Constant-Depth Arithmetic Circuits for Linear Algebra Problems

Robert Andrews* Avi Wigderson†

April 16, 2024

Abstract

We design polynomial size, constant depth (namely, AC^0) arithmetic formulae for the greatest common divisor (GCD) of two polynomials, as well as the related problems of the discriminant, resultant, Bézout coefficients, squarefree decomposition, and the inversion of structured matrices like Sylvester and Bézout matrices. Our GCD algorithm extends to any number of polynomials. Previously, the best known arithmetic formulae for these problems required super-polynomial size, regardless of depth.

These results are based on new algorithmic techniques to compute various symmetric functions in the roots of polynomials, as well as manipulate the multiplicities of these roots, without having access to them. These techniques allow AC^0 computation of a large class of linear and polynomial algebra problems, which include the above as special cases.

We extend these techniques to problems whose inputs are *multivariate* polynomials, which are represented by AC^0 arithmetic circuits. Here too we solve problems such as computing the GCD and squarefree decomposition in AC^0.

Contents

*School of Mathematics, Institute for Advanced Study. Supported by NSF grant CCF-1900460 and by the Erik Ellentuck Endowed Fellowship Fund. Email: randrews@ias.edu.

†School of Mathematics, Institute for Advanced Study. Supported by NSF grant CCF-1900460. Email: avi@ias.edu.

1 Introduction

1.1 Background

Arithmetic complexity theory studies the computation of polynomials (and rational functions) using the basic arithmetic operations over a field. Like Boolean complexity theory, it is a vast, developed, and very active field, with its own computational models, complexity classes, algorithms, lower bounds, reductions, and complete problems. Extensive texts on different parts of the field include von zur Gathen and Gerhard [vzGG13], Bürgisser [Bür00], and Shpilka and Yehudayoff [SY10]. Our main model of computation is arithmetic circuits, focusing on their size (up to polynomial factors), and in more detail on their depth (up to constant factors).

Arithmetic algorithms were devised by mathematicians for centuries, long before computers ushered in applications in symbolic computation and computer algebra, which have invigorated the theoretical study of arithmetic complexity. The Euclidean algorithm for computing the GCD (originally described for integers, but works equally well for polynomials) is perhaps the oldest nontrivial algorithm to appear in print. Another is Gaussian elimination (whose origins also appear

in ancient texts) for computing the determinant of a matrix, which is essential for linear algebra. In a sense, the story below concerns the relative difficulty of these two basic problems, determinant and GCD, from the viewpoint of parallel computation.

The development of parallel computing architectures invited the design of fast parallel algorithms, in both the Boolean and arithmetic settings. The GCD and determinant algorithms above, while polynomial time, are described in a way which looks "inherently sequential," and parallelizing them presented a nontrivial challenge. The breakthrough of Csanky in 1976 [Csa76], providing an NC^2 (polynomial size, $O(\log^2 n)$ depth) algorithm for the determinant, was followed by a decade of intense activity in which many more parallel algorithms were developed. One natural consequence was that practically all linear algebra problems, such as inverting matrices, solving systems of linear equations, and computing the rank of a matrix, were also in NC^2. Many other problems in polynomial algebra were known or were found to have linear algebraic formulations, and so could be reduced to the determinant. Examples include polynomial division with remainder, decoding of (some) linear codes, and the GCD problem for (any number of) univariate polynomials [vzGat84].[1]

A more general understanding of the parallel complexity of arithmetic computation followed another breakthrough, this time of Hyafil in 1979 [Hya79], later sharpened by Valiant, Skyum, Berkowitz, and Rackoff [VSBR83], who gave a generic *depth reduction* of arithmetic circuits. Using as input size both the number of input variables as well as the degree of the computed polynomials, they showed that *any* polynomial-size circuit can be parallelized, namely simulated in similar size and $O(\log^2 n)$ depth! In other words, in the arithmetic setting, *every* sequential algorithm can be efficiently parallelized: $\mathsf{P} = \mathsf{NC}^2$. In particular, fast parallel algorithms for determinant and GCD could be derived directly from their sequential analogues via this simulation. (Note that such a collapse is believed *not* to hold in the Boolean setting, and in particular *Integer* GCD is one canonical example of a problem believed not to have a shallow Boolean circuit.)

While very satisfying, the above general results above get stuck at $O(\log^2 n)$ depth, and though the circuits are of polynomial size, the resulting formulae have quasi-polynomial size $n^{O(\log n)}$. Which of these problems have polynomial-size formulae, namely are in the class NC^1 of logarithmic depth (with bounded fan-in gates)? How about constant parallel time? There are clearly natural linear and polynomial algebra problems that can be performed even in AC^0, i.e., have polynomial size formulae of *constant depth*, allowing gates of unbounded fan-in. Easy examples include polynomial addition, multiplication, univariate polynomial evaluation, and interpolation (obtaining the coefficients of a polynomial from point evaluations).

The same golden decade (from the mid 1970's to mid 1980's, summarized beautifully in von zur Gathen's 1986 survey [vzGat86]) provided further nontrivial algorithms putting certain polynomial and linear algebra problems in AC^0. (In some cases they were stated as NC^1 algorithms, but it is easy to see that they can also be implemented in AC^0.) Bini [Bin84] devised an ingenious AC^0 algorithm to invert *triangular* Toeplitz matrices.[2] This was used by Bini and Pan [BP85] to give an AC^0 algorithm for polynomial division with remainder. (We will give very different AC^0 algorithms for these problems.) Another important example was Ben-Or's AC^0 circuit to compute the elementary symmetric polynomials.[3] As these polynomials are natural arithmetic analogues of Boolean threshold functions, this reveals the surprising power of arithmetic circuits over Boolean circuits in the bounded-depth regime (in all other regimes, arithmetic circuits are considered "weaker"), as the majority function is well-known not to have Boolean AC^0 circuits.

[1] A minor but important point which should be mentioned is that arithmetic circuits formally cannot compute discontinuous functions like GCD, and one has to add to them (as is standard in the field) the ability of branching on testing of a given field element is zero or not.

[2] Toeplitz matrices have constant diagonals.

[3] The reader unfamiliar with this gem is encouraged to find any efficient algorithm for them.

Indeed, while AC^0 lower bounds in the Boolean setting has been known since the 1980's, it took over 30 years to obtain any analogous lower bound in the arithmetic one. This was finally done by Limaye, Srinivasan, and Tavenas in 2021 [LST21]. They proved that any constant-depth circuit for the product of n 2×2 matrices (a problem in NC^1) must have super-polynomial size! As the determinant can efficiently simulate any formula (by Valiant's completeness result for the class VF [Val79]), the same lower bound follows for the determinant.[4] Moreover, following [CKL+23], we know that, in contrast to families like the elementary symmetric and power sum polynomials, which are in AC^0, some natural families of symmetric polynomials such as the Schur polynomials are as hard as the determinant, and thus can't be in AC^0.

What about the GCD? Is it in AC^0? As hard as the determinant? Or somewhere in the middle?

1.2 Our Results

We give AC^0 algorithms (more precisely, constant depth and polynomial size formulae) for a host of problems from linear and polynomial algebra. We note that in all cases these problems were not known to have polynomial size formulae, regardless of depth.

While arithmetic circuits are a non-uniform model of computation, and in particular allow access to arbitrary constants in the underlying field, all our algorithms are uniform, and may alternatively be viewed arithmetic PRAM algorithms with a polynomial number of processors running in constant parallel time. This view may be advantageous, as PRAMs allow more basic operations, like branching on a zero-test; such a test must be added to the model of arithmetic circuits for discontinuous functions like GCD.

All our results hold over every field of characteristic 0, and every field of large enough[5] positive characteristic. Input and output polynomials, which will typically be univariate,[6] are always described by their coefficients, and so their degrees will be implicitly counted in the input (and output) size.

We state our results informally here, and defer formal results to the technical sections, after formally defining our computational model, which is essentially arithmetic circuits over a field. As is standard in the field (and many of the results in the previous section), when computing a discontinuous function like the GCD, a circuit can also branch by testing if a field element equals zero.

Our first main theorem resolves the complexity of the GCD.

Theorem 1.1 (see Theorem 8.1 and Corollary 8.2)**.** *Given two polynomials $f, g \in \mathbb{F}[x]$, their GCD and LCM can be computed in AC^0.*

We discuss some concrete and abstract extensions of this algorithm. The concrete ones involve some polynomials and matrices related to the GCD, which we now define.

Assume that the polynomials f and g are monic and of degrees n and m, respectively, and factor completely over the algebraic closure $\overline{\mathbb{F}}$ of $\mathbb{F}$ as $f = \prod_i (x - \alpha_i)$ and $g = \prod_i (x - \beta_i)$, with possible repetitions in case of multiplicities. The *resultant* $\operatorname{res}(f, g) = \prod_{i,j} (\alpha_i - \beta_j) \in \mathbb{F}$ is nonzero if and only if $\gcd(f, g) = 1$. A well known special case of the resultant is the *discriminant* of a polynomial, defined as $\operatorname{disc}(f) \coloneqq \operatorname{res}(f, f')$ where f' is the first derivative of f, which is zero precisely when f has a double root. For quadratic polynomials, the discriminant takes the familiar form $\operatorname{disc}(ax^2 + bx + c) = b^2 - 4ac$.

When $\gcd(f, g) = 1$, it is well-known that there are (unique) polynomials a of degree $< m$ and b of degree $< n$ such that $af + bg = 1$. (A similar formula holds when the GCD has positive degree.)

[4] Of course, this can be seen by a simple direct reduction as well.

[5] Only polynomial in the input size.

[6] In some cases they will have a constant number of variables.

The polynomials a and b are known as the *Bézout coefficients* of f and g. The equation $af + bg = 1$ is actually a linear system whose variables are the coefficients of the unknown polynomials a and b, given by an $(n+m) \times (n+m)$ matrix called the *Sylvester matrix* $\mathrm{Syl}(f, g)$ (which can be seen in Definition 2.14) whose entries are the given coefficients of f and g. In fact, the resultant $\mathrm{res}(f, g)$ is precisely the determinant of the Sylvester matrix $\mathrm{Syl}(f, g)$. A related matrix, important for rational interpolation and Padé approximation of polynomials is the Bézout matrix of f and g, denoted $\mathrm{Bez}(f, g)$.

The natural problems associated with these objects have fast parallel algorithms as well.

Theorem 1.2 (see Theorems 6.1, 6.7 and 6.10 and Corollary 8.3)**.** *Given two polynomials $f, g \in \mathbb{F}[x]$, their resultant and Bézout coefficients, as well as the inverses of their Sylvester and Bézout matrices, can be computed in* AC^0.

We note that the previously-mentioned problems of inverting triangular Toeplitz matrices and polynomial division with remainder are in fact corollaries of the theorem above through simple reductions. Although these problems were already known to have AC^0 algorithms by Bini [Bin84] and Bini and Pan [BP85], our algorithms are of a very different flavor!

For the more abstract extensions of our GCD algorithm, let us rewrite the factorization of the polynomials f and g slightly differently. Let $\gamma_1, \ldots, \gamma_k$ be the union of their roots over the algebraic closure $\overline{\mathbb{F}}$ (namely, the union of the α_i and β_i above). We can write $f = \prod_{i=1}^{k} (x - \gamma_i)^{a_i}$ and $g = \prod_{i=1}^{k} (x - \gamma_i)^{b_i}$, where the a_i and b_i are the multiplicities of the root γ_i in f and g, respectively. Clearly, their GCD is given by

$$\gcd(f, g) = \prod_{i=1}^{k} (x - \gamma_i)^{\min(a_i, b_i)}.$$

Similarly, their LCM is given by

$$\mathrm{lcm}(f, g) = \prod_{i=1}^{k} (x - \gamma_i)^{\max(a_i, b_i)}.$$

Consider now the product of f and g, which also has a trivial AC^0 algorithm. In this notation we have

$$f \cdot g = \prod_{i=1}^{k} (x - \gamma_i)^{a_i + b_i}.$$

One is naturally led to consider what other functions of the exponents are computable in AC^0. How about product of the exponents, namely

$$f \diamond g \coloneqq \prod_{i=1}^{k} (x - \gamma_i)^{a_i b_i}?$$

It is not even clear at first sight that this is an algebraic operation computable by arithmetic circuits at all. However, with the same techniques we use to compute the GCD, we can actually prove that it is. Indeed, *any* function is.

Theorem 1.3 (see Theorem 9.2)**.** *Let $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be any integer function. Given two polynomials $f, g \in \mathbb{F}[x]$ as above, the polynomial*

$$f \diamond_P g \coloneqq \prod_{i=1}^{k} (x - \gamma_i)^{P(a_i, b_i)}$$

can be computed in AC^0.

Note that the size of the circuit computing $f \diamond_P g$ is polynomial in the degrees of the input *and* output of the problem, the latter being $\max\{P(i,j) : i,j \in [k]\}$, which is of course necessary.

The same theorem extends verbatim for any *constant* number c of polynomials, and for any c-ary integer function P. As an example, for $c = 5$, we can take P to return the median of its inputs and compute the corresponding polynomial in AC^0. The intuitive meaning of some operations we can perform so efficiently is far from obvious, and it would be interesting to find a real application for some such function P.

Another natural extension is to consider an arbitrary number of input polynomials. This is not obvious, even for the GCD function. Indeed, for three polynomials f, g, and h, there is no analogue of the resultant, i.e., a single polynomial function of the coefficients of f, g, and h which is nonzero if and only if $\gcd(f,g,h) = 1$. Nevertheless, we can compute the GCD of an arbitrary number of polynomials, again in AC^0.

Theorem 1.4 (see Theorem 8.4 and Corollary 8.6)**.** *Given any number of polynomials* $f_1, f_2, \ldots, f_m \in \mathbb{F}[x]$, $\gcd(f_1, f_2, \ldots, f_m)$ *and* $\operatorname{lcm}(f_1, f_2, \ldots, f_m)$ *can be computed in* AC^0.

Again, one can note that the GCD and LCM functions perform min and max operations, respectively, on the exponents of each linear factor of the f_j. Moreover, the product of m polynomials performs addition on the exponents, and is also in AC^0. So, we are lead to ask: what other functions of the exponents give rise to such fast parallel algorithms? Here we also have a fairly general result. Keeping with the notation above, let $\gamma_1, \ldots, \gamma_k \in \overline{\mathbb{F}}$ be the union of the roots of m polynomials $f_1, \ldots, f_m \in \mathbb{F}[x]$ and write

$$f_i(x) = \prod_{j=1}^{k} (x - \gamma_j)^{a_{i,j}}.$$

As in the case of two polynomials, we can apply *any* function to the exponents of each factor. This holds when the function $P : \mathbb{N}^m \to \mathbb{N}$ is given in the dense representation (so P is specified by a list of roughly d^m numbers, where d bounds the degree of the f_i), and also when P is described more succinctly as a sort of circuit over the integers (see Definition 9.5 for the precise definition).

Theorem 1.5 (see Theorems 9.4 and 9.6)**.** *Let* $P : \mathbb{N}^m \to \mathbb{N}$ *be any integer function. Given* m *polynomials* $f_1, \ldots, f_m \in \mathbb{F}[x]$ *as above, the polynomial*

$$\diamond_P(f_1, \ldots, f_m) \coloneqq \prod_{i=1}^{k} (x - \gamma_i)^{P(a_{1,i}, \ldots, a_{m,i})}$$

can be computed in AC^0.

Our results also extend to the multivariate setting. Here, the input polynomials can be themselves be given as arithmetic circuits. Using standard tools to reduce questions about multivariate factorization (like the GCD) to their univariate counterparts, our earlier algorithm for the univariate GCD can be used to compute the GCD of multivariate polynomials.

Theorem 1.6 (see Theorem 10.8)**.** *Given multivariate polynomials* $f_1, \ldots, f_m \in \mathbb{F}[\overline{x}]$*, if* $f_1, \ldots, f_m$ *can be computed in* AC^0*, then* $\gcd(f_1, \ldots, f_m)$ *and* $\operatorname{lcm}(f_1, \ldots, f_m)$ *can be computed in* AC^0.

1.3 Our Techniques and Their Origins

To summarize this section in a sentence, everything is about efficiently computing interesting and useful *symmetric* functions over the roots of given polynomials using only their coefficients as input. We'll now give a taste of the main ones.

Fix an integer n. Assume again that a degree n polynomial $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{F}[x]$ factors completely over $\overline{\mathbb{F}}$ as

$$f = \prod_{i=1}^{n} (x - \alpha_i),$$

where some of the α_i may repeat due to multiplicity. We denote by $\overline{\alpha}$ the vector of the α_i in some arbitrary order (it will not matter which). We stress that our algorithms have no access to these roots (indeed, the roots may not even be in $\mathbb{F}$), only to the coefficients $a_j \in \mathbb{F}$ of f. However, it is well known that these coefficients are (up to sign) the *elementary symmetric polynomials* of the roots. More precisely,

$$a_j = (-1)^{n-j} e_{n-j}(\overline{\alpha}),$$

where $e_d(\overline{z}) \coloneqq \sum_{S \subset [n], |S|=d} \prod_{j \in S} z_j$.

The famous Girard–Newton identities relate the elementary symmetric polynomials to another important family of symmetric polynomials, the *power sum polynomials*, defined by $p_k(\overline{z}) \coloneqq \sum_{j=1}^{n} z_j^k$. More precisely, for any integer m, the first m power sums are polynomial expressions in the first m elementary symmetric polynomials, and vice versa.[7] Thus, the coefficients of a polynomial give us access to new symmetric functions of its roots: the power sums.

There are several different ways to express these relations, each with their own uses. For example, noticing that the relations above are *triangular* was a key fact used in Csanky's original NC^2 algorithm for the determinant [Csa76]. For us, an exponential form encompassing the relations between the generating functions $\sum_m e_m t^m$ and $\sum_m p_m t^m$ (see Section 3) will be key. Simple use of interpolation gives rise to our first tool (observed by many), namely that this conversion between the two families can be performed in AC^0.

Theorem 1.7 (Folklore)**.** *For every m, there are AC^0 circuits that given the first m elementary symmetric polynomials $e_j(\overline{z})$ as inputs, output the first m power sums $p_j(\overline{z})$, and vice versa.*

For example, this fact (in *one* direction) has been used in [SW01] to obtain smaller depth-4 and depth-6 formulas for the elementary symmetric polynomials than Ben-Or's depth-3 formulas mentioned above.

But for us, a central inspiration came from the paper of Bostan, Flajolet, Salvy, and Schost [BFSS06]. While they only discuss sequential algorithms, they use this conversion (in *both* directions) to manipulate roots of given polynomials in very interesting ways that look similar to polynomials we want to compute, such as the resultant. In particular, given polynomials f and g having roots α's and β's as above, they compute the polynomials $f \oplus g \coloneqq \prod_{i,j}(x - (\alpha_i + \beta_j))$ and $f \otimes g \coloneqq \prod_{i,j}(x - \alpha_i \beta_j)$. Indeed, they can replace sum and product with any fixed bivariate polynomial!

Observing that actually $(f(x) \oplus g(-x))|_{x=0} = \operatorname{res}(f, g)$, and inspecting their sequential algorithm to verify that it can be parallelized using the folklore theorem above, we already get an AC^0 circuit for the resultant! This indeed was our starting point.

To get our results, we will need to obtain more symmetric functions of the roots of a given polynomial f. It goes without saying that the *fundamental theorem of symmetric polynomials* states that *any* n-variate symmetric polynomial can be written as a fixed polynomial in the n elementary

[7] Note that while $e_m(\overline{z}) = 0$ for $m > n$, this is not the case for $p_m(z)$. Still, the statement above holds for every m.

symmetric ones (and hence, by the Girard–Newton identities, also in terms of the first n power sums). The whole question for us is which of these conversions can be performed in AC^0.

A first step is the simple observation that for any given polynomial g, we can compute in AC^0 the sum $\sum_{i=1}^{n} g(\alpha_i)$ over the roots of f, by using the power sums $p_k(\overline{\alpha})$. It would actually be useful to do the same for the product $\prod_{i=1}^{n} g(\alpha_i)$, as this is yet another expression for the resultant $\operatorname{res}(f, g)$. We prove a more general result, computing in AC^0 every elementary symmetric polynomial over the values $r(\alpha_i)$ where r is any given rational function.

Theorem 1.8 (see Lemma 5.2)**.** *Given $f, g \in \mathbb{F}[x]$ and any integer d, we can compute $e_d(g(\alpha_1), \ldots, g(\alpha_n))$ in AC^0. Moreover given another polynomial h which has no roots in common with f, we can compute*

$$e_d\left(\frac{g(\alpha_1)}{h(\alpha_1)}, \ldots, \frac{g(\alpha_n)}{h(\alpha_n)}\right)$$

in AC^0.

Towards getting a handle over the multiplicities of roots of f, we first note that all derivatives $f^{(r)}(x)$ of f can be easily computed in AC^0. A root α has multiplicity at least r if and only if f and its first $r-1$ derivatives vanish at this root. This is used to construct a polynomial g (with a constant number of variables besides x), to which the theorem above can be applied (see details in Section 7). This allows us to *filter* out the roots of f with multiplicities above (or below) a given threshold r, and hence obtain those of multiplicity precisely r. An important consequence is the ability to compute the *squarefree decomposition* of f. We conclude the techniques section with this consequence.

Theorem 1.9 (see Lemma 7.6)**.** *Given $f \in \mathbb{F}[x]$, we can compute in AC^0 the (unique) sequence of polynomials $f_1, f_2, \ldots, f_n \in \mathbb{F}[x]$ such that no f_r has a double root, $\gcd(f_i, f_j) = 1$, and $f = \prod_{r=1}^{n} f_r^r$.*

This ability to filter out the roots by multiplicity is key to most of our results, and the reader is invited to see how compute $\gcd(f, g)$ using it.

1.4 Organization

The rest of this paper is organized as follows. We start with preliminary material in Section 2. In Section 3, we review folklore AC^0 implementations of Newton's identities, which will be an essential tool for all of our results. Section 4 is a warm-up, where we show how Newton's identities can be used to solve some interesting toy problems in AC^0. Our work starts in earnest in Section 5, where we develop tools to evaluate symmetric functions of the roots of a given polynomial. We then apply these tools in Section 6, where we compute the determinants and inverses of the Sylvester and Bézout matrices in AC^0.

Section 7 is the main technical section in this work, where we introduce filtering and thresholding, two techniques that let us design AC^0 algorithms that manipulate the factorization pattern of a polynomial without having explicit access to its roots. Section 8 applies the results of Section 7 to compute the GCD and LCM of many polynomials in AC^0. We generalize this result in Section 9 to arbitrary functions of root multiplicities. In Section 10, we extend our univariate algorithms to the multivariate setting. Finally, Section 11 concludes with some open problems.

2 Preliminaries

To keep this work self-contained, this section includes a number of well-known results from arithmetic complexity and computer algebra.

2.1 Notation

We work over a field $\mathbb{F}$ of characteristic zero or of polynomially-large characteristic. For example, to compute the GCD of two polynomials of degree d over a field of positive characteristic, we require $\mathrm{char}(\mathbb{F}) \geqslant 2d+1$. The precise requirements on the characteristic of $\mathbb{F}$ will be specified in the statements of our results.

When we analyze our algorithms, it will often be convenient to work with the factorization of a polynomial $f(x)$ into linear factors $f(x) = \prod_i (x - \alpha_i)$. In general, such a factorization only exists over the algebraic closure $\overline{\mathbb{F}}$ of the base field. This factorization is only used in the analysis of our algorithms; in particular, we do not assume that the field $\mathbb{F}$ is algebraically closed.

We abbreviate a vector $(x_1, \ldots, x_n)$ as $\overline{x}$. We denote by $\mathbb{F}[\overline{x}]$ the polynomial ring in the variables $x_1, \ldots, x_n$. For a vector $\overline{a} \in \mathbb{N}^n$, we abbreviate the monomial $\prod_{i=1}^n x_i^{a_i}$ as $\overline{x}^{\overline{a}}$. We let $\|\overline{a}\|_1 \coloneqq \sum_{i=1}^n a_i$ denote the ℓ_1 norm of $\overline{a}$. We use $\mathbb{F}(\overline{x})$ and $\mathbb{F}[\![\overline{x}]\!]$ to denote the field of rational functions and ring of formal power series, respectively, in the variables $x_1, \ldots, x_n$. Given a polynomial $f \in \mathbb{F}[x]$ and a natural number $r \in \mathbb{N}$, we write $f^{(r)}(x)$ for the r^{th} derivative of f. For two polynomials $f, g \in \mathbb{F}[x]$, we write $f \mid g$ to denote that f divides g.

Throughout, if the input to an algorithmic problem consists of univariate polynomials $f_1, \ldots, f_m \in \mathbb{F}[x]$, we assume that the polynomials $f_1, \ldots, f_m$ are monic, i.e., that the leading coefficient of each f_i is 1. This is done for the sake of notational convenience; all of our algorithms easily extend to handle non-monic inputs. As our inputs are assumed to be monic, we likewise adopt the convention that the GCD and LCM are defined to be monic polynomials. From here on, all univariate polynomials are assumed to be monic unless specified otherwise.

