will split into factors all linear in x_n. A representation of the elements of $\overline{\mathbb{F}(x_1, \ldots, x_{n-1})}$ as a finite circuit is impossible (eg. $\sqrt{x_1}$). On the other hand, *all* the roots (wrt a new variable y) are actually elements from the power series ring $\mathbb{F}[[x_1, \ldots, x_n]]$, after a *random* linear transformation on the variables, $\tau : \overline{x} \mapsto \overline{x} + \overline{\alpha} y + \overline{\beta}$, is applied (Theorem 17). This is a direct consequence of the classical idea of Newton iteration in the formal power series setting [BCS13, Theorem 2.31].

We try to explain the above idea using the following example. Consider $f = (y^2 - x^3) \in \mathbb{F}[x, y]$. Does it have a factor of the form $y - g$ where $g \in \mathbb{F}[[x]]$? The answer is clearly 'no' as $x^{3/2}$ does not have any power series representation in $\mathbb{F}[[x]]$. But, what if we shift x randomly? For example, if we use the shift $y \mapsto y, x \mapsto x + 1$. Then, by Taylor series around 1, we see that $(x+1)^{3/2}$ has a power series expansion, namely $1 + \frac{3}{2}x + \frac{3/2 \times 1/2}{2!}x^2 + \ldots$.

Formally, Theorem 17 shows that under a random $\tau : \overline{x} \mapsto \overline{x} + \overline{\alpha} y + \overline{\beta}$ where $\overline{\alpha}, \overline{\beta} \in_r \mathbb{F}^n$, polynomial f can be factored as $f(\tau\overline{x}) = \prod_{i=1}^{d_0} (y - g_i)^{\gamma_i}$, where $g_i \in \mathbb{F}[[\overline{x}]]$ with the constant terms $g_i(\overline{0})$ being distinct, $d_0 := \deg(\mathrm{rad}(f))$ and $\gamma_i > 0$.

Computing factors of a polynomial can be *reduced* to approximating power series roots. Theorem 17 implies that any factor g of degree d can be completely split as $\prod_{i=1}^{d} (y - g_i)$, where g_i is a power series (which are also roots of f). If we know the approximations of g_i, up to degree d (notationally; $g_i^{\leq d} := g_i \bmod \langle x_1, \ldots, x_n \rangle^{d+1}$), we can compute g exactly. In particular, suppose $G = \prod_{i=1}^{d} (y - g_i^{\leq d})$. Observe that the factor $g = G^{\leq d}$, and thus, we have computed g accurately. For the details, see Section 3.1.

The power series roots can be approximated using the classical Newton iteration method [BCS13, Theorem 2.31], which works when the root we want to compute has multiplicity one. If a factor has multiplicity $e \geq 2$, then all its roots would have multiplicity e. If e is poly(s), then we can take $(e-1)$-th derivative of the polynomial to be factored. In this derivative polynomial (which can be computed by a small circuit), the roots we wanted have multiplicity 1 (and thus one can use the *classical Newton Iteration*, see Lemma 15). If e is exponential in s, computing circuits of derivatives of order e may lead to an exponential blow-up of size [Kal87]. In that case, we devise a new method that approximates all the roots *simultaneously*. If the number of roots is poly(s) (equivalently, the degree of the radical is poly(s)), then our method shows that factors have small circuits. We briefly sketch the overall idea in the next paragraph.

Recursive root finding via matrices (allRootsNI): We *simultaneously* find the approximations of all the power series roots g_i of $f(\tau\overline{x})$. At each recursive step, we get a better precision wrt degree. We show that knowing approximations $g_i^{<\delta}$, of g_i up to degree $\delta-1$, is enough to (simultaneously for all i) calculate approximations of $g_i^{\leq\delta}$, up to degree δ of g_i. This new technique, of finding approximations of the power series roots, is at the core of Theorem 1.

Define, $\tilde{f}(\overline{x},y) := f(\tau\overline{x}) = \prod_i (y-g_i)^{\gamma_i}$. Applying the derivative operator ∂_y on $\tilde{f}$, we get a classic identity (which we call *logarithmic derivative identity* [5]): $(\partial_y \tilde{f})/\tilde{f} = \sum_i \gamma_i/(y-g_i)$. Reduce the above identity modulo $I^{\delta+1}$ and define $\mu_i := g_i(\overline{0}) \equiv g_i \mod I$. This gives us (see Claim 2):

$$\frac{\partial_y \tilde{f}}{\tilde{f}} = \sum_{i=1}^{d_0} \frac{\gamma_i}{y-g_i} \equiv \sum_{i=1}^{d_0} \frac{\gamma_i}{y-g_i^{<\delta}} + \sum_{i=1}^{d_0} \frac{\gamma_i \cdot g_i^{=\delta}}{(y-\mu_i)^2} \mod I^{\delta+1}.$$

In terms of the d_0 unknowns $g_i^{=\delta}$, the above is a linear equation. (Note- We treat γ_i, μ_i's as known.) As y is a free variable above, we can fix it to d_0 "random" elements c_i in $\mathbb{F}$, $i \in [d_0]$. One would expect these fixings to give a linear system with a unique solution for the unknowns. We can express the system of linear equations succinctly in the following matrix representation: $M \cdot v_\delta = W_\delta \mod I^{\delta+1}$. Here M is a $d_0 \times d_0$ matrix; each entry is denoted by $M(i,j) := \frac{\gamma_i}{(c_i-\mu_j)^2}$. Vector v_δ respectively W_δ is a $d_0 \times 1$ matrix where each entry is denoted by $v_\delta(i) := g_i^{=\delta}$ respectively $W_\delta(i) := \frac{\partial_y \tilde{f}}{\tilde{f}}\big|_{y=c_i} - G_{i,\delta}$, where $G_{i,\delta} := \sum_{k=1}^{d_0} \gamma_k/(c_i - g_k^{<\delta})$. We ensure that $\{c_i, \mu_i \mid i \in [d_0]\}$ are distinct, and show that the determinant of M is nonzero (Lemma 19). So, by knowing approximations up to $\delta-1$, we can recover δ-th (and thus *up to* degree δ as well) part by solving the above system as $v_\delta = M^{-1} W_\delta \mod I^{\delta+1}$.

An important point is that the random c_i's will ensure: all the reciprocals involved in the calculation above do exist mod $I^{\delta+1}$.

To see the gory details, see the proof of Theorem 1 in Section 4.1. For our results on restricted models like formulas and ABPs (Theorem 3), we use the classical Newton iteration method (Lemma 15).

Comparisons with other techniques: Most of the works on multivariate polynomial factoring use Hensel Lifting (lifting factorization modulo powers of an ideal). Hensel lifting and Newton iteration (lifting roots) are mathematically related, and various versions of them are equivalent. See [VzG84] for comparisons between these two techniques. Nevertheless, for showing closure results in different models, one viewpoint may be more useful than the other. Kaltofen's classic works mostly used the Hensel lifting viewpoint, although his bivariate factoring [Kal85b] showed either can be used.

Sasaki and Sasaki [SS93] used power series roots for multivariate factoring over different fields, but their way of approximating the roots (via a version of Hensel lifting) and reconstructing factors from roots (via different linear

[5] Very recently, this identity has found applications in other contexts, eg. constant top-fanin (& bottom-fanin) depth-4 PIT [DDS21b], and restricted de-bordering results in GCT [DDS21a]. Since, it converts the product gate to a sum gate, quite often the expressions become very useful.

combinations of powers of roots) are different from us. Our approach of using power series roots is inspired by the approach of Oliveira [Oli16]. There are significant technical differences (in handling the non-monic case and in using different versions of Newton iteration and *most* importantly, the *simultaneous* root approximation rather than one at a time), still the *core* idea of using approximate roots to prove factor size bound is same. Some of the algorithmic ideas (reconstructing a factor by computing a minimal polynomial of an approximate root using linear algebra) can be found in the classic LLL algorithm [LLL82] and the bivariate polynomial factoring algorithm of Kaltofen [Kal85b]. Later, Bürgisser [Bür04] used Newton iteration to approximate a root and then find a minimal polynomial for the root by solving a linear system (the trick of using an additional variable t, performing a substitution $x_j \mapsto tx_j$ and then seeing the polynomials in t with coefficients over $\mathbb{F}(\overline{x})$, helped to extend the bivariate case to general multivariate). Our work uses ideas from some of these prior works.

There are numerical methods (similar to Newton iteration) to simultaneously approximate all the roots of a univariate polynomial [Dur60, Ker66, Ehr67, Abe73]. However, it is well known that the Newton Iteration *fails* to approximate the roots that repeat (see [Lec02]) in the algebraic setting. The same holds for these techniques as well. In contrast, the allRootsNI technique can handle roots with high multiplicity.

Organization of the article. In the next section, we recall some basic operations for different algebraic models of computation and some necessary algebraic tools which will be useful for us. In Section 3, we discuss the classical Newton iteration formula and the usefulness of power series root approximation to factoring. We prove Theorem 1–2 in Section 4 which focuses on factor size bound of high degree polynomials computed by small sized circuits. Section 5 is devoted to proving Theorem 3, which focuses on factoring (size bound and algorithm) in the *quasi*polynomial size regime for different algebraic models. Section 6 talks about results for approximative complexity classes and special fields (when it is of low characteristic or it is not algebraically closed). Finally, we conclude with some open questions in Section 7. Appendix A is for a *detour* and can be seen as a numerical analog (and application) of Theorem 1.

2 PRELIMINARIES

This section is mainly divided into two subsections– Section 2.1 is on the basic definition and operations of different algebraic models which will be used to upper bound the size of factors computed by respective models and exactly *compute* it at times, while Section 2.2 is for congregating the useful mathematical tools required for our results. Before that, we talk about the fundamental mathematical notion that we use throughout the article, namely power series.

The *formal power series* ring is denoted as $\mathbb{F}[[x_1, \ldots, x_n]]$. The elements of this ring are multivariate formal power series, with degree as precision. So, an element f is written as $f = \sum_{i=0}^{\infty} f^{=i}$, where $f^{=i}$ is the *homogeneous part* of degree i of f. In algebra texts, it is also called the *completion* of $\mathbb{F}[x_1, \ldots, x_n]$ wrt the ideal $\langle x_1, \ldots, x_n \rangle$ (see [Kem10, Chap.13]). The *truncation* $f^{\leq d}$, i.e. homogeneous parts up to degree d, can be obtained by reducing modulo the ideal $\langle \overline{x} \rangle^{d+1}$. Here d is seen as the *precision* parameter of the respective approximation of f. We will also denote $f^{<d}$ as degree $< d$ part of f i.e. $f^{<d} := f^{\leq d-1}$. In fact, it is obvious that $f^{\leq d} = f^{<d} + f^{=d}$ for all $d \geq 0$.

The advantages of the ring $\mathbb{F}[[\overline{x}]]$ are many. They usually emerge because of the *inverse* identity:

$$(1 - x_1)^{-1} \;=\; \sum_{i \geq 0} x_1^i$$

which would not have made sense in $\mathbb{F}[\overline{x}]$ but is available now. This fact will be used in division elimination in different algebraic models (Lemma 5). For fast algorithms for various basic operations on formal power series, we refer the classical papers [BK78, KT78]. For an overview of various issues related to the implementation of multivariate power series in a computer algebra system, we refer [ABK+21].

The individual degree of f with respect to variable x_i, denoted by $\deg_{x_i}(f)$, is the largest power of x_i appearing in a monomial of f. The individual degree of f is the maximum individual degree with respect to each variables x_i. The *sparsity* of f denotes the number of nonzero monomials present in f.

2.1 A Primer on Algebraic Models

In our proofs, we will need some basic results about formulas, ABPs, and circuits. In particular, we can efficiently eliminate a division gate, extract a homogeneous part, and compute derivatives. For more exposure, see [SY10, Sap19].

A formula is a *rooted binary tree* where internal nodes are labeled + or × and leaf nodes are labeled from the set $\mathbb{F} \bigcup X$, where X is the set of indeterminates. The size is the number of nodes and edges of the tree.

Algebraic circuits correspond to *directed acyclic* graphs where each node is a source node (indegree 0) labeled from the set $\mathbb{F} \bigcup X$, or has indegree 2 (fanin 2) [6] and is labeled + or ×. A designated sink node (outdegree 0) is the output node. Each node computes a polynomial in an obvious way, and the graph computes the polynomial at the output node. The acyclicity constraint ensures that there is a linear ordering of the nodes such that each node, or instruction, only uses previously computed polynomials. The fanout or the outdegree of any intermediate node can be > 1 and thus, the *reusal* of nodes helps to compute more intricate polynomials in small-sized circuits, which might not be possible in formulas.

ABP is a *skew* circuit, i.e. each multiplication gate has fanin two with at least one of its inputs being a variable or a field constant. A completely different definition can be given via layered graphs or iterated matrix multiplication or symbolic determinant. It is well-known that they are all equivalent up to polynomial blow up [Mah14].

Definition 4 (Algebraic Branching Program). *An algebraic branching program (ABP) is a layered graph with a unique source vertex (say s) and a unique sink vertex (say t). All edges are from layer i to $i+1$ and each edge is labelled by a linear polynomial. The polynomial computed by the ABP is defined as $f = \sum_{\gamma: s \rightsquigarrow t} wt(\gamma)$, where for every path γ from s to t, the weight $wt(\gamma)$ is defined as the product of the labels over the edges forming γ.*

The size *of the ABP is defined as the total number of edges in the ABP.* Width *is the maximum number of vertices in a layer.*

Equivalently, one can define f as a product of matrices (of dimension at most the width), each one having linear polynomials as entries. For more details, see [SY10].

Determinant is in VBP and is computable by a $n^{O(\log n)}$ size formula. It is a famous result that the ABP model is the same as symbolic determinant [MV97].

Computing homogeneous components and coefficients of a polynomial

Let $f(\overline{x}, y)$ be polynomial of degree d in variables y and $\overline{x} = (x_1, \ldots, x_n)$. Let C be a formula (respectively circuit or ABP) of size s that computes a polynomial f. Write f as a polynomial in y, with coefficients from $\mathbb{F}[\overline{x}]$, such that $f(x, y) = \sum_{i=0}^{d} f_i\, y^i$. Then, the coefficients $f_i(\overline{x})$ have formulas (respectively circuits or ABPs) of size $\text{poly}(s, d)$.

First we replace every variable x_i by Tx_i to get a formula computing $P(T) = \sum_{i=0}^{d} P_i T^i$. Now we take d+1 many distinct numbers $\alpha_0, \ldots, \alpha_d$ that we substitute for T. We create $d+1$ many arithmetic formulas computing $P(\alpha_0), \ldots, P(\alpha_d)$. We get the following equation where the leftmost matrix is the Vandermonde matrix.

[6] The original circuit can have unbounded fanin. However, it is not hard to convert the same into fanin 2 with constant blowup in size.

$$\begin{bmatrix} 1 & \alpha_0 & \cdots & \alpha_0^d \\ 1 & \alpha_1 & \cdots & \alpha_1^d \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_d & \cdots & \alpha_d^d \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_d \end{bmatrix} = \begin{bmatrix} P(\alpha_0) \\ P(\alpha_1) \\ \vdots \\ P(\alpha_d) \end{bmatrix}$$

As the Vandermonde matrix is invertible, we get that each of P_i is a linear combination of $P(\alpha_0), \dots, P(\alpha_d)$ and has an arithmetic formula (respectively circuit or ABP) of size $\mathrm{poly}(s, d)$. We refer to [Sap19, Lemma 5.4].

But, for a size s circuit, the degree can be exponential in s. In that scenario, the above interpolation trick would not be efficient. However, Strassen in [Str73] argued that for *any* d, there is a *multi-output* homogeneous circuit computing all i-th degree homogeneous part for $i \le d$ of size $O(s \cdot d^2)$. Note that this doesn't involve the actual degree of the original circuit Φ. We briefly give an idea of how to construct a new circuit Ψ of the claimed functionality:

For every gate v in Φ, we define $d+1$ gates in Φ, which we denote $(v, 0), \dots, (v, d)$ in such a way that (v, i) computes the i-th degree homogeneous part computed at v-th node in Φ; polynomial computed at v in Φ, call it Φ_v and at (v, i)-th gate in Ψ, we call it $\Psi_{(v,i)}$. We construct Ψ inductively as follows. If v is an input gate (variables or constant), we can clearly define (v, i) as an input gate with the appropriate properties. If $\Phi_v = \Phi_u + \Phi_w$, define $\Psi_{(v,i)} = \Psi_{(u,i)} + \Psi_{(w,i)}$ for all i. If $\Phi_v = \Phi_u \times \Phi_w$, define $\Psi_{(v,i)} = \sum_{j=0}^{i} \Psi_{(u,j)} \times \Psi_{(w,i-j)}$. Induction implies that Ψ computes each homogeneous part of Φ, gate by gate. Every gate in Φ corresponds to at most $O((d+1)^2)$ gates in Ψ (each product gate requires $O((d+1)^2)$ additional sum gates), and so size of Ψ is $O(s \cdot d^2)$.

This does not work for formulas. We refer to [SY10, Theorem 2.2] and [Sap19, Lemma 5.2] for more details.

Basic operations on formulas, circuits and ABPs

We use the following standard results on size bounds for performing some basic operations (eg. division elimination, taking derivatives) of circuits, formulas, ABPs.

