Foundations and Trends® in
Theoretical Computer Science
Vol. 5, Nos. 3–4 (2009) 207–388
© 2010 A. Shpilka and A. Yehudayoff
DOI: 10.1561/0400000039

Arithmetic Circuits: A Survey of Recent Results and Open Questions

By Amir Shpilka and Amir Yehudayoff

Contents

Foundations and Trends® in
Theoretical Computer Science
Vol. 5, Nos. 3–4 (2009) 207–388
© 2010 A. Shpilka and A. Yehudayoff
DOI: 10.1561/0400000039

Arithmetic Circuits: A Survey of Recent Results and Open Questions

Amir Shpilka[1] **and Amir Yehudayoff**[2]

[1] *Faculty of Computer Science, Technion, Haifa 32000, Israel, shpilka@cs.technion.ac.il*
[2] *Faculty of Mathematics, Technion, Haifa 32000, Israel, amir.yehudayoff@gmail.com*

Abstract

A large class of problems in symbolic computation can be expressed as the task of computing some polynomials; and arithmetic circuits form the most standard model for studying the complexity of such computations. This algebraic model of computation attracted a large amount of research in the last five decades, partially due to its simplicity and elegance. Being a more structured model than Boolean circuits, one could hope that the fundamental problems of theoretical computer science, such as separating P from NP, will be easier to solve for arithmetic circuits. However, in spite of the appearing simplicity and the vast amount of mathematical tools available, no major breakthrough has been seen. In fact, all the fundamental questions are still open for this model as well. Nevertheless, there has been a lot of progress in the area and beautiful results have been found, some in the last few years. As examples we mention the connection between polynomial identity testing and lower bounds of Kabanets and Impagliazzo, the lower bounds of Raz for multilinear formulas, and two new approaches for proving lower bounds: Geometric Complexity Theory and Elusive Functions.

The goal of this monograph is to survey the field of arithmetic circuit complexity, focusing mainly on what we find to be the most interesting and accessible research directions. We aim to cover the main results and techniques, with an emphasis on works from the last two decades. In particular, we discuss the recent lower bounds for multilinear circuits and formulas, the advances in the question of deterministically checking polynomial identities, and the results regarding reconstruction of arithmetic circuits. We do, however, also cover part of the classical works on arithmetic circuits. In order to keep this monograph at a reasonable length, we do not give full proofs of most theorems, but rather try to convey the main ideas behind each proof and demonstrate it, where possible, by proving some special cases.

1 Introduction

Arithmetic circuits are the most natural and standard model for computing polynomials. In this model the inputs are variables $x_1, \ldots, x_n$, and the computation is performed using the arithmetic operations $+, \times$ and may involve constants from a field $\mathbb{F}$. The output of an arithmetic circuit is thus a polynomial (or a set of polynomials) in the input variables. The complexity measures associated with such circuits are size and depth which capture the number of operations and the maximal distance between an input and an output, respectively.

The most fundamental problems in algebraic complexity are related to the complexity of arithmetic circuits: providing efficient algorithms for algebraic problems (e.g., matrix multiplication), proving lower bounds on the size and depth of arithmetic circuits, giving efficient deterministic algorithms for polynomial identity testing, and finding efficient reconstruction algorithms for polynomials computed by arithmetic circuits (the latter problem is sometimes referred to as learning arithmetic circuits or interpolating arithmetic circuits).

In the past 50 years, we have seen a flurry of beautiful and efficient algorithms for algebraic problems. For example, Cooley and Tukey's algorithm for the Discrete Fourier Transform [38], Strassen's algorithm and those following it for Matrix Multiplication [39, 131] (see [30] for a detailed survey of algorithms for matrix multiplication), algorithms for factoring polynomials (see [72, 146, 147] for surveys of results in this area), and Csanky's algorithm for parallel computation of determinant as well as all other linear algebra problems [40]. In this survey we shall not give details of these algorithms, but rather focus on complexity questions related to arithmetic circuits, mainly on the problem of proving lower bounds for arithmetic circuits and the question of deterministically deciding polynomial identities.

Arithmetic circuits are a highly structured model of computation compared to Boolean circuits. For example, when studying arithmetic circuits we are interested in *syntactic* computation of polynomials, whereas in the study of Boolean circuits we are interested in the *semantics* of the computation. In other words, in the Boolean case we are not interested in any specific polynomial representation of the function but rather we just want to compute some representation of it, while in the arithmetic world we focus on a specific representation of the function. As such, one may hope that the P vs. NP question will be easier to solve in this model. However, in spite of many efforts, we are still far from understanding this fundamental problem. In fact, our understanding of most problems is far from being complete. In particular, we do not have strong lower bounds for arithmetic circuits; We do not know how to deterministically and efficiently determine whether a given arithmetic circuit computes the zero polynomial; and we do not know how to efficiently reconstruct a circuit using only queries to the polynomial it computes. Although seemingly different, these three problems are strongly related to each other, and it is usually the case that a new understanding of one problem sheds light on the other problems as well.

In recent years there has been some progress on these important problems for several interesting classes of arithmetic circuits. In this monograph we aim to describe this recent progress. In particular, we shall cover the new lower bounds on the size of multilinear circuits, the new identity testing algorithms for several restricted classes of circuits and their connection to circuit lower bounds, and the recent reconstruction algorithms for depth-3 arithmetic circuits. We also present many open questions that we view as natural "next step" questions, given our current state of knowledge.

1.1 Basic Definitions

Before any further discussion, we give the basic definitions related to arithmetic circuits.

Definition 1.1 (Arithmetic circuits). An *arithmetic circuit* Φ over the field $\mathbb{F}$ and the set of variables X (usually, $X = \{x_1,\ldots,x_n\}$) is a directed acyclic graph as follows. The vertices of Φ are called *gates*. Every gate in Φ of in-degree 0 is labeled by either a variable from X or a field element from $\mathbb{F}$. Every other gate in Φ is labeled by either $\times$ or $+$ and has in-degree 2. An arithmetic circuit is called a *formula* if it is a directed tree whose edges are directed from the leaves to the root.

Every gate of in-degree 0 is called an *input gate* (even when the gate is labeled by a field element). Every gate of out-degree 0 is called an *output gate*. Every gate labeled by $\times$ is called a *product gate* and every gate labeled by $+$ is called a *sum gate*. The *size* of Φ, denoted $|\Phi|$, is the number of edges in Φ. The depth of a gate v in Φ, denoted depth (v), is the length of the longest directed path reaching v. The depth of Φ is the maximal depth of a gate in Φ. When speaking of bounded depth circuits — circuits whose depth is bounded by a constant independent of $|X|$ — we do not have a restriction on the fan-in. For two gates u and v in Φ, if (u,v) is an edge in Φ, then u is called a *child* of v, and v is called a *parent* of u.

An arithmetic circuit computes a polynomial in a natural way: An input gate labeled by $\alpha \in \mathbb{F} \cup X$ computes the polynomial α. A product gate computes the product of the polynomials computed by its children. A sum gate computes the sum of the polynomials computed by its children.

For a gate v in Φ, define Φ_v to be the sub-circuit of Φ rooted at v. Denote by X_v the set of variables that occur in the circuit Φ_v. We usually denote by f_v the polynomial in $\mathbb{F}[X_v]$ computed by the gate v in Φ. We sometimes abuse notation and denote by Φ_v the polynomial computed by v as well. Define the *degree* of a gate v, denoted $\deg(v)$, to be the total degree of the polynomial f_v (e.g., the total degree of $x_1^2x_2 + x_1 + 1$ is three, whereas the individual degrees are at most two). The *degree* of Φ is the maximal degree of a gate in Φ.

It is clear that every polynomial $f \in \mathbb{F}[X]$ can be computed by an arithmetic circuit and by an arithmetic formula. The main question is how many gates are needed for the computation.

The definition above shows an evident difference between arithmetic circuits and Boolean circuits. While Boolean circuits can perform operations on the "bit representation" of the input field elements, that are not necessarily the arithmetic operations, arithmetic circuits cannot. Nevertheless, most algorithms for algebraic problems fit naturally into the framework of arithmetic circuits.

One last thing to note is that we always regard an arithmetic circuit as computing a polynomial in $\mathbb{F}[X]$ and not a function from $\mathbb{F}^{|X|}$ to $\mathbb{F}$. In general, every polynomial defines a unique function, but a function can usually be expressed as a polynomial in many ways. For example, the polynomial $x^2 - x$ is not the zero polynomial as it has nonzero coefficients. However, over the field with two elements, $\mathbb{F}_2$, it computes the zero function. This distinction is especially important when studying the identity testing problem. This is another difference between the Boolean world and the arithmetic world.

Remark 1.1. For the rest of the survey, unless otherwise stated, the results hold for arbitrary fields. In most cases, for simplicity of discussion and notation, we do not explicitly state the dependence on the field. In general, the question of which field we are working over is important and can make a difference, both from a theoretical point of view and from a practical point of view. The main examples of fields that the reader should bear in mind are prime fields and the real numbers.

1.2 Arithmetic Complexity

Arithmetic complexity classes were first defined in the seminal works of Valiant [138, 141]. Valiant gave analogous definitions for the classes P and NP in the algebraic world, and showed complete problems for these classes. We now give a very brief overview of these classes and state the main known results. As this material was covered in many places, we do not give any proofs here. For a more detailed treatment and proofs, we refer the interested reader to Refs. [29, 30, 61].

We begin by defining the class VP, the algebraic analog of the class P. Originally, Valiant called this class the class of p-bounded polynomials (computed by "polynomially bounded" circuits), but nowadays the notation VP is used (where V is an acronym for Valiant).

Definition 1.2. A family of polynomials $\{f_n\}$ over $\mathbb{F}$ is *p-bounded* if there exists some polynomial $t : \mathbb{N} \to \mathbb{N}$ such that for every n, both the number of variables in f_n and the degree of f_n are at most $t(n)$, and there is an arithmetic circuit of size at most $t(n)$ computing f_n. The class $\mathsf{VP}_{\mathbb{F}}$ consists of all p-bounded families over $\mathbb{F}$.

The polynomial $f_n(x) = x^{2^n}$, for example, can be computed by size $O(n)$ circuits, but it is not in VP as its degree is not polynomial. One motivation for this degree restriction comes from computation over, say, the rational numbers: if the degree is too high then we cannot efficiently represent the value of the polynomial on a given input by a "standard" Boolean circuit. Also note that in the definition we do not require the circuit computing f_n to have a polynomial degree, but, as we shall later see, this property holds without loss of generality (see Theorem 2.2 below).

An interesting family in VP is the family of determinants,

$$\mathrm{DET}_n(X) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_{i=1}^{n} x_{i,\sigma(i)},$$

where $X = (x_{i,j})$ is an $n \times n$ matrix, S_n is the set of permutations of n elements and $\mathrm{sgn}(\sigma)$ is the signature of the permutation σ. It is a nice exercise to find a polynomial size arithmetic circuit for DET_n that does not use divisions.

Remark 1.2. For the rest of the survey we sometimes say polynomial and mean a family of polynomials, e.g., when we talk of the determinant polynomial we actually talk about the family of determinant polynomials.

We now define VNP, the algebraic analog of the class NP.

Definition 1.3. A family of polynomials $\{f_n\}$ over $\mathbb{F}$ is *p-definable* if there exist two polynomially bounded functions $t, k : \mathbb{N} \to \mathbb{N}$ and a family $\{g_n\}$ in $\mathsf{VP}_{\mathbb{F}}$ such that for every n,

$$f_n(x_1, \ldots, x_{k(n)}) = \sum_{w \in \{0,1\}^{t(n)}} g_{t(n)}(x_1, \ldots, x_{k(n)}, w_1, \ldots, w_{t(n)}).$$

The class $\mathsf{VNP}_{\mathbb{F}}$ consists of all p-definable families over $\mathbb{F}$.

Roughly speaking, VNP is the class of polynomials f so that given a monomial, one can efficiently compute the coefficient of this monomial in f (this does not follow immediately from the definition, for more details see, e.g., Refs. [61, 141]). To better understand the connection to NP, one can think of the variables $w = (w_1, \ldots, w_{t(n)})$ as the "witness," and so summing over all witnesses is the arithmetic analog of searching for a witness in NP. The existential quantifier in the definition of NP is translated to the algebraic operation of addition. In some sense, this makes VNP a version of $\#\mathsf{P}$ as well. The canonical example for a family in VNP is the family of permanents of $n \times n$ matrices

$$\mathrm{PERM}_n(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}. \tag{1.1}$$

One way to see that permanent is in VNP is by Ryser's formula that also gives the smallest known circuit computing permanent (which is also a depth-3 circuit).

Fact 1.1 ([114]). For every $n \in \mathbb{N}$, $\mathrm{PERM}_n(X) = \sum_{T \subseteq [n]} (-1)^{n-|T|} \prod_{i=1}^{n} \sum_{j \in T} x_{i,j}$.

It follows by definition that $\mathsf{VP} \subseteq \mathsf{VNP}$. Valiant's hypothesis says that VP is a strict subclass of VNP.

Valiant's hypothesis I: $\mathsf{VP} \neq \mathsf{VNP}$.

As arithmetic circuits are more structured than Boolean circuits, one could hope that proving Valiant's hypothesis should be easier than its Boolean counterpart. A very weak version of this statement was proved in Ref. [61], where it was shown that in a non-associative world, where variables are not assumed to satisfy the identity $(xy)z = x(yz)$, **Valiant's hypothesis I** holds. This non-associative statement is an evidence for the obvious: the more structured the world is, the easier it is to prove lower bounds. Specifically, in a non-associative world algorithms cannot exploit "symmetries" that follow from associativity. Indeed, in such a world it is more difficult to design algorithms and lower bounds are easier to prove.

Besides defining the classes VP and VNP, Valiant also gave complete problems for these classes. He described a reduction between families of polynomials and gave complete families with respect to it.

Definition 1.4. A polynomial $f(x_1,\ldots,x_n)$ over $\mathbb{F}$ is called a *projection* of a polynomial $g(y_1,\ldots,y_m)$ over $\mathbb{F}$ if there exists an assignment $\rho \in (\{x_1,\ldots,x_n\} \cup \mathbb{F})^m$ such that $f(x_1,\ldots,x_n) \equiv g(\rho_1,\ldots,\rho_m)$. In other words, f can be derived from g by a simple substitution. This definition can be extended to projections between families of polynomials. The family $\{f_n\}$ is a *p-projection* of the family $\{g_n\}$ if there exists a polynomially bounded $t : \mathbb{N} \to \mathbb{N}$ such that for every n, f_n is a projection of $g_{t(n)}$.

Both VP and VNP are closed under projections, e.g., if f is in VP then any projection of f is also in VP. Valiant showed that permanent is complete for the class VNP.

Theorem 1.1 ([138]). For any field $\mathbb{F}$ such that $\mathrm{char}(\mathbb{F}) \neq 2$, the family $\{\mathrm{PERM}_n\}$ is VNP-complete. Namely, any family in VNP is a p-projection of it.

Valiant's hypothesis I is thus equivalent to proving a super-polynomial lower bound on the size of circuits computing the permanent. We note that a stronger version of Theorem 1.1 was proved in

Ref. [61], where it was shown that permanent is VNP-complete even in a very weak computational world where the variables are not assumed to be commutative nor associative.

Valiant also showed that determinant is VP-complete with respect to *quasi-polynomial* projections.

Theorem 1.2 ([138]). The family $\{\mathrm{DET}_n\}$ is VP-complete with respect to quasi-polynomial projections. That is, for any family $\{f_n\}$ in VP there exists a function $t : \mathbb{N} \to \mathbb{N}$ satisfying[1] $t(n) = n^{O(\log n)}$ such that f_n is a projection of $\mathrm{DET}_{t(n)}$. In fact, if we change the definition of VP to VQP by replacing polynomial by quasi-polynomial (i.e., $2^{\mathrm{polylog}(n)}$), then determinant is VQP-complete.

This theorem follows immediately from the next two theorems that show that arithmetic circuits are "shallow," and that determinant can "simulate" small formulas.

Theorem 1.3 ([143]). Let f be a degree r polynomial computed by a size s circuit. Then f can be computed by a circuit of size $\mathrm{poly}(r, s)$ and depth $O(\log r(\log r + \log s))$.

Theorem 1.3 was proved in a seminal work of Valiant et al. [143]. It is commonly rephrased as $\mathsf{VP} = \mathsf{VNC}^2$, where VNC^k denotes polynomial size and polynomial degree arithmetic circuits of depth $O(\log^k n)$. Clearly, $\mathsf{VNC}^1 \subseteq \mathsf{VNC}^2 \subseteq \ldots \subseteq \mathsf{VP}$, and Theorem 1.3 shows that in fact the chain halts after two steps. Since determinant is in VP, Theorem 1.3 implies that determinant has a formula of quasi-polynomial size (more generally, every polynomial in VNC^2 has a formula of quasi-polynomial size).

Theorem 1.4([138]). For any polynomial f in $\mathbb{F}[X]$ that can be computed by a formula of size s over $\mathbb{F}$, there is a matrix A of dimensions $(s+1) \times (s+1)$ whose entries are in $X \cup \mathbb{F}$ such that $\mathrm{DET}(A) = f$.

---

[1] Unless stated otherwise, logarithms are in base two.

As determinant is complete for VQP, an algebraic analog of the P vs. NP question is the question of "embedding" permanent in determinant.

Valiant's hypothesis II: VNP $\not\subseteq$ VQP.

This hypothesis is also known as *Valiant's extended hypothesis.* Stated differently, the hypothesis is that the permanent does not belong to VQP. Thus, in order to prove Valiant's extended hypothesis it suffices to prove that one cannot represent PERM_n as the determinant of a matrix of dimension quasi-polynomial in n. Currently, the best lower bounds on the dimension of such a matrix are given by the following theorem of [31, 92].

Theorem 1.5 ([31, 92]). Let $\mathbb{F}$ be a field of characteristic different than two and let $X = (x_{i,j})_{i,j \in [n]}$ be a matrix of variables. Then, any matrix A whose entries are linear functions in $\{x_{i,j}\}_{i,j \in [n]}$ over $\mathbb{F}$ such that $\mathrm{DET}(A) = \mathrm{PERM}_n(X)$ must be of dimension at least $n^2/2$.

Here is a rough sketch of the idea behind Mignon and Ressayre's proof of Theorem 1.5. Compute the rank of the Hessian matrix, i.e., the matrix of second partial derivatives, of both $\mathrm{PERM}_n(X)$ and $\mathrm{DET}(A)$. This rank for $\mathrm{PERM}_n(X)$ is at least (roughly) n^2, whereas for $\mathrm{DET}(A)$ this rank is of order D, where D is the dimension of A.

Valiant's extended hypothesis gives a way for reformulating a question about circuits as a purely algebraic question: the VQP vs. VNP problem is equivalent to the problem of embedding the permanent inside the determinant. One advantage of this formulation is that the combinatorial structure of circuits does not appear in it.

Open Problem 1. Improve the lower bound on the dimension of a matrix A with entries that are linear functions in $\{x_{i,j}\}_{i,j \in [n]}$ such that $\mathrm{DET}(A) = \mathrm{PERM}_n(X)$.

1.3 Arithmetic Circuit Classes

In addition to the general model of arithmetic circuits, introduced in Section 1.1, we will be considering several other, more restricted, classes of arithmetic circuits. In particular, we will be interested in bounded depth arithmetic circuits, and even more specifically in depth-3 and depth-4 circuits, in multilinear circuits, noncommutative circuits and more. We shall now define some of these classes and discuss their importance.

The model of bounded depth circuits was already defined in Section 1.1. Two important subclasses of bounded depth circuits that we shall focus on in this monograph are depth-3 circuits, also known as $\Sigma\Pi\Sigma$ circuits and depth-4 circuits known as $\Sigma\Pi\Sigma\Pi$ circuits. A $\Sigma\Pi\Sigma$ circuit is a depth-3 circuit with an addition gate at the top, a middle layer of multiplication gates, and then a level of addition gates at the bottom. A $\Sigma\Pi\Sigma$ circuit with s multiplication gates compute polynomials of the form $\sum_{i=1}^{s}\prod_{j=1}^{d_i}\ell_{i,j}(x_1,\ldots,x_n)$, where the $\ell_{i,j}$'s are linear functions. Although a very restricted model this is the first class for which we do not have any strong lower bounds, over fields of characteristic zero (see Section 3.5). Moreover, in Section 3.8.2 we discuss a result of Raz [105] showing that strong lower bounds for (a restricted subclass of) $\Sigma\Pi\Sigma$ circuits imply super-polynomial lower bound on the formula complexity of permanent.

Similar to depth-3 circuits, a $\Sigma\Pi\Sigma\Pi$ circuit is composed of four alternating layers of addition and multiplication gates. Thus, a size s $\Sigma\Pi\Sigma\Pi$ circuit computes a polynomial of the form $\sum_{i=1}^{s}\prod_{j=1}^{d_i}f_{i,j}(x_1,\ldots,x_n)$, where the $f_{i,j}$'s are polynomials of degree at most s having at most s monomials (i.e., they are s-sparse polynomials). The importance of $\Sigma\Pi\Sigma\Pi$ circuits stems for two main reasons. Depth-4 is the first depth for which we do not have strong lower bounds for any field of characteristic different than 2 (over $\mathbb{F}_2$ lower bounds follow from the results of Razborov and Smolensky [112, 130]). The best known lower bounds, due to Raz [103], are smaller than n^2 (see Section 3.5). Another important reason is that, with respect to proving exponential lower bounds, $\Sigma\Pi\Sigma\Pi$ circuits are as interesting as general arithmetic circuits. Namely, an n-variate degree n polynomial can be computed by a sub-exponential arithmetic circuit if and only if it can be computed by a sub-exponential $\Sigma\Pi\Sigma\Pi$ circuit. This result, due to Agrawal and Vinay [5], is discussed in Section 2.4. Furthermore, derandomizing the polynomial identity testing problem for such circuits is almost equivalent to derandomizing it for general arithmetic circuits.

Thus, in order to understand the main open problems in arithmetic circuit complexity, one can focus on depth-4 circuits, instead of general arithmetic circuits, without loss of generality.

Another important model that we discuss in this monograph is *multilinear* circuits. A polynomial $f \in \mathbb{F}[X]$ is called *multilinear* if the individual degree of each variable in f is at most one. An arithmetic circuit Φ is called *multilinear* if every gate in Φ computes a multilinear polynomial. An arithmetic circuit Φ is called *syntactically multilinear* if for every product gate $v = v_1 \times v_2$ in Φ, the two sets X_{v_1} and X_{v_2} are disjoint (recall that X_u is the set of variables that occur in the circuit Φ_u). Syntactically multilinear circuits are clearly multilinear but the other direction is not true in general.

While being a very restricted model of computation, multilinear circuits and formulas form a very interesting class as for many multilinear polynomials, e.g., permanent and iterated matrix multiplication, the currently best arithmetic circuits computing them are multilinear. Indeed, computing a multilinear polynomial with a circuit that is not multilinear requires some "non-intuitive" cancellations of monomials. We do not however, that such "clever" cancellations occur, e.g., in small arithmetic circuits computing the determinant. In particular, we do not know today of polynomial size multilinear circuits computing the determinant. Being a natural model for computing multilinear polynomials, multilinear circuits are an interesting and an important class of circuits and we discuss the best results known for them.

In addition to bounded depth circuits and multilinear circuits we shall also study monotone circuits, noncommutative circuits, circuits with bounded coefficients and read-once formulas. We shall give the relevant definitions when we first discuss each of these classes.

1.4 Road Map

Here is a short overview of the content of this survey.

1.4.1 Structural Results

Due to its algebraic nature, the model of arithmetic circuits is more structured than the model of Boolean circuits. As such, we are able to prove results in the arithmetic world that in the Boolean case are still open. In Section 2 we discuss some of the works on the structure of arithmetic circuits. These structural properties of arithmetic circuits are also used as starting points to proving lower bounds. We now discuss three examples of such structural results and their connection to lower bounds.

A striking result due to [143] is that in the arithmetic world $\mathsf{VP} = \mathsf{VNC}^2$ (see Theorem 1.3). This is in contrast to the Boolean world, where it is conjectured that $\mathsf{P} \neq \mathsf{NC}$. Subsequently, Agrawal and Vinay [5] proved a depth-4 version of this statement, showing that in order to prove exponential lower bounds on the size of general arithmetic circuits one just needs to prove exponential lower bounds on the size of depth-4 circuits.