For a natural number $d \in \mathbb{N}$, we write

$$p_d(\overline{x}) \coloneqq \sum_{i=1}^n x_i^d$$
$$e_d(\overline{x}) \coloneqq \sum_{\substack{S \subseteq [n] \\ |S| = d}} \prod_{i \in S} x_i$$

for the degree-d power sum and elementary symmetric polynomials, respectively. These two families of polynomials play an essential role throughout our work. We adopt the conventions that $p_0(\overline{x}) = n$, $e_0(\overline{x}) = 1$, and $e_d(\overline{x}) = 0$ for $d > n$.

For a matrix $A \in \mathbb{F}^{n \times n}$, we denote by $\mathrm{adj}(A) \in \mathbb{F}^{n \times n}$ the *adjugate* of A, defined as

$$\mathrm{adj}(A)_{i,j} \coloneqq (-1)^{i+j} \det(A_{-j,-i}),$$

where $A_{-j,-i}$ is the submatrix of A obtained by deleting the j^{th} row and i^{th} column. The adjugate satisfies the identity $\mathrm{adj}(A)A = \det(A) I_n$, where I_n is the $n \times n$ identity matrix. In particular, when A is invertible, the inverse of A is given by $\frac{1}{\det(A)} \mathrm{adj}(A)$.

2.2 Arithmetic Circuits

2.2.1 Circuits and Complexity Classes

We use arithmetic circuits as our basic model of computation.

Definition 2.1. Let $\mathbb{F}$ be a field and let $\mathbb{F}(\overline{x})$ be the field of rational functions in the variables $x_1, \ldots, x_n$. An *arithmetic circuit over* $\mathbb{F}$ is a directed acyclic graph. Vertices of in-degree zero are called *input gates* and are each labeled by a variable x_i or a field element $\alpha \in \mathbb{F}$. Vertices of positive

in-degree are called *internal gates* and are labeled by an element of $\{+, \times, \div\}$. Vertices of out-degree zero are called *output gates*. Each gate of the circuit computes a rational function in $\mathbb{F}(\overline{x})$ in the natural way. If $\{f_1, \ldots, f_m\}$ are the functions computed by the output gates of the circuit, we say that the circuit computes $\{f_1, \ldots, f_m\}$. The *size* of the circuit is the number of wires in the circuit. The *depth* of the circuit is the length of the longest path from an input gate to an output gate. $\Diamond$

Naturally, one can define complexity classes of (families of) rational functions in terms of their arithmetic circuit complexity. The following classes capture efficient low-depth computation. Needless to say, these are separate classes for every field $\mathbb{F}$, which we suppress.

Definition 2.2. Let $f = (f_1, f_2, \ldots)$ be a family of rational functions. We say that $f \in \mathsf{NC}^i$ if f_n can be computed by an arithmetic circuit of fan-in two, size $n^{O(1)}$, and depth $O(\log^i n)$. We say that $f \in \mathsf{AC}^i$ if f_n can be computed by an arithmetic circuit of *unbounded* fan-in, size $n^{O(1)}$, and depth $O(\log^i n)$. $\Diamond$

In arithmetic circuit complexity, it is standard to restrict attention to families of polynomials $(f_1, f_2, \ldots)$ with the additional restriction that $\deg(f_n) \leqslant n^{O(1)}$. Under this restriction, a well-known result of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] shows that any circuit of size $n^{O(1)}$ can be converted to one of size $n^{O(1)}$ and depth $O(\log^2 n)$. A careful reading of their proof shows that depth $O(\log n)$ suffices if unbounded fan-in is allowed. Thus, when we restrict attention to polynomials of degree $n^{O(1)}$, we have the collapse

$$\mathsf{AC}^0 \subsetneq \mathsf{NC}^1 \subseteq \mathsf{AC}^1 = \mathsf{NC}^2 = \mathsf{AC}^2 = \mathsf{NC}^3 = \cdots.$$

Further inspection of the proof of [VSBR83] shows that depth $O(\log n)$ can be attained using addition gates of unbounded fan-in and multiplication gates of fan-in two, corresponding to the class SAC^1 (for *semi-unbounded* AC^1). The strict inclusion $\mathsf{AC}^0 \subsetneq \mathsf{NC}^1$ is a straightforward corollary of the depth hierarchy theorem of Limaye, Srinivasan, and Tavenas [LST21].

We also mention the Boolean complexity class DET [Coo85], which consists of all problems that are logspace-reducible to the determinant of an integer matrix. This includes many familiar linear-algebraic problems, including matrix powering, matrix inverse, and computing the characteristic polynomial of a matrix. In the Boolean setting, it is known that $\mathsf{NL} \subseteq \mathsf{DET} \subseteq \mathsf{NC}^2$. It is conjectured that $\mathsf{DET} \not\subseteq \mathsf{NC}^1$. If $\mathsf{DET} \subseteq \mathsf{NC}^1$ were true, then we would have the chain of inclusions

$$\mathsf{NL} \subseteq \mathsf{DET} \subseteq \mathsf{NC}^1 \subseteq \mathsf{L},$$

implying the unlikely collapse $\mathsf{L} = \mathsf{NC}^1 = \mathsf{NL}$.

The algebraic analogue of DET, often denoted VBP, is the class of polynomial families $(f_1, f_2, \ldots)$ computable by arithmetic branching programs of size $n^{O(1)}$. We will not give a precise definition of arithmetic branching programs here, as we will not make use of them. As in the Boolean setting, it is conjectured that the determinant and matrix inverse are *not* computable by arithmetic circuits of fan-in two, size $n^{O(1)}$, and depth $O(\log n)$. The recent work of Limaye, Srinivasan, and Tavenas [LST21] shows unconditionally that the determinant and related problems are not computable by arithmetic circuits of size $n^{O(1)}$ and depth $O(1)$, i.e., that the determinant is not computable in AC^0.

2.2.2 Piecewise Arithmetic Circuits

By definition, arithmetic circuits can only represent rational functions. However, natural functions of interest, like the GCD, are not rational. For example, it is easy to see that

$$\gcd(x - \alpha, x - \beta) = \begin{cases} x - \alpha & \text{if } \alpha = \beta, \\ 1 & \text{otherwise.} \end{cases}$$

This is not a continuous function of α and β, so we cannot hope to compute the GCD using an arithmetic circuit.

To compute the GCD, we have to add a branching instruction to our model of computation. In arithmetic complexity, this is typically done by allowing an algorithm to test if a computed quantity equals zero and branch accordingly. This sort of operation is standard: even the Euclidean algorithm uses zero-testing to detect the degree of a remainder. We formalize this via arithmetic circuits that define a function piecewise. This is a natural and simple extension of arithmetic circuits that allows them to compute functions like the GCD. Although we provide a precise definition here for completeness, we encourage the reader to keep in mind standard arithmetic circuits as the model of computation.

We now define piecewise arithmetic circuits. Formally, these are two collections $\mathcal{A}$ and $\mathcal{B}$ of (standard) arithmetic circuits. The first collection $\mathcal{A}$ is used to compute a function $f : \mathbb{F}^n \to \mathbb{F}^m$ piecewise, and the second collection $\mathcal{B}$ is used to decide which circuit from $\mathcal{A}$ should be applied to a given input $\overline{\alpha} \in \mathbb{F}^n$.

Definition 2.3. Let $f : \mathbb{F}^n \to \mathbb{F}^m$ be a piecewise rational function. We say that f is *computed piecewise* by an arithmetic circuit if there is a sequence of tuples of circuits $(C_1, T_1), \ldots, (C_m, T_m)$ such that

$$f(\overline{\alpha}) = \begin{cases} C_1(\overline{\alpha}) & \text{if } T_1(\overline{\alpha}) \neq 0 \\ C_2(\overline{\alpha}) & \text{if } T_1(\overline{\alpha}) = 0 \text{ and } T_2(\overline{\alpha}) \neq 0 \\ \vdots & \\ C_m(\overline{\alpha}) & \text{if } \bigwedge_{i=1}^{m-1} T_i(\overline{\alpha}) = 0 \text{ and } T_m(\overline{\alpha}) \neq 0. \end{cases}$$

We refer to the circuits $C_1, \ldots, C_m$ as the *computation circuits* and to $T_1, \ldots, T_m$ as the *test circuits*. The *size* of the circuit computing f is given by $\sum_{i=1}^{m} |C_i| + |T_i|$, where $|C_i|$ and $|T_i|$ are the sizes of C_i and T_i, respectively. The *depth* of the circuit computing f is given by $\max_{i \in [m]}(\text{depth}(C_i), \text{depth}(T_i))$, where $\text{depth}(C_i)$ and $\text{depth}(T_i)$ are the depths of C_i and T_i, respectively. $\lozenge$

Of course, one can generalize Definition 2.3 to allow for more complex logic in how the values of the test circuits determine which computation circuit to use. That level of generality will not be necessary for our results.

Continuing the example of the GCD of two linear polynomials, we can rewrite $\gcd(x - \alpha, x - \beta)$ as

$$\gcd(x - \alpha, x - \beta) = \begin{cases} 1 & \text{if } \alpha - \beta \neq 0 \\ x - \alpha & \text{if } \alpha - \beta = 0, \end{cases}$$

which matches the form of Definition 2.3. Functions computed piecewise by low-depth arithmetic circuits can be evaluated quickly in parallel, first by evaluating the test circuits $T_1, \ldots, T_m$ in parallel and then evaluating the appropriate computation circuit C_i.

We now define complexity classes that correspond to functions computed piecewise by small, low-depth arithmetic circuits.

Definition 2.4. Let $f = (f_1, f_2, \ldots)$ be a family of piecewise rational functions. We say that $f \in \mathsf{NC}^i$ if f_n can be computed piecewise by arithmetic circuits of fan-in two, size $n^{O(1)}$, and depth $O(\log^i n)$. We say that $f \in \mathsf{AC}^i$ if f_n can be computed piecewise by arithmetic circuits of *unbounded* fan-in, size $n^{O(1)}$, and depth $O(\log^i n)$. $\lozenge$

The following elementary lemma that says two piecewise arithmetic circuits can be composed in an efficient manner.

Lemma 2.5. *Let $f : \mathbb{F}^m \to \mathbb{F}^k$ and $g : \mathbb{F}^n \to \mathbb{F}^m$ be piecewise rational functions. Suppose that f can be computed piecewise by an arithmetic circuit of size s_1 and depth Δ_1, and that g can be computed piecewise by an arithmetic circuit of size s_2 and depth Δ_2. Then the composition $f \circ g : \mathbb{F}^n \to \mathbb{F}^k$ can be computed piecewise by an arithmetic circuit of size $O(s_1^2 s_2 + s_1 s_2^2)$ and depth $\Delta_1 + \Delta_2 + 1$.*

Proof. In words, to compute the composition $f \circ g$ on input $\overline{\alpha} \in \mathbb{F}^n$, we first test $\overline{\alpha}$ to determine which piece of g to apply to $\overline{\alpha}$. We then test $g(\overline{\alpha})$ to determine which piece of f should be applied.

To formalize this algorithm using arithmetic circuits, let $(C_{f,1}, T_{f,1}), \ldots, (C_{f,r}, T_{f,r})$ be the family of circuits that compute f and let $(C_{g,1}, T_{g,1}), \ldots, (C_{g,t}, T_{g,t})$ be the family of circuits that compute g. For $(i,j) \in [r] \times [t]$, define

$$C_{f \circ g,(i,j)}(\overline{x}) \coloneqq (C_{f,i} \circ C_{g,j})(\overline{x})$$
$$T_{f \circ g,(i,j)}(\overline{x}) \coloneqq (T_{f,i} \circ C_{g,j})(\overline{x}) \times T_{g,j}(\overline{x}).$$

With $[r] \times [t]$ ordered as $(1,1) \prec (2,1) \prec \cdots \prec (r,1) \prec (1,2) \prec \cdots$, it is clear that the sequence of circuit tuples $(C_{f \circ g,(i,j)}, T_{f \circ g,(i,j)})_{i,j}$ computes the composition $f \circ g$.

Note that the size and depth of $C_{f \circ g,(i,j)}$ are bounded by $s_1 + s_2$ and $\Delta_1 + \Delta_2$, respectively. Likewise, the size and depth of $T_{f \circ g,(i,j)}$ are bounded by $s_1 + 2s_2$ and $\Delta_1 + \Delta_2 + 1$, respectively. Using the naïve bounds $r \leqslant s_1$ and $t \leqslant s_2$, we can bound the total size of the circuits computing $f \circ g$ by $(2s_1 + 3s_2)rt \leqslant O(s_1^2 s_2 + s_1 s_2^2)$. It is clear that the maximum depth of a circuit computing $f \circ g$ is bounded by $\Delta_1 + \Delta_2 + 1$. □

2.3 Known AC^0 Algorithms

In this subsection, we collect previously-known algorithmic results that can be implemented in AC^0. We start with basic operations on univariate polynomials: addition, multiplication, and derivatives. Note that the inputs to these problems are the coefficients of two univariate polynomials $f, g \in \mathbb{F}[x]$ and the outputs are polynomial functions of the coefficients of f and g.

Lemma 2.6. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Then the coefficients of $f(x) + g(x)$, $f(x) \cdot g(x)$, and $f^{(r)}(x)$ can all be computed in AC^0.*

Our next tool is polynomial interpolation, which we use extensively. Let $f \in \mathbb{F}[\overline{x}, y]$ be a polynomial of degree d. We can write f as a polynomial in y whose coefficients are polynomials in $\overline{x}$, i.e., there are polynomials $f_0, \ldots, f_d \in \mathbb{F}[\overline{x}]$ such that

$$f(\overline{x}, y) = \sum_{i=0}^{d} f_i(\overline{x}) y^i.$$

Given evaluations $f(\overline{x}, \alpha_1), \ldots, f(\overline{x}, \alpha_{d+1})$ at $d+1$ distinct values for y, each of the f_i can be expressed as a linear combination of these evaluations. In particular, if f can be computed by a circuit of size s and depth Δ, then the coefficients $f_0, \ldots, f_d$ can each be computed by a circuit of size $O(sd)$ and depth $\Delta + 1$. We record this observation in the following lemma.

Lemma 2.7. *Let $\mathbb{F}$ be a field. Let $f \in \mathbb{F}[\overline{x}, y]$ be a polynomial of degree d. Let $f_0, \ldots, f_d \in \mathbb{F}[\overline{x}]$ be polynomials such that*

$$f(\overline{x}, y) = \sum_{i=0}^{d} f_i(\overline{x}) y^i.$$

Suppose that f can be computed by an arithmetic circuit of size s and depth Δ. Then for each $i \in \{0, 1, \ldots, d\}$, there is a circuit of size $O(sd)$ and depth $\Delta + 1$ that computes f_i. If $|\mathbb{F}| \leqslant d$, then the circuit computing f_i is defined over an extension $\mathbb{K} \supseteq \mathbb{F}$ such that $|\mathbb{K}| \geqslant d + 1$.

A surprising application of interpolation, discovered by Ben-Or, shows that the elementary symmetric polynomials $e_d(x_1, \ldots, x_n)$ can be computed by arithmetic circuits of size $O(n^2)$ and depth 3. This is done by applying interpolation to the polynomial

$$\prod_{i=1}^{n}(1 + yx_i) = \sum_{i=0}^{n} e_d(\overline{x})y^i,$$

which clearly can be computed by a circuit of size $O(n)$ depth 2.

Theorem 2.8 ([Ben-Or]). *The elementary symmetric polynomials $e_d(x_1, \ldots, x_n)$ can be computed in AC^0.*

The next algorithms we quote perform linear algebra with structured matrices. They take as input Toeplitz matrices (or equivalently, Hankel matrices), which we now define.

Definition 2.9. Let $A \in \mathbb{F}^{n\times n}$ be an $n \times n$ matrix. We say that A is a *Toeplitz* matrix if there are field elements $\alpha_{-n+1}, \ldots, \alpha_{n-1} \in \mathbb{F}$ such that $A_{i,j} = \alpha_{i-j}$. We say that A is a *Hankel* matrix if there are field elements $\alpha_1, \ldots, \alpha_{2n-1} \in \mathbb{F}$ such that $A_{i,j} = \alpha_{i+j-1}$. ◊

As an example, the matrices

$$T = \begin{pmatrix} x_3 & x_4 & x_5 \\ x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 \end{pmatrix} \qquad H = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \end{pmatrix}$$

are 3×3 Toeplitz and Hankel matrices, respectively. It is easy to see that by reversing the order of the rows, a Toeplitz matrix becomes Hankel and vice-versa. A beautiful algorithm of Bini [Bin84] shows that triangular Toeplitz matrices can be inverted in AC^0. (The result in [Bin84] is stated as an NC^1 algorithm, but it is clear that AC^0 suffices.)

Theorem 2.10 ([Bin84]). *Let $X \in \mathbb{F}^{n\times n}$ be a triangular Toeplitz matrix. Then the inverse X^{-1} can be computed in AC^0.*

As an application of this algorithm, Bini and Pan [BP85] showed that polynomial division with remainder can be performed in AC^0. (Once again, the result in Bini and Pan [BP85] is stated as an NC^1 algorithm, but AC^0 suffices.)

Theorem 2.11 ([BP85]). *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Let $q, r \in \mathbb{F}[x]$ be the unique polynomials such that $f = qg + r$ and $\deg(r) < \deg(g)$. Then the coefficients of q and r can be computed in AC^0.*

Later in Section 6, we will also describe AC^0 algorithms for inverting triangular Toeplitz matrices and for polynomial division with remainder. Our algorithms employ a different approach than that used by Bini [Bin84] and Bini and Pan [BP85].

The last algorithm we quote is Strassen's theorem on division elimination in arithmetic circuits.

Theorem 2.12 ([Str73]). *Let $f(\overline{x}) \in \mathbb{F}[\overline{x}]$ be a multivariate polynomial of degree $n^{O(1)}$. Suppose $f(\overline{x})$ can be computed in AC^0. Then there is a division-free AC^0 circuit that computes $f(\overline{x})$.*

2.4 The Euclidean Algorithm and the Resultant

This subsection recalls basic definitions and results related to greatest common divisors and the Euclidean algorithm. A more thorough treatment of this material can be found in the delightful book of von zur Gathen and Gerhard [vzGG13]. We start with the definition of the Bézout coefficients of a pair of polynomials.

Definition 2.13. Let $f(x) = \sum_{i=0}^{n} f_i x^i$ and $g(x) = \sum_{i=0}^{m} g_i x^i$ be univariate polynomials of degrees n and m, respectively. Let $d \coloneqq \deg(\gcd(f, g))$. The *Bézout coefficients* of f and g are the unique polynomials $a, b \in \mathbb{F}[x]$ that satisfy

1. $\deg(a) < \deg(g) - d$,
2. $\deg(b) < \deg(f) - d$, and
3. $a(x)f(x) + b(x)g(x) = \gcd(f, g)$. ◊

Note that Bézout coefficients are unchanged by multiplying or dividing f and g by a common factor. Next, we recall the definition of the Sylvester matrix of a pair of polynomials.

Definition 2.14. Let $f(x) = \sum_{i=0}^{n} f_i x^i$ and $g(x) = \sum_{i=0}^{m} g_i x^i$ be univariate polynomials of degrees n and m, respectively. The *Sylvester matrix* of f and g is the $(n + m) \times (n + m)$ matrix given by

$$\operatorname{Syl}(f, g) \coloneqq \begin{pmatrix} f_n & & & & g_m & & & & \\ f_{n-1} & f_n & & & g_{m-1} & g_m & & & \\ \vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & \\ \vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & \\ \vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \ddots \\ \vdots & \vdots & & \vdots & & g_0 & & & g_m \\ f_0 & \vdots & & \vdots & & & \ddots & & \vdots \\ & f_0 & & \vdots & & & & \ddots & \vdots \\ & & \ddots & \vdots & & & & & \ddots \\ & & & f_0 & & & & & g_0 \end{pmatrix}.$$

The *resultant* of f and g is $\operatorname{res}(f, g) \coloneqq \det \operatorname{Syl}(f, g)$. ◊

Let $f, g \in \mathbb{F}[x]$ be polynomials of degrees n and m, respectively. For a natural number $d \in \mathbb{N}$, denote by $\mathbb{F}[x]_{<d}$ the space of univariate polynomials of degree less than d. The Sylvester matrix $\operatorname{Syl}(f, g)$ corresponds to the linear map

$$\begin{aligned} \mathbb{F}[x]_{<m} \times \mathbb{F}[x]_{<n} &\to \mathbb{F}[x]_{<n+m} \\ (a, b) &\mapsto af + bg \end{aligned}$$

written in the monomial basis $\{1, x, x^2, \ldots\}$. One can show that this map is an isomorphism of vector spaces if and only if $\gcd(f, g) = 1$ (see, e.g., [vzGG13, Section 6.3]). In particular, the resultant $\operatorname{res}(f, g) = \det \operatorname{Syl}(f, g)$ characterizes when two polynomials share a common factor.

Lemma 2.15 (see, e.g., [vzGG13, Corollary 6.17])**.** *Let $f, g \in \mathbb{F}[x]$. Then $\gcd(f, g) = 1$ if and only if $\operatorname{res}(f, g) \neq 0$. Moreover, $\gcd(f, g) = 1$ if and only if $\operatorname{Syl}(f, g)$ is invertible.*

From the description of the Sylvester matrix as the linear map $(a, b) \mapsto af + bg$, it is easy to see that the inverse of $\mathrm{Syl}(f, g)$ allows us to recover the Bézout coefficients of f and g when $\gcd(f, g) = 1$. More generally, for any $\ell \in [n + m]$, the entries of the ℓ^{th} column of $\mathrm{Syl}(f, g)^{-1}$ correspond to the coefficients of polynomials $a_\ell, b_\ell \in \mathbb{F}[x]$ such that $a_\ell f + b_\ell g = x^{n+m-\ell}$. We record this observation in the following lemma.

Lemma 2.16. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degrees n and m, respectively. Assume that $\gcd(f, g) = 1$, so $\mathrm{Syl}(f, g)^{-1}$ is invertible. For $\ell \in [n + m]$, the ℓ^{th} column of $\mathrm{Syl}(f, g)^{-1}$ corresponds to the coefficients of polynomials $a_\ell, b_\ell \in \mathbb{F}[x]$ such that*

1. *$\deg(a_\ell) < \deg(g)$,*
2. *$\deg(b_\ell) < \deg(f)$, and*
3. *$a_\ell f + b_\ell g = x^{n+m-\ell}$.*

If the polynomials f and g happen to split into linear factors (as they do over an algebraically closed field), then we can write the resultant $\mathrm{res}(f, g)$ as a simple function of the roots of f and g. Although we will not directly compute the roots of f or g, the following identity will be crucial in designing an AC^0 circuit to compute the resultant.

Lemma 2.17 (see, e.g., [CLO05, Section 3.1])**.** *Let $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$ and $g(x) = \prod_{i=1}^{m}(x - \beta_i)$ be univariate polynomials. Then the resultant $\mathrm{res}(f, g)$ is given by*

$$\mathrm{res}(f, g) = \prod_{i=1}^{n}\prod_{j=1}^{m}(\alpha_i - \beta_j) = \prod_{i=1}^{n} g(\alpha_i) = (-1)^{nm}\prod_{i=1}^{m} f(\beta_i).$$

The resultant also allows us to detect when a polynomial has a double root. Recall that a *double root* of a polynomial $f \in \mathbb{F}[x]$ is a point $\alpha \in \mathbb{F}$ such that $f(\alpha) = f'(\alpha) = 0$. That is, a double root corresponds to a shared root of f and its derivative. This leads to the discriminant of a polynomial, a specialization of the resultant.

Definition 2.18. Let $f(x) \in \mathbb{F}[x]$ be a monic univariate polynomial. The *discriminant* of f, denoted $\mathrm{disc}(f)$, is defined as $\mathrm{disc}(f) \coloneqq (-1)^{\binom{n}{2}} \mathrm{res}(f, f')$. ◊

For quadratic polynomials, the discriminant is given by the familiar formula

$$\mathrm{disc}(ax^2 + bx + c) = b^2 - 4ac.$$

As remarked above, a double root is precisely a common root of f and its derivative f'. The discriminant detects when a polynomial has a double root.

Lemma 2.19. *Let $f \in \mathbb{F}[x]$. Then f has a double root if and only if $\mathrm{disc}(f) = 0$.*

Just as we can express the resultant $\mathrm{res}(f, g)$ as a simple function of the roots of f and g, we can likewise express the discriminant $\mathrm{disc}(f)$ as a simple function of the roots of f.

Lemma 2.20. *Let $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$ be a univariate polynomial. Then*

$$\mathrm{disc}(f) = (-1)^{\binom{n}{2}} \prod_{\substack{i,j\in[n]\\ i\neq j}} (\alpha_i - \alpha_j) = \prod_{i<j}(\alpha_i - \alpha_j)^2.$$

The resultant was originally found by Bézout [Béz64] as the determinant of what is now called the Bézout matrix of two polynomials, which we define below.