Lemma 5. *(Eliminate single division [Str73], [SY10, Theorem 2.1]) Let f and g be two degree-D polynomials, each computed by a circuit (respectively ABP respectively formula) of size-s with $g(\overline{0}) \neq 0$. Then $f/g \bmod \langle \overline{x} \rangle^{d+1}$ can be computed by $O((s+d)d^3)$ (respectively $O(sd^2 D)$ respectively $O(sd^2 D^2)$) size circuit (respectively ABP respectively formula).*

Proof. Assume wlog that $g(\overline{0}) = 1$; we can ensure this by appropriate normalization. So, we have the following power series identity in $\mathbb{F}[[\overline{x}]]$:

$$\frac{f}{g} = \frac{f}{(1-(1-g))} = f + f \cdot (1-g) + f \cdot (1-g)^2 + f \cdot (1-g)^3 + \cdots .$$

Note that this is a valid identity as $1-g$ is constant free. For all $d \ge 0$, LHS=RHS $\bmod \langle \overline{x} \rangle^{d+1}$.

If we want to compute $f/g \bmod \langle \overline{x} \rangle^{d+1}$, we can take the RHS of the above identity up to the term $f(1-g)^d$ and discard the remaining terms of degree greater than d. The degree$> d$ monomials can be truncated, using Strassen's *homogenization* trick, in the case of circuits and an *interpolation* trick in the case of formulas (which also works for ABPs and low degree circuits); see subsection 2.1 above. A careful analysis shows that the size blow-up is at most $O((s+d)d^2 \cdot d)$ (respectively $O(sd \cdot D \cdot d)$ respectively $O(sd \cdot D^2 \cdot d)$) for circuits (respectively ABP respectively formula).

It is easy to see that using the above result, we get $\mathrm{poly}(s, d)$ size circuit (respectively ABP respectively formula) for computing $f/g \bmod \langle \overline{x} \rangle^{d+1}$. ◻

Remark. Note that it may happen that $g(\overline{0}) = 0$, thus $1/g$ does not exist in $\mathbb{F}[[\overline{x}]]$, yet f/g may be a polynomial of degree d. In such a case, we need to discuss a modified *normalization* that works. We can shift the polynomials f, g by some random $\overline{\alpha} \in \mathbb{F}^n$. The constant term of the shifted polynomial is nonzero with high probability (Lemma 8). Now, we compute $f(\overline{x}+\overline{\alpha})/g(\overline{x}+\overline{\alpha})$ using the method described above. Finally, we recover the polynomial f/g by applying the reverse shift $\overline{x} \mapsto \overline{x} - \overline{\alpha}$.

What if our model has several division gates?

Lemma 6. *(Div. gates elimination [SY10, Theorem 2.12]) Let f be a polynomial computed by a circuit (respectively formula), using division gates, of size s. Then, $f \bmod \langle \overline{x} \rangle^{d+1}$ can be computed by poly(sd) size circuit (respectively formula).*

Proof idea. We pre-process the circuit (respectively formula) so that the only division gate used in the modified circuit (respectively formula) is at the top. Now to remove the single division gate at the top, we use the above power series trick.

The idea of the pre-processing is the following. We can separately keep track of numerator and denominator computed at each gate and simulate addition, multiplication and division gates in the original circuit and incrementally go from bottom to top. In particular, if at the i-th depth we have $+$ gate coming from two nodes at the $i+1$-th depth computing u_1/v_1 and u_2/v_2 respectively, then at that specific node it computes $u_1/v_1 + u_2/v_2 = (u_1 \cdot v_2 + u_2 \cdot v_1)/(v_1 \cdot v_2)$. Similarly, for $\times$ gate, it is $(u_1 \cdot u_2)/(v_1 \cdot v_2)$. Note that, given u_1, v_1 and u_2, v_2, it only incurs constantly many additional gates. Thus in a whole, this pre-processing incurs only $O(s)$ additional blow up in the case of circuits. In the case of formulas one has to ensure that in any path from the leaf to the root, there are only $O(\log sd)$ division gates. ◻

Lemma 7 (Derivative computation). *If a polynomial $f(\overline{x}, y)$ can be computed by a circuit (respectively formula respectively ABP) of size s and degree d. Then, any $\frac{\partial^k f}{\partial y^k}$ can be computed by circuit (respectively formula respectively ABP) of size poly(sk).*

Proof. The idea is to use the homogenization and interpolation properties [Sap19, Section 5.1-2].

Let $f(\overline{x}, y) = c_0 + c_1 y + c_2 y^2 + \cdots + c_\delta y^\delta$, where $c_0, c_1, \ldots, c_\delta \in \mathbb{F}[\overline{x}]$. Given the circuit (respectively formula respectively ABP) computing polynomial $f(\overline{x}, y)$, we can get the circuits (respectively formula respectively ABP) computing $c_0, \ldots, c_\delta$ using homogenization and interpolation as discussed before. Given $c_0, \ldots, c_\delta$, computing $\frac{\partial^k f}{\partial y^k}$ in size poly(sd) is trivial. We use this approach of computing derivative when the polynomial is of degree $d \le$ poly(s). ◻

Remark. In the case of high degree circuits, we cannot use the above approach. [Kal87, Theorem 1] shows that $\frac{\partial^k f}{\partial y^k}$ can be computed by a circuit of size $O(k^2 s)$, i.e. the degree of the circuit does not matter. The main idea is to use the Leibniz product rule of k-th order derivative inductively. Kaltofen [Kal87] gave evidence that it is unlikely that we can compute $\frac{\partial^k f}{\partial y^k}$ in circuit size poly$(s, \log k)$. Otherwise, the permanent polynomial would have circuits of polynomial size.

Closure properties for VNP

We now discuss some closure properties of the class VNP which will be crucially required for proving Theorem 3.

VNP-size parameter (w, v) of F refers to w being the witness size and v being the size of the verifier circuit f.

Let $F(\overline{x}, y), G(\overline{x}, y), H(\overline{x})$ have verifier polynomials f, g and h with the VNP size parameters (w_f, v_f),(w_g, v_g), (w_h, v_h) respectively. Let the degree of F wrt y be d. Then, the following closure properties can be shown ([BCS13] or [Bür13, Theorem 2.19]):

(1) Add (respectively Multiply): $F+G$ (respectively FG) has VNP-size parameter (w_f+w_g, v_f+v_g+3).
(2) Coefficient: $F_i(\overline{x})$ has VNP-size parameter $(w_f, (d+1)(v_f+1))$, where $F(\overline{x},y) =: \sum_{i=0}^{d} F_i(\overline{x})y^i$.
(3) Compose: $F(\overline{x}, H(\overline{x}))$ has VNP-size parameter $((d+1)(w_f+dw_h), (d+1)^2(v_f+v_h+1))$.

Proof. All the above statements are easy to prove using the definition of VNP.

(1)
$$
\begin{aligned}
(FG)(\overline{x},y) &= \left(\sum_{u\in\{0,1\}^{w_f}} f(\overline{x},u_1,\ldots,u_{w_f})\right)\cdot\left(\sum_{u\in\{0,1\}^{w_g}} g(\overline{x},u_1,\ldots,u_{w_g})\right)\\
&= \sum_{u\in\{0,1\}^{w_f+w_g}} A(\overline{x},u_1,\ldots,u_{w_f+w_g})
\end{aligned}
$$
where,
$$A(\overline{x},u_1,\ldots,u_{w_f+w_g}) = f(\overline{x},u_1,\ldots,u_{w_f})\cdot g(\overline{x},u_{w_f+1},\ldots,u_{w_f+w_g})$$
Trivially, A has size v_f+v_g+3 (extra: one node, two edges) and witness size is w_f+w_g. Similarly, with $F+G$.

(2) Interpolation gives, $f_i(\overline{x}) = \sum_{j=0}^{d}\alpha_j F(\overline{x},\beta_j)$, for some distinct arguments $\beta_j\in\mathbb{F}$. Clearly, $F(\overline{x},\beta_j)$ has VNP-size parameter (w_f,v_f). Using the previous addition property we get that the verifier circuit has size $(d+1)(v_f+1)$. Witness size remains w_f as we can reuse the witness string of F.

(3) Write $F(\overline{x},y) =: \sum_{i=0}^{d} F_i(\overline{x})y^i$. We know that F_i has VNP-size parameter $(w_f,(d+1)(v_f+1))$. For $0\le i\le d$, H^i has VNP-size parameter $(iw_h,(i+1)v_h)$ using i-fold product (Item 1). Substituting $y=H$ in F, we can calculate the VNP-size parameter.
Suppose F_i and H^i have corresponding verifier circuits A_i and B_i respectively. Then,
$$
\begin{aligned}
F(\overline{x},H(\overline{x})) &= \sum_{i=0}^{d} F_i(\overline{x})H^i(\overline{x})\\
&= \sum_{i=0}^{d}\left(\sum_{u\in\{0,1\}^{w_f}} A_i(\overline{x},u)\right)\cdot\left(\sum_{u\in\{0,1\}^{iw_h}} B_i(\overline{x},u)\right)
\end{aligned}
$$
Thus, the witness size is $<(d+1)(w_f+dw_h)$. The corresponding verifier circuit size is $<(d+1)^2(v_f+v_h+1)$.

□

2.2 Mathematical Toolkit

This section is dedicated for assembling the mathematical tools which we will need throughout.

In the following part, we use the DeMillo-Lipton-Schwartz-Zippel lemma, now called the *Polynomial Identity Lemma*, which basically shows that a *non*zero polynomial evaluates to *non*zero at a *random* point from a large enough field. See [CKS19b] and references therein for more details and the history of this lemma.

Lemma 8 (Polynomial Identity Lemma [Ore22, DL78, Zip79, Sch80]). *Let $p(x_1,\ldots,x_n)$ be an n-variate nonzero polynomial of total degree d. Let $S\subseteq\mathbb{F}$ be a finite set. For $\overline{\alpha}\in S^n$ picked independently and uniformly at random,*
$$\Pr[p(\overline{\alpha})=0] \;\le\; \frac{d}{|S|}\,.$$

Remark. The above lemma implies that PIT ∈ co-RP, i.e. there is an *efficient* polynomial time randomized algorithm to test whether a given polynomial is zero or not.

By a random choice $\alpha \in_r \mathbb{F}$, we always mean that we choose uniformly at random from a fixed finite set $S \subseteq \mathbb{F}$ of large enough size (say, $\geq 2d$ if d is the degree of the nonzero polynomial where we substitute $\overline{\alpha} \in S^n$).

We will use properties of $\gcd(f, g)$ and a related determinant polynomial called *resultant*.

Sylvester matrix & resultant. First, let us look at the notion of the resultant of two univariate polynomials. Let $p(x), q(x) \in \mathbb{F}[x]$ be of degree a, b respectively. From Euclid's extended algorithm, it can be shown that there exist two polynomials $u(x), v(x) \in \mathbb{F}[x]$ such that $u(x)p(x) + v(x)q(x) = \gcd(p(x), q(x))$. This is known as *Bezout's identity*. If $\gcd(p(x), q(x)) = 1$, then (u, v) with $\deg(u) < b$ and $\deg(v) < a$ is unique. Let $u(x) = u_0 + u_1 x + u_2 x^2 + \ldots + u_{b-1} x^{b-1}$ and $v(x) = v_0 + v_1 x + \ldots + v_{a-1} x^{a-1}$.

Now, if we use the equation $u(x)p(x)+v(x)q(x) = \gcd(p(x), q(x))$ and compare the coefficients of x^i, for $0 \leq i < a+b$, we get a system of linear equations in the $a + b$ many unknowns (u_i and v_i). The system of linear equations can be represented in the matrix form as $Mx = y$, where x consists of the unknowns. The resultant of f, g is defined as the determinant of the matrix M. It is easy to see that M is invertible if and only if the polynomials are coprime.

Now, the notion of resultants can be extended to multivariate, by defining the resultant of polynomials $f(\overline{x}, y)$ and $g(\overline{x}, y)$ wrt some variable y. The idea is the same as before; now we take gcd wrt the variable y and get a system of linear equations from Bezout's identity. The matrix can be explicitly written with entries being polynomial coefficients (or they could be from $\mathbb{F}[[\overline{x}]]$). This is known as the Sylvester matrix, which we define next.

Definition 9. *Let $f(\overline{x}, y) = \sum_{i=0}^{l} f_i(\overline{x}) y^i$ and $g(\overline{x}, y) = \sum_{i=0}^{m} g_i(\overline{x}) y^i$. Define* Sylvester *matrix of f and g wrt y as the following $(m + l) \times (m + l)$ matrix:*

$$Syl_y(f, g) := \begin{pmatrix} f_0 & 0 & \cdots & 0 & g_0 & 0 & \cdots & 0 \\ f_1 & f_0 & \cdots & 0 & g_1 & g_0 & \cdots & 0 \\ f_2 & f_1 & & \vdots & \vdots & g_1 & & \vdots \\ \vdots & \vdots & \ddots & f_0 & g_m & \vdots & \ddots & 0 \\ f_l & f_{l-1} & & f_1 & 0 & g_m & & g_0 \\ 0 & f_l & & f_2 & 0 & 0 & & g_1 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & f_l & 0 & 0 & \cdots & g_m \end{pmatrix}.$$

Now, the resultant can be formally defined as follows (for more details and alternate definitions, see [LN97, Chap.1]).

Definition 10. *Given two polynomials $f(\overline{x}, y)$ and $g(\overline{x}, y)$, define the resultant of f and g wrt y as the determinant of the Sylvester matrix,*

$$Res_y(f, g) \ := \ det(Syl_y(f, g)) \, .$$

From the definition, it can be seen that $\mathrm{Res}_y(f, g)$ is a polynomial in $\mathbb{F}[x]$ with degree bounded by $2\deg(f)\deg(g)$. Now, we state the following fundamental property of the Resultant, which is crucially used.

Proposition 11 (Resultant vs. Gcd). *(1) Let $f, g \in \mathbb{F}[\overline{x}, y]$ be polynomials with positive degree in y. Then, $Res_y(f, g) = 0 \iff f$ and g have a common factor in $\mathbb{F}[\overline{x}, y]$, which has positive degree in y.*

(2) There exists $u, v \in \mathbb{F}[\overline{x}, y]$ such that $uf + vg = Res_y(f, g)$.

The proof of this standard proposition can be found in many standard books on algebra including [vzGG13, Section 6]. In the following lemma, by coprimality wrt y, we mean that there is no common factor that has a monomial involving the variable y. For example, $x^2 y$ and its derivative wrt y (which is x^2) are coprime wrt y.

Lemma 12 (Squarefree-ness). *Let $f \in \mathbb{F}(\overline{x})[y]$ be a polynomial with $\deg_y(f) \geq 1$. f is squarefree iff f and $f' := \partial_y f$ are coprime wrt y.*

Proof. The main idea is to show that there does not exist $g \in \mathbb{F}(\overline{x})[y]$ with positive degree in y such that $g \mid \gcd_y(f(\overline{x},y), f'(\overline{x},y))$. This is true because– suppose g is an irreducible polynomial with positive degree in y that divides both $f(\overline{x},y)$ and $f'(\overline{x},y)$. So,

$$f(\overline{x},y) = gh \implies f'(\overline{x},y) = gh' + g'h \implies g \mid g'h\,.$$

As g is irreducible and $\deg_y(g') < \deg_y(g)$ we deduce that $g \mid h$. Hence, $g^2 \mid f$. This contradicts the hypothesis that f is squarefree. □

Now, we state another standard lemma, which is useful to us and proven using the property of Resultant.

Lemma 13 (Coprimality). *Let $f, g \in \mathbb{F}(\overline{x})[y]$ be coprime polynomials wrt y (& nontrivial in y). Then, for $\overline{\beta} \in_r \mathbb{F}^n$, $f(\overline{\beta},y)$ and $g(\overline{\beta},y)$ are coprime (& nontrivial in y).*

Proof. Consider $f = \sum_{i=1}^{d} f_i y^i$ and $g = \sum_{i=1}^{e} g_i y^i$. Choose a random $\overline{\beta} \in_r \mathbb{F}^n$. Then, by Proposition 11 & Lemma 8, $f_d \cdot g_e \cdot \mathrm{Res}_y(f,g)$ at $\overline{x} = \overline{\beta}$ is nonzero. This, in particular, implies that $\mathrm{Res}_y(f(\overline{\beta},y), g(\overline{\beta},y)) \neq 0$.
This implies, by Proposition 11, $f(\overline{\beta},y)$ and $g(\overline{\beta},y)$ are coprime. □

Lemma 14 (Transform to monic). *For a polynomial $f(\overline{x})$ of total degree $d \geq 0$ and random $\alpha_i \in_r \mathbb{F}$, the transformed polynomial $g(\overline{x},y) := f(\overline{x} + \overline{\alpha}y + \overline{\beta})$ has a nonzero constant as the coefficient of y^d, and degree wrt y is d.*

Proof. Suppose the transformation is $x_i \mapsto x_i + \alpha_i y + \beta_i$ where $i \in [n]$. Write $f = \sum_{|\overline{\beta}|=d} c_{\overline{\gamma}} \overline{x}^{\overline{\gamma}} +$ lower degree terms . The coefficient of y^d in g is $\sum_{|\overline{\gamma}|=d} c_{\overline{\gamma}} \overline{\alpha}^{\overline{\gamma}}$. Clearly, for a random $\overline{\alpha}$ this coefficient will not vanish (Lemma 8), and it is the highest degree monomial in g.
This ensures $\deg_y(g) = \deg(f) = d$ and that g is monic wrt y. □

Remark. In the above statement/proof, we do not *need* $\overline{\beta}$, in particular, $\overline{\beta} = \overline{0}$ works as well. Random $\overline{\beta}$ is required to ensure the existence of different power series roots, see the next section for the details. As we need the more general linear shift later, we use it here as well.