A surprising result due to Baur and Strassen [16], that strongly relies on the underlying algebraic structure, states that computing a polynomial $f(x_1,\ldots,x_n)$ is essentially equivalent to simultaneously computing f and all of its n partial derivatives $\frac{\partial f}{\partial x_1},\ldots,\frac{\partial f}{\partial x_n}$. Thus, proving a lower bound on the size of a circuit computing a set of polynomials is as difficult as proving a lower bound for a single polynomial. Alternatively, perhaps more optimistically, proving lower bounds should not be so difficult, as instead of proving a lower bound on the computation of a single polynomial we can try and prove a lower bound for circuits computing many polynomials. This principle actually turned out to be useful in at least two cases: showing that divisions are not necessary in computing polynomials by general arithmetic circuits [133] and proving lower bounds for multilinear circuits [107].

An interesting fact is that arithmetic circuits computing homogeneous polynomials can be transformed to be homogeneous, with only a small overhead. Recently, Raz [105] proved that for formulas, this transformation can be done at a smaller cost than what was known before. In particular, Raz showed that if one can prove (very) strong lower bounds on tensor rank then one obtains super-polynomial lower bounds on formula-size. Since tensor rank is no other than the size of the smallest set-multilinear depth-3 circuit computing the "tensor," Raz's result says that a very strong lower bound for the (very restricted) model of set-multilinear depth-3 circuits implies a lower bound for general formulas.

1.4.2 Lower Bounds for Arithmetic Circuits

One of the biggest challenges of algebraic complexity is proving lower bounds on circuit-size. Unlike the case of Boolean circuits, super-linear lower bounds on the size of general arithmetic circuits are known [16, 133]. Contrarily, no strong lower bounds are known for bounded depth arithmetic circuits. In particular, no super-quadratic lower bound is known even for circuits of depth 4, when $\mathrm{char}(\mathbb{F}) \neq 2$.

In Section 3 we survey the known lower bounds and discuss some proofs in more detail. In particular, we explain Strassen's degree bound that gives a super-linear lower bound for general circuits [133] and Kalorkoti's quadratic lower bound on the size of general formulas [70]. We discuss the lower bounds for the size of bounded depth circuits [122, 103]. We then consider in detail depth-3 circuits, which is the "first" model for which proving lower bounds seems to be a difficult task.

In this section we also explain the following two-step "technique" for proving lower bound for arithmetic circuits. The first step is based on the fact that polynomials computed by small arithmetic circuits can be presented as a sum of a small number of products of "simpler" polynomials (this is one of the structural theorems that we prove). The second step is using the so-called partial derivative method to bound the complexity of such polynomials. By applying these two steps, we derive lower bounds for various classes of arithmetic circuits, such as monotone arithmetic circuits [68, 109, 121, 135] and multilinear formulas [102, 104, 108].

Finally, we present several approaches for proving lower bounds on circuit-size, and discuss the possibility of generalizing the *Natural Proofs* approach of Razborov and Rudich [111] to the algebraic setting.

1.4.3 Polynomial Identity Testing

Polynomial identity testing (PIT) is the problem of deciding whether a given arithmetic circuit computes the identically zero polynomial. Many randomized algorithms are known for this problem yet its deterministic complexity is still far from understood. Recently, it was discovered that this problem is strongly related to the question of proving lower bounds [69].

In Section 4 we first survey and sketch the proofs of randomized algorithms for PIT. We then discuss the relation between lower bounds and derandomization of PIT algorithms. One of the surprising results in this context is that a deterministic (black-box) polynomial-time algorithms for PIT of depth-4 arithmetic circuits implies a (quasi-polynomial time) derandomization of the problem for general arithmetic circuits.

We then present several deterministic algorithms for restricted classes of arithmetic circuits. We do not cover all known algorithms but rather present what we view as the most notable techniques in the area. Specifically, we give one of the many algorithms for sparse polynomials [86]. We show a polynomial-time algorithm for PIT of noncommutative formulas that is based on the partial derivative method [106]. We then describe two algorithms for depth-3 circuits with a bounded top fan-in. The first is the local ring algorithm of [81] that works in the non-black-box model (which we refer to as the *white-box model*) and the second is the algorithm of [43, 76] that is based on the *rank* method (with the strengthening of [80, 117, 118]). After that, we present two results for depth-4 circuits. The first is by [116] that gave a polynomial time PIT for the so-called *diagonal* circuits, based on the ideas of [106]. The second result is by [75] that gave a PIT algorithm for depth-4 multilinear circuits with bounded top fan-in, based on ideas from [76] and [127]. Finally, we present the algorithm of [126, 127] for identity testing of sums of read-once formulas that strengthen some of the results for depth-3 circuits and that influenced [75].

1.4.4 Reconstruction of Arithmetic Circuits

In Section 5 we consider the problem of reconstructing arithmetic circuits, which is the algebraic analog of the learning problem of Boolean circuits. This problem is clearly related to PIT, as an identity testing algorithm for a circuit class gives a way of distinguishing between different circuits from that class and can thus be helpful in designing a learning algorithm.

We discuss the similarities and differences between the reconstruction problem and analogous problems in the Boolean world. We then give some hardness results on the reconstruction problem. After that we discuss several known reconstruction algorithms. First, we explain how to reconstruct sparse polynomials. Then we discuss the multiplicity automata technique of [17] and its extension for arithmetic circuits [85]. Basically, this technique can be thought of as learning via partial derivatives. At the end, we move to depth-3 circuits with a bounded top fan-in and sketch the algorithms of [77, 125] that are based on ideas from the identity testing algorithm of [43, 77].

1.5 Additional Reading

We decided to focus this survey on recent results in arithmetic circuit complexity, mainly on lower bounds and identity testing algorithms, and so many beautiful results in algebraic complexity, both new and old, were left out. We now mention some of the topics that are not discussed in this monograph and give references to relevant papers. Most of these topics are discussed in the comprehensive book [30] and the (unfortunately, still relevant) survey of Strassen [134].

One important area that we do not cover is algorithms for algebraic problems, an area that has been yielding many beautiful works. A partial list of algorithms include Cooley and Tukey's FFT algorithm [38], fast matrix multiplication [39] (and the new algorithmic approach of [36, 37]), efficient polynomial factorization (see the surveys [72, 146] and the recent [84]) and the deterministic primality testing algorithm of [4].

Another topic that we do not really discuss is that of linear and bilinear complexity. Here, one is interested in the complexity of computing linear transformations and bilinear forms using linear or bilinear circuits, respectively. The complexity of computing univariate polynomials is another topic that we decided not to include. The interested reader is referred to the aforementioned book [30] and survey [134].

Several other models of algebraic computations also received a lot of attention. Among them we mention the Blum–Shub–Smale model of computing over the reals and algebraic decision trees, more information can be found in [9, 21, 30].

2 Structural Results

Arithmetic circuits are very structured compared to Boolean circuits. For example, in the algebraic world we have useful notions such as degree, homogeneity, and partial derivatives that do not have "nice" counterparts in the Boolean world. In this section, we discuss some results that describe deep understanding regarding the structure of arithmetic circuits that were obtained in the last four decades. Some of these results are fairly basic, like transforming a circuit into a homogeneous one. But others, such as the seminal work [143] showing that in the arithmetic world $\mathsf{P} = \mathsf{NC}^2$, are more sophisticated. We will try and give the main ideas underlying each of the results without necessarily giving full proofs.

We begin this chapter by proving a basic result: the existence of universal arithmetic circuits (Theorem 2.1). This result plays an important role in the Elusive functions approach of Raz [103] that we present in Section 3.8.3 (although, for simplicity, we present there a specific instance that does not rely on Theorem 2.1). We then discuss Homogenization (Theorems 2.2 and 2.3) and multilinearization of arithmetic circuits. After that we move to discussing the important result of Baur and Strassen on computing partial derivatives (Theorem 2.5). Following that we describe results on depth reduction (Theorems 2.7, 2.8, and 2.11), and prove Theorem 1.3. We end this chapter by explaining how to cope with division gates (Theorems 2.13 and 2.14).

2.1 Universal Circuits

We start with a basic result of Raz [103] showing the existence of a universal arithmetic circuit.

Recall that a polynomial f is *homogeneous* if all of its monomials have the same degree. Given a polynomial f, we denote by $H_i(f)$ its homogeneous part of degree i. Namely, all monomials of degree exactly i appearing in f. Clearly, $f = \sum_{i=0}^{r} H_i[f]$, where r is the degree of f. We say that a circuit is *homogeneous* if each of its gates computes a homogeneous polynomial.

Definition 2.1 (Universal circuits). A circuit Φ is called *universal* for n inputs and n outputs circuits of size s, that compute homogenous polynomials of degree r, if the following holds: For every n homogeneous polynomials $f_1(x_1,\ldots,x_n),\ldots,f_n(x_1,\ldots,x_n)$ of degree r, that can be simultaneously computed by a circuit of size s, there exists a circuit Ψ computing $f_1,\ldots,f_n$ as well, such that the computation graph of Ψ is the same as the graph of Φ.

In other words, a circuit is universal if for any circuit Ψ of size s, there is an appropriate labeling of the inputs of Φ such that the resulting circuit computes the same polynomials as Ψ. Alternatively, every circuit Ψ is a "projection" of the universal circuit.

The universal circuit that we shall describe has the following additional structure, which gives some information on "how do arithmetic circuits compute polynomials." Let Φ be a homogenous circuit. We say that Φ is in a normal-homogenous-form if it satisfies the following properties: 1. All input gates are labeled by a variable, specifically, no input gate is labeled by a field element. 2. All edges leaving input gates are connected to sum gates. 3. All output gates are sum gates. 4. Gates are alternating, namely, if v is a product gate and u is a child of v then u is a sum gate, and vice versa. 5. The fan-in of every product gate is exactly two (we do not restrict the fan-in of sum gates). 6. The out-degree of every addition gate is at most one.

The following theorem tells us that we can efficiently construct a universal circuit.

Theorem 2.1 ([103]). For any nonzero integers $s \geq n$ and r, we can construct in time poly(s,r) a circuit Φ in a normal-homogenous-form with at most $O(r^4 s)$ nodes that is universal for n inputs and n outputs circuits of size s that compute homogenous polynomials of degree r.

Proof. [Sketch] The proof is in two steps. First we show that for every circuit Ψ there exists a circuit in normal-homogeneous-form computing the same polynomial. More accurately, Ψ will be a "projection" of the circuit that we will find. Then we show that how to construct a universal circuit by basically demonstrating a circuit that contains as "sub-circuits" all possible normal-homogeneous-form circuits (with the appropriate set of parameters).

Assume that we are given a circuit Ψ, computing a polynomial f, that we wish to "embed" in the universal circuit. Theorem 2.2 below shows that there exists a homogeneous circuit, of size $O(r^2 s)$, computing $H_0(f),\ldots,H_r(f)$ simultaneously. While this circuit is not necessarily in normal-homogeneous-form, we can make it such by a few simple changes: Condition 1 is easily satisfied by labeling input gates by new variables when needed (thus, f is a obtained by substituting values to some variables of the new circuit). Conditions 2, 3, and 6 are satisfied by adding a few sum gates when needed. Condition 4 is satisfied by adding product gates when needed. Condition 5 is satisfied as we do not change the fan-in of product gates.

It remains to show that the graph we obtained can be embedded in a universal graph. The proof of Theorem 2.2 gives a bound on the number of sum and product gates in the circuit that we obtained at the end of the first step. Assume that there were s_+ sum gates and $s_\times$ product gates. Consider a circuit consisting of $2r$ alternating levels of sum and product gates containing s_+ and $s_\times$ gates each, respectively. Now, for every possible edge between a sum gate and a product gate we put a little gadget, involving a new "help-variable," such that by setting the help-variable to 0 we essentially "erase" the edge and by setting it to 1 we keep the edge. It is not difficult to see that such a gadget is easy to construct and therefore we basically cover all possible circuits of normal-homogeneous-form (of a certain size and degree) in this way. It is clear that the circuit that was obtained in the first step can be computed by making the appropriate assignment to the variables of this circuit. □

2.2 Homogenization

We now discuss a simple and useful property of algebraic computation. Namely, that we can decompose any computation to its homogeneous parts without increasing the size by too much. This useful observation is implicit in Ref. [133].

Theorem 2.2 ([133]). If f has an arithmetic circuit Φ of size s, then for every $r \in \mathbb{N}$, there is a homogeneous circuit Ψ of size at most $O(r^2 s)$ computing $H_0[f], H_1[f], \ldots, H_r[f]$.

This transformation allows us to assume without loss of generality that circuits for families in VP have polynomial degrees as well, i.e., that all their intermediate computations are also low degree polynomials.

Proof. [Sketch] We describe how to construct Ψ. For every gate v in Φ, we define $r+1$ gates in Ψ, which we denote $(v,0),\ldots,(v,r)$, in such a way that (v,i) computes $H_i(\Phi_v)$. We construct Ψ inductively as follows. If v is an input gate, we can clearly define (v,i) as an input gate with the appropriate properties. If $\Phi_v = \Phi_u + \Phi_w$, define $\Psi_{(v,i)} = \Psi_{(u,i)} + \Psi_{(w,i)}$ for all i. If $\Phi_v = \Phi_u \times \Phi_w$, define $\Psi_{(v,i)} = \sum_{j=0}^{i} \Psi_{(u,j)} \times \Psi_{(w,i-j)}$. Induction implies that Ψ has the claimed functionality. Every gate in Φ corresponds to at most $O((r+1)^2)$ gates in Ψ (each product gate requires $O((r+1)^2)$ additional sum gates), and so $|\Psi| = O(r^2 s)$. □

The process described above takes a general circuit and transforms it into a homogeneous one with the same functionality while not increasing the size by much. However, as described this process may increase the depth by a factor that is logarithmic in the degree (due to the loss in each product gate). In the case of formulas, an increase in depth is translated to the exponent. In particular, it may transform a general formula of size s to a homogeneous formula of size $s^{O(\log r)}$. Thus, this procedure is not efficient when working with formulas. Recently, Raz [105] showed that, for a certain range of parameters, one can obtain an efficient construction for formulas as well. For example, it follows from Raz's work that a polynomial size formula computing a homogeneous polynomial of degree $O(\log n)$ can be assumed to be homogeneous, without loss of generality.

Theorem 2.3([105]). Let f be a homogeneous polynomial of degree r that can be computed by an arithmetic formula of size s. Then, f can be computed by a homogeneous formula of size $\mathrm{poly}(s)\binom{r+O(\log s)}{r}$.

Proof. [Sketch] Let Φ be a formula of size s computing f. Theorem 2.7 below, which is a simple observation, tells us that we can assume, without loss of generality, that the depth of Φ is at most $O(\log s)$. This may cause a polynomial increase in size.

The homogeneous formulas that we shall construct will have depth $O(\log s \cdot \log r)$, but nevertheless their sizes will be much smaller than $2^{O(\log s \cdot \log r)}$. The basic construction is similar to the one in Theorem 2.2 and the main difference is that we count the number of times each gate is "used" in a more accurate way.

For every integer d, denote by $\mathcal{G}_d$ the family of monotone non-increasing functions χ from $\{0,1,\ldots,d\}$ to $\{0,1,\ldots,r\}$. I.e., all functions χ satisfying $\chi(i+1) \le \chi(i)$ for every $i \in \{0,\ldots,d-1\}$. Clearly, the size of $\mathcal{G}_d$ is $\binom{r+d+1}{r}$.

For every gate v in Φ, define the *product-height* of v, denoted $\mathrm{ph}(v)$, to be the number of product gates on the path from v to the output gate. Here is how we transform Φ to a homogeneous formula Ψ: Every gate v in Φ with $\mathrm{ph}(v) = d$ will be duplicated to $|\mathcal{G}_d|$ gates in Ψ labeled (v,χ), for every $\chi \in \mathcal{G}_d$, so that $\Psi_{(v,\chi)}$ computes $H_{\chi(d)}(\Phi_v)$, the $\chi(d)$-th homogeneous part of the polynomial v computes in Φ. We construct Ψ by induction. When v is an input gate it is clear how to label

(v,χ) for all χ. When $v = v_1 + v_2$, then $\mathrm{ph}(v) = \mathrm{ph}(v_1) = \mathrm{ph}(v_2)$, and we define $(v,\chi) = (v_1,\chi) + (v_2,\chi)$ for all χ. Finally, when $v = v_1 \times v_2$, we do the following. Consider some $\chi \in \mathcal{G}_d$ where $d = \mathrm{ph}(v)$. For every $\ell \in \{0,\ldots,\chi(d)\}$, define $\chi_\ell \in \mathcal{G}_{d+1}$ as

$$\chi_\ell(i) = \begin{cases} \chi(i) & \text{if } i \in [d] \\ \ell & \text{if } i = d+1 \end{cases}.$$

Set

$$(v,\chi) = \sum_{\ell=0}^{\chi(d)} (v_1,\chi_\ell) \times (v_2,\chi_{\chi(d)-\ell}).$$

Induction implies that Ψ is homogeneous with the claimed functionality. The way we construct Ψ also implies that it is a formula. The crucial point is that when $v = v_1 \times v_2$, the function χ_ℓ tells us for which value of χ we "used" the nodes $(v_1,\chi_\ell),(v_2,\chi_\ell)$. The size of Ψ is as claimed, since the number of times we duplicate each vertex is at most $\binom{r+\mathrm{depth}(\Phi)+1}{r} \leq \binom{r+O(\log s)}{r}$. □

We do not know whether computing a polynomial by a homogeneous formula, when the degree is not too small, can cause a severe blow-up in size.

Open Problem 2. Are homogeneous formulas super-polynomially weaker than general formulas?

In Ref. [63] it was shown that for multilinear formulas (i.e., formulas that each of their gates computes a multilinear polynomial, see Section 2.2.1) homogeneity is indeed a weakness. Namely, in some cases, homogeneous multilinear formulas are super-polynomially weaker than multilinear formulas.

2.2.1 Multilinearization

As with homogenization we can study the task of transforming a circuit computing a multilinear polynomial into a multilinear circuit. In contrast to homogenization, the most efficient way known today for transforming a general circuit to a multilinear one causes an exponential blow-up in size [100]. In spite of this, we do not have strong lowers bound for multilinear computation (see Section 3.6). Thus, we do not know whether this loss in size is necessary or not. Specifically, the following question is open.

Open Problem 3. Are multilinear circuits super-polynomially weaker than general circuits?

Furthermore, given a multilinear circuit it is very natural to try and transform it into a syntactically multilinear circuit. However, the following question is still open.

Open Problem 4. Are multilinear circuits super-polynomially more powerful than syntactically multilinear circuits?

If instead of circuits we were to consider formulas then the situation becomes much simpler. In particular, syntactically multilinear formulas are equivalent to multilinear formulas [104].

Theorem 2.4 ([104]). If f can be computed by a multilinear formula of size s, then f can also be computed by a syntactically multilinear formula of size s.

Proof. [Sketch] The idea is to go over the formula gate by gate and guarantee that it satisfies the necessary condition. Let $v = v_1 \times v_2$ be a gate in a multilinear formula Φ and assume that $x \in X_{v_1} \cap X_{v_2}$. As Φ is multilinear, the polynomial $\Phi_v = \Phi_{v_1}\Phi_{v_2}$ is multilinear, and so the degree of x in either Φ_{v_1} or Φ_{v_2} is zero. Without loss of generality, assume that the degree of x in Φ_{v_1} is zero. In this case, substitute $x = 0$ in Φ_{v_1}. This does not affect the overall functionality of Φ. □

2.3 Partial Derivatives

We now describe a fairly surprising result by Baur and Strassen [16]. We start by defining the notion of partial derivatives. The *partial derivative* with respect to a variable x is defined as follows: $\partial_x(x) = 1$, and for a polynomial f that does not contain x, $\partial_x(f) = 0$. To define $\partial_x(f)$ for a general polynomial, we use the two identities $\partial_x(f + g) = \partial_x(f) + \partial_x(g)$ and $\partial_x(fg) = \partial_x(f)g + f\partial_x(g)$. It is not difficult to prove that this is a well-defined notion that makes sense over any field. Note however that while the definition is the same for all fields, a partial derivative can behave differently over different fields. For example, over the real numbers $\partial_x(x^2) = 2x$, while over the field with two elements $\partial_x(x^2) = 0$. A useful property of this definition is that it satisfies the chain rule for partial derivatives.

Baur and Strassen showed that computing $f(x_1,\ldots,x_n)$ is as hard as simultaneously computing the $n + 1$ polynomials $f, \partial_{x_1}(f),\ldots,\partial_{x_n}(f)$.

Theorem 2.5 ([16]). Let $f(x_1,\ldots,x_n)$ be a polynomial that has a circuit Φ of size s and depth d. Then, there exists a circuit Ψ of size $O(s)$ and depth $O(d)$ computing (simultaneously) the polynomials $\partial_{x_1}(f),\ldots,\partial_{x_n}(f)$.

This result is very interesting as a priori one might expect that the circuit complexity of $\partial_{x_1}(f),\ldots,\partial_{x_n}(f)$ will be of order $n \cdot s$ rather than $O(s)$, as these are n polynomials that are, intuitively, "as complex" as f is.

To prove the theorem we need to define how to take a partial derivative with respect to a gate of the circuit. We give a slightly more general definition than what the proof requires as it will be useful in Section 2.4. Let v, w be two gates in Φ and denote by f_v and f_w the polynomials computed at v and w, respectively. Let $\Phi_{w=y}$ be the circuit in which we delete the two edges entering w and label it with a new variable y (if w is an input gate then we simply label it with y). Denote by $f_{v,w}(X,y)$ the polynomial computed by v in $\Phi_{w=y}$. Finally, define

$$\partial_w f_v = (\partial_y f_{v,w})|_{y=f_w},$$

namely, the polynomial obtained by substituting f_w to y in the polynomial $\partial_y f_{v,w}$. Clearly, the polynomial $\partial_w f_v$ is defined over the variables X.

Proof. We prove the existence of Ψ by induction on s, computing derivatives from the root down (in fact, we describe how to construct Ψ given Φ). If Φ is an input gate, constructing Ψ is straightforward. Otherwise, let v be the deepest gate in Φ (i.e., the gate that is the farthest from the output gate) and denote its children, which are input gates, by u, w. Consider the circuit $\Phi_{v=y}$, and denote its output polynomial by $f_{v=y}$. As we deleted two edges, $\Phi_{v=y}$ is smaller than Φ. By induction, there exists a circuit Ψ' computing $\partial_{x_1}(f_{v=y}), \ldots, \partial_{x_n}(f_{v=y}), \partial_y(f_{v=y})$ of size $O(s-1)$. Denote by X' the set of variables that label either u or w in Φ (X' could be empty). Note that $f = f_{v=y}\big|_{y=f_v}$, where f_v is the polynomial that v computes in Φ. The chain rule for partial derivatives implies that for every x_i,

$$\partial_{x_i}(f) = \partial_{x_i}(f_{v=y})\big|_{y=f_v} + \partial_y(f_{v=y})\big|_{y=f_v} \cdot \partial_{x_i}(f_v).$$

Therefore, for every $x_i \notin X'$, $\partial_{x_i}(f) = \partial_{x_i}(f_{v=y})\big|_{y=f_v}$. Since $\partial_{x_i}(f_v)$ is either a variable or a field element, and since the size of X' is at most two, we can compute $\{\partial_{x_i}(f)\}_{x_i \in X'}$ by adding at most a constant number of gates and using the gates in Ψ'. The size of Ψ is thus at most $O(s-1) + O(1) = O(s)$. The statement about the depth follows by induction as well. □

Theorem 2.5 shows that computing the first partial derivatives of an arithmetic circuit is an easy task. What about second order partial derivatives? We can define $\partial_{x_i,x_j}(f) = \partial_{x_i}(\partial_{x_j}(f))$ (the order does not matter). It is not known whether one can compute all the order n^2 second partial derivatives with only a constant increase in size. Having such a result will automatically imply an optimal circuit for matrix multiplication: Assume we wish to multiply two $n \times n$ matrices A, B. Consider the polynomial $xABy$ with x, y two n-dimensional vectors. This polynomial has an $O(n^2)$ size circuit as we can compute xA and By and then take their inner product. On the other hand, the second partial derivative of $xABy$ with respect to x_i, y_j is $(AB)_{i,j}$. Therefore, if the answer to the following open problem is positive then we can multiply two matrices in time $O(n^2)$.

Open Problem 5. Does an analog of Theorem 2.5 hold for second-order partial derivatives?

We mention that Raz [101] showed that any bounded-coefficients circuit (i.e., a circuit that uses only, say, real numbers of absolute value at most one) computing the product of two $n \times n$ real matrices must be of size at least $\Omega(n^2 \log n)$ (see Section 3.7). This lower bound rules out the possibility that a transformation like the one in Ref. [16] (namely, a transformation that does not introduce large constants) can yield a constant blow-up in size for computing second-order partial derivatives.