Definition 2.21. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most n. The *Bézout matrix of order n* associated with the polynomials f and g, denoted by $\mathrm{Bez}_n(f, g)$, is the $n \times n$ matrix that satisfies the identity

$$\frac{f(x)g(y) - f(y)g(x)}{x - y} = \sum_{i,j=0}^{n-1} \mathrm{Bez}_n(f, g)_{i,j} x^i y^j. \qquad \Diamond$$

For more on Bézout matrices and their applications to control theory and elimination theory, we refer to [BP94, Chapter 2, Section 9] and references therein. In this work, we restrict our attention to computing the determinant and inverse of Bézout matrices.

The determinant of the Bézout matrix provides an alternate way to compute the resultant of two polynomials.

Lemma 2.22. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials and let $d = \max(\deg(f), \deg(g))$. Then* $\det \mathrm{Bez}_d(f, g) = \mathrm{res}(f, g)$.

The inverse of the Bézout matrix $\mathrm{Bez}_n(f, g)$ is easily described in terms of the polynomials f and g. The following proposition shows that the inverse of a Bézout matrix is a Hankel matrix whose entries can be efficiently computed from the coefficients of f and g. Conversely, the inverse of any Hankel matrix is a Bézout matrix of some pair of polynomials.

Proposition 2.23 ([BP94, Chapter 2, Proposition 9.3])**.** *Let $f \in \mathbb{F}[x]$ be a monic polynomial of degree n and let $g \in \mathbb{F}[x]$ be a polynomial of degree at most n. If $\mathrm{res}(f, g) \neq 0$, then $\mathrm{Bez}_n(f, g)$ is invertible. Moreover, the inverse $\mathrm{Bez}_n(f, g)^{-1}$ can be described as follows. Let $p \in \mathbb{F}[x]$ be a polynomial of degree at most $n - 1$ that satisfies the congruence*

$$p(x)g(x) \equiv 1 \pmod{f(x)}.$$

Let $h(x) = \sum_{i=0}^{\infty} h_i x^i$ be the power series expansion of $\frac{x^n p(1/x)}{x^n f(1/x)}$. Then the inverse of $\mathrm{Bez}_n(f, g)$ is given by the Hankel matrix

$$\mathrm{Bez}_n(f, g)^{-1} = \begin{pmatrix} h_1 & h_2 & h_3 & \cdots & h_n \\ h_2 & h_3 & h_4 & \cdots & h_{n+1} \\ h_3 & h_4 & h_5 & \cdots & h_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_n & h_{n+1} & h_{n+2} & \cdots & h_{2n-1} \end{pmatrix}.$$

2.5 Squarefree Decomposition

This subsection recalls the *squarefree decomposition* of a polynomial, a structured partial factorization. We first recall the notion of a squarefree polynomial.

Definition 2.24. Let $f \in \mathbb{F}[x]$ be a univariate polynomial. We say that f is *squarefree* if there is no non-constant polynomial $g \in \mathbb{F}[x]$ such that g^2 divides f. $\Diamond$

As an example, the polynomial $(x - 1)(x - 2)$ is squarefree, but $(x - 1)^2(x - 2)$ is not. By factoring a polynomial f as a product of irreducible polynomials and erasing the exponents, one obtains the squarefree part of f.

Definition 2.25. Let $f \in \mathbb{F}[x]$ be a univariate polynomial. Let $f = \prod_{i=1}^{m} f_i^{d_i}$ be the factorization of f into irreducible polynomials, where $f_1, \ldots, f_m \in \mathbb{F}[x]$ are irreducible in $\mathbb{F}[x]$ and are pairwise coprime. The *squarefree part* of f is given by $\prod_{i=1}^{m} f_i$. $\lozenge$

Finally, we define the squarefree decomposition of a polynomial $f \in \mathbb{F}[x]$. The squarefree decomposition is a partial factorization of f into a structured product of squarefree polynomials. The task of computing the squarefree decomposition is referred to as *squarefree factorization* and is a basic step in algorithms for factoring polynomials; see [vzGG13, Chapter 14] for more.

Definition 2.26. Let $f \in \mathbb{F}[x]$ be a univariate polynomial. The *squarefree decomposition* of $f(x)$ is the unique sequence of monic squarefree pairwise coprime polynomials $(f_1, \ldots, f_m)$ such that $f = \prod_{i=1}^{m} f_i^i$ and $f_m \neq 1$. $\lozenge$

3 Symmetric Polynomials and Newton's Identities

In this section, we study two important families of symmetric polynomials, the elementary symmetric and power sum polynomials. Applying these functions to the roots of a univariate polynomial f give different representations of f, and these representations are useful for different algorithmic tasks. As we will see, the representation using the elementary symmetric polynomials works nicely with additive operations, while the power sum polynomials are better suited for multiplicative operations. It is extremely convenient that one can pass between these representations in AC^0—we exposit these reductions in this section.

Suppose we are given a polynomial $f \in \mathbb{F}[x]$ by its list of coefficients. What do the coefficients tell us about the polynomial? Recall that if f factorizes as $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$, then f can be written as

$$f(x) = \sum_{i=0}^{n} (-1)^{n-i} e_{n-i}(\overline{\alpha})\, x^i,$$

where the polynomial $e_d(x_1, \ldots, x_n)$ is given by

$$e_d(x_1, \ldots, x_n) = \sum_{\substack{S \subseteq [n] \\ |S| = d}} \prod_{i \in S} x_i.$$

That is, the degree-i coefficient of f is, up to a sign, the elementary symmetric polynomial of degree $n - i$ evaluated at $\overline{\alpha}$.

The elementary symmetric polynomials are essential in the study of symmetric polynomials, i.e., polynomials that are invariant under permutations of the variables. The *fundamental theorem of symmetric polynomials* says that for any symmetric polynomial $f(\overline{x})$, there exists a (not necessarily symmetric) polynomial $g(\overline{y})$ such that

$$f(\overline{x}) = g(e_1(\overline{x}), \ldots, e_n(\overline{x})),$$

and moreover this polynomial g is unique. That is, any symmetric polynomial can be written as a polynomial combination of the elementary symmetric polynomials. This means that from the coefficients of f, we can compute *any* symmetric function of the roots $\alpha_1, \ldots, \alpha_n$. The central theme of this paper is to understand which of these functions can be computed *efficiently*.

In this section, we study an important family of symmetric polynomials, the power sum polynomials. Recall that the degree-d power sum polynomial $p_d(x_1, \ldots, x_n)$ is given by

$$p_d(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i^d.$$

An analogue of the fundamental theorem of symmetric polynomials holds for the power sum polynomials when $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n$. Given a symmetric polynomial $f(\overline{x})$, there is a (not necessarily symmetric) polynomial $h(\overline{y})$ such that

$$f(\overline{x}) = h(p_1(\overline{x}), \ldots, p_n(\overline{x})),$$

and like before, this h is unique.

The two versions of the fundamental theorem of symmetric polynomials mentioned above imply that if we are given the coefficients $e_1(\overline{\alpha}), \ldots, e_n(\overline{\alpha})$ of a polynomial f, we can compute the power sums $p_1(\overline{\alpha}), \ldots, p_n(\overline{\alpha})$ of its roots, and vice-versa. This computation can be made efficient by making use of explicit identities that relate the elementary symmetric and power sum polynomials. Such identities are well-known, and go by the name of Newton's identities (or the Girard–Newton identities). For $1 \leqslant d \leqslant n$, we have

$$p_d(\overline{\alpha}) = (-1)^{d-1} d \cdot e_d(\overline{\alpha}) + \sum_{i=1}^{d-1} (-1)^{d-1+i} e_{d-i}(\overline{\alpha}) p_i(\overline{\alpha}),$$

and for $d > n$, we instead have

$$0 = \sum_{i=d-n}^{d} (-1)^{i-1} e_{d-i}(\overline{\alpha}) p_i(\overline{\alpha}).$$

If we are given the values of $e_1(\overline{\alpha}), \ldots, e_n(\overline{\alpha})$, we can compute the values of $p_d(\overline{\alpha})$ iteratively using dynamic programming. Conversely, when $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n$, Newton's identities show that the elementary symmetric polynomials can be computed from the power sum polynomials. The requirement that the field has large characteristic stems from the need to invert d when computing the value of $e_d(\overline{\alpha})$. As before, given the values of $p_1(\overline{\alpha}), \ldots, p_n(\overline{\alpha})$, we can iteratively compute the $e_d(\overline{\alpha})$ via dynamic programming.

Although these conversions between the elementary symmetric and power sum polynomials are efficient, their natural implementations are iterative, which would yield circuits of large depth. To obtain AC^0 algorithms, we make use of well-known alternate forms of Newton's identities as identities of formal power series. These identities and algorithms are by no means new. For example, Shpilka and Wigderson [SW01] made use of these identities to compute the elementary symmetric polynomials using circuits of depth 4 and depth 6 that are smaller than the circuits of size $O(n^2)$ and depth 3 constructed by Ben-Or. Because a low-depth implementation of Newton's identities plays such a fundamental role in our work, we include a description of it for the sake of completeness.

We now define the *Newton series* of a polynomial $f(x)$. This is the formal power series whose coefficients are the power sum polynomials evaluated at the roots of f.

Definition 3.1 ([BFSS06])**.** Let $f(x) \in \mathbb{F}[x]$ be a univariate polynomial of degree n. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ be the roots of f, counted with multiplicity. The *Newton series* of f, denoted $\mathrm{Newton}(f)$, is the formal power series in $\mathbb{F}[\![t]\!]$ given by

$$\mathrm{Newton}(f) \coloneqq \sum_{k=0}^{\infty} p_k(\overline{\alpha})\, t^k. \qquad \lozenge$$

3.1 From Coefficients to Power Sums

We will now see how to efficiently convert between the values of $e_1(\overline{\alpha}), \ldots, e_n(\overline{\alpha})$ and $p_1(\overline{\alpha}), \ldots, p_n(\overline{\alpha})$. That is, we will convert between the representations of a polynomial $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$ by its coefficients and by its Newton series $\mathrm{Newton}(f)$.

We first consider the task of computing Newton(f) up to a specified degree, given the coefficients of f. To do this, we make use of the fact that Newton(f) can be expressed as a rational function in t, where the numerator and denominator can be easily computed from f. Below, we recall the notion of the *reversal* of a polynomial.

Definition 3.2. Let $f(x) = \sum_{i=0}^{n} a_i x^i$ be a univariate polynomial of degree n with $a_n \neq 0$. The *reversal* of f, denoted $\mathrm{rev}(f)$, is the univariate polynomial

$$\mathrm{rev}(f)(x) \coloneqq \sum_{i=0}^{n} a_{n-i} x^i. \qquad \lozenge$$

The Newton series Newton(f) of a polynomial f can be written as a rational function in terms of the reversal of f.

Lemma 3.3 ([BFSS06, Lemma 1])**.** *Let $f(x) = \sum_{i=0}^{n} a_i x^i$ be a univariate polynomial of degree n with $a_n \neq 0$. Then* Newton(f) *is a rational function in t given by*

$$\mathrm{Newton}(f) = \frac{\mathrm{rev}(f')(t)}{\mathrm{rev}(f)(t)}.$$

Using the above expression for Newton(f) as a rational function, we can compute Newton(f) to degree d by expanding $\frac{1}{\mathrm{rev}(f)(t)}$ as a power series in t up to degree d, multiplying by $\mathrm{rev}(f')(t)$, and using polynomial interpolation to recover the coefficients of Newton(f).

Lemma 3.4. *Let $f \in \mathbb{F}[x]$ be a univariate polynomial of degree n and let $d \geqslant n$. Then the coefficients of* Newton(f) *up to degree d can be computed in AC^0.*

Proof. Write $f(x) = \sum_{i=0}^{n} f_i x^i$. Let $g(x) \coloneqq \mathrm{rev}(f')(x)$ and let $h(x) \coloneqq \mathrm{rev}(f)(x) - f_n$. Note that $h(0) = 0$. Let p_k be the degree-k coefficient of Newton(f). Lemma 3.3 implies

$$\sum_{k=0}^{\infty} p_k x^k = \frac{g(x)}{f_n + h(x)}.$$

Because $h(0) = 0$, we can invert $f_n + h(x)$ in $\mathbb{F}[\![x]\!]$, giving us

$$\sum_{k=0}^{\infty} p_k h^k = g(x) \sum_{k=0}^{\infty} \left(\frac{-h(x)}{f_n} \right)^k.$$

This implies

$$g(x) \sum_{k=0}^{d} \left(\frac{-h(x)}{f_n} \right)^k = \sum_{k=0}^{d} p_k x^k + x^{d+1} r(x)$$

for some polynomial $r(x) \in \mathbb{F}[x]$. Let $s(x) \coloneqq g(x) \sum_{k=0}^{d} \left(\frac{-h(x)}{f_n} \right)^k$. The above equality implies that we can recover the desired coefficients $p_0, \ldots, p_d$ of Newton(f) by interpolating the coefficients of $s(x)$.

Observe that the coefficients of $g(x)$ and $h(x)$ can be computed from the coefficients of $f(x)$ in a straightforward manner by a circuit of depth 1 and size $O(n)$. This, together with the definition of $s(x)$, yields a circuit of size $O(n + d)$ and depth 4 that computes $s(x)$. Note that $\deg(s) \leqslant d(n + 1)$. By Lemma 2.7, we can compute the coefficients of $s(x)$ using a circuit of size $O(nd^2)$ and depth 5 as desired. $\square$

3.2 From Power Sums to Coefficients

To recover a polynomial from its Newton series, we make use of the following identity. Shpilka and Wigderson [SW01] used this identity to construct smaller low-depth circuits for the elementary symmetric polynomials.

Lemma 3.5. *Fix $n \in \mathbb{N}$. Let $\exp(t) \coloneqq \sum_{k=0}^{\infty} \frac{1}{k!} t^k \in \mathbb{F}[\![t]\!]$ denote the exponential power series. Then we have the equality of formal power series*

$$\sum_{k=0}^{n} e_k(\overline{x}) t^k = \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} p_k(\overline{x}) t^k \right).$$

Note that in a sense, the power sum polynomials correspond to the logarithm of the elementary symmetric polynomials. We will witness the power of this in the next section.

We now show how to recover the coefficients of a polynomial from its Newton series. Note that $\mathrm{Newton}(f) = \mathrm{Newton}(\alpha f)$ for any polynomial f and nonzero $\alpha \in \mathbb{F}$, so we cannot hope to recover the leading coefficient of f. This will not be an issue for us, as we restrict our attention to monic polynomials throughout this work.

Lemma 3.6. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $f \in \mathbb{F}[x]$ be a univariate polynomial of degree n. Given the coefficients of $\mathrm{Newton}(f)$ of degree up to n as input, the coefficients of $f(x)$ can be computed in AC^0.*

Proof. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ denote the roots of $f(x)$. Recall that $e_k(\overline{\alpha})$ is the coefficient of x^{n-k} in $f(x)$ and $p_k(\overline{\alpha})$ is the coefficient of t^k in $\mathrm{Newton}(f)$. Lemma 3.5 implies

$$\sum_{k=0}^{n} e_k(\overline{\alpha}) x^k = \exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} p_k(\overline{\alpha}) x^k \right).$$

Let

$$g(x) \coloneqq \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k} p_k(\overline{\alpha}) x^k.$$

Note that because $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n$, the polynomial g is well-defined. The preceding identity implies that

$$\sum_{k=0}^{n} \frac{g(x)^k}{k!} = \sum_{k=0}^{n} e_k(\overline{\alpha}) x^k + x^{n+1} h(x),$$

where $h(x) \in \mathbb{F}[x]$ is some polynomial in x. Let $r(x) \coloneqq \sum_{k=0}^{n} \frac{g(x)^k}{k!}$. (As with g, the polynomial r is well-defined because $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > n$.) By interpolating the coefficients of $r(x)$, we can recover the values $e_1(\overline{\alpha}), \ldots, e_n(\overline{\alpha})$.

Observe that the coefficients of $g(x)$ are easily determined from the coefficients of $\mathrm{Newton}(f)$. This yields an arithmetic circuit of size $O(n)$ and depth 1 that computes $g(x)$. The truncated exponential $r(x)$ can then be computed by an arithmetic circuit of size $O(n)$ and depth 3. Note that $\deg(r) \leqslant n^2$, so Lemma 2.7 implies that we can compute the coefficients of r using an arithmetic circuit of size $O(n^3)$ and depth 4. In particular, this circuit computes the values $e_1(\overline{\alpha}), \ldots, e_n(\overline{\alpha})$. The polynomial $f(x)$ is given by $f(x) = \sum_{i=0}^{n} (-1)^{n-i} e_{n-i}(\overline{\alpha}) x^i$, so multiplying each $e_i(\overline{\alpha})$ by $(-1)^i$ results in the coefficients of $f(x)$. □

4 Exact Division and Roots of Perfect Powers

In this section, we warm up with two applications of the algorithms described in Section 3. We design AC^0 algorithms to compute the quotient f/g when g is promised to divide f, and to compute $f^{1/r}$ when f is promised to be an r^{th} power of a polynomial. These are toy problems, and the simple algorithms we design are meant to illustrate the utility of representing a polynomial by its Newton series. The Newton series plays the role of a logarithm, converting multiplication to addition and division to subtraction, which is easy to see from the fact that $\text{Newton}(fg) = \text{Newton}(f) + \text{Newton}(g)$.

We now describe an AC^0 algorithm to compute the quotient f/g when g is promised to divide f.

Lemma 4.1. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be monic univariate polynomials of degree at most d given by their coefficients. Suppose that g divides f. Then the coefficients of f/g can be computed in AC^0.*

Proof. Suppose $f(x)$ and $g(x)$ factor as

$$f(x) = \prod_{i=1}^{n}(x - \alpha_i)\prod_{i=1}^{m}(x - \beta_i)$$
$$g(x) = \prod_{i=1}^{n}(x - \alpha_i),$$

where $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \in \mathbb{F}$ and the α_i and β_i are not necessarily distinct. Applying Lemma 3.4 to $f(x)$ and $g(x)$, we can compute the power sums $p_k(\overline{\alpha}) + p_k(\overline{\beta})$ and $p_k(\overline{\alpha})$, respectively, for all $k \in [n+m]$ in AC^0. Taking differences, we obtain the power sums $p_k(\overline{\beta})$ for all $k \in [m]$. Applying Lemma 3.6 to the power sums $p_k(\overline{\beta})$ yields the coefficients of $\prod_{i=1}^{m}(x - \beta_i)$ in AC^0 as desired. □

Next, we describe an AC^0 algorithm to compute the r^{th} root $f^{1/r}$ when f is promised to be an r^{th} power of a polynomial.

Lemma 4.2. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f \in \mathbb{F}[x]$ be a univariate polynomial of degree at most d given by its coefficients and let $r \in \mathbb{N}$. Suppose that $f = g^r$ for some $g \in \mathbb{F}[x]$. Then the coefficients of g can be computed in AC^0.*

Proof. Suppose that g factors as

$$g(x) = \prod_{i=1}^{n}(x - \alpha_i),$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and the α_i are not necessarily distinct. Because $f = g^r$, this implies that f factors as

$$f(x) = \prod_{i=1}^{n}(x - \alpha_i)^r.$$

Applying Lemma 3.4 to f, we compute in AC^0 the power sums

$$\sum_{i=1}^{n} r \cdot \alpha_i^k = r \cdot p_k(\overline{\alpha})$$

for each $k \in [n]$. Dividing by r yields $p_k(\overline{\alpha})$ for all $k \in [n]$. Applying Lemma 3.6 to the power sums $p_k(\overline{\alpha})$ produces the coefficients of g, as desired. □

5 Computing Symmetric Functions of the Roots of a Polynomial

In this section, we return to the topic of computing symmetric functions of the roots of a polynomial $f \in \mathbb{F}[x]$ when f is given by its coefficients. The coefficients of f are the elementary symmetric functions of its roots. As we saw in Section 3, we can compute the power sums of the roots in AC^0. In this section, we expand our toolbox, finding more symmetric functions that can be evaluated at the roots of a given polynomial f in AC^0.

5.1 Polynomial Functions

Let $f, g \in \mathbb{F}[x]$ be univariate polynomials and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ be the roots of f, counted with multiplicity. In this subsection, we evaluate the elementary symmetric polynomials at the values $g(\alpha_1), \ldots, g(\alpha_n)$.

We start by computing the sum $\sum_{i=1}^{n} g(\alpha_i)$.

Lemma 5.1. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the roots of f, counted with multiplicity. Then the sum $\sum_{i=1}^{n} g(\alpha_i)$ can be computed in AC^0.*

Proof. Letting $g(x) = \sum_{i=0}^{m} g_i x^i$, we can rewrite the sum $\sum_{i=1}^{n} g(\alpha_i)$ as

$$\sum_{i=1}^{n} g(\alpha_i) = \sum_{i=1}^{n} \sum_{j=0}^{m} g_j \alpha_i^j = \sum_{j=0}^{m} g_j \sum_{i=1}^{n} \alpha_i^j = \sum_{j=0}^{m} g_j p_j(\overline{\alpha}).$$

We can compute the power sums $p_k(\overline{\alpha})$ for $0 \leqslant k \leqslant m$ in AC^0 by applying Lemma 3.4 to $f(x)$. The sum $\sum_{j=0}^{m} g_j p_j(\overline{\alpha})$ can then be computed by a subcircuit of size $O(m)$ and depth 2. □

This shows that the sum $\sum_{i=1}^{n} g(\alpha_i)$ is easy to compute. It would be interesting to find an AC^0 algorithm to compute the product $\prod_{i=1}^{n} g(\alpha_i)$, since this equals $\operatorname{res}(f, g)$ by Lemma 2.17. We will not only compute the product $\prod_{i=1}^{n} g(\alpha_i)$, but we will in fact compute all the elementary symmetric polynomials evaluated at $g(\alpha_1), \ldots, g(\alpha_n)$. To do this, it suffices (by Newton's identities) to compute the power sums $\sum_{i=1}^{n} g(\alpha_i)^k$, which can be done by a straightforward application of Lemma 5.1.

Lemma 5.2. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the roots of f, counted with multiplicity. Then for any $d \in [n]$, the elementary symmetric polynomial $e_d(g(\alpha_1), \ldots, g(\alpha_n))$ can be computed in AC^0.*

Proof. For any $k \in [n]$, the coefficients of $g(x)^k$ can be computed in AC^0 using Lemma 2.7. By Lemma 5.1, we can compute the sum $\sum_{i=1}^{n} g(\alpha_i)^k$ in AC^0 for each $k \in [n]$. Applying Lemma 3.6 to the power sums $\{p_k(g(\alpha_1), \ldots, g(\alpha_n)) : k \in [n]\}$ yields the elementary symmetric polynomials $\{e_k(g(\alpha_1), \ldots, g(\alpha_n)) : k \in [n]\}$, again in AC^0. □

Later in Section 7, it will be useful for us to compute the product of only the nonzero values of $g(\alpha_i)$. We can do this by computing all elementary symmetric polynomials $e_k(g(\alpha_1), \ldots, g(\alpha_n))$ and then (piecewise) selecting the largest index k such that $e_k(g(\alpha_1), \ldots, g(\alpha_n)) \neq 0$.

Lemma 5.3. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the roots of f, counted with multiplicity. Let $S = \{i \in [n] : g(\alpha_i) \neq 0\}$. Then the product $\prod_{i \in S} g(\alpha_i)$ can be computed piecewise in AC^0.*

Proof. By relabeling the roots of f, we may assume without loss of generality that $S = \{1, \dots, d\}$. Let

$$\begin{aligned} \lambda_1 &\coloneqq g(\alpha_1) \neq 0 \\ &\vdots \\ \lambda_d &\coloneqq g(\alpha_d) \neq 0 \\ \lambda_{d+1} &\coloneqq g(\alpha_{d+1}) = 0 \\ &\vdots \\ \lambda_n &\coloneqq g(\alpha_n) = 0, \end{aligned}$$

where $d \in \mathbb{N}$ is unknown to us. By Lemma 5.2, we can compute $e_k(\overline{\lambda})$ for all $k \in [n]$ in AC^0, and we will leverage this to compute $\prod_{i=1}^{d} \lambda_i$.

When $k > d$, the degree-k elementary symmetric polynomial evaluated at $\overline{\lambda}$ vanishes. To see this, expand the elementary symmetric polynomial as

$$e_k(\lambda_1, \dots, \lambda_n) = \sum_{\substack{S \subseteq [d] \\ T \subseteq \{d+1, \dots, n\} \\ |S|+|T|=k}} \prod_{i \in S} \lambda_i \prod_{j \in T} \lambda_j.$$

For $k > d$, every term in the above sum corresponds to a choice of $T \subseteq \{d+1, \dots, n\}$ that is nonempty. Each such term incurs a factor of $\lambda_j = 0$, so the sum simplifies to zero.