3 FACTORIZATION OF POLYNOMIALS OVER POWER SERIES RING

First, we present the proof of classical Newton iteration (in the setting of formal power series). This is also a formal power series version of implicit function theorem [KP12, Sec.1.3].

Lemma 15. *(Power series root [BCS13, Theorem 2.31]) Let $P(\overline{x},y) \in \mathbb{F}(\overline{x})[y]$, $P'(\overline{x},y) = \frac{\partial P(\overline{x},y)}{\partial y}$ and $\mu \in \mathbb{F}$ be such that $P(\overline{0},\mu) = 0$ but $P'(\overline{0},\mu) \neq 0$. Then, there is a unique power series S such that $S(\overline{0}) = \mu$ and $P(\overline{x},S) = 0$ i.e.*

$$y - S(\overline{x}) \mid P(\overline{x},y)\,.$$

Moreover, there exists a rational function y_t, $\forall t \geq 0$, such that

$$y_{t+1} \;=\; y_t - \frac{P(\overline{x},y_t)}{P'(\overline{x},y_t)} \text{ and } S \equiv y_t \bmod \langle \overline{x} \rangle^{2^t} \text{ with } y_0 = \mu\,.$$

Proof. We give an inductive proof of existence and uniqueness together. Suppose $P = \sum_{i=0}^{d} c_i y^i$. We show that there is y_t, a rational function $\frac{A_t}{B_t}$ such that $y_t \in \mathbb{F}[[\overline{x}]]$, For all $t \geq 0$, $P(\overline{x}, y_t) \equiv 0 \bmod \langle \overline{x} \rangle^{2^t}$ and for all $t \geq 1$, $y_t \equiv y_{t-1} \bmod \langle \overline{x} \rangle^{2^{t-1}}$. The proof is by induction. Let $y_0 := \mu$. Thus, the base case is true. Now suppose such y_t exists. Define $y_{t+1} := y_t - \frac{P(\overline{x},y_t)}{P'(\overline{x},y_t)}$.

Now, $y_t \equiv y_{t-1} \bmod \langle \overline{x} \rangle^{2^{t-1}} \implies y_t(\overline{0}) = \mu$. Hence $P'(\overline{x}, y_t)|_{\overline{x}=\overline{0}} = P'(\overline{0}, \mu) \neq 0$ and so $P'(\overline{x}, y_t)$ is a unit in the power series ring. So, $y_{t+1} \in \mathbb{F}[[\overline{x}]]$. Let us verify that it is an improved root of P; we use Taylor expansion. For the algebraic version of Taylor expansion, see the treatment in [Bou13].

$$
\begin{aligned}
P(\overline{x}, y_{t+1}) &= P\left(\overline{x},\ y_t - \frac{P(\overline{x}, y_t)}{P'(\overline{x}, y_t)}\right) \\
&= P(\overline{x}, y_t) - P'(\overline{x}, y_t)\frac{P(\overline{x}, y_t)}{P'(\overline{x}, y_t)} + \frac{P''(\overline{x}, y_t)}{2!}\left(\frac{P(\overline{x}, y_t)}{P'(\overline{x}, y_t)}\right)^2 - \ \cdots \\
&\equiv 0 \mod \langle \overline{x} \rangle^{2^{t+1}} .
\end{aligned}
$$

Thus, $P(\overline{x}, y_{t+1}) \equiv 0 \bmod \langle \overline{x} \rangle^{2^{t+1}}$ and $y_{t+1} \equiv y_t \bmod \langle \overline{x} \rangle^{2^t}$. This completes the induction step.

Moreover, it is not hard to see that there exists a unique power series S such that $S \equiv y_t \bmod \langle \overline{x} \rangle^{2^t}$ for all $t \geq 0$. It is unique as μ is a non-repeated root of $P(\overline{0}, y)$. As it holds for all $t \geq 0$, we must have $P(\overline{x}, S) = 0$, as otherwise $P(\overline{x}, S) \not\equiv 0 \bmod \langle \overline{x} \rangle^{2^t}$ for some $t \geq 1$. This, in particular, would imply that $P(\overline{x}, y_t) \not\equiv 0 \bmod \langle \overline{x} \rangle^{2^t}$ which is a contradiction! Thus, $P(\overline{x}, S) = 0 \iff y - S \mid P$. ◻

Remark. In a more general situation, where we have a system of n polynomials or power series in several variables $z_1, \ldots, z_n, x_1, \ldots, x_m$, we can compute the power series roots $g_1(\overline{x}), \ldots, g_n(\overline{x})$ using a multidimensional version of Newton iteration using the Jacobian. See [Bou13] for details. Also, note that there is a *slow* version of Newton iteration, which is $y_{t+1} = y_t - \frac{P(\overline{x},y_t)}{\partial_y P(\overline{0},y_0)}$. To approximate roots up to degree d, we have to use this version of Newton iteration d many times repeatedly. For restricted models, we show that the fast version of NI has crucial advantage over the slow version. There are applications of power series roots/implicit function theorem paradigm in algebraic complexity [DSY09, KS16, PSS16, BJ18], in coding theory [AP00, NRS17, BSCI+20], in polynomial system solving [GHM+98, Kri02].

Now, to get the factorization over $\mathbb{F}[\overline{x}]$, we look into the analytic factorization pattern of a polynomial over the power series ring $\mathbb{F}[[x_1, \ldots, x_n]]$. We need the notion of *uniqueness* of factorization, which the following classic proposition ensures.

Proposition 16. *[ZS75, Chap.VII] The power series ring $\mathbb{F}[[x_1, \ldots, x_n]]$ is a unique factorization domain (UFD), and so is $\mathbb{F}[[\overline{x}]][y]$.*

Now, we show that by applying a random linear map, any polynomial splits completely over the ring $\mathbb{F}[[\overline{x}]]$. (Recall: $\mathbb{F}$ is algebraically closed.)

Theorem 17 (Power Series Complete Split). *Let $f \in \mathbb{F}[\overline{x}]$ with $deg(rad(f)) =: d_0 > 0$. Consider $\alpha_i, \beta_i \in_r \mathbb{F}$ and the map $\tau : x_i \mapsto \alpha_i y + x_i + \beta_i$, $i \in [n]$, where y is a new variable.*

Then, over $\mathbb{F}[[\overline{x}]]$, $f(\tau\overline{x}) = k \cdot \prod_{i \in [d_0]} (y - g_i)^{\gamma_i}$, where $k \in \mathbb{F}^*$, $\gamma_i > 0$, and $g_i(\overline{0}) := \mu_i$. Moreover, μ_i's are distinct nonzero field elements.

Proof. Let the complete irreducible factorization of f be $\prod_{i \in [m]} f_i^{e_i}$. We apply a random τ so that f, and thus all its factors, become monic in y (Lemma 14). The monic factors $\tilde{f}_i := f_i(\tau\overline{x})$ remain irreducible (because τ is invertible). Also,

$\tilde{f}_i(\overline{0},y) = f_i(\overline{\alpha}y+\overline{\beta})$ and $\partial_y\tilde{f}_i(\overline{0},y)$ remain coprime (because $\overline{\beta}$ is random, apply Lemma 13). In other words, $\tilde{f}_i(\overline{0},y)$ is squarefree (Lemma 12).

In particular, one can write $\tilde{f}_i(\overline{0},y)$ as $\prod_{j=1}^{\deg(f_i)}(y-\mu_{i,j})$ for distinct nonzero field elements $\mu_{i,j}$ (ignoring the constant, which is the coefficient of the highest degree of y in $\tilde{f}_i$). Using classical Newton Iteration (see Lemma 15), one can write $\tilde{f}_i(\overline{x},y)$ as a product of power series $\prod_{j=1}^{\deg(f_i)}(y-g_{i,j})$, with $g_{i,j}(\overline{0}) := \mu_{i,j}$. Thus, each $f_i(\tau\overline{x})$ can be factored into linear factors in $\mathbb{F}[[\overline{x}]][y]$.

As the polynomials f_i are irreducible and coprime, by Lemma 13, it is clear that $\tilde{f}_i(\overline{0},y)$, $i \in [m]$, are mutually coprime. Thus, $\mu_{i,j}$ are distinct and they are $\sum_i \deg(f_i) = d_0$ many. Hence, after proper renaming of the roots $g_{i,j}$, $f(\tau\overline{x})$ can be completely factored as $\prod_{i\in[m]} f_i(\tau\overline{x})^{e_i} = \prod_{i\in[d_0]}(y-g_i)^{\gamma_i}$, with $\gamma_i > 0$ and the field constants $g_i(\overline{0})$ being distinct. □

Corollary 18. *Suppose g is a polynomial factor of f. As before let $f(\tau\overline{x}) = \prod_{i\in[m]} f_i(\tau\overline{x})^{e_i} = k \cdot \prod_{i\in[d_0]}(y-g_i)^{\gamma_i}$. As $g(\tau x) \mid f(\tau\overline{x})$ we deduce that $g(\tau x) = k' \cdot \prod (y-g_i)^{c_i}$ with $0 \le c_i \le \gamma_i$ and $k' \in \mathbb{F}^*$. If g irreducible, then $c_i \in \{0,1\}$.*

Moreover, we can get back g by applying τ^{-1} on the resulting polynomial $g(\tau\overline{x})$.

3.1 Reducing factoring to power series root approximation:

Using the power series complete split (Theorem 17), we show that multivariate polynomial factoring reduces to power series root finding up to a certain precision. Following the above notation f splits as $f(\tau\overline{x}) = \prod_{i=1}^{d_0}(y-g_i)^{\gamma_i}$. For all $t \ge 0$, it is easy to see that $f(\tau\overline{x}) \equiv \prod_{i=1}^{d_0}(y-g_i^{\le t})^{\gamma_i} \mod I^{t+1}$, where $I := \langle x_1,\ldots,x_n\rangle$. Note that there is a one-one correspondence, induced by τ, between the polynomial factors of f and $f(\tau\overline{x})$ (because τ is invertible and f is y-free).

Next, we show case by case how to find a *polynomial* factor of $f(\tau\overline{x})$ from the approximate power series roots.

Case 1- Computing a linear factor of the form $y-g(\overline{x})$: If the degree of the input polynomial is d, all the non-trivial factors have degrees $\le (d-1)$. So, if we compute the approximations of all the power series roots (wrt y) up to the precision of degree $t = d-1$, then we can recover all the factors of the form $y - g(x_1,\ldots,x_n)$. Technically, this is supported by the uniqueness of the power series factorization (Proposition 16).

Case 2- Computing a monic non-linear factor: Assume that a factor g of total degree t is of the form $y^k + c_{k-1}y^{k-1} + \cdots + c_1y + c_0$, where for all i, $c_i \in \mathbb{F}[\overline{x}]$. Now, this factor g also splits into linear (in y) factors over $\mathbb{F}[[\overline{x}]]$ and obviously, these linear factors are also linear factors of the original polynomial $f(\tau\overline{x})$. So we have to take the right combination of some k power series roots, with their approximations (up to the degree t wrt $\overline{x}$), and take the product mod I^{t+1}. Note that if we only want to give an existential proof of the size bound of the factors (as required for Theorem 1), we need not find the combination of the power series roots forming a factor algorithmically. Doing it through brute-force search takes exponential time ($\binom{d}{k}$ choices). Interestingly, using a classical idea (solving a linear system), it can be done in randomized polynomial time. We will spell out the ideas later while discussing the algorithm part of Theorem 3.

4 MAIN RESULTS: HIGH DEGREE CIRCUITS

This section proves Theorems 1–2. The proofs are self-contained and we assume for the sake of simplicity that the underlying field $\mathbb{F}$ is algebraically closed and has characteristic 0. When this is not the case, we discuss the corresponding theorems in Section 6.

4.1 Factors of a circuit with low-degree radical: Proof of Theorem 1

We simultaneously find the approximations of all the power series roots g_i of $f(\tau\overline{x})$. At each recursive step, we get a better precision wrt degree. We show that knowing the approximations $g_i^{<\delta}$, of g_i (for all i) up to degree $\delta-1$, is enough to calculate approximations of g_i (simultaneously for all i) up to degree δ.

In this section, we use Theorem 17.

Proof of Theorem 1. From the hypothesis $f = u_0 u_1$. Define $\deg(f) =: d$. Suppose $u_1 = h_1^{e_1} \dots h_m^{e_m}$, where h_i's are coprime irreducible polynomials. Let d_0 be the degree of $\mathrm{rad}(u_1) = \prod_i h_i$. Note that $\deg(h_i), m \le d_0$ and the multiplicity $e_i \le d \le 2^s$, where s is the size bound of the input circuit. Thus, to get the size bound of any factor of u_1, it is enough to show that for each i, h_i has a circuit of size $\mathrm{poly}(sd_0)$.

Using Theorem 17, we have $\tilde{f}(\overline{x}, y) := f(\tau\overline{x}) = k \cdot u_0(\tau\overline{x}) \cdot \prod_{i\in[d_0]}(y-g_i)^{\gamma_i}$, with $g_i(\overline{0}) := \mu_i$ being distinct nonzero field elements. As $\deg(h_i) \le d_0$; from Corollary 18, we deduce that for nonzero $k_i \in \mathbb{F}$,

$$h_i(\tau\overline{x}) \;\equiv\; k_i \cdot \prod_{i\in[d_0]} (y - g_i^{\le d_0})^{\delta_i} \mod I^{d_0+1} \quad \text{where} \quad I \;:=\; \langle x_1, \dots, x_n \rangle$$

where exponent $\delta_i \in \{0, 1\}$. We can get h_i by applying τ^{-1}. Hence, it is enough to bound the size of $g_i^{\le d_0}$.

Let $\tilde{u_0} := u_0(\tau\overline{x})$. From the repeated applications of Leibniz rule of the derivative ($\partial_y(FG) = (\partial_y F)G + F(\partial_y G)$.), we deduce a classic logarithmic derivative identity,

$$\frac{\partial_y \tilde{f}}{\tilde{f}} \;=\; \frac{\partial_y \tilde{u_0}}{\tilde{u_0}} + \sum_{i=1}^{d_0} \frac{\gamma_i}{(y-g_i)}.$$

At this point, we move to the formal power series so that the reciprocals can be approximated as polynomials. Note that $y - g_i$ is invertible in $\mathbb{F}[[\overline{x}]]$ when y is assigned any value $c_i \in \mathbb{F}$, which is not equal to μ_i. We intend to find $g_i \mod I^\delta$ inductively, for all $\delta \ge 1$. We assume that μ_i's and γ_i's are known. Suppose, we have recovered up to $g_i \mod I^\delta$ and we want to recover $g_i \mod I^{\delta+1}$. The relevant recurrence, for $\delta \ge 1$, is:

Claim 2 (Recurrence). $\displaystyle\sum_{i=1}^{d_0} \gamma_i \cdot \frac{g_i^{=\delta}}{(y-\mu_i)^2} \;\equiv\; \frac{\partial_y \tilde{f}}{\tilde{f}} - \frac{\partial_y \tilde{u_0}}{\tilde{u_0}} - \sum_i \frac{\gamma_i}{(y-g_i^{<\delta})} \mod I^{\delta+1}.$

Proof of Claim 2. Using a power series calculation (Lemma 20), we have

$$\frac{1}{y-g_i} \equiv \frac{1}{y - \left(g_i^{<\delta} + g_i^{=\delta}\right)} \;\equiv\; \frac{1}{y-g_i^{<\delta}} + \frac{g_i^{=\delta}}{(y-\mu_i)^2} \mod I^{\delta+1}.$$

Multiplying by γ_i and summing over $i \in [d_0]$, the claim follows. □

We will compute $g_i^{\le\delta}$ incrementally; in particular knowing approximation up to the $\delta-1$ homogeneous parts of g_i, we will find the δ-th part by solving a linear system. Concretely, assume that we have already computed a rational function $g'_{i,\delta-1}$ of the form $g'_{i,\delta-1} := C_{i,\delta-1}/D_{i,\delta-1}$ such that $g'_{i,\delta-1}$ approximates g_i correctly *up to* $\delta - 1$, i.e. $g'_{i,\delta-1} \equiv g_i^{<\delta} \mod I^\delta$.

In the free variable y in Claim 2, we plug-in d_0 random field values c_i and get the following system of linear equations: $M \cdot v_\delta = W_\delta$, where M is a $d_0 \times d_0$ matrix while both v_δ and W_δ are both $d_0 \times 1$ matrices. Both M and W_δ are known matrices; in particular:

$$M(i,j) \;:=\; \frac{\gamma_j}{(c_i-\mu_j)^2}\,, \qquad W_\delta(i) \;:=\; \left.\left(\frac{\partial_y \tilde{f}}{\tilde{f}} - \frac{\partial_y \tilde{u_0}}{\tilde{u_0}}\right)\right|_{y=c_i} - \tilde{G}_{i,\delta} \;\text{ where }\; \tilde{G}_{i,\delta} \;:=\; \sum_{j=1}^{d_0} \frac{\gamma_j}{(c_i - g'_{j,\delta-1})}$$

We want to solve for the *unknown* v_δ whose i-th entry is $v_\delta(i)$. We define the *rational* function $g'_{i,\delta}$ *iteratively* as follows:

$$g'_{i,\delta} = \begin{cases} \mu_i, & \text{when } \delta = 0 \\ g'_{i,\delta-1} + v_\delta(i), & \delta \geq 1 \end{cases}$$

We show that $g'_{i,\delta}$ approximates g_i up to δ where we crucially use the fact that M is invertible (Lemma 19).

