comput. complex. (2021) 30:15

1016-3328/21/020001-47
published online October 15, 2021
https://doi.org/10.1007/s00037-021-00215-0

computational complexity

FACTORIZATION OF POLYNOMIALS GIVEN BY ARITHMETIC BRANCHING PROGRAMS

Amit Sinhababu and Thomas Thierauf

Abstract. Given a multivariate polynomial computed by an arithmetic branching program (ABP) of size s, we show that all its factors can be computed by arithmetic branching programs of size $\text{poly}(s)$. Kaltofen gave a similar result for polynomials computed by arithmetic circuits. The previously known best upper bound for ABP-factors was $\text{poly}(s^{\log s})$.

Keywords. Arithmetic Branching Program, Multivariate Polynomial Factorization, Hensel Lifting, Newton Iteration, Hardness vs Randomness

Subject classification. 68W30

1. Introduction

Polynomial factoring is a classical question in algebra. For factoring multivariate polynomials, we have to specify a model for representing polynomials. A standard model in algebraic complexity to represent polynomials is *arithmetic circuits* (aka *straight-line programs*). Other well-known models are *arithmetic branching programs* (ABP), *arithmetic formulas*, *dense representations*, where the coefficients of *all* n-variate monomials of degree $\leq d$ are listed, or *sparse representations*, where only the nonzero coefficients are listed. Given a polynomial in some model, one can ask for efficient algorithms for computing its factors represented in the same model. That leads to the following question.

Question 1.1 (Factor size upper bound). *Given a polynomial of degree d and size s in a representation, do all of its factors have size* $\mathrm{poly}(s,d)$ *in the same representation?*

For example, in the dense representation, the size of the input polynomial and the output factors is the same, namely $\binom{n+d}{d}$, for n-variate polynomials of degree d. But for other representations, the factor of a polynomial may take *larger* size than the polynomial itself. For example, in the sparse representation the polynomial $x^d - 1$ has size 2, but its factor $1 + x + \cdots + x^{d-1}$ has size d.

Arithmetic circuits. The algebraic complexity class VP contains all families of polynomials $\{f_n\}_n$ that have degree $\mathrm{poly}(n)$ and arithmetic circuits of size $\mathrm{poly}(n)$. Kaltofen (1989) showed that VP is closed under factoring: Given a polynomial $f \in \mathrm{VP}$ of degree d computed by an arithmetic circuit of size s, all its factors can be computed by an arithmetic circuit of size $\mathrm{poly}(s,d)$.

Arithmetic branching programs. Our main result (Theorem 4.1) is an analog of Kaltofen's result:

ABPs are closed under factoring.

That is, the factors of a polynomial given by an ABP have polynomially bounded ABP-size.

Note that Kaltofen's proof technique for circuit factoring does not give a $\mathrm{poly}(s,d)$-size upper bound for the size of formulas or ABPs. The construction there results in a circuit, even if the input polynomial is given as a formula or an ABP. Converting a circuit to an arithmetic formula or an ABP may cause super-polynomial blowup of size.

Analogous to VP, classes VF and VBP contain families of polynomials that can be computed by polynomial-size arithmetic formulas and branching programs, respectively. Note that the size also bounds the degree of the polynomials in these models. Arithmetic branching programs are an intermediate model in terms of computational power, between arithmetic formulas and arithmetic circuits,

$$\mathrm{VF} \subseteq \mathrm{VBP} \subseteq \mathrm{VP}\,.$$

ABPs are interesting in algebraic complexity as they essentially capture the power of linear algebra, for example, they can efficiently compute determinants. The *determinantal complexity* of a polynomial $f(x_1, \ldots, x_n)$ is the smallest possible dimension of a matrix M that has linear polynomials as entries, such that $\det(M) = f$. If f is computed by an ABP of size s, then the determinantal complexity of f is at most s. In the opposite direction, if the determinantal complexity of f is s, then the ABP-size of f is $\text{poly}(s)$.

ABPs have several equivalent characterizations. They can be captured via iterated matrix multiplication, weakly-skew circuits, skew circuits, and determinants of a symbolic matrices. See Mahajan (2014) for an overview of these connections.

The classes VF, VBP, VP are conjectured to be different from each other. For example, the determinant is in VBP, but is conjectured to be not in VF. Similarly, we have polynomials that are in VP, but are conjectured to be not in VBP. Note that if VBP is indeed strictly contained in VP, then the closure under factoring of VP does not directly imply closure under factoring of VBP or VF.

Proof technique. A standard technique to factor multivariate polynomials has typically two main steps. Starting from two coprime *univariate* factors, the first step uses a method called *Hensel lifting* to lift the factors to high enough precision. The second step, sometimes called the *jump step* or *reconstruction step*, consists of reconstructing a factor from a corresponding lifted factor by solving a system of linear equations.

The earlier works for polynomial factorization use a version of Hensel lifting, where in each iteration the lifted factors remain *monic*. It seems as this version is not efficient for ABPs. We observe that monicness of the lifted factor is not necessary for the jump step. This allows us to use a simplified version of Hensel lifting that is efficient for ABPs. Finally, we use the fact that the determinant can be computed efficiently by ABPs.

Essentially, the only reason we cannot extend our result to *formulas* is due to absence of small arithmetic formulas to compute determinants. If the input polynomial is given as a formula, the

preprocessing steps and Hensel lifting keep the formula size small. Just the last step, the factor reconstruction involves computing determinants, and that is where small formulas fail.

Remark 1.2. *If the input polynomial is given as a circuit, we can construct circuits for factors in the same way as constructing ABPs. Hence, as a by-product, we also literally provide another proof for the classical circuit factoring result of Kaltofen.*

Motivation. To ask for the size of the factors in the same model as the input polynomial is clearly a very natural question. Note that Kaltofen's result would already give small *circuits* for the factors in the more restricted models like formulas and branching programs. Below, we give two more concrete reasons why it is interesting to study this question.

Closure under factoring in a model indicates *robustness* under multiplication of hard polynomials in that model. Namely, with a polynomial family $\{f_n\}$ that is *hard* for such a class, i.e., requires super-polynomial size, we get a whole class of polynomials that are hard as well, namely, for all polynomials g, all its nonzero multiples $\{gf_n\}$. Lower bounds for all the nonzero multiples of an explicit hard polynomial may lead to lower bounds for ideal proof systems (Forbes *et al.* 2016). Closure under factoring is thus relevant also in the connection between algebraic complexity and proof complexity (Forbes *et al.* 2016).

Closure under factoring is used in the hardness vs. randomness trade-off results in algebraic complexity. See, for example, the excellent survey of Kumar & Saptharishi (2019) for details on this topic. The celebrated result of Kabanets & Impagliazzo (2004, Theorem 7.7) showed that a sufficiently strong lower bound for arithmetic circuits would *derandomize* polynomial identity testing (PIT). The proof of derandomization uses a hard polynomial as well as the upper bound on the size of factors of a polynomial computed by the circuit (Kaltofen 1989). As a corollary of our result, we get a similar statement in terms of ABPs: an exponential, or super-polynomial lower bound for ABPs for an explicit multilinear polynomial yields quasi-poly, respectively, sub-exponential, black-box derandomization of PIT for polynomials in VBP.

Construction and reduction to PIT. We formulate our main result, Theorem 4.1, as an existential claim: There *exist* small ABPs for the factors of a polynomial given by an ABP. But in fact, one can construct these ABPs for the factors efficiently.

The construction algorithm is randomized. But randomization is only used for polynomial identity testing (PIT) of ABPs. Hence, similar as Kopparty, Saraf & Shpilka (2015) showed for circuits, the construction can be efficiently reduced to PIT.

There are two versions of PIT: *white-box* and *black-box*. For both, one has to find out whether a given polynomial p is identically zero. In the white-box case, polynomial p is explicitly given, for example, as an ABP. In the black-box case, one knows only the size of the ABP and is only allowed to evaluate p at various points.

For circuits, black-box factoring is equivalent to black-box PIT, and white-box factoring is equivalent to white-box PIT (Kopparty *et al.* 2015). In Section 5, we will explain that this holds similarly for ABPs. But these equivalences are not the main point of our paper. We focus more on how simple factorization can be kept. For that, we describe a way to factorize in Sections 3 and 4 that, except for the very last step, also works for formulas. However, this procedure reduces only to black-box PIT, a stronger oracle than white-box PIT.

For the reduction to white-box PIT, it is crucial that in a preprocessing step, the input polynomial is transformed into a square-free polynomial. The standard approach for reducing to the square-free case is via dividing the polynomial by the GCD of the polynomial and its derivative, see Kopparty, Saraf & Shpilka (2015). The GCD of two polynomials given by ABPs can be directly computed by a small ABP using subresultants. On the other hand, we observe that there is no need to reduce to the square-free case. It suffices to have *one* irreducible factor of multiplicity one. This weaker transformation involves only derivatives and can be computed by small ABPs, as well as small arithmetic formulas. We describe both the approaches in Section 5.

Comparison with prior works. There are several proofs of the closure of VP under factors (Bürgisser 2004, 2013; Chou *et al.*

2019a; Dutta *et al.* 2018; Kaltofen 1986, 1987, 1989; Kopparty *et al.* 2015; Oliveira 2016). None of the previous proofs directly extends to the closure of VBP, i.e., branching programs, under factors.

Recently, Dutta, Saxena & Sinhababu (2018) and also Oliveira (2016) considered factoring in restricted models like formulas, ABPs, and small-depth circuits. They reduce polynomial factoring to approximating power series roots of the polynomial to be factored. Then, they use the well-known technique of *Newton iteration* for approximating the roots. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$. If $p(\boldsymbol{x}, y)$ is the given polynomial and $q(\boldsymbol{x})$ is a root w.r.t. y, i.e., $p(\boldsymbol{x}, q(\boldsymbol{x})) = 0$, then $y - q(\boldsymbol{x})$ is a factor of p. Newton iteration repeatedly uses the following recursive formula to approximate q:

$$y_{t+1} = y_t - \frac{p(\boldsymbol{x}, y_t)}{p'(\boldsymbol{x}, y_t)},$$

where p' is the derivative of p w.r.t. variable y.

If p is given as a circuit, the circuit for y_{t+1} is constructed from the circuit of y_t. For the circuit model, we can assume that $p(\boldsymbol{x}, y)$ has a single leaf node y where we feed y_t. But for formula and branching programs, we may have d many leaves labeled by y, where d is the degree of p in terms of y. As we cannot reuse computations in formula or branching programs, we have to make d copies of y_t in each round. This leads to $d^{\log d}$ blowup in size.

Oliveira (2016) used the idea of approximating roots via an approximator polynomial function of the coefficients of a polynomial. This gives good upper bound on the size of factors of ABPs, formulas, and bounded depth circuits under the assumption that the individual degrees of the variables in the input polynomial are bounded by a constant.

Recently, Chou, Kumar & Solomon (2019a) proved closure of VP under factoring using Newton iteration for several variables for a system of polynomial equations. This approach also faces the same problem for the restricted models.

Instead of lifting roots, another classical technique for multivariate factoring is *Hensel lifting*, where factors modulo an ideal are lifted. Hensel lifting has a slow version, where the power of the ideal increases by one in each round. The other version due to

Zassenhaus (1969) is fast, the power of the ideal gets doubled in each round.

Kaltofen's proofs use slow versions of Hensel lifting iteratively for d rounds, where d is the degree of the given polynomial (Kaltofen 1987, 1989). That leads to an exponential blowup of size in models where the previous computations cannot be reused, as using previous lifts twice would need two copies each time.

Kopparty, Saraf & Shpilka (2015) use the standard way of doing fast Hensel lifting for $\log d$ rounds, where in each round the lifted factors are kept monic. To achieve this, one has to compute a polynomial division with remainder. Implementing this version of Hensel lifting for ABPs or formulas seems to require to make d copies of previous computations in each round. Thus, that way would lead to a $d^{\log d}$ size blowup.

Here, we use a classic version of fast Hensel lifting, that needs $\log d$ rounds and additionally, in each round, we have to make copies of previous computations only constantly many times. As we mentioned earlier, we avoid to maintain the monicness.

Though various versions of Hensel lifting (factorization lifting) and Newton iteration techniques (root lifting) are equivalent in the sense that one can be derived from the other (von zur Gathen 1984), it is interesting that the former gives a better factor size upper bound for the model of ABP.

Organization of the paper. In Section 2, we give some basic facts on algebra and the computational model of ABP that we use in this paper. In Section 3, we discuss the preprocessing steps and other lemmas that we use to prove in our main result. In Section 4, we prove our main result (Theorem 4.1). In Section 5, we discuss how our proof can be converted to a randomized algorithm for computing the factors. We also highlight the steps where randomization is used and argue that they can deterministically be turned into PIT-problems. In Section 6, we discuss how to apply Theorem 4.1 to extend the classic hardness to derandomization result of Kabanets and Impagliazzo to the model of ABPs.