On the other hand, the degree-d elementary symmetric polynomial evaluated at $\overline{\lambda}$ equals $\prod_{i=1}^{d} \lambda_i$. This follows from the fact that every term in the expansion of $e_d(\overline{\lambda})$ corresponding to a nonempty choice of $T \subseteq \{d+1, \dots, n\}$ is zero, so the sum simplifies to the single nonzero term corresponding to $T = \varnothing$.

This allows us to compute the product $\prod_{i=1}^{d} \lambda_i$ if we know the value of d. We can recover d by finding the largest index $\hat{d}$ at which the degree-$\hat{d}$ elementary symmetric polynomial is nonzero. This $\hat{d}$ is precisely d, the number of nonzero λ_i.

To formalize this algorithm as a piecewise AC^0 circuit, we use the circuit computing $e_k(\overline{\lambda})$ as both the $(n-k+1)$-th computation circuit and test circuit. □

Remark 5.4. Note that Lemmas 5.1 to 5.3 extend to the setting where $g \in \mathbb{F}[x, \overline{y}]$ is a polynomial in many variables. In this variant, we want to compute the sum

$$\sum_{i=1}^{n} g(\alpha_i, \overline{y}),$$

and more generally the elementary symmetric polynomial

$$e_d(g(\alpha_1, \overline{y}), \dots, g(\alpha_n, \overline{y})).$$

To do this, regard $g \in \mathbb{F}[\overline{y}][x]$ as a univariate polynomial in x whose coefficients are polynomials in $\overline{y}$. When g is given by its coefficients as a polynomial in $\mathbb{F}[x, \overline{y}]$, it is straightforward to form the coefficients of $g \in \mathbb{F}[\overline{y}][x]$ in AC^0. If g is instead given by an AC^0 circuit, then we can obtain AC^0 circuits that compute the coefficients of $g \in \mathbb{F}[\overline{y}][x]$ using Lemma 2.7. With this modification, the proofs of Lemmas 5.1 to 5.3 go through without modification. The ability to apply Lemmas 5.1 to 5.3 in this setting will be an essential tool that we use throughout our work. ◊

5.2 Rational Functions

Lemmas 5.1 and 5.2 generalize to the setting where we want to evaluate a rational function g/h at the roots of a polynomial f. We start by summing a rational function g/h over the roots of f. Of course, this requires that h is nonzero at the roots of f.

Lemma 5.5. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $f, g, h \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the roots of f, counted with multiplicity. Assume that $h(\alpha_i) \neq 0$ for all $i \in [n]$. Then the sum $\sum_{i=1}^{n} \frac{g(\alpha_i)}{h(\alpha_i)}$ can be computed in AC^0.*

Proof. Writing the sum $\sum_{i=1}^{n} \frac{g(\alpha_i)}{h(\alpha_i)}$ over a common denominator, we have

$$\sum_{i=1}^{n} \frac{g(\alpha_i)}{h(\alpha_i)} = \frac{\sum_{i=1}^{n} g(\alpha_i) \prod_{j \neq i} h(\alpha_j)}{\prod_{i=1}^{n} h(\alpha_i)}.$$

Applying Lemma 5.2 to f and h allows us to compute $\prod_{i=1}^{n} h(\alpha_i)$ in AC^0, so we are left with the task of computing $\sum_{i=1}^{n} g(\alpha_i) \prod_{j \neq i} h(\alpha_j)$.

Let y be a fresh variable. Observe that when we expand the polynomial

$$r(y) \coloneqq \prod_{i=1}^{n} (g(\alpha_i)y + h(\alpha_i)),$$

the coefficient of the degree-1 term is precisely $\sum_{i=1}^{n} g(\alpha_i) \prod_{j \neq i} h(\alpha_j)$. By applying Lemma 5.2 to the polynomials $f(x)$ and $g(x)y + h(x)$, we obtain a circuit of constant depth and polynomial size that computes $r(y)$. As $\deg(r) = n$, Lemma 2.7 implies that we can compute the coefficients of $r(y)$ in AC^0. This yields an AC^0 algorithm to compute the sum $\sum_{i=1}^{n} g(\alpha_i) \prod_{j \neq i} h(\alpha_j)$, and hence an AC^0 algorithm to compute $\sum_{i=1}^{n} \frac{g(\alpha_i)}{h(\alpha_i)}$. □

We now use Lemma 5.5 to compute any elementary symmetric function of the values $\frac{g(\alpha_1)}{h(\alpha_1)}, \ldots, \frac{g(\alpha_n)}{h(\alpha_n)}$.

Lemma 5.6. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $f, g, h \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the roots of f, counted with multiplicity. Assume that $h(\alpha_i) \neq 0$ for all $i \in [n]$. Then for any $d \in [n]$, the elementary symmetric function*

$$e_d\left(\frac{g(\alpha_1)}{h(\alpha_1)}, \ldots, \frac{g(\alpha_n)}{h(\alpha_n)}\right)$$

can be computed in AC^0.

Proof. For any $k \in [n]$, we can compute the coefficients of $g(x)^k$ and $h(x)^k$ in AC^0 using Lemma 2.7. Applying Lemma 5.5 to $f(x)$, $g(x)^k$, and $h(x)^k$ computes the sum $\sum_{i=1}^{n} \frac{g(\alpha_i)^k}{h(\alpha_i)^k}$ in AC^0. We then compute the desired elementary symmetric polynomial by invoking Lemma 3.6 on the power sums $\sum_{i=1}^{n} \frac{g(\alpha_i)^k}{h(\alpha_i)^k}$. □

6 The Sylvester and Bézout Matrices

In this section, we apply the results of Section 5 to compute the determinant and inverse of the Sylvester and Bézout matrices of a pair of polynomials in AC^0. We also extend our division algorithm from Lemma 4.1 to handle polynomial division with remainder.

6.1 The Resultant and Discriminant

We start by designing a constant-depth circuit to compute the resultant of two polynomials. As the resultant is precisely the determinant of the Sylvester (Definition 2.14) and Bézout matrices (Lemma 2.22), this provides an AC^0 algorithm to compute the determinants of matrices of these forms.

To compute the resultant, let $f, g \in \mathbb{F}[x]$ be monic polynomials and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ be the roots of f, counted with multiplicity. By Lemma 2.17, we know that

$$\operatorname{res}(f, g) = \prod_{i=1}^{n} g(\alpha_i).$$

This is precisely the n^{th} elementary symmetric polynomial evaluated at $(g(\alpha_1), \ldots, g(\alpha_n))$. Thus, we can compute $\operatorname{res}(f, g)$ using a direct application of Lemma 5.2.

Theorem 6.1. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Then the resultant $\operatorname{res}(f, g)$ can be computed in AC^0.*

Proof. This is an immediate consequence of Lemma 2.17 and Lemma 5.2. □

As an immediate corollary, we obtain an AC^0 algorithm to compute the discriminant of a single polynomial.

Corollary 6.2. *Let $f \in \mathbb{F}[x]$ be a univariate polynomial given by its coefficients. Then the discriminant $\operatorname{disc}(f)$ can be computed in AC^0.*

Proof. Because $\operatorname{disc}(f) = (-1)^{\binom{n}{2}} \operatorname{res}(f, f')$, this immediately follows from Theorem 6.1. □

The resultant and discriminant are extremely useful tools in algebra and number theory. Using well-known applications of the resultant and discriminant, we present three corollaries of Theorem 6.1 and Corollary 6.2.

Our first corollary deals with computing implicit equations of rational plane curves. A *rational plane curve* is the (Zariski closure of the) image of a rational map

$$\begin{aligned} \gamma : \mathbb{F} &\to \mathbb{F}^2 \\ t &\mapsto \left(\frac{f(t)}{h(t)}, \frac{g(t)}{h(t)} \right) \end{aligned}$$

where $f, g, h \in \mathbb{F}[t]$ are univariate polynomials. For example, the unit circle has a rational parameterization given by

$$t \mapsto \left(\frac{1 - t^2}{1 + t^2}, \frac{2t}{1 + t^2} \right).$$

The careful reader will notice that $(-1, 0)$ on the unit circle, but is not in the image of this map. This is addressed by taking the closure of the image in the *Zariski topology*, the standard topology used in algebraic geometry. We will not define the Zariski topology here, and instead refer the reader interested in the precise details to [CLO15].

Given a rational parameterization of a plane curve Γ, it is often useful to find an implicit equation for Γ. An *implicit equation* is a polynomial $r \in \mathbb{F}[x, y]$ such that $r(a, b) = 0$ if and only if $(a, b) \in \Gamma$. Clearly, an implicit equation gives rise to an algorithm that decides if a given point (a, b) lies on the curve Γ: compute $r(a, b)$ and check if this value equals zero. In the case of the unit circle, one implicit

equation is given by $x^2 + y^2 - 1$. (Other implicit equations for the unit circle are $(x^2 + y^2 - 1)^n$ for $n \in \mathbb{N}$, and these are essentially all possible equations for the unit circle.)

Resultants provide a straightforward method to compute an implicit equation of a rational plane curve. Given a curve parameterized by

$$t \mapsto \left(\frac{f(t)}{h(t)}, \frac{g(t)}{h(t)}\right),$$

one can show that the polynomial

$$r(x, y) \coloneqq \operatorname{res}_t(x \cdot h(t) - f(t), y \cdot h(t) - g(t)),$$

where $x \cdot h(t) - f(t)$ and $y \cdot h(t) - g(t)$ are regarded as polynomials in t with coefficients in $\mathbb{F}[x, y]$, is an implicit equation of the plane curve. By computing this resultant with Theorem 6.1 and then interpolating the coefficients of $r(x, y)$ using Lemma 2.7, we obtain an AC^0 algorithm to compute an implicit equation of a given rational plane curve.

Corollary 6.3. *Let $f, g, h \in \mathbb{F}[t]$ be univariate polynomials given by their coefficients. Let $\Gamma \subseteq \mathbb{F}^2$ be the plane curve corresponding to the map $t \mapsto (f(t)/h(t), g(t)/h(t))$. Then the coefficients of an implicit equation $r \in \mathbb{F}[x, y]$ for Γ can be computed in AC^0.*

The second application of Theorem 6.1 is again to geometry. Given the implicit equations of two plane curves $\Gamma_1, \Gamma_2 \in \mathbb{F}^2$, we would like to compute their intersection $\Gamma_1 \cap \Gamma_2$. Equivalently, we want to solve the system of polynomial equations $f(x, y) = g(x, y) = 0$, where f and g are the implicit equations of Γ_1 and Γ_2, respectively. Using resultants, we can reduce this problem to solving polynomial equations in one variable. View f and g as elements of $\mathbb{F}[x][y]$ and let

$$h(x) \coloneqq \operatorname{res}_y(f(x, y), g(x, y))$$

be their resultant. If $h(x) = 0$, then f and g share a common factor, so the intersection $\Gamma_1 \cap \Gamma_2$ is infinite and corresponds to this common factor.

Suppose instead that $h(x) \neq 0$, so that $\Gamma_1 \cap \Gamma_2$ is finite. Standard properties of resultants imply that if $(a, b) \in \Gamma_1 \cap \Gamma_2$, then $h(a) = 0$. Thus, to compute $\Gamma_1 \cap \Gamma_2$, it suffices to first compute the roots of h, and for each such root $a \in \mathbb{F}$, find common roots of the univariate polynomials $f(a, y)$ and $g(a, y)$. For more on resultants in elimination theory, see [CLO15, Chapter 3].

Our third and final application is to combining the roots of polynomials in nontrivial ways. Suppose we are given the coefficients of $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$ and $g(x) = \prod_{i=1}^{m}(x - \beta_i)$. Consider the polynomials

$$(f \oplus g)(x) \coloneqq \prod_{i,j}(x - (\alpha_i + \beta_j))$$
$$(f \otimes g)(x) \coloneqq \prod_{i,j}(x - \alpha_i\beta_j),$$

which are called the *composed sum* and *composed product* of f and g, respectively.

The composed sum and composed product are useful in implementing arithmetic for algebraic numbers. Recall that a number $\alpha \in \mathbb{C}$ is *algebraic* if there is a nonzero polynomial $f \in \mathbb{Q}[x]$ such that $f(\alpha) = 0$. For every algebraic number $\alpha \in \mathbb{C}$, there is a nonzero polynomial $f \in \mathbb{Q}[x]$ of minimal degree that vanishes at α; such a polynomial is the *minimal polynomial* of α. A natural way to represent algebraic numbers is by their minimal polynomial. If $\alpha, \beta \in \mathbb{C}$ are represented by

polynomials f and g, respectively, then the sum $\alpha + \beta$ is a root of the composed sum $f \oplus g$. This implies that the minimal polynomial of $\alpha + \beta$ is an irreducible factor (over $\mathbb{Q}[x]$) of $f \oplus g$, which we can find by factoring $f \oplus g$. Likewise, the minimal polynomial of $\alpha\beta$ is an irreducible factor of $f \otimes g$.

Fast algorithms to compute the composed sum and composed product were given by Bostan, Flajolet, Salvy, and Schost [BFSS06], where the key tool was a fast algorithm to convert between the coefficient representation of a polynomial and its Newton series. Using properties of resultants, it is a straightforward exercise (see, e.g., [CLO15, Section 3.6, Exercise 19]) to show that

$$(f \oplus g)(x) = \operatorname{res}_y(f(y), g(x-y))$$
$$(f \otimes g)(x) = \operatorname{res}_y(f(y), y^m g(x/y)),$$

where $f(y)$, $g(x-y)$, and $y^m g(x/y)$ are viewed as polynomials in y with coefficients in $\mathbb{F}[x]$. As a corollary of Theorem 6.1, we conclude AC^0 algorithms to compute the composed sum and composed product. For more applications of the composed sum and composed product, see [BFSS06, Section 5].

Corollary 6.4. *Let $f, g \in \mathbb{F}[x]$ be univariate polynomials given by their coefficients. Suppose that f and g factor as $f(x) = \prod_{i=1}^{n}(x-\alpha_i)$ and $g(x) = \prod_{i=1}^{m}(x-\beta_i)$, where the $\alpha_i, \beta_i \in \mathbb{F}$ are not necessarily distinct, i.e., f and g are not necessarily squarefree. Then the coefficients of the polynomials*

$$(f \oplus g)(x) \coloneqq \prod_{i,j}(x - (\alpha_i + \beta_j))$$
$$(f \otimes g)(x) \coloneqq \prod_{i,j}(x - \alpha_i\beta_j)$$

can be computed in AC^0.

6.2 Division with Remainder

In this subsection, we take a brief detour from Sylvester and Bézout matrices to extend our division algorithm from Lemma 4.1 to handle polynomial division with remainder. Recall that if $f, g \in \mathbb{F}[x]$ are univariate polynomials, then there are unique polynomials $q, r \in \mathbb{F}[x]$ such that $f = qg + r$ and $\deg(r) < \deg(g)$. The polynomial r is the *remainder* of f divided by g. We will describe an AC^0 algorithm to compute the remainder r, which leads to an algorithm for division with remainder by dividing $f - r$ by g using Lemma 4.1. The technique we use to compute the remainder r will be useful later in Section 6.3, where we design an AC^0 algorithm to invert the Sylvester matrix $\operatorname{Syl}(f, g)$.

For the moment, suppose that g is squarefree, i.e., that g has m distinct roots $\beta_1, \ldots, \beta_m \in \mathbb{F}$, all of multiplicity 1. This is a mild assumption, and we will sketch how to remove it below. Assuming g is squarefree, we can explicitly write the remainder r using polynomial interpolation. Consider the polynomial $\hat{r}(x)$ obtained by interpolating the values of f at the points $\beta_1, \ldots, \beta_m$. The Lagrange interpolation formula lets us to write $\hat{r}$ as

$$\hat{r}(x) = \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} \frac{x - \beta_j}{\beta_i - \beta_j}.$$

By construction, we have $\deg(\hat{r}) \leqslant m - 1 < \deg(g)$. The polynomial $f - \hat{r}$ is zero at each β_i. Because g is squarefree, it follows that g divides $f - \hat{r}$. That is, there is some polynomial $\hat{q}$ such that $f - \hat{r} = \hat{q}g$. Rearranging, we have $f = \hat{q}g + \hat{r}$, so by uniqueness of the quotient and remainder, we have $q = \hat{q}$ and $r = \hat{r}$. Thus, to compute the remainder r, it suffices to interpolate a polynomial that

agrees with f at the roots of g, at least when g is squarefree. We will perform this interpolation by implementing the Lagrange interpolation formula without explicitly computing the roots of g, making use of the tools developed in Section 5.

When g is not squarefree, we can still perform the above interpolation, but g may not divide $f - \hat{r}$. If $\beta_i \in \mathbb{F}$ is a multiple root of g, we would need to interpolate $\hat{r}$ so that $f - \hat{r}$ vanishes at β_i to the same order as g. However, we do not know the multiset of multiplicities of the roots of g, so it is not clear that this strategy will generalize nicely.

Instead, we make use of the fact that a polynomial g is squarefree if and only if $\operatorname{disc}(g) \neq 0$. With $\hat{r}$ defined through Lagrange interpolation as above, it is easy to see that $\operatorname{disc}(g) \cdot \hat{r} = \operatorname{disc}(g) \cdot r$. When $\operatorname{disc}(g) = 0$, this is obvious, and when $\operatorname{disc}(g) \neq 0$, g is squarefree, so the preceding sketch implies $\hat{r} = r$. Thus, to obtain r, we can divide $\operatorname{disc}(g) \cdot \hat{r}$ by $\operatorname{disc}(g)$. Although the resulting circuit is defined only when $\operatorname{disc}(g) \neq 0$, Strassen's theorem on division elimination (Theorem 2.12) implies that we can transform this circuit into an equivalent one that does not use division. This results in a circuit that correctly computes the remainder r even when $\operatorname{disc}(g) = 0$.

We now design an AC^0 algorithm to compute polynomial remainders.

Lemma 6.5. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $r \in \mathbb{F}[x]$ be the remainder of f divided by g. Then the coefficients of r can be computed in AC^0.*

Proof. Let $n \coloneqq \deg(f)$ and $m \coloneqq \deg(g)$. Recall that $r \in \mathbb{F}[x]$ is the unique polynomial of degree less than m such that $f - r$ is a multiple of g. Below, we will show that the coefficients of $\operatorname{disc}(g)r(x)$ can be computed in AC^0. Before doing this, we explain why this yields an AC^0 algorithm to compute the coefficients of $r(x)$.

We can compute $\operatorname{disc}(g)$ in AC^0 via Corollary 6.2. By dividing $\operatorname{disc}(g)r(x)$ by $\operatorname{disc}(g)$, we obtain an AC^0 algorithm *with division* that computes $r(x)$. Because this algorithm divides by $\operatorname{disc}(g)$, its output is only defined when $\operatorname{disc}(g) \neq 0$. We would like to apply Theorem 2.12 to conclude a division-free AC^0 algorithm that computes the remainder $r(x)$, which would allow us to compute $r(x)$ even when $\operatorname{disc}(g) = 0$. If we could do this, then the coefficients of $r(x)$ can be computed in AC^0 by applying Lemma 2.7 to this division-free algorithm. To apply Theorem 2.12, we need to verify that the coefficients of $r(x)$ are polynomial functions of the coefficients of f and g.

To show that the coefficients of $r(x)$ are polynomial functions of the coefficients of f and g, let q be the quotient of f by g, i.e., let q be the polynomial such that $f = qg + r$. Write $q = \sum_{i=0}^{n-m} q_i x^i$ and $r = \sum_{i=0}^{m-1} r_i x^i$. By equating coefficients of powers of x on both sides of $f = qr + r$, we obtain the linear system

$$\begin{pmatrix} 1 & & & & & & \\ g_{m-1} & \ddots & & & & & \\ \vdots & & 1 & & & & \\ \vdots & & g_{m-1} & 1 & & & \\ g_0 & & \vdots & & 1 & & \\ & \ddots & \vdots & & & \ddots & \\ & & g_0 & & & & 1 \end{pmatrix} \begin{pmatrix} q_{n-m} \\ \vdots \\ q_0 \\ r_{m-1} \\ \vdots \\ r_0 \end{pmatrix} = \begin{pmatrix} 1 \\ f_{n-1} \\ f_{n-2} \\ \vdots \\ f_0 \end{pmatrix}.$$

Let $\widetilde{\operatorname{Syl}}(g, 1)$ denote the matrix on the left-hand side above. (This matrix corresponds to $\operatorname{Syl}(g, 1)$ when 1 is treated as a polynomial of degree $n - m + 1$.) The matrix $\widetilde{\operatorname{Syl}}(g, 1)$ is lower triangular

and has ones along the diagonal, so it clearly has determinant 1. This implies that the entries of $\widetilde{\mathrm{Syl}}(g,1)^{-1}$ are polynomial functions of the coefficients of g. It follows that

$$\begin{pmatrix} q_{n-m} \\ \vdots \\ q_0 \\ r_{m-1} \\ \vdots \\ r_0 \end{pmatrix} = \widetilde{\mathrm{Syl}}(g,1)^{-1} \begin{pmatrix} 1 \\ f_{n-1} \\ f_{n-2} \\ \vdots \\ f_0 \end{pmatrix}$$

are polynomial functions of the coefficients of f and g, as desired.

It remains to show that we can compute $\mathrm{disc}(g)r(x)$ in AC^0. Suppose $\mathrm{disc}(g) \neq 0$, so that g has m simple roots $\beta_1, \ldots, \beta_m \in \mathbb{F}$. In this case, we can explicitly write $r(x)$ using Lagrange interpolation as

$$r(x) = \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} \frac{x - \beta_j}{\beta_i - \beta_j}.$$

Combined with Lemma 2.20, we can write $\mathrm{disc}(g)r(x)$ as

$$\begin{aligned} \mathrm{disc}(g)r(x) &= \left((-1)^{\binom{m}{2}} \prod_{i \in [m]} \prod_{j \neq i} (\beta_i - \beta_j) \right) \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} \frac{x - \beta_j}{\beta_i - \beta_j} \\ &= (-1)^{\binom{m}{2}} \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} (x - \beta_j) \prod_{k \neq j} (\beta_j - \beta_k) \\ &= (-1)^{\binom{m}{2}} \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} (x - \beta_j) \cdot g'(\beta_j). \end{aligned}$$

Let y be a new variable. Observe that the above sum corresponds to the coefficient of y^{m-1} when the product

$$\prod_{i=1}^{m} (f(\beta_i) + y(x - \beta_i) g'(\beta_i))$$

is expanded as a polynomial in y with coefficients in $\mathbb{F}[x]$. This is precisely the kind of expression that Lemma 5.2 allows us to compute.

Let z be a fresh variable. Define $h(x, y, z) \in \mathbb{F}[x, y, z]$ by

$$h(x, y, z) \coloneqq f(z) + y(x - z) g'(z).$$

It is clear that we can compute $h(x, y, z)$ in AC^0. Applying Lemma 5.2, we compute

$$\prod_{i=1}^{m} h(x, y, \beta_i) = \prod_{i=1}^{m} (f(\beta_i) + y(x - \beta_i) g'(\beta_i)),$$

where $\beta_1, \ldots, \beta_m \in \mathbb{F}$ are the roots of g. We can then interpolate the coefficient of y^{m-1} in the above polynomial using Lemma 2.7. In all, this yields an AC^0 algorithm to compute

$$\hat{r}(x) \coloneqq (-1)^{\binom{m}{2}} \sum_{i=1}^{m} f(\beta_i) \prod_{j \neq i} (x - \beta_j) \cdot g'(\beta_j).$$

As the analysis above shows, when $\operatorname{disc}(g) \neq 0$, we have the polynomial identity $\hat{r}(x) = \operatorname{disc}(g)r(x)$. It remains to show that the identity $\hat{r}(x) = \operatorname{disc}(g)r(x)$ still holds when $\operatorname{disc}(g) = 0$.

In the case $\operatorname{disc}(g) = 0$, the polynomial g has a double root. Without loss of generality, suppose that $\beta_1 = \beta_2$. By definition of a double root, we have $g'(\beta_1) = g'(\beta_2) = 0$. This implies that the product

$$\prod_{i=1}^{m}(f(\beta_i) + y(x - \beta_i)g'(\beta_i))$$

has degree at most $m-2$ in y, so the coefficient of y^{m-1} is zero, hence $\hat{r}(y) = 0$. Because $\operatorname{disc}(g) = 0$, we again have the desired equality $\hat{r}(y) = 0 = \operatorname{disc}(g)r(y)$. □

As a corollary, we obtain an AC^0 algorithm for polynomial division with remainder.