Claim 3 (Self-correction). Let $i \in [d_0]$ and $\delta \geq 0$. Then, $g'_{i,\delta} \equiv g_i^{\leq \delta} \bmod I^{\delta+1}$.

Proof of Claim 3. We prove this by induction on δ. It is true for $\delta = 0$ by definition. Suppose it is true for $\delta - 1$. This means we have $g'_{i,\delta-1} \equiv g_i^{<\delta} \bmod I^\delta$ for all i. Note that, $g'_{i,\delta-1} \in \mathbb{F}(\overline{x}) \bigcap \mathbb{F}[[\overline{x}]]$. Let us write

$$g'_{i,\delta-1} \;=:\; g_i^{<\delta} + A_{i,\delta} + A'_{i,\delta} \;\text{ where }\; A'_{i,\delta} \equiv 0 \bmod I^{\delta+1}$$

Here $A_{i,\delta}$ is homogeneous of degree δ. Hence, for $i \in [d_0]$, the linear constraint is:

$$\sum_{j=1}^{d_0} \gamma_j \cdot \frac{v_\delta(j)}{(c_i - \mu_j)^2} \equiv \frac{\partial_y \tilde{f}}{\tilde{f}} - \frac{\partial_y \tilde{u_0}}{\tilde{u_0}} - \sum_j \frac{\gamma_j}{(c_i - g'_{j,\delta-1})} \bmod I^{\delta+1} \tag{1}$$

The "garbage" term $A_{j,\delta}$ in RHS can be isolated using Lemma 20 as:

$$\frac{1}{(c_i - g'_{j,\delta-1})} \equiv \frac{1}{c_i - \left(g_j^{<\delta} + A_{j,\delta}\right)} \equiv \frac{1}{(c_i - g_j^{<\delta})} + \frac{A_{j,\delta}}{(c_i - \mu_j)^2} \bmod I^{\delta+1} \tag{2}$$

Plugging Eqn. (2) into (1), we get:

$$\sum_{j=1}^{d_0} \frac{\gamma_j \cdot v_\delta(j)}{(c_i - \mu_j)^2} \equiv \frac{\partial_y \tilde{f}}{\tilde{f}} - \frac{\partial_y \tilde{u_0}}{\tilde{u_0}} - \sum_{j=1}^{d_0} \frac{\gamma_j}{c_i - g_j^{<\delta}} - \sum_{j=1}^{d_0} \frac{\gamma_j \cdot A_{j,\delta}}{(c_i - \mu_j)^2} \bmod I^{\delta+1}.$$

Rewriting this, using Claim 2, we get:

$$\sum_{j=1}^{d_0} \frac{\gamma_j}{(c_i - \mu_j)^2} \left(v_\delta(j) + A_{j,\delta}\right) \equiv \sum_{j=1}^{d_0} \frac{\gamma_j}{(c_i - \mu_j)^2} \cdot g_j^{=\delta} \bmod I^{\delta+1}.$$

Rearranging, we get that

$$\sum_{j=1}^{d_0} \gamma_j \cdot \frac{(v_\delta(j) + A_{j,\delta} - g_j^{=\delta})}{(c_i - \mu_j)^2} \equiv 0 \bmod I^{\delta+1}.$$

As we vary $i \in [d_0]$, we deduce, by Lemma 19, that $v_\delta(j) + A_{j,\delta} - g_j^{=\delta} \equiv 0 \bmod I^{\delta+1}$. Hence, for all $j \in [d_0]$:

$$\begin{aligned} g'_{j,\delta} &= g'_{j,\delta-1} + v_\delta(j) \\ &\equiv (g_j^{<\delta} + A_{j,\delta}) + (g_j^{=\delta} - A_{j,\delta}) \\ &\equiv g_j^{\leq\delta} \bmod I^{\delta+1}. \end{aligned}$$

□

Lemma 19 (Matrix inverse). *Let $\mu_i, i \in [d]$, be distinct nonzero elements in $\mathbb{F}$. Define a $d \times d$ matrix A with the (i, j)-th entry $\frac{1}{(y_i - \mu_j)^2}$. Its entries are in the function field $\mathbb{F}(\overline{y})$. Then, det$(A) \neq 0$.*

Corollary. *As* $det(A) \in \mathbb{F}(y)\backslash\{0\}$, *using Polynomial Identity Lemma (Lemma 8), it follows that the matrix* M *where* $M(i,j) := \frac{1}{(c_i-\mu_j)^2}$ *for* $c_i \in_r \mathbb{F}$, *is* invertible.

Proof of Lemma 19. The idea is to consider the power series of the function $1/(y_i - \mu_j)^2$ and show that a monomial appears nontrivially in that of $\det(A)$.

We first need a claim about the coefficient operator on the determinant.

Claim 4. Let $f_j = \sum_{i\geq 0} \beta_{j,i} x^i$ be a power series in $\mathbb{F}[[x]]$, for $j \in [d]$. Then, $\mathrm{Coeff}_{\overline{x}^{\overline{\alpha}}} \circ \det\left(f_j(x_i)\right) = \det\left(\beta_{j,\alpha_i}\right)$.

Proof of Claim 4. Observe that the rows of the matrix have disjoint variables. Thus, $x_i^{\alpha_i}$ could be produced only from the i-th row. This proves:

$$\mathrm{Coeff}_{\overline{x}^{\overline{\alpha}}} \circ \det\left(f_j(x_i)\right) = \det\left(\mathrm{Coeff}_{x_i^{\alpha_i}} \circ f_j(x_i)\right) = \det\left(\beta_{j,\alpha_i}\right) .$$

□

By Taylor expansion we have

$$\frac{1}{(x-\mu)^2} = \frac{1}{\mu^2}\sum_{j\geq 1} j\left(\frac{x}{\mu}\right)^{j-1} .$$

Hence, the coefficient of y_i^{i-1} in $A(i,j)$ is

$$\frac{1}{\mu_j^2}\frac{i}{\mu_j^{i-1}} = \frac{i}{\mu_j^{i+1}} .$$

By the above claim, the coefficient of $\prod_{i\in[d]} y_i^{i-1}$ in $\det(A)$ is: $\det\left(\left(\frac{i}{\mu_j^{i+1}}\right)\right)$. By cancelling i (from each row) and $1/\mu_j^2$ (from each column), we simplify it to the Vandermonde determinant:

$$\det\begin{bmatrix} \frac{1}{\mu_1^0} & \frac{1}{\mu_2^0} & \cdots & \frac{1}{\mu_d^0} \\ \frac{1}{\mu_1^1} & \frac{1}{\mu_2^1} & \cdots & \frac{1}{\mu_d^1} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{1}{\mu_1^{d-1}} & \frac{1}{\mu_2^{d-1}} & \cdots & \frac{1}{\mu_d^{d-1}} \end{bmatrix} = \prod_{i<j\in[d]}\left(\frac{1}{\mu_i} - \frac{1}{\mu_j}\right) \neq 0 .$$

Hence, the determinant of A is nonzero. □

Remarks.

(1) If the characteristic of $\mathbb{F}$ is a prime $p \geq 2$, then the above proof needs a slight modification. One should consider the coefficient of $\prod_{i\in[d]} y_i^{s_i-1}$ in $\det(A)$ for a set $S = \{s_1,\ldots,s_d\}$ of distinct non-negative integers that are not divisible by p. Moreover, must consider 'random' μ_i's to deduce $\det(A) \neq 0$.

(2) The matrix A defined by $A(i,j) := 1/(y_i - \mu_j)$ is the famous *Cauchy* matrix. It is not hard to show that the same proof of Lemma 19 works to establish the Cauchy matrix's invertibility. For details on Cauchy matrix, see [Wik].

The following lemma is crucially used in Equation 2.

Lemma 20 (Series inverse). *Let* $\delta \geq 1$. *Assume that* A *is a polynomial of degree* $< \delta$ *and* B *is a homogeneous polynomial of degree* $\delta \geq 1$, *such that* $A(\overline{0}) =: \mu \neq 0$. *Then, we have the following identity in* $\mathbb{F}[[\overline{x}]](y) \cap \mathbb{F}[[\overline{x}]][[y]]$:

$$\frac{1}{y-(A+B)} \equiv \frac{1}{y-A} + \frac{B}{(y-\mu)^2} \mod \langle\overline{x}\rangle^{\delta+1}$$

Remark. Since $\mu \neq 0$ and $\delta \geq 1$, the expression $1/(y-(A+B))$ is a power series in both $\overline{x}$ and y (and also a rational function in y, with coefficients from $\mathbb{F}[[\overline{x}]]$). This can been easily seen just by considering $(-1/(A+B)) \cdot 1/(1-y/(A+B))$ and noting the fact that $1/(A+B) \in \mathbb{F}[[\overline{x}]]$.

Proof. We will use the notation $A^{[1,\delta-1]}$ to refer to the sum of the homogeneous parts of A of degrees between 1 and $\delta-1$ (equivalently, it is $A^{<\delta}-\mu$). Note that $B \cdot A^{[1,\delta-1]}$ vanishes mod $\langle\overline{x}\rangle^{\delta+1}$. Now, in $\mathbb{F}[[\overline{x}]][[y]]$,

$$\begin{aligned}
\frac{1}{y-(A+B)} &\equiv \frac{1}{y-\mu-\left(A^{[1,\delta-1]}+B\right)} \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-\mu}\left(\frac{1}{1-\frac{A^{[1,\delta-1]}+B}{y-\mu}}\right) \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-\mu}\left(1+\left(\frac{A^{[1,\delta-1]}+B}{y-\mu}\right)+\left(\frac{A^{[1,\delta-1]}+B}{y-\mu}\right)^2+\ldots\ldots\right) \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-\mu}\left(1+\left(\frac{A^{[1,\delta-1]}+B}{y-\mu}\right)+\left(\frac{A^{[1,\delta-1]}}{y-\mu}\right)^2+\left(\frac{A^{[1,\delta-1]}}{y-\mu}\right)^3+\ldots\ldots\right) \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-\mu}\left(1+\left(\frac{A^{[1,\delta-1]}}{y-\mu}\right)+\left(\frac{A^{[1,\delta-1]}}{y-\mu}\right)^2+\ldots\ldots\right)+\frac{B}{(y-\mu)^2} \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-\mu}\left(\frac{1}{1-\frac{A^{[1,\delta-1]}}{y-\mu}}\right)+\frac{B}{(y-\mu)^2} \bmod \langle\overline{x}\rangle^{\delta+1} \\
&\equiv \frac{1}{y-A}+\frac{B}{(y-\mu)^2} \bmod \langle\overline{x}\rangle^{\delta+1} \quad .
\end{aligned}$$

□

Size analysis: Here we give the overall process of finding factors using the allRootsNI technique and analyze the circuit size needed at each step to establish the size bound of the factors. As discussed before, we need to analyze only the power series root approximation $g_i^{\leq\delta}$ or $g'_{i,\delta}$.

At the $(\delta-1)$-th step of the allRootsNI process, we have a multi-output circuit (with division gates) computing $g'_{i,\delta-1}$ as a rational function, for all $i \in [d_0]$. Specifically, let us assume that $g'_{i,\delta-1} =: C_{i,\delta-1}/D_{i,\delta-1}$, where $D_{i,\delta-1}$ is invertible in $\mathbb{F}[[\overline{x}]]$. So, the circuit computing $g'_{i,\delta-1}$ has a division gate at the top that outputs $C_{i,\delta-1}/D_{i,\delta-1}$. We would eliminate this division gate only in the end (see the standard Lemma 6). Now we show how to construct the circuit for $g'_{i,\delta}$, given the circuits for $g'_{i,\delta-1}$.

From $v_\delta = M^{-1}W_\delta$, it is clear that there exist field elements β_{ij} such that

$$v_\delta(i) = \sum_{j=1}^{d_0} \beta_{ij} \cdot W_\delta(j) = \sum_{j=1}^{d_0} \beta_{ij} \cdot \left(\left(\frac{\partial_y \tilde{f}}{\tilde{f}} - \frac{\partial_y \tilde{u_0}}{\tilde{u_0}}\right)\Bigg|_{y=c_j} - \tilde{G}_{j,\delta}\right)$$

Initially, we precompute, for all $j \in [d_0]$, $(\partial_y \tilde{f}/\tilde{f} - \partial_y \tilde{u_0}/\tilde{u_0})|_{y=c_j}$: Note that $\partial_y \tilde{f}$ has poly(s) size circuit (high degree of the circuit does not matter, see Lemma 7). Invertibility of $\tilde{f}|_{y=c_j}$ and $\tilde{u_0}|_{y=c_j}$ follows from the fact that we chose c_j's randomly. In particular, $\tilde{f}(\overline{0},y)$, and so $\tilde{u_0}(\overline{0},y)$, have roots in $\mathbb{F}$ which are distinct from c_j, $j \in [d_0]$. Thus, $\tilde{f}(\overline{x},c_j)$ and $\tilde{u_0}(\overline{x},c_j)$ have nonzero constants and so are invertible in $\mathbb{F}[[\overline{x}]]$. Similarly, $\gamma_\ell/(c_j - g'_{\ell,\delta-1})$ exists in $\mathbb{F}[[\overline{x}]]$.

Thus, the matrix recurrence allows us to calculate the polynomials $C_{i,\delta}$ and $D_{i,\delta}$, given their $\delta - 1$ analogs, by adding poly(d_0) many wires and nodes. The precomputations cost us size poly(s, δ). Hence, both $C_{i,\delta}$ and $D_{i,\delta}$ has poly(s, δ, d_0) sized circuit.

We can assume we have only one division gate at the top, as for each gate G we can keep track of the numerator and the denominator of the rational function computed at G, and easily simulate all the algebraic operations in this representation. When we reach precision $\delta = d_0$, we can eliminate the division gate at the top. As D_{i,d_0} is a unit, we can compute its inverse using the power series inverse formula and approximate only up to degree d_0 (Lemma 5). Finally, the circuit for the polynomial $g_i^{\leq d_0} \equiv C_{i,d_0}/D_{i,d_0} \mod I^{d_0+1}$, for all $i \in [d_0]$, has size poly(s, d_0).

Altogether, it implies that any factor of u_1 has a circuit of size poly(s, d_0). □

4.2 Low degree factors of general circuits: Proof of Theorem 2

As a consequence of the reduction presented in Section 3.1, a direct approach to proving the Factor Conjecture is via computing arithmetic circuits of small size giving approximations (up to some low degree) of power series roots that have *high* multiplicity. First, we need to find methods for approximating roots with multiplicity ≥ 2. The classical Newton iteration formula fails here, but a simple modification of Newton iteration works if we know the multiplicity of the root. In the numerical analysis literature [DB08], this is known as modified/generalized Newton iteration with multiplicity. Using this method, we give a plausible approach to prove the Factor Conjecture.

Newton Iteration with multiplicity: The following is a generalized Newton Iteration formula, as it works with any multiplicity $e > 0$.

Lemma 21 (NI with multiplicity). *If $f(\overline{x}, y) = \prod_{i=1}^{d_0}(y - g_i)^{\gamma_i}$, where $g_i \mod I$ are non-zero and distinct, and $\gamma_i > 0$, then the each power series g_i can be approximated by the recurrence:*

$$y_{t+1} \;:=\; y_t \;-\; \gamma_i \cdot \left.\frac{f}{\partial_y f}\right|_{y=y_t} \tag{3}$$

where $y_t \equiv g_i \mod I^{2^t}$.

Proof. We will show the above for g_1. We remark that y_t is a rational function; this is easy to prove by simple induction (on t). Rewrite

$$\frac{\partial_y f}{f} \;=\; \sum_{i=1}^{d_0} \frac{\gamma_i}{(y - g_i)} \;=\; \frac{(1 + L_1) \cdot \gamma_1}{(y - g_1)} \quad \text{where } \; L_1 \;:=\; \sum_{1 < i \leq d_0} \frac{\gamma_i}{y - g_i} \cdot \frac{y - g_1}{\gamma_1}$$

This implies $f/\partial_y f = (1 + L_1)^{-1} \cdot (y - g_1)/\gamma_1$. Now, if we put $y = y_t := g_1^{<2^t}$, then $y_t - g_i = g_1^{<2^t} - g_i$ is a unit in $\mathbb{F}[[\overline{x}]]$ for $i \neq 1$ (because it is a nonzero constant mod I) i.e. $1/(y_t - g_i)$ must be a power series. Also, $y_t - g_1 = g_1^{<2^t} - g_1 \equiv 0 \mod I^{2^t}$. Together they imply that $L_1|_{y=y_t} \equiv 0 \mod I^{2^t}$. Hence,

$$f/\partial_y f\big|_{y=y_t} \equiv (y_t - g_1)/\gamma_1 \cdot (1 + L_1)^{-1} \equiv ((y_t - g_1)/\gamma_1) \cdot (1 - L_1 + L_1^2 - \ldots) \equiv (y_t - g_1)/\gamma_1 \mod I^{2^{t+1}} . \tag{4}$$

The last expression implies that $y_t - \gamma_1 \cdot f/\partial_y f\big|_{y=y_t} \equiv g_1 \bmod I^{2^{t+1}}$, as desired. □

Remarks. (1) Note that, when $\gamma_1 = 1$, (i.e g_1 is a *simple root* of f), the above is an alternate proof of the classical Newton Iteration (NI) [New69] that finds a simple root in a recursive way (see Lemma 15).