2.4 Depth Reduction

In this section we discuss depth reduction for arithmetic computation. The main idea will be to express the circuit using partial derivatives of intermediate computations. We start by discussing some properties of partial derivatives. In what follows we consider a homogeneous circuit Φ and use the notations introduces in Section 2.3.

Claim 2.6. Let v, w be two gates of a homogeneous circuit Φ. Denote with f_v and f_w the polynomial computes by v and w, respectively.

(1) Either $\partial_w f_v$ is zero or $\partial_w f_v$ is a homogeneous polynomial of total degree $\deg(v) - \deg(w)$.
(2) Assume that v is a product gate with children v_1 and v_2 such that $\deg(v_1) \geq \deg(v_2)$. If $\deg(w) > \deg(v)/2$ then $\partial_w f_v = f_{v_2} \cdot \partial_w f_{v_1}$.
(3) Assume that v is a sum gate with children v_1 and v_2. Then $\partial_w f_v = \partial_w f_{v_1} + \partial_w f_{v_2}$.

Proof. To see Item 1 recall that both f_v, f_w are homogeneous. As $f_v = f_{v,w}|_{y=f_w}$, and by definition of $\partial_w f_v$, it follows that if $\partial_w f_v$ is nonzero then it is of degree $\deg(v) - \deg(w)$. For Item 2 note that since v is a product gate, $\deg(v) = \deg(v_1) + \deg(v_2)$. By assumption we have $\deg(v_2) < \deg(w)$. The gate w is thus not in the sub-circuit rooted at v_2 (as the circuit is homogeneous). Hence $\partial_w f_{v_2} = 0$. Item 3 follows from the definition of $\partial_w f_v$. □

We start with the simplest type of depth reduction, that of formulas (such a reduction holds for Boolean formulas as well).

Theorem 2.7. Let f be computed by an arithmetic formula of size s. Then f can also be computed by an arithmetic formula of depth $O(\log s)$ (and hence of size $\mathrm{poly}(s)$).

Proof. [Sketch] The proof is by a simple induction on the formula-size. For simplicity, in this proof we define the size of a formula to be the number of input gates in it (this is the same as the number of edges up to a constant factor). Let Φ be a formula of size s computing f. Let v be the root of Φ. Choose a vertex w in Φ so that $s/3 \leq |\Phi_w| \leq 2s/3$. Such a vertex always exists as if w is a gate with children w_1, w_2 then $|\Phi_w| = |\Phi_{w_1}| + |\Phi_{w_2}|$. Since Φ is a formula, $f_{v,w}$ is linear in y (recall the definition of $f_{v,w}$ in Section 2.3). I.e., $f_{v,w} = \partial_y f_{v,w} \cdot y + f_{v,w}|_{y=0}$. This implies that:

$$f = f_v = f_{v,w}|_{y=f_w} = \partial_w f_v \cdot f_w + f_{v,w}|_{y=0}. \tag{2.1}$$

Each of the polynomials $f_w, f_{v,w}|_{y=0}$ is computed by a formula of size at most $2s/3$. It can also be shown that $\partial_w f_v$ can be computed by a formula of size $|\Phi_v| - |\Phi_w| \leq 2s/3$ (this follows by induction on v). Therefore, induction on s implies that each of these three polynomials can be computed by a formula of depth $O(\log(s) - \log(2/3))$. Equation (2.1) completes the proof. □

We continue with the more elaborate depth reduction, that of circuits. It exhibits a remarkable property of arithmetic circuits: without loss of generality, we can assume that polynomial size circuits of polynomial degree are of poly-logarithmic depth. This was proved in the seminal work [143]. Such a property is not believed to hold in the Boolean setting. One example of its power is that the determinant of an $n \times n$ matrix can be computed by a polynomial size circuit of depth $O(\log^2 n)$ (although the first prove of this claim [40] did not use a depth reduction).

Valiant Skyum Berkowitz Rackoff

Csanky

Theorem 2.8 ([143]). For every homogeneous degree r polynomial f computed by a circuit Φ of size s, there is a homogeneous circuit Ψ

of size $\text{poly}(r,s)$ computing f with the following additional structure. (1) The circuit Ψ has alternating levels of sum and product gates. (2) Each product gate v in Ψ computes the product of five polynomials, each of degree at most $2\deg(v)/3$. (3) Sum gates have arbitrary fan-in. In particular, the number of levels in Ψ is $O(\log r)$.

It is clear that Theorem 2.8 implies Theorem 1.3, which states that if f is a polynomial of degree r computed by a size s circuit, then f can be computed by a size $\text{poly}(s,r)$ circuit with fan-in at most two of depth $O(\log r(\log r + \log s))$.

Proof. To prove the theorem we shall use properties of partial derivatives in order to express $f_v, \partial_w f_v$ as functions in lower degree polynomial, as we now describe. For an integer $m \in \mathbb{N}$, denote by $\mathcal{G}_m$ the set of multiplication gates t in Φ with children t_1 and t_2 such that $m < \deg(t)$ and $\deg(t_1), \deg(t_2) \leq m$.

Claim 2.9. Let $m > 0$ be an integer, and let v, w be two gates so that $\deg(w) \leq m < \deg(v) < 2\deg(w)$. Then,

$$f_v = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_v \quad \text{and} \quad \partial_w f_v = \sum_{t \in \mathcal{G}_m} \partial_w f_t \cdot \partial_t f_v.$$

Proof. We first show that $f_v = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_v$.

Since $m < \deg(v)$, there is a directed path from (at least one gate in) $\mathcal{G}_m$ to v in Φ. We prove the claim by induction on the length of the longest directed path from $\mathcal{G}_m$ to v. Let v_1 and v_2 be the children of v in Φ.

Induction Base: Assume that $v \in \mathcal{G}_m$. Thus, $\deg(v_1) \leq m$ and $\deg(v_2) \leq m$. So, every gate t in $\mathcal{G}_m$, other than v, is not in Φ_v, which implies that $\partial_t f_v = 0$. Hence, since $\partial_v f_v = 1$,

$$f_v = f_v \cdot 1 + \sum_{t \in \mathcal{G}_m : t \neq v} f_t \cdot 0 = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_v.$$

Induction Step: Consider the following two cases:

Case one: Assume that v is an addition gate. Since the circuit is homogeneous, $\deg(v_1) = \deg(v_2) = \deg(v)$ (otherwise we can ignore one of the gates). So, by induction,

$$f_{v_1} = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_{v_1} \text{ and } f_{v_2} = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_{v_2}.$$

Claim 2.6 hence implies that

$$f_v = f_{v_1} + f_{v_2} = \sum_{t \in \mathcal{G}_m} f_t \cdot (\partial_t f_{v_1} + \partial_t f_{v_2}) = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_v.$$

Case two: Assume that v is a product gate, and assume without loss of generality that $\deg(v_1) \geq \deg(v_2)$. Since $v \notin \mathcal{G}_m$, we have $m < \deg(v_1) \leq 2m$. So, by induction,

$$f_{v_1} = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_{v_1}.$$

For all gates $t \in \mathcal{G}_m$, we have $\deg(v) \leq 2m < 2\deg(t)$. By Item 2 of Claim 2.6 we get

$$f_v = f_{v_1} \cdot f_{v_2} = \sum_{t \in \mathcal{G}_m} f_t \cdot (f_{v_2} \cdot \partial_t f_{v_1}) = \sum_{t \in \mathcal{G}_m} f_t \cdot \partial_t f_v.$$

The proof of the second claim, i.e., that $\partial_w f_v = \sum_{t \in \mathcal{G}_m} \partial_w f_t \cdot \partial_t f_v$, follows the same lines and is left to the reader. □

We now proceed with the proof of Theorem 2.8. First, we assume without loss of generality that $s \geq n$. Second, using Theorem 2.2, we can assume without loss of generality that the circuit computing f, Φ, is a homogeneous arithmetic circuit of size $s' = O(r^2 s)$. To prove the theorem we construct Ψ. The construction is done in steps where at the i-th step we do the following:

(1) Compute all polynomials f_v, for gates v in Φ such that $2^{i-1} < \deg(v) \leq 2^i$.
(2) After we finish the first part, we compute all polynomials $\partial_w f_v$, for all (appropriate) gates v, w so that

$$2^{i-1} < \deg(v) - \deg(w) \leq 2^i \quad \text{and} \quad \deg(v) < 2\deg(w).$$

We will show that we can preform the first part of the i-th step by adding a layer consisting of $O(s'^2)$ product gates, of fan-in at most 3, and a layer of $O(s')$ sum gates, each of fan-in $O(s')$. Similarly, for the second part we need to add a layer consisting of $O(s'^3)$ product gates, of fan-in at most 3, and a layer of $O(s'^2)$ sum gates, each of fan-in $O(s')$, to the circuit that was computed in the first part. Overall, this increases the size by $O(s'^3)$ and the depth by $O(1)$.

The Construction of Ψ

We use the following conventions: Gates in Φ are denoted by v, t, and w. For a gate v, we denote by v' the gate in Ψ computing f_v. For two (appropriate) gates v,w, we denote by (w,v) the gate in Ψ computing $\partial_w f_v$.

Step 0: For every gate v in Φ such that $\deg(v) \leq 1$, the polynomial f_v is linear. So, since $s' \geq n$, we can compute f_v with a linear arithmetic circuit of size $O(s')$ and depth $O(1)$. For every two gates v and w in Φ such that $\deg(v) - \deg(w) \leq 1$, Claim 2.6 implies that $\partial_w f_v$ is linear as well. So, again, we can compute $\partial_w f_v$ with a linear circuit of size $O(s')$ and depth $O(1)$.

Step $i+1$: Assume that we already completed step i.

We first compute the f_v's. Let v be a gate of degree $2^i < \deg(v) \leq 2^{i+1}$, and denote

$$m = 2^i.$$

Recall that if a gate t is not in Φ_v, then $\partial_t f_v = 0$. Thus, by Claim 2.9,

$$f_v = \sum_{t \in T_v} f_t \cdot \partial_t f_v = \sum_{t \in T_v} f_{t_1} \cdot f_{t_2} \cdot \partial_t f_v, \tag{2.2}$$

where T_v is the set of gates $t \in \mathcal{G}_m$, with children t_1 and t_2, such that t is in Φ_v. Clearly, $m < \deg(t) \leq 2m$ and $\deg(t_1), \deg(t_2) \leq m$. Hence, $\deg(v) - \deg(t) \leq 2^{i+1} - 2^i = 2^i$ and $\deg(v) \leq 2^{i+1} < 2\deg(t)$. Therefore, f_{t_1}, f_{t_2}, and $\partial_t f_v$ were already computed. Using Equation (2.2), we can compute f_v, using previously computed gates, by adding one layer of $O(s')$ product gates with fan-in 3 and a sum gate with fan-in $O(s')$.

It remains to compute the $\partial_w f_v$'s. Let v,w be two gates so that

$$2^i < \deg(v) - \deg(w) \leq 2^{i+1} \quad \text{and} \quad \deg(v) < 2\deg(w).$$

Now, denote $m = 2^i + \deg(w)$. Thus, $\deg(w) \leq m < \deg(v) < 2\deg(w)$. Recall that if a gate t is not in Φ_v, then $\partial_t f_v = 0$. Also, by Claim 2.6, if a gate t admits $\deg(t) > \deg(v)$, then $\partial_t f_v = 0$. Hence, by Claim 2.9,

$$\partial_w f_v = \sum_{t \in T_v} \partial_w f_t \cdot \partial_t f_v.$$

Clearly, for $t \in T_v$, we have $\deg(t) \leq \deg(v) < 2\deg(w)$. Denote the children of $t \in T$ by t_1 and t_2, and assume (without loss of generality) that w is in Φ_{t_1}. In particular, $\deg(w) \leq \deg(t_1)$ and $\deg(t_1) \geq \deg(t_2)$. Using Item 2 of Claim 2.6, we obtain

$$\partial_w f_v = \sum_{t \in T_v} f_{t_2} \cdot \partial_w f_{t_1} \cdot \partial_t f_v. \tag{2.3}$$

We now show that all the polynomials f_{t_2}, $\partial_w f_{t_1}$, and $\partial_t f_v$ were already computed, including the first part of the $i+1$ step described above.

Since $\deg(v) \leq 2^{i+1} + \deg(w) \leq 2^{i+1} + \deg(t_1) = 2^{i+1} + \deg(t) - \deg(t_2)$, it holds that $\deg(t_2) \leq 2^{i+1} + \deg(t) - \deg(v) \leq 2^{i+1}$. Therefore, f_{t_2} was already computed (perhaps in the first part of the $i+1$ step).

As $\deg(t_1) \leq m = 2^i + \deg(w)$, we have $\deg(t_1) - \deg(w) \leq 2^i$. Since $\deg(t_1) \leq \deg(t) \leq \deg(v) < 2\deg(w)$, the polynomial $\partial_w f_{t_1}$ was already computed.

Finally, from $\deg(t) > m = 2^i + \deg(w)$, we get $\deg(v) - \deg(t) < \deg(v) - 2^i - \deg(w) \leq 2^{i+1} - 2^i = 2^i$. Since $\deg(v) \leq 2^{i+1} + \deg(w) \leq 2(2^i + \deg(w)) < 2\deg(t)$, the polynomial $\partial_t f_v$ was already computed.

Concluding, using Equation (2.3) we can thus compute $\partial_w f_v$ by adding $O(s')$ product gates of fan-in at most 3 and a sum gate of fan-in $O(s')$.

We thus almost completed the proof of Theorem 2.8. The only remaining problem may be that not in every product gate the degrees of the children are small enough. By slightly modifying the proof it is not hard to get the claim at the cost of increasing the fan-in of each multiplication gate to 5 (e.g., by replacing f_{t_1}, in Equation (2.2), with its expression as a sum of products of lower degree polynomials, if needed). □

A natural thing to ask is whether Theorem 2.8 can be strengthened if we have a bound on the homogeneous formula complexity of f.

The following theorem of Hrubeš and Yehudayoff [63] provides such a strengthening. We leave the proof as an exercise to the reader.

Theorem 2.10 ([63]). For every homogeneous polynomial f of degree r computed by a homogeneous formula Φ of fan-in at most two and of size s, there exist degree r polynomials $f_1,\ldots,f_{s'}$, where $s' \le s+1$, such that the following holds. (1) $f = f_1 + \cdots + f_{s'}$. (2). Every f_i is of the form $f_i = f_{i,1} \cdot f_{i,2} \cdots f_{i,k_i}$ where $(1/3)^j r \le \deg(f_{i,j}) \le (2/3)^j r$ and $\deg(f_{i,k_i}) = 1$. (3) The total number of variables in $f_{1,1},\ldots,f_{s',k_{s'}}$, counted with multiplicities, is at most s. (4) Each $f_{i,j}$ can be computed by a homogeneous formula of size at most s. (5) If Φ is multilinear then so are the formulas for the $f_{i,j}$'s.

Theorem 2.8 can also be used to imply the following "depth-4 version of Theorem 1.3" by [5]. Recall that $\Sigma\Pi\Sigma\Pi$ is the class of depth-4 circuits composed of alternating levels of sum and product gates with a sum gate at the top.

Theorem 2.11 ([5]). Let $f(x_1,\ldots,x_n)$ be a polynomial of degree $r = O(n)$ over $\mathbb{F}$. If there exists a circuit of size $s = 2^{o(r+r\log(n/r))}$ for f, then there exists a $\Sigma\Pi\Sigma\Pi$ circuit of size $2^{o(r+r\log(n/r))}$ for f as well. Furthermore, the fan-in of the top layer of product gates is bounded by $\ell(n)$, where ℓ is any sufficiently slowly growing function tending to infinity with n, and the fan-in of the bottom layer of product gates is bounded by $o(r)$.

Proof. [Sketch] Consider the circuit guaranteed by Theorem 2.8. To prove the theorem, we break that circuit to two parts: the first part is composed of the topmost t levels of multiplication gates together with the addition gates above them, and the second part is the rest of the circuit. Consider the polynomial computed by the first part (imagine that we connect new variables to the bottom layer of this circuit). It is a polynomial of degree at most 5^t in $\mathrm{poly}(r,s)$ variables, and hence can be computed by a depth-2 circuit of size $O(5^t\binom{\mathrm{poly}(r,s)+5^t}{5^t})$; this is the sum of monomials representation. On the other hand, each gate in the second part computes a polynomial of degree at most $D = \deg(f)/(3/2)^t$, and hence it has a depth-2 circuit of size order $\binom{n+D}{D}$. By composing these circuits we obtain a depth-4 circuit of size $O(5^t\binom{\mathrm{poly}(r,s)+5^t}{5^t}\binom{n+D}{D})$ computing f. Optimization over t completes the proof. □

An immediate corollary of the above theorem is that if we can prove an $\exp(n)$ lower bound for the size of depth-4 circuits, computing polynomials of degree $\Omega(n)$, then we automatically get an $\exp(n)$ lower bound for the size of general arithmetic circuits! Very recently, Koiran [88] proved the following extension of Theorem 2.11, using the connection between arithmetic circuits and arithmetic branching programs (see Section 3.4).

Theorem 2.12 ([88]). Let f be an n-variate polynomial of degree r that can be computed by a polynomial size arithmetic circuit. Then f can be computed by a $\Sigma\Pi\Sigma\Pi$ circuit with $n^{O(\log r)}$ addition gates and $n^{O(\sqrt{r}\log r)}$ multiplication gates. Furthermore, multiplication gates have fan-in at most $\sqrt{3r}+1$.

Thus, a $2^{n^{1/2+\varepsilon}}$ lower bound on the depth-4 complexity of the permanent (of $n \times n$ matrices) is sufficient for separating VP from VNP. Currently, the best lower bound for depth-4 circuits (when $\mathrm{char}(\mathbb{F}) \neq 2$) is of the form n^{1+c} for some constant $c > 0$, see Refs. [103, 122].

Open Problem 6. Prove super-polynomial lower bounds for the size of depth-4 circuits over fields of characteristic $\neq 2$.

2.5 Coping with Division Gates

A natural question to ask is why not add the division operator $\div$ to the set of allowed arithmetic operations. Indeed this model was considered in the past. Note that when we allow divisions, each gate computes a rational function instead of a polynomial. Thus, we require that the circuit never divides by the zero polynomial. An answer to the above question was first given by Strassen [133]. Strassen showed that over infinite fields divisions do not add power to the model (more generally, Strassen's approach works when the field contains nonzeros for the polynomial that we divide by [24]). This result was recently extended to finite fields as well [62]. In the proof of this theorem we again use the homogenization procedure discussed above.

Theorem 2.13 ([62, 133]). If a polynomial $f \in \mathbb{F}[x_1,\dots,x_n]$ of degree r can be computed by an arithmetic circuit Φ of size s using the operations $+, \times, \div$, then there is a circuit Ψ of size $\mathrm{poly}(s,r,n)$ that uses only the arithmetic operations $+, \times$ and computes f.

Proof. [Sketch] We start by sketching the proof for large enough fields. We first show that by slightly increasing the size of the circuit we can assume that the only division gate appears at the top of the circuit. In other words, we can assume that $f = h \div g$ where h and g are computed by the two children of the output gate, and the output gate is the only gate labeled by $\div$. To see this, duplicate each gate v to two gates $(v, \text{numerator})$ and $(v, \text{denominator})$ in a way such that $(v, \text{numerator})$ computes the numerator of the rational function that v computes and $(v, \text{denominator})$ computes the denominator of the function computed by v. This can be easily done using the following identities: $h_1/g_1 + h_2/g_2 = (h_1g_2 + h_2g_1)/(g_1g_2)$, $h_1/g_1 \times h_2/g_2 = (h_1h_2)/(g_1g_2)$, and $h_1/g_1 \div h_2/g_2 = (h_1g_2)/(h_2g_1)$.

Finally, we show how to eliminate the only division gate. Write $f = h/g$. Since the field is large enough, $g(\alpha_1,\dots,\alpha_n) \neq 0$ for some $\alpha_1,\dots,\alpha_n \in \mathbb{F}$. By translating the input and multiplying by a field element, with a slight increase in size, we can thus assume that $g(0,\dots,0) = 1$. In this case,

$$f = \frac{h}{g} = \frac{h}{1-(1-g)} = \sum_{j=0}^{\infty} h(1-g)^j,$$

where the last equality means that every homogeneous part of f is the same as the homogeneous part of the right-hand side. As $1-g$ does not contain constants, the minimal degree of a monomial in $(1-g)^j$ is j. Therefore, for every $i \in \{0,\dots,r\}$,

$$H_i(f) = \sum_{j=0}^{\infty} H_i(h(1-g)^j) = \sum_{j=0}^{i} H_i(h(1-g)^j).$$

We can therefore efficiently compute all the homogeneous parts of f, using the homogeneous parts of the polynomials $\{h(1-g)^j : j \in [r]\}$, each of which has a circuit of size $\text{poly}(s,r)$. Theorem 2.2 now completes the proof.

To prove the theorem for small fields, we need to "simulate" a large field. We now explain how this can be done without increasing the size by much. To make a field $\mathbb{F}$ "larger" we consider a field extension $\mathbb{E}$ of $\mathbb{F}$ that is large enough for our purposes (to eliminate divisions we need $\mathbb{E}$ to be of size $\text{poly}(r,n)$). The point is that we can think of the elements of $\mathbb{E}$ as vectors with entries in $\mathbb{F}$, and can thus simulate the arithmetic operations over $\mathbb{E}$ by operations over $\mathbb{F}$: If $\bar{a} = (a_1,\ldots,a_k)$ and $\bar{b} = (b_1,\ldots,b_k)$ are two elements of $\mathbb{E} \cong \mathbb{F}^k$, where k is the degree of $\mathbb{E}$ over $\mathbb{F}$ ($k = O(\log(rs))$ suffices), then the sum of $\bar{a}$ and $\bar{b}$ over $\mathbb{E}$ is just the entry-wise sum of $\bar{a}$ and $\bar{b}$ as vectors in $\mathbb{F}^k$. The product of $\bar{a}$ and $\bar{b}$ over $\mathbb{E}$ can be defined coordinate-wise as $(\bar{a} \times \bar{b})_i = \lambda_i(\bar{a},\bar{b})$ for $i \in [k]$, where λ_i is a (fixed) bilinear form, namely, a $k \times k$ matrix with entries in $\mathbb{F}$. To simulate a circuit over $\mathbb{E}$ by a circuit over $\mathbb{F}$, we "duplicate" each gate k times and simulate the arithmetic operations over $\mathbb{E}$ by the arithmetic operations over $\mathbb{F}$, as described above. □

We have just seen that we can eliminate division gates from circuits without paying too much. Can we do the same for formulas? It turns out that we can, as the following theorem shows.

Theorem 2.14. If a polynomial $f \in \mathbb{F}[x_1,\ldots,x_n]$ of degree r can be computed by a formula Φ of size s using the operations $+,\times,\div$ when the size of $\mathbb{F}$ is at least $sr+1$, then there is a formula Ψ computing f that uses only $+,\times$ of size $\text{poly}(s,r)$. For arbitrary finite fields, the size of Ψ is at most $(sr)^{O(\log\log(sr))}$.

Proof. [Sketch] The proof is similar to the proof of Theorem 2.13. We, therefore, explain mostly the parts of the argument that are different from the proof for circuits. Theorem 2.7 tells us that without loss of generality the depth of Φ is $O(\log s)$. We transform Φ to a formula that has only one $\div$ gate at the top, as in the proof of Theorem 2.13. This step increases the depth of Φ by a constant factor, so its depth remains

$O(\log s)$. Write $f = h/g$, where both h and g have formulas of depth $O(\log s)$. Again, for every $i \in \{0,\ldots,r\}$,

$$H_i(f) = \sum_{j=0}^{\infty} H_i(h(1-g)^j) = \sum_{j=0}^{i} H_i(h(1-g)^j).$$

Each $h(1-g)^j$ has a formula Ψ_j of depth $D = O(\log s + \log r)$ and degree $R \leq rs$. Unlike the case of arithmetic circuits, we do not know an efficient way of transforming a formula to a homogeneous form (unless the degree is small). However, this is not a problem as we do not need a homogeneous formula, but rather we just need to compute $H_i(\Psi_j)$.

When the field is large we achieve this by multiplying each variable x_k in Ψ_j by a new variable y to obtain a new formula $\Psi'_j(y)$ of depth $O(D)$. We have that

$$\Psi'_j(y) = \sum_{i \in \{0,\ldots,R\}} y^i H_i(\Psi_j).$$

We can thus evaluate $\Psi'_j(y)$ at $R+1$ different y's, and use interpolation to recover $H_i(\Psi_j)$. Note that the division operations required for interpolation are of *field* elements and not of polynomials. This can be done using a formula of depth $O(D + \log R)$, in particular, a formula of size poly(r,s).