Corollary 6.6. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $q, r \in \mathbb{F}[x]$ be the unique polynomials that satisfy $f = qg + r$ and $\deg(r) < \deg(g)$. Then the coefficients of q and r can be computed piecewise in AC^0.*

Proof. Applying Lemma 6.5 to f and g, we compute in piecewise AC^0 a polynomial $\hat{r}$ of degree $\deg(\hat{r}) < \deg(g)$ such that $\hat{r} \equiv f \pmod{g}$. In particular, $f - \hat{r}$ is a multiple of g, so there is a polynomial $\hat{q} \in \mathbb{F}[x]$ such that $f - \hat{r} = \hat{q}g$. We can compute $\hat{q}$ in AC^0 using Lemma 4.1. Thus, we have $f = \hat{q}g + \hat{r}$ and $\deg(\hat{r}) < \deg(g)$. Uniqueness of polynomial division with remainder implies that $\hat{q} = q$ and $\hat{r} = r$ are the quotient and remainder, respectively, of f divided by g. □

6.3 Inverting the Sylvester Matrix

This subsection describes an AC^0 circuit that computes the adjugate $\operatorname{adj}\operatorname{Syl}(f, g)$ of the Sylvester matrix. Using the fact that $A^{-1} = \frac{1}{\det A}\operatorname{adj} A$ for any matrix A and that $\operatorname{res}(f, g) = \det\operatorname{Syl}(f, g)$ can be computed in AC^0, this yields an AC^0 algorithm to compute the inverse $\operatorname{Syl}(f, g)^{-1}$ of the Sylvester matrix when $\operatorname{Syl}(f, g)$ is invertible. As a corollary of this and Lemma 2.16, we obtain an AC^0 algorithm that computes the Bézout coefficients of f and g when $\gcd(f, g) = 1$. The algorithm and its proof of correctness are very similar to Lemma 6.5. The primary difference is that instead of interpolating f over the roots of g, we need to interpolate $1/f$ over the roots of g.

Theorem 6.7. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then the entries of* $\operatorname{adj}\operatorname{Syl}(f, g)$ *can be computed in AC^0.*

Proof. Let $n \coloneqq \deg(f)$ and $m \coloneqq \deg(g)$. Recall Lemma 2.16: for $\ell \in [n + m]$, the entries in the ℓ^{th} column of $\operatorname{adj}\operatorname{Syl}(f, g)$ correspond to the coefficients of polynomials $a_\ell, b_\ell \in \mathbb{F}[x]$ such that

$$a_\ell(x)f(x) + b_\ell(x)g(x) = \operatorname{res}(f, g)x^{n+m-\ell},$$

where $\deg(a_\ell) < m$ and $\deg(b_\ell) < n$. Below, we will show that the coefficients of $\operatorname{disc}(g)a_\ell(x)$ and $\operatorname{disc}(f)b_\ell(x)$ can be computed in AC^0. We can also compute $\operatorname{disc}(f)$ and $\operatorname{disc}(g)$ in AC^0 using Corollary 6.2. This yields an AC^0 algorithm with division that computes the coefficients of a_ℓ and b_ℓ. Because the coefficients of a_ℓ and b_ℓ are the entries of the ℓ^{th} column of $\operatorname{adj}\operatorname{Syl}(f, g)$, the coefficients of a_ℓ and b_ℓ are polynomial functions in the coefficients of f and g. This allows us to apply Theorem 2.12, which yields the claimed AC^0 algorithm to compute the entries of $\operatorname{adj}\operatorname{Syl}(f, g)$.

It remains to show that for every ℓ, we can compute the coefficients of the polynomials $\mathrm{disc}(g)a_\ell(x)$ and $\mathrm{disc}(f)b_\ell(x)$. We start by computing $\mathrm{disc}(g)a_\ell(x)$. By exchanging the roles of f and g in the algorithm below, we can likewise compute $\mathrm{disc}(f)b_\ell(x)$.

Suppose $\mathrm{disc}(g) \neq 0$, so that g has m simple roots $\beta_1, \ldots, \beta_m$. In this case, we can explicitly write $a_\ell(x)$ using Lagrange interpolation as

$$a_\ell(x) = \mathrm{res}(f, g) \sum_{i=1}^{m} \frac{\beta_i^{n+m-\ell}}{f(\beta_i)} \prod_{j \neq i} \frac{x - \beta_j}{\beta_i - \beta_j}.$$

Using Lemma 2.17 and Lemma 2.20 to expand $\mathrm{res}(f, g)$ and $\mathrm{disc}(g)$, we have

$$\begin{aligned}
\mathrm{disc}(g)a_\ell(x) &= \left((-1)^{\binom{m}{2}} \prod_{i \in [m]} \prod_{j \neq i} (\beta_i - \beta_j)\right)\left((-1)^{nm} \prod_{i=1}^{m} f(\beta_i)\right) \sum_{i=1}^{m} \frac{\beta_i^{n+m-\ell}}{f(\beta_i)} \prod_{j \neq i} \frac{x - \beta_j}{\beta_i - \beta_j} \\
&= (-1)^{nm+\binom{m}{2}} \sum_{i=1}^{m} \beta_i^{n+m-\ell} \prod_{j \neq i} f(\beta_j)(x - \beta_j) \prod_{k \neq j} (\beta_j - \beta_k) \\
&= (-1)^{nm+\binom{m}{2}} \sum_{i=1}^{m} \beta_i^{n+m-\ell} \prod_{j \neq i} f(\beta_j) g'(\beta_j)(x - \beta_j).
\end{aligned}$$

Let y be a new variable. Observe that the summation above is the coefficient of y^{m-1} when the product

$$\prod_{i=1}^{m} (\beta_i^{n+m-\ell} + y f(\beta_i) g'(\beta_i)(x - \beta_i))$$

is expanded as a polynomial in y with coefficients in x. We will compute this product by an application of Lemma 5.2.

Let z be a fresh variable. Define $h(x, y, z) \in \mathbb{F}[x, y, z]$ by

$$h(x, y, z) \coloneqq z^{n+m-\ell} + y(x - z) f(z) g'(z).$$

It is clear from the definition that h can be computed in AC^0. Using Lemma 5.2, we compute

$$\prod_{i=1}^{m} h(x, y, \beta_i) = \prod_{i=1}^{m} (\beta_i^{n+m-\ell} + y(x - \beta_i) f(\beta_i) g'(\beta_i))$$

in AC^0, where $\beta_1, \ldots, \beta_m \in \mathbb{F}$ are the (not necessarily distinct) roots of g. Interpolating the coefficient of y^{m-1} using Lemma 2.7 and multiplying by $(-1)^{nm+\binom{m}{2}}$, we have an AC^0 algorithm to compute

$$\hat{a}(x) \coloneqq (-1)^{nm+\binom{m}{2}} \sum_{i=1}^{m} \beta_i^{n+m-\ell} \prod_{j \neq i} f(\beta_j) g'(\beta_j)(x - \beta_j).$$

As the analysis above shows, when $\mathrm{disc}(g) \neq 0$, we have the equality $\hat{a}(x) = \mathrm{disc}(g)a_\ell(x)$. It remains to show that this equality holds when $\mathrm{disc}(g) = 0$.

If $\mathrm{disc}(g) = 0$, then g has a double root. Without loss of generality, suppose that $\beta_1 = \beta_2$. By definition of a double root, we have $g'(\beta_1) = g'(\beta_2)$. This implies that the product

$$\prod_{i=1}^{m} (\beta_i^{n+m-\ell} + y(x - \beta_i) f(\beta_i) g'(\beta_i))$$

has degree at most $m - 2$ in y, so $\hat{a}(x) = 0 = \mathrm{disc}(g)a_\ell(x)$ as desired. □

Recall that for coprime polynomials $f, g \in \mathbb{F}[x]$, the coefficients of the Bézout coefficients $a, b \in \mathbb{F}[x]$ appear in the last column of $\mathrm{Syl}(f,g)^{-1}$. Using the fact that $\mathrm{Syl}(f,g)^{-1} = \frac{1}{\mathrm{res}(f,g)} \operatorname{adj} \mathrm{Syl}(f,g)$ and that both $\mathrm{res}(f,g)$ and $\operatorname{adj} \mathrm{Syl}(f,g)$ can be computed in AC^0, we see that $\mathrm{Syl}(f,g)^{-1}$ can be computed in AC^0, provided that $\mathrm{Syl}(f,g)^{-1}$ exists. This yields an AC^0 algorithm to compute the Bézout coefficients of two coprime polynomials, which we record in the following corollary. We will remove the requirement that f and g are coprime later in Section 8.

Corollary 6.8. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that $\gcd(f,g) = 1$. Then the Bézout coefficients of f and g can be computed in AC^0.*

As a second corollary of Theorem 6.7, we obtain an AC^0 algorithm to invert triangular Toeplitz matrices. This follows from the simple fact that a triangular Toeplitz matrix can be embedded as a block of the Sylvester matrix for a particular pair of polynomials. A different AC^0 algorithm for this problem was described by Bini [Bin84].

Corollary 6.9. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than n. Let $A \in \mathbb{F}^{n \times n}$ be a triangular Toeplitz matrix. Then the inverse A^{-1} can be computed in AC^0.*

Proof. Write

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ 0 & a_0 & a_1 & \cdots & a_{n-2} \\ 0 & 0 & a_0 & \cdots & a_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_0 \end{pmatrix}.$$

Let $f(x) \coloneqq \sum_{i=0}^{n-1} a_i x^i$ and let $g(x) \coloneqq x^{n-1}$. Observe that the Sylvester matrix $\mathrm{Syl}(f,g)$ is a $2n \times 2n$ matrix that decomposes into $n \times n$ blocks as

$$\mathrm{Syl}(f,g) = \begin{pmatrix} B & I_n \\ A & 0 \end{pmatrix},$$

where B is an $n \times n$ matrix corresponding to the coefficients of f and I_n is the $n \times n$ identity matrix. The inverse of $\mathrm{Syl}(f,g)$ has the block decomposition

$$\mathrm{Syl}(f,g)^{-1} = \begin{pmatrix} 0 & A^{-1} \\ I_n & -BA^{-1} \end{pmatrix}.$$

In particular, the upper-right $n \times n$ block of $\mathrm{Syl}(f,g)^{-1}$ corresponds to A^{-1}. Thus, we can compute A^{-1} in AC^0 by inverting $\mathrm{Syl}(f,g)$ using Theorem 6.7. □

6.4 Inverting the Bézout Matrix

In this subsection, we design an algorithm to compute the inverse of the Bézout matrix $\mathrm{Bez}_n(f,g)$ of two polynomials. This is an easy corollary of the fact that we can compute the Bézout coefficients of coprime polynomials in AC^0.

Theorem 6.10. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $\delta = \max(\deg(f), \deg(g))$. Suppose that $\gcd(f,g) = 1$, so that the matrix $\mathrm{Bez}_\delta(f,g)$ is invertible. Then the inverse $\mathrm{Bez}_\delta(f,g)^{-1}$ can be computed in AC^0.*

Proof. Without loss of generality, suppose that $\delta = \deg(f)$. By Proposition 2.23, to invert $\mathrm{Bez}_\delta(f, g)$, it suffices to compute a polynomial $p \in \mathbb{F}[x]$ of degree at most $n-1$ such that $p(x)g(x) \equiv 1 \pmod{f(x)}$. The desired polynomial p is precisely the Bézout coefficient of g, which we can compute in AC^0 using Corollary 6.8. □

7 Operations on Roots

In this section, we develop tools to manipulate the factors of univariate polynomials without having explicit access to the factorization of a polynomial. These results are the technical highlight of this work and will be essential for our AC^0 algorithm for the GCD.

7.1 Filtering

All of the algorithms we have seen so far proceed by computing symmetric functions of the roots of two univariate polynomials f and g. To compute the GCD, we also need a means of comparing the (multisets of) roots of polynomials. The algorithm of Theorem 6.1 for the resultant tells us if f and g share a common root, but the resultant itself does not tell us any more about how the roots of f and g compare. We can learn more by inspecting other elementary symmetric functions of the evaluations of g at the roots of f (and vice-versa). Through a careful application of Lemma 5.3, we can filter the common roots of f and g out of f (and likewise, out of g), as we now illustrate.

Let

$$f(x) \coloneqq \prod_{i=1}^{n}(x-\alpha_i)\prod_{i=1}^{d}(x-\gamma_i)$$
$$g(x) \coloneqq \prod_{i=1}^{m}(x-\beta_i)\prod_{i=1}^{d}(x-\gamma_i)$$

be two squarefree polynomials, where $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \gamma_1, \ldots, \gamma_d$ are $n+m+d$ distinct field elements. Although the values of n, m, and d are known to us for the purpose of analysis, these values are unknown to the algorithm. Our goal is to compute $\prod_{i=1}^{d}(x-\gamma_i)$, which allows us to distinguish the shared roots of f and g from the set of all roots of f. Note that this will allow us to compute the GCD when f and g are squarefree, which represents serious progress towards a general algorithm for the GCD!

Let y be a fresh variable and consider the polynomial

$$h(x, y) \coloneqq (y-x)\, g(x),$$

viewed as an element of $\mathbb{F}(y)[x]$. The evaluations of $h(x, y)$ at the roots of $f(x)$ are given by

$$\begin{aligned}
\lambda_1 &\coloneqq h(\alpha_1, y) = (y-\alpha_1)\, g(\alpha_1) \neq 0\\
&\vdots\\
\lambda_n &\coloneqq h(\alpha_n, y) = (y-\alpha_n)\, g(\alpha_n) \neq 0\\
\lambda_{n+1} &\coloneqq h(\gamma_1, y) = (y-\gamma_1)\, g(\gamma_1) = 0\\
&\vdots\\
\lambda_{n+d} &\coloneqq h(\gamma_d, y) = (y-\gamma_d)\, g(\gamma_d) = 0.
\end{aligned}$$

By taking h to be a multiple of g, we ensure that $h(x, y)$ vanishes whenever we substitute a root of g for x. Applying Lemma 5.3, we can compute

$$\prod_{i=1}^{n} \lambda_i = \prod_{i=1}^{n} (y - \alpha_i) g(\alpha_i)$$

piecewise in AC^0. Normalizing this polynomial to have leading coefficient 1 yields $\prod_{i=1}^{n}(y - \alpha_i)$, which is precisely the product of factors of f that are not shared with g. From here, we can obtain $\prod_{i=1}^{d}(y - \gamma_i)$ as the quotient

$$\prod_{i=1}^{d} (y - \gamma_i) = \frac{f(y)}{\prod_{i=1}^{n}(y - \alpha_i)},$$

which can be computed in AC^0 using Lemma 4.1.

When the polynomials f and g are not squarefree, the algorithm sketched above separates the factors of f into two sets: those that correspond to roots of g, and those that are not roots of g. The factors of f maintain their multiplicity in the output of this algorithm. We can use this to compute a polynomial h such that h and $\gcd(f, g)$ have the same set of irreducible factors, but possibly with different multiplicities. Later in Section 8, we will see how to compute the GCD with the correct multiplicities.

We now formalize the result of the preceding sketch.

Lemma 7.1. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that f factors as $\prod_{i=1}^{n}(x - \alpha_i)^{a_i}$. Then the coefficients of the polynomials*

$$f_{g=0}(x) \coloneqq \prod_{i : g(\alpha_i) = 0} (x - \alpha_i)^{a_i}$$
$$f_{g\neq 0}(x) \coloneqq \prod_{i : g(\alpha_i) \neq 0} (x - \alpha_i)^{a_i}$$

can be computed piecewise in AC^0.

Proof. Let y be a fresh variable. Define $h(x, y) \in \mathbb{F}[x, y]$ as

$$h(x, y) \coloneqq (y - x)\, g(x).$$

For a root α_i of f, the polynomial $y - \alpha_i$ is clearly nonzero, so $h(\alpha_i, y) = 0$ if and only if $g(\alpha_i) = 0$. By Lemma 5.3, we can compute the polynomial

$$r(y) \coloneqq \prod_{i : g(\alpha_i) \neq 0} h(\alpha_i, y) = \prod_{i : g(\alpha_i) \neq 0} (y - \alpha_i)^{a_i} g(\alpha_i)^{a_i}$$

piecewise in AC^0. The leading coefficient of $r(y)$ is given by $\prod_{i : g(\alpha_i) \neq 0} g(\alpha_i)^{a_i}$. By interpolating the coefficients of $r(y)$ and normalizing $r(y)$ to have leading coefficient 1, we obtain

$$\prod_{i : g(\alpha_i) \neq 0} (y - \alpha_i)^{a_i} = f_{g\neq 0}(y).$$

We then compute $f_{g=0}(x)$ in AC^0 using the identity $f_{g=0}(x) = \frac{f(x)}{f_{g\neq 0}(x)}$ and Lemma 4.1. □

The preceding lemma easily generalizes to the case where we have multiple polynomials $g_1, \ldots, g_m \in \mathbb{F}[x]$ and want to extract from f all factors that correspond to common roots of $g_1, \ldots, g_m$. As we saw in the proof of Lemma 7.1, multiplying $(y-x)$ by $g(x)$ suppressed the common roots of f and g. To handle multiple polynomials $g_1, \ldots, g_m$, we need to find a polynomial in x whose roots are exactly the common roots of the g_i. If we add a fresh variable z, then this is easy to do with the polynomial

$$g(x, z) \coloneqq g_1(x) + z g_2(x) + \cdots + z^{m-1} g_m(x).$$

We now extend Lemma 7.1 to filter out the common roots of multiple polynomials from a given polynomial f. The proof is a straightforward generalization of the proof of Lemma 7.1 using $g(x, z)$ in place of $g(x)$.

Lemma 7.2. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g_1, \ldots, g_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let*

$$V \coloneqq \{\alpha \in \mathbb{F} : g_1(\alpha) = \cdots = g_m(\alpha) = 0\}$$

be the set of common roots of $g_1, \ldots, g_m$. Suppose that f factors as $\prod_{i=1}^{n} (x - \alpha_i)^{a_i}$. Then the coefficients of the polynomials

$$f_{\in V}(x) \coloneqq \prod_{i : \alpha_i \in V} (x - \alpha_i)^{a_i}$$
$$f_{\notin V}(x) \coloneqq \prod_{i : \alpha_i \notin V} (x - \alpha_i)^{a_i}$$

can be computed piecewise in AC^0.

Proof. Let y and z be fresh variables. Define $g(x, z) \in \mathbb{F}[x, z]$ as

$$g(x, z) \coloneqq \sum_{i=1}^{m} z^{i-1} g_i(x)$$

and define $h(x, y, z) \in \mathbb{F}[x, y, z]$ by

$$h(x, y, z) \coloneqq (y - x) g(x, z).$$

For a root α_i of f, the polynomial $y - \alpha_i$ is nonzero, so $h(\alpha_i, y, z) = 0$ if and only if $g(\alpha_i, z) = 0$, which occurs exactly when $g_1(\alpha_i) = \cdots = g_m(\alpha_i) = 0$. By Lemma 5.3, we can compute the polynomial

$$r(y, z) \coloneqq \prod_{i : \alpha_i \notin V} h(\alpha_i, y, z) = \prod_{i : \alpha_i \notin V} (y - \alpha_i)^{a_i} g(\alpha_i, z)^{a_i}$$

piecewise in AC^0.

For each i such that $\alpha_i \notin V$, let

$$j_i \coloneqq \max\{j \in [m] : g_j(\alpha_i) \neq 0\}.$$

View $r(y, z)$ as a polynomial in z with coefficients in $\mathbb{F}[y]$. The leading coefficient of the term

$$(y - \alpha_i)^{a_i} g(\alpha_i, z)^{a_i} = (y - \alpha_i)^{a_i} (g_1(\alpha_i) + z g_2(\alpha_i) + \cdots + z^{m-1} g_m(\alpha_i))^{a_i}$$

is

$$(y - \alpha_i)^{a_i} g_{j_i}(\alpha_i)^{a_i}.$$

Leading coefficients are homomorphic with respect to multiplication, so the leading coefficient of $r(y, z)$ (as a polynomial in $\mathbb{F}[y][z]$) is given by

$$s(y) \coloneqq \prod_{i:\alpha_i \notin V} (y - \alpha_i)^{a_i} g_{j_i}(\alpha_i)^{a_i}.$$

Because $\deg(r(y, z)) \leqslant dm$, the coefficients of $r(y, z)$ can be computed in AC^0 via Lemma 2.7. Using these coefficients as test circuits (ordered from high to low degree), we can branch and continue the computation piecewise using the leading coefficient $s(y)$ of $r(y, z)$.

As a polynomial in y, the leading coefficient of $s(y)$ itself is $\prod_{i:\alpha_i \notin V} g_{j_i}(\alpha_i)^{a_i}$. By interpolating the coefficients of $s(y)$ using Lemma 2.7, we can normalize $s(y)$ to have leading coefficient 1. This normalization incurs another branching operation using the coefficients of $s(y)$ as the test circuits, as we do not know the precise degree of $s(y)$. This yields an AC^0 circuit that computes the polynomial $\prod_{i:\alpha_i \notin V}(y - \alpha_i)^{a_i} = f_{\notin V}(y)$. We can then use the identity $f_{\in V}(x) = \frac{f(x)}{f_{\notin V}(x)}$ and Lemma 4.1 to compute $f_{\in V}$ in AC^0. □

7.2 Thresholding

In the previous subsection, we saw how to distinguish the common roots of two polynomials from the roots of a single polynomial. To compute the GCD, we also need to distinguish between roots of a polynomial of different multiplicities. As we shall see, this can be done as a straightforward application of Lemma 7.2.

Let $f, g \in \mathbb{F}[x]$ be univariate polynomials and suppose that f and g factor as

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}$$

$$g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{b_i}.$$

Here, we have switched notation from previous sections, and now take $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ to be the union of the roots of f and g, as this will be convenient for us here. Because the α_i are the union of the roots of f and g, some of the a_i or b_i may be zero.

Recall that a point $\alpha \in \mathbb{F}$ is a root of f of multiplicity r if and only if f and its first $r - 1$ derivatives vanish at α, i.e.,

$$f(\alpha) = f'(\alpha) = \cdots = f^{(r-1)}(\alpha) = 0.$$

This suggests that we can distinguish between factors of f of different multiplicities by invoking Lemma 7.2 with $f, f', \ldots, f^{(r-1)}$ as the filter polynomials. More generally, we can filter the roots of f by their multiplicity in g.

For a parameter $r \in \mathbb{N}$, let $f_{g<r}$ be the polynomial defined as

$$f_{g<r}(x) \coloneqq \prod_{i:b_i<r} (x - \alpha_i)^{a_i}.$$

That is, $f_{g<r}$ selects from f the factors $(x - \alpha_i)$ that occur with multiplicity less than r in g, but preserves the multiplicity a_i with which $(x - \alpha_i)$ occurs in f. The next lemma describes an AC^0 algorithm that piecewise computes the coefficients of $f_{g<r}$ given the coefficients of f and g and the value of r.

Lemma 7.3. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that f and g factor as $f(x) = \prod_{i=1}^{n}(x-\alpha_i)^{a_i}$ and $g(x) = \prod_{i=1}^{n}(x-\alpha_i)^{b_i}$, where $\alpha_i \in \mathbb{F}$ and $a_i, b_i \in \mathbb{N}$. Let $r \in \mathbb{N}$. Then the coefficients of the polynomials*

$$f_{g \geqslant r}(x) \coloneqq \prod_{i : b_i \geqslant r} (x-\alpha_i)^{a_i}$$
$$f_{g < r}(x) \coloneqq \prod_{i : b_i < r} (x-\alpha_i)^{a_i}$$

can be computed piecewise in AC^0.

Proof. Note that if $r > \deg(g)$, then $f_{g \geqslant r} = 1$ and $f_{g<r} = f$, so we may assume without loss of generality that $r \leqslant \deg(g) < \operatorname{char}(\mathbb{F})$. Define $V_{\geqslant r} \subseteq \mathbb{F}$ to be the roots of g of multiplicity at least r. Because $r < \operatorname{char}(\mathbb{F})$, we can express $V_{\geqslant r}$ as

$$\begin{aligned} V_{\geqslant r} &\coloneqq \{\alpha \in \mathbb{F} : (x-\alpha)^r \mid g(x)\} \\ &= \{\alpha \in \mathbb{F} : g(\alpha) = g^{(1)}(\alpha) = \cdots = g^{(r-1)}(\alpha) = 0\}. \end{aligned}$$

Note that $f_{g \geqslant r}(x)$ and $f_{g<r}(x)$ satisfy

$$f_{g \geqslant r}(x) = \prod_{i : \alpha_i \in V_{\geqslant r}} (x-\alpha_i)^{a_i}$$
$$f_{g < r}(x) = \prod_{i : \alpha_i \notin V_{\geqslant r}} (x-\alpha_i)^{a_i}.$$

We can compute the coefficients of $g^{(1)}(x), \ldots, g^{(r-1)}(x)$ in AC^0. A direct application of Lemma 7.2 computes the coefficients of $f_{g \geqslant r}(x)$ and $f_{g<r}(x)$ piecewise in AC^0. □

Suppose we are instead given multiple polynomials $f, g_1, \ldots, g_m \in \mathbb{F}[x]$ that factor as

$$\begin{aligned} f(x) &= \prod_{i=1}^{n}(x-\alpha_i)^{a_i} \\ g_1(x) &= \prod_{i=1}^{n}(x-\alpha_i)^{b_{1,i}} \\ &\vdots \\ g_m(x) &= \prod_{i=1}^{n}(x-\alpha_i)^{b_{m,i}}. \end{aligned}$$

Lemma 7.3 allows us to extract factors from f based on their multiplicity in one of the g_i. Just as Lemma 7.2 generalized Lemma 7.1 to allow for multiple filter polynomials, we can generalize Lemma 7.3 to filter the roots of f according to their multiplicities in multiple polynomials, as the following lemma shows.