2. Preliminaries

We consider multivariate polynomials over a field $\mathbb{F}$ of characteristic 0. A polynomial p is called *irreducible*, if it cannot be factored into the product of two non-constant polynomials. Polynomial p is called *square-free*, if for any non-constant factor q, the polynomial q^2 is not a factor of p.

By $\deg(p)$, we denote the total degree of p. Let x and $\boldsymbol{z} = (z_1, \ldots, z_n)$ be variables and $p(x, \boldsymbol{z})$ be a $(n+1)$-variate polynomial. Then, we can view p as a univariate polynomial $p = \sum_i a_i(\boldsymbol{z})\, x^i$ over $\mathbb{K}[x]$, where $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$. The *$x$-degree of p* is denoted by $\deg_x(p)$. It is the highest degree of x in p. Polynomial p is called *monic in x*, if the coefficient $a_{d_x}(\boldsymbol{z})$ is the constant 1 polynomial, i.e., $a_{d_x}(\boldsymbol{z}) = 1$, where $d_x = \deg_x(p)$.

By $\mathrm{poly}(n)$, we denote the class of polynomials in $n \in \mathbb{N}$.

Polynomial Identity Test (PIT). There is a randomized algorithm by the DeMillo–Lipton–Schwartz–Zippel to find out whether a given polynomial is identical zero; see Chou *et al.* (2019b) and the references therein for more details and history of this theorem.

THEOREM 2.1 (Polynomial Identity Test). *Let $p(\boldsymbol{x})$ be an n-variate nonzero polynomial of total degree d. Let $S \subseteq \mathbb{F}$ be a finite set. For $\boldsymbol{\alpha} \in S^n$ picked independently and uniformly at random,*

$$\Pr[\, p(\boldsymbol{\alpha}) = 0 \,] \;\leq\; \frac{d}{|S|}\,.$$

Since we assume the field $\mathbb{F}$ to have characteristic 0, we can choose the set S in Theorem 2.1 large enough, for example, $|S| = 2d$, to keep the probability $\Pr[\, p(\boldsymbol{\alpha}) = 0 \,]$ small. In case of finite fields, we may have to work over a field extension so that the field is large enough.

Rings and ideals. Let $\mathcal{R}$ be a commutative ring with identity. A set $\mathcal{I} \subseteq \mathcal{R}$ is an *ideal of $\mathcal{R}$*, if it is closed under addition and under scalar multiplications by elements of $\mathcal{R}$. That is, for every $r \in \mathcal{R}$ and every $a \in \mathcal{I}$, the product ar is in $\mathcal{I}$.

Two elements $r, s \in \mathcal{R}$ are *congruent modulo* $\mathcal{I}$, if $r - s \in \mathcal{I}$. This is denoted as

$$r \equiv s \pmod{\mathcal{I}}.$$

For a set $S \subseteq \mathcal{R}$, the *ideal generated by* S is denoted by $\langle S \rangle$. It consists of all elements $r \in \mathcal{R}$ that can be written as a finite sum,

$$r = \sum_{i=1}^{\ell} r_i s_i,$$

for some $\ell \geq 1$, $r_i \in \mathcal{R}$, and $s_i \in S$, for $i = 1, \ldots, \ell$. It is easy to check that $\langle S \rangle$ is indeed an ideal of $\mathcal{R}$.

For an ideal $\mathcal{I}$ of $\mathcal{R}$ and $m \geq 1$, the *mth power of* $\mathcal{I}$ is the ideal $\mathcal{I}^m$ generated by the elements s of the form $s = a_1 a_2 \cdots a_m$, where $a_i \in \mathcal{I}$, for $i = 1, \ldots, m$.

We will apply these notions in the following setting. The ring $\mathcal{R}$ will be the polynomial ring $\mathcal{R} = \mathbb{K}[x, y]$, for two variables x, y and another polynomial ring $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$, where $\mathbb{F}$ is a field and $\boldsymbol{z}$ is a tuple of variables. Note that $\mathbb{K}[x, y]$ is commutative and has an identity. As ideal, we consider $\mathcal{I} = \langle y \rangle$, the ideal generated by polynomial $y \in \mathbb{K}[x, y]$. Then, $\mathcal{I}$ contains all polynomials that have factor y,

$$\mathcal{I} = \{\, py \mid p \in \mathbb{K}[x, y] \,\}.$$

Similarly, the mth power of $\mathcal{I}$ contains all polynomials that have factor y^m,

$$\mathcal{I}^m = \{\, py^m \mid p \in \mathbb{K}[x, y] \,\}.$$

Computational models. An *arithmetic circuit* is a directed acyclic graph, whose leaf nodes are labeled by the variables $x_1, \ldots, x_n$ and various constants from the underlying field. The other nodes are labeled by sum gates or product gates. A node labeled by a variable or constant computes the same. A node labeled by sum or product computes the sum or product of the polynomials computed by nodes connected by incoming edges. The *size of an arithmetic circuit* is the total number of its edges.

An *arithmetic formula* is a special kind of arithmetic circuit. A formula has the structure of a directed acyclic tree. Every node in

a formula has out-degree at most one. As we cannot *reuse* computations in a formula, it is considered to be weaker than circuits.

An *arithmetic branching program* (ABP) is a layered directed acyclic graph with a single source node and a single sink node. An edge of an ABP is labeled by a variable or a constant from the field. The weight corresponding to a path from the source to the sink is the product of the polynomials labeling the edges on the path. The polynomial $f(x_1, \ldots, x_n)$ computed by the ABP is the sum of the weights of the all possible paths from source to sink.

The *size of an ABP* is the number of its edges. The size of the smallest ABP computing f is denoted by $\text{size}_{\text{ABP}}(f)$. The degree of a polynomial computed by an ABP of size s is at most s.

Instead of *arithmetic*, the above models are also called *algebraic circuits*, *algebraic formulas*, and *algebraic branching programs*, respectively.

Properties of ABPs. *Univariate* polynomials have small ABPs. Let $p(x)$ be a univariate polynomial of degree d over any field. It can be computed by an ABP of size $2d + 1$, actually even by a formula of that size.

For univariate polynomials $p(x), q(x)$ over any field, the *extended Euclidean algorithm* computes the GCD $h = \gcd(p, q)$ and also the *Bézout-coefficients*, polynomials a, b such that $ap+bq = h$, where $\deg(a) < \deg(q)$ and $\deg(b) < \deg(p)$. Let p have the larger degree, $d = \deg(p) \geq \deg(q)$. Then, clearly $\deg(h), \deg(a), \deg(b) \leq d$, and consequently, all these polynomials, p, q, h, a, b have ABP-size at most $2d + 1$.

Let $p(\boldsymbol{x}), q(\boldsymbol{x})$ be multivariate polynomials in $\boldsymbol{x} = (x_1, \ldots, x_n)$. For the ABP-size with respect to *addition* and *multiplication*, we have

1. $\text{size}_{\text{ABP}}(p + q) \leq \text{size}_{\text{ABP}}(p) + \text{size}_{\text{ABP}}(q)$,

2. $\text{size}_{\text{ABP}}(pq) \leq \text{size}_{\text{ABP}}(p) + \text{size}_{\text{ABP}}(q)$.

For the sum of two ABPs B_p, B_q, one can put B_p and B_q in parallel by merging the two source nodes of B_p and B_q into one new source node, and similar for the two sink nodes. For the product,

one can put B_p and B_q in series by merging the sink of B_p with the source of B_q.

Another basic operation is *substitution*. Let $p(x_1, \ldots, x_n)$ and $q_1(\boldsymbol{x}), \ldots, q_n(\boldsymbol{x})$ be polynomials. Let $\text{size}_{\text{ABP}}(q_i) \leq s$, for $i = 1, \ldots, n$. Then, we have

$$\text{size}_{\text{ABP}}(p(q_1, \ldots, q_n)) \leq s \cdot \text{size}_{\text{ABP}}(p).$$

To get an ABP for $p(q_1(\boldsymbol{x}), \ldots, q_n(\boldsymbol{x}))$, replace an edge labeled x_i in the ABP B_p for p by the ABP B_{q_i} for q_i.

It is known that the *determinant of a symbolic matrix* of dimension n can be computed by an ABP of size $\text{poly}(n)$ (Berkowitz 1984; Chistov 1985; Csanky 1976; Mahajan & Vinay 1999). By substitution, the entries of the matrix can itself be polynomials computed by ABPs.

Resultant, subresultant, and GCD. Given two polynomials $p(x, \boldsymbol{y})$ and $q(x, \boldsymbol{y})$ in variables x and $\boldsymbol{y} = (y_1, \ldots, y_n)$, consider them as polynomials in x with coefficients in $\mathbb{F}[\boldsymbol{y}]$. The *resultant of p and q w.r.t. x*, denoted by $\text{Res}_x(p, q)$, is the determinant of the Sylvester matrix of p and q. For the definition of the Sylvester matrix, see von zur Gathen & Gerhard (2013, Section 6.3). Note that $\text{Res}_x(p, q)$ is a polynomial in $\mathbb{F}[\boldsymbol{y}]$. When we have ABPs for the coefficients of p and q, there is a poly-size ABP that computes the resultant $\text{Res}_x(p, q)$.

Basic properties of the resultant are that it can be represented as a linear combination of p and q and that it provides information about the GCD of p and q.

Lemma 2.2 (Resultant and GCD). *Let $p(x, \boldsymbol{y})$ and $q(x, \boldsymbol{y})$ be polynomials of degree $\leq d$ and $h = \gcd(p, q)$.*

(i) $\deg(\text{Res}_x(p, q)) \leq 2d^2$,

(ii) $\exists u, v \in \mathbb{F}[x, \boldsymbol{y}] \quad up + vq = \text{Res}_x(p, q)$,

(iii) $\text{Res}_x(p, q) = 0 \iff \deg_x(h) > 0$.

See von zur Gathen & Gerhard (2013, Section 6.3) for a proof of the lemma. Note that by item (iii), we can find out whether p and q have a non-trivial GCD via a polynomial identity test (PIT). In the contrapositive, when $\mathrm{Res}_x(p,q) \neq 0$, we have that $\deg_x(h) = 0$, and hence, h is a polynomial just in $\boldsymbol{y}$. It is known that when p and q are monic in x, actually $h = \gcd(p,q) = 1$.

Subresultants of p and q are polynomials $S_j(p,q) \in \mathbb{F}[x,\boldsymbol{y}]$ of x-degree j, computed from determinants of submatrices of the Sylvester matrix, for $j = 0, 1, \ldots, \deg(q) \leq \deg(p)$. For the definition, see, for example, the textbook Geddes *et al.* (1992, Chapter 7). See also Sasaki & Suzuki (1992, Section 4) for a quick overview. Like for the resultant, when we have ABPs for the coefficients of p and q, there are poly-size ABPs that compute the subresultants.

The interest in subresultants comes from the fact that they can be used to get poly-size ABPs to compute the GCD. Let $h = \gcd(p,q)$ and $d = \deg_x(h)$. Note first that d can be determined by several zero-tests of minors of the Sylvester matrix of p and q, see Geddes *et al.* (1992, Theorem 7.3). It follows from the *fundamental theorem of polynomial remainder sequence*, see Geddes *et al.* (1992, Theorem 7.4), that the dth subresultant $S_d(p,q)$ is a multiple of h: There is a $k \in \mathbb{F}[\boldsymbol{y}]$ such that $S_d(p,q) = kh$.

In our case, polynomials p and q will be monic in x. Hence, the GCD h will be monic in x too. Thus, the leading coefficient of $S_d(p,q)$ w.r.t. x is the above polynomial $k \in \mathbb{F}[\boldsymbol{y}]$. So if we divide $S_d(p,q)$ by its leading coefficient w.r.t. x, we recover the GCD h. For ABPs, the division can be done with small increase in size using Strassen's classic technique.

Lemma 2.3. *Given two monic polynomials $p, q \in \mathbb{F}[x,\boldsymbol{z}]$ computed by ABPs of size at most s, there is an efficient randomized algorithm that outputs an ABP of size* $\mathrm{poly}(s)$ *that computes their GCD.*

3. Preprocessing steps and algebraic tool kit

Before we start the Hensel lifting process, a polynomial should fulfill certain properties that the input polynomial might not have. In this section, we describe transformations of a polynomial that

achieve these properties such that ABPs can compute the transformation and its inverse, and factors of the polynomials are maintained.

We also explain how to compute homogeneous components and how to solve linear systems via ABPs. We show how to handle the special case when the given polynomial is just a power of an irreducible polynomial.

3.1. Computing homogeneous components and coefficients of a polynomial. Let $p(x, \boldsymbol{z})$ be a polynomial of degree d in variables x and $\boldsymbol{z} = (z_1, \ldots, z_n)$. Let B_p be an ABP of size s that computes a polynomial p. Write p as a polynomial in x, with coefficients from $\mathbb{F}[\boldsymbol{z}]$,

$$p(x, \boldsymbol{z}) = \sum_{i=0}^{d} p_i(\boldsymbol{z})\, x^i\,.$$

We show that all the coefficients $p_i(\boldsymbol{z})$ have ABPs of size $\mathrm{poly}(s, d)$.