When the field is small, we need to work over a field extension of degree $k \leq O(\log(rs))$ over $\mathbb{F}$. To do this, as in Theorem 2.13 above, we can work with $k \times k$ matrices. This may increase the depth by a factor of $O(\log k)$, which in terms of size means a factor of $O(\log k)$ in the exponent. □

2.6 Discussion

In this section we proved several fundamental results on the structure of arithmetic circuits. We have seen that there is a universal circuit, discussed efficient ways of transforming a circuit to a homogeneous one, proved that first partial derivatives can be computed efficiently (simultaneously), learned how to eliminate division gates, and more.

While the importance of results like Theorem 2.8 implying $\mathsf{VP} \neq \mathsf{VNC}^2$ is evident, the significant of the other results will become even clearer in Section 3, where they will play an instrumental role in proving lower bounds for different models of arithmetic circuits. As we shall see in Section 3, almost all known lower bound proofs can be presented as following a very general scheme: first prove that polynomials that are computed by the underlying circuit class poses some “simple” structure. This “simple” structure will help us to understand and isolate the “weaknesses” of that circuit class. The lower bound will then follow by constructing an explicit polynomial that does not admit this “simple” structure.

3

Lower Bounds

In this section we survey most of the known lower bounds on the size of arithmetic circuits and formulas. We start by discussing the existence of hard polynomials, i.e., polynomials that cannot be computed by small circuits. We then consider lower bounds for *explicit* polynomials, namely, polynomials in VNP. We first deal with lower bounds for general circuits and formulas, and then move to lower bounds for restricted models, such as monotone, bounded depth, and multilinear circuits. To conclude, we describe several possible approaches for proving lower bounds.

3.1 Existence of Hard Zero-One Polynomials

Finding explicit hard polynomials is the goal we are after in this section. In order to even start our journey, we may ask ourselves "do there exist hard polynomials?" The answer to this question is, of course, yes. Over finite fields, standard counting arguments tell us that there exist hard polynomials with zero-one coefficients. The idea is to show that the number of polynomials that have small circuits is much smaller than the total number of polynomials, and so most polynomials are in fact hard. Over infinite fields, the existence of hard polynomials follows by counting dimensions instead of counting polynomials, as there is an infinite number of circuits and polynomials. In particular, it is less obvious how to prove the existence of hard polynomials with zero-one coefficients, over infinite fields.

In Ref. [62] it was shown[1] that over any field there exist hard polynomials with zero-one coefficients. The proof uses notions from algebraic geometry, the degree and the dimension of a variety, to show that there are few polynomials with zero-one coefficients that are computed by small circuits.

Theorem 3.1. For every $n,r \in \mathbb{N}$ and for every field $\mathbb{F}$, there exists a polynomial f in $\mathbb{F}[x_1,\ldots,x_n]$ of degree r with zero-one coefficients so that every circuit computing f has size at least $\Omega\left(\sqrt{\binom{n+r}{r}}\right)$.

A key observation behind this proof is thinking of a circuit as defining a polynomial map. We shall discuss this idea in more detail in Section 3.8.

Proof. [Sketch] Our goal is to show that the number of polynomials with zero-one coefficients that are computed by small circuits is small. The tricky part is to count this number when the field is infinite as, for example, there are infinitely many circuits of size one. To do so, we use a dimension argument, which is encapsulated in the following lemma. We will not prove the lemma (its proof relies on an appropriate dimension argument).

Lemma 3.2 ([62]). Let $\mathbb{F}$ be a field, and let $F : \mathbb{F}^n \to \mathbb{F}^m$ be a polynomial map of degree $r > 0$ (i.e., each coordinate of F is an n-variate polynomial of degree at most r). Then $|F(\mathbb{F}^n) \cap \{0,1\}^m| \le (2r)^n$.

To use this lemma we show how to construct a polynomial map from each circuit. For a circuit Φ of size at most s and degree at most

[1] In Ref. [145] it is claimed, without a proof, that the methods of Heintz and Sieveking [57] imply a similar statement.

r in the variables $X = \{x_1,\ldots,x_n\}$, let Ψ be the circuit obtained by replacing every field element that occurs in Φ by a new variable z_i, $i \in [s]$. The circuit Ψ thus computes a polynomial in the variables X and $Z = \{z_i\}_{i\in[s]}$. We call such a circuit Ψ a *skeleton*, as it contains no field elements. The circuit Φ is thus a projection of Ψ: the polynomial computed by Φ can be obtained from the polynomial computed by Ψ by an appropriate substitution of field elements to the variables Z.

We now explain how the polynomial computed by Ψ can be viewed as a polynomial map $F = F_\Psi$ from $\mathbb{F}^s$ to $\mathbb{F}^N$, with $N = \binom{n+r}{r}$. The coordinates of F are labeled by monomials M of degree at most r in the variables X, and we denote $F = (F_M)_M$. There are N such monomials, and so there are N entries in F. The entry $F_M \in \mathbb{F}[Z]$ is the coefficient of the monomial M in the polynomial computed by Ψ. In particular, each entry F_M is a polynomial of degree at most 2^s in the Z variables. Thus, F is a degree 2^s polynomial map.

We conclude that when Φ computes a polynomial with zero-one coefficients, then the vector of coefficients it is contained in the image of F_Ψ. However, Lemma 3.2 tells us that the size of the image of F_Ψ inside $\{0,1\}^N$ is at most $(2^{s+1})^s = 2^{O(s^2)}$. Thus every skeleton can "cover" at most $2^{O(s^2)}$ such polynomials. We are almost done. It remains to observe that there are relatively few skeletons: by counting the number of directed acyclic graphs (there are no field elements in a skeleton) we see that the number of skeletons is at most $s^{O(s)} \leq 2^{O(s^2)}$.

This implies that the number of polynomials with zero-one coefficients that can be computed by circuits of size at most s is at most $2^{O(s^2)}$. To complete the proof, notice that the number of polynomials with zero-one coefficients in n variables and degree r is 2^N. This implies that most polynomials in n variables and degree r require circuits of size at least $\Omega(\sqrt{N})$. □

3.2 General Circuits and Formulas

We now describe Strassen's super-linear lower bound on the size of general circuits [132]. As a comparison, we mention that no super-linear lower bound on the size of Boolean circuits is known.

Theorem 3.3 ([132]). Every circuit computing the n polynomials $x_1^r,\ldots,x_n^r$ has size $\Omega(n\log r)$.

Proof. [Sketch] The key ingredient in the proof is Bezout's theorem from algebraic geometry, which states the following. For every k polynomials $f_1,\ldots,f_k$ of degrees $r_1,\ldots,r_k$ over some set of variables, the number of solutions (with multiplicities) to the k equations $f_1 = f_2 = \cdots = f_k = 0$ is either at most $r_1 \cdot r_2 \cdots r_k$ or infinite. For example, when $f_i = x_i^r - 1$, $i \in \{1,\ldots,n\}$, the number of solutions to the equations is r^n when the field is algebraically closed.

Now, assume that there is a circuit Φ of size s computing $x_1^r,\ldots,x_n^r$. The goal is to show that the circuit Φ defines a list of roughly s degree two equations whose solution set is "equivalent" to the set of solutions of $x_1^r - 1 = \cdots = x_n^r - 1 = 0$. By Bezout's theorem this will imply that s must be large. Here is a sketch of the argument. For every gate v in Φ, introduce a new variable y_v. For every input gate v in Φ labeled by α, which is either a variable or a field element, write the equation $y_v - \alpha = 0$. For every $v = u \star w$ with $\star \in \{+,\times\}$, write the equation $y_v - (y_u \star y_w) = 0$. For every output gate v (i.e., a gate computing x_i^r) also write the equation $y_v - 1 = 0$. We thus have a list of at most $2s$ equations in the variables $x_1,\ldots,x_n$ and $\{y_v\}_{v\in\Phi}$, each of degree at most two. Denote this list of equation by $\mathcal{E}$. It is not hard to see that since Φ computes $x_1^r,\ldots,x_n^r$, every solution to $x_1^r - 1 = \cdots = x_n^r - 1 = 0$ gives a solution to $\mathcal{E}$ and vice versa. Bezout's theorem tells us that the number of solution to $\mathcal{E}$ is at most 2^{2s}, as the number of solutions is finite. Therefore, $r^n \le 2^{2s}$ or $s \ge \Omega(n\log r)$. □

Remark 3.1. The proof actually shows that the number of product gates in Φ is at least $\Omega(n\log r)$. This lower bound is tight, as there is a circuit of size $O(n\log r)$ computing these polynomials.

The lower bound above is for a circuit computing a family of n polynomials. However, Baur and Strassen managed to get such a lower bound for a circuit computing a single polynomial [16]. The idea is to use Theorem 2.5 and define a polynomial whose partial derivatives are the x_i^r-s.

Theorem 3.4 ([16, 132]). The size of a circuit computing $x_1^r + \cdots + x_n^r$ is at least $\Omega(n \log r)$, as long as r does not divide the characteristic of the underlying field.

Proof. Theorem 2.5 implies that if there is a circuit of size s for $x_1^r + \cdots + x_n^r$, then there is a circuit of size at most $O(s)$ computing $rx_1^{r-1}, \ldots, rx_n^{r-1}$. Theorem 3.3 thus implies that $s \geq \Omega(n \log r)$. $\square$

Improving this lower bound is a long-standing open problem.

Open Problem 7. Prove a better lower bound for general circuits. For example, prove a super-linear lower bound for the size of a circuit computing an explicit polynomial of constant degree.

For arithmetic formulas better lower bounds are known. Kalorkoti proved an $\Omega(n^3)$ lower bound on the formula-size of determinant [70]. His argument can be thought of as the algebraic analog of Nechiporuk's argument (for the Boolean setting), that gives a lower bound on the Boolean formula-size of a Boolean function f by counting the number of different sub-functions of f. Kalorkoti also proved a quadratic lower bound on the size of formulas computing $\sum_{i \in [n]} \sum_{j \in [n]} x_i^j y_j$ (the lower bound for determinant is not quadratic in the number of variables). This quadratic lower bound is tight, and so this method cannot be used to prove stronger lower bounds.

Theorem 3.5 ([70]). Every formula computing the determinant of an $n \times n$ matrix is of size at least $\Omega(n^3)$.

In contrast, the smallest formula computing determinant is of size $n^{O(\log n)}$.

Proof. [Sketch] We use the notion of *transcendental degree* (abbreviated td). A set of polynomials $g_1, \ldots, g_t \in \mathbb{F}[X]$ is said to be *algebraically independent* if the only polynomial $F \in \mathbb{F}[y_1,\dots,y_t]$ satisfying $F(g_1,\dots,g_t) \equiv 0$ is the zero polynomial. Define $\mathrm{td}(g_1,\dots,g_t)$ as the maximal size of a subset S of $[t]$ so that $\{g_i\}_{i\in S}$ are algebraically independent. Using this notion we define the following complexity measure for polynomials: Let $f \in \mathbb{F}[X]$ be a polynomial and $X' \subset X$ a set of variables. The measure $\mathrm{td}_{X'}(f)$ is defined as follows. Expand f according to X', namely, write $f = \sum_{q\in Q} g_q \cdot q$, where Q is the set of all monomials (including the monomial 1) in X' of degree at most $\deg(f)$, and each g_q is a polynomial in $X \setminus X'$. Define $\mathrm{td}_{X'}(f)$ as the transcendental degree of $\{g_q\}_{q\in Q}$.

The main step in Kalorkoti's proof is the following lemma (he considers the case when division gates are allowed, which makes his proof somewhat more elaborate).

Lemma 3.6. Let $f \in \mathbb{F}[X]$ and $X_1,\dots,X_t$ be a partition of X. Then every arithmetic formula for f must be of size at least $\Omega(\sum_{i\in[t]} \mathrm{td}_{X_i}(f))$.

Proof. [Sketch] The basic property we shall use is that for every $X' \subseteq X$ and two polynomials f_1 and f_2, both $\mathrm{td}_{X'}(f_1 + f_2)$ and $\mathrm{td}_{X'}(f_1 \cdot f_2)$ are at most $\mathrm{td}_{X'}(f_1) + \mathrm{td}_{X'}(f_2)$. With this in mind, the lemma follows by induction on the size of the smallest formula Φ computing f. If Φ is a single variable, the lemma trivially holds as $\mathrm{td}_{X_i}(x)$ is one if x is in X_i and zero otherwise. If $\Phi = \Phi_1 \star \Phi_2$ with $\star \in \{+,\times\}$, then for every $i \in [t]$, we have $\mathrm{td}_{X_i}(\Phi) \le \mathrm{td}_{X_i}(\Phi_1) + \mathrm{td}_{X_i}(\Phi_2)$. Induction, therefore, implies

$$\begin{aligned} |\Phi| \ge |\Phi_1| + |\Phi_2| &= \Omega\left(\sum_{i\in[t]} \mathrm{td}_{X_i}(\Phi_1) + \sum_{i\in[t]} \mathrm{td}_{X_i}(\Phi_2)\right) \\ &\ge \Omega\left(\sum_{i\in[t]} \mathrm{td}_{X_i}(\Phi)\right). \end{aligned}$$ □

To conclude the lower bound for determinant, partition $X = (x_{i,j})$ to $X_1,\dots,X_n$ with $X_i = \{x_{i+j-1,j}\}_{j\in[n]}$, where the indices are taken modulo n. Using the lemma above, it suffices to show that

$\sum_{i \in [n]} \mathrm{td}_{X_i}(\mathrm{DET}) = \Omega(n^3)$. By symmetry it is sufficient to prove that $\mathrm{td}_{X_1}(\mathrm{DET}) \geq \Omega(n^2)$. To see this, observe that determinant contains all monomials of the form $x_{i,j}x_{j,i} \cdot \prod_{\ell \notin \{i,j\}} x_{\ell,\ell}$ with $i \neq j$ in $[n]$. Thinking of $x_{i,j}x_{j,i}$ as the coefficient of the monomial $\prod_{\ell \notin \{i,j\}} x_{\ell,\ell}$ in the variables X_1, we have that $\mathrm{td}_{X_1}(\mathrm{DET})$ is at least the transcendental degree of $\{x_{i,j}x_{j,i}\}_{i \neq j}$, which is $\Omega(n^2)$. As a concluding remark, we note that for any partition of the matrix X to sets $X_1, \ldots, X_t$ we have $\sum_{i \in [t]} \mathrm{td}_{X_i}(\mathrm{DET}) \leq n^4$. □

Proving better lower bounds for arithmetic formulas is a great challenge.

Open Problem 8. Prove a super-polynomial lower bound on the size of arithmetic formulas.

3.3 Monotone Circuits

Monotone arithmetic circuits are circuits over order fields, such as $\mathbb{Q}$ and $\mathbb{R}$, that use only positive numbers as coefficients. Intuitively, such circuits compute polynomials in a "straightforward" way, as there are no cancellations during the computation. This simple model of computation has been studied in many works, and exponential lower bounds for the size of monotone circuits are well known (this is true for the arithmetic case as well as for the Boolean case). Jerrum and Snir [68] proved the following lower bound (see also Refs. [121, 135]). We present a simpler but less accurate analysis that is based on the structural understanding given in Section 2.

Theorem 3.7 ([68]). Every monotone circuit computing the permanent of an $n \times n$ matrix has size $2^{\Omega(n)}$.

Proof. [Sketch] Let Φ be a monotone circuit computing $\mathrm{PERM}(X)$, where X is an $n \times n$ matrix. Induction on the size of Φ, as in Theorem 2.7 above, implies that the permanent can be decomposed as $\mathrm{PERM}(X) = \sum_{i=1}^{s} g_i h_i$, where $s = O(|\Phi|)$, all the coefficients in g_i and

h_i are non-negative, the degree of each g_i is between $n/3$ and $2n/3$ and $\deg(g_i) + \deg(h_i) = n$ (we do not assume anything on the complexity of g_i and h_i). Since Φ is monotone, every monomial that occurs in $g_i h_i$ occurs in the permanent as well.

Fix some $g = g_i$ and $h = h_i$ as above. Let R_g be the rows i in X so that a variable from row i occurs in g, and similarly define R_h. Let C_g be the columns j in X so that a variable from column j occurs in g, and similarly define C_h. Monotonicity implies that $R_g \cap R_h = \emptyset$ and $C_g \cap C_h = \emptyset$. In addition, it implies $|R_g| = |C_g|$ and $|R_h| = |C_h|$ (as otherwise, gh will contain a monomial that does not belong to the permanent). As $\deg(gh) = n$ it follows that the size of R_g is the degree of g and the size of R_h is the degree of h. The properties above imply that the number of monomials in gh is at most $(|R_g|)!(|R_h|)! \leq (2n/3)!(n/3)!$. Since the number of monomials that occur in permanent is $n!$, the size of Φ is at least $\Omega\left(\binom{n}{n/3}\right) = 2^{\Omega(n)}$. □

A different method for proving lower bounds that are tight for monotone circuits was recently presented in Ref. [109] (the above lower bound for permanent is not tight; the smallest known monotone circuit for permanent is of size $2^{O(n \log n)}$). The idea behind the proof is to first decompose the polynomial in question in a similar way to the proof above, i.e., to represent it as a sum of a product of polynomials, and then to use results from additive combinatorics to conclude the lower bound.

Valiant showed that even one "negation" gate is exponentially powerful [140], that is, there exists a polynomial f that has a polynomial size circuit with only one "minus" gate, but every monotone circuit for f has exponential size. We note that given an arithmetic circuit, we can efficiently transform it to a circuit with just one "minus" gate (similar to the first step in the elimination of division gates, see Theorem 2.13). Recently, it was shown that even for constant degree polynomials one "minus" gate gives additional power [64], although their arguments only apply to formulas.

Open Problem 9. Prove a separation between general and monotone circuits for polynomials of constant degree.

3.4 Noncommutative Computation

In the previous section we considered a restricted model of computation. The restriction was a "physical" one, a restriction on the coefficients used in the circuit. Here we consider a different type of restriction, a restriction on the way of "interpreting" the circuit. Given an arithmetic circuit Φ, we can interpret the polynomial computed by Φ in the "usual" way, or we can interpret the polynomial computed by Φ in a *noncommutative* way. Namely, we can think of the variables as living in a noncommutative world, so that the circuit may *not* "assume" that $xy = yx$ for variables x, y. For example, from a commutative point of view $(x + y) \times (x - y) = x^2 - y^2$, whereas the noncommutative way gives $(x + y) \times (x - y) = x^2 + yx - xy + y^2$. Hence, since we have less possible ways to create cancellations, it is more difficult to compute a given polynomial in the noncommutative model than in the commutative model. This makes the task of proving lower bounds for noncommutative circuits easier than for commutative circuits. Indeed, in terms of proving lower bounds, more is known about the noncommutative world than the commutative one.

We start by discussing Nisan's noncommutative formula-size lower bound [98].

Theorem 3.8 ([98]). There exists an explicit noncommutative polynomial f of polynomial degree that can be computed by a noncommutative circuit of linear size, but every noncommutative formula for f has exponential size. In addition, viewed as a commutative polynomial, f can be computed by a polynomial size commutative formula.

This exponential lower bound yields an exponential separation between noncommutative circuit-size and formula-size. This is in contrast to the commutative world, where every circuit having a polynomial size and degree can be simulated by a quasi-polynomial size formula (an immediate corollary of Theorem 2.8). Another interesting point is that the polynomial f given in Theorem 3.8, viewed as a commutative polynomial, has a polynomial size commutative formula, and hence noncommutative formulas are exponentially weaker than commutative ones. Nisan also proved similar exponential lower bounds on the formula-size of the noncommutative permanent and determinant.

Chien and Sinclair strengthened Nisan's result in the following way [35]. Nisan proved that, say, determinant cannot be computed by small noncommutative formulas as a *formal polynomial.* This does not rule out the possibility that over some other noncommutative domains, that are more structured than the free noncommutative algebra, it is possible to compute the determinant by a small formula. Chien and Sinclair proved that even when the computation is over 2×2 real matrices, which form the "most structured non-trivial noncommutative domain," there are no small formulas for determinant.

We now sketch the proof of Theorem 3.8.

Proof. [Sketch] To understand noncommutative formulas, Nisan defined the notion of an *arithmetic branching program* (ABP).

Definition 3.1 (ABP). An *ABP* is a layered graph with $d+1$ layers as follows. The layers are labeled by $0,1,\ldots,d$. The edges of the graph go from layer i to layer $i+1$. Every edge e is labeled by a homogeneous linear form $\ell_e = \sum_{j\in[n]} c_{e,j}x_j$. Layer 0 has only one vertex called the *source*, and layer d has only one vertex called the *sink*. For every directed path from the source to the sink $\gamma = (e_1,e_2,\ldots,e_d)$, define the polynomial associated to γ, denoted $f[\gamma]$, as follows: $f[\gamma] = \ell_{e_1}\ell_{e_2}\cdots\ell_{e_d}$. The polynomial computed by an ABP is $\sum_\gamma f[\gamma]$.

The first step in the proof is to simulate noncommutative formulas by ABPs. This is done in two stages. In the first stage we construct by induction a "messy" ABP as follows. We define ABP(Φ) to be the ABP corresponding to the formula Φ by induction. When Φ is an input gate, it is clear how to define ABP(Φ). To construct ABP($\Phi_1 \times \Phi_2$) we connect ABP(Φ_1) and ABP(Φ_2) sequentially, and to construct ABP($\Phi_1 + \Phi_2$) we connect ABP(Φ_1) and ABP(Φ_2) in a parallel way. In the second stage we transform the "messy" ABP to an ABP as follows. Split every vertex v to r vertices $(v,1),\ldots,(v,r)$, where r is the degree of the polynomial computed by Φ. Put all vertices of the form (v,i) in layer i. Wire the edges so that (v,i) computes the homogeneous part of degree i of v (here the wiring is slightly more elaborate than in Theorem 2.2). The second stage may cause an increase in size by a factor of $O(r)$.

The second step in the proof is to find a "weakness" of noncommutative ABPs. To do so, Nisan used the so-called partial derivative matrix. Given a homogeneous noncommutative polynomial f of degree r in $x_1,\ldots,x_n$ and $k \in \{0,1,\ldots,r\}$, define $M_k(f)$ as the n^k by n^{r-k} matrix whose $((i_1,\ldots,i_k),(j_1,\ldots,j_{r-k}))$ entry is the coefficient of the monomial $x_{i_1}\cdots x_{i_k}x_{j_1}\cdots x_{j_{r-k}}$ in f. Note that due to non-commutativity, every noncommutative monomial of degree r can be uniquely decomposed to a product of two monomials of degrees k and $r-k$. The "weakness" of noncommutative ABPs is that if the k-th layer in a noncommutative ABP has t vertices, then the polynomial f computed by the ABP admits $\mathrm{rank}(M_k(f)) \leq t$. Indeed, denote by $h_1,\ldots,h_t$ the polynomials computed by the vertices in layer k, thinking of these vertices as sinks. Denote by $g_1,\ldots,g_t$ the polynomials computed by the sink when the sources are the t vertices in layer k, respectively. Thus, $f = \sum_{i=1}^t h_i g_i$ and so $M_k(f) = \sum_{i=1}^t M_k(h_i g_i)$. It is not hard to see that each of the matrices $M_k(h_i g_i)$ has rank one, and hence $\mathrm{rank}(M_k(f)) \leq t$.

Therefore, if a polynomial f of degree $2r$ admits $\mathrm{rank}(M_r(f)) \geq R$, then every noncommutative ABP for f has size at least R, and so every noncommutative formula for f is large as well. We now give an example of a degree $2n$ polynomial with $R = 2^n$. Let $x_1^0,x_1^1,x_2^0,x_2^1,\ldots,x_n^0,x_n^1$ be $2n$ variables. For every $a = (a_1,\ldots,a_n) \in \{0,1\}^n$, define the degree $2n$ monomial $m_a = x_n^{a_n}x_{n-1}^{a_{n-1}}\cdots x_1^{a_1}x_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$. Set $f = \sum_a m_a$. Thus, $M_n(f)$ is a permutation matrix of size $2^n \times 2^n$ and so its rank is 2^n. By the discussion above, f requires exponentially large noncommutative formulas. Note, however, that f can be computed by a linear size noncommutative circuit (of linear depth, of course); define $f_0 = 1$ and $f_i = x_i^0 f_{i-1} x_i^0 + x_i^1 f_{i-1} x_i^1$ for $i \in [n]$. It is clear that $f = f_n$ has a small circuit. $\square$

Albeit this understanding of noncommutative formulas, we do not know any lower bound for noncommutative circuits, that is better than the lower bound for commutative circuits that we already know.