Lemma 7.4. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f, g_1, \ldots, g_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that f and g_j*

factor as $f(x) = \prod_{i=1}^{n}(x-\alpha_i)^{a_i}$ *and* $g_j(x) = \prod_{i=1}^{n}(x-\alpha_i)^{b_{j,i}}$*, where* $\alpha_i \in \mathbb{F}$ *and* $a_i, b_{j,i} \in \mathbb{N}$*. Let* $r_1, \ldots, r_m \in \mathbb{N}$*. Then the coefficients of the polynomials*

$$f_{\overline{g}\geqslant\overline{r}}(x) \coloneqq \prod_{i:b_{1,i}\geqslant r_1 \wedge \cdots \wedge b_{m,i}\geqslant r_m} (x-\alpha_i)^{a_i}$$
$$f_{\overline{g}\not\geqslant\overline{r}}(x) \coloneqq \prod_{i:b_{1,i}< r_1 \vee \cdots \vee b_{m,i}< r_m} (x-\alpha_i)^{a_i}$$

can be computed piecewise in AC^0*.*

Proof. Note that if $r_i > \deg(g_i)$ for some $i \in [m]$, then $f_{\overline{g}\geqslant\overline{r}} = 1$ and $f_{\overline{g}\not\geqslant\overline{r}} = f$, so we may assume without loss of generality that $r_i \leqslant \deg(g_i) < \operatorname{char}(\mathbb{F})$ for all $i \in [m]$.

Define $V_{\geqslant\overline{r}} \subseteq \mathbb{F}$ to be the set of $\alpha \in \mathbb{F}$ that occur with multiplicity at least r_i in g_i for all $i \in [m]$. Because $r_i < \operatorname{char}(\mathbb{F})$ for all $i \in [m]$, we can write $V_{\geqslant\overline{r}}$ as

$$\begin{aligned} V_{\geqslant\overline{r}} &\coloneqq \bigcap_{i=1}^{m} \{\alpha \in \mathbb{F} : (x-\alpha)^{r_i} \mid g_i(x)\} \\ &= \bigcap_{i=1}^{m} \{\alpha \in \mathbb{F} : g_i(\alpha) = g_i^{(1)}(\alpha) = \cdots = g_i^{(r_i-1)}(\alpha) = 0\} \\ &= \{\alpha \in \mathbb{F} : g_i^{(j)}(\alpha) = 0 \ \forall i \in [m], j \in \{0,1,\ldots,r_i-1\}\}. \end{aligned}$$

Note that $f_{\overline{g}\geqslant\overline{r}}(x)$ and $f_{\overline{g}\not\geqslant\overline{r}}(x)$ satisfy

$$f_{\overline{g}\geqslant\overline{r}}(x) = \prod_{i:\alpha_i \in V_{\geqslant\overline{r}}} (x-\alpha_i)^{a_i}$$
$$f_{\overline{g}\not\geqslant\overline{r}}(x) = \prod_{i:\alpha_i \notin V_{\geqslant\overline{r}}} (x-\alpha_i)^{a_i}.$$

We can compute the coefficients of $g_i^{(j)}(x)$ for all $i \in [m]$ and $j \in \{0,1,\ldots,r_i-1\}$ in AC^0. A direct application of Lemma 7.2 computes the coefficients of $f_{\overline{g}\geqslant\overline{r}}(x)$ and $f_{\overline{g}\not\geqslant\overline{r}}(x)$ piecewise in AC^0. □

Remark 7.5. Although Lemma 7.4 holds for an arbitrary threshold vector $\overline{r} \in \mathbb{N}^m$, our applications only make use of the case where all entries of $\overline{r}$ are equal. ◊

7.3 Squarefree Decomposition

We conclude this section with an application of Lemma 7.3 to computing the squarefree decomposition of a univariate polynomial $f \in \mathbb{F}[x]$. Typically, the squarefree decomposition is computed by reduction to the GCD. If f factors as $f = \prod_{i=1}^{n}(x-\alpha_i)^{a_i}$, then it is easy to show that

$$\gcd(f, f', \ldots, f^{(r-1)}) = \prod_{i=1}^{n}(x-\alpha_i)^{\max(a_i-r,0)}.$$

Denoting the squarefree decomposition of f by $(f_1, \ldots, f_m)$, one can write f_r as

$$f_r = \frac{\gcd(f, f', \ldots, f^{(r-1)})}{\gcd(f, f', \ldots, f^{(r)})}.$$

This reduction, combined with an NC^2 algorithm for the GCD of many polynomials, leads to the NC^2 algorithm of von zur Gathen [vzGat84] for the squarefree decomposition. In his survey, von zur Gathen [vzGat86] asked if there is an NC^1 algorithm to compute the squarefree decomposition. As an application of Lemma 7.3, we will show that the squarefree decomposition can in fact be computed piecewise in AC^0.

We note that while the squarefree decomposition is usually obtained by reducing to the GCD, this is not the route we take. Our techniques seem to naturally proceed in the reverse direction, instead computing the GCD by reducing to the squarefree decomposition.

Lemma 7.6. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f \in \mathbb{F}[x]$ be a univariate polynomial of degree d given by its coefficients. Then the squarefree decomposition of f can be computed piecewise in AC^0.*

Proof. Suppose that f factors as

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}.$$

Let $d \coloneqq \deg(f)$. For all $r \in [d]$, we can compute

$$f_{\leqslant r}(x) \coloneqq \prod_{i : a_i \leqslant r} (x - \alpha_i)^{a_i}$$

in AC^0 piecewise using Lemma 7.3. We then compute, for all $r \in [d]$, the polynomials

$$f_{=r}(x) \coloneqq \prod_{i : a_i = r} (x - \alpha_i)^{a_i} = \frac{f_{\leqslant r}(x)}{f_{\leqslant r-1}(x)}$$

in AC^0 using Lemma 4.1. Observe that if $(f_1, \ldots, f_d)$ is the squarefree decomposition of f, then the f_i satisfy $f_r(x)^r = f_{=r}(x)$. We can obtain $f_r(x)$ in AC^0 by applying Lemma 4.2 to $f_{=r}(x)$. □

As a corollary of Lemma 7.6, we can also compute the squarefree part of a univariate polynomial $f \in \mathbb{F}[x]$ piecewise in AC^0.

Corollary 7.7. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $f \in \mathbb{F}[x]$ be a univariate polynomial of degree d given by its coefficients. Then the squarefree part of f can be computed piecewise in AC^0.*

Proof. By Lemma 7.6, the squarefree decomposition $(f_1, \ldots, f_m)$ of f can be computed piecewise in AC^0. The squarefree part of f is given by $\prod_{i=1}^{m} f_i$. We can compute the coefficients of $\prod_{i=1}^{m} f_i$ in AC^0 by Lemma 2.7. □

8 Greatest Common Divisor and Least Common Multiple

In this section, we describe piecewise AC^0 algorithms to compute the greatest common divisor and least common multiple of m univariate polynomials $f_1, \ldots, f_m \in \mathbb{F}[x]$. We also describe a piecewise AC^0 algorithm to compute the Bézout coefficients of a pair of polynomials $f_1, f_2 \in \mathbb{F}[x]$. Our algorithms for the greatest common divisor and least common multiple turn out to be straightforward applications of Lemma 7.4 and Corollary 7.7. To compute the Bézout coefficients, we reduce to the case where f_1 and f_2 are coprime by dividing out $\gcd(f_1, f_2)$ using Lemma 4.1, at which point we can apply Corollary 6.8.

8.1 Two Polynomials

In this subsection, we compute the greatest common divisor, least common multiple, and Bézout coefficients of a pair of polynomials, all in (piecewise) AC^0. We start with the greatest common divisor.

Theorem 8.1. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $2d$. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then their greatest common divisor $\gcd(f, g)$ can be computed piecewise in AC^0.*

Proof. Suppose that f and g factor as

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}$$
$$g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{b_i},$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_i, b_i \in \mathbb{N}$. Note that some of the a_i and b_i may be zero. Let $s(x)$ be the squarefree part of $f(x)g(x)$, i.e.,

$$s(x) \coloneqq \prod_{i=1}^{n} (x - \alpha_i).$$

We may compute $s(x)$ piecewise in AC^0 using Corollary 7.7.

For a parameter $r \in \mathbb{N}$, let

$$s_{\geqslant r}(x) \coloneqq \prod_{i : a_i \geqslant r \wedge b_i \geqslant r} (x - \alpha_i).$$

We can compute $s_{\geqslant r}(x)$ piecewise in AC^0 for all $1 \leqslant r \leqslant \min(\deg(f), \deg(g))$ by applying Lemma 7.4 to $s(x)$, $f(x)$, and $g(x)$. Observe that

$$\begin{aligned}
\gcd(f(x), g(x)) &= \prod_{i=1}^{n} (x - \alpha_i)^{\min(a_i, b_i)} \\
&= \prod_{r=1}^{\min(\deg(f), \deg(g))} \prod_{i : \min(a_i, b_i) \geqslant r} (x - \alpha_i) \\
&= \prod_{r=1}^{\min(\deg(f), \deg(g))} s_{\geqslant r}(x),
\end{aligned}$$

so we can compute $\gcd(f, g)$ piecewise in AC^0. We can then interpolate the coefficients of $\gcd(f, g)$ in AC^0 using Lemma 2.7. □

As an easy corollary of Theorem 8.1, we obtain an AC^0 algorithm to compute the least common multiple of two polynomials.

Corollary 8.2. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $2d$. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then $\operatorname{lcm}(f, g)$ can be computed in piecewise AC^0.*

Proof. We can compute $\gcd(f(x), g(x))$ piecewise in AC^0 using Theorem 8.1. The coefficients of the product $f(x)g(x)$ can be computed in AC^0 using Lemma 2.7. Using the identity

$$\operatorname{lcm}(f(x), g(x)) = \frac{f(x)g(x)}{\gcd(f(x), g(x))},$$

we can compute $\operatorname{lcm}(f(x), g(x))$ piecewise in AC^0 using Lemma 4.1. □

By combining Theorem 8.1 with Corollary 6.8, we obtain an AC^0 algorithm to compute the Bézout coefficients of any pair of polynomials.

Corollary 8.3. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $2d$. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then the Bézout coefficients of f and g can be computed piecewise in AC^0.*

Proof. Using Theorem 8.1 and Lemma 4.1, we can compute $\hat{f}(x) \coloneqq \frac{f(x)}{\gcd(f(x),g(x))}$ and $\hat{g}(x) \coloneqq \frac{g(x)}{\gcd(f(x),g(x))}$ piecewise in AC^0. By construction, we have $\gcd(\hat{f}(x), \hat{g}(x)) = 1$. Observe that the Bézout coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ are equal to those of $f(x)$ and $g(x)$. We may compute the Bézout coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ in AC^0 using Corollary 6.8. □

8.2 Multiple Polynomials

We now extend the results of the previous subsection to compute greatest common divisors and least common multiples of many polynomials. The proof of Theorem 8.1 easily generalizes to compute the GCD of many polynomials.

Theorem 8.4. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than md. Let $f_1, \ldots, f_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then their greatest common divisor $\gcd(f_1, \ldots, f_m)$ can be computed piecewise in AC^0.*

Proof. Suppose that $f_1, \ldots, f_m$ factor as

$$\begin{aligned} f_1(x) &= \prod_{i=1}^{n} (x - \alpha_i)^{a_{1,i}} \\ &\vdots \\ f_m(x) &= \prod_{i=1}^{n} (x - \alpha_i)^{a_{m,i}}, \end{aligned}$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_{j,i} \in \mathbb{N}$. Note that some of the $a_{j,i}$ may be zero. Let $s(x)$ be the squarefree part of $f_1(x) \cdots f_m(x)$, i.e.,

$$s(x) \coloneqq \prod_{i=1}^{n} (x - \alpha_i).$$

We may compute $s(x)$ piecewise in AC^0 using Corollary 7.7.

For a parameter $r \in \mathbb{N}$, let

$$s_{\geqslant r}(x) \coloneqq \prod_{i : a_{1,i} \geqslant r \wedge \cdots \wedge a_{m,i} \geqslant r} (x - \alpha_i).$$

Let $\delta \coloneqq \min_{i\in[m]} \deg(f_i)$. We can compute the coefficients of $s_{\geqslant r}(x)$ in AC^0 for all $1 \leqslant r \leqslant \delta$ by applying Lemma 7.4 to $s(x)$ and $f_1(x), \ldots, f_m(x)$.

Observe that

$$\begin{aligned}\gcd(f_1(x), \ldots, f_m(x)) &= \prod_{i=1}^{n} (x-\alpha_i)^{\min(a_{1,i},\ldots,a_{m,i})} \\ &= \prod_{r=1}^{\delta} \prod_{i:\min(a_{1,i},\ldots,a_{m,i})\geqslant r} (x-\alpha_i) \\ &= \prod_{r=1}^{\delta} s_{\geqslant r}(x),\end{aligned}$$

so we can compute $\gcd(f_1(x), \ldots, f_m(x))$ piecewise in AC^0. We can then interpolate the coefficients of $\gcd(f_1(x), \ldots, f_m(x))$ in AC^0 using Lemma 2.7. □

Using Theorem 8.4, we can also compute the least common multiple of multiple polynomials $f_1(x), \ldots, f_m(x)$ in AC^0. Although the identity $f(x)g(x) = \gcd(f(x), g(x)) \operatorname{lcm}(f(x), g(x))$ no longer holds for $m \geqslant 3$ polynomials, a suitable analogue of this identity will allow us to reduce the task of computing $\operatorname{lcm}(f_1, \ldots, f_m)$ to computing $\gcd(f_1, \ldots, f_m)$. The following elementary lemma relates the GCD and LCM of $m \geqslant 3$ polynomials.

Lemma 8.5. *Let $\mathbb{F}$ be any field. Let $f_1, \ldots, f_m \in \mathbb{F}[x]$ be univariate polynomials. Let $p \in \mathbb{F}[x]$ be a polynomial such that $f_i \mid p$ for all $i \in [m]$. Then*

$$\operatorname{lcm}(f_1, \ldots, f_m) = \frac{p}{\gcd\left(\frac{p}{f_1}, \ldots, \frac{p}{f_m}\right)}.$$

Using Lemma 8.5, we now compute the LCM of many polynomials in AC^0.

Corollary 8.6. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than m^2d^2. Let $f_1, \ldots, f_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Then their least common multiple $\operatorname{lcm}(f_1, \ldots, f_m)$ can be computed piecewise in AC^0.*

Proof. Let $s(x)$ be the squarefree part of $f_1(x) \cdots f_m(x)$. We may compute $s(x)$ piecewise in AC^0 using Corollary 7.7. Without loss of generality, suppose that $d = \max_{i\in[m]}(\deg(f_j))$. For each $i \in [m]$, the factors of f_i appear in f_i with multiplicity at most d, and each factor of f_i appears as a factor of s, so it follows that $f_i \mid s^d$.

By Lemma 8.5, we have

$$\operatorname{lcm}(f_1, \ldots, f_m) = \frac{s^d}{\gcd\left(\frac{s^d}{f_1}, \ldots, \frac{s^d}{f_m}\right)}.$$

Using Lemma 4.1, we can compute the coefficients of $s(x)^d/f_1(x), \ldots, s(x)^d/f_m(x)$ in AC^0. Note that $\deg(s) \leqslant md$, so $\deg(s^d/f_i) \leqslant \deg(s^d) \leqslant md^2$. Because $\operatorname{char}(\mathbb{F}) > m^2d^2$, we can compute $\gcd\left(\frac{s(x)^d}{f_1(x)}, \ldots, \frac{s(x)^d}{f_m(x)}\right)$ piecewise in AC^0 using Theorem 8.4. We can then compute the LCM by taking the quotient as above, which can be done in AC^0 using Lemma 4.1. □

9 Arbitrary Functions of Root Multiplicities

The GCD and LCM manipulate the multiplicities of the roots of two polynomials f and g in specific ways: the GCD assigns a root α its minimum multiplicity in f and g, while the LCM assigns α its maximum multiplicity. What other functions can we apply to the root multiplicities in AC^0? We discuss this first in the case of two polynomials, where *any* function of the root multiplicities can be implemented in AC^0, and then handle the case of many polynomials, where again any function (reasonably encoded) can be implemented in AC^0.

9.1 Two Polynomials

We now consider a generalization of the problem of computing greatest common divisors and least common multiples. Suppose we have two polynomials $f, g \in \mathbb{F}[x]$, which factor as

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}$$
$$g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{b_i},$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the union of the sets of roots of f and g and $a_i, b_i \in \mathbb{N}$, where some of the a_i or b_i may be zero. The greatest common divisor and least common multiple are given by

$$\gcd(f(x), g(x)) = \prod_{i=1}^{n} (x - \alpha_i)^{\min(a_i, b_i)}$$
$$\operatorname{lcm}(f(x), g(x)) = \prod_{i=1}^{n} (x - \alpha_i)^{\max(a_i, b_i)}.$$

That is, to form the greatest common divisor, if a factor $(x - \alpha_i)$ appears with multiplicity a_i in f and multiplicity b_i in g, the same factor should appear with multiplicity $\min(a_i, b_i)$ in $\gcd(f, g)$. The least common multiple replaces $\min(a_i, b_i)$ with $\max(a_i, b_i)$. As we saw in Section 8, we can implement these minimum and maximum operations without directly factorizing f and g.

What other functions of these multiplicities can we implement in (piecewise) AC^0? Addition is easy, as we have

$$f(x) \cdot g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i + b_i}.$$

What about products of multiplicities? That is, can we efficiently compute the coefficients of the polynomial

$$(f \diamond g)(x) \coloneqq \prod_{i=1}^{n} (x - \alpha_i)^{a_i b_i}?$$

It is not obvious that the coefficients of $f \diamond g$ can even be expressed algebraically in terms of the coefficients of f and g, much less that there is an efficient algorithm for this problem. However, a modification of our GCD algorithm from Theorem 8.1 enables us to compute $f \diamond g$, and indeed solve a more general problem.

Suppose we are given an *arbitrary* function $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and we would like to compute the polynomial

$$(f \diamond_P g)(x) \coloneqq \prod_{i=1}^{n} (x - \alpha_i)^{P(a_i, b_i)}.$$

By combining our algorithms for the squarefree decomposition and the GCD, we can design a piecewise AC^0 algorithm for $f \diamond_P g$ in a rather straightforward manner.

We start with the case where the function $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a delta function. That is, we consider functions of the form $\delta_{i,j} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ given by

$$\delta_{i,j}(a,b) \coloneqq \begin{cases} 1 & \text{if } a = i \text{ and } b = j \\ 0 & \text{otherwise.} \end{cases}$$

Every function $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ can be expressed as a sum of delta functions, so handling delta functions will allow us to easily handle arbitrary functions.

Lemma 9.1. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $2d$. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that f and g factor as*

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}$$
$$g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{b_i},$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_i, b_i \in \mathbb{N}$, where some of the a_i or b_i may be zero. Then the coefficients of the polynomial

$$(f \diamond_{\delta_{i,j}} g)(x) \coloneqq \prod_{k : a_k = i, b_k = j} (x - \alpha_k)$$

can be computed piecewise in AC^0.

Proof. Without loss of generality, let $d = \max(\deg(f), \deg(g))$. Let $(f_1, \ldots, f_d)$ and $(g_1, \ldots, g_d)$ be the squarefree decompositions of f and g, respectively. Recall that, by definition of the squarefree decomposition, we have $f = \prod_i f_i^i$, each polynomial f_i is squarefree, and $\gcd(f_i, f_j) = 1$ when $i \neq j$, and likewise for g. For notational ease, define $f_0 \coloneqq 1$ and $g_0 \coloneqq 1$.

Observe that for $i, j \in \{0, 1, \ldots, d\}$, we have

$$\gcd(f_i(x), g_j(x)) = \prod_{k : a_k = i, b_k = j} (x - \alpha_k) = (f \diamond_{\delta_{i,j}} g)(x).$$

Thus, we can compute $f \diamond_{\delta_{i,j}} g$ piecewise in AC^0 by first computing the squarefree decompositions of f and g using Lemma 7.6 and then computing $\gcd(f_i, g_j)$ via Theorem 8.1. □

We now compute $f \diamond_P g$ for an arbitrary function $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by writing P as a sum of delta functions.

Theorem 9.2. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $2d$. Let $f, g \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $P : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be an arbitrary function. Suppose that f and g factor as*

$$f(x) = \prod_{i=1}^{n} (x - \alpha_i)^{a_i}$$
$$g(x) = \prod_{i=1}^{n} (x - \alpha_i)^{b_i},$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_i, b_i \in \mathbb{N}$, where some of the a_i or b_i may be zero. Then the coefficients of the polynomial

$$(f \diamond_P g)(x) \coloneqq \prod_{i=1}^{n} (x - \alpha_i)^{P(a_i, b_i)}$$

can be computed piecewise in AC^0.

Proof. Using Lemma 9.1, we can compute

$$(f \diamond_{\delta_{i,j}} g)(x) = \prod_{k : a_k = i, b_k = j} (x - \alpha_i)$$

piecewise in AC^0. We can then write $f \diamond_P g$ as

$$\begin{aligned}(f \diamond_P g)(x) &= \prod_{i=0}^{d} \prod_{j=0}^{d} \prod_{k : a_k = i, b_k = j} (x - \alpha_k)^{P(i,j)} \\ &= \prod_{i=0}^{d} \prod_{j=0}^{d} (f \diamond_{\delta_{i,j}} g)(x)^{P(i,j)}.\end{aligned}$$

This results in a piecewise AC^0 circuit that computes $f \diamond_P g$. Applying Lemma 2.7 yields a piecewise AC^0 circuit that computes the coefficients of $f \diamond_P g$. □

9.2 Multiple Polynomials

Just as the greatest common divisor and least common multiple can be defined for a set of more than two polynomials, we can extend the setting and result of Theorem 9.2 to more polynomials. Suppose we are given polynomials $f_1, \ldots, f_m \in \mathbb{F}[x]$ where f_i factors as

$$f_i(x) = \prod_{j=1}^{n} (x - \alpha_j)^{a_{i,j}},$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ are the union of the set of roots of $f_1, \ldots, f_m$. For a function $P : \mathbb{N}^m \to \mathbb{N}$, we can define

$$\diamond_P(f_1, \ldots, f_m) \coloneqq \prod_{j=1}^{n} (x - \alpha_j)^{P(a_{1,j}, \ldots, a_{m,j})}.$$

As we will see, an appropriate generalization of Theorem 9.2 allows us to compute $\diamond_P(f_1, \ldots, f_m)$ in AC^0.

Here, we have two results, depending on how the function $P : \mathbb{N}^m \to \mathbb{N}$ is encoded. Our first result works with the dense representation, where the function P is specified as a list of input-output pairs. If the polynomials $f_1, \ldots, f_m$ have degree bounded by d, then the polynomial $\diamond_P(f_1, \ldots, f_m)$ depends only on the restricted function $P : \{0, 1, \ldots, d\}^m \to \mathbb{N}$, which can be represented by a list of $(d+1)^m$ values. Denote by $|P|_d$ the number of nonzero outputs of the restriction $P : \{0, 1, \ldots, d\}^m \to \mathbb{N}$. We will construct a circuit of constant depth and size polynomial in m and $|P|_d$ that computes $\diamond_P(f_1, \ldots, f_m)$.

Our second result handles a more succinct representation of the function $P : \mathbb{N}^m \to \mathbb{N}$. Namely, we consider functions represented by a special kind of circuit over the integers we call a "tropical threshold circuit." (See Definition 9.5.) Again, we show that $\diamond_P(f_1, \ldots, f_m)$ can be computed

efficiently with respect to the size of the given tropical threshold circuit that computes P. When the circuit representing P is itself of constant depth, we again compute $\diamond_P(f_1, \ldots, f_m)$ in AC^0.

We start by computing $\diamond_P(f_1, \ldots, f_m)$ when $P : \mathbb{N}^m \to \mathbb{N}$ is given in the sparse representation. As in the case of two polynomials, we start with delta functions.

Lemma 9.3. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than md. Let $f_1, \ldots, f_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Suppose that f_i factors as*

$$f_i(x) = \prod_{j=1}^{n} (x - \alpha_j)^{a_{i,j}}$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_{i,j} \in \mathbb{N}$, where some of the $a_{i,j}$ may be zero. Then for any $i_1, \ldots, i_m \in \{0, 1, \ldots, d\}$, the coefficients of the polynomial

$$\diamond_{\delta_{i_1,\ldots,i_m}}(f_1, \ldots, f_m) \coloneqq \prod_{k : a_{1,k} = i_1 \wedge \cdots \wedge a_{m,k} = i_m} (x - \alpha_k)$$

can be computed piecewise in AC^0.