(2) There is a subtle point about Equation 3 when $\gamma_1 \geq 2$. The denominator $\partial_y f|_{y=y_t}$ is zero mod I, thus, its reciprocal does not exist! However, the ratio $(f/\partial_y f)\big|_{y=y_t}$ does exist in $\mathbb{F}[[\overline{x}]]$ (Equation 4); one can think of first reducing

the rational function $(f/\partial_y f)$, to A/B, where A and B are coprime (wrt y) and then evaluating at $y = y_t$. In particular, this means that $(f/\partial_y f)\big|_{y=y_t}$ *always exists* even if $y_t = g_1$, for some t and $\gamma_1 \geq 2$; in this case both f and $\partial_y f$, at g_1 are 0, while the reduced expression is eventually 0 mod $I^{2^{t+1}}$ (and **not** anything illegal like $0/0$).

(3) Also, note that efficiently reducing the rational function $(f/\partial_y f)$ is still *open*. Although f and $\partial_y f$ have size s circuits, we do not know if their gcd has a circuit of size poly(s) as the gcd can be of high degree. Even if one can compute gcd by a small circuit, we have to perform division by a high degree polynomial, which is again an open question [Kal87].

(4) If $\gamma_1 = 1$ then the denominator $\partial_y f|_{y=y_t}$ is nonzero mod I, thus, it is invertible in $\mathbb{F}[[\overline{x}]]$ and that is necessary for fast algebraic circuit computation (esp. division elimination).

Using Newton iteration with multiplicity, our Theorem 2 reduces factor conjecture to a new problem, the *modular division* problem. The problem is to show that if f/g has a representative in $\mathbb{F}[[\overline{x}]]$, where polynomials f and g can be computed by a circuit of size s, then f/g mod $\langle \overline{x}^d \rangle$ can be computed by a circuit of size poly(sd). Note that if g is invertible in $\mathbb{F}[[\overline{x}]]$, then the question of modular division can be solved using Strassen's method of division elimination [Str73]. But, in our case g is not invertible in $\mathbb{F}[[\overline{x}]]$ (though f/g is well-defined). The trick of applying a random shift to make the denominator g invertible does not work here as f/g is not a polynomial.

Proof of Theorem 2. As discussed before, to show the size bound for an arbitrary factor (with low degree) of f, it is enough to show the size bound for the approximations of power series roots. From Theorem 17, $\tilde{f}(\overline{x}, y) = f(\tau\overline{x}) = k \cdot \prod_{i=1}^{d_0}(y - g_i)^{\gamma_i}$, with $g_i(\overline{0}) := \mu_i$ being distinct.

Fix an i from now on. We assume we know the multiplicity γ_i. Note that we need to know the multiplicity of the root *exactly* to apply NI with multiplicity; here, we will simply guess them non-uniformly. To calculate $g_i^{\leq d}$, we iteratively use Newton iteration with multiplicity for $\log d + 1$ many times. We know that there are rational functions $\hat{g}_{i,t}$ such that $\hat{g}_{i,t+1} := \hat{g}_{i,t} - \gamma_i \cdot \frac{\tilde{f}}{\partial_y \tilde{f}}\big|_{y=\hat{g}_{i,t}}$ and $\hat{g}_{i,t} \equiv g_i \bmod \langle \overline{x} \rangle^{2^t}$. We compute the rational functions $\hat{g}_{i,t}$ incrementally, $0 \leq t \leq \log d + 1$, by a circuit with division gates. As before, $\tilde{f}$ and $\partial_y \tilde{f}$ have poly(s) size circuits (Lemma 7).

If $\hat{g}_{i,t}$ has S_t size circuit with division, then $S_{t+1} = S_t + \text{poly}(s)$. Hence, $\hat{g}_{i,\log d+1}$ has poly$(s, \log d)$ size circuit with the division gates.

By keeping track of the numerator and the denominator of the rational function computed at each gate, we can assume that the only division gate is at the top. As the size of $\hat{g}_{i,\log d+1}$ was initially poly$(s, \log d)$ with intermediate division gates, it is easy to see that when division gates are pushed at the top, it computes A/B with the size of both A and B still poly$(s, \log d)$.

Finally, a degree d polynomial factor $h|f$ will require us to estimate polynomials $g_i^{\leq d}$ for that many i. Thus, such a factor has poly(sd) size circuit, using a single modular division. ◻

5 CLOSURE OF RESTRICTED COMPLEXITY CLASSES: PROOF OF THEOREM 3

This subsection is dedicated towards proving closure results for certain algebraic complexity classes. In fact, for fields like $\mathbb{Q}$, $\mathbb{Q}_p$, or $\mathbb{F}_q$ for prime-power q, we also give efficient randomized algorithms to output the complete factorization of polynomials belonging to that class (stated as Theorem 26). We use the notation $g \,||\, f$ to denote that g divides f but g^2 does not divide f. Again, we denote $I := \langle x_1, \ldots, x_n \rangle$.

Proof of Theorem 3. There are essentially two parts in the proof. The first part talks only about the existential closure results. In the second part, we discuss the algorithmic approach.

Proof of closure: Given f of degree d, we randomly shift by $\tau : x_i \mapsto x_i + y\alpha_i + \beta_i$. From Theorem 17 we have that $\tilde{f}(\overline{x}, y) := f(\tau\overline{x})$ splits like $\tilde{f} = \prod_{i=1}^{d_0}(y - g_i)^{\gamma_i}$, with $g_i(\overline{0}) =: \mu_i$ being distinct. Here is the detailed size analysis of the factors of polynomials represented by various models of our interest.

Size analysis for formula: Suppose f has a formula of size $n^{O(\log n)}$. To show size bound for all the factors, it is enough to show that the approximations of the power series roots, i.e. $g_i^{\leq d}$ has size $n^{O(\log n)}$ size formula. This follows from the reduction of factoring to approximations of power series roots (Section 3.1).

We differentiate $\tilde{f}$ wrt y, $(\gamma_i - 1)$ many times, so that the multiplicity of the root we want to recover becomes exactly one. The differentiation would keep the size $\text{poly}(n^{\log n})$ (Lemma 7). Now, we have $(y - g_i) \mid\mid \tilde{f}^{(\gamma_i - 1)}$, and we can apply classical Newton iteration formula (Lemma 15). For all $0 \leq t \leq \log d + 1$, we compute A_t and B_t such that $A_t/B_t \equiv g_i \bmod I^{2^t}$. Moreover, B_t is invertible in $\mathbb{F}[[\overline{x}]]$ (because g_i is a simple root of $\tilde{f}^{(\gamma_i - 1)}$).

To implement this iteration using the formula model, each time there would be a blow-up of d^2. Note that in a formula, there can be many copies of the same variable in the leaf nodes and if we want to feed something in that variable, we have to make equally many copies. That means we may need to make s (which is $\text{size}(f)$) many copies at each step. We claim that it can be reduced to only d^2 many copies.

We can pre-compute (with blow-up at most $\text{poly}(sd)$) all the coefficients $C_0, \ldots, C_d$ wrt y, given the formula of $\tilde{f} =: C_0 + C_1 y + \cdots + C_d y^d$ using interpolation. We can do the same for the derivative formula. For details on this interpolation trick, see Section 2.1. Using interpolation, we can convert the formula of $\tilde{f}$ and its derivative to the form $C_0 + C_1 y + \cdots + C_d y^d$. In this modified formula, there are $O(d^2)$ many leaves labelled as y. So in the modified formula of the polynomial $\tilde{f}$ and in its derivative, we are computing and plugging in (for y) d^2 copies of $g_i^{<2^t}$ to get $g_i^{<2^{t+1}}$. This leads to d^2 blow up at each step of the iteration.

As the denominators B_t are invertible, we can keep track of the division gates across iterations and, in the end, eliminate them, causing a one-time size blow up of $\text{poly}(sd)$ (Lemma 6).

Now, assume that $\max(\text{size}(A_t), \text{size}(B_t)) \leq S_t$. Then we have $S_{t+1} \leq O(d^2 S_t) + \text{poly}(sd)$. Finally, we have $S_{\log d+1} = \text{poly}(sd) \cdot d^{2\log d} = \text{poly}(n^{\log n})$.

Hence, $g_i^{\leq d} \equiv A_{\log d+1}/B_{\log d+1} \bmod I^{d+1}$ has $\text{poly}(n^{\log n})$ size formula, and so does every polynomial factor of f after applying τ^{-1}.

Size analysis for ABP: This analysis is similar to that of the formula model, as the size blow-up in each NI iteration for differentiation, division, and truncation (to degree $\leq d$) is the same as that for formulas. A noteworthy difference is that we need to eliminate division in *every* iteration (Lemma 5) and we cannot postpone it. This leads to a blow-up of d^4 in each step. Hence, $S_{\lg d+1} = \text{poly}(sd) \cdot d^{4\log d} = \text{poly}(n^{\log n})$.

Size analysis for VNP: Suppose f can be computed by a verifier circuit of size, and witness size, $n^{O(\log n)}$. We call both the verifier circuit size and witness size as size parameter. Now, our given polynomial $\tilde{f}$ has $n^{O(\log n)}$ size parameters. As before, it is enough to show that $g_i^{\leq d}$ has $n^{O(\log n)}$ size parameters.

For the pre-processing (taking $\gamma_i - 1$-th derivative of $\tilde{f}$ wrt y), the blow-up in the size parameters is only $\text{poly}(n^{\log n})$. Now we analyze the blow-up due to classical Newton iteration. We compute A_t and B_t such that $A_t/B_t \equiv g_i \bmod I^{2^t}$. Using the closure properties of VNP (discussed in Section 2.1), we see that each time there is a blow-up of d^4. The main reason for this blow-up is due to the *composition* operation, as we are feeding a polynomial into another polynomial.

Assume that the verifier circuit $\max(\text{size}(A_t), \text{size}(B_t)) \leq S_t$ and witness size $\leq W_t$. Then we have $S_{t+1} \leq O(d^4 S_t) + \text{poly}(n^{\log n})$. So, finally, we have $S_{\log d+1} = \text{poly}(sd) \cdot d^{4\log d} = \text{poly}(n^{\log n})$. It is clear that $g_i^{\leq d} \equiv A_{\log d+1}/B_{\log d+1}$ mod

I^{d+1} has poly($n^{\log n}$) size verifer circuit. The same analysis works for W_t, and the witness size remains $n^{O(\log n)}$. Moreover, we get the corresponding bounds for every polynomial factor of f after applying τ^{-1}.

Remark. Recently, Chou, Kumar and Solomon [CKS19b] have improved our result on VNP, showing that VNP is closed under factors. Also note that, we get a short non-algorithmic proof of the closure of VP under factors using Newton iteration and the reduction of factor computation to approximating power series roots (Section 3.1). [CKS19a] gave another short proof for the same using the multidimensional version of Newton iteration.

5.1 Randomized factoring algorithm for formulas and ABPs

This subsection is dedicated to the design and analysis of the constructive (algorithmic) part to factorize a given poly(n) degree and poly($n^{\log n}$) size formula or ABP.

We need the following lemma (adapted from [KSS15]) that discusses how to perform linear algebra when the coefficients of vectors are given as formulas (respectively ABPs).

Lemma 22. *(Linear algebra using PIT [KSS15, Lemma 2.6]) Let $M = (M_{i,j})_{k\times n}$ be a matrix (where k is $n^{O(1)}$) with each entry being a degree $\leq n^{O(1)}$ polynomial in $\mathbb{F}[\overline{x}]$. Suppose, we have an algebraic formula (respectively ABP) of size $\leq n^{O(\log n)}$ computing each entry. Then, there is a randomized poly($n^{\log n}$)-time algorithm that either:*

- *finds a formula (respectively ABP) of size poly($n^{\log n}$) computing a nonzero $u \in (\mathbb{F}[\overline{x}])^n$ such that $Mu = 0$, or*
- *outputs 0 which declares that $u = 0$ is the only solution.*

Proof. This was proved in [KSS15, Lemma 2.6] for the circuit model. Since we are using a different model, we repeat the details. The idea is the following. Iteratively, for every $r = 1, \ldots, n$ we shall find an $r \times r$ minor contained in the first r columns that is full rank. While continuing this process, we either reach $r = n$ in which case it means that the matrix has full column rank, hence, $u = 0$ is the only solution, or we get stuck at some value say $r = r_0$. We use the fact that r_0 is rank, and using this minor we construct the required nonzero vector u.

We explain the process in a bit more detail. Using a randomized algorithm, we look for some nonzero entry in the first column. If no such entry is found we can simply take $u = (1, 0, \ldots, 0)$. So assume that such a nonzero entry is found. After permuting the rows we can assume wlog that this is $M_{1,1}$. Thus, we have found a 1×1 minor satisfying the requirements. Assume that we have found an $r \times r$ full rank minor that is composed of the first r rows and columns (we can always rearrange and hence it can be assumed wlog that they correspond to first r rows and columns). Denote this minor by M_r.

Now for every $(r + 1) \times (r + 1)$ submatrix of M contained in the first $r + 1$ columns and containing M_r, we check whether the determinant is 0 by randomized algorithm (Lemma 8). If any of these submatrices have nonzero determinant, then we pick one of them and call it M_{r+1}. Otherwise, we have found that first $r+1$ columns of M are linearly dependent. As M_r is full rank, there is $v \in \mathbb{F}(\overline{x})^r$ such that $M_r v = (M_{1,r+1}, \ldots, M_{r,r+1})^T$. This can be solved by applying Cramer's rule. The i-th entry of v is of the form $\det(M_r^{(i)})/\det(M_r)$, where $M_r^{(i)}$ is obtained by replacing i-th column of M_r with $(M_{1,r+1}, \ldots, M_{r,r+1})^T$. Observe that $\det(M_r)$, as well as $\det(M_r^{(i)})$, are both in $\mathbb{F}[\overline{x}]$.

Then it is immediate that $u := (\det(M_r^{(1)}), \ldots, \det(M_r^{(r)}), -\det(M_r), 0, \ldots, 0)^T$ is the desired vector.

To find M_r, each time we have to calculate the determinant and decide whether it is 0 or not. This is simply PIT for a determinant polynomial with entries of algebraic complexity $n^{O(\log n)}$ and degree $n^{O(1)}$. So, we have a comparable randomized algorithm for this. Determinant of a symbolic $n \times n$ matrix has $n^{O(\log n)}$ size formula (respectively poly(n) ABP) [MV97]. When the entries of the matrix have $n^{O(\log n)}$ size formula (respectively ABP), altogether, the determinant polynomial has the same algebraic complexity. There are $< n^2$ PIT invocations to test zeroness of the determinant. Altogether, we have a poly($n^{\log n}$)-time randomized algorithm for this (Lemma 8). □

Before moving to the constructive part, we discuss a new method for computing gcd of two polynomials, which not only fits well in the algorithm but is also of independent interest. We recall the definition of gcd of two polynomials f, g in the ring $\mathbb{F}[\overline{x}]$: $\gcd(f,g) =: h \iff h|f, h|g$ and $(h'|f, h'|g \Rightarrow h'|h)$. It is unique up to constant multiples.

Claim 5 (Computing formula gcd). Given two polynomials $f, g \in \mathbb{F}[\overline{x}]$ of degree d and computed by a formula (respectively ABP) of size s. One can compute a formula (respectively ABP) for $\gcd(f,g)$, of size poly$(s, d^{\log d})$, in randomized poly$(s, d^{\log d})$ time.

Proof of Claim 5. The idea is the following. Suppose, $\gcd(f,g) =: h$ is of degree $d > 0$, then we will compute $h(\tau\overline{x})$ for a random map τ as in Theorem 17. We know wlog that $\tilde{f} := f(\tau\overline{x}) = \prod_i (y - A_i)^{a_i}$ and $\tilde{g} := g(\tau\overline{x}) = \prod_i (y - B_i)^{b_i}$, where $A_i, B_i \in \mathbb{F}[[\overline{x}]]$. Since $\mathbb{F}[\overline{x}] \subset \mathbb{F}[[\overline{x}]]$ are UFDs (Proposition 16), we could say wlog that $h(\tau\overline{x}) = \prod_{i \in S}(y - A_i)^{\min(a_i, b_i)}$, where $S = \{i \mid A_i = B_i\}$ after possible rearrangement. Now, as τ is a random invertible map, we can assume that, for $i \neq j$, $A_i \neq B_j$ and that $A_i(\overline{0}) \neq B_j(\overline{0})$ (Lemma 13). So, it is enough to compute $A_i^{\leq d}$ and $B_j^{\leq d}$ and compare them using evaluation at $\overline{0}$. If indeed $A_i = B_i$, then $A_i^{\leq d} = B_i^{\leq d}$. If they are not, they mismatch at the constant term itself! Hence, we know the set S and so we are done once we have the power series roots with repetition.