In Section 3.8.5 we present an approach of Hrubeš et al. [60] for proving lower bounds for noncommutative circuits.

Open Problem 10. Prove a super-polynomial lower bound on the size of noncommutative circuits.

We remark that the fact that $\text{rank}(M_k(f)) \leq t$ (in the notations of the proof) will play an important role in Section 4.5 where we discuss deterministic algorithms for *polynomial identity testing* of noncommutative formulas.

3.5 Constant Depth Circuits

We now address a second family of restricted circuits, constant depth circuits (recall that for constant depth circuits the fan-in is unbounded). One of the main motivations for studying restricted families of circuits is to gain a better understanding of the general model. In Boolean circuit complexity, the model of bounded depth circuits has received a lot of attention and exponential lower bounds (that deteriorate as the depth increases) were proved for it [53], but no lower bounds for general Boolean circuits were derived from those results. In particular, currently we do not know how to translate a constant depth lower bound of the form $\exp(n)$, for any Boolean function f, to anything more than a proof that f cannot be computed by linear size and logarithmic depth Boolean circuits (see e.g., the recent Survey [144]). However, in sharp contrast to the Boolean case, if we could prove an $\exp(n)$ lower bound for any n-variate multilinear polynomial even for depth-4 arithmetic circuits then this immediately implies an $\exp(n)$ lower bound for general arithmetic circuits [5, 103, 88]. Thus, understanding shallow arithmetic circuits is almost as difficult and important task as understanding general circuits. This fact gives a very strong motivation for studying small depth circuits.

We start this section with a brief survey of known lower bounds for the size of constant depth circuits. We then consider depth-3 circuit, the "first depth that we do not understand." Finally, we discuss circuits of arbitrary constant depth.

The simplest type of constant depth circuits are depth-2 circuits. Such circuits compute either a sum of monomials or a product of linear functions. This means that the size of a depth-2 circuit computing an irreducible f is just the number of monomials in f.

Our understanding of (even) depth-3 circuits is far from complete. Recall that depth-3 $\Sigma\Pi\Sigma$ circuits compute a sum of products of linear forms (there is also the less interesting case of $\Pi\Sigma\Pi$ circuits, that are "closer" to depth-2 circuits, which we do not discuss here). Over finite fields, Grigoriev and Karpinski [50] and Grigoriev and Razborov [52] proved exponential lower bound on the size of $\Sigma\Pi\Sigma$ circuits. In fact, their lower bound holds for any circuit computing the *function* MOD_q (where q is a prime number different than the characteristic of the field), which is stronger than a lower bound for computing some specific polynomial-representation of the function (recall that the MOD_q function is defined as $\mathrm{MOD}_q(x_1,\ldots,x_n) = 1$ iff $\sum_i x_i = 0 \mod q$, thinking of x_i as an integer).

Theorem 3.9 ([50, 52]). Let $p \neq q$ be two primes. Then, every $\Sigma\Pi\Sigma$ circuit computing the n-variate MOD_q function over $\mathbb{F}_p$ must be of size at least $2^{\Omega(n)}$.

Proof. [Sketch] Let Φ be a $\Sigma\Pi\Sigma$ circuit over the field with p elements $\mathbb{F}_p$. The idea is to partition the product gates of Φ into two sets, and to find the weakness of each of these sets. We partition the gates of Φ according to their *rank*: Every product gate v in Φ of degree d_v multiplies d_v linear functions. Define the *rank* of v as the dimension of the span of these d_v linear functions. Partition the product gates of Φ into V_{HIGH} and V_{LOW}, where V_{HIGH} is the set of product gates of rank at least R (we shall choose $R = \varepsilon n$ for a constant $\varepsilon > 0$ that depends on p and q), and V_{LOW} contains the rest of the gates.

We now describe the weakness of $\Sigma\Pi\Sigma$ circuits (over finite fields). If we substitute random field elements as inputs, each gate in V_{HIGH} is nonzero with probability at most $(1-1/p)^R = 2^{-c_p R}$. On the other hand, each product gate in V_{LOW} can be represented by a low degree polynomial, namely, a polynomial of degree at most pR. The union bound hence implies that a $\Sigma\Pi\Sigma$ circuit Φ of size s can be well approximated by a low degree polynomial. More accurately, there exists a polynomial $g \in \mathbb{F}_p[x_1,\ldots,x_n]$ of degree at most pR so that

$$\Pr_a[\Phi(a) \neq g(a)] \leq 2^{-c_p R} s,$$

where a is uniform in $\mathbb{F}_p^n$.

Smolensky [130] proved that the MOD_q function cannot be well approximated by low degree polynomials over the Boolean cube. Intuitively, this property of the MOD_q function should complete our proof. However, Smolensky's result holds over the Boolean cube, and does not imply the needed property over $\mathbb{F}_p$. To overcome this, Grigoriev and Razborov use the observation that the uniform distribution over $\mathbb{F}_p^n$ is a convex combination of distributions that are uniform over translates of $\mathbb{F}_2^n$ in $\mathbb{F}_p^n$. The proof now immediately follows. $\square$

Over large fields, however, the best lower bound known on the size of $\Sigma\Pi\Sigma$ circuits is quadratic [129, 123]. In fact, over infinite fields no super-linear lower bound on the number of *product gates* in depth-3 circuits is known. We shall now give a proof of a nearly quadratic lower bound for depth-3 circuits computing the determinant.

Theorem 3.10 ([129]). Every $\Sigma\Pi\Sigma$ circuit computing the determinant of an $n \times n$ matrix must be of size at least $\Omega(n^4/\log n)$.

Proof. [Sketch] The proof combines two common methods in lower bounds proofs. We eliminate gates of high degree by restricting our inputs to some affine space (thus making certain linear functions vanish). Then, we claim that the space of partial derivatives of the remaining circuit is of somewhat low rank whereas that of the determinant has high rank. For the latter property we shall rely on the fact that determinant is downward-self-reducible in a strong sense.

Let us start with some notation. Recall that for a polynomial $f \in \mathbb{F}[x_1,\ldots,x_n]$, we denote with $\partial_{x_i}(f)$ the partial derivative of f with respect to x_i. We also define $\partial_{\{x_1,x_2\}}(f)$ as $\partial_{x_1}(\partial_{x_2}(f)) = \partial_{x_2}(\partial_{x_1}(f))$. Notice that the order does not matter. For any $S \subseteq [n]$, we can similarly define $\partial_S(f)$. Finally, for an integer k, we denote by $\partial(f;k)$ the vector space over $\mathbb{F}$ spanned by all polynomials of the form $\partial_S(f)$, where S is a sized k set.

We shall use the following two simple claims.

Claim 3.11. For every product gate v of degree r in a $\Sigma\Pi\Sigma$ circuit, the dimension of $\partial(f_v;k)$ is at most $\binom{r}{k}$, where f_v is the polynomial that v computes.

Proof. Assume $f_v = \prod_{i=1}^{r} \ell_i$, where each ℓ_i is a linear form. The space $\partial(f_v;k)$ is spanned by the set $\{\prod_{i\in T} \ell_i : T \subseteq [r],\ |T| = r-k\}$, whose size is at most $\binom{r}{k}$. □

Claim 3.12. For every k, the dimension of $\partial(\mathrm{DET}_n;k)$ is at least $\binom{n}{k}^2$, where DET_n is the determinant of an $n\times n$ matrix $X = (x_{i,j})$.

Proof. For every two sets $R = \{r_1 < r_2 < \cdots < r_k\}$ and $C = \{c_1 < c_2 < \cdots < c_k\}$ of $[n]$, let $S(R,C) = \{x_{r_1,c_1},\ldots,x_{r_k,c_k}\}$. The polynomial $\partial_{S(R,C)}(\mathrm{DET}_n)$ is the determinant of the $n-k\times n-k$ matrix that is obtained by deleting the rows indexed by R and columns indexed by C (these are all monomials that "contain" the monomial $x_{r_1,c_1}x_{r_2,c_2}\cdots x_{r_k,c_k}$). The set of all such $\partial_{S(R,C)}(\mathrm{DET}_n)$ is linearly independent over $\mathbb{F}$. The number of such pairs (R,C) is $\binom{n}{k}^2$. □

The two claims above tell us that if all product gates in a $\Sigma\Pi\Sigma$ circuit Φ computing DET_n are of degree at most D, then the size of Φ is at least $\binom{n}{k}^2/\binom{D}{k}$ for every k. Choosing $k = n^2/(D\cdot e)$, Stirling's approximation tells us that the size of such a Φ is at least $\Omega(e^{n^2/(D\cdot e)})$. Thus, if D is linear in n then we get an exponential lower bound. However, in general this does not have to be the case and so our argument is based on the following case analysis. If there are "many" gates of "high" degree then clearly the circuit has many wires. I.e., a lower bound immediately follows. On the other hand if the circuit has a few gates of high degree then by restricting the inputs to a subspace we can eliminate all gates and still get a circuit computing the determinant of a slightly smaller matrix. We now explain the details behind this argument: So far we established that when all product gates are of small degree, the circuit must be large. In particular, if all the product gates containing the variable $x_{1,n}$ are of degree at most D, then the size of Φ is at least $\Omega(e^{(n-1)^2/(D\cdot e)})$. Indeed, consider the circuit $\Psi = \Phi|_{x_{1,n}=1} - \Phi|_{x_{1,n}=0}$ where we eliminate all product gates not containing $x_{1,n}$ in Φ and only maintain the gates of Φ containing $x_{1,n}$ (each such gate will "appear" twice in Ψ). Clearly Ψ computes DET_{n-1} and all its gates are of degree at most D. Hence a lower bound on the size of Ψ and hence on the size of Φ follows. Clearly, the same reasoning works if we consider $x_{1,n-1}, x_{1,n-2}, \ldots, x_{1,1}$ and not just $x_{1,n}$.

If this is not the case, i.e., there is a gate $v_{1,n}$ of degree at least D that contains $x_{1,n}$, then there is a linear function ℓ_1 so that the restriction of the polynomial $\Phi_{v_{1,n}}$ to the subspace $x_{1,n} = \ell_1$ is zero, i.e., $\Phi_{v_{1,n}}|_{x_{1,n}=\ell_1} = 0$. Applying this restriction, we can eliminate at least D wires from Φ. We repeat this line of argument sequentially for $x_{1,n-1}, x_{1,n-2}, \ldots, x_{1,2}$. If at some point we find that all gates containing $x_{1,i}$ (after the substitutions made to $x_{1,n}, \ldots, x_{1,i+1}$) are of degree D then we proceed as in our former argument. Otherwise we find an appropriate substitution to $x_{1,i}$. At the end of the process we get that either there are at least $\Omega(e^{n^2/(De)})$ gates in Φ or that we eliminated $(n-1)D$ wires. In the latter case we set $x_{1,1} = 1$ and $x_{i,1} = 0$ for all $i \in \{2, \ldots, n\}$. This gives a circuit computing DET_{n-1}. To conclude, if we denote by $\Sigma\Pi\Sigma(\mathrm{DET}_n)$ the size of the smallest $\Sigma\Pi\Sigma$ circuit computing the determinant of an $n \times n$ matrix, then for every D,

$$\Sigma\Pi\Sigma(\det_n) \geq \max(\Omega(e^{(n-1)^2/(D\cdot e)}), \Sigma\Pi\Sigma(\det_{n-1}) + (n-1)D).$$

Choosing $D = n^2/(4e\log n)$, the theorem follows by induction. □

Open Problem 11. Prove super-polynomial lower bounds on the size of $\Sigma\Pi\Sigma$ circuits over infinite fields.

For depths higher than three weaker lower bounds are known. Raz [103] and Shoup and Smolensky [122] proved a lower bound of (roughly) $n^{1+1/d}$ on the size of depth-d circuits over arbitrary fields. The main difference between [122] and [103] is that [122] proved[2] the lower bound for a polynomial of degree n, whereas Raz [103] proved the lower bound for a polynomial of constant degree, $O(d)$. This difference is particularly interesting when we are dealing with constant depth circuits, since without loss of generality we can assume that the best arithmetic circuit for a constant degree polynomial is of constant depth. Hence, if the lower bound of Raz [103] was a sufficiently large polynomial rather than just super-linear, then it would hold for general, arbitrary depth, circuits as well. We now discuss Raz's lower bound [103].

Theorem 3.13 ([103]). For every $d \in \mathbb{N}$, there exists a polynomial f with zero-one coefficients of degree at most $O(d)$ in $O(nd)$ variables so that every depth-d circuit computing f is of size at least $n^{1+\Omega(1/d)}$.

Raz's proof uses the language of Elusive Functions. To simplify the presentation, we prove the theorem without using elusive functions (for more details about Elusive Functions see Section 3.8.3). The argument that we give is close in spirit to the proof of Shoup and Smolensky [122].

Proof. [Sketch] We shall use the following complexity measure. Let $\mathbb{G}$ be a field extension of $\mathbb{F}$ (e.g., $\mathbb{G} = \mathbb{F}(Y)$, the field of rational functions over a new set of variables Y). Let $\mathcal{F} = \{f_a : a \in [n]\}$ be a family of homogeneous degree one polynomials in the variables $X = \{x_b : b \in [n]\}$ over $\mathbb{G}$. Think of each f_a as a vector of coefficients: for a variable x_b, denote by $f[a,b]$ the coefficient of x_b in f_a. Define $\mathcal{D}_m(\mathcal{F})$ as the span over $\mathbb{F}$ of all expressions of the form $\prod_{(a,b)\in S} f[a,b]$, where S is a subset of $[n] \times [n]$ of size m. As $\mathbb{G}$ is an extension field of $\mathbb{F}$, this vector space is not trivial. We are mainly interested in the dimension of $\mathcal{D}_m(\mathcal{F})$ as a vector space over $\mathbb{F}$.

The following claim is the basic observation behind the lower bound. A variant of this lemma appeared in Ref. [122].

[2] In fact, this can be deduced using the techniques of Shoup and Smolensky [122], but it was not stated nor proved there.

Lemma 3.14. Let $\mathcal{F}$ be a family of homogeneous degree one polynomials that can be computed by a depth-d circuit Φ of size s over $\mathbb{G}$. Then the dimension of $\mathcal{D}_m(\mathcal{F})$ is at most $\binom{s+m}{m}^d$.

Proof. As every polynomial in $\mathcal{F}$ is linear and homogeneous, we can assume without loss of generality that every gate in Φ computes a homogeneous degree one polynomial. We can transform Φ to such a form, without increasing the size and depth, by keeping only the "linear homogeneous part" of every gate (here we allow gates to compute arbitrary linear combinations of their inputs).

Denote by T_i the set of elements of $\mathbb{G}$ that label edges that enter gates of depth $i-1$ in Φ. By the assumed structure of Φ, every coefficient in $f[a,b]$ is a sum of products of the form $\prod_{i\in[d]} t_i$ with $t_i \in T_i \cup \{1\}$. Thus $\mathcal{D}_m(\mathcal{F})$ is contained in the span over $\mathbb{F}$ of all expressions of the form

$$(t_{1,1}t_{1,2}\cdots t_{1,m})(t_{2,1}t_{2,2}\cdots t_{2,m})\cdots(t_{d,1}t_{d,2}\cdots t_{d,m})$$

with every $t_{i,j} \in T_i \cup \{1\}$. By a rough estimate, the number of such expressions is at most $\binom{s+m}{m}^d$. $\square$

To prove a lower bound, it thus suffices to find a family $\mathcal{F}$ so that the dimension of $\mathcal{D}_m(\mathcal{F})$ is large. Raz defined such a family as follows. Assume that n is prime, and consider the field of rational functions $\mathbb{G} = \mathbb{F}(Y)$ with $Y = \{y_{i,j} : i \in [5d], j \in [n]\}$ and the variables $X = \{x_i : i \in [n]\}$. For every $a,b \in [n]$, define $f[a,b] \in \mathbb{G}$ as

$$f[a,b] = \prod_{i\in[5d]} y_{i,ai+b},$$

where $ai+b$ is taken modulo n. This defines a family $\mathcal{F}$ of n homogeneous degree one polynomials in the variables X over $\mathbb{G}$. An important point in the argument is that we can also think of $\mathcal{F}$ as a family of polynomials over the field $\mathbb{F}$ in the variables X,Y (so the "final" family of polynomials is in $\mathbb{F}[X,Y]$). Note that since the output polynomials are homogeneous linear forms in the X variables, we can still assume that each gate in the circuit computes a homogeneous linear form in X and therefore the argument remains the same.

Claim 3.15. The dimension of $\mathcal{D}_m(\mathcal{F})$, with $\mathcal{F}$ defined above, for $m = \lfloor n^{1-1/(2d)} \rfloor$, is at least $\frac{1}{2}\binom{n^2}{m}$.

Proof. For every set $S \subset [n] \times [n]$ of size m, define $f[S] = \prod_{(a,b) \in S} f[a,b]$. Notice that $f[S]$ is a monomial in the Y variables. It thus suffices to show that the number of different Y monomials in the set $\{f[S] : S \subset [n] \times [n], \|S| = m\}$ is at least one half of the total number of such sets S.

We say that a set S is *unique* if for every $(a,b) \notin S$, one of the variables in $\{y_{i,ai+b} : i \in [5d]\}$ does not appear in $f[S]$. The term "unique" originates from the following property. If S is unique and $f[S'] = f[S]$, then every (a,b) not in S is also not contained in S', and therefore $S = S'$.

We now claim that at least half of the sets $S \subset [n] \times [n]$ of size m are unique. This will complete the proof of the claim. To see that this holds, let us choose S uniformly at random and prove that S is unique with probability at least one half. For every $(a,b) \in [n] \times [n]$ and $i \in [5d]$, denote by $L_{a,b,i}$ the "line" $\{(a',b') \in [n] \times [n] : (a - a') + i(b - b') = 0 \bmod n\}$. For every (a,b), the following events are equal

$$\begin{aligned}&\{\text{one of the variables } \{y_{i,ai+b} : i \in [5d]\} \text{ does not appear in } f[S]\}\\ &\qquad \equiv \{\text{there exists } i \in [5d] \text{ so that } S \cap L_{a,b,i} = \emptyset\}.\end{aligned}$$

Denote by $\mathcal{E}_{a,b,i}$ the event $\{S \cap L_{a,b,i} \neq \emptyset\}$. The size of $L_{a,b,i} \setminus \{(a,b)\}$ is $n-1$ and the size of $[n] \times [n] \setminus \{(a,b)\}$ is $n^2 - 1$. As for $i \neq i'$ in $[5d]$, the two lines $L_{a,b,i}, L_{a,b,i'}$ intersect only at the point (a,b), for every a,b,i, we have

$$\begin{aligned}&\Pr[\mathcal{E}_{a,b,i} | \mathcal{E}_{a,b,1}, \ldots, \mathcal{E}_{a,b,i-1}, (a,b) \notin S]\\ &\qquad \leq \Pr[\mathcal{E}_{a,b,i} | (a,b) \notin S] \leq |S|(n-1)/(n^2-1),\end{aligned}$$

which implies

$$\begin{aligned}&\Pr\left[\bigcap_{i \in [5d]} \mathcal{E}_{a,b,i} \middle| (a,b) \notin S\right]\\ &\qquad \leq (|S|(n-1)/(n^2-1))^{5d} \leq n^{-5d/(2d)} \leq n^{-2}/2.\end{aligned}$$

Applying the union bound again, the probability that S in not unique is at most $n^2 n^{-2}/2 = 1/2$. $\square$

Lemma 3.14 with $m = \lfloor n^{1-1/(2d)} \rfloor$ and Claim 3.15 imply that if a depth-d circuit of size s computes $\mathcal{F}$ then $\binom{n^2}{m}/2 \le \binom{s+m}{m}^d$, which implies $s \ge n^{1+\Omega(1/d)}$. It just remains to conclude the same lower bound for a single polynomial rather than a family of n polynomials. Theorem 2.5 tells us that the constant depth circuit-size of $\mathcal{F}$ is equivalent to that of

$$f = \sum_{a \in [n]} z_a f_a,$$

where $\mathcal{F} = \{f_a : a \in [n]\}$, and $Z = \{z_a : a \in [n]\}$ is a new set of variables. This completes the proof (to conclude the theorem for all integers n, we can use the density of prime numbers). We remark that the "final" polynomial for which the lower bounds is proved, is a polynomial in the $n + 5nd + n$ variables X, Y, Z over $\mathbb{F}$. As explained above, this does not change the conclusion. $\square$

To conclude our discussion of constant depth circuits, we consider a special type of $\Sigma\Pi\Sigma$ circuits which we suggest as a step toward solving Problem 11.

Denote $\sigma_{k,m} = \sum_{S \subset [m] : |S| = k} \prod_{i \in S} x_i$, the k-th symmetric polynomial in m variables. Ben-Or gave a $\Sigma\Pi\Sigma$ circuit of size $O(n^2)$ that, simultaneously, computes $\sigma_{0,n}, \ldots, \sigma_{n,n}$ (see Ref. [129]). Here is a rough description of this circuit: evaluate the polynomial $f(z) = (z + x_1)(z + x_2) \cdots (z + x_n) = \sum_{i=0}^{n} z^{n-i} \sigma_i$ at $n + 1$ different points, and recover the coefficient of z^{n-i}, which is σ_i, using interpolation. This construction works as long as the field is large enough, namely, contains at least $n + 1$ points so that interpolation is possible. In contrast, in the Boolean world we know that computing the symmetric polynomials requires exponentially large constant depth circuits. This is another evidence of the difference between the finite field case and the infinite field case.

In [123] it was shown that for every polynomial $f(x_1, \ldots, x_n)$ of degree r over the field of complex numbers $\mathbb{C}$, there exists $m = m(f) \in \mathbb{N}$ so that $f = \sigma_{r,m}(y_1, \ldots, y_m)$, where $y_1, \ldots, y_m$ are linear functions in $x_1,\ldots,x_n$. Using Ben-Or's construction, we conclude that, over $\mathbb{C}$, every polynomial f has a $\Sigma\Pi\Sigma$ circuit of size roughly $(m(f))^2$. Therefore, proving lower bounds on $m(f)$ over $\mathbb{C}$ is easier than proving lower bound on the size of a $\Sigma\Pi\Sigma$ circuit for f.

Open Problem 12. Prove super-polynomial lower bounds on $m(f)$ over $\mathbb{C}$.

Currently the best lower bound known on $m(f)$ is linear in n [123]. Thus, even the problem of proving super-linear lower bounds on $m(f)$ is open.

3.6 Multilinear Circuits and Formulas

In the multilinear world our understanding of formulas, for which strong lower bounds are known, is quite good (but still far from being complete), whereas our understanding of circuits, for which only weak bounds are known, could be greatly improved. We start by stating the lower bounds and separations known for multilinear formulas, and explaining the ideas behind the proofs of these results. We then sketch the proof of the best known lower bound for multilinear circuits, and finally we discuss lower bounds and separations for constant depth multilinear circuits. The reader is advised to recall the relevant definitions from Section 2.2.1.

The first super-polynomial lower bounds on the size of multilinear formulas were proved by Raz [104]. We shall give a sketch of the proof idea below.

Theorem 3.16 ([104]). Every multilinear formula computing either the permanent or the determinant of an $n \times n$ matrix must be of size at least $n^{\Omega(\log n)}$.

Subsequently, Raz showed a super-polynomial separation between multilinear circuit-size and formula-size [102]. Later, Raz and Yehudayoff [108] gave a simplified proof for this separation.

Theorem 3.17 ([102]). There exists a polynomial f that can be computed by a polynomial size syntactically multilinear circuit, but every multilinear formula for f must be of super-polynomial size.

As for general circuits (Theorem 2.8 above), syntactically multilinear circuits can be simulated efficiently by syntactically multilinear circuits of depth $\log^2 n$. Theorem 3.17 therefore shows that in the multilinear world $\mathsf{NC}^1 \neq \mathsf{NC}^2$. In addition, by examining the proof of Theorem 3.17 we see that the "ideas" of Theorem 3.16 are not sufficient for proving super-polynomial lower bounds on the size of syntactically multilinear circuits (as the lower bound proof is the same as in Theorem 3.16).

The best lower bound known on the size of syntactically multilinear circuits is slightly better than the one known for general circuits [107].

Theorem 3.18 ([107]). There is an explicit polynomial $f \in \mathsf{VNP}$ with 0/1 coefficients so that every syntactically multilinear circuit computing f must be of size $\Omega(n^{4/3}/\log^2 n)$.