Proof. Without loss of generality, let $d = \max_{i \in [m]} \deg(f_i)$. For each $i \in [m]$, let $(f_{i,1}, \ldots, f_{i,d})$ be the squarefree decomposition of f_i. For notational ease, define $f_{i,0} \coloneqq 1$.

Observe that

$$\gcd(f_{1,i_1}(x), \ldots, f_{m,i_m}(x)) = \prod_{k : a_{1,k} = i_1 \wedge \cdots \wedge a_{m,k} = i_m} (x - \alpha_k) = \diamond_{\delta_{i_1,\ldots,i_m}}(f_1, \ldots, f_m).$$

Thus, we can compute $\diamond_{\delta_{i_1,\ldots,i_m}}(f_1, \ldots, f_m)$ piecewise in AC^0 by first computing the squarefree decompositions of f_i for all $i \in [m]$ using Lemma 7.6 and then computing $\gcd(f_{1,i_1}, \ldots, f_{m,i_m})$ via Theorem 8.4. □

We now extend Lemma 9.3 to compute $\diamond_P(f_1, \ldots, f_m)$ for arbitrary functions $P : \mathbb{N}^m \to \mathbb{N}$.

Theorem 9.4. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than md. Let $f_1, \ldots, f_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $P : \mathbb{N}^m \to \mathbb{N}$ be a function given in the dense representation. Suppose that f_i factors as*

$$f_i(x) = \prod_{j=1}^{n} (x - \alpha_j)^{a_{i,j}}$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and $a_{i,j} \in \mathbb{N}$, where some of the $a_{i,j}$ may be zero. Then the coefficients of the polynomial

$$\diamond_P(f_1, \ldots, f_m) \coloneqq \prod_{j=1}^{n} (x - \alpha_j)^{P(a_{1,j}, \ldots, a_{m,j})}$$

can be computed piecewise in AC^0.

Proof. Using Lemma 9.3, we can compute

$$\diamond_{\delta_{i_1,\ldots,i_m}}(f_1, \ldots, f_m) = \prod_{k : a_{1,k} = i_1 \wedge \cdots \wedge a_{m,k} = i_m} (x - \alpha_k)$$

piecewise in AC^0. We can then write $\diamond_P(f_1, \ldots, f_m)$ as

$$\begin{aligned}\diamond_P(f_1, \ldots, f_m)(x) &= \prod_{i_1=0}^{d} \cdots \prod_{i_m=0}^{d} \prod_{k : a_{1,k} = i_1 \wedge \cdots \wedge a_{m,k} = i_m} (x - \alpha_k)^{P(i_1, \ldots, i_m)} \\ &= \prod_{i_1=0}^{d} \cdots \prod_{i_m=0}^{d} \diamond_{\delta_{i_1, \ldots, i_m}}(f_1, \ldots, f_m)^{P(i_1, \ldots, i_m)}.\end{aligned}$$

This yields a piecewise circuit of constant depth and size $m^{O(1)}|P|_d$ that computes $\diamond_P(f_1, \ldots, f_m)$, where $|P|_d$ denotes the number of nonzero outputs of the restriction $P : \{0, 1, \ldots, d\}^m \to \mathbb{N}$. Applying Lemma 2.7 yields a piecewise AC^0 circuit that computes the coefficients of $\diamond_P(f_1, \ldots, f_m)$. □

For constant m, Theorem 9.4 yields a circuit for $\diamond_P(f_1, \ldots, f_m)$ of size that is polynomially-bounded in the description length of $f_1, \ldots, f_m$. When m is large, for which functions $P : \mathbb{N}^m \to \mathbb{N}$ can we compute $\diamond_P(f_1, \ldots, f_m)$ more efficiently than Theorem 9.4? We can do this when P is one of the m-ary addition, maximum, minimum, or threshold functions. By an m-ary threshold function, we mean a Boolean function of the form

$$\operatorname{Thr}_{\overline{r}}(a_1, \ldots, a_m) = \begin{cases} 1 & \text{if } a_i \geqslant r_i \text{ for all } i \in [m], \\ 0 & \text{otherwise,} \end{cases}$$

where $\overline{r} \in \mathbb{N}^m$ is a fixed vector of thresholds, or its negation $\neg \operatorname{Thr}_{\overline{r}}$.

To compute $\diamond_P(f_1, \ldots, f_m)$ when P is the addition function, we use the basic fact that

$$\diamond_+(f_1, \ldots, f_m) = \prod_{i=1}^{m} f_i.$$

For the maximum and minimum functions, this is a straightforward application of the identities

$$\begin{aligned}\diamond_{\max}(f_1, \ldots, f_m) &= \operatorname{lcm}(f_1, \ldots, f_m) \\ \diamond_{\min}(f_1, \ldots, f_m) &= \gcd(f_1, \ldots, f_m)\end{aligned}$$

combined with Theorem 8.4 and Corollary 8.6. When P is a threshold function $\operatorname{Thr}_{\overline{r}}$ or its negation, the polynomial $\diamond_P(f_1, \ldots, f_m)$ corresponds to one of the outputs of Lemma 7.4.

Naturally, if the function $P : \mathbb{N}^m \to \mathbb{N}$ has a succinct description in terms of these basic functions, then we can also hope to compute $\diamond_P(f_1, \ldots, f_m)$ efficiently. We formalize this notion of complexity using tropical threshold circuits, defined below. The adjective "tropical" refers to tropical geometry [MS15], a form of algebraic geometry done over the tropical semiring $(\mathbb{R} \cup \{+\infty\}, \min, +)$ where addition and multiplication are replaced by minimum and addition, respectively.

Definition 9.5. A *tropical threshold circuit* is a directed acyclic graph. Vertices of in-degree zero are called *input gates* and each are labeled by some variable x_i. Vertices of positive in-degree are called *internal gates* and are labeled by an element of $\{+, c\times, \min, \max, \operatorname{Thr}_{\overline{r}}, \neg \operatorname{Thr}_{\overline{r}}\}$. Vertices of out-degree zero are called *output gates*. Each gate computes a function $\mathbb{N}^m \to \mathbb{N}$ in the natural way. The *size* of the circuit is the number of gates in the circuit and the sum of all constants appearing in $c\times$ gates. The *depth* of the circuit is the length of the longest path from an input gate to an output gate. ◊

We now state a version of Theorem 9.4 that shows we can compute $\diamond_P(f_1, \ldots, f_m)$ in size and depth proportional to the size and depth of a tropical threshold circuit that computes the function $P : \mathbb{N}^m \to \mathbb{N}$.

Theorem 9.6. *Let $\mathbb{F}$ be a field of characteristic zero. Let $f_1, \dots, f_m \in \mathbb{F}[x]$ be univariate polynomials of degree at most d given by their coefficients. Let $P : \mathbb{N}^m \to \mathbb{N}$ be a function computed by a tropical threshold circuit of size s and depth Δ. Suppose that f_i factors as*

$$f_i(x) = \prod_{j=1}^{n} (x - \alpha_j)^{a_{i,j}}$$

where $\alpha_1, \dots, \alpha_n \in \mathbb{F}$ and $a_{i,j} \in \mathbb{N}$, where some of the $a_{i,j}$ may be zero. Then the coefficients of the polynomial

$$\diamond_P(f_1, \dots, f_m) \coloneqq \prod_{j=1}^{n} (x - \alpha_j)^{P(a_{1,j}, \dots, a_{m,j})}$$

can be computed piecewise by a circuit of depth $O(\Delta)$ and size $(smd)^{O(1)}$.

Proof. We proceed by induction on Δ, simulating the tropical threshold circuit that computes P. When $\Delta = 1$, the function P must be computed by a single gate of a tropical threshold circuit. The following case analysis shows that for each gate type in a tropical threshold circuit, there is a piecewise arithmetic circuit of size $(md)^{O(1)}$ and depth $O(1)$ that computes $\diamond_P(f_1, \dots, f_m)$.

- If P is computed by an addition gate, we have

  $$\diamond_P(f_1, \dots, f_m) = \prod_{i=1}^{m} f_i,$$

  which can clearly be computed by a circuit of size $(md)^{O(1)}$ and depth $O(1)$.

- If P is computed by a $c\times$ gate, then

  $$\diamond_P(f) = f^c,$$

  which is easily computed by a circuit of size $(md)^{O(1)} + s$ and depth $O(1)$. Here, we use the fact $c \leqslant s$ by definition, so implementing the powering operation $f \mapsto f^c$ can be done with at most s wires.

- If P is computed by a minimum gate, we have

  $$\diamond_P(f_1, \dots, f_m) = \gcd(f_1, \dots, f_m),$$

  which can be computed piecewise by a circuit of size $(md)^{O(1)}$ and depth $O(1)$ using Theorem 8.4. Likewise, if P is computed by a maximum gate, we have

  $$\diamond_P(f_1, \dots, f_m) = \operatorname{lcm}(f_1, \dots, f_m),$$

  which can be computed in the claimed size and depth using Corollary 8.6.

- Suppose P is computed by a threshold gate $\operatorname{Thr}_{\overline{r}}$ or its negation $\neg \operatorname{Thr}_{\overline{r}}$. Let $h(x)$ be the squarefree part of $f_1(x) \cdots f_m(x)$. We can compute h piecewise with a circuit of size $(md)^{O(1)}$ and depth $O(1)$ using Corollary 7.7. By applying Lemma 7.4 to h and $f_1, \dots, f_m$ with threshold vector $\overline{r}$, we obtain $\diamond_{\operatorname{Thr}_{\overline{r}}}(f_1, \dots, f_m)$ and $\diamond_{\neg \operatorname{Thr}_{\overline{r}}}(f_1, \dots, f_m)$ piecewise using a circuit of size $(md)^{O(1)}$ and depth $O(1)$.

Overall, we obtain a circuit of size $(md)^{O(1)}$ and depth c for $\diamond_P(f_1, \ldots, f_m)$, where $c \in \mathbb{N}$ is some universal constant.

Suppose now that $\Delta \geqslant 2$. Let $P_1, \ldots, P_s$ be the functions computed by the children of the output gate in the tropical threshold circuit computing P. By induction, we can compute $g_i \coloneqq \diamond_{P_i}(f_1, \ldots, f_m)$ in size $(smd)^{O(1)}$ and depth $c \cdot (\Delta - 1)$. Let P_{out} be the function labeling the output gate of the circuit that computes P. By the same analysis as in the base case, we can compute

$$\diamond_{P_{\text{out}}}(g_1, \ldots, g_s) = \diamond_P(f_1, \ldots, f_m)$$

using $(md)^{O(1)}$ additional gates and increasing the depth by c. This yields a circuit that computes $\diamond_P(f_1, \ldots, f_m)$ of size $(smd)^{O(1)}$ and depth $c \cdot \Delta = O(\Delta)$ as claimed. □

10 Extensions to Multivariate Polynomials

Suppose we are given an arithmetic circuit of size s that computes a multivariate polynomial $f \in \mathbb{F}[\overline{x}]$. What can we say, if anything, about the complexity of the factors of f? A landmark result of Kaltofen [Kal89] shows that every irreducible factor of f can be computed by an arithmetic circuit of size $\text{poly}(s, \deg(f))$. This result is also constructive: there is a randomized polynomial-time algorithm that receives a circuit for f as input and outputs circuits for the irreducible factors of f. This was later extended to the black-box model by Kaltofen and Trager [KT90], where the factorization algorithm is given access to an evaluation oracle for f and must implement an evaluation oracle for each of the irreducible factors of f. Kaltofen's algorithm raises a natural question: what other classes of circuits are closed under factorization? By suitably modifying Kaltofen's algorithm, Sinhababu and Thierauf [ST21] showed that VBP, the class of polynomials computable by polynomial-size arithmetic branching programs, is closed under factorization. It is an open question whether NC^1 or AC^0 are closed under factorization, but partial results are known for both classes [DSY09; Oli16; CKS19; DSS22]. The fact that factors of small circuits can themselves be computed by small circuits is not only interesting in its own right, but it also plays a key role in the algebraic hardness versus randomness paradigm [KI04]. To establish hardness-to-pseudorandomness results for weaker circuit classes (where we have more hope of proving unconditional lower bounds), it would be enough to show that these weaker classes are closed under factorization.

In this section, we provide evidence that AC^0 and NC^1 are closed under factorization by showing that AC^0 and NC^1 are closed under the related operations of squarefree decomposition and GCD. For this, we need to extend some of our results from the univariate to the multivariate setting. By applying Lemma 7.6 in a straightforward manner, we show that for every polynomial $f \in \mathsf{AC}^0$, all elements $(f_1, \ldots, f_m)$ of the squarefree decomposition of f can themselves be computed in AC^0. Likewise, when $f \in \mathsf{NC}^1$, all elements of the squarefree decomposition of f can be computed in NC^1. A similar application of Theorem 8.1 shows that AC^0 and NC^1 are closed under taking greatest common divisors. Just like Kaltofen's theorem on factorization, these results are algorithmic, where the corresponding algorithms run in randomized polynomial time. In the case of AC^0, these algorithms can be derandomized in subexponential time by using the deterministic subexponential-time polynomial identity testing algorithm of [LST21] for AC^0 circuits.

Before moving to the details, a word on why it is reasonable for algorithms for univariate polynomials to be applicable in the multivariate setting. Thanks to Gauss's Lemma (Lemma 10.1), questions related to multivariate factorization in $\mathbb{F}[\overline{x}, y] \cong \mathbb{F}[\overline{x}][y]$ are often reducible to questions about factorization in $\mathbb{F}(\overline{x})[y]$. That is, we can regard a multivariate polynomial $f(\overline{x}, y) \in \mathbb{F}[\overline{x}, y]$ as a univariate polynomial in y whose coefficients come from the field $\mathbb{K} \coloneqq \mathbb{F}(\overline{x})$, and under certain assumptions on f, enlarging the ring of coefficients from $\mathbb{F}[\overline{x}]$ to $\mathbb{F}(\overline{x})$ does not affect the factorization

of f. In this setting, it is natural to apply our earlier univariate algorithms to solve factorization problems. Although the field $\mathbb{K}$ contains elements of very high complexity, Lemma 2.7 implies that when the input f is represented as a small arithmetic circuit, the $\mathbb{K}$-coefficients of f are also representable as small arithmetic circuits. This allows us to apply our machinery from the univariate setting without a large blow-up in complexity.

Of course, we need to ensure that the answer to our problem over $\mathbb{K}(y)$ is the same as the answer over $\mathbb{F}[\overline{x}][y]$. This is not always the case. For example, over $\mathbb{F}[x_1, x_2][y]$, we have

$$\gcd(x_1 x_2, x_1) = x_1,$$

whereas over $\mathbb{F}(x_1, x_2)[y]$, we have

$$\gcd(x_1 x_2, x_1) = 1.$$

As we will see, Gauss's Lemma guarantees that if the polynomial $f \in \mathbb{F}[\overline{x}][y]$ is monic in y, then the factorizations of f into irreducibles over $\mathbb{F}[\overline{x}][y]$ and over $\mathbb{F}(\overline{x})[y]$ are the same. In particular, if we were to take the GCD in $\mathbb{K}(y)$ of two monic polynomials, we would obtain the same result as the GCD in $\mathbb{F}[\overline{x}][y]$. Although our input polynomials are not guaranteed to be monic, we will see that there is a simple, standard way to reduce to the monic case.

10.1 Preliminaries on Polynomial Factorization and Identity Testing

We now recall some preliminary material on polynomial factorization and polynomial identity testing. This material is standard, and the reader familiar with these problems can safely skip ahead to the next subsection, returning here as necessary.

10.1.1 Gauss's Lemma

We first state Gauss's Lemma, which allows us to pass from $\mathbb{F}[\overline{x}][y]$ to $\mathbb{F}(\overline{x})[y]$ when studying questions related to polynomial factorization. The version of Gauss's Lemma we state here suffices for our purposes; for a more general statement of the lemma, see von zur Gathen and Gerhard [vzGG13, Corollary 6.10].

Lemma 10.1 (Gauss's Lemma [vzGG13, Corollary 6.10])**.** *Let $f \in \mathbb{F}[\overline{x}, y]$ be monic in y. Then f is irreducible in $\mathbb{F}(\overline{x})[y]$ if and only if f is irreducible in $\mathbb{F}[\overline{x}][y] \cong \mathbb{F}[\overline{x}, y]$.*

If a polynomial $f \in \mathbb{F}[\overline{x}, y]$ is monic in y, then every irreducible factor of f must also be monic in y, and so Gauss's Lemma implies that the irreducible factors of f over $\mathbb{F}[\overline{x}][y]$ are also irreducible over $\mathbb{F}(\overline{x})[y]$. In particular, for polynomials that are monic in y, their factorization into irreducibles is the same in $\mathbb{F}[\overline{x}][y]$ and $\mathbb{F}(\overline{x})[y]$.

Corollary 10.2. *Let $f \in \mathbb{F}[\overline{x}, y]$ be monic in y. Then the factorization of f into irreducibles is the same over $\mathbb{F}(\overline{x})[y]$ and $\mathbb{F}[\overline{x}][y]$.*

Corollary 10.2 is useful for reducing multivariate factorization problems to univariate ones. If a polynomial $f \in \mathbb{F}[\overline{x}, y]$ is monic in y, then we lose no information about the factorization of f by viewing f as a univariate polynomial in y with coefficients in the field $\mathbb{F}(\overline{x})$. A standard technique to reduce to the monic case is to apply a random change of variables to a polynomial $f \in \mathbb{F}[\overline{x}]$.

Lemma 10.3. *Let $f \in \mathbb{F}[\overline{x}]$ and let $\overline{\alpha} \in \mathbb{F}^n$. Let $d = \deg(f)$ and let $f_d \in \mathbb{F}[\overline{x}]$ be the top-degree homogeneous component of f. Let y be a fresh variable and define*

$$\hat{f}(\overline{x}, y) \coloneqq f(\overline{x} + y \cdot \overline{\alpha}) = f(x_1 + y \cdot \alpha_1, \ldots, x_n + y \cdot \alpha_n).$$

Then the following hold.

1. *If $f_d(\overline{\alpha}) \neq 0$, then $\frac{1}{f_d(\overline{\alpha})}\hat{f}(\overline{x}, y)$ is monic in y.*

2. *$g(\overline{x})$ is an irreducible factor of $f(\overline{x})$ in $\mathbb{F}[\overline{x}]$ if and only if $g(\overline{x} + y \cdot \overline{\alpha})$ is an irreducible factor of $\hat{f}(\overline{x}, y)$ in $\mathbb{F}[\overline{x}, y]$.*

Proof. Write f as a sum of monomials

$$f(\overline{x}) = \sum_{\overline{e} \in \mathbb{N}^n} c_{\overline{e}} \overline{x}^{\overline{e}}.$$

Expand $\hat{f}(\overline{x}, y)$ using the binomial theorem as

$$\hat{f}(\overline{x}, y) = \sum_{\overline{e} \in \mathbb{N}^n} c_{\overline{e}} \cdot (\overline{x} + y \cdot \overline{\alpha})^{\overline{e}} = \sum_{\overline{e} \in \mathbb{N}^n} c_{\overline{e}} \sum_{\overline{a} + \overline{b} = \overline{e}} \binom{\overline{e}}{\overline{a}} \overline{x}^{\overline{a}} \overline{\alpha}^{\overline{b}} y^{\left\|\overline{b}\right\|_1},$$

where $\binom{\overline{e}}{\overline{a}} \coloneqq \prod_{i=1}^{n} \binom{e_i}{a_i}$. A term in the above summation is divisible by y^d if and only if $\left\|\overline{b}\right\|_1 \geqslant d$. Each term in the above sum satisfies $\left\|\overline{b}\right\|_1 \leqslant \left\|\overline{e}\right\|_1$ and $\left\|\overline{e}\right\|_1 \leqslant \deg(f) = d$, so the terms divisible by y^d are those for which $\left\|\overline{b}\right\|_1 = \left\|\overline{e}\right\|_1$, which implies $\overline{b} = \overline{e}$ and consequently $\overline{a} = \overline{0}$. Thus, the coefficient of y^d in the expansion of $\hat{f}(\overline{x}, y)$ is given by

$$\sum_{\overline{e} \in \mathbb{N}^n} c_{\overline{e}} \overline{\alpha}^{\overline{e}} = f_d(\overline{\alpha}).$$

Hence $\frac{1}{f_d(\overline{\alpha})}\hat{f}(\overline{x}, y)$ is monic in y if $f_d(\overline{\alpha}) \neq 0$.

The second claim follows from the fact that the map $(\overline{x}, y) \mapsto (\overline{x} + y \cdot \overline{\alpha}, y)$ is invertible with inverse $(\overline{x}, y) \mapsto (\overline{x} - y \cdot \overline{\alpha}, y)$. That is, suppose $f(\overline{x}) = g(\overline{x})h(\overline{x})$ where g is irreducible and h is nonzero. Then it is clear that

$$f(\overline{x} + y \cdot \overline{\alpha}) = g(\overline{x} + y \cdot \overline{\alpha}) \cdot h(\overline{x} + y \cdot \overline{\alpha}),$$

so $g(\overline{x} + y \cdot \overline{\alpha})$ is a factor of $f(\overline{x} + y \cdot \overline{\alpha})$. To see that $g(\overline{x} + y \cdot \overline{\alpha})$ is irreducible, suppose that $g(\overline{x} + y \cdot \overline{\alpha})$ factors as

$$g(\overline{x} + y \cdot \overline{\alpha}) = r(\overline{x}, y) \cdot s(\overline{x}, y).$$

Then we have

$$g(\overline{x}) = r(\overline{x} - y \cdot \overline{\alpha}, y) \cdot s(\overline{x} - y \cdot \overline{\alpha}, y).$$

Because g is irreducible, one of the above factors factors must be an element of $\mathbb{F}$. Without loss of generality, we have $s(\overline{x} - y \cdot \overline{\alpha}, y) \in \mathbb{F}$, so it follows that $s(\overline{x}, y) \in \mathbb{F}$, which implies that $g(\overline{x} + y \cdot \overline{\alpha})$ is irreducible. Thus, if $g(\overline{x})$ is an irreducible factor of $f(\overline{x})$, then $g(\overline{x} + y \cdot \overline{\alpha})$ is an irreducible factor of $\hat{f}(\overline{x}, y)$.

A symmetric argument shows that if $\hat{g}(\overline{x}, y)$ is an irreducible factor of $\hat{f}(\overline{x}, y)$, then $\hat{g}(\overline{x} - y \cdot \overline{\alpha}, y)$ is an irreducible factor of $f(\overline{x})$. Because f does not depend on y, this implies that there is some polynomial $g(\overline{x})$ such that $g(\overline{x}) = \hat{g}(\overline{x} - y \cdot \overline{\alpha}, y)$, so $\hat{g}$ has the form $\hat{g}(\overline{x}, y) = g(\overline{x} + y \cdot \overline{\alpha})$ as claimed. □

10.1.2 Polynomial Identity Testing

Polynomial identity testing (abbreviated as PIT) is the algorithmic problem of deciding if a given arithmetic circuit C computes the identically zero polynomial. In arithmetic complexity, polynomial identity testing is the central example of a problem that can be solved efficiently with the use of randomness, but for which an efficient deterministic algorithm is not known. The following lemma, often referred to as the Schwartz–Zippel lemma, provides the basis for an efficient randomized algorithm to solve PIT.

Lemma 10.4 ([Sch80; Zip79]). *Let $f \in \mathbb{F}[\overline{x}]$ be a nonzero polynomial of degree d. Then for any finite $S \subseteq \mathbb{F}$, we have*

$$\Pr_{\overline{\alpha} \in S^n}[f(\overline{\alpha}) = 0] \leqslant \frac{d}{|S|}.$$

To test if a circuit C of degree d computes the zero polynomial, choose a set $S \subseteq \mathbb{F}$ of size $2d$ and evaluate C at a randomly chosen point $\overline{\alpha} \in S^n$. The algorithm reports that C computes the zero polynomial if and only if $C(\overline{\alpha}) = 0$. It is clear that this algorithm has one-sided error (it is always correct when C computes the zero polynomial), and the Schwartz–Zippel lemma implies that this algorithm has error probability at most $1/2$, so PIT is in coRP.

The design of deterministic algorithms for polynomial identity testing is a large, active area of research, with close connections to arithmetic circuit lower bounds [KI04; KS19]. Our focus here is not on the design of new algorithms for PIT, but rather on applications of existing and future algorithms. For an introduction to identity testing, we refer the reader to Shpilka and Yehudayoff [SY10].