The argument is similar to Strassen's *homogenization* technique for arithmetic circuits, an efficient way to compute all the homogeneous components of a polynomial. The same technique can be used for ABPs. See Saptharishi (2021, Lemma 5.2 and Remark). Here, we sketch the proof idea.

Each node v of B_p we split into $d+1$ nodes $v_0, \ldots, v_d$, such that node v_i computes the degree i part of the polynomial computed by node v, for $i = 0, 1, \ldots, d$. Consider an edge e between node u and v in B_p.

- If e is labeled with a constant $c \in \mathbb{F}$ or a variable z_i, then we put an edge between u_i and v_i with label c or z_i, respectively.
- If e is labeled with variable x, then we put an edge between u_i and v_{i+1} with label 1.

The resulting ABP has one source node and $d+1$ sink nodes. The ith sink node computes $p_i(\boldsymbol{z})$.

For each edge of B_p, we get either d or $d+1$ edges in the new ABP. Hence, its size is bounded by $s(d+1)$.

LEMMA 3.1 (Coefficient extraction). *Let* $p(x, \boldsymbol{z}) = \sum_{i=0}^{d} p_i(\boldsymbol{z})\, x^i$ *be a polynomial. Then, for all* i*,*

$$\mathrm{size}_{\mathrm{ABP}}(p_i) \leq (d+1)\, \mathrm{size}_{\mathrm{ABP}}(p)\,.$$

The technique can easily be extended to constantly many variables. For two variables, consider $p(x, y, \boldsymbol{z}) = \sum_{i,j} p_{i,j}(\boldsymbol{z})\, x^i y^j$. Then, from an ABP of size s for p we get ABPs for the coefficients $p_{i,j}(\boldsymbol{z})$ of size $s(d+1)^2$ similarly as above.

In *homogenization*, we want to compute the homogeneous components of p. That is, write $p(\boldsymbol{z}) = \sum_{i=0}^{d} p_i(\boldsymbol{z})$, where p_i is a homogeneous polynomial of degree i. From an ABP B_p for p, we get ABPs for the p_i's similarly as above: In the definition of the new edges, only for constant label, we put the edge from u_i to v_i. In case of any variable label z_j, we put the edge from u_i to v_{i+1} with label z_j. Then, the ith sink node computes $p_i(\boldsymbol{z})$. The size is bounded by $s(d+1)$.

LEMMA 3.2 (Homogenization). *Let* $p(\boldsymbol{z}) = \sum_{i=0}^{d} p_i(\boldsymbol{z})$ *be a polynomial, where* p_i *is a homogeneous polynomial of degree* i*, for* $i = 0, 1, \ldots, d$*. Then, for all* i*,*

$$\mathrm{size}_{\mathrm{ABP}}(p_i) \leq (d+1)\, \mathrm{size}_{\mathrm{ABP}}(p).$$

3.2. Computing q from $p = q^e$. A special case is when the given polynomial $p(\boldsymbol{z})$ is a power of one irreducible polynomial $q(\boldsymbol{z})$, i.e., $p = q^e$, for some $e > 1$. This case is handled separately. Kaltofen (1987) showed how to compute q for circuits, ABPs, and arithmetic formulas. Here, we give a short proof from Dutta (2018).

LEMMA 3.3. *Let* $p = q^e$*, for polynomials* $p(\boldsymbol{z}), q(\boldsymbol{z})$*. Then,*

$$\mathrm{size}_{\mathrm{ABP}}(q) \leq \mathrm{poly}(\mathrm{size}_{\mathrm{ABP}}(p)).$$

PROOF. We may assume that p is nonzero; otherwise, the claim is trivial. We want to apply Newton's binomial theorem to compute $q = p^{1/e}$. For this, we need that $p(0, \ldots, 0) = 1$. If this is not the case, we first transform p as follows:

1. If $p(0,\ldots,0) = 0$, let $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_n)$ be a point where $p(\boldsymbol{\alpha}) \neq 0$. By Theorem 2.1, a random point $\boldsymbol{\alpha}$ will work, with high probability. Now we shift the variables and work with the shifted polynomial $p_1(\boldsymbol{z}) = p(\boldsymbol{z}+\boldsymbol{\alpha})$.

   Still, $p_1(0,\ldots,0)$ might be different from 1. In this case, we also apply the next item to p_1.

2. If $p(0,\ldots,0) = a_0 \neq 0,1$, then we work with $p_2(\boldsymbol{z}) = p(\boldsymbol{z})/a_0$. Then, $p_2(0,\ldots,0) = 1$.

Note that both transformations are easily reversible. Hence, in the following we simply assume that $p(0,\ldots,0) = 1$.

By Newton's binomial theorem, we have

$$q = p^{1/e} = (1+(p-1))^{1/e} = \sum_{i=0}^{\infty} \binom{1/e}{i} (p-1)^i \,. \tag{3.4}$$

Note that $p^{1/e}$ is a polynomial of degree $d_q = \deg(q)$. Since $p-1$ is constant-free, the terms $(p-1)^j$ in the right-hand side of (3.4) have degree $> d_q$, for $j > d_q$. Thus, (3.4) turns into a finite sum modulo the ideal $\langle \boldsymbol{z} \rangle^{d_q+1}$,

$$q = \sum_{i=0}^{d_q} \binom{1/e}{i} (p-1)^i \bmod \langle \boldsymbol{z} \rangle^{d_q+1} \,. \tag{3.5}$$

Define $Q = \sum_{i=0}^{d_q} \binom{1/e}{i}(p-1)^i$. Let $\mathrm{size}_{\mathrm{ABP}}(p) = s$. Then, we clearly have $\mathrm{size}_{\mathrm{ABP}}(Q) \leq \mathrm{poly}(s)$. To compute $q = Q \bmod \langle \boldsymbol{z} \rangle^{d_q+1}$, we have to truncate the terms in Q of degree $> d_q$. This can be done by computing the homogeneous components of Q as described in Lemma 3.2. We conclude that $\mathrm{size}_{\mathrm{ABP}}(q) \leq \mathrm{poly}(s)$. □

Recall that the underlying field $\mathbb{F}$ has characteristic 0. Note that we above proof would not work when $\mathbb{F}$ had finite characteristic p that divides e.

3.3. Reducing the multiplicity of a factor. In the earlier works on bivariate and multivariate polynomial factoring, typically the problem is reduced to factoring a *square-free* polynomial. This

is convenient at various places in the Hensel lifting process. The technique to reduce to the square-free case is via taking the GCD of the input polynomial and its derivative.

As we will see, we do not need the polynomial to be square-free. It suffices to have one irreducible factor with multiplicity one, and another coprime factor. Thereby, we avoid GCD computations and the approach stays feasible even for formulas.

Let $p(\boldsymbol{z})$ be the given polynomial, for $\boldsymbol{z} = (z_1, \ldots, z_n)$. The special case that p is a power of one irreducible polynomial we just handled in Section 3.2. Hence, we may assume that p has at least two irreducible factors. So, let $p = q^e\, p_0$, where q is irreducible and coprime to p_0.

Consider the derivative of p w.r.t. some variable, such that q also depends on this variable, say z_1.

$$\frac{\partial p}{\partial z_1} = q^{e-1}\left((e-1)\frac{\partial q}{\partial z_1}p_0 + q\frac{\partial p_0}{\partial z_1}\right). \tag{3.6}$$

Note that q does not divide the factor $\left((e-1)\frac{\partial q}{\partial z_1}p_0 + q\frac{\partial p_0}{\partial z_1}\right)$ in (3.6). Hence, the multiplicity of factor q in $\frac{\partial p}{\partial z_1}$ is reduced by one compared to p.

For the ABP-size, we write p as a polynomial in z_1, i.e., $p(\boldsymbol{z}) = \sum_{i=0}^{d} a_i z_1^i$, where the coefficients a_i are polynomials in $z_2, \ldots, z_n$. By Lemma 3.1, when p has an ABP of size s, the coefficients a_i can be computed by ABPs of size $s' = s(d+1)$. We observe that then the coefficients of the derivative polynomial $\frac{\partial p}{\partial z_1} = \sum_{i=1}^{d} i a_i z_1^{i-1}$ have ABPs of size $s'+1$.

We repeat taking derivatives $k = e-1$ times and get $\frac{\partial^k p}{\partial z_1^k}$, which has the irreducible factor q with multiplicity one, as desired.

The coefficients of $\frac{\partial^k p}{\partial z_1^k}$ can be computed by ABPs of size $s'+1$. This yields an ABP of size $\mathrm{poly}(s)$ that computes $\frac{\partial^k p}{\partial z_1^k}$.

3.4. Transforming to a monic polynomial. Given a polynomial $p(\boldsymbol{z})$ in variables $\boldsymbol{z} = (z_1, \ldots, z_n)$ over field $\mathbb{F}$, there is a standard trick to make it monic in a new variable x by applying a linear transformation on the variables: for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$, let

$$\tau_{\boldsymbol{\alpha}} : \quad z_i \mapsto \alpha_i x + z_i,$$

for $i = 1, \ldots, n$. Let $p_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$ be the resulting polynomial. Note that p and $p_{\boldsymbol{\alpha}}$ have the same total degree, say d.

To see what the transformation does, consider the terms of degree d in p. Let $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ such that $|\boldsymbol{\beta}| = \sum_{i=1}^{n} \beta_i = d$. We denote the term $\boldsymbol{z}^{\boldsymbol{\beta}} = z_1^{\beta_1} \cdots z_n^{\beta_n}$. Then, the homogeneous component of degree d in p can be written as $a_d(\boldsymbol{z}) = \sum_{|\boldsymbol{\beta}|=d} c_{\boldsymbol{\beta}} \boldsymbol{z}^{\boldsymbol{\beta}}$. Note that a_d is a nonzero polynomial.

For the transformed polynomial $p_{\boldsymbol{\alpha}}$, we have $\deg_x(p_{\boldsymbol{\alpha}}) = d$ and the coefficient of the leading x-term x^d is $a_d(\alpha) = \sum_{|\boldsymbol{\beta}|=d} c_{\boldsymbol{\beta}} \boldsymbol{\alpha}^{\boldsymbol{\beta}}$. By Theorem 2.1, when we pick $\boldsymbol{\alpha}$ at random, $a_d(\alpha)$ will be a nonzero constant in $\mathbb{F}$ with high probability. In this case, $\frac{1}{a_d(\alpha)} p_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$ is monic in x.

LEMMA 3.7 (Transformation to monic). *Let $p(\boldsymbol{z})$ be polynomial of total degree d. Let $S \subseteq \mathbb{F}$ be a finite set. For $\boldsymbol{\alpha} \in S^n$ picked independently and uniformly at random,*

$$\Pr\left[\frac{1}{a_d(\alpha)} p_{\boldsymbol{\alpha}}(x, \boldsymbol{z}) \text{ is monic in } x \right] \geq 1 - \frac{d}{|S|},$$

where $a_d(\alpha)$ is the coefficient of x^d in $p_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$.

Given an ABP of size s that computes $p(\boldsymbol{z})$, we can construct another ABP of size $3s$ that computes $p_{\boldsymbol{\alpha}}(x, \boldsymbol{z})$. For the new ABP, replace edge labeled by z_i by the ABP computing $\alpha_i x + z_i$. For each old edge, this requires adding two new edges with labels α_i and x.

Note that we can derandomize the transformation with an oracle for ABP–PIT.

3.5. Handling the starting point of Hensel lifting. After doing the above preprocessing steps on the given polynomial $p(\boldsymbol{z})$, we call the transformed polynomial $f(x, \boldsymbol{z})$. We can assume that f of degree d can be factorized as $f = gh$, where g and h are coprime and g is irreducible. We start Hensel lifting by factoring the *univariate* polynomial $f(x, 0, \ldots, 0) \equiv f(x, \boldsymbol{z}) \pmod{\boldsymbol{z}}$. Clearly, we have the factorization $f(x, 0, \ldots, 0) = g(x, 0, \ldots, 0)\, h(x, 0, \ldots, 0)$, but these two factors might not be coprime. In this case, we do another transformation.