As a side remark, we note that one can show that a lower bound of s on the size of syntactically multilinear circuits automatically implies an $\Omega(s/n^3)$ lower bound on the size of noncommutative circuits [60].

Open Problem 13. Prove a super-polynomial lower bound on the size of multilinear circuits.

We now move on to discussing the ideas required to prove Theorem 3.16. The two main tools required for proving lower bounds for multilinear formulas are the partial derivative matrix and random partitions of the variables. These tools were first used in this context by Raz, and they could be seen as an adaptation/evolvement of similar ideas from the earlier works [53, 98, 100]. We first give a brief description of these concepts, and then discuss how to combine them to a proof.

The *partial derivative matrix* was previously discussed in the context of noncommutative computation (see Section 3.4). Here we use it in a slightly different manner. Given a polynomial f in two sets of variables $Y = \{y_1,\ldots,y_m\}$ and $Z = \{z_1,\ldots,z_m\}$, define $M(f)$ as the 2^m by 2^m matrix whose (S,T) entry, $S,T \subseteq [m]$, is the coefficient of the monomial $\prod_{i\in S} y_i \prod_{j\in T} z_j$ in f. The partial derivative matrix has a few useful and simple properties (for more details see, e.g., Ref. [104]).

Lemma 3.19. Given two polynomials f and g,

(1) $\text{rank}(M(f+g)) \le \text{rank}(M(f)) + \text{rank}(M(g))$,
(2) $\text{rank}(M(f\cdot g)) \le \text{rank}(M(f)) \cdot \text{rank}(M(g))$ (here we take the multilinear part of $f\cdot g$), and
(3) $\text{rank}(M(f)) \le 2^{\min(Y(f),Z(f))} \le 2^{(Y(f)+Z(f))/2}$, where $Y(f)$ is the number of Y variables that occur in f and $Z(f)$ is the number of Z variables that occur in f.

As in the noncommutative world, we think of the rank of this matrix as an "evidence of hardness" of a given polynomial. However, the situation in the multilinear world is not as simple as in the noncommutative world. For example, the partial derivative matrix of $f = (y_1+z_1)(y_2+z_2)\cdots(y_m+z_m)$ has full rank, but f has a linear size formula. To overcome this difficultly, Raz suggested to consider the rank with respect to a random partition of the variables to two sets. Specifically, let $X = \{x_1,\ldots,x_{2m}\}$ be a set of variables. A *partition* A is a one-to-one map from X to $Y \cup Z$. In other words, it is a renaming of the variables X by Y and Z. For a polynomial f in X, define $f[A]$ as the polynomial in Y,Z that is obtained by substituting every x_i by $A(x_i)$ in f. For our needs we will only consider partitions in which $|Y| = |Z|$. The following theorem is the crux of Raz's idea and demonstrates a "weakness" of multilinear formulas.

Theorem 3.20. ([102]) There is a constant $\varepsilon > 0$ so that if a multilinear formula of size $n^{\varepsilon \log n}$ computes an n-variate polynomial f, then there exists a partition A such that the rank of $M(f[A])$ is not full.

We say that a polynomial f has *full rank* if $M(f[A])$ has full rank for every A. Theorem 3.20 thus tells us that every full rank polynomial requires multilinear formulas of super-polynomial size. Now, to prove a lower bound for multilinear formulas, we just need to find a full rank polynomial. We address this issue after sketching the proof of the theorem (we present a slightly different proof than Raz's).

Proof. [Sketch] The first step is to decompose f into "building blocks," which we call log-product polynomials. A polynomial g is *log-product* if it is a product of $t = (\log n)/100$ polynomials $g = g_1 \cdot g_2 \cdots g_t$ so that the set of variables X can be partitioned into t sets $X_1, \ldots, X_t$, each of size at least $n^{1/2}$, where every g_i is defined over the variable-set X_i. Let Φ be a multilinear formula of size $s-1$ computing f.

Lemma 3.21. Every polynomial f computed by a multilinear formula of size s can be written as a sum of $s+1$ polynomials $f = f_1 + \cdots + f_{s+1}$, where each f_i is a *log-product* polynomial.

Proof. [Sketch] Find a gate v in Φ so that $n/3 \leq |X_v| \leq 2n/3$. By induction, Φ_v is a sum of $|\Phi_v|$ log-product polynomials in the variables X_v. As Φ is multilinear it holds that $f = g \cdot \Phi_v + h$, where g is a polynomial in $X \setminus X_v$ and h is the polynomial computed by Φ after labeling v by zero. The structure of f now follows by using the induction hypothesis for h. □

The second part of the proof is the following probability estimate. Choose A uniformly at random from the set of all partitions. For every log-product polynomial $g = g_1 \cdot g_2 \cdots g_t$, the probability of the event $\mathcal{E}(g) = \{\mathrm{rank}(M(g[A])) \geq 2^{n/2 - n^{1/16}}\}$ is at most $n^{-\Omega(t)} = n^{-\Omega(\log n)}$. This probability estimate, together with the union bound and Item (1) from Lemma 3.19 complete the proof of the theorem, since

$$\begin{aligned}\Pr_A[\mathrm{rank}(M(f[A])) = 2^{n/2}] &\leq \Pr_A[\mathcal{E}(f_1) \cup \mathcal{E}(f_2) \cup \cdots \cup \mathcal{E}(f_s)] \\ &\leq s n^{-\Omega(\log n)} \leq 1/2,\end{aligned}$$

for $\varepsilon > 0$ a small enough constant. To prove this probability estimate, we argue that $\mathcal{E} = \mathcal{E}(g)$ is roughly the intersection of $\Omega(t)$ independent events, each of which happens with probability at most order $n^{-3/16}$. We first write this statement (which completes the proof) formally, and then explain it in more detail. The formal statement is

$$\begin{aligned}\Pr[\mathcal{E}] &\le \Pr[\forall\ j \in [t] \quad \mathrm{rank}(M(g_j[A])) \ge 2^{|X_j|/2 - n^{1/16}}] \\ &\le \Pr[\forall\ j \in [t] \quad \big||A(X_j) \cap Y| - |X_j|/2\big| \le n^{1/16}] \\ &\le (cn^{1/16}|X_j|^{-1/2})^t \le n^{-\Omega(t)},\end{aligned}$$

where $c > 0$ is a constant and $X_1, \ldots, X_t$ is the partition of X corresponding to the representation of g as a log-product polynomial. The first inequality holds (from a contra-positive point of view) by properties (3.19) and (3.19) from Lemma 3.19: if there exists $j' \in [t]$ so that $\mathrm{rank}(M(g_{j'}[A])) < 2^{|X_{j'}|/2 - n^{1/16}}$ then

$$\mathrm{rank}(M(g[A])) \le 2^{\sum_{j \neq j'} |X_j|/2} \cdot \mathrm{rank}(M(g_{j'}[A])) < 2^{n/2 - n^{1/16}}.$$

The second inequality holds by property (3.19) from Lemma 3.19, as for any two real numbers α, β we have $\min(\alpha, \beta) = (\alpha + \beta)/2 - |\alpha - \beta|/2$. The third inequality holds by the union bound and standard properties of the hyper-geometric distribution. Finally, the last inequality holds as $|X_j| \ge n^{1/2}$. □

Raz also showed that both determinant and permanent are full rank polynomials (in some, more general, sense), thus proving Theorem 3.16. To be able to view permanent and determinant as full rank polynomials, we define a slightly more general family of partitions. A partition A of an $n \times n$ matrix X is a map from X to $Y \cup Z \cup \{0,1\}$, where $Y = \{y_1, \ldots, y_m\}$, $Z = \{z_1, \ldots, z_m\}$ and $m = [n^{1/3}]$, defined as follows. Let $R_1 = \{r_1(1), \ldots, r_1(m)\}$, $R_2 = \{r_2(1), \ldots, r_2(m)\}$ be two disjoint subsets of $[n]$, and let $C_1 = \{c_1(1), \ldots, c_1(m)\}$, $C_2 = \{c_2(1), \ldots, c_2(m)\}$ be another pair of disjoint subsets of $[n]$. These two pairs correspond to subsets of rows and columns of X, respectively. For every $i \in [m]$, A can be defined as

$$\begin{pmatrix} x_{r_1(i),c_1(i)} & x_{r_1(i),c_2(i)} \\ x_{r_2(i),c_1(i)} & x_{r_2(i),c_2(i)} \end{pmatrix} \to_A \begin{pmatrix} y_i & z_i \\ 1 & 1 \end{pmatrix} \quad \text{or as}$$

$$\begin{pmatrix} x_{r_1(i),c_1(i)} & x_{r_1(i),c_2(i)} \\ x_{r_2(i),c_1(i)} & x_{r_2(i),c_2(i)} \end{pmatrix} \to_A \begin{pmatrix} y_i & 1 \\ z_i & 1 \end{pmatrix}.$$

Denote $\{j_1 < \cdots < j_{n-2m}\} = [n] \setminus (R_1 \cup R_2)$ and $\{\ell_1 < \cdots < \ell_{n-2m}\} = [n] \setminus (C_1 \cup C_2)$. For every $i \in [n-2m]$, set $A(x_{j_i,\ell_i}) = 1$. For every variable $x \in X$ that A is not already defined on, set $A(x) = 0$. So, up to a permutation, the matrix X after the substitution A looks like

$$\begin{pmatrix} B_1 & & & & & \\ & \cdots & & & & \\ & & B_m & & & \\ & & & 1 & & \\ & & & & \cdots & \\ & & & & & 1 \end{pmatrix},$$

where each B_i is a 2×2 matrix such that $\mathrm{DET}(B_i) = y_i - z_i$. For every A, we have $\mathrm{DET}[A] = \prod_{i \in [m]}(y_i - z_i)$, and so the rank of $M(\mathrm{DET}[A])$ is full. It turns out that similar probability estimates to the ones in the proof of Theorem 3.20 hold when we consider the uniform distribution on such partitions instead of the uniform distribution on all partitions.

Although we know how to prove lower bounds for multilinear formulas, we still do not understand well enough their computational power compared to that of general formulas.

Open Problem 14. Are multilinear formulas weaker than general formulas?

We now give a sketch of the proof of Theorem 3.18. In fact we will sketch the proof of the following theorem. Theorem 3.18 follows by exhibiting a polynomial with 0/1 coefficients that have the required properties. We shall not give the details of this construction here.

Theorem 3.22 ([107]). Let Ψ be a syntactically multilinear arithmetic circuit over the field $\mathbb{G}$ and the set of variables $X = \{x_1, \ldots, x_n\}$ computing f. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables (where $n = 2m$ and m is even). If for all partitions A of X to Y and Z $\mathrm{Rank}(M(f[A])) = 2^m$, then $|\Psi| = \Omega(\frac{n^{4/3}}{\log^2 n})$.

Proof. [Sketch] The proof combines the ideas of Theorems 3.20 and 2.5 together with the general theme of looking for a "special" structure of polynomials computed by small circuits.

The first step of the proof is the following analog of Theorem 2.5. The proof resembles the proof of Theorem 2.5 and so we leave it to the reader. Note that the lemma actually guarantees the existence of a syntactic multilinear circuit Ψ' that has more structure than in the case of general circuits.

Lemma 3.23. Let Ψ be a syntactically multilinear arithmetic circuit over the field $\mathbb{F}$ and the set of variables X computing f. Then, there exists an arithmetic circuit Ψ' over the field $\mathbb{F}$ and the set of variables X such that the following hold:

1. Ψ' computes all n partial derivatives $\partial_{x_1}(f),\dots,\partial_{x_n}(f)$.
2. $|\Psi'| = O(|\Psi|)$.
3. Ψ' is a syntactically multilinear arithmetic circuit.
4. For every $i \in [n]$, if v_i is the gate computing $\partial_{x_i}(f)$ in Ψ', then $x_i \notin \mathrm{var}(v_i)$, where $\mathrm{var}(v_i)$ is the set of variables in the sub-circuit rooted at v_i.

Another important ingredient in the proof is the following. Let Φ be a syntactically multilinear arithmetic circuit over the field $\mathbb{F}$ and the variables $X = \{x_1,\dots,x_n\}$. Fix $\tau = 3\log n$. Define $\mathcal{L}(\Phi,\tau)$, the set of *lower leveled* gates in Φ, as the set of all gates u in Φ, such that $2\tau < |\mathrm{var}(u)| < n - 2\tau$, and u has a father u' such that $|\mathrm{var}(u')| \geq n - 2\tau$. The next theorem shows that if a circuit does not have enough lower leveled gates then it computes a (relatively) low rank polynomial.

Theorem 3.24. Let f be a polynomial that is computed by a syntactically multilinear arithmetic circuit Φ, over the field $\mathbb{F}$ and the set of variables $X = \{x_1,\dots,x_n\}$. Let $Y = \{y_1,\dots,y_m\}$ and $Z = \{z_1,\dots,z_m\}$ be two sets of variables (where $n = 2m$ and m is even). Let $\tau = 3\log n$, and let $\mathcal{L} = \mathcal{L}(\Phi,\tau)$ be the set of lower leveled gates in Φ. Let $c > 0$ be a small enough constant ($c = 1/1000$ suffices). Assume $|\mathcal{L}| < \frac{c}{\tau}n^{1/3}$.

Then, there exists a partition A of X to Y and Z such that

$$\mathrm{rank}(M(f[A])) < 2^{m-1}.$$

At first sight this theorem seems very weak as it only guarantees the existence of (roughly) $n^{1/3}$ gates in the lower leveled set. The picture becomes clearer when one considers Lemma 3.23 and the following simple observation.

Observation 3.1. Let A be a partition of X. If $\mathrm{Rank}(M(f[A])) = 2^m$. Then, for every $x_i \in X$

$$\mathrm{rank}(M(\partial_{x_i}(f)[A])) = 2^{m-1}.$$

Our goal will be to show that for every variable x_i the set of lower leveled gates in the sub-circuit computing $\partial_{x_i}(f)$ is large and that these sets are "almost" disjoint.

Indeed, let Ψ' be the arithmetic circuit computing all n partial derivatives of f given by Lemma 3.23. Let $\tau = 3\log n$, and let $\mathcal{L} = \mathcal{L}(\Psi', \tau)$ be the set of lower leveled gates in Ψ'. Define $\mathcal{U} = \mathcal{U}(\Psi', \tau)$, the set of *upper leveled* gates in Ψ', to be the set of all gates $u \in \Psi'$ that have a child in $\mathcal{L}$ and that satisfy $|\mathrm{var}(u)| \geq n - 2\tau$. To prove the theorem, we will bound from below the size of $\mathcal{U}$.

Let $i \in [n]$. Set $g_i = \partial_{x_i} f$. Let v_i be the gate computing g_i in Ψ'. Denote by Ψ'_i the arithmetic circuit Ψ'_{v_i}. Define $\mathcal{L}_i = \mathcal{L}(\Psi'_i, \tau)$ to be the set of lower leveled gates in Ψ'_i. It is not difficult to see that $\mathcal{L}_i \subseteq \mathcal{L}$. In addition, Theorem 3.24 and Observation 3.1 imply that $|\mathcal{L}_i| \geq \frac{c}{\tau} n^{1/3}$. For a gate v in Ψ', define $C_v = |\{i \in [n] : v \text{ is a gate in } \Psi'_i\}|$. For $i \in [n]$, define $\mathcal{U}_i = \{u' \in \mathcal{U} : u' \text{ is a gate in } \Psi'_i\}$. Thus, for all $i \in [n]$, we have $\mathcal{U}_i \subseteq \mathcal{U}$. Hence,

$$\sum_{i \in [n]} |\mathcal{U}_i| = |\{(u', i) : u' \in \mathcal{U} \quad \text{and} \quad i \in [n]$$
$$\text{are such that } u' \text{ is a gate in } \Psi'_i\}| = \sum_{u' \in \mathcal{U}} C_{u'}.$$

Let $i \in [n]$. As $\mathcal{L}_i \subseteq \mathcal{L}$ it follows that for every gate $u \in \mathcal{L}_i$ there is a corresponding gate $u' \in \mathcal{U}_i$, which is a father of u. Thus, since the in-degree of the gates in $\mathcal{U}_i$ is 2, we have $|\mathcal{L}_i| \leq 2|\mathcal{U}_i|$. Recall that, for all $u' \in \mathcal{U}$, it holds that $|\mathrm{var}(u')| \geq n - 2\tau$. Property 4 of Lemma 3.23 now implies that every $u' \in \mathcal{U}$ admits $C_{u'} \leq n - |\mathrm{var}(u')| \leq n - (n - 2\tau) = 2\tau$. Combining all of the above we get

$$\frac{c}{\tau} n^{4/3} \leq \sum_{i \in [n]} |\mathcal{L}_i| \leq 2 \sum_{i \in [n]} |\mathcal{U}_i| = 2 \sum_{u' \in \mathcal{U}} C_{u'} \leq 2|\mathcal{U}| \cdot 2\tau.$$

In particular, $|\mathcal{U}| = \Omega(\frac{n^{4/3}}{\log^2 n})$ and the result follows.

Thus, all that is left to do is to prove Theorem 3.24. This is the main technical difficulty of Raz et al. [107].

Proof. [Sketch of Proof of Theorem 3.24] The proof is based on a structure theorem and a probability estimate. Denote $\mathcal{L} = \{u_1, \ldots, u_\ell\}$. Set $Y_i = Y_{u_i}$ and $Z_i = Z_{u_i}$ to be the set of Y variables and Z variables that appear in the sub-circuit of $\Phi[A]$ rooted at u_i, respectively. Denote with v_f the output gate of Φ.

Theorem 3.25. For every partition A, $f[A]$ can be expressed as $f[A] = \sum_{i \in [\ell]} g_i \Phi_{u_i} + g$, where $g, g_1, \ldots, g_\ell \in \mathbb{F}[Y, Z]$ are multilinear polynomials such that for all $i \in [\ell]$, the set of variables that occur in g_i and the set $Y_i \cup Z_i$ are disjoint, g is the polynomial computed by v_f after substituting (in $\Phi[A]$) every $u \in \mathcal{L}$ by 0 and, furthermore, $\deg(g) \leq 4\tau$.

The proof of the theorem is by induction on the structure of Φ and we leave it to the readers. Another ingredient required for the proof of Theorem 3.24 is the following probabilistic argument whose proof is also left to the readers. We say that a gate v is k-unbalanced if $||Y_v| - |Z_v|| \geq 2k$.

Lemma 3.26. Let Ψ be an arithmetic circuit over the field $\mathbb{F}$ in the set of variables $X = \{x_1, \ldots, x_n\}$. Let $Y = \{y_1, \ldots, y_m\}$ and $Z = \{z_1, \ldots, z_m\}$ be two sets of variables (where $n = 2m$ and m is even). Let

$X_1 \subset X$ be a subset of X of size $n/4$. Let A be a random partition of X to Y and Z, conditioned on the event $A(X_1) \subset Y$. Let β be such that $0 < \beta < 1$, and let v be a gate inΨ such that $n^\beta < |X_v| < n - n^\beta$. Then, for any integer $k \in \mathbb{N}$,

$$\Pr_A[v \text{ is not } k\text{-unbalanced in } \Psi[A] \mid A(X_1) \subset Y] = O(kn^{-\beta/2}).$$

We continue with the proof of Theorem 3.24. The first step is to use Lemma 3.26 together with a "clever" union bound to show that there exists a partition A such that every $u \in \mathcal{L}$ is τ-unbalanced in $\Phi[A]$. In the second step, we use Theorem 3.25 to express $f[A]$ as $f[A] = \sum_{i\in[\ell]} g_i \Phi_{u_i} + g$. Since each $u \in \mathcal{L}$ is τ-unbalanced, Lemma 3.19 implies that $\mathrm{rank}(M(g_i \Phi_{u_i}[A])) \leq 2^{m-\tau}$. As $\deg(g) \leq 4\tau$ it holds that $\mathrm{rank}(M(g[A])) \leq m^{4\tau}$ (this is an upper bound on the number of nonzero rows and columns in the matrix $M(g[A])$). Hence, $\mathrm{rank}(M(f[A])) \leq |\mathcal{L}| \cdot 2^{m-\tau} + m^{4\tau} < 2^{m-1}$, where the last inequality holds as $\tau = 3\log n$ and $m = n/2$. □

This completes the overview of the proof of Theorem 3.22. □

For bounded depth multilinear circuits we can prove much stronger lower bounds, similar to those known for bounded depth Boolean circuits [53].

Theorem 3.27 ([110]). Every depth-d multilinear circuits computing either the permanent or the determinant of an $n \times n$ matrix must be of size $2^{n^{\Omega(1/d)}}$.

In fact, for bounded depth multilinear circuits we have a strong separation result.

Theorem 3.28 ([110]). For every constant d, there is an n-variate polynomial that is computed by a polynomial-size polynomial-degree syntactically multilinear circuit of product-depth[3] d, but every

[3] The product-depth of a gate v is the maximal number of product gates in a directed path reaching v. Depth and product-depth are equivalent up to a factor of two, for circuits of unbounded fan-in.

multilinear circuit of product-depth $d-1$ computing it must be of size at least $n^{\Omega(\log^{1/2d}(n))}$.

We now explain the idea behind the proof of Theorem 3.27. The proof of Theorem 3.28 is more involved and we shall not discuss it here (the lower bound part is similar to the proof of Theorem 3.27, but the construction requires a detailed discussion). To demonstrate the main idea we give a high-level description of the proof of the following theorem.

Theorem 3.29 ([110]). There is a constant $\varepsilon > 0$ so that if a multilinear product-depth d circuit of size at most $2^{n^{\varepsilon/d}}$ computes an n-variate polynomial f, then there exists a partition A such that the rank of $M(f[A])$ is not full. (Here n is large enough compared to d.)

Proof. [High-level] The main idea behind the proof of the $n^{\Omega(\log n)}$ lower bound for the size of multilinear formulas was to express a multilinear formula of size s as a sum of s polynomials, each of which is a product of $t = O(\log n)$ polynomials with at least $n^{1/2}$ variables each, and then applying a random partition to the variables. Then, using probability estimates, we obtained a lower bound of the form $n^{\Omega(t)}$. A similar though more elaborate analysis can be performed for constant depth multilinear circuits as well. In what follows we assume (using Theorem 2.4) that Φ is a syntactic multilinear formula (making Φ a formula may increase the size by a polynomial factor).

Definition 3.2. A polynomial f over the variables $X = \{x_1,\ldots,x_n\}$ is called *d-weak* if for $t = [n^{1/(2d)}]$ one of the following holds:

(1) $f = g_1 \cdot g_2 \cdots g_t$, where for every $i \neq j$ in $[k]$ the two polynomials f_i, f_j are defined over disjoint sets of variables, and for every i in $[k]$ the variable-set of f_i has size at least t.

(2) $f = \ell \cdot g$, where ℓ is a linear form over at least t variables and the variable-set of g is disjoint from that of ℓ.

The following lemma is the basic decomposition required for the lower bound proof.

Lemma 3.30. Let $d \in \mathbb{N}$ and let Φ be a multilinear circuit of product-depth d and size s over n variables computing h. Then there exist d-weak polynomials $h_1,\ldots,h_{s+1}$ so that

$$h = h_1 + h_2 + \cdots + h_{s+1}.$$

Proof. [High-level] The lemma follows by similar arguments to the proof of Theorem 2.7. The basic observation is that (as long as Φ contains at least $n/2$ variables) either

(1) Φ has a product gate v that has at least t children, each computing a polynomial over at least t variables; or
(2) Φ contains a sum gate v that computes a linear function over at least t variables.

This property holds as Φ has product-depth d and is syntactically multilinear. With this property (as in Theorem 2.7), we can write h as

$$h = g_v h_v + h'_v,$$

where h_v is the polynomial computed at v, g_v is a polynomial whose variable-set is disjoint from that of h_v, and h'_v is another polynomial that can be computed by a smaller formula. The lemma now follows by induction on the size of Φ, as the polynomial $g_v \cdot h_v$ is d-weak. □

As in the proof of Theorem 3.20, we will use random partitions. A one-to-one map from X to Y, Z with $|Y| = |X|/2$ is called a *partition*. In the following A is chosen uniformly at random from the family of all partitions.