There is a search-to-decision reduction for PIT, analogous to the search-to-decision reduction for Boolean satisfiability. Suppose $f(\overline{x})$ is a nonzero polynomial of degree d. Lemma 10.4 implies that for any set $S = \{\alpha_1, \ldots, \alpha_{d+1}\} \subseteq \mathbb{F}$, there is a point $\overline{\beta} \in S^n$ such that $f(\overline{\beta}) \neq 0$. In particular, for some $i \in [d+1]$, the polynomial $f(\alpha_i, x_2, \ldots, x_n)$ is nonzero, and we can find such an i by solving PIT for the polynomials $f(\alpha_i, x_2, \ldots, x_n)$. This produces $\beta_1 \in \mathbb{F}$ such that $f(\beta_1, x_2, \ldots, x_n) \neq 0$. By recursion, we can find the remaining values $\beta_2, \ldots, \beta_n \in \mathbb{F}$ such that $f(\overline{\beta}) \neq 0$. We record this search-to-decision reduction for polynomial identity testing in the following lemma.

Lemma 10.5. *Let $\mathcal{C}$ be a class of arithmetic circuits closed under substitution. Given a $\mathcal{C}$-circuit $C(x_1, \ldots, x_n)$ of degree d, we can either determine that C computes the zero polynomial or find point $\overline{\beta} \in \mathbb{F}^n$ such that $C(\overline{\beta}) \neq 0$ using $O(nd)$ calls to an oracle that solves PIT for $\mathcal{C}$-circuits.*

The breakthrough lower bounds for AC^0 circuits by Limaye, Srinivasan, and Tavenas [LST21] resulted in a deterministic subexponential-time PIT algorithm for AC^0 circuits. We will make use of this algorithm later to derandomize our multivariate algorithms in subexponential time.

Theorem 10.6 ([LST21]). *Let $\mathbb{F}$ be a field of characteristic zero. For all $\varepsilon > 0$, there is a deterministic algorithm that receives as input an n-variate arithmetic circuit C of size s and depth $\Delta \leqslant o(\log\log\log n)$ and decides if C computes the zero polynomial in time $(s^{\Delta+1}n)^{O(n^\varepsilon)}$.*

10.2 Multivariate Algorithms from Univariate Algorithms

The algorithms of Sections 7 and 8 for the squarefree decomposition, GCD, and LCM were all stated and proved for univariate polynomials over a field $\mathbb{F}$. If we have a multivariate polynomial $f \in \mathbb{F}[\overline{x}]$, we can always regard f as a univariate in one variable, say x_n, whose coefficients are from the field $\mathbb{K} = \mathbb{F}(x_1, \ldots, x_{n-1})$. Our earlier univariate algorithms make no assumptions about the underlying field, other than requiring the characteristic to be sufficiently large. Because $\operatorname{char}(\mathbb{K}) = \operatorname{char}(\mathbb{F})$, we can apply these univariate algorithms to $f \in \mathbb{K}[x_n]$ without any modification to the algorithm. This raises two questions that must be addressed. The first is an issue of representation: the field $\mathbb{K} = \mathbb{F}(x_1, \ldots, x_{n-1})$ is more complicated than $\mathbb{F}$, and we need a succinct way to encode the $\mathbb{K}$-coefficients of a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$. The second deals with correctness: why does solving a problem such as the GCD over $\mathbb{F}(x_1, \ldots, x_{n-1})[x_n]$ yield the solution to the same problem over $\mathbb{F}[x_1, \ldots, x_{n-1}][x_n] \cong \mathbb{F}[x_1, \ldots, x_n]$?

The issue of representation is easily dealt with. The natural way to represent the input f is via an arithmetic circuit. Writing

$$f(\overline{x}) = \sum_{i=0}^{d} f_i(x_1, \ldots, x_{n-1}) x_n^i,$$

we see that the $\mathbb{K}$-coefficients of f are exactly the polynomials $f_0, \ldots, f_d$. An application of Lemma 2.7 shows that the coefficients f_i can be computed by circuits whose size and depth is comparable to that of the circuit for f.

To deal with correctness, Corollary 10.2 guarantees that our reduction to the univariate case will produce the correct answer, provided the polynomial f is monic in some variable. We achieve this using Lemma 10.3, which provides a linear change of variables that ensure f is monic in a fresh variable y. Because the resulting change of variables is invertible, we will be able to easily recover the solution that corresponds to the original input f.

We now implement the argument sketched above to compute the squarefree decomposition of a multivariate polynomial represented by an arithmetic circuit. We focus on the classes AC^0 and NC^1, as it was previously known that NC^2 and VBP were closed under squarefree decomposition.

Theorem 10.7. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than d. Let $\mathcal{C} \in \{\mathsf{AC}^0, \mathsf{NC}^1\}$. Let $\mathcal{O}$ be an oracle that solves polynomial identity testing for $\mathcal{C}$-circuits. There is a deterministic, polynomial-time algorithm with oracle access to $\mathcal{O}$ that does the following.*

1. *The algorithm receives as input a $\mathcal{C}$-circuit that computes a polynomial $f \in \mathbb{F}[\overline{x}]$ of degree d.*
2. *The algorithm outputs a collection of $\mathcal{C}$-circuits $C_1, \ldots, C_m$ such that C_i computes f_i, where $(f_1, \ldots, f_m)$ is the squarefree decomposition of f.*

Proof. Let $f_{\mathrm{top}} \in \mathbb{F}[\overline{x}]$ be the top-degree homogeneous component of f. Applying Lemma 2.7 to f, we obtain a $\mathcal{C}$-circuit that computes f_{top}. By Lemma 10.5, we can find a point $\overline{\alpha} \in \mathbb{F}^n$ such that $f_{\mathrm{top}}(\overline{\alpha}) \neq 0$ using $O(nd)$ calls to the PIT oracle $\mathcal{O}$. Let y be a fresh variable and define

$$\hat{f}(\overline{x}, y) \coloneqq \frac{1}{f_{\mathrm{top}}(\overline{\alpha})} f(\overline{x} + y \cdot \overline{\alpha}).$$

By Lemma 10.3, the polynomial $\hat{f}$ is monic and the irreducible factors of $\hat{f}$ are in one-to-one correspondence with those of f.

The correspondence between the irreducible factors of f and $\hat{f}$ implies that the elements of the squarefree decomposition of f and $\hat{f}$ are in one-to-one correspondence. To see this, write the factorization of f into irreducibles as

$$f = \prod_{i=1}^{r} f_i^{d_i},$$

where each $f_i \in \mathbb{F}[\overline{x}]$ is irreducible and $\gcd(f_i, f_j) = 1$ for $i \neq j$. Because the factors of f and $\hat{f}$ are in one-to-one correspondence, the factorization of $\hat{f}$ into irreducibles is given by

$$\hat{f} = \prod_{i=1}^{r} f_i(\overline{x} + y \cdot \overline{\alpha})^{d_i}.$$

The squarefree decomposition of $\hat{f}$ consists of the polynomials

$$\hat{g}_j \coloneqq \prod_{i : d_i = j} f_i(\overline{x} + y \cdot \overline{\alpha})$$

for $j \in [d]$. Applying the change of variables $(\overline{x}, y) \mapsto (\overline{x} - y \cdot \overline{\alpha}, y)$ to $\hat{g}_j$, we obtain

$$g_j \coloneqq \prod_{i : d_i = j} f_i(\overline{x}),$$

which is precisely the j^{th} element of the squarefree decomposition of f. Thus, if we can compute the squarefree decomposition of $\hat{f}$, then the change of variables $(\overline{x}, y) \mapsto (\overline{x} - y \cdot \overline{\alpha}, y)$ yields the squarefree decomposition of f.

It remains to compute the squarefree decomposition of $\hat{f}$. Write $\hat{f}(\overline{x}, y) = \sum_{i=0}^{d} \hat{f}_i(\overline{x}) y^i$. By Lemma 2.7, each $\hat{f}_i$ can be computed by a $\mathcal{C}$-circuit. Regarding $\hat{f}(\overline{x}, y) \in \mathbb{F}(\overline{x})[y]$ as a polynomial in y with coefficients in $\mathbb{F}(\overline{x})$, the coefficients of $\hat{f}$ are precisely $\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_d$. Applying the piecewise AC^0 algorithm (recall Definition 2.3) of Lemma 7.6 to $\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_d$, we obtain a collection of multi-output circuits where one circuit in this collection correctly computes the squarefree decomposition of $\hat{f}$. Each circuit in this collection is the composition of a $\mathcal{C}$-circuit and an AC^0 circuit. For $\mathcal{C} \in \{\mathsf{AC}^0, \mathsf{NC}^1\}$, this composition results in a $\mathcal{C}$-circuit as claimed.

To select the circuit that correctly computes the squarefree decomposition of $\hat{f}$ requires us to evaluate the test circuits of the piecewise AC^0 algorithm on the coefficients $\hat{f}_0, \ldots, \hat{f}_d$. The test circuits are also the composition of a $\mathcal{C}$-circuit and an AC^0 circuit, which results in a $\mathcal{C}$-circuit. Deciding if a given test circuit is zero or nonzero can be done using the PIT oracle $\mathcal{O}$. This allows us to correctly select and output the circuit that computes the squarefree decomposition of $\hat{f}$. □

Using an essentially identical argument, we can design an algorithm to compute the GCD and LCM of multiple polynomials given as arithmetic circuits. As above, we focus on the classes AC^0 and NC^1, since NC^2 and VBP were known to be closed under taking GCD's and LCM's.

Theorem 10.8. *Let $\mathbb{F}$ be a field of characteristic zero or characteristic greater than $m^2 d^2$. Let $\mathcal{C} \in \{\mathsf{AC}^0, \mathsf{NC}^1\}$. Let $\mathcal{O}$ be an oracle that solves polynomial identity testing for $\mathcal{C}$-circuits. There is a deterministic, polynomial-time algorithm with oracle access to $\mathcal{O}$ that does the following.*

1. *The algorithm receives as input m $\mathcal{C}$-circuits that compute polynomials $f_1, \ldots, f_m \in \mathbb{F}[\overline{x}]$ of degree at most d.*
2. *The algorithm outputs $\mathcal{C}$-circuits $C_{\gcd}$ and C_{lcm} that compute $\gcd(f_1, \ldots, f_m)$ and $\operatorname{lcm}(f_1, \ldots, f_m)$, respectively.*

Proof. The proof is essentially the same as Theorem 10.7, replacing the use of Lemma 7.6 with either Theorem 8.4 or Corollary 8.6 as appropriate. The only difference is the fact that we need to find a change of variables that makes all of the polynomials $f_1, \ldots, f_m$ simultaneously monic in a fresh variable y.

To find this change of variables, let $d_i \coloneqq \deg(f_i)$. For $i \in [m]$, let $f_{i,\text{top}} \in \mathbb{F}[\overline{x}]$ be the top-degree homogeneous component of f_i. Applying Lemma 2.7 to f_i, we obtain a $\mathcal{C}$-circuit that computes $f_{i,\text{top}}$. It follows that the product $\prod_{i=1}^{m} f_{i,\text{top}}$ can be computed by a $\mathcal{C}$-circuit. Lemma 10.5 implies that we can find a point $\overline{\alpha} \in \mathbb{F}^n$ such that $\prod_{i=1}^{m} f_{i,\text{top}}(\overline{\alpha}) \neq 0$ using $O(nmd)$ calls to the PIT oracle $\mathcal{O}$. This implies that $f_{i,\text{top}}(\overline{\alpha}) \neq 0$ for each $i \in [m]$.

Let y be a fresh variable and define

$$\hat{f}_i(\overline{x}, y) \coloneqq \frac{1}{f_{i,\text{top}}(\overline{\alpha})} f_i(\overline{x} + y \cdot \overline{\alpha}).$$

Because $f_{i,\text{top}}(\overline{\alpha}) \neq 0$ for each $i \in [m]$, Lemma 10.3 implies that $\hat{f}_i$ is monic in y for each $i \in [m]$. At this point, the proof proceeds in the same manner as the proof of Theorem 10.7. □

Finally, we observe that in the case of AC^0 circuits, the preceding algorithms for the multivariate squarefree decomposition, GCD, and LCM can all be derandomized in subexponential time. This is done by implementing the PIT oracles in Theorems 10.7 and 10.8 using Theorem 10.6.

Corollary 10.9. *In the case $\mathcal{C} = \mathsf{AC}^0$, the algorithms of Theorems 10.7 and 10.8 can be implemented in deterministic subexponential time.*

11 Conclusions and Open Problems

In this work, we introduced techniques for manipulating evaluations of symmetric polynomials $\{e_k(\alpha_1, \ldots, \alpha_n) : k \in [n]\}$ and $\{e_k(\beta_1, \ldots, \beta_m) : k \in [m]\}$ without having explicit access to the α_i and β_j. We have seen how to change the points at which the e_i are evaluated (Section 5) and how to implement combinatorial operations like set difference, intersection, and union on the sets $\{\alpha_1, \ldots, \alpha_n\}$ and $\{\beta_1, \ldots, \beta_m\}$ (Sections 7 and 8). We also saw how to incorporate multiplicities into these set operations when $\{\alpha_1, \ldots, \alpha_n\}$ and $\{\beta_1, \ldots, \beta_m\}$ are regarded as multisets. These techniques naturally led to efficient algorithms for symmetric functions (such as the discriminant) and bi-symmetric functions (such as the resultant and the GCD). How far can these techniques be pushed, and what are their limitations?

A natural question along these lines is to understand which symmetric polynomials can be computed in AC^0. Chaugule, Kumar, Limaye, et al. [CKL+23] showed that certain Schur polynomials are at least as hard as the determinant, hence are not in AC^0. Curticapean, Limaye, and Srinivasan [CLS22] showed that some families of monomial symmetric polynomials are VNP-hard, i.e., a polynomial-size circuit for them would imply that the permanent has polynomial-size circuits. As the permanent is known to not be in AC^0, the same lower bound holds for the monomial symmetric polynomials. Can we hope to classify which families of symmetric polynomials are in AC^0?

A related question deals with the complexity of the fundamental theorem of symmetric polynomials. If $f(x_1, \ldots, x_n)$ is symmetric, then there is a unique g such that $f(\overline{x}) = g(e_1(\overline{x}), \ldots, e_n(\overline{x}))$. How do the complexity of f and g relate to one another? Bläser and Jindal [BJ19] showed that f and g have comparable circuit complexity. A variation on this for arithmetic branching programs is a key ingredient in the result of Chaugule, Kumar, Limaye, et al. [CKL+23] mentioned above. What can be said about the complexity of f and g when they are written as formulas, or as low-depth circuits?

Many problems in polynomial algebra can be solved in NC^2 by reduction to linear algebra. As we have seen, these algorithms can be improved all the way to AC^0 for the GCD and related problems. However, this only scratches the surface of linear algebra's applications to polynomial algebra. The fact that these natural NC^2 algorithms can be improved suggests that there is more to say about the parallel complexity of algebraic problems.

One interesting problem left open by this work is to determine the parallel complexity of the extended Euclidean scheme. The *extended Euclidean scheme* of two polynomials $f, g \in \mathbb{F}[x]$ consists of the sequence of remainders and Bézout coefficients produced during the execution of the Euclidean algorithm on input f and g. Linear algebra provides an NC^2 algorithm to compute the extended Euclidean scheme [vzGat84]. Can the techniques of this paper be used to design circuits of lower depth? The extended Euclidean scheme has many applications, including rational interpolation, Padé approximation, continued fraction expansion, counting real roots of polynomials in $\mathbb{R}[x]$ (as a consequence of Sturm's theorem), and solving Toeplitz and Hankel systems of linear equations (see [vzGG13, Chapters 4 and 5] and [BGY80] for details). As a step towards understanding the complexity of the extended Euclidean scheme, can we find faster parallel algorithms for any of these related problems?

References

[Béz64] Étienne Bézout. "Recherches sur le degré des équations résultantes de l'évanouissement des inconnues et sur les moyens que l'on doit employer pour trouver ces équations". In: *Histoire de l'académie royale des sciences* (1764), pp. 288–338 (cit. on p. 16).

[BFSS06] Alin Bostan, Philippe Flajolet, Bruno Salvy, and Éric Schost. "Fast computation of special resultants". In: *Journal of Symbolic Computation* 41.1 (2006), pp. 1–29. ISSN: 0747-7171. DOI: 10.1016/j.jsc.2005.07.001 (cit. on pp. 7, 18, 19, 27).

[BGY80] Richard P. Brent, Fred G. Gustavson, and David Y. Y. Yun. "Fast solution of Toeplitz systems of equations and computation of Padé approximants". In: *Journal of Algorithms* 1.3 (1980), pp. 259–295. ISSN: 0196-6774. DOI: 10.1016/0196-6774(80)90013-9 (cit. on p. 55).

[Bin84] Dario Bini. "Parallel Solution of Certain Toeplitz Linear Systems". In: *SIAM Journal on Computing* 13.2 (1984), pp. 268–276. DOI: 10.1137/0213019 (cit. on pp. 3, 5, 13, 32).

[BJ19] Markus Bläser and Gorav Jindal. "On the Complexity of Symmetric Polynomials". In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Ed. by Avrim Blum. Vol. 124. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 47:1–47:14. ISBN: 978-3-95977-095-8. DOI: 10.4230/LIPIcs.ITCS.2019.47 (cit. on p. 55).

[BP85] Dario Bini and Victor Ya. Pan. "Fast parallel polynomial division via reduction to triangular toeplitz matrix inversion and to polynomial inversion modulo a power". In: *Information Processing Letters* 21.2 (1985), pp. 79–81. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(85)90037-7 (cit. on pp. 3, 5, 13).

[BP94] Dario Bini and Victor Y. Pan. *Polynomial and Matrix Computations*. Progress in Theoretical Computer Science. Birkhäuser Boston, MA, 1994. ISBN: 978-0-8176-3786-6. DOI: 10.1007/978-1-4612-0265-3 (cit. on p. 16).

[Bür00] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Springer-Verlag Berlin Heidelberg, 2000. DOI: 10.1007/978-3-662-04179-6 (cit. on p. 2).

[CKL+23] Prasad Chaugule, Mrinal Kumar, Nutan Limaye, Chandra Kanta Mohapatra, Adrian She, and Srikanth Srinivasan. "Schur Polynomials Do Not Have Small Formulas If the Determinant does not". In: *Computational Complexity* 32 (2023). DOI: 10.1007/s00037-023-00236-x (cit. on pp. 4, 55).

[CKS19] Chi-Ning Chou, Mrinal Kumar, and Noam Solomon. "Closure Results for Polynomial Factorization". In: *Theory of Computing* 15.13 (2019). Preliminary version in the *33rd Annual Computational Complexity Conference (CCC 2018)*, pp. 1–34. DOI: 10.4086/toc.2019.v015a013 (cit. on p. 49).

[CLO05] David A. Cox, John Little, and Donal O'Shea. *Using Algebraic Geometry*. 2nd ed. Springer New York, NY, 2005. DOI: 10.1007/b138611 (cit. on p. 15).

[CLO15] David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra*. 4th ed. Undergraduate texts in mathematics. Springer, 2015. ISBN: 978-3-319-16720-6 (cit. on pp. 25–27).

[CLS22] Radu Curticapean, Nutan Limaye, and Srikanth Srinivasan. "On the VNP-Hardness of Some Monomial Symmetric Polynomials". In: *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*. Ed. by Anuj Dawar and Venkatesan Guruswami. Vol. 250. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 16:1–16:14. ISBN: 978-3-95977-261-7. DOI: 10.4230/LIPIcs.FSTTCS.2022.16 (cit. on p. 55).

[Coo85] Stephen A. Cook. "A taxonomy of problems with fast parallel algorithms". In: *Information and Control* 64.1 (1985), pp. 2–22. DOI: 10.1016/S0019-9958(85)80041-3 (cit. on p. 10).

[Csa76] L. Csanky. "Fast Parallel Matrix Inversion Algorithms". In: *SIAM Journal on Computing* 5.4 (1976), pp. 618–623. DOI: 10.1137/0205040 (cit. on pp. 3, 7).

[DSS22] Pranjal Dutta, Nitin Saxena, and Amit Sinhababu. "Discovering the Roots: Uniform Closure Results for Algebraic Classes Under Factoring". In: *J. ACM* 69.3 (June 2022). ISSN: 0004-5411. DOI: 10.1145/3510359 (cit. on p. 49).

[DSY09] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. "Hardness-Randomness Tradeoffs for Bounded Depth Arithmetic Circuits". In: *SIAM J. Comput.* 39.4 (2009), pp. 1279–1293. DOI: 10.1137/080735850 (cit. on p. 49).

[Hya79] Laurent Hyafil. "On the Parallel Evaluation of Multivariate Polynomials". In: *SIAM Journal on Computing* 8.2 (1979), pp. 120–123. DOI: 10.1137/0208010 (cit. on p. 3).

[Kal89] Erich Kaltofen. "Factorization of Polynomials Given by Straight-Line Programs". In: *Advances in Computing Research* 5 (1989), pp. 375–412 (cit. on p. 49).

[KI04] Valentine Kabanets and Russell Impagliazzo. "Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds". In: *Computational Complexity* 13.1-2 (2004), pp. 1–46. DOI: 10.1007/s00037-004-0182-6 (cit. on pp. 49, 52).

[KS19] Mrinal Kumar and Ramprasad Saptharishi. "Hardness-Randomness Tradeoffs for Algebraic Computation". In: *Bull. Eur. Assoc. Theor. Comput. Sci.* 129 (2019), pp. 56–87 (cit. on p. 52).

[KT90] Erich Kaltofen and Barry M. Trager. "Computing with Polynomials Given By Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators". In: *J. Symb. Comput.* 9.3 (1990), pp. 301–320. DOI: 10.1016/S0747-7171(08)80015-6 (cit. on p. 49).

[LST21] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. "Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits". In: *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2021)*. 2021, pp. 804–814. DOI: 10.1109/FOCS52979.2021.00083 (cit. on pp. 4, 10, 49, 52).

[MS15] Diane Maclagan and Bernd Sturmfels. *Introduction to tropical geometry*. Vol. 161. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2015, pp. xii+363. ISBN: 978-0-8218-5198-2. DOI: 10.1090/gsm/161 (cit. on p. 47).

[Oli16] Rafael Oliveira. "Factors of low individual degree polynomials". In: *Computational Complexity* 25.2 (2016), pp. 507–561. ISSN: 1016-3328. DOI: 10.1007/s00037-016-0130-2. Preliminary version in the *30th Annual Computational Complexity Conference (CCC 2015)* (cit. on p. 49).

[Sch80] Jacob T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *J. ACM* 27.4 (1980), pp. 701–717. DOI: 10.1145/322217.322225 (cit. on p. 52).

[ST21] Amit Sinhababu and Thomas Thierauf. "Factorization of Polynomials Given by Arithmetic Branching Programs". In: *Computational Complexity* 30.15 (2021). DOI: 10.1007/s00037-021-00215-0 (cit. on p. 49).

[Str73] Volker Strassen. "Vermeidung von Divisionen". In: *J. Reine Angew. Math.* 264 (1973), pp. 184–202. ISSN: 0075-4102 (cit. on p. 13).

[SW01] Amir Shpilka and Avi Wigderson. "Depth-3 arithmetic circuits over fields of characteristic zero". In: *Computational Complexity* 10.1 (2001), pp. 1–27. ISSN: 1016-3328. DOI: 10.1007/PL00001609 (cit. on pp. 7, 18, 20).

[SY10] Amir Shpilka and Amir Yehudayoff. "Arithmetic Circuits: A survey of recent results and open questions". In: *Foundations and Trends in Theoretical Computer Science* 5.3-4 (2010), pp. 207–388. DOI: 10.1561/0400000039 (cit. on pp. 2, 52).

[Val79] Leslie G. Valiant. "Completeness Classes in Algebra". In: *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC 1979)*. Atlanta, Georgia, USA: Association for Computing Machinery, 1979, pp. 249–261. ISBN: 9781450374385. DOI: 10.1145/800135.804419 (cit. on p. 4).

[VSBR83] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. "Fast parallel computation of polynomials using few processors". In: *SIAM J. Comput.* 12.4 (1983), pp. 641–644. ISSN: 0097-5397. DOI: 10.1137/0212043 (cit. on pp. 3, 10).

[vzGat84] Joachim von zur Gathen. "Parallel Algorithms for Algebraic Problems". In: *SIAM Journal on Computing* 13.4 (1984), pp. 802–824. DOI: 10.1137/0213050 (cit. on pp. 3, 39, 55).

[vzGat86] Joachim von zur Gathen. "Parallel arithmetic computations: A survey". In: *Proceedings of the 11th Internationl Symposium on the Mathematical Foundations of Computer Science (MFCS 1986)*. Ed. by Jozef Gruska, Branislav Rovan, and Juraj Wiedermann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 93–112. ISBN: 978-3-540-39909-4. DOI: 10.1007/BFb0016236 (cit. on pp. 3, 39).

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 3rd ed. Cambridge University Press, 2013 (cit. on pp. 2, 14, 17, 50, 55).

[Zip79] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *Proceedings of the International Symposium on Symbolic and Algebraic Computation, EUROSAM 1979*. 1979, pp. 216–226. DOI: 10.1007/3-540-09519-5_73 (cit. on p. 52).