Remark 3.8. *Although it would suffice for our purpose to start with two coprime factors, the transformation below produces one irreducible factor.*

Let g_0 be an irreducible factor of the polynomial $g(x, 0, \ldots, 0)$. Then, we have for some univariate polynomial $h_0'(x)$ and for $h_0(x) = h_0'(x)\, h(x, 0, \ldots, 0)$,

$$\begin{aligned} g &\equiv g_0\, h_0' \pmod{\boldsymbol{z}}, \\ f &\equiv g_0\, h_0 \pmod{\boldsymbol{z}}. \end{aligned}$$

We want that g_0 is coprime to h_0' and h_0. Directly, this might *not* be the case because all factors of $f(x, 0, \ldots, 0)$ might have multiplicity > 1. However, we argue how to ensure this after a random shift $\boldsymbol{\alpha}$ of f. That is, we consider the function $f(x, \boldsymbol{z} + \boldsymbol{\alpha})$

1. First, we show how to achieve that g_0 is coprime to h_0'.

   Since g is irreducible, it is also square-free, and hence, we get $\gcd(g, \frac{\partial g}{\partial x}) = 1$. The *discriminant* of g is the resultant

   $$r(\boldsymbol{z}) = \mathrm{Res}_x(g, \frac{\partial g}{\partial x}).$$

   Polynomial $r(\boldsymbol{z})$ is nonzero and of degree $\leq 2d^2$, by Lemma 2.2. Hence, at a random point $\boldsymbol{\alpha} \in [4d^2]^n$, we have $r(\boldsymbol{\alpha}) \neq 0$, with high probability. At such a point $\boldsymbol{\alpha}$, we have that $g(x, \boldsymbol{\alpha})$ is square-free. Therefore, $g(x, \boldsymbol{z})$ is square-free modulo $(\boldsymbol{z} - \boldsymbol{\alpha})$, or, equivalently, $g(x, \boldsymbol{z} + \boldsymbol{\alpha})$ is square-free modulo $\boldsymbol{z}$. Hence, when we define g_0 and h_0' from $g(x, \boldsymbol{z} + \boldsymbol{\alpha})$ instead of $g(x, \boldsymbol{z})$, they will be coprime.

2. Similarly, we can achieve that g_0 is coprime to h_0. By the first item, it now suffices to get g_0 coprime to $h(x, 0, \ldots, 0)$.

   For showing this, we use that g_0 is coprime to h_0' and prove that $g(x, 0, \ldots, 0)$ is coprime to $h(x, 0, \ldots, 0)$. Consider the resultant of g and h w.r.t. x, the polynomial $r'(\boldsymbol{z}) = \mathrm{Res}_x(g, h)$ has degree $\leq 2d^2$. Since g and h are coprime, $r'(\boldsymbol{z}) \neq 0$. Hence, at a random point $\boldsymbol{\alpha} \in [4d^2]^n$, we have $r'(\boldsymbol{\alpha}) \neq 0$ with high probability, and hence, $g(x, \boldsymbol{\alpha})$ and $h(x, \boldsymbol{\alpha})$ are coprime univariate polynomials. Therefore, $g(x, \boldsymbol{z})$ and $h(x, \boldsymbol{z})$

are coprime modulo $(\boldsymbol{z}-\boldsymbol{\alpha})$, or, equivalently, $g(x, \boldsymbol{z}+\boldsymbol{\alpha})$ and $h(x, \boldsymbol{z}+\boldsymbol{\alpha})$ are coprime modulo $\boldsymbol{z}$.

Combining the two items, a random point $\boldsymbol{\alpha} \in [4d^2]^n$ will fulfill both properties with high probability. So instead of factoring $f(x, \boldsymbol{z})$, we do a coordinate transformation $\boldsymbol{z} \mapsto \boldsymbol{z}+\boldsymbol{\alpha}$ and factor $f(x, \boldsymbol{z}+\boldsymbol{\alpha})$ instead. From these factors, we easily get the factors of $f(x, \boldsymbol{z})$ by inverting the transformation.

Remark 3.9. *(i) The construction maintains the monicness: when $f(x, \boldsymbol{z})$ is monic in x, the same holds for $f(x, \boldsymbol{z}+\boldsymbol{\alpha})$.*

(ii) The ABP-size of $f(x, \boldsymbol{z}+\boldsymbol{\alpha})$ is at most twice the ABP-size of $f(x, \boldsymbol{z})$.

(iii) The only use of randomness to efficiently construct *an ABP for $f(x, \boldsymbol{z}+\boldsymbol{\alpha})$ is a PIT for the resultant polynomials r, r'. At this point, it is not clear that they have small ABPs. After we prove (in Theorem 4.1) that all factors of ABPs have small ABP size, we get that the polynomials r, r' have small ABPs. Thus, we can use an explicit hitting set/black-box PIT for the class of small ABPs to derandomize this step. For black-box PIT, we just need ABP-size upper bounds of g and h. For getting a deterministic polynomial time factoring algorithm, we have to try all points in the hitting set to get a shift $\boldsymbol{\alpha}$ that works.*

In the next section, we do another transformation on the input polynomial. We apply a map on the variables that maps x to x and z_i is mapped to yz_i, for a new variable y and $i = 1, \ldots, n$. Then, we factorize the transformed polynomial modulo y. Note that in this case, going modulo y has the same effect of going modulo $\boldsymbol{z}$. So we can use the above argument to ensure the starting condition for Hensel lifting is satisfied.

3.6. Reducing multivariate factoring to the bivariate case. Factoring multivariate polynomials can be reduced to the case of *bivariate* polynomials, see Kopparty, Saraf & Shpilka (2015). Let x, y, and $\boldsymbol{z} = (z_1, \ldots, z_n)$ be variables, and let $f(x, \boldsymbol{z})$ be the given

polynomial. With $f \in \mathbb{F}[x, \boldsymbol{z}]$, we associate the polynomial $\widehat{f} \in \mathbb{F}[x, y, \boldsymbol{z}]$ defined by

$$\widehat{f}(x, y, \boldsymbol{z}) = f(x, yz_1, \ldots, yz_n)\,.$$

The point now is to consider $\widehat{f}$ as a polynomial in $\mathbb{F}[\boldsymbol{z}][x, y]$, that is, as a bivariate polynomial in x and y with coefficients in $\mathbb{F}[\boldsymbol{z}]$. We list some properties.

1. $f(x, \boldsymbol{z}) = \widehat{f}(x, 1, \boldsymbol{z})$,
2. $\deg(\widehat{f}) \leq 2\deg(f)$,
3. f monic in $x \implies \widehat{f}$ monic in x,
4. $f = gh \implies \widehat{f} = \widehat{g}\,\widehat{h}$,
5. $\widehat{f} = \widetilde{g}\,\widetilde{h} \implies f = \widetilde{g}(x, 1, \boldsymbol{z})\,\widetilde{h}(x, 1, \boldsymbol{z})$.

By property 4, factors of f yield factors of $\widehat{f}$. The following lemma shows that also the irreducibility of the factors is maintained.

Lemma 3.10. *Let f be monic in x and g be a monic irreducible factor of f. Then, $\widehat{g}$ is a monic irreducible factor of $\widehat{f}$.*

Proof. By properties 3 and 4, $\widehat{g}$ is a monic factor of $\widehat{f}$. We argue that $\widehat{g}$ is irreducible.

Let $\widehat{g} = uv$ be a factorization of $\widehat{g}$. By item 5, this yields a factorization of g as $g = u(x, 1, \boldsymbol{z})\, v(x, 1, \boldsymbol{z})$. Since g is irreducible, either $u(x, 1, \boldsymbol{z})$ or $v(x, 1, \boldsymbol{z})$ is constant. Because $\widehat{g}$ is monic in x, either u or v must be constant too. □

Thus, to get an ABP for an irreducible factor g of f, first we show that there is an ABP for the irreducible factor $\widehat{g}$. This yields an ABP for g by substituting $g = \widehat{g}(x, 1, \boldsymbol{z})$.

Given an ABP B_f of size s for f, we get an ABP $B_{\widehat{f}}$ for $\widehat{f}$ by putting an edge labeled y in series with every edge labeled z_i in B_f, so that $B_{\widehat{f}}$ computes yz_i at every place where B_f uses z_i. Hence, the size of $B_{\widehat{f}}$ is at most $2s$.

3.7. Solving a linear system with polynomials as matrix entries. We show how to solve a linear system $M\boldsymbol{v} = 0$ for a polynomial matrix M with entries from $\mathbb{F}[\boldsymbol{z}]$ given as ABPs. We are seeking for a nonzero vector $\boldsymbol{v}$. Note that such a $\boldsymbol{v}$ exists over the ring $\mathbb{F}[\boldsymbol{z}]$ iff it exists over the field $\mathbb{F}(\boldsymbol{z})$.

Except for minor modifications, this follows from classical linear algebra. Kopparty *et al.* (2015, Lemma 2.6) have shown the same result for circuits. The proof works as well for ABPs.

LEMMA 3.11 (Solving linear systems (Kopparty *et al.* 2015)). *Let $M = (m_{i,j}(\boldsymbol{z}))_{i,j}$ be a polynomial matrix of dimension $k \times m$ and variables $\boldsymbol{z} = (z_1, \ldots, z_n)$, where the entries are polynomials $m_{i,j} \in \mathbb{F}[\boldsymbol{z}]$ that can be computed by ABPs of size s.*

Then, there is an ABP of size $\mathrm{poly}(k, m, s)$ *that computes a nonzero vector $\boldsymbol{v} \in \mathbb{F}[\boldsymbol{z}]^m$ such that $M\boldsymbol{v} = 0$, if it exists.*

PROOF. After swapping rows of M, we ensure that the $j \times j$ submatrix M_j that consists of the first j rows and the first j columns has full rank, iteratively for $j = 1, 2, \ldots$.

For $j = 1$, this means to find a nonzero entry in the first column and swap that row with the first row. If the first column is a zero-column, then $\boldsymbol{v} = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^T$ is a solution and we are done. To extend from j to $j+1$, suppose we have ensured that M_j has full rank. Now we search for a row from row $j+1$ on, such that after a swap with row $j+1$, the submatrix M_{j+1} has full rank, i.e., its determinant is nonzero. This can be tested by Theorem 2.1. If no such row exists, then the process stops at j. If $j = m$, then M has full rank and the zero vector is the only solution. Otherwise, assume the above process stops with $j < m$.

Now Cramer's rule can be used to find the unique solution $\boldsymbol{u} = \begin{pmatrix} u_1 & u_2 & \cdots & u_j \end{pmatrix}^T$ of the system

$$M_j \boldsymbol{u} = \begin{pmatrix} m_{1,j+1} & m_{2,j+1} & \cdots & m_{j,j+1} \end{pmatrix}^T . \tag{3.12}$$

We have $u_i = \frac{\det M_j^i}{\det M_j}$, where M_j^i is the matrix obtained by replacing the ith column of M_j by the vector $\begin{pmatrix} m_{1,j+1} & \cdots & m_{j,j+1} \end{pmatrix}^T$. Now, define

$$\boldsymbol{v} = \begin{pmatrix} \det M_j^1 & \det M_j^2 & \cdots & \det M_j^j & -\det M_j & 0 & \cdots & 0 \end{pmatrix}^T .$$

Then, $\boldsymbol{v}$ is a nonzero solution to the original system. Note that the vector on the right-hand side of (3.12) might be the zero vector, in which case also $\boldsymbol{u}$ will be the zero vector. Still $\boldsymbol{v}$ is nonzero because it has the nonzero entry $-\det M_j$ at position $j+1$.

The entries of $\boldsymbol{v}$ are determinants of matrices with entries computed by ABPs of size s. Hence, all the entries of $\boldsymbol{v}$ have ABPs of size $\mathrm{poly}(k, m, s)$. □

Remark 3.13. *The ABP in Lemma 3.11 can be constructed by a randomized algorithm in time* $\mathrm{poly}(k, m, s)$. *Randomization comes in by Theorem 2.1 to compute the matrices* M_j. *In fact, the determinant polynomials we get for PIT can be computed by ABPs. Hence, the construction algorithm can be made deterministic with an oracle for ABP–PIT.*

4. Factors of arithmetic branching programs

In this section, we prove that ABPs are closed under factoring over fields of characteristic 0. Over fields of characteristic p, our proof fails if one of the irreducible factors has multiplicity $e > 0$, where p divides e.

Theorem 4.1. *Let* p *be a polynomial over a field* $\mathbb{F}$ *with characteristic* 0. *For all factors* q *of* p, *we have*

$$\mathrm{size}_{\mathrm{ABP}}(q) \leq \mathrm{poly}(\mathrm{size}_{\mathrm{ABP}}(p)) .$$

We prove Theorem 4.1 in the rest of this section. First, observe that it suffices to prove the $\mathrm{poly}(s)$ size upper bound for the irreducible factors of p. This also yields a $\mathrm{poly}(s)$ bound for all the factors.

In case when p is irreducible there is nothing to show, because the only factor, p, has a small ABP by assumption. The case when $p = q^e$ is proved in Section 3.2. So it remains to consider the general case when $p = p_1^{e_1} \cdots p_m^{e_m}$, for $m \geq 2$, where $p_1, \ldots, p_m$ are the different irreducible factors of p. We want to prove an ABP-size upper bound for an irreducible factor, say p_1.

We start by several transformations on the input polynomial $p(\boldsymbol{z})$, where $\boldsymbol{z} = (z_1, \ldots, z_n)$.

1. As described in Section 3.3, taking $k = e_1 - 1$ times the derivative w.r.t. some variable, say z_1, we get the polynomial $\widehat{p}(\boldsymbol{z}) = \frac{\partial^k p(\boldsymbol{z})}{\partial z_1^k}$, where the factor p_1 has multiplicity 1.