Lemma 3.31. Let h be a d-weak polynomial. Then

$$\Pr_A[\mathrm{rank}(M(h[A])) \geq 2^{n/2 - t^{1/4}}] \leq 2^{-\Omega(t)}.$$

Proof. Assume that h can be written as $h = g_1 \cdot g_2 \cdots g_t$ as in the definition of d-weak. Similar to the proof of Theorem 3.20, the event $\{\mathrm{rank}(M(h[A])) \geq 2^{n/2-t^{1/4}}\}$ is roughly the intersection of t independent events, the probability of which is at most $1/2$. Otherwise, $h = \ell \cdot g$, and then for all A,

$$\mathrm{rank}(M(h[A])) \leq \mathrm{rank}(M(\ell[A])) \cdot \mathrm{rank}(M(g[A])) \leq 2 \cdot 2^{(n-t)/2}. \qquad \square$$

The two lemmas imply that

$$\begin{aligned}\Pr_A[\mathrm{rank}(M(f[A])) = 2^{n/2}] &\leq \sum_{i \in [s+1]} \Pr_A[\mathrm{rank}(M(f_i[A])) \\ &\geq 2^{n/2}/(s+1)] \leq (s+1)2^{-\Omega(t)} < 1,\end{aligned}$$

where $s \leq 2^{n^{\varepsilon/d}}$ is the size of the circuit for f and $f_1, \ldots, f_{s+1}$ are d-weak polynomials (assuming $\varepsilon > 0$ is small enough). $\square$

In conclusion, we see that strong lower bounds are known for multilinear formulas and for bounded depth multilinear circuits. However, many open problems still remain, most significantly proving super polynomial lower bounds for general multilinear circuits (or even just for syntactically multilinear circuits).

3.7 Circuits with Bounded Coefficients

When computing a polynomial (over, say, the complex numbers) whose coefficients have bounded absolute value, say, at most 1, it is not clear whether the use of large constants can help the computation. In particular it seems unnatural to use large coefficients in order to compute a polynomial that has, say, 0/1 coefficients. This motivates the study of circuits with bounded coefficients. A circuit with *bounded coefficients* is a circuit in which the input gates are labeled by variables and not by field elements, and the edges are labeled by field elements with absolute value at most 1 (any other bound $c \geq 1$ on the absolute value can be translated to 1 by a simple reduction). The computation is performed as before with the notable difference that every edge is multiplied by the constant labeling it. Such circuits were suggested by Morgenstern [94], and were later studied by Chazelle [33] and Raz [101] as well. Morgenstern proved the following lower bound [94].

Theorem 3.32. ([94]) Every circuit with bounded coefficients computing the discrete Fourier transform of $\bar{x} = (x_1, \ldots, x_n)$ must be of size $\Omega(n \log n)$.

This lower bound is tight, as there is a circuit with bounded coefficients of size $O(n \log n)$ computing the Fourier transform [38] (at least when n is a power of two).

Proof. [Sketch] For simplicity, we shall consider computation over $\mathbb{R}$. Consider a circuit Φ computing n linear forms $\langle \bar{\alpha}_1, \bar{x} \rangle, \ldots, \langle \bar{\alpha}_n, \bar{x} \rangle$ in the variables $\bar{x}$, where $\langle \cdot, \cdot \rangle$ denotes the usual inner product in $\mathbb{R}^n$ (note that DFT has exactly this form). Since these are homogeneous linear forms, we can assume without loss of generality that Φ is a *linear circuit*, that is, all computations done in Φ are of the form $\Phi_v = a_1 \Phi_{v_1} + a_2 \Phi_{v_2}$, with $a_1, a_2 \in \mathbb{R}$. As we consider the bounded coefficients model we have that $|a_1|, |a_2| \leq 1$. For every gate v in Φ, we can thus associate a vector $\bar{\beta}_v$ in $\mathbb{R}^n$ so that v computes $\langle \bar{\beta}_v, \bar{x} \rangle$. The main idea behind the proof is to consider the volume of the polytope defined by the linear functions computed at the gates of the circuit. Let $\bar{\alpha}_1, \ldots, \bar{\alpha}_n$ be n vectors in $\mathbb{R}^n$. The *volume* of these n vectors is the absolute value of the determinant of the $n \times n$ matrix whose rows are $\bar{\alpha}_1, \ldots, \bar{\alpha}_n$. We now define a progress measure that does not increase fast for circuits with bounded coefficients. For a linear circuit Ψ, denote by volume(Ψ) the maximum volume of $\bar{\beta}_{v_1}, \ldots, \bar{\beta}_{v_n}$, over all choices of n gates $v_1, \ldots, v_n$ in Ψ. Since the coefficients in Φ are bounded in absolute value by 1, a sum gate can increase the volume, of the circuits computed by its children, by at most a factor of two. Indeed, it follows by a simple induction that the volume of a size s circuit, with coefficients bounded by 1, is at most 2^s. Since the volume of the n vectors defining the Fourier transform is $n^{n/2}$, any circuit with bounded coefficients computing it has size at least $\Omega(n \log n)$. □

Generalizing Morgenstern's ideas, Raz proved the following lower bound [101].

Theorem 3.33 ([101]). Every circuit with bounded coefficients computing the product of two $n \times n$ matrices must be of size $\Omega(n^2 \log n)$.

Note that these two lower bounds hold for polynomials of constant degree. In contrast, no super-linear lower bound is known on the size of general circuits computing constant degree polynomials (recall Open Problem 7).

3.8 Approaches for Proving Lower Bounds

In this section we shortly discuss several general approaches for proving lower bounds. We shall not give the most general treatment of each of these approaches but rather give the main ideas, mostly by considering some specific examples.

3.8.1 Rigidity

The notion of rigidity was introduced by Valiant [137] for the purpose of proving size-depth tradeoffs, namely, showing that a circuit of depth $O(\log n)$ and linear size cannot compute some polynomial f. We will demonstrate this method for the task of computing a linear transformation, although it is more general and can take different forms in different contexts.

We say that a matrix M is (S,R)-*rigid* if by changing at most S entries in every row of M one cannot decrease its rank below R. The integer S is called the *sparsity* parameter and R the *rank* parameter.

Theorem 3.34 ([137]). If an $n \times n$ matrix M is (S,R)-rigid with $S = n^{1/10}$ and $R \geq n/100$ then any circuit computing Mx, for $x = (x_1,\ldots,x_n)$, cannot be of (simultaneously) depth $O(\log n)$ and size $O(n)$.

Proof. [Sketch] Assume toward a contradiction that there exists a circuit Φ of depth $O(\log n)$ and size $s = O(n)$ computing Mx. We can assume without loss of generality that every gate in Φ computes a homogeneous linear form (at the cost of a constant blow-up in size).

Valiant's idea is to show that there exists a set V of gates in Φ of size $|V| = o(s)$, so that after deleting all the edges connected to V, the circuit becomes of depth at most $(\log n)/10$. This is a combinatorial claim on graphs of logarithmic depth and has nothing to do with the function being computed by the circuit. We leave the proof of this claim to the reader.

With this observation in mind, one can argue that M has a "special" structure. Denote by F_i the linear form defined by the i-th row of M. Now, if we delete V from Φ, then we are left with a circuit of depth at most $(\log n)/10$, and therefore each output gate originally computing some F_i is now connected to at most $n^{1/10}$ input gates. We can thus express each F_i as a linear combination of the linear forms $\{\Phi_v\}_{v \in V}$ and of at most $n^{1/10}$ input variables. In other words, the matrix M can be written as a sum of a matrix of rank at most $o(s) = o(n)$ (whose rows are spanned by $\{\Phi_v\}_{v \in V}$) and a sparse matrix having at most $n^{1/10}$ nonzero entries in every row. This is a contradiction, as M is rigid. $\square$

To prove a size-depth tradeoff, it thus suffices to find an explicit *rigid* matrix, a matrix that cannot be expressed as a sum of a low rank matrix and a sparse matrix. Counting arguments imply that most matrices are in fact rigid. However, and as usual, we do not know of any explicit rigid matrix. It seems that the main difficulty in finding a rigid matrix comes from the fact that we need to argue about two different structures at the same time: low rank matrices and sparse matrices. We have a good understanding of each of these families, but we do not understand how to jointly analyze them. For some recent result on rigidity see Ref. [42].

3.8.2 Tensor Rank

The tensor rank approach is quite general as well. Here we give two examples of its possible applicability: a circuit-size lower bound and a formula-size lower bound. We start by defining some notions related to tensors. A three-dimensional *tensor* T is a map $T : [n]^3 \to \mathbb{F}$. A tensor is a higher dimensional version of a matrix: an $n \times n$ matrix M is simply a map $M : [n]^2 \to \mathbb{F}$. A *rank one* tensor is a tensor T so that $T(i_1, i_2, i_3) = t_1(i_1)t_2(i_2)t_3(i_3)$, where t_1, t_2, and t_3 are maps from $[n]$

to $\mathbb{F}$. This coincides with the definition of rank one matrices: a matrix M has rank one if $M(i_1,i_2) = m_1(i_1)m_2(i_2)$. The *rank* of a tensor T is the minimal $R \in \mathbb{N}$ so that $T = \sum_{j=1}^{R} T_j$ with every T_j being a rank one tensor. Tensor rank is a generalization of matrix rank, and it is not difficult to see that, e.g., the rank of a three-dimensional tensor is at most n^2. As the next theorem shows, tensor rank is an important measure that is closely related to the circuit complexity of the underlying tensor.

Theorem 3.35 ([133]). Let T be a three-dimensional tensor of tensor rank at least R. Then the polynomial $F(x,y,z) = \sum_{i,j,k \in [n]} T(i,j,k)x_i y_j z_k$ requires circuits of size at least $\Omega(R)$, over $\mathbb{R}$.

Proof. [Sketch] Let T_i be the matrix defined by $T_i(j,k) = T(i,j,k)$. Consider a circuit Φ computing the n bilinear forms $F_1,\ldots,F_n$ defined by $T_1,\ldots,T_n$, respectively. Namely, $F_i = \sum_{j,k} T_i(j,k)y_j z_k$. By Theorem 2.5, the circuit complexity of $F_1,\ldots,F_n$ is the same as that of F, up to constant factors. By possibly increasing the size by a constant factor we can assume without loss of generality that every gate v in Φ computes a linear form or a bilinear form $f_v = \sum_{j,k} T_v(j,k)y_j z_k$. Consider the set V of product gates $v = u \times w$ in Φ so that f_u and f_w are linear forms. By a homogenization argument, f_u is a linear form in $y_1,\ldots,y_n$ and f_w is a linear form in $z_1,\ldots,z_n$. A simple argument shows that each F_i is a linear combination of $\{f_v\}_{v \in V}$. By simple algebraic manipulations one gets that the tensor rank of T is at most $|V|$. The size of Φ is hence at least $\Omega(|V|) \geq \Omega(R)$. $\square$

We remark that circuits of the kind that was discussed in the proof are called *bilinear circuit*. The best known lower bounds for bilinear circuits for problems such as polynomial multiplication or matrix multiplication are given in Refs. [19, 25, 74, 124].

Tensor rank is a generalization of matrix rank (which we understand pretty well) to higher dimensions. However, we do not know of any explicit tensor of high rank. This shows the striking difference between tensors and matrices. Another evidence of that difference is reflected in the computational complexity of the two notions. Computing the rank of a matrix is fairly easy, e.g., using Gaussian elimination. On the other hand, Håstad proved that computing the tensor rank of a given tensor is NP-hard [54].

Counting arguments show that most three-dimensional tensors are of rank $\Omega(n^2)$ (similar to the existence of hard polynomials). However, the following question is open.

Open Problem 15. Find an explicit three-dimensional tensor whose rank is $\omega(n)$.

We now discuss a recent connection between tensor rank and formula-size lower bounds that was found by Raz [105]. For this we need to consider k-dimensional tensors, that are a straightforward generalization of three-dimensional tensors.

Theorem 3.36. ([105]) Let $T : [n]^k \to \mathbb{F}$ be an explicit tensor (i.e., given $i_1, \ldots, i_k$ we can compute $T(i_1, \ldots, i_k)$ in polynomial time) with $k = O(\log n / \log\log n)$. If the tensor rank of T is at least $n^{k(1-o(1))}$ (in particular, k must tend to infinity as n tends to infinity, since the tensor rank of T is always at most n^{k-1}) then permanent requires formulas of super-polynomial size.

Proof. [Sketch] The proof consists of two different parts. First we analyzes set-multilinear polynomials. A *set-multilinear* polynomial with respect to a set S of size k is a polynomial in k disjoint sets of variables $\{X_i\}_{i \in S}$ so that every monomial that appears in it is of the form $\prod_{i \in S} x_i$ with $x_i \in X_i$. Raz showed that as long as $k \leq O(\log n / \log\log n)$ we can efficiently transform a general formula computing a set-multilinear polynomial to a set-multilinear formula computing the same polynomial. Namely, to a formula so that for every $v = v_1 \times v_2$ in it, the polynomial f_{v_1} is set-multilinear with respect to a subset $S_1 \subseteq S$, the polynomial f_{v_2} is set-multilinear with respect to a subset $S_2 \subseteq S$, and $S_1 \cap S_2 = \emptyset$. In particular, f_v is set-multilinear with respect to $S_1 \cup S_2$.

Theorem 3.37. ([105]) Let Φ be a formula of size s that computes a set-multilinear polynomial with respect to $[k]$. Then there exists a set-multilinear formula Ψ of size at most $(O(\log s))^k s$ that computes the same polynomial.

Proof. The proof is similar to the proof of Theorem 2.3. □

Secondly, we prove that the maximal tensor rank that a set-multilinear formula can compute is relatively small. Note that a tensor $T : [n]^k \to \mathbb{F}$ defines a set-multilinear polynomial f_T in the variables $\{x_{i,j} : i \in [n], j \in [k]\}$ in a natural way: the coefficient of $\prod_{j=1}^{k} x_{i_j,j}$ in f_T is $T(i_1, i_2, \ldots, i_k)$.

Lemma 3.38. Let $T : [n]^k \to \mathbb{F}$ be a tensor with $k \leq O(\log n / \log\log n)$. If there exists a set-multilinear formula of size n^c for the polynomial f_T then the tensor rank of T is at most $n^{k(1-2^{-O(c)})}$.

Proof. [High-level] Raz proved the following intuitive statement. A set-multilinear formula of size n^c that computes the "polynomial with the maximal tensor rank" must have a very specific structure: in particular, the topmost $2^{-O(c)}k$ gates must be product gates. To prove this he used the following three simple properties of tensor rank: (1) the tensor rank of an ℓ-dimensional tensor is at most $n^{\ell-1}$, (2) the tensor rank of $\sum_i T_i$ is at most the sum of the tensor ranks of the T_i's, and (3) the tensor rank of $T_1 \otimes T_2$ is at most the product of the tensor ranks of T_1 and T_2 (if $T_1 : [n]^{k_1} \to \mathbb{F}$ and $T_2 : [n]^{k_2} \to \mathbb{F}$, then $T_1 \otimes T_2 : [n]^{k_1+k_2} \to \mathbb{F}$ is defined by $T_1 \otimes T_2(\bar{i}_1, \bar{i}_2) = T_1(\bar{i}_1)T_2(\bar{i}_2)$). The proof of this structure is too long to include here, but the idea is that properties (2) and (3) above tell us that tensor rank cannot increase too quickly in a formula, and property (1) tells us that we should do the sum operations "as low as possible." This argument could be thought of as a convexity argument.

With this structure, we can write f_T as a product of at least $k' = 2^{-O(c)}k$ polynomials of the form f_{T_i}. Property (1) tells us that the tensor rank of each T_i is at most n^{k_i-1}, where $\sum_i k_i = k$. Now, property (3) implies that the tensor rank of T is at most $n^{k-k'}$, since $T = T_1 \otimes T_2 \otimes \cdots \otimes T_{k'}$, as claimed. □

The theorem and lemma above tell us that if a k-dimensional tensor T with $k \le O(\log n/\log\log n)$ has tensor rank at least $n^{k(1-o(1))}$ then f_T cannot be computed by a polynomial size formula, as otherwise it can also be computed by a polynomial size set-multilinear formula and this is impossible. The conclusion about permanent holds by its completeness and the assumption that T is explicit (see Theorem 1.1). $\square$

Currently, the best lower bound on tensor rank is $\Omega(n^{\lfloor k/2 \rfloor})$ by Nisan and Wigderson [100] for k-dimensional tensors, e.g., the symmetric tensor $T(i_1,\dots,i_k) = 1$ iff $|\{i_1,\dots,i_k\}| = k$. Here is a sketch of the simple proof of such a lower bound: take a tensor T and write it as an $n^{\lfloor k/2 \rfloor}$ by $n^{\lceil k/2 \rceil}$ matrix M by partitioning its k "dimensions" to two disjoint sets. Observe that if T has rank one then M has rank one as well. This implies that the tensor rank of T is at least the rank of the matrix M. Hence, if M has full rank then T has rank at least $n^{\lfloor k/2 \rfloor}$. We note, however, that much tighter lower bounds are required in order to obtain lower bounds for the formula-size of the permanent.

3.8.3 Elusive Functions

We now discuss the Elusive Functions approach for proving circuit-size lower bounds suggested by Raz [103]. Here too we focus on a particular example in order to explain the more general approach.

There are several structure results that we can use as the starting point of the argument. Here we (roughly) use the one given by Theorem 2.8. Let f be a polynomial of degree d in the variables $x_1,\dots,x_n$. Given a circuit Φ computing f, there exist $s \sim |\Phi|$ polynomials $f_1,\dots,f_s$ such that

$$f = \sum_i f_i, \tag{3.1}$$

where every f_i can be written as $f_i = f_{i,1} \cdot f_{i,2}$, where $f_{i,1}, f_{i,2}$ are of degrees at most $2d/3$.

We now wish to understand how the coefficients of the monomials in f_i look like. For every $T = (t_1,\dots,t_n) \in \mathbb{N}^n$ we define the monomial $x_T = x_1^{t_1} \cdots x_n^{t_n}$. We denote $\deg(T) = \sum_k t_k$. Given $T \in \mathbb{N}^n$ and

$j \in \{1,2\}$, let $y_{i,j}(T)$ be the coefficient of the monomial x_T in $f_{i,j}$. Now, think of the $y_{i,j}(T)$'s as new variables and define $Y_i = Y_{i,1} \cup Y_{i,2}$, where $Y_{i,j} = \{y_{i,j}(T) : \deg(T) \leq 2d/3\}$. It follows that the coefficient of x_T in f_i is $\sum_{T_1+T_2=T} y_{i,1}(T_1) \cdot y_{i,2}(T_2)$. In particular, this coefficient is a degree two polynomial in the variables Y_i. This argument shows that each coefficient in f is a degree two polynomial in the variables $Y = \bigcup_i Y_i$. Denote the size of Y by S. We have that $|Y| = S \leq O(n^{2d/3}s)$.

Let $N = \binom{n+d}{d}$. We think of $[N]$ as being the set of all $T \in \mathbb{N}^n$ of degree at most d. Consider the polynomial map φ from $\mathbb{F}^S$ to $\mathbb{F}^N$ that is defined above. Namely, the T-th output of φ is $\sum_{i=1}^{s} \sum_{T_1+T_2=T} y_{i,1} \cdot y_{i,2}$, i.e., the degree two polynomial corresponding to the coefficient of the monomial x_T in f. Now, every substitution of field elements to the variables Y defines a degree d polynomial in the variables $x_1,\ldots,x_n$. We say that this polynomial is in the image of φ. More importantly, every degree d polynomial that has a circuit of size roughly s is contained in the image of φ. In other words, *in order to find an explicit polynomial that cannot be computed by circuits of size s, we just need to find an explicit point in $\mathbb{F}^N$ that is not in the image of the degree two map φ.* Why should such a point exist? Well, if the circuits are not too large, i.e., $S \ll N$, then the input dimension of φ, S, is much smaller than the output dimension, N, so such points must exist (see also Lemma 3.2). Raz called such points *Elusive Functions* as they elude the maps defined by small circuits. To conclude, separating VNP from VP boils down to solving an algorithmic problem: finding a point outside the image of a *given* degree two polynomial map.

The example above considers the problem of eluding degree two polynomial maps. More generally, Raz proved that if one can elude degree r maps $\varphi : \mathbb{F}^S \to \mathbb{F}^N$, for many settings of r, S, N, then a lower bound on the circuit complexity of the permanent follows.

3.8.4 Geometric Complexity Theory

We now address Mulmuley and Sohoni's approach for showing that $\mathsf{VP} \neq \mathsf{VNP}$ [95, 96], which is called *geometric complexity theory* (GCT). For simplicity, we focus on the case where the underlying field is $\mathbb{C}$.

We also focus on a specific goal: showing that permanent cannot be "embedded" in a polynomial size determinant, solving Open Problem 1. To argue this, GCT suggests to look at the symmetries of both determinant and permanent.

Consider the determinant of an $n \times n$ matrix $X = (x_{i,j})$. As a vector of coefficients, it is an N-dimensional vector with N being roughly $\binom{n^2}{n}$, as it is a multilinear homogeneous polynomial of degree n in n^2 variables. Every $n \times n$ matrix U can *act* on N-dimensional vectors by linearly transforming the variables X: Given a polynomial $f(X)$, which is also an N-dimensional vector, we define $U \circ f(X) = f(U^{-1}X)$ (indeed, $(U_1U_2) \circ f = U_1 \circ (U_2 \circ f)$). An important property of determinant (that in fact uniquely defines it) is that if U is in $\mathrm{SL}(n,\mathbb{C})$, that is, $\mathrm{DET}(U) = 1$, then $U \circ \mathrm{DET} = \mathrm{DET}$. Moreover, any U such that $U \circ \mathrm{DET} = \mathrm{DET}$ is in $\mathrm{SL}(n,\mathbb{C})$. In other words, $\mathrm{SL}(n,\mathbb{C})$ is in the *stabilizer* of determinant.

The stabilizer of permanent is different. If we act with $\mathrm{SL}(n,\mathbb{C})$ on permanent, we may change it to a different polynomial. However, every $n \times n$ permutation matrix Π is in the stabilizer of the permanent,

$$\Pi \circ \mathrm{PERM}(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{\Pi(i),\sigma(\Pi(i))} = \sum_{\pi \in S_n} \prod_{i=1}^{n} x_{i,\pi(i)} = \mathrm{PERM}(X).$$

This difference between determinant and permanent is the reason, according to GCT, that the permanent cannot be embedded in a small determinant. In rough terms, permanent is much less symmetric than determinant, its stabilizer is not as rich, and so in order to embed permanent in determinant we must use high dimensions, to be able to fit the disorder. To prove this intuitive claim, GCT suggests to use notions from many other areas of mathematics, such as representation theory, invariant theory, and algebraic geometry. It seems, however, that the current mathematical knowledge is very far from being able to prove even simpler claims.

The appealing part of GCT is that it seems to overcome the barrier of *natural proofs* (see Section 3.9 below). Specifically, it is known that the stabilizers of both determinant and permanent *characterize* them, e.g., the only polynomial f so that $A \circ f = f$ for all A in the stabilizer of determinant is $f = \mathrm{DET}$, and similarly for the permanent. This is highly unusual, in the sense that a random polynomial has a trivial stabilizer, and in particular it is not characterized by it. Intuitively, this means that if indeed we prove a lower bound based on these arguments, it may work only for a small family of polynomials. Thus, (perhaps) overcoming the natural proofs barrier.

3.8.5 Sum-of-Squares Problem

In [60] a new approach for proving lower bounds on the size of noncommutative circuits was given. Specifically, Hrubeš et al. [60] showed a reduction between the noncommutative complexity of permanent to a classical mathematical problem called the sum-of-squares problem, that arises in the areas of topology and algebra. The reader is referred to the introduction of [60] for more background. We give a brief and partial survey of this result.

For an integer n, define the sum-of-square complexity of n as the minimal integer k so that there exist n complex matrices $M_1,\dots,M_n$, each of dimension $n \times k$, so that

(1) for every $i \in [n]$, we have $M_i M_i^t = I$, where I is the $n \times n$ identity and M^t is M transposed,
(2) for every $i \neq j$ in $[n]$, we have $M_i M_j^t = -M_j M_i^t$.

Equivalently, the sum-of-squares complexity of n is the minimal integer k so that there exist complex bilinear forms $f_1,\dots,f_k$, in $\{x_1,\dots,x_n\}$ and $\{y_1,\dots,y_n\}$, so that the following equality holds

$$(x_1^2 + \cdots + x_n^2)(y_1^2 + \cdots + y_n^2) = f_1^2 + \cdots + f_k^2.$$

The following theorem relates sum-of-squares complexity to noncommutative circuit complexity. The sum-of-squares complexity can be defined over any field, but is related to circuit complexity only over fields of characteristic different than two, in which $\sqrt{-1}$ exists.

Theorem 3.39 ([60]). If the sum-of-square complexity of n is at least $\Omega(n^{1+\varepsilon})$ with $\varepsilon > 0$ a constant, then permanent requires noncommutative circuits of exponential size over the complex numbers.