Using univariate factoring, wrt y, we get all the multiplicities, of the roots, a_i and b_i, additionally, we get the corresponding starting points of classical Newton iteration, i.e. $A_i(\overline{0})$ and $B_i(\overline{0})$'s. Using NI, one can compute $A_i^{\leq d}$ and $B_i^{\leq d}$, for all i. Suppose, after rearrangement of A_i and B_i's (if necessary), we have $A_i = B_i$ for $i \in [s] =: S$ and $A_i \neq B_j$ for $i \in [s+1, d]$, $j \in [s+1, d]$. Lemma 13 can be used to deduce that $A_i(\overline{0}) \neq B_j(\overline{0})$ for $i, j \in [1, d] - S$. So, we have in $\gcd(\tilde{f}, \tilde{g}) = \prod_{i \in S}(y - A_i)^{\min(a_i, b_i)}$: the index set S, the exponents and $A_i(\overline{0})$'s computed.

Size analysis: To compute $A_i^{\leq d}$ (similarly $B_i^{\leq d}$), we use the classical newton iteration (Lemma 15) after differentiating (up to order to make multiplicity-1) to make A_i a *simple* root (i.e. multiplicity 1) of the differentiated polynomial. That leads to a polynomial blowup in size (Lemma 7). It is clear that at each NI step, there will be a multiplicative d^2 blow up (due to interpolation, division and truncation). There are $\log d$ iterations in NI. Altogether the truncated roots have poly$(s, d^{\log d})$ size formula (respectively ABP). This directly implies that $\gcd(\tilde{f}, \tilde{g})$ has poly$(s, d^{\log d})$ size formula (respectively ABP). By taking the product of the linear factors, truncating to degree d, and applying τ^{-1}, we can compute the polynomial $\gcd(f, g)$.

Randomization is needed for τ and possibly for the univariate factoring over $\mathbb{F}$. Also, it is important to note that $\mathbb{F}$ may not be algebraically closed. Then one has to go to an extension, do the algebraic operations and return to $\mathbb{F}$. For details, see Section 6.2. □

Randomized Algorithm. We give the broad s of our algorithm below. We are given $f \in \mathbb{F}[\overline{x}]$, of degree $d > 0$, as input.

(1) Choose $\overline{\alpha}, \overline{\beta} \in_r \mathbb{F}^n$ and apply $\tau : x_i \to x_i + \alpha_i y + \beta_i$. Denote the transformed polynomial $f(\tau\overline{x})$ by $\tilde{f}(\overline{x}, y)$. Wlog, from Theorem 17, $\tilde{f}$ has factorization of the form $\prod_{i=1}^{d_0}(y - g_i)^{\gamma_i}$, where $\mu_i := g_i(\overline{0})$ are distinct.
(2) Factorize $\tilde{f}(\overline{0}, y)$ over $\mathbb{F}[y]$. This will give γ_i and μ_i's.
(3) Fix $i = i_0$. Differentiate $\tilde{f}$, wrt y, $(\gamma_{i_0} - 1)$ many times to make g_{i_0} a simple root.
(4) Apply Newton iteration (NI), on the differentiated polynomial, for $k := \lceil \log(2d^2 + 1) \rceil$ iterations; starting with the approximation $\mu_{i_0} \pmod I$. We get $g_{i_0}^{<2^k}$ at the end of the process (mod I^{2^k}).

(5) Apply the transformation $x_i \mapsto Tx_i$ (T acts as a degree-counter). Consider $\tilde{g}_{i_0} := g_{i_0}^{<2^k}(T\overline{x})$. Solve the following homogeneous linear system of equations, over $\mathbb{F}[\overline{x}]$, in the unknowns u_{ij} and v_{ij}'s,

$$\sum_{0\le i+j<d} u_{ij}\cdot y^i T^j \;=\; (y-\tilde{g}_{i_0})\cdot \sum_{\substack{0\le i<d\\ 0\le j<2^k}} v_{ij}\cdot y^i T^j \mod T^{2^k}.$$

Solve this system, using Lemma 22, to get a nonzero polynomial (if one exists) $u := \sum_{0\le i+j<d} u_{ij}\cdot y^i T^j$.

(6) If there is no solution, return "f is irreducible".

(7) Otherwise, find the minimal solution wrt $\deg_y(u)$ by brute force (try all possible degrees wrt y; it is in $[d-1]$).

(8) Compute $G(\overline{x},y,T) := \gcd_y(u(\overline{x},y,T), \tilde{f}(T\overline{x},y))$ using Claim 5.

(9) Compute $G(\overline{x},y,1)$ and transform it by $\tau^{-1}: x_i \mapsto x_i - \alpha_i y - \beta_i$, $i\in[n]$, and $y\mapsto y$. Output this as an irreducible polynomial factor of f.

Claim 6 (Existence). If f is reducible, then the linear system (Step 5) has a non-trivial solution.

Proof of Claim 6. If f is reducible, then let $f = \prod f_i^{e_i}$ be its prime factorization. Assume wlog that $y - g_{i_0} \mid \tilde{f}_1 := f_1(\tau\overline{x})$. Of course $0 < \deg_y(\tilde{f}_1) = \deg(f_1) < d$.

Observe that we are done by picking u to be $\tilde{f}_1(T\overline{x},y)$. For, total degree of f_1 is $< d$, and so that of $\tilde{f}_1(T\overline{x},y)$ wrt the variables y,T is $< d$.

Moreover, $y - g_{i_0} \mid \tilde{f}_1 \implies \tilde{f}_1 = (y-g_{i_0})v$, for some $v\in\mathbb{F}[[\overline{x}]][y]$ with $\deg_y < d$. Hence,

$$\tilde{f}_1 \;\equiv\; (y - g_{i_0}^{<2^k})\cdot v \mod I^{2^k} \quad\implies\quad u \;\equiv\; (y-\tilde{g}_{i_0})\cdot v(T\overline{x},y) \mod T^{2^k}$$

This shows the existence of a nontrivial solution of the linear system (Step 5). □

Now, we show that if the linear system has a solution, then the solution corresponds to a non-trivial polynomial factor of f.

Claim 7 (Step 8's success). If the linear system (Step 5) has a non-trivial solution, then $0 < \deg_y G \le \deg_y u < d$.

Proof of Claim 7. Suppose (u,v) is the solution provided by the algorithm in Lemma 22 (u being in the unknown LHS and v being the unknown RHS). Consider $G = \gcd_y(u, \tilde{f}(Tx,y))$. We know that there are polynomials a and b such that $au + b\tilde{f}(Tx,y) = \mathrm{Res}_y(u,\tilde{f}(Tx,y))$ (Section 2.2). Consider $\deg_T(\mathrm{Res}_y(u,\tilde{f}(Tx,y))$. As the degree of T in u and $\tilde{f}(Tx,y)$ can be at most d, hence degree of T in Resultant can be atmost $2d^2$ (Section 2.2). Clearly, $\deg_y G \le \deg_y u < d$. If $\deg_y G = 0$ then the resultant of $u, \tilde{f}(T\overline{x},y)$ wrt y will be nonzero (Proposition 11). Suppose the latter happens.

Now, we have $u = (y-\tilde{g}_{i_0})v \mod T^{2^k}$. Since $y - g_{i_0} \mid \tilde{f}$ we get that $y - g_{i_0}(T\overline{x}) \mid \tilde{f}(T\overline{x},y)$. Assume that $\tilde{f}(Tx,y) =: (y - g_{i_0}(T\overline{x}))\cdot w$.

Thus, we can rewrite the previous equation as: $au + b\tilde{f}(T\overline{x},y) \equiv (y-\tilde{g}_{i_0})(av+bw) \equiv \mathrm{Res}_y(u,\tilde{f}(Tx,y)) \mod T^{2^k}$. Note that the latter is nonzero mod T^{2^k} because the resultant is a nonzero polynomial of $\deg_T < 2^k$. Putting $y = \tilde{g}_{i_0}$ the LHS vanishes, but RHS does not (because it is independent of y). This gives a contradiction.

Thus, $\mathrm{Res}_y(u,\tilde{f}(Tx,y) = 0$. This implies that $0 < \deg_y G < d$. □

Next we show that if one takes the minimal solution u (wrt degree of y), it will correspond to an irreducible factor of f. We will use the same notation as above.

Claim 8 (Irred. factor). Suppose $y - g_{i_0} \mid \tilde{f}_1$ and f_1 is an irreducible factor of f. Then, $G = c \cdot \tilde{f}_1(Tx, y)$, for $c \in \mathbb{F}^*$, and $\deg_y(G) = \deg_y(u) = \deg_y(f_1)$ in Step 8.

Proof of Claim 8. Suppose f is reducible, hence, as shown above, G is a non-trivial factor of $\tilde{f}(T\overline{x}, y)$. Recall that $\tilde{f}(T\overline{x}, y) = \prod_i (y - g_i(T\overline{x}))^{\gamma_i}$ is a factorization over $\mathbb{F}[[\overline{x}, T]]$. We have that $y - \tilde{g}_{i_0} \mid G \bmod T^{2^k}$. Thus, $y - g_{i_0}(T\overline{x}) \mid G$ absolutely (because the power series ring is a UFD and use Theorem 17). So, $y - g_{i_0}(T\overline{x}) \mid \gcd_y(G, \tilde{f}_1(T\overline{x}, y))$ over the power series ring. Since, $\tilde{f}_1(T\overline{x}, y)$ is an irreducible polynomial, we can deduce that $\tilde{f}_1(T\overline{x}, y) \mid G$ in the polynomial ring. So, $\deg_y(f_1) \leq \deg_y(G)$.

We have $\deg_y(\tilde{f}_1(T\overline{x}, y)) = \deg(f_1) =: d_1$. By the above discussion, the linear system in the Step 7 will not have a solution of $\deg_y(u)$ below d_1. Let us consider the linear system in the Step 7 that wants to find u of $\deg_y = d_1$. This system has a solution, namely the one with $u := \tilde{f}_1(T\overline{x}, y) \bmod T^{2^k}$. Then, by the above claim, we will get the G as well in the subsequent Step 8. This gives $\deg_y(G) \leq \deg_y(u) = d_1$. With the previous inequality, we get $\deg_y(G) = \deg_y(u) = \deg_y(f_1)$. In particular, G and $\tilde{f}_1(Tx, y)$ are the same up to a nonzero constant multiple. □

Alternative to Claim 5: The above proof (Claim 8) suggests that the gcd question of Step 8 is rather special: One can just write u as $\sum_{0 \leq i \leq d_1} c_i(\overline{x}, T) y^i$ and then compute the polynomial $G = \sum_{0 \leq i \leq d_1} (c_i / c_{d_1}) \cdot y^i$ as a formula (respectively ABP), by eliminating division (Lemma 5).

Once we have the polynomial G we can fix $T = 1$ and apply τ^{-1} to get back the irreducible polynomial factor f_1 (with power series root g_{i_0}).

The running time analysis of the algorithm is by now routine. If we start with an f computed by a formula (respectively ABP) of size $n^{O(\log n)}$, then as observed before, one can compute $\tilde{g}_{i_0}$ which has $n^{O(\log n)}$ size formula (respectively ABP). This takes care of s 1-4.

Now, solve the linear system in s 5-7 of the algorithm. Each entry of the matrix is a formula (respectively ABP) size $n^{O(\log n)}$. The time complexity is similar by invoking Lemma 22.

Step 8 is to compute gcd of two $n^{O(\log n)}$ size formulas (respectively ABPs) which again can be done in $n^{O(\log n)}$ time giving a size $n^{O(\log n)}$ formula (respectively ABP) as discussed above.

This completes the randomized poly$(n^{\log n})$-time algorithm that outputs $n^{O(\log n)}$ sized factors. □

Remarks.

(1) The above results hold true for the classes $VBP(s), VF(s), VNP(s)$ for any size function $s = n^{\Omega(\log n)}$.

(2) By using a *reversal* technique [Oli16, Section 1.1.2] and a modified τ, our size bound can be shown to be poly$(s, d^{\log r})$, where r (respectively d) is the individual-degree (respectively degree) bound of f. So, when r is constant, we get a factor as a poly(s)-size formula (respectively ABP). Oliveira [Oli16] proved the same result for formulas.

6 EXTENSIONS

6.1 Closure of approximative complexity classes

This section shows that all our closure under factoring results can be naturally generalized to corresponding approximative algebraic complexity classes.

In computer science, the notion of approximative algebraic complexity emerged in early works on matrix multiplication (the notion of border rank, see [BCS13]). It is also an important concept in the geometric complexity theory program (see [GMQ16]). The notion of approximative complexity can be motivated through two ways, *topological* and *algebraic* and both the perspectives are known to be equivalent. Both allow us to talk about the *convergence* $\epsilon \to 0$.

In what follows, we 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, Defn.2.1].

Definition 23 (Approximative closure of a class [BIZ18]). *Let C be an algebraic complexity class over field $\mathbb{F}$. A family (f_n) of polynomials from $\mathbb{F}[\overline{x}]$ 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 $g_n(\overline{x}) = f_n(\overline{x}) + \epsilon f_{n;1}(\overline{x}) + \epsilon^2 f_{n;2}(x) + \cdots + \epsilon^{t(n)} f_{n;t(n)}(\overline{x})$.*

The above definition can be used to define closures of classes like VF, VBP, VP, VNP which are denoted as $\overline{\mathrm{VF}}$, $\overline{\mathrm{VBP}}$, $\overline{\mathrm{VP}}$, $\overline{\mathrm{VNP}}$ respectively. In these cases one can assume wlog that the degrees of g_n and $f_{n;i}$ are $\mathrm{poly}(n)$.

Following Bürgisser [Bür04]:- Let $K := \mathbb{F}(\epsilon)$ be the rational function field in variable ϵ over the field $\mathbb{F}$. Let R denote the subring of K that consists of rational functions defined in $\epsilon = 0$. Eg. $1/\epsilon \notin R$ but $1/(1+\epsilon) \in R$.

Definition 24. *[Bür04, Defn.3.1] Let $f \in \mathbb{F}[x_1, \ldots, x_n]$. The* approximative complexity $\overline{size}(f)$ *is the smallest number r, such that there exists F in $R[x_1, \ldots, x_n]$ satisfying $F|_{\epsilon=0} = f$ and circuit size of F over* constants K *is $\leq r$.*

Note that the circuit of F may be using division by ϵ implicitly in an intermediate step. So, we cannot merely assign $\epsilon = 0$ and get a circuit free of ϵ. Also, the degree involved can be arbitrarily large wrt ϵ. Thus, potentially $\overline{\mathrm{size}}(f)$ can be smaller than $\mathrm{size}(f)$.

Using this new notion of size, one can define the analogous class $\overline{\mathrm{VP}}$. It is known to be closed under factors [Bür04, Theorem 4.1]. The idea is to work over $\mathbb{F}(\epsilon)$, instead of working over $\mathbb{F}$, and use Newton iteration to approximate power series roots. Note that in the case of $\overline{\mathrm{VF}}$, $\overline{\mathrm{VBP}}$, $\overline{\mathrm{VP}}$ and $\overline{\mathrm{VNP}}$ the polynomials have $\mathrm{poly}(n)$ degree. So, by using repeated differentiation, we can assume the power series root (of $\tilde{f} := f(\tau\overline{x})$) to be simple (i.e. multiplicity= 1) and apply classical NI. Here we give a brief sketch of the overall idea.

Root finding using NI over K. For degree-d $f \in \mathbb{F}[\overline{x}]$ if $\overline{\mathrm{size}}(f) = s$ then: $\exists F \in R[\overline{x}]$ with a size s circuit satisfying $F|_{\epsilon=0} = f$. The degree of F wrt $\overline{x}$ may be greater than d. In that case, we can extract the part up to degree d and truncate the rest [Bür04, Prop.3.1]. So wlog $\deg_{\overline{x}}(F) = \deg(f)$.

By applying a random τ (using constants $\mathbb{F}$) we can assume that $\tilde{F} := F(\tau\overline{x}) \in R[\overline{x}, y]$ is *monic* (i.e. leading-coefficient, wrt y in $\tilde{F}$, is invertible in R). Otherwise, $\deg_y(\tilde{F}) = \deg_y(\tilde{f}) = \deg_{\overline{x}}(f)$ will decrease on substituting $\epsilon = 0$ contradicting $F|_{\epsilon=0} = f$. Wlog, we can assume that the leading-coefficient of $\tilde{F}$ wrt y is 1 and the y-monomial's degree is d. From now on we have $\tilde{F}|_{\epsilon=0} = \tilde{f}$ and both have their leading-coefficients 1 wrt y.

Let μ be a root of $\tilde{f}(\overline{0}, y)$ of multiplicity one (as discussed before). Since $\tilde{F}(\overline{0}, y) \equiv \tilde{f}(\overline{0}, y) \bmod \epsilon$, we can build a power series root $\mu(\epsilon) \in \mathbb{F}[[\epsilon]]$ of $\tilde{F}(\overline{0}, y)$ using NI, with μ as the starting point. But $\mu(\epsilon)$ may not converge in K. To overcome this obstruction, [Bür04] devised a clever trick.

Define $\hat{F} := \tilde{F}(\overline{x}, y + \mu + \epsilon) - \tilde{F}(\overline{0}, \mu + \epsilon)$. Note that $(\overline{0}, 0)$ is a simple root of $\hat{F}(\overline{x}, y)$ [Bür04, Eqn.5]. So, a power series root y_∞ of $\hat{F}$ can be built iteratively by classic NI (Lemma 15):

$$y_{t+1} \ := \ y_t \ - \ \left.\frac{\hat{F}}{\partial_y \hat{F}}\right|_{y=y_t} .$$

Where, $y_\infty \equiv y_t \mod \langle \bar{x} \rangle^{2^t}$. One can easily prove that y_t is defined over the coefficient field K, using induction on t.