2. Next, by Lemma 3.7, we transform $\widehat{p}(\boldsymbol{z})$ to a polynomial $\widetilde{p}(x, \boldsymbol{z})$ that is monic in x, for a new variable x. Thereby, also the factors of $\widehat{p}(\boldsymbol{z})$ are transformed, maintaining their irreducibility and multiplicity. The degree of $\widetilde{p}$ is twice the degree of $\widehat{p}$.

3. At this point, we may have to shift the variables $\boldsymbol{z}$ as described in Section 3.5 to ensure the properties needed for starting Hensel lifting. This shift preserves the monicness and the irreducibility of the factors.

4. Finally, the transformation to a bivariate polynomial is explained in Section 3.6. This yields polynomial $P(x, y, \boldsymbol{z})$, with new variable y and monic in x. We rewrite P as a polynomial in x and y with coefficients in the ring $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$ and call the representation f. That is, $f(x, y) \in \mathbb{K}[x, y]$. By Lemma 3.10, the transformation maintains irreducible factors. Note also that by the definition of P, we have $f(x, 0) = P(x, 0, 0, \ldots, 0) = f(x, y) \bmod y$, so that $f(x, y) \bmod y$ is univariate.

The main part now is to factor $f(x, y) \in \mathbb{K}[x, y]$, say $f = gh$, where $g \in \mathbb{K}[x, y]$ is irreducible and coprime to $h \in \mathbb{K}[x, y]$, and f, g, h are monic in x and have x-degree ≥ 1. Let d be the total degree of f in x, y.

From the factor g of f, we will recover the factor p_1 of p by reversing the above transformations. We show that g can be computed by an ABP of size poly(s). It follows that the irreducible factor p_1 has an ABP of size poly(s).

The strategy is to first factor the univariate polynomial $f \bmod y$, and then apply Hensel lifting to get a factorization of $f \bmod y^t$, for large enough t. Finally, from the lifted factors modulo y^t, we compute the absolute factors of f.

4.1. Hensel lifting. Hensel lifting is named after Kurt Hensel (Hensel 1899, 1904, 1908, 1918). Predecessors of the technique

were already known to Gauß, see Frei (2005) for a very detailed outline of the historical development. There are various versions and descriptions of Hensel lifting in the literature, see, for example, Sudan (1998, Lecture 7). In our case, an ABP should be able to perform several iterations of the lifting. Therefore, we use the lifting in a way suitable for ABPs. In particular, in contrast to other presentations, we will *not* maintain the monicness of the lifted factors.

Hensel lifting works over rings $\mathcal{R}$ modulo an ideal $\mathcal{I} \subseteq \mathcal{R}$. In our case, $\mathcal{R} = \mathbb{K}[x, y]$, where $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$, and $\mathcal{I} = \langle y \rangle^k$, for some $k \geq 1$.

Definition 4.2 (Lifting). *Let $\mathcal{R}$ be a ring and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal. Let $f, g, h, a, b \in \mathcal{R}$ such that $f \equiv gh \pmod{\mathcal{I}}$ and $ag + bh \equiv 1 \pmod{\mathcal{I}}$. Then, we call $g', h' \in \mathcal{R}$* a lift of g, h with respect to f and $\mathcal{I}$, *if*

(i) $f \equiv g'h' \pmod{\mathcal{I}^2}$,

(ii) $g' \equiv g \pmod{\mathcal{I}}$ *and* $h' \equiv h \pmod{\mathcal{I}}$, *and*

(iii) $\exists a', b' \in \mathcal{R} \quad a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$.

Remark 4.3. *(i) We will skip mentioning f or $\mathcal{I}$ in a lift when it is clear from the context.*

(ii) The three conditions in Definition 4.2 are the invariants *when iterating the lifting.*

(iii) Note that condition (iii) is actually redundant. It already follows from the assumptions together with the second condition. This can be seen in the proof of Lemma 4.4, where a lift g', h' from g, h is constructed, together with a', b'. When we show that condition (iii) holds, we do not *use the specific form of g', h' constructed there, and it suffices to have condition (ii).*

The following lemma presents a method for lifting that is usually attributed to Hensel. There is also a certain *uniqueness property*: Note that for any $u \in \mathcal{I}$, we have $(1+u)(1-u) \equiv 1 \pmod{\mathcal{I}^2}$.

Hence, when g', h' are a lift of g, h, $g'(1+u)$, $h'(1-u)$ are another lift of g, h. The uniqueness property states that there are no other lifts than these.

LEMMA 4.4 (Hensel Lifting). *Let $\mathcal{R}$ be a ring and $\mathcal{I} \subseteq \mathcal{R}$ be an ideal. Let $f, g, h, a, b \in \mathcal{R}$ such that $f \equiv gh \pmod{\mathcal{I}}$ and $ag+bh \equiv 1 \pmod{\mathcal{I}}$. Then, we have*

(i) (Existence). *There exists a lift g', h' of g, h.*

(ii) (Uniqueness). *For any other lift g^*, h^* of g, h, there exists a $u \in \mathcal{I}$ such that*

$$g^* \equiv g'(1+u) \pmod{\mathcal{I}^2} \quad \text{and} \quad h^* \equiv h'(1-u) \pmod{\mathcal{I}^2}.$$

PROOF. We first show the existence part. Let

1. $e = f - gh$,
2. $g' = g + be$ and $h' = h + ae$,
3. $c = ag' + bh' - 1$,
4. $a' = a(1-c)$ and $b' = b(1-c)$.

We verify that g', h' are a lift of g, h. Because $f \equiv gh \pmod{\mathcal{I}}$, we have $e = f - gh \equiv 0 \pmod{\mathcal{I}}$. In other words, $e \in \mathcal{I}$. Hence, we get condition (ii) that $g' \equiv g \pmod{\mathcal{I}}$ and $h' \equiv h \pmod{\mathcal{I}}$.

Next, we show condition (i) that $f \equiv g'h' \pmod{\mathcal{I}^2}$.

$$\begin{aligned} f - g'h' &= f - (g+be)(h+ae) \\ &= f - gh - e(ag+bh) - abe^2 \\ &\equiv e - e(ag+bh) \pmod{\mathcal{I}^2} \\ &\equiv e(1-(ag+bh)) \pmod{\mathcal{I}^2} \\ &\equiv 0 \pmod{\mathcal{I}^2}. \end{aligned}$$

In the second line, note that $e^2 \in \mathcal{I}^2$. The last equality holds because $e \in \mathcal{I}$ and $1-(ag+bh) \in \mathcal{I}$.

To show condition (iii), we verify that $a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$. First, observe that

$$\begin{aligned} c &= ag' + bh' - 1 \\ &\equiv ag + bh - 1 \pmod{\mathcal{I}} \\ &\equiv 0 \pmod{\mathcal{I}}. \end{aligned}$$

Hence, $c \in \mathcal{I}$ and we conclude that $a' \equiv a \pmod{\mathcal{I}}$ and $b' \equiv b \pmod{\mathcal{I}}$. Now,

$$\begin{aligned} a'g' + b'h' - 1 &= a\,(1-c)g' + b\,(1-c)h' - 1 \\ &= ag' + bh' - 1 - c\,(ag' + bh') \\ &= c - c\,(ag' + bh') \\ &= c\,(1 - (ag' + bh')) \\ &= -c^2 \\ &\equiv 0 \pmod{\mathcal{I}^2}. \end{aligned}$$

For the uniqueness part, let g^*, h^* be another lift of g, h. Let $\alpha = g^* - g'$ and $\beta = h^* - h'$. By Definition 4.2 (ii), we have $g' \equiv g \equiv g^* \pmod{\mathcal{I}}$ and $h' \equiv h \equiv h^* \pmod{\mathcal{I}}$ and therefore $\alpha, \beta \in \mathcal{I}$.

We first show

$$\beta g' + \alpha h' \equiv 0 \pmod{\mathcal{I}^2}. \tag{4.5}$$

$$\begin{aligned} \beta g' + \alpha h' &= \beta g' + (g^* - g')h' \\ &= \beta g' + g^* h' - g' h' \\ &\equiv \beta g' + g^* h' - g^* h^* \pmod{\mathcal{I}^2} \\ &\equiv \beta g' - \beta g^* \pmod{\mathcal{I}^2} \\ &\equiv -\alpha\beta \pmod{\mathcal{I}^2} \\ &\equiv 0 \pmod{\mathcal{I}^2}. \end{aligned}$$

Define $u = a'\alpha - b'\beta$. Because $\alpha, \beta \in \mathcal{I}$, also $u \in \mathcal{I}$. Then,

by (4.5) and because $a'g' + b'h' \equiv 1 \pmod{\mathcal{I}^2}$, we have

$$\begin{aligned} g'(1+u) &= g'(1+(a'\alpha - b'\beta)) \\ &= g' + a'g'\alpha - b'g'\beta \\ &\equiv g' + a'g'\alpha + b'h'\alpha \pmod{\mathcal{I}^2} \\ &\equiv g' + \alpha \pmod{\mathcal{I}^2} \\ &\equiv g^* \pmod{\mathcal{I}^2}. \end{aligned}$$

Similarly, we get $h^* \equiv h'(1-u) \pmod{\mathcal{I}^2}$. □

For the ABP-size, recall that the size just adds up when doing additions or multiplications. Hence, when f, g, h, a, b have ABPs of size $\leq s$ and we construct ABPs for g', h', a', b' according to steps 1–4 in the above proof, we get ABPs of size $O(s)$. In more detail, the reader may verify that the ABPs for g' and h' have size $\leq 4s$ and the ABPs for a' and b' have size $\leq 10s$.

Similarly, with respect to the degree, when f, g, h, a, b have degree $\leq d$, g', h', a', b' have degree $O(d)$. Namely, g', h' have degree $\leq 3d$, and a', b' have degree $\leq 5d$.

Remark 4.6. *In the* monic version *of Hensel lifting, there is a division in addition to the four steps from above. When we assume that g is monic, we can compute polynomials q and r such that $g' - g = qg + r$, where $\deg_x(r) < \deg_x(g)$. Then, one can show that $\widehat{g} = g + r$ and $\widehat{h} = h'(1+q)$ are a lift of g, h w.r.t. f, and $\widehat{g}$ is again monic. Also the Bézout-coefficients $\widehat{a}, \widehat{b}$ can be computed. For $\widehat{c} = a\widehat{g} + b\widehat{h} - 1$, let $\widehat{a} = a(1-\widehat{c})$ and $\widehat{b} = b(1-\widehat{c})$.*

An advantage of the monic version is that the result is really unique (modulo $\mathcal{I}^2$). There is no $1+u$ factor between monic lifts. A disadvantage is the extra division which would blow up the ABP-size too much.

4.2. Iterating Hensel lifting. We apply Hensel lifting iteratively in the ring $\mathcal{R} = \mathbb{K}[x, y]$, where $\mathbb{K} = \mathbb{F}[\boldsymbol{z}]$. Let $f \in \mathbb{K}[x, y]$ be a polynomial of total degree d in x, y that can be factored into $f = gh$, where $g \in \mathbb{K}[x, y]$ is irreducible and coprime to $h \in \mathbb{K}[x, y]$, and f, g, h are monic in x and have x-degree ≥ 1.

To start the Hensel lifting procedure, we factor the univariate polynomial $f(x,0) = f \bmod y$ as $f(x,0) = g_0(x)\, h_0(x)$, where g_0 is a divisor of $g \bmod y$, and coprime to h_0 and $\deg_x(g_0) \geq 1$. Recall that by the preprocessing in Section 3.5, we may assume that there is such a decomposition of $f(x,0)$.

By the Euclidean algorithm, there are polynomials $a_0(x), b_0(x)$ such that $a_0 g_0 + b_0 h_0 = 1$. Hence, we have $a_0 g_0 + b_0 h_0 \equiv 1 \pmod{I_0}$, for $\mathcal{I}_0 = \langle y \rangle$, and initiate Hensel lifting with

$$f \equiv g_0 h_0 \pmod{\mathcal{I}_0}.$$

The ABP-size of g_0, h_0, a_0, b_0 is bounded by the ABP-size of f, actually by $\deg_x(f)$, because we have univariate polynomials here.

We iteratively apply Hensel lifting to g_0, h_0 as described in the proof of Lemma 4.4. Each time, the ideal gets squared. For $k \geq 1$, let $\mathcal{I}_k = \mathcal{I}_0^{2^k}$. That is, we get polynomials g_k, h_k such that

$$f \equiv g_k h_k \pmod{\mathcal{I}_k},$$

and g_k, h_k are a lift of g_{k-1}, h_{k-1} w.r.t. f and $\mathcal{I}_{k-1}$.

For the ABP-size of g_k and h_k, we observed at the end of Section 4.1 that the size increases by a constant factor in each iteration. Hence, when we start with $\mathrm{size}_{\mathrm{ABP}}(f) = s$, after k iterations, we get $\mathrm{size}_{\mathrm{ABP}}(g_k), \mathrm{size}_{\mathrm{ABP}}(h_k) = s\, 2^{O(k)}$.

Similarly, the degree of the lifted polynomials increases by a constant factor in each iteration. We start with $\deg(f) = d$. Then, after k iterations, we get $\deg(g_k), \deg(h_k) = d\, 2^{O(k)}$.

The following lemma states that g_k divides g modulo $\mathcal{I}_k$, for all $k \geq 0$. In a sense, the g_k's approximate a factor of g modulo increasing powers of y.

Lemma 4.7. *With the notation from above, for all $k \geq 0$ and some polynomial h'_k,*

$$g \equiv g_k h'_k \pmod{\mathcal{I}_k} \quad \text{and} \quad h_k \equiv h\, h'_k \pmod{\mathcal{I}_k}.$$

Moreover, g_k, h'_k are a lift of g_{k-1}, h'_{k-1} w.r.t. g, for $k \geq 1$, and $\deg_x(h'_k \bmod \mathcal{I}_k) \leq \deg_x(h_k)$.

PROOF. The proof is by induction on $k \geq 0$. For the base case, we have that g_0 divides g modulo I_0, as explained above. Thus, for some polynomial h_0' that is coprime to g_0, we have

$$g \equiv g_0 h_0' \pmod{\mathcal{I}_0}.$$

Hence, we have $h_0 \equiv h_0' h \pmod{\mathcal{I}_0}$. Note that h_0' might be just 1.

For the inductive step, assume that

(4.8)
$$g \equiv g_{k-1} h_{k-1}' \pmod{\mathcal{I}_{k-1}} \quad \text{and} \quad h_{k-1} \equiv h\, h_{k-1}' \pmod{\mathcal{I}_{k-1}}.$$

Let g_k', h_k'' be a lift of g_{k-1}, h_{k-1}' w.r.t. g and $\mathcal{I}_{k-1}$, so that in particular

$$g_k' h_k'' \equiv g \pmod{\mathcal{I}_k}. \tag{4.9}$$

We claim that then g_k', $h\, h_k''$ are a lift of g_{k-1}, $h\, h_{k-1}'$, i.e., of g_{k-1}, h_{k-1} by (4.8), w.r.t. f.

CLAIM 4.10. *g_k', $h\, h_k''$ are a lift of g_{k-1}, h_{k-1} w.r.t. f and $\mathcal{I}_{k-1}$.*

PROOF. We check the three conditions for a lift in Definition 4.2. For the product condition (i), we have by (4.9)

$$g_k'\, h\, h_k'' = (g_k'\, h_k'')\, h \equiv gh \equiv f \pmod{\mathcal{I}_k}.$$

For condition (ii), we have $g_k' \equiv g_{k-1} \pmod{\mathcal{I}_{k-1}}$ by assumption and similarly

$$h\, h_k'' \equiv h\, h_{k-1}' \equiv h_{k-1} \pmod{\mathcal{I}_{k-1}}.$$

By Remark 3 after Definition 4.2, condition (iii) already follows now. This proves Claim 4.10. □

Recall that also g_k, h_k are a lift of g_{k-1}, h_{k-1}. Hence, by the uniqueness property of Hensel lifting, there is a $u \in \mathcal{I}_{k-1}$ such that

(4.11)
$$g_k' \equiv g_k\,(1+u) \pmod{\mathcal{I}_k} \quad \text{and} \quad h\, h_k'' \equiv h_k\,(1-u) \pmod{\mathcal{I}_k}.$$

Now observe that we can move the factor $1+u$: We have that $g_k\,(1+u)$, $h\, h_k''$ are a lift of g_{k-1}, h_{k-1} and then also g_k, $h\, h_k''\,(1+u)$ are a lift of g_{k-1}, h_{k-1}.

Claim 4.12. *$g_k,\ h\,h_k''\,(1+u)$ are a lift of g_{k-1}, h_{k-1} w.r.t. f and $\mathcal{I}_{k-1}$.*

Proof. We check the conditions for a lift in Definition 4.2. The first two of them are trivial: Moving the factor $1+u$ clearly does not change the product. Because $u \in \mathcal{I}_{k-1}$, we still have the equality with the factors g_{k-1} and h_{k-1} modulo $\mathcal{I}_{k-1}$, respectively.

By the remark after Definition 4.2, the third condition already follows, but it is also easy to check now:

Let $a, b \in \mathcal{R}$ such that $ag_k + bh_k \equiv 1 \pmod{\mathcal{I}_k}$. It follows by Equation (4.11) that

$$\begin{aligned} ag_k + bh\,h_k''(1+u) &\equiv ag_k + bh_k(1-u)(1+u) \pmod{\mathcal{I}_k} \\ &\equiv ag_k + bh_k(1-u^2) \pmod{\mathcal{I}_k} \\ &\equiv 1 \pmod{\mathcal{I}_k}. \end{aligned}$$

This proves Claim 4.12. □

Now, define $h_k' = h_k''(1+u)$. Note that

$$h_k' \equiv h_k'' \equiv h_{k-1}' \pmod{\mathcal{I}_{k-1}}. \tag{4.13}$$

By (4.11), we have

$$h\,h_k' \equiv h\,h_k''\,(1+u) \equiv h_k(1-u)(1+u) \equiv h_k \pmod{\mathcal{I}_k}. \tag{4.14}$$

By (4.14), we have

$$f = gh \equiv g_k h_k \equiv g_k h\,h_k' \pmod{\mathcal{I}_k}. \tag{4.15}$$

It follows from (4.15) that $gh \equiv g_k h_k' h \pmod{\mathcal{I}_k}$. Now we want to cancel h in the last equation and conclude that $g \equiv g_k h_k' \pmod{\mathcal{I}_k}$. This we can do because h is monic in x, it does not contain a factor y, i.e., $h \notin \mathcal{I}_0$. Hence, together with (4.13), we conclude that g_k, h_k' are a lift of g_{k-1}, h_{k-1}' w.r.t. g.

For the x-degree of h_k', consider the equation $h_k \equiv h\,h_k' \pmod{\mathcal{I}_k}$. Since h is monic in x, the highest x-degree term in the product $h\,(h_k' \bmod \mathcal{I}_k)$ will survive the modulo operation. Therefore, $\deg_x(h_k' \bmod \mathcal{I}_k) \le \deg_x(h) + \deg_x(h_k' \bmod \mathcal{I}_k) = \deg_x(h_k)$. □

4.3. Factor reconstruction for ABP. We show how to get the absolute factor g of f from the lifted factor. This is called the *jump step* in Sudan's lecture notes (Sudan 1998). The difference to the earlier presentations is that our lifted factor might not be monic.

Let $f = gh$, where f has total degree d, factor g is irreducible and coprime to h, and f, g, h are monic in x. In the previous section, we started with a factorization $f \equiv g_0 h_0 \pmod{\mathcal{I}_0}$, where g_0 is irreducible and coprime to h_0. Moreover, $g \equiv g_0 h_0' \pmod{\mathcal{I}_0}$, for some h_0' such that $h_0 = h\, h_0' \pmod{\mathcal{I}_0}$.

Then, we apply Hensel lifting, say t-times. We will see below that it suffices for our purpose to have

$$2^t \geq 2d^2 + 1.$$

Hence, we define $t = \lceil \log(2d^2 + 1) \rceil$. By Lemma 4.7, we get a factorization $f \equiv g_t h_t \pmod{\mathcal{I}_t}$ such that

$$g \equiv g_t h_t' \pmod{\mathcal{I}_t}, \tag{4.16}$$

for some h_t' such that $h_t \equiv h\, h_t' \pmod{\mathcal{I}_t}$.

Equation (4.16) gives us a relation between the known g_t and the unknown g, via the unknown h_t'. We set up a linear system of equations to find a polynomial $\widetilde{g} \in \mathbb{K}[x, y]$ with the same x-degree as g, such that

$$\widetilde{g} \equiv g_t \widetilde{h} \pmod{\mathcal{I}_t}, \tag{4.17}$$

for some polynomial $\widetilde{h}$. We give some more details to the linear system next.

Details for setting up the linear system. Equation (4.17) can be used to set up a homogeneous system of linear equations. For the degree bounds of the polynomials, let $d_x = \deg_x(g)$ and $d_y = \deg_y(g)$. Let $D_x = \deg_x(g_t \bmod \mathcal{I}_t)$ and $D_y = 2^t - 1$. Let

$D'_x = \deg_x(h_t)$. Let

$$g_t \equiv \sum_{i\le D_x,\ j\le D_y} c_{i,j}\, x^i y^j \pmod{\mathcal{I}_t},$$
$$\widetilde{g} = r_{d_{x,0}}\, x^{d_x} + \sum_{i<d_x,\ j\le d_y} r_{i,j}\, x^i y^j,$$
$$\widetilde{h} = \sum_{i\le D'_x,\ j\le D_y} s_{i,j}\, x^i y^j,$$

for coefficients $c_{i,j}, r_{i,j}, s_{i,j} \in \mathbb{K} = \mathbb{F}[\boldsymbol{z}]$.

Since we are working with $g_t \bmod \mathcal{I}_t$, we do not consider the terms with powers y^k, where $y^k \in \mathcal{I}_t$. Similarly, recall from Lemma 4.7 that $\deg_x(h'_t \bmod \mathcal{I}_t) \le \deg_x(h_t) = D'_x$. Therefore, we set up coefficients for $\widetilde{h}$ only up to x-degree D'_x.

Since we have an ABP that computes g_t, there are ABPs for computing the coefficients $c_{i,j}$ of g_t by Lemma 3.1. The coefficients $r_{i,j}, s_{i,j}$ of $\widetilde{g}$ and $\widetilde{h}$ we treat as unknowns. Equation (4.17) now becomes

$$r_{d_{x,0}}\, x^{d_x} + \sum_{i<d_x,\ j\le d_y} r_{i,j} x^i y^j \equiv \sum_{i\le D_x,\ j\le D_y} c_{i,j} x^i y^j \sum_{i\le D'_x,\ j\le D_y} s_{i,j} x^i y^j \pmod{y^{2^t}}. \tag{4.18}$$

Now we equate the coefficients of the monomials $x^k\, y^l$ on both sides in (4.18), for all k, l that occur in (4.18) such that $l \le D_y$. By restricting the exponent of y to D_y, the $(\bmod\ \mathcal{I}_t)$-operation is already implemented. The equations we get are now absolute equations, without modulo operations.

We get a homogeneous system of $(D_x + D'_x + 1)(D_y + 1)$ many equations in $1 + d_x(d_y + 1) + (D'_x + 1)(D_y + 1)$ many unknowns $r_{i,j}$ and $s_{i,j}$. This system can be expressed in the form $M\boldsymbol{v} = 0$, for a matrix M and unknown vector $\boldsymbol{v}$. By Lemma 3.11, an ABP can efficiently compute a nonzero solution vector $\boldsymbol{v}$ of polynomials from $\mathbb{F}[\boldsymbol{z}]$. Note that by (4.16), a non-trivial solution is guaranteed to exist.

Obtaining g from $\widetilde{g}$. Recall that g is monic, whereas we put leading coefficient $r_{d_{x,0}} \in \mathbb{F}[\boldsymbol{z}]$ at x^{d_x} in $\widetilde{g}$. The reason to do so is that we want the linear system to be homogeneous, which would not be the case when we would fix the coefficient to be 1. Hence, our solution $\widetilde{g}$ might not be monic.

The following lemma shows that when we divide $\widetilde{g}$ by its leading coefficient $r_{d_{x,0}}$, we get precisely g.

LEMMA 4.19. *Let $\widetilde{g}$ be a solution of (4.18) with leading x-coefficient $r_{d_{x,0}} \in \mathbb{F}[\boldsymbol{z}]$, for $t = \lceil \log(2d^2+1) \rceil$. Then,*

$$g = \frac{\widetilde{g}}{r_{d_{x,0}}}.$$

PROOF. Consider the resultant $r(y) = \mathrm{Res}_x(g, \widetilde{g})$. We show that $r(y) = 0$. Then, it follows from Lemma 2.2 that g and $\widetilde{g}$ share a common factor with positive x-degree. Since g is irreducible, it must be a divisor of $\widetilde{g}$. As g is monic, we have $\widetilde{g} = r_{d_{x,0}}\, g$ as claimed. As g and $\widetilde{g}$ have the same x-degree, $r_{d_{x,0}}$ is a polynomial in $\mathbb{F}[\boldsymbol{z}]$.

To argue that $r(y) = 0$, recall from Lemma 2.2 that the resultant can be written as $r(y) = ug + v\widetilde{g}$, for some polynomials u and v. Since $\deg(g), \deg(\widetilde{g}) \leq d$, we have $\deg(r) \leq 2d^2$. By (4.16) and (4.17), we have

$$ug + v\widetilde{g} \equiv g_t(uh'_t + v\widetilde{h}) \pmod{\mathcal{I}_t}$$

Consider g_t and $w = uh'_t + v\widetilde{h}$ as polynomials in y with coefficients in x. Let

$$g_t \equiv c_0(x) + c_1(x)y + \cdots + c_{D_y}(x)y^{D_y} \pmod{\mathcal{I}_t},$$

where $c_i \in \mathbb{K}[x]$, for $i = 0, 1, \ldots, D_y$ and $D_y = 2^t - 1$. By the properties of Hensel lifting, we have $g_t \equiv g_0 \pmod{\mathcal{I}_0}$, and therefore, $c_0(x) = g_0(x)$. Recall that g_0 is non-constant, $\deg(g_0) \geq 1$.