Note that $\hat{F}|_{\epsilon=0} = \tilde{f}(\overline{x}, y + \mu) - \tilde{f}(\overline{0}, \mu) = \tilde{f}(\overline{x}, y + \mu)$. So, y_∞ is associated with a root of $\tilde{f}$ as well. This implies that by using several such roots y_∞, we can get an appropriate product $\hat{G} \in R[\overline{x}, y]$, such that an actual polynomial factor of $\tilde{f}$ (over field $\mathbb{F}$) equals $\hat{G}|_{\epsilon=0}$.

The above process, when combined with the first part of the proof of Theorem 3, does imply:

Theorem 25 (Approximative factors). *The approximative complexity classes* $\overline{VF}(n^{\log n})$, $\overline{VBP}(n^{\log n})$ *and* $\overline{VNP}(n^{\log n})$ *are closed under factors.*

The same question for the classes $\overline{\mathrm{VF}}$, $\overline{\mathrm{VBP}}$ and $\overline{\mathrm{VNP}}$ we leave as an open question. (Though, for the respective bounded individual-degree polynomials we have the result as before.)

6.2 When field $\mathbb{F}$ is not algebraically closed

We show that all our results "partially" hold true for fields $\mathbb{F}$ which are not algebraically closed. The standard technique used in all the proofs is the structural result (Theorem 17) which talks about power series roots with respect to y. Recall that we use a random linear map $\tau : x_i \mapsto x_i + \alpha_i y + \beta_i$, where $\alpha_i, \beta_i \in_r \mathbb{F}$, to make the input polynomial f monic in y and the individual degree of y equal to $d := \deg(f)$. If we set all the variables to zero except y, we get a univariate polynomial $\tilde{f}(\overline{0}, y)$ whose roots we are interested in finding explicitly.

The other common technique in our proofs is the classical NI, which starts with just one field root, say μ_1 of $\tilde{f}(\overline{0}, y)$, and builds the full power series on it. Let $E \subsetneq \overline{\mathbb{F}}$ be the smallest field where a root μ_1 can be found. Say, $g | \tilde{f_1}(\overline{0}, y)$ is the minimal polynomial for μ_1. The degree of the extension $E := \mathbb{F}[z]/(g(z))$ is at most d. So, computations over E can be done efficiently. The key idea is to view $E/\mathbb{F}$ as a vector space and simulate the arithmetic operations over E by operations over $\mathbb{F}$. The details of this kind of simulation can be seen in [vzGG13]. In circuits, it means that we make $\deg(E/\mathbb{F})$ copies of each gate and simulate the algebraic operations on these 'tuples' following the $\mathbb{F}$-module structure of $E[\overline{x}]$.

Once we have found all the power series roots of $\tilde{f}(\overline{x}, y)$ over $E[[\overline{x}]]$, say starting from each of the conjugates $\mu_1, \ldots, \mu_i \in E$, it is easy to get a polynomial factor in $E[\overline{x}, y]$. This factor will not be in $\mathbb{F}[\overline{x}, y]$, unless E is a splitting field of $\tilde{f_1}(\overline{0}, y)$. A more practical method is: While solving the linear system over E in s 5-7 (Algorithm in Theorem 3) we can demand an $\mathbb{F}$-solution u. Basically, at the level of the algorithm in Lemma 22, we can rewrite the linear system $Mw = (\sum_{0 \le i \le d} M_i z^i) \cdot w = 0$ as $M_i w = 0$ $(i \in [0, d])$, where the entries of the matrix M_i are given as formulas (respectively ABP) computing a poly(n) degree polynomial in $\mathbb{F}[\overline{x}]$. This way we get the desired $\mathbb{F}$-solution u. Then, s 8-9 will yield an irreducible polynomial factor of f in $\mathbb{F}[\overline{x}, y]$. This sketches the following more practical version of Theorem 3.

Theorem 26. *For $\mathbb{F}$ a number field, a local field, or a finite field (with characteristic $> \deg(f)$), there exists a randomized poly$(sn^{\log n})$-time algorithm that: for a given $n^{O(\log n)}$ size formula (respectively ABP) f of poly(n)-degree and bitsize s, outputs $n^{O(\log n)}$ sized formulas (respectively ABPs) corresponding to each of the nontrivial factors of f.*

Note that over these fields there are famous randomized algorithms to factor univariate polynomials in the base case, see [vzGG13, Part III] & [Pau01]. See [Gao03] for multivariate factoring over the algebraic closure of a field.

The allRootsNI method in Theorem 1 seems to require all the roots $\mu_i, i \in [d_0]$, to begin with. Let $\tilde{u}_1 := \mathrm{rad}(u_1(\tau\overline{x}))$. Since μ_i's are in the *splitting field* $E \subset \overline{\mathbb{F}}$ of $\mathrm{rad}(\tilde{u}_1(\overline{0}, y))$, we do indeed get the size bound of the power series roots $g_i^{\le d_0}$ of $\tilde{u}_1$ assuming the constants from E. As seen in the proof, any irreducible polynomial factor $\tilde{h}_i := h_i(\tau\overline{x})$ of $\mathrm{rad}(\tilde{u}_1)$ is some product of these $(y - g_i^{\le d_0})$'s mod I^{d_0+1}. So, for the polynomial $\tilde{h}_i$ in $\mathbb{F}[\overline{x}, y]$ we get a size upper bound over constants E. We leave it as an open question to transfer it over constants $\mathbb{F}$ (note: $E/\mathbb{F}$ can be of exponential degree).

6.3 Multiplicity issue in prime characteristic

The main obstruction in prime characteristic is when the multiplicity of a factor is a p-multiple, where $p \geq 2$ is the characteristic of $\mathbb{F}$. In this case, all versions of the Newton iteration fail. This is because the derivative of a p-powered polynomial vanishes. When p is greater than the degree of the input polynomial, these problems do not occur, so all our theorems hold (also see Section 6.2).

When p is smaller than the degree of the input polynomial in Theorem 3, adapting an idea from [KSS15, Section 3.1], we claim that we can give $n^{O(\lambda \log n)}$-sized formula (respectively ABP) for the p^{e_i}-th power of f_i, where f_i is a factor of f whose multiplicity is divisible exactly by p^{e_i}, and λ is the number of distinct p-powers that appear.

Note that presently it is an open question to show that: If a circuit (respectively formula respectively ABP) of size s computes f^p, then f has a poly(sp)-sized circuit (respectively formula respectively ABP).

Theorem 3 can be extended to all characteristics as follows.

Theorem 27. *Let $\mathbb{F}$ be of characteristic $p \geq 2$. Suppose the poly(n)-degree polynomial given by a $n^{O(\log n)}$ size formula (respectively ABP) factors into irreducibles as $f(\overline{x}) = \prod_i f_i^{p^{e_i} j_i}$, where $p \nmid j_i$. Let $\lambda := \#\{e_i | i\}$.*

Then, there is a poly($n^{\lambda \log n}$)-size formula (respectively ABP) computing $f_i^{p^{e_i}}$ over $\overline{\mathbb{F}}_p$.

Proof sketch. Note that $\lambda = O(\log_p n)$.

Let the transformed polynomial of degree d split into power series roots as follows: $\tilde{f} := f(\tau\overline{x}, y) = \prod_{i=1}^{d_0} (y - g_i)^{\gamma_i}$.

$p \nmid \gamma_i$: If g_i is such that $p \nmid \gamma_i$, then we can find the corresponding power series roots using Newton iteration and recover all such factors. After recovering all such irreducible polynomial factors, we can divide $\tilde{f}$ by their product. Let $G := \tilde{f} / \prod_{p \nmid \gamma_i} (y - g_i)^{\gamma_i}$. Clearly, G is now a p-power polynomial.

$p \mid \gamma_i$: Computing the highest power of p that divides the exponent of G (given by a formula respectively ABP) is easy. First, write the polynomial as $G = c_0 + c_1 y + \cdots + c_d y^d$ using interpolation. Note that it is a p^e-th power iff: $c_i = 0$ whenever $p^e \nmid i$, and p^{e+1} does not have this property. After computing the right value of p^e, we can reduce factoring to the case of a non-p-power.

Rewrite G as $\hat{G} := \sum_{p^e | i} c_i(\overline{x}) \cdot y^{i/p^e}$, i.e. replacing y^{p^e} by y. Clearly, g is an irreducible factor of G iff $\hat{g}$ is an irreducible factor of $\hat{G}$.

We can now apply NI to find the roots of $\tilde{G}$, that have multiplicity coprime to p. Divide by their product and then repeat the above.

Size analysis. If G can be computed by a size s formula (respectively ABP), $\hat{G}$ can be computed by a size $O(d^2 s)$ formula (respectively ABP). Similarly, a single division gate leads to a blow-up by a factor of $O(d^2)$. The number of times we need to eliminate division is at most $\lambda \log d$. So the overall size is $n^{O(\lambda \log n)}$.

However, the splitting field E where we get all the roots of $\tilde{f}(\overline{0}, y)$ may be of degree $\Omega(d!)$. So, we leave the efficiency aspects of the algorithm as an open question. ◻

High degree case. Note that the above idea cannot be implemented efficiently in the case of high degree circuits. Still we can extend our Theorem 1 using allRootsNI. The key observation is that the allRootsNI formula still holds but the summands that appear are exactly the ones corresponding to g_i with $\gamma_i \neq 0 \bmod p$.

This motivates the definition of a partial radical: $\mathrm{rad}_p(f) := \prod_{p \nmid e_i} f_i$, if the prime factorization of f is $\prod_i f_i^{e_i}$.

Theorem 28. *Let $\mathbb{F}$ be of characteristic $p \geq 2$. Let $f = u_0 u_1$ such that size(f)+size(u_0) $\leq s$. Any factor of $rad_p(u_1)$ has size poly($s + \deg(rad_p(u_1))$) over $\overline{\mathbb{F}}$.*

Proof idea: Observe that the roots with multiplicity divisible by p do not contribute to the allRootsNI process. So, the process works with $\mathrm{rad}_p(u_1)$, and the linear algebra complexity involved is polynomial in its degree.

7 CONCLUSION

We analyzed the complexity of approximating power series roots (up to some degree) of a multivariate polynomial in various models. As the roots are related to factors, we get results on polynomial factoring in various models as well.

Finally, we list a few open questions related to multivariate factoring.

(1) The Factor Conjecture states that for a nonzero polynomial f: $g \mid f \implies \mathrm{size}(g) \le \mathrm{poly}(\mathrm{size}(f), \deg(g))$. Motivated by Theorem 1, we would like to strengthen the Factor conjecture to a conjecture about the squarefree part of a circuit:

Conjecture 1 (Radical Conjecture). For a nonzero f: $\min\{\deg(\mathrm{rad}(f)), \mathrm{size}(\mathrm{rad}(f))\} \le \mathrm{poly}(\mathrm{size}(f))$.

Is the above conjecture true if we replace size by $\overline{\mathrm{size}}$?

(2) Suppose we have a circuit with division gates computing a polynomial of degree d. Can we get a circuit of size $\mathrm{poly}(s)$ computing the same polynomial without using any division gate [Kal87]? A positive answer to this question would prove the Factor conjecture as a corollary. Strassen's classic result gives a $\mathrm{poly}(s, d)$ bound for this problem. Recently, [DJPS21] showed that division elimination can be efficiently done if the divisor is a *low* degree polynomial (computed by a small circuit).

(3) Given two polynomials computed by circuits of size s, can we get a circuit computing their gcd in size $\mathrm{poly}(s)$? Kaltofen [Kal87] gave $\mathrm{poly}(s, d_g)$ size upper bound, where d_g is the degree of the gcd.

(4) Is VF closed under factoring? We might consider Theorem 3 as a positive evidence. Additionally, a special case when $f = g^e$ for some irreducible polynomial g, can be solved. This is easy to see using the classic Taylor series of $(1 + f)^{1/e}$, where $f \in \langle \overline{x} \rangle$.

In fact, what about the classes which are contained in $VF(n^{\log n})$ but larger than VF. For example, is $\mathrm{VF}(n^{\log \log n})$ closed under factoring?

(5) Can we compute factors of polynomials computed by circuits of low depth by keeping the depth constant and the size small [DSY09, Oli16]? To further motivate this question, we mention the recent breakthrough by Limaye, Srinivasan and Tavenas [LST21] where the authors gave the *first* superpolynomial lower bound against constant depth circuits. They also gave the first subexponential PIT for the same model using an arithmetic hardness vs randomness result from [CKS19b]. The work of [CKS19b] needed a size upper bound of roots of low depth polynomials (different from the related prior work by [DSY09]). The application of [CKS19b] in [LST21] shows the importance of studying factoring and root finding for restricted classes to settle fundamental questions in algebraic complexity.

Finally, our results weaken when the underlying field $\mathbb{F}$ is not algebraically closed or has a small prime characteristic (Sections 6.2, 6.3). Can we strengthen the methods to work for all $\mathbb{F}$?

Acknowledgements. We thank Rafael Oliveira for extensive discussions regarding his works and circuit factoring in general. In particular, we used his suggestions about VNP and $\overline{\mathrm{VP}}$ in our results. We thank Manindra Agrawal, Sumanta Ghosh, Partha Mukhopadhyay, Thomas Thierauf, and Nikhil Balaji for helpful discussions. We are grateful to the organizers of WACT'16 (Tel Aviv, Israel) and Dagstuhl'16 (Germany) for the stimulating workshops. P.D. would like to thank CSE, IIT Kanpur for the hospitality, Google India Research Program Team for the support of all travel expenses and PhD Fellowship. N.S. thanks for the funding support from DST (DST/SJF/MSA-01/2013-14), DST-SERB (CRG/2020/000045) and N. Rama Rao Chair. A.S. would like to thank Microsoft Research Lab India, IARCS, and ACM India for supporting with travel grants for conferences. Previously, A.S. was supported by MHRD (Govt of India) graduate student fellowship. Currently, he is supported by DFG grant TH 472/5-1.

REFERENCES

[Abe73] Oliver Aberth. Iteration methods for finding all zeros of a polynomial simultaneously. *Mathematics of computation*, 27(122):339–344, 1973.

[ABK+21] Mohammadali Asadi, Alexander Brandt, Mahsa Kazemi, Marc Moreno Maza, and Erik Postma. Multivariate power series in maple. *arXiv preprint arXiv:2106.15519*, 2021.

[AGS19] Manindra Agrawal, Sumanta Ghosh, and Nitin Saxena. Bootstrapping variables in algebraic circuits. *Proceedings of the National Academy of Sciences*, 116(17):8107–8118, 2019.

[AP00] Daniel Augot and Lancelot Pecquet. A hensel lifting to replace factorization in list-decoding of algebraic-geometric and reed-solomon codes. *IEEE Transactions on Information Theory*, 46(7):2605–2614, 2000.

[AV08] Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 67–75. IEEE, 2008.

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

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer Science & Business Media, 1998.

[Ber70] Elwyn R Berlekamp. Factoring polynomials over large finite fields. *Mathematics of computation*, 24(111):713–735, 1970.

[BIZ18] Karl Bringmann, Christian Ikenmeyer, and Jeroen Zuiddam. On algebraic branching programs of small width. *J. ACM*, 65(5), 2018. (Preliminary version in CCC'17).

[BJ18] Markus Bläser and Gorav Jindal. On the complexity of symmetric polynomials. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[BK78] Richard P Brent and Hsiang T Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM (JACM)*, 25(4):581–595, 1978.

[BOC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[Bou13] Nicolas Bourbaki. *Algebra II: Chapters 4-7*. Springer Science & Business Media, 2013.

[BSCI+20] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed–solomon codes. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2020. https://eccc.weizmann.ac.il/report/2020/083/.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989.

[BSV20] Vishwas Bhargava, Shubhangi Saraf, and Ilya Volkovich. Deterministic factorization of sparse polynomials with bounded individual degree. *J. ACM*, 67(2), May 2020.

[Bür01] Peter Bürgisser. On implications between P-NP-hypotheses: Decision versus computation in algebraic complexity. In *MFCS*, pages 3–17. Springer, 2001.

[Bür04] Peter Bürgisser. The complexity of factors of multivariate polynomials. *Foundations of Computational Mathematics*, 4(4):369–396, 2004. (Preliminary version in FOCS 2001).

[Bür13] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Science & Business Media, 2013.

[CG00] David G Cantor and Daniel M Gordon. Factoring polynomials over ρ-adic fields. In *International Algorithmic Number Theory Symposium*, pages 185–208. Springer, 2000.

[Chi94] Alexander L Chistov. Algorithm of polynomial complexity for factoring polynomials over local fields. *Journal of mathematical sciences*, 70(4):1912–1933, 1994.

[CKS19a] Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. Closure of vp under taking factors: a short and simple proof. *arXiv preprint arXiv:1903.02366*, 2019.

[CKS19b] Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. Closure results for polynomial factorization. *Theory of Computing*, 15(13):1–34, 2019.

[CRS96] Richard Courant, Herbert Robbins, and Ian Stewart. *What is Mathematics?: an elementary approach to ideas and methods*. Oxford University Press, USA, 1996.

[CZ81] David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981.

[DB08] Germund Dahlquist and Åke Björck. Numerical methods in scientific computing, volume I. *Society for Industrial and Applied Mathematics*, 2008.