Now consider w. Let $j \geq 0$ be the least power of y that appears in w and let its coefficient be $w_j(x)$. Suppose for the sake of contradiction that $j < 2^t$. Then, the least power of y in $g_t w$ is also j, and its coefficient is $g_0(x)w_j(x)$, which is a nonzero polynomial in x.

The monomials present in $g_0(x)w_j(x)y^j$ cannot be canceled by other monomials in $g_t w$ because they have larger y-degree. It follows that $g_t w \bmod \mathcal{I}_t$ is not free of x. On the other hand, $r(y) \equiv g_t w \pmod{\mathcal{I}_t}$ and $r(y) \in \mathbb{K}[y]$ is a polynomial with no variable x. This is a contradiction.

We conclude that $j \geq 2^t$, which means that $w \equiv 0 \pmod{\mathcal{I}_t}$. Hence, we get $r(y) \equiv 0 \pmod{\mathcal{I}_t}$. Recall that $\deg_y(r) \leq 2d^2$ and $2^t > 2d^2$. Hence, the $\pmod{\mathcal{I}_t}$-operation has no effect here and we can conclude that indeed $r(y) = 0$. □

The final division to obtain an ABP for g can be accomplished by adapting Strassen's division elimination for ABPs. The size increase is polynomial in the ABP-size of $\widetilde{g}$.

Remark 4.20. *The ABPs for $\widetilde{g}$ and g can also be algorithmically constructed. One point to notice here is that when we set up the linear system above, we used the degrees $d_x = \deg_x(g)$ and $d_y = \deg_y(g)$ of g that we actually do not have in hand at that point. We just have the degree d of f as an upper bound. So algorithmically, we will search for the degree. That is, we set up linear systems with*

$$\widetilde{g} \;=\; r_{\widetilde{d}_x,0}\, x^{\widetilde{d}_x} + \sum_{i<\widetilde{d}_x,\ j\leq\widetilde{d}_y} r_{i,j}\, x^i y^j \,,$$

for increasing values $\widetilde{d}_x, \widetilde{d}_y = 1, 2, \ldots, d$. The resultant argument in Lemma 4.19 shows that g is a divisor of $\widetilde{g}$. Hence, the minimal value for $\widetilde{d}_x$ where we get a nonzero solution $\widetilde{g}$ will be the right value, i.e., when $\widetilde{d}_x = d_x$.

A special case is when f is irreducible. In this case, we have $g = f$, and equivalently, $d_x = \deg_x(f)$. Hence, we can detect this case by comparing the x-degrees of f and g. Note that Lemma 4.19 also holds in this case. The linear system (4.18) is set up such that the solution $\widetilde{g}$ has $\deg(\widetilde{g}) \leq d$. This suffices to argue that factor g, in this case f, is a divisor of $\widetilde{g}$, and hence, we compute g which is equal to f.

4.4. Size analysis. We summarize the bound on the ABP-size of the factor computed. Given polynomial p of degree d_p and $\mathrm{size}_{\mathrm{ABP}}(p) = s$, we have seen that the preprocessing transformations

yield a polynomial f of degree $d_f \leq 2d_p$ and $\text{size}_{\text{ABP}}(f) = \text{poly}(s)$. Then, we do $t = \log(2d_f^2 + 1)$ iterations of Hensel lifting. The initial polynomials f_0, g_0, h_0 have ABP-size bounded by $2d_f$. Hence, the polynomials after the last iteration have ABP-size bounded by $2^t \,\text{poly}(s) = \text{poly}(s, d_p) = \text{poly}(s)$.

From the lifted factor, we construct the actual factor of f. This step involves solving a linear system. We argued that the resulting polynomial g has ABP-size $\text{poly}(s)$.

Finally, we reverse the transformations from the beginning and get a factor of p that has an ABP of size $\text{poly}(s)$. This finishes the proof of Theorem 4.1.

5. Construction algorithm and reduction to PIT

Theorem 4.1 is non-constructive, and it states the *existence* of small size ABPs for the factors of a polynomial. In this section, we argue that we can efficiently *construct* the ABPs for the factors by a randomized algorithm. In fact, all randomized steps of the algorithm are PITs for polynomials computed by small ABPs. Hence, the algorithm can be made deterministic when it is equipped with a (functional) oracle for PIT for ABPs. This is analogous to what Kopparty, Saraf & Shpilka (2015) showed for arithmetic circuits.

THEOREM 5.1. *Given a polynomial $p(\boldsymbol{z})$ computed by an ABP of size s, there is a randomized* $\text{poly}(s)$*-time algorithm to compute all its irreducible factors represented as ABPs. Moreover, the factorization problem reduces in* $\text{poly}(s)$*-time to PIT for ABPs.*

We already mentioned for many steps that they are constructive. Here, we summarize the construction and fill the gaps.

Step 1: Transformation to monic. We modify the order of the steps as described in the proof of Theorem 4.1 and start with the transformation to make the input polynomial $p(\boldsymbol{z})$ monic in a new variable x, as described in Section 3.4. For the ease of notation, we still call the polynomial p, it is now $p(x, \boldsymbol{z})$, however.

Step 2: If $p = q^e$. Next, we consider the case of Section 3.2, i.e., when $p = q^e$, for some polynomial q and an integer $e \geq 2$. Note that to prove Theorem 4.1, we could simply assume that q is irreducible and that we know e. Now, we have also to find out algorithmically whether this is the case. What we do is, we try for all possible values of e that are divisors of d, starting from the maximum possible value d in decreasing order. When we reach $e = 1$, we proceed to step 3.

For each e, we compute the polynomial $q = p^{\frac{1}{e}}$ as described in Section 3.2. Now we have to check whether q is indeed a factor of p.

Recall that p is monic in x. Hence, a factor must be monic too. So if q is not monic, we can proceed to the next e. Otherwise, we test if q is a divisor of p. This we can do by using the classical Euclidean univariate long-division algorithm. Here, we need that p and q are monic polynomials in x. By Lemma 3.1, we can get the ABPs that compute the coefficients of the polynomials p and q w.r.t. x. Now, we can compute the ABPs that compute the quotient and the remainder in randomized polynomial time using univariate long division. To test whether the remainder is zero, we need a PIT for ABPs.

At this point, we have verified that $p = q^e$. Still, polynomial q might not be irreducible. To check this, we try to factorize q. That is, we restart the algorithm on q. Since e is maximum such that $p = q^e$, polynomial q itself cannot be of the form $q = h^{e'}$, for some $e' \geq 2$. Hence, for factoring q, we can directly go to step 3.

Step 3: Reducing the multiplicity of a factor. Now p is either an irreducible polynomial or a reducible polynomial of the form $p = q_1^{e_1} \cdots q_m^{e_m}$, for monic square-free polynomials $q_1, \ldots, q_m$, where $1 \leq e_1 \leq e_2 < \cdots \leq e_m \leq d$, for some $m \geq 2$.

If p is irreducible or all the irreducible factors have multiplicity one, then we proceed to step 4. Otherwise, to reduce the multiplicity, we take derivatives as described in Section 3.3. However, we do not know the exponents $e_1, e_2, \ldots, e_m$. Therefore, we take derivatives of p of order e, for all $1 \leq e \leq d-2$, and factorize the corresponding derivatives of p. Note that for $e = e_i - 1$, the eth

derivative contains the square-free factor q_i.

Recall that the derivative also contains other factors, that are not factors of p, see equation (3.6) on page 16. However, we can always check factors via the division subroutine mentioned in step 2.

Step 4: Preprocessing for Hensel lifting. We shift the variables as explained in Section 3.5 to fulfill the assumptions needed for Hensel lifting.

Step 5: Reduction to bivariate We introduce a new variable y as explained in Section 3.6 and consider the resulting polynomial $f(x, y)$ as bivariate with coefficients in $\mathbb{F}(\boldsymbol{z})$. For the current e from step 3, polynomial f contains all irreducible factors of p that have multiplicity $e - 1$.

Step 6: Hensel lifting. The univariate polynomial $f(x, 0)$ can be represented in the dense way. Note that its coefficients are over $\mathbb{F}$. We can use any known univariate factorization algorithm, like the famous LLL-algorithm over rationals. We try all the irreducible factors of multiplicity one of $f(x, 0)$ as starting point for g_0 as described in Section 4.2. Note that there are at most d irreducible factors of $f(x, 0)$. Hence, we can try all of them. We can use the extended Euclidean algorithm to compute the polynomials a_0 and b_0 such that $a_0 g_0 + b_0 h_0 = 1$.

Then, we apply Hensel lifting and linear system solving as described in Sections 4.2 and 4.3. This yields a factor, say f_1, of f.

Step 7: Factor test. We reverse the preprocessing steps for f_1 to get a candidate factor, say q, of the original polynomial p. For some choices made in previous steps, q is some other polynomial and not a factor of p. But this we can check by using the division algorithm. If q has the same degree as the original polynomial p, we conclude that p is irreducible.

Summary. The algorithm described above will efficiently compute all the irreducible factors of the original polynomial p. The

multiplicity of an irreducible factor q is the largest e from step 3 where we found q.

Randomness is used in steps 1 and 4, in step 6 for solving linear systems over $\mathbb{F}(\boldsymbol{z})$, and in steps 2 and 7 for testing divisibility. All these steps can be derandomized by a black-box ABP–PIT oracle, equivalently by explicit hitting sets for the class of polynomials in VBP.

Shpilka & Volkovich (2010) observed that also conversely, PIT reduces to factoring. Namely, for a polynomial $p(\boldsymbol{z})$, let x, y be new variables and define $\widehat{p}(x, y, \boldsymbol{z}) = p(\boldsymbol{z}) + xy$. Then, we have

$$p \equiv 0 \iff \widehat{p} \text{ is reducible.}$$

In all models, formulas, ABPs, and circuits, when p has size s, $\widehat{p}$ has size $s + 2$. This is for the white-box and the black-box case.

We do not get an equivalence however, since we have a white-box factoring algorithm and a reduction to black-box PIT: Recall from Section 3.5 that we considered the discriminant of the factor g that we do not have in hand at that point. Note that a black-box oracle is stronger than a white-box oracle.

Kopparty, Saraf & Shpilka (2015) showed that for circuits, factoring reduces to white-box PIT. The crux lies in the preprocessing. When we follow their method, we get similarly a reduction to white-box PIT for ABPs. We give a short sketch of the steps next. Since it involves computing GCDs, the preprocessing step is no longer feasible for formulas however.

Reduction to white-box PIT. Let $p(\boldsymbol{z})$ be the given polynomial over a field $\mathbb{F}$ of characteristic 0. First, we make the given polynomial $p(\boldsymbol{z})$ monic in a new variable x as described in Section 3.4. Then, we divide p by the GCD of p and its derivative w.r.t. some variable of p, to get the square-free part of p. The GCD has a small ABP by Lemma 2.3. Note that the division does not increase the size of ABP by much. Instead of factoring the original polynomial, we now factorize its square-free part, denoted by f.

Still we have the same problem to start with Hensel lifting as described in Section 3.5 for the factor g: Even though polyno-

mial $f(x, \boldsymbol{z})$ is square-free, $f(x, 0, \ldots, 0)$ may have repeated factors. By a similar argument as in Section 3.5, via the discriminant of f, we get that $f(x, \boldsymbol{\alpha})$ is square-free at a random point $\boldsymbol{\alpha}$, with high probability. Moreover, via a decision to search reduction for white-box PIT (Kopparty *et al.* 2015, Lemma 2.2), we can efficiently compute such an $\boldsymbol{\alpha}$.

Next, we reduce to a bivariate polynomial over a larger ring by using the transformation described in Section 3.6. This transformation does not use any randomness. The rest of the steps are Hensel lifting and reconstructing the factor using linear system solving as described in Sections 4.2 and 4.3. We can see that the final step of reconstruction can be derandomized by a white-box PIT oracle.

To determine all the irreducible factors, Kopparty, Saraf & Shpilka (2015) first find an irreducible factor, then divide by it and factor again. In the ABP-model, we cannot recursively divide. Instead, we start from all irreducible factors of the univariate polynomial and lift and reconstruct all the irreducible factors.

Black-box ABP factoring. Kaltofen & Trager (1990) introduced *black-box factoring*, where we only have black-box access to the input polynomial. The task is to construct black-boxes for the factors. That is, given any point, we have to output the evaluations of the irreducible factors at that point.