[DDS21a] Pranjal Dutta, Prateek Dwivedi, and Nitin Saxena. Demystifying the border of depth-3 algebraic circuits. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2021)*, 2021.

[DDS21b] Pranjal Dutta, Prateek Dwivedi, and Nitin Saxena. Deterministic identity testing paradigms for bounded top-fanin depth-4 circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 11:1–11:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[DJPS21] Pranjal Dutta, Gorav Jindal, Anurag Pandey, and Amit Sinhababu. Arithmetic circuit complexity of division and truncation. In *36th Computational Complexity Conference (CCC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193 – 195, 1978.

[DMS19] Ashish Dwivedi, Rajat Mittal, and Nitin Saxena. Efficiently factoring polynomials modulo p^4. *arxiv preprint arXiv:1901.06628*, 2019. To appear in ISSAC'19.

[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)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[DSY09] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM Journal on Computing*, 39(4):1279–1293, 2009. (Preliminary version in STOC'08).

[Dur60] Émile Durand. Solutions numériques des équations algébriques. Tome I, Équations du type F(x)= 0. In *Racines d'une Polynôme*, pages 279–281. Masson, 1960.

[Dut21] Pranjal Dutta. Real τ-conjecture for sum-of-squares: A unified approach to lower bound and derandomization. In *International Computer Science Symposium in Russia*, pages 78–101. Springer, 2021.

[Ehr67] Louis W Ehrlich. A modified Newton method for polynomials. *Communications of the ACM*, 10(2):107–108, 1967.

[FS15] Michael A Forbes and Amir Shpilka. Complexity theory column 88: Challenges in polynomial factorization. *ACM SIGACT News*, 46(4):32–49, 2015.

[FSTW16] Michael A Forbes, Amir Shpilka, Iddo Tzameret, and Avi Wigderson. Proof complexity lower bounds from algebraic circuit complexity. In *Proceedings of the 31st Conference on Computational Complexity*, page 32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[Gao03] Shuhong Gao. Factoring multivariate polynomials via partial differential equations. *Mathematics of computation*, 72(242):801–822, 2003.

[GHM+98] M Giusti, J Heintz, JE Morais, J Morgenstem, and LM Pardo. Straight-line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124(1-3):101–146, 1998.

[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)*, volume 55, pages 34:1–34:14, 2016.

[Gre16] Bruno Grenet. Bounded-degree factors of lacunary multivariate polynomials. *Journal of Symbolic Computation*, 75:171–192, 2016.

[Gro15] Joshua A Grochow. Unifying known lower bounds via geometric complexity theory. *Computational Complexity*, 24(2):393–475, 2015.

[Gro20] Joshua A Grochow. Complexity in ideals of polynomials: questions on algebraic complexity of circuits and proofs. *Bulletin of EATCS*, 1(130), 2020.

[GS98] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 28–37. IEEE, 1998.

[GTZ88] Patrizia Gianni, Barry Trager, and Gail Zacharias. Gröbner bases and primary decomposition of polynomial ideals. *Journal of Symbolic Computation*, 6(2):149–167, 1988.

[IKRS12] Gábor Ivanyos, Marek Karpinski, Lajos Rónyai, and Nitin Saxena. Trading GRH for algebra: algorithms for factoring polynomials and related structures. *Mathematics of Computation*, 81(277):493–531, 2012.

[Jan11] Maurice J Jansen. Extracting roots of arithmetic circuits by adapting numerical methods. In *2nd Symposium on Innovations in Computer Science (ICS 2011)*, pages 87–100, 2011.

[Kal85a] Erich Kaltofen. Computing with polynomials given by straight-line programs I: greatest common divisors. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 131–142, 1985.

[Kal85b] Erich Kaltofen. Polynomial-time reductions from multivariate to bi-and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

[Kal86] Erich Kaltofen. Uniform closure properties of p-computable functions. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 330–337, 1986.

[Kal87] Erich Kaltofen. Single-factor hensel lifting and its application to the straight-line complexity of certain polynomials. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 443–452. ACM, 1987.

[Kal89] Erich Kaltofen. Factorization of polynomials given by straight-line programs. *Randomness and Computation*, 5:375–412, 1989.

[Kal90] Erich Kaltofen. Polynomial factorization 1982-1986. *Dept. of Comp. Sci. Report*, pages 86–19, 1990.

[Kal92] Erich Kaltofen. Polynomial factorization 1987–1991. *LATIN'92*, pages 294–313, 1992.

[Kay11] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1409–1421. Society for Industrial and Applied Mathematics, 2011.

[Kem10] Gregor Kemper. *A course in Commutative Algebra*, volume 256. Springer Science & Business Media, 2010.

[Ker66] Immo O Kerner. Ein Gesamtschrittverfahren zur Berechnung der Nullstellen von Polynomen. *Numerische Mathematik*, 8(3):290–294, 1966.

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 355–364. ACM, 2003.

[KK08] Erich Kaltofen and Pascal Koiran. Expressing a fraction of two determinants as a determinant. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 141–146. ACM, 2008.

[KP12] Steven G Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012.

[Kri02] Teresa Krick. Straight-line programs in polynomial equation solving. *Foundations of computational mathematics: Minneapolis*, 312:96–136, 2002.

[KS06] Neeraj Kayal and Nitin Saxena. Complexity of ring morphism problems. *Computational Complexity*, 15(4):342–390, 2006.

[KS09] Zohar S Karnin and Amir Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Computational Complexity, 2009. CCC'09. 24th Annual IEEE Conference on*, pages 274–285. IEEE, 2009.

[KS16] Mrinal Kumar and Shubhangi Saraf. Arithmetic circuits with locally low algebraic rank. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 34:1–34:27, 2016.

[KSS15] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and polynomial factorization. *computational complexity*, 24(2):295–331, 2015.

[KST19] Mrinal Kumar, Ramprasad Saptharishi, and Anamay Tengse. Near-optimal bootstrapping of hitting sets for algebraic circuits. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 639–646. SIAM, 2019.

[KT78] HT Kung and Joseph Frederick Traub. All algebraic functions can be computed fast. *Journal of the ACM (JACM)*, 25(2):245–260, 1978.

[Lan85] Susan Landau. Factoring polynomials over algebraic number fields. *SIAM Journal on Computing*, 14(1):184–195, 1985.

[Lec02] Grégoire Lecerf. Quadratic newton iteration for systems with multiplicity. *Foundations of Computational Mathematics*, 2(3):247–293, 2002.

[Len83] Arjen K Lenstra. Factoring polynomials over algebraic number fields. In *European Conference on Computer Algebra*, pages 245–254. Springer, 1983.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LLMP90] Arjen K Lenstra, Hendrik W Lenstra, Mark S Manasse, and John M Pollard. The number field sieve. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 564–572. ACM, 1990.

[LN97] Rudolph Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, Cambridge, UK, 1997.

[LS78] Richard J Lipton and Larry J Stockmeyer. Evaluation of polynomials with super-preconditioning. *Journal of Computer and System Sciences*, 16(2):124–139, 1978.

[LST21] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial lower bounds against low-depth algebraic circuits. *Electron. Colloquium Comput. Complex.*, 28:81, 2021.

[LV16] Anand Louis and Santosh Srinivas Vempala. Accelerated newton iteration for roots of black box polynomials. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS'16*, pages 732–740, 2016.

[Mah14] Meena Mahajan. Algebraic complexity classes. In *Perspectives in Computational Complexity*, pages 51–75. Springer, 2014.

[Mul12a] Ketan D. Mulmuley. The GCT program toward the P vs. NP problem. *Commun. ACM*, 55(6):98–107, June 2012.

[Mul12b] Ketan D. Mulmuley. Geometric complexity theory V: Equivalence between blackbox derandomization of polynomial identity testing and derandomization of Noether's normalization lemma. In *FOCS*, pages 629–638, 2012.

[Mul17] Ketan Mulmuley. Geometric complexity theory V: Efficient algorithms for Noether normalization. *Journal of the American Mathematical Society*, 30(1):225–309, 2017.

[MV97] Meena Mahajan and V Vinay. A combinatorial algorithm for the determinant. In *SODA*, pages 730–738, 1997.

[New69] Isaac Newton. De analysi per aequationes numero terminorum infinitas [On analysis by infinite series] (in latin). 1669. (published in 1711 by William Jones).

[NRS17] Vincent Neiger, Johan Rosenkilde, and Éric Schost. Fast computation of the roots of polynomials over the ring of power series. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 349–356, 2017.

[Oli16] Rafael Oliveira. Factors of low individual degree polynomials. *Computational Complexity*, 2(25):507–561, 2016. (Preliminary version in CCC'15).

[Ore22] ystein Ore. Über höhere kongruenzen. *Norsk Mat. Forenings Skrifter*, 1(7):15, 1922.

[Pau01] Sebastian Pauli. Factoring polynomials over local fields. *Journal of Symbolic Computation*, 32(5):533–547, 2001.

[Pla77] David Alan Plaisted. Sparse complex polynomials and polynomial reducibility. *Journal of Computer and System Sciences*, 14(2):210–221, 1977.

[PSS16] Anurag Pandey, Nitin Saxena, and Amit Sinhababu. Algebraic independence over positive characteristic: New criterion and applications to locally low algebraic rank circuits. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 74:1–74:15, 2016. (In print, Computational Complexity, 2018).

[Sap19] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2019. https://github.com/dasarpmar/lowerbounds-survey/releases.

[Sch77] Claus-Peter Schnorr. Improved lower bounds on the number of multiplications/divisions which are necessary to evaluate polynomials. In *International Symposium on Mathematical Foundations of Computer Science*, pages 135–147. Springer, 1977.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[Sch82] Arnold Schönhage. The fundamental theorem of algebra in terms of computational complexity. *Manuscript. Univ. of Tübingen, Germany*, 1982.

[Sin16] Gaurav Sinha. Reconstruction of real depth-3 circuits with top fan-in 2. In *31st Conference on Computational Complexity*, 2016.

[SS93] Tateaki Sasaki and Mutsuko Sasaki. A unified method for multivariate polynomial factorizations. *Japan journal of industrial and applied mathematics*, 10(1):21–39, 1993.
[ST20] Amit Sinhababu and Thomas Thierauf. Factorization of polynomials given by arithmetic branching programs. In *35th Computational Complexity Conference (CCC 2020)*, 2020. https://eccc.weizmann.ac.il/report/2020/077/.
[Str73] Volker Strassen. Vermeidung von divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.
[Sud97] Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *Journal of complexity*, 13(1):180–193, 1997.
[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010.
[Tay15] Brook Taylor. Methodus incrementorum directa et inversa [direct and reverse methods of incrementation] (in latin). 1715. (Translated into English in Struik, D. J. (1969). A Source Book in Mathematics 1200–1800. Cambridge, Massachusetts: Harvard University Press. pp. 329–332.).
[Val79] Leslie G. Valiant. Completeness classes in algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261, 1979.
[VL97] Paul MB Vitanyi and Ming Li. An introduction to kolmogorov complexity and its applications. 34(10), 1997.
[VSBR83] Leslie G. Valiant, Sven Skyum, Stuart Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.
[VzG84] Joachim Von zur Gathen. Hensel and newton methods in valuation rings. *Mathematics of Computation*, 42(166):637–661, 1984.
[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.
[vZGH96] Joachim von Zur Gathen and Silke Hartlieb. *Factorization of polynomials modulo small prime powers*. Univ.-Gesamthochsch.-Paderborn, Fachbereich Mathematik-Informatik, 1996.
[vzGK85] Joachim von zur Gathen and Erich Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265–287, 1985.
[Wik] Wikipedia. Cauchy matrix. https://en.wikipedia.org/wiki/Cauchy_matrix.
[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, 1979.
[ZS75] Oscar Zariski and Pierre Samuel. *Commutative algebra. II. Reprint of the 1960 edition*, volume 29. Graduate Texts in Mathematics, 1975.

A NUMERICAL ANALOG OF THEOREM 1 : PROOF OF CLAIM 1

Claim 1 (restated). *For each root $a \in (0,1)$ of $f(x)$, there is some 2^m-bit approximation a' such that bitsize$(a') \leq O((s+m)\cdot\log(\frac{1}{\epsilon}))$, where bitsize$(f) =: s$, and $\epsilon \in (0,1)$ lower bounds the gap between a and the other roots of $f(x)$.*

Proof sketch– Now we give the main s in the numerical result (the algebraic analog Theorem 1 would be more involved to prove as it is stronger). We use general Newton iteration (or, NI with multiplicity). Observe that, we are interested in an existential statement; so, assume that we know the multiplicity γ of the root a before-hand. Suppose, $f(x) = (x-a)^\gamma g(x)$ where $g(a) \neq 0$. Assume that $g(x) = \prod_{i=1}^{r}(x-b_i)^{\gamma_i}$.

Hypothesis says that $|a-b_i| \geq \epsilon$, for all i. We will use general NI to approximate 2^m bits of a (in this case after the decimal-point only). We will start with y_0 such that $|y_0 - a| \leq \epsilon \cdot 2^{-3(m+s)-1}$. Thus, to start with, y_0 has a trivial circuit of bitsize $O((m+s)\log(\frac{1}{\epsilon}))$.

Next, we use the general NI formula [DB08, Eqn.6.3.13], i.e.

$$y_{t+1} \;=\; y_t \;-\; \gamma \cdot \left.\frac{f}{f'}\right|_{y_t} .$$

Finally, we need to show that the process has *quadratic* convergence. Inductively, we want to show that for all $t \geq 0$,

$$|y_t - a| \;\leq\; \epsilon \cdot 2^{-3(m+s)} \cdot 2^{-2^t} . \tag{5}$$

Note that the above inequality implies that y_m is a 2^m-bit approximation of a. Moreover, computing y_{t+1} as a circuit– given the value y_t and circuits for $f(x), f'(x)$ –requires $O(s)$ additional bitsize (remember $\div$ is allowed in the circuit). So, y_m will have a circuit of bitsize $O((s+m)\log(\frac{1}{\epsilon}))$.

We are only left to prove Equation 5. We have,

$$\begin{aligned}\frac{f(y_t)}{f'(y_t)} &= \frac{(y_t-a)^{\gamma}g(y_t)}{(y_t-a)^{\gamma-1}\cdot(\gamma g(y_t)+(y_t-a)g'(y_t))}\\ &= \frac{(y_t-a)g(y_t)}{\gamma g(y_t)+(y_t-a)g'(y_t)}\,.\end{aligned}$$

Hence,

$$\begin{aligned}|y_{t+1}-a| &= \left|y_t-a-\gamma\frac{f(y_t)}{f'(y_t)}\right|\\ &= \left|(y_t-a)\left(1-\frac{\gamma g(y_t)}{\gamma g(y_t)+(y_t-a)g'(y_t)}\right)\right|\\ &= |y_t-a|^2\cdot\left|\frac{g'(y_t)}{\gamma g(y_t)+(y_t-a)g'(y_t)}\right|\\ &= |y_t-a|^2\cdot\left|\gamma\frac{g(y_t)}{g'(y_t)}+(y_t-a)\right|^{-1}\\ &\le |y_t-a|^2\cdot\left(\left|\frac{g(y_t)}{g'(y_t)}\right|-|y_t-a|\right)^{-1}\,.\end{aligned}$$

Observe that by Leibniz rule:

$$\frac{g'(y_t)}{g(y_t)} = \sum_{i=1}^{r}\frac{\gamma_i}{y_t-b_i}\,.$$

By the induction hypothesis $|y_t-a| \le \epsilon 2^{-3(m+s)}\cdot 2^{-2^t}$; so, we have for all $i\in[r]$:

$$|y_t-b_i| \ge |a-b_i|-|y_t-a| \ge \epsilon\cdot(1-2^{-2^t}2^{-3(m+s)})\,.$$

Since $\gamma_i, r \le 2^s$ (because bitsize-s circuit can have degree at most 2^s), we can upper bound as:

$$\left|\frac{g'(y_t)}{g(y_t)}\right| \le \sum_{i\in[r]}\frac{\gamma_i}{|y_t-b_i|} \le \frac{2^{2s}}{\epsilon(1-2^{-2^t}2^{-3(m+s)})}\,.$$

This means, $\left|\frac{g(y_t)}{g'(y_t)}\right| \ge 2^{-2s-1}\epsilon$. Consequently,

$$\begin{aligned}\left|\frac{g(y_t)}{g'(y_t)}\right|-|y_t-a| &\ge 2^{-2s-1}\epsilon-\epsilon 2^{-3(m+s)}\cdot 2^{-2^t}\\ &= \epsilon 2^{-3s}\cdot\left(2^{s-1}-2^{-3m}\cdot 2^{-2^t}\right)\end{aligned}$$

Hence,

$$\begin{aligned}|y_{t+1}-a| &\le |y_t-a|^2\cdot\left(\left|\frac{g(y_t)}{g'(y_t)}\right|-|y_t-a|\right)^{-1}\\ &\le 2^{-2^{t+1}}(\epsilon\cdot 2^{-3(m+s)})^2\cdot\left(\epsilon 2^{-3s}\right)^{-1}\\ &\le 2^{-2^{t+1}}\cdot\epsilon\cdot 2^{-3(m+s)}\end{aligned}$$

This finishes the inductive step and we are done. □

We leave some interesting questions open: Can we improve bitsize(a) to poly$(s+m)$? Can we prove a bound for bitsize(a) without requiring ÷ gates?