Kopparty, Saraf & Shpilka (2015) observed that the algorithm of Kaltofen and Trager can be seen as a deterministic reduction to black-box PIT for arithmetic circuits. One can ask whether this holds similarly for ABPs, i.e., given a black-box ABP for factoring, can it be reduced to black-box ABP–PIT.

In the case of circuits, this is achieved via an effective version of Hilbert's irreducibility theorem due to Kaltofen. In the proof of this theorem, *monic* Hensel lifting is used, see, for example, (Sudan 1998, Lecture 9) for an exposition. As elaborated in Section 4.1, it is not clear whether monic Hensel lifting can be done by small ABPs. Again, we can modify the proof to get along without the monicness property. Since there are many details to take care of, we will give a proof that black-box ABP factoring reduces to black-box ABP–PIT in a separate note.

Assumption on the underlying Field. Note that although we assume the underlying field to be of characteristic 0, our results extend to characteristic p as well, except for the case when a factor is of the form g^{p^k}, where g is irreducible. In that case, we would fail to output the factor g, as we cannot reduce the multiplicity here by taking derivative. If the input polynomial is $f = g^{p^k}h$ we would be able to efficiently output only the factors $G = g^{p^k}$ and h. It is an open question to show that g has a small formula/ABP/circuit, if g^{p^k} has a small formula/ABP/circuit (Kopparty *et al.* 2015). Also note that in characteristic p, the running time of the known deterministic univariate polynomial factoring algorithms depend on p. The randomized algorithms of Berlekamp and Cantor–Zassenhaus depend on $\log p$ and it is major open question to get a deterministic algorithm in $\text{poly}(\log p)$. To get a complete derandomization of multivariate factorization over characteristic p, one has to overcome these additional challenges beside the derandomization of PIT.

6. Applications

6.1. Root finding. Given a polynomial $p \in \mathbb{F}[x, \boldsymbol{z}]$, the *root finding* problem asks for a polynomial $r \in \mathbb{F}[\boldsymbol{z}]$ such that $p(r(\boldsymbol{z}), \boldsymbol{z}) = 0$. By a lemma of Gauß, r is a root of p iff $x - r(\boldsymbol{z})$ is an irreducible factor of p. By Theorem 4.1, when p is given by an ABP, we get an ABP for $x - r(\boldsymbol{z})$. Setting $x = 0$ and inverting the sign give an ABP for $r(\boldsymbol{z})$.

Corollary 6.1. *The solutions of the root finding problem for a polynomial p given by an ABP can be computed by ABPs of size* $\text{poly}(\text{size}_{\text{ABP}}(p))$.

Open question: Approximating power series roots. Assume that $\mathbb{F}$ is algebraically closed and of characteristic zero. Let $p \in \mathbb{F}[x, \boldsymbol{z}]$ be a multivariate polynomial of degree d that is monic in x. Then, with high probability after a random shift, polynomial p splits into power series roots from $\mathbb{F}[[\boldsymbol{z}]]$,

$$p = \prod_{i=1}^{d_r} (x - r_i(\boldsymbol{z}))^{e_i},$$

where $r_i \in \mathbb{F}[[\boldsymbol{z}]]$ and d_r is the degree of the square-free part, the *radical* of p, see, e.g., Dutta *et al.* (2018) for details. An obvious question now is whether we can approximate the roots r_i up to a given degree D.

QUESTION 6.2. *If $p(x, \boldsymbol{z})$ is given as an ABP or a formula of size s, can its power series roots r_i up to degree D be computed by ABPs or formulas of size* $\mathrm{poly}(s, D)$*?*

For *circuits*, the first author together with Dutta and Saxena (Dutta *et al.* 2018) showed that indeed the power series roots have circuits of size $\mathrm{poly}(s, d, D)$. They argue via Newton iteration. Recall that for ABPs and formulas, this yields only size bounds of $d^{\log D}$.

An obvious idea now how to answer the question would be to do non-monic Hensel lifting as described in Sections 4.1 and 4.2 with respect to powers of y, and do the lifting until the power of y exceeds degree D. However, it seems as the proof of Lemma 4.19 does not go through anymore. We cannot get that the degree of the resultant $r(y) = \mathrm{Res}_x(g, \widetilde{g})$ of the solution $\widetilde{g}$ to the linear system we set up and the approximate power series root g we want to recover, is less than the power 2^t of y in the ideal $\mathcal{I}_t$ we are working in.

6.2. Hardness versus randomness. As an application of Theorem 4.1, we get that lower bounds for ABPs imply a black-box derandomization of ABP–PIT, similar to the result of Kabanets & Impagliazzo (2004, Theorem 7.7) for arithmetic circuits.

THEOREM 6.3 (Hitting set from hard polynomial). *Let $\{q_m\}_{m \geq 1}$ be a multilinear polynomial family such that q_m is computable in time $2^{O(m)}$, but has no ABP of size $2^{o(m)}$. Then, one can compute a hitting set for ABPs of size s in time $s^{O(\log s)}$.*

The proof is similar to the proof given by Kabanets & Impagliazzo (2004, Theorem 7.7) for circuits. One can replace circuits by ABPs everywhere in the proof, except in place, where they invoke

Kaltofen's factor size upper bound for circuits. Now, this can be replaced by Theorem 4.1 for ABPs, and hence, the whole proof now works for ABPs as well, see Forbes & Shpilka (2015) or Kumar & Saptharishi (2019) for a quick overview and Kabanets & Impagliazzo (2004, Theorem 7.7) for the full proof.

7. Conclusion and open problems

We prove that the class of polynomials computed by ABPs is closed under factors. Our proof seems not to extend to the model of arithmetic formulas. The bottleneck is the last step, as the determinant of a symbolic matrix $(x_{i,j})_{n\times n}$ may not have poly(n) size formulas. One way to avoid computing the determinant is by making the lifted factor monic in each round of Hensel lifting. The uniqueness of monic Hensel lifting would guarantee that we get the factors directly. But the direct implementation of monic Hensel lifting leads to a quasi-polynomial blowup of formula size because it involves polynomial division in each step. Another way could be to give a formula size upper bound for h'_k in Lemma 4.7. We do not get such a bound directly from the proof. So the closure of formulas under factors remains an open problem.

If one could show that arithmetic formulas are *not* closed under factors, i.e., if some polynomial $f(x_1, \ldots, x_n)$ exists that requires formula of size $\geq n^{\log n}$, but has a nonzero multiple of formula size poly(n), then, by our result, VF would be separated from VBP and by Kaltofen's result, VF would be separated from VP. Note that if $f = g/h$ and g, h have small ABPs/formulas, then f also has a small ABP/formula (Kaltofen & Koiran 2008).

Besides arithmetic formulas, there are other models for which poly(s, d) upper bound on the size of factors are not known, for example, read-once oblivious arithmetic branching programs (ROABP) and constant depth arithmetic circuits.

Acknowledgements

We acknowledge the support from DFG Grant TH 472/5-1. We thank Nitin Saxena, Pranjal Dutta, Arpita Korwar, Sumanta Ghosh, Zeyu Guo, and Mrinal Kumar for helpful discussions. We thank

Vishwas Bhargav for pointing out to us that the subresultant technique of computing GCD is suitable for the ABP-model. We thank Nutan Limaye for asking about the ABP-size for approximate power series roots. A.S. would like to thank the Institute of Theoretical Computer Science at Ulm University for the hospitality.

Funding
Open Access funding enabled and organized by Projekt DEAL

References

Stuart J. Berkowitz (1984). On computing the determinant in small parallel time using a small number of processors. *Information processing letters* **18**(3), 147–150.

Peter Bürgisser (2004). The complexity of factors of multivariate polynomials. *Foundations of Computational Mathematics* **4**(4), 369–396.

Peter Bürgisser (2013). *Completeness and reduction in algebraic complexity theory*, volume 7 of *Algorithms and Computation in Mathematic*. Springer.

Alexander L. Chistov (1985). Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. In *International Conference on Fundamentals of Computation Theory*, 63–69. Springer.

Chi-Ning Chou, Mrinal Kumar & Noam Solomon (2019a). Closure of VP under taking factors: a short and simple proof. Technical Report arXiv:1903.02366, arXiv.

Chi-Ning Chou, Mrinal Kumar & Noam Solomon (2019b). Closure Results for Polynomial Factorization. *Theory of Computing* **15**(13), 1–34.

L. Csanky (1976). Fast Parallel Matrix Inversion Algorithms. *SIAM Journal on Computing* **5**(4), 618–623.

Pranjal Dutta (2018). *Discovering the roots: Unifying and extending results on multivariate polynomial factoring in algebraic complexity*. Master's thesis, Chennai Mathematical Institute.

Pranjal Dutta, Nitin Saxena & Amit Sinhababu (2018). Discovering the roots: Uniform closure results for algebraic classes under factoring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 1152–1165. ACM.

Michael A. Forbes & Amir Shpilka (2015). Complexity Theory Column 88: Challenges in Polynomial Factorization. *ACM SIGACT News* **46**(4), 32–49.

Michael A. Forbes, Amir Shpilka, Iddo Tzameret & Avi Wigderson (2016). Proof complexity lower bounds from algebraic circuit complexity. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, 32. LIPIcs.

Günther Frei (2005). The Unpublished Section Eight: On the Way to Function Fields over a Finite Field. In *The Shaping of Arithmetic after C. F. Gauß' Disquisitiones Arithmeticae*, chapter II.4, 159–198. Springer.

Joachim von zur Gathen (1984). Hensel and Newton methods in valuation rings. *Mathematics of Computation* **42**(166), 637–661.

Joachim von zur Gathen & Jürgen Gerhard (2013). *Modern Computer Algebra*. Cambridge University Press.

Carl Friedrich Gauss (1870). *Werke, Band II, zweiter Abdruck*. Dietrich, Göttingen.

Keith O. Geddes, Stephen R. Czapor & George Labahn (1992). *Algorithms for Computer Algebra*. Springer Science & Business Media.

Kurt Hensel (1899). Über eine neue Begründung der Theorie der algebraischen Zahlen. In *Jahresbericht der Deutschen Mathematiker-Vereinigung*, volume 6, 83–88. Teubner.

Kurt Hensel (1904). Neue Grundlagen der Arithmetik. *Journal für die reine und angewandte Mathematik* **127**, 51–84.

Kurt Hensel (1908). *Theorie der algebraischen Zahlen*. Teubner.

Kurt Hensel (1918). Eine neue Theorie der algebraischen Zahlen. *Mathematische Zeitschrift* **2**, 433–452.

Valentine Kabanets & Russell Impagliazzo (2004). Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity* **13**(1–2), 1–46.

Erich Kaltofen (1986). Uniform Closure Properties of P-Computable Functions. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, 330–337.

Erich Kaltofen (1987). Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials. In *Proceedings of the 19th annual ACM Symposium on Theory of Computing (STOC)*, 443–452. ACM.

Erich Kaltofen (1989). Factorization of polynomials given by straight-line programs. *Randomness and Computation* **5**, 375–412.

Erich Kaltofen (1995). Effective Noether irreducibility forms and applications. *Journal of Computer and System Sciences* **50**(2), 274–295.

Erich Kaltofen & Pascal Koiran (2008). Expressing a fraction of two determinants as a determinant. In *Proceedings of the 21st International Symposium on Symbolic and Algebraic Computation*, 141–146. ACM.

Erich Kaltofen & Barry M. Trager (1990). Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation* **9**(3), 301–320.

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

Mrinal Kumar & Ramprasad Saptharishi (2019). Hardness-Randomness Tradeoffs for Algebraic Computation. *Bulletin of EATCS* **3**(129).

Meena Mahajan (2014). Algebraic complexity classes. In *Perspectives in Computational Complexity*, 51–75. Springer.

Meena Mahajan & V Vinay (1999). Determinant: Old algorithms, new insights. *SIAM Journal on Discrete Mathematics* **12**(4), 474–490.

Rafael Oliveira (2016). Factors of low individual degree polynomials. *computational complexity* **2**(25), 507–561.

Ramprasad Saptharishi (2021). A survey of lower bounds in arithmetic circuit complexity. https://github.com/dasarpmar/lowerbounds-survey/releases.

Tateaki Sasaki & Masayuki Suzuki (1992). Three new algorithms for multivariate polynomial GCD. *Journal of symbolic computation* **13**(4), 395–411.

Amir Shpilka & Ilya Volkovich (2010). On the Relation between Polynomial Identity Testing and Finding Variable Disjoint Factors. In *International Colloquium on Automata, Languages and Programming*, 408–419. Springer.

Madhu Sudan (1998). Algebra and Computation. http://people.csail.mit.edu/madhu/FT98/course.html. Lecture Notes.

Hans Zassenhaus (1969). On Hensel factorization, I. *Journal of Number Theory* **1**(3), 291–311.

Manuscript received 10 March 2021

Amit Sinhababu and Thomas Thierauf
Aalen University, Aalen, Germany
amitkumarsinhababu@gmail.com
thomas.thierauf@uni-ulm.de
https://image.informatik.htw-aalen.de/~thierauf/