We do not include the proof of this statement here, as it is too technical. We note, however, that Hrubeš et al. [60] also proved lower bounds on the sum-of-squares complexity in some restricted cases. For example, if the matrices M_i are restricted to be integer matrices, then k must be at least $\Omega(n^{6/5})$. We also note that the sum-of-squares problem has a few other equivalent formulations, e.g., as a question about degree four polynomials and as a question about "embedding of unit spheres."

3.9 Natural Proofs for Arithmetic Circuits?

As mentioned before, almost no lower bounds are known for Boolean circuits. A remarkable result of Razborov and Rudich [111] gives a partial explanation for this lack of success. In their work, Razborov and Rudich introduced the concept of *natural proofs* and showed that a natural proof cannot yield a super-polynomial lower bound and that all existing techniques give rise to natural proofs. Specifically, a property $\mathcal{P}$ of Boolean functions from $\{0,1\}^n$ to $\{0,1\}$ is *natural* if it satisfies the following requirements:

(1) "many" functions have $\mathcal{P}$,
(2) it is "easy" to verify whether a given function has $\mathcal{P}$ by looking at its truth table, and
(3) any Boolean circuit computing a function with the property $\mathcal{P}$ has to be of "large" size.

Razborov and Rudich showed that *all* known lower bounds were proved by analyzing natural properties (where the meaning of "many," "easy" and "large," depend on the specific lower bound). The key point in their argument is that a natural proof of a super-polynomial lower bound implies that there are no families of pseudorandom functions. Namely, families of functions that "behave like" the family of all functions, from the point of view of a computationally efficient observer: properties (1) and (2) imply that any family of pseudorandom functions must contain a function with the property $\mathcal{P}$. However, under widely believed complexity assumptions, families of pseudorandom functions that can be computed by small circuits exist, and so condition (3) is violated. Another way of stating this results is: if hard Boolean functions exist (i.e., our hardness assumptions are justified and so pseudorandom functions exist) then it is hard to prove hardness results.

For arithmetic circuits no such argument exists. By considering known lower bounds one sees that they are all "natural" in a sense. For example, the known lower bounds for multilinear complexity and noncommutative complexity are obtained by studying the rank of the partial derivative matrix. Notice that (1) simple counting arguments show that the rank of the partial derivative matrix of almost all polynomials is high; and (2) given a polynomial as a list of coefficients, we can efficiently compute the rank of its partial derivatives matrix.

In order to apply Razborov and Rudich's reasoning all that is required is a construction of a family of pseudorandom polynomials (based, of course, on reasonable hardness assumptions). Unfortunately no such construction is known today. A natural approach would be to try to apply the construction given in Ref. [111], that is based on the construction of Goldreich et al. [48], in the arithmetic setting. The problem with this idea is that the construction of Goldreich et al. is highly sequential and by arithmetizing it one gets a polynomial of an exponentially high degree. For a further discussion see Aaronson's blog post on the topic [1].

Open Problem 16. Give an arithmetic analog of Razborov and Rudich's result. Specifically, give an algebraic analog for pseudorandom function generators. I.e., a family of pseudorandom functions that can be computed by polynomial size (and polynomial degree) arithmetic circuits.

3.10 Meta Lower Bounds

This chapter dealt with the fundamental problem of proving that a certain computational task is hard. Specifically, with the problem of finding explicit polynomials that require large circuits or formulas. We conclude the chapter with a meta discussion of the basic ideas behind this line of research.

When trying to understand what a given computational class (like monotone circuits, multilinear circuits, etc.) can or cannot do, it is natural to try and bring this class into some normal form. The advantage of such a normal form is that it may unveil a certain weakness of the underlying computational class. Finding such a weakness is, in many cases, the key step in proving hardness results. Indeed, this line of thought appeared many times in this chapter. Below is a table that summarizes this approach for different computational classes.

Circuit class	**Normal form**	**Weakness**
Monotone circuits	$f = \sum_i h_i g_i$, each h_i, g_i monotone of degree $\sim r/2$	f has a few monomials
Noncommutative formulas	Arithmetic branching programs	Partial derivative matrix of low rank
$\Sigma\Pi\Sigma$ over $\mathbb{F}_p$	In normal form	Well approximated by a low degree polynomial
Constant depth circuits	In normal form	Vector space of coefficients of low dimension
Multilinear formulas	Sum of log-product polynomials	Partial derivative matrix of low rank, for a random partition
Multilinear circuits	Sum of “structured” polynomials	Partial derivative matrix of low rank for *some* partition
Bounded coefficient	Linear/bilinear circuits	Compute matrices with small determinants
	(Approaches)	
Linear circuits	In normal form	Cannot compute rigid matrices
Bilinear circuits	In normal form	Result has a low tensor rank

4

Polynomial Identity Testing

Polynomial Identity Testing (PIT) is a fundamental problem in algebraic complexity: We are given an arithmetic circuit computing a multivariate polynomial over some field, and we have to determine whether that polynomial is identically zero or not. In other words, we want the polynomial to be formally zero (i.e., that all its coefficients are zero) and not just zero as a function over the field. As a simple example for the difference between these two notions of "zero," consider $x^2 - x$, which is the zero function over $\mathbb{F}_2$ but not the zero polynomial. Note, however, that if the size of the field is higher than the degree of the polynomial then a polynomial is formally zero if and only if it is zero as a function. This is a well-known fact that can be found, e.g., in Ref. [6].

Solving polynomial identities is a central question in both complexity theory and algorithm-design. For example, the best parallel algorithms for finding perfect matchings are based on testing whether a given determinant is formally zero or not [32, 78, 90, 97]. Other algorithms based on identity testing are the primality testing algorithm of [3, 4], algorithms for testing equivalence of read-once branching programs [22], and more. In complexity theory, identity testing played a major role in results such as $\mathsf{IP} = \mathsf{PSPACE}$ [91, 120], MIP = NEXPTIME [15], and the proof of the PCP theorem [10, 11]. Identity testing algorithms also lead to interpolation/learning algorithms for sparse polynomials, see [18, 51, 86] and references within, for depth-3 circuits [125, 76], and for read-once arithmetic formulas [126, 127].

There are two well-studied models in which the PIT problem is considered. The first is the so-called *black-box* model in which the only access to the circuit is by asking for its value on inputs of our choice. It is clear that every deterministic algorithm in the black-box model must produce a *test set* for the circuit, namely, a set of points such that if the circuit vanishes on all points in the set then the circuit computes the zero polynomial. Note that such an algorithm is non-adaptive as it stops at the first nonzero input. The second setting is the *non-black-box* model in which the circuit is given as input. In particular, we have access to the polynomials that are computed at the gates of the circuit. We call this model the *white-box* model. Clearly, the white-box model is the easier amongst the two, although the PIT problem is notoriously hard also for this model.

The difficulty of the PIT problem stems from that the polynomial is not given explicitly as a list of coefficients, but rather implicitly by a circuit or a formula. Converting an implicit representation to an explicit one requires, in general, exponential time and so cannot be performed efficiently. On the other hand, the advantage of such an implicit representation is that the underlying polynomial can be evaluated efficiently on any given input, as long as field operations can be done efficiently. Using this property, randomized (block-box) algorithms were designed for the problem [41, 119, 148]. These randomized algorithms require assignments from a relatively large field. To better understand this, consider a black-box algorithm testing only zero-one assignments. As the "black-box" may contain a polynomial "accepting" only a single input one can easily prove that 2^n queries are required to determine whether it computes the zero polynomial or not. Therefore, when considering PIT over finite fields we allow the algorithm to test assignments from a polynomially large extension field.

The challenge that remains is to design deterministic algorithms for PIT, or, less ambitiously, to reduce the amount of randomness required for solving the problem. The importance of derandomizing PIT follows from its many applications. For example, the famous algorithm for primality testing of Agrawal et al. [4] is based on giving a deterministic algorithm for a specific polynomial identity, using the derandomization ideas of Agrawal and Biswass [3]. Specifically, their algorithm verifies that $(x+a)^n - x^n - a$ is the zero polynomial modulo n, for special values of a (indeed, the identity is zero for every $a \in \mathbb{Z}_n$ if and only if n is a prime). Another example is that derandomization of PIT will imply a deterministic parallel algorithm for finding perfect matchings [97] which is an important open problem. Besides its algorithmic implications, derandomization of PIT may lead to strong lower bounds for arithmetic circuits. In particular, as we shall soon see, if PIT can be solved deterministically in the black-box model, then there exists an "explicit" polynomial that requires exponential size arithmetic circuits.

Determining the complexity of PIT is one of the greatest challenges of theoretical computer science. It is one of a few problems for which we have coRP algorithms [41, 119, 148], but no deterministic sub-exponential time algorithms. A partial explanation for the hardness of obtaining deterministic algorithms was given by Kabanets and Impagliazzo [69], who showed that an efficient deterministic algorithm for PIT, even in the white-box model, implies that NEXP does not have polynomial size arithmetic circuits. Namely, if PIT has polynomial time deterministic algorithms, then either permanent cannot be computed by polynomial size arithmetic circuits or $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$. In Refs. [2, 56] it was observed that a deterministic polynomial time black-box PIT algorithm implies an exponential lower bound for a polynomial whose coefficients can be computed in PSPACE. While this gives a stronger conclusion, we note that the result of [69] holds both in the black-box model and in the white-box model, whereas that of [2, 56] only holds in the black-box model. Kabanets and Impagliazzo also showed an implication in the other direction: a super-polynomial lower bound for the size of arithmetic circuits yields a deterministic sub-exponential time algorithms for PIT. In Ref. [44] an (almost) analogous result to [69] was obtained for bounded depth circuits. These results highlight the tight connection between PIT and lower bounds, and raise the question of obtaining efficient deterministic PIT algorithms for models in which strong lower bounds are known.

We organize this chapter according to what we view as the four main themes in the study of the PIT problem. The first is devising randomness-efficient randomized algorithms for the problem. These algorithms work for general arithmetic circuits. The second research direction is trying to better understand the hardness of derandomizing PIT, mainly by connecting deterministic PIT algorithms to circuit lower bounds. The third theme is the study of PIT for restricted models of arithmetic circuits. Here the main research goal is not the restricted model itself, but rather the understanding of the more general case, via some special instantiations of it. Finally, we discuss results connecting PIT to other problems in algebraic complexity such as polynomial factorization and the isolation lemma (first defined and proved in Ref. [97]).

4.1 Generators and Hitting Sets

We start by defining the basic notions of generators and hitting sets and shortly discuss the relation between them. A polynomial mapping $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_n) : \mathbb{F}^t \to \mathbb{F}^n$ is a *generator* for the circuit class $\mathcal{M}$ if for every nonzero n-variate polynomial[1] $f \in \mathcal{M}$, it holds that $f(\mathcal{G}) \not\equiv 0$. In other words, the polynomial $f \circ \mathcal{G}$ is not formally zero. The *image* of the map $\mathcal{G}$ is $\mathrm{Im}(\mathcal{G}) = \mathcal{G}(\mathbb{F}^t)$. Ideally, t should be very small compared to n. A set $\mathcal{H} \subseteq \mathbb{F}^n$ is a *hitting set* for a circuit class $\mathcal{M}$ if for every nonzero polynomial $f \in \mathcal{M}$, there exists $a \in \mathcal{H}$ such that $f(a) \neq 0$. A generator for $\mathcal{M}$ can also be viewed as a polynomial map whose image contains a hitting set for $\mathcal{M}$ (when the field is large enough). That is, for every nonzero $f \in \mathcal{M}$, there exists $a \in \mathrm{Im}(\mathcal{G})$ such that $f(a) \neq 0$. The following lemma shows that we can efficiently construct a generator given a hitting set and vice versa. From this point on generators are always polynomial maps.

Lemma 4.1. Let $|\mathbb{F}| > n$. Given a hitting set $\mathcal{H} \subseteq \mathbb{F}^n$ for a circuit class $\mathcal{M}$, there is an algorithm that, in time $\mathrm{poly}(|\mathcal{H}|, n)$, constructs

[1] We write $f \in \mathcal{M}$ when f can be computed by a circuit from $\mathcal{M}$.

a map $\mathcal{G} : \mathbb{F}^t \to \mathbb{F}^n$ with $t = \lceil \log_n |\mathcal{H}| \rceil$ that is a generator for $\mathcal{M}$. Furthermore, the individual degrees of each $\mathcal{G}_i$ are bounded by $n-1$.

In the other direction, let $\mathcal{G} : \mathbb{F}^t \to \mathbb{F}^n$ be a generator for a circuit class $\mathcal{M}$ such that the individual degrees of each $\mathcal{G}_i$ are bounded by r. If $\mathcal{M}$ contains polynomials with individual degrees at most D then for every set $S \subseteq \mathbb{F}$ of size $|S| > rD$ it holds that $\mathcal{H} = \mathcal{G}(S^t)$ is a hitting set for $\mathcal{M}$.

Proof. [Sketch] Denote $\mathcal{H} = \{\bar{a}_1, \dots, \bar{a}_h\}$, where $h = |\mathcal{H}|$. Let $\bar{y}_1, \dots, \bar{y}_h$ be different elements in $\mathbb{F}^t$. We shall define $\mathcal{G}$ so that $\mathcal{G}(\bar{y}_i) = \bar{a}_i$. To do so, we need to solve a set of linear equations, in the coefficients of $\mathcal{G}$, that always has a solution due to choice of parameters.

The proof of the other direction follows by the fact that if $f : \mathbb{F}^t \to \mathbb{F}$ has individual degrees at most rD then $f(S^t) \not\equiv 0$. While this is a well-known fact we state it here as we shall use it many times implicitly. The statement that we give is from Ref. [6]. The proof is by a simple induction and is omitted. □

Fact 4.1 ([6]). Let $f(x_1, \dots, x_n)$ be a polynomial over an arbitrary field $\mathbb{F}$. Suppose that the degree of f as a polynomial in x_i is at most r_i, for $1 \le i \le n$. Let $S_i \subseteq \mathbb{F}$ be a set of size at least $r_i + 1$.[2] Then, if f is not formally zero then there exists $(\alpha_1, \dots, \alpha_n) \in S_1 \times S_2 \times \dots \times S_n$ such that $f(\alpha_1, \dots, \alpha_n) \neq 0$.

The following is an immediate and important property of generators.

Observation 4.1. Let $f = f_1 \cdot f_2 \cdots f_k$ be a product of nonzero polynomials $f_i \in \mathcal{M}$ and let $\mathcal{G}$ be a generator for $\mathcal{M}$. Then $f(\mathcal{G}) \not\equiv 0$.

At times, we would like to use only partial substitutions. Given a subset $I \subseteq [n]$, we define the mapping $\mathcal{G}^I$ as $(\mathcal{G}^I)_i = \mathcal{G}_i$ when $i \in I$ and $(\mathcal{G}^I)_i = x_i$ when $i \notin I$. In words, $\mathcal{G}^I$ is the same as $\mathcal{G}$ in entries

[2] This is where we need $\mathbb{F}$ to be "large enough."

in I and is the "original" variables in entries not in I. In addition, in a somewhat abuse of notations, define $f \circ \mathcal{G}^I \overset{\text{def}}{=} f|_{\mathcal{G}^I}$ to be the polynomial resulting from substituting the polynomial $\mathcal{G}_i$ to the variable x_i in f, for every $i \in I$.

Observation 4.2. Let $\mathcal{M}$ be a class of circuits and let $\mathcal{G}$ be a generator for n-variate polynomials in $\mathcal{M}$. Let $I \subseteq [n]$ and $f \in \mathcal{M}$ be a nonzero polynomial. Then $f|_{\mathcal{G}^I} \not\equiv 0$. Moreover, if $|\mathbb{F}|$ is large enough then there exists $a \in \operatorname{Im}\left(\mathcal{G}^I\right)$ such that $f(a) \neq 0$.

Finally, we observe that adding generators for two different circuit classes gives one generator for both classes.

Observation 4.3. Let $\mathcal{G}' : \mathbb{F}^{t'} \to \mathbb{F}^n$ be a generator for $\mathcal{M}'$ and $\mathcal{G}'' : \mathbb{F}^{t''} \to \mathbb{F}^n$ be a generator for $\mathcal{M}''$. Assume (w.l.o.g.) that the zero vector is in the image of both generators.[3] Then $\mathcal{G} : \mathbb{F}^{t'+t''} \to \mathbb{F}^n$ defined as $\mathcal{G}(y', y'') = \mathcal{G}'(y') + \mathcal{G}''(y'')$ is a generator for both $\mathcal{M}'$ and $\mathcal{M}''$.

4.2 Randomized Algorithms

There are several different randomized algorithms for the PIT problem. The so-called Schwartz–Zippel algorithm [41, 119, 148] is based on the observation that by substituting random values to the variables, from a large enough domain, one gets, with high probability, a zero value only if the polynomial is zero. This is perhaps the simplest possible randomized algorithm known and in spite (or perhaps because) of its simplicity it has found many applications in theoretical computer science [91, 97, 120]. The Schwartz–Zippel algorithm has a randomness-error tradeoff; in order to reduce the error the algorithm uses more random bits. Chen and Kao [34] and Lewin and Vadhan [89] designed algorithms that have time-error tradeoff; to achieve smaller error the algorithms need to run for a longer time. These algorithms require less random bits,

[3] This is a small technical requirement that is needed for handling models in which shifting an input by a constant is not permitted (e.g., $\Sigma\Pi$ circuits). Clearly, adding the zero vector to the image can be done at a very small cost.

compared to the Schwartz–Zippel algorithm, but have worse running time.[4] The Chen–Kao algorithm [34] works over the rational numbers, while the Lewin–Vadhan algorithm [89] works for any field. A different algorithmic approach was suggested by Agrawal and Biswass [3]. They map the polynomial computed by the circuit to a univariate polynomial of a very high degree, and then compute this polynomial modulo a random low degree polynomial chosen from a carefully specified family of low degree polynomials. This is a white-box algorithm, as in order to compute the remainder of the polynomial division, one needs to access the intermediate computations performed at the gates of the circuit. The advantage of this approach was demonstrated in Ref. [4], where a deterministic algorithm for primality testing, that follows this scheme was given.

4.2.1 The Schwartz–Zippel Algorithm

As mentioned above, the Schwartz–Zippel algorithm is based on the observation that a nonzero low degree polynomial does not have too many zeros. The proof is by an easy induction on the number of variables and is left to the reader.

Lemma 4.2 ([119, 148]). Let $f(x_1,\dots,x_n)$ be a nonzero polynomial of degree at most r, and let $T \subseteq \mathbb{F}$. If we choose $a = (a_1,\dots,a_n) \in T^n$ uniformly at random, then $\Pr[f(a) = 0] \le r/|T|$.

The lemma suggests a randomized algorithm for PIT: given a degree r polynomial $f(x_1,\dots,x_n)$, pick at random $a \in T^n$ and check whether $f(a) = 0$. If $f \not\equiv 0$, the probability of error is at most $r/|T|$, and if $f \equiv 0$, we are always correct. To achieve error of at most ϵ, we should pick a set T of size $|T| \ge r/\epsilon$. This requires $n \cdot \lceil \log(r/\epsilon) \rceil$ random bits. Another corollary of Lemma 4.2 is that there is a small hitting set for all polynomial size arithmetic circuits. The proof is by a simple application of the union bound.

[4] We note that using expanders graphs one can always obtain time-error tradeoffs (see, e.g., [58]). The advantage of Refs. [34, 89] is that they provide additional insight into the structure of the PIT problem.

Theorem 4.3. For every n, r, s and a field $\mathbb{F}$ of size $|\mathbb{F}| \geq \max(r^2, s)$, there exists a set $\mathcal{H} \subseteq \mathbb{F}^n$ of size $|\mathcal{H}| = \text{poly}(r, s)$ that is a hitting set for all circuits of size at most s and degree at most r.

Proof. [Sketch] We first describe the proof over finite fields. By counting the number of directed acyclic graphs (DAGs for short) we get that there are at most $|\mathbb{F}|^{2s} \cdot s^{2s}$ arithmetic circuits over $\mathbb{F}$ of size at most s (this bound holds for circuits of fan-in two). Let $\mathcal{H}$ be a set of size $4rs$ chosen uniformly at random from $\mathbb{F}^n$. Lemma 4.2 tells us that the probability that a nonzero circuit Φ of degree at most r vanishes on all points in $\mathcal{H}$ is at most $(r/|\mathbb{F}|)^{|\mathcal{H}|} \leq |\mathbb{F}|^{-4rs}$. By the union bound, it follows that there is such a hitting set $\mathcal{H}$ for all nonzero circuits of size at most s and degree at most r.

When $|\mathbb{F}|$ is infinite, say $\mathbb{F} = \mathbb{Q}$, a slightly different argument is used. Here, the set of all polynomials computed by some arithmetic circuit of size s is contained in the union of s^{2s} manifolds, one for each possible DAG, of dimension s each, that are obtained by varying the field constants. One can now prove that when we pick a point at random (say, according to the Gaussian measure) from $\mathbb{F}^n$, the set of polynomials that vanish at that point and that are computed by size s arithmetic circuits is a manifold of dimension at most $s - 1$. Repeating this argument $O(s)$ times we get the required result. $\square$

The hitting set guaranteed by the theorem is not explicit, of course, and the main challenge is to come up with explicit constructions of hitting sets.

4.2.2 Time-Error Tradeoff

We now present the Chen–Kao and Lewin–Vadhan algorithms that give time-error tradeoffs. Namely, to reduce the error the algorithm does not invest more random bits, but rather runs longer. As mentioned earlier, using expander graphs, one can obtain such algorithms in a "generic" way (see Section 1.3.3 in [58]). However, the results of Refs. [34, 89] provide more information on the structure of the PIT problem compared to the generic algorithm and highlight how algebraic independence could be used in this context.

Theorem 4.4 ([34]). Let $f \in \mathbb{Q}[x_1,\dots,x_n]$ have individual degree at most r_i in x_i, for every $i \in [n]$, and denote $r = \max_i r_i$. Assume that each coefficient of f has bit length at most L. Then there is a randomized algorithm that for every $\epsilon > 0$, decides correctly, with probability at least $1-\epsilon$, whether $f \equiv 0$, using $\sum_{i=1}^{n} \lceil \log(r_i+1) \rceil$ random bits and running time $T = \mathrm{poly}((L + r\log(n\log r))/\varepsilon)$.

For comparison, consider the case when f is multilinear. In this case, in order to get error $\epsilon = 1/\mathrm{poly}(n)$, the Schwartz–Zippel algorithm needs $O(n\log n)$ random bits, whereas the Chen–Kao algorithm uses only $O(n)$ random bits, but requires a longer running time.

Proof. [Sketch] The idea behind the algorithm is that if $(a_1,\dots,a_n)$ are algebraically independent over $\mathbb{Q}$ then $f \equiv 0$ iff $f(a_1,\dots,a_n) = 0$. Clearly, algebraically independent numbers are transcendental over $\mathbb{Q}$, so instead of using them, we should pick numbers whose annihilating polynomial is of high degree. However, such numbers must be irrationals and so we cannot use them as inputs either (they have an infinite length bit representation). Nevertheless, one can hope that by truncating the a_i's we still get numbers that low degree polynomials do not vanish on. This is of course not true as for any given (rational) n-tuple we can find a low degree polynomial f that vanishes on it. Chen and Kao solve this by considering the truncation of all conjugates to $(a_1,\dots,a_n)$ and then picking a conjugate at random. The point is that no low degree polynomial can vanish on too many of those conjugates. We now present the formal argument.

Denote $k_i = \lceil \log(r_i+1) \rceil$ and $k = \max_i k_i$. Let $\{p_{i,j}\}_{i\in[n],j\in[k]}$ be the first nk primes. For every $i \in [n]$ and $j \in [k_i]$, let $b_{i,j} \in \{0,1\}$ be a bit, and denote $b = (b_{i,j})_{i,j}$. The basic observation behind the algorithm is that if we set $\pi_i^{(b)} = \sum_{j=1}^{k_i} (-1)^{b_{i,j}} \sqrt{p_{i,j}}$, then Galois theory (see, e.g., Morandi [93]) tells us that

$$f(x_1,\dots,x_n) \equiv 0 \quad \Leftrightarrow \quad f(\pi_1^{(b)},\dots,\pi_n^{(b)}) = 0, \tag{4.1}$$

as the field $\mathbb{Q}(\pi_1^{(b)},\ldots,\pi_{j+1}^{(b)})$ has degree $2^{k_{j+1}}$ over $\mathbb{Q}(\pi_1^{(b)},\ldots,\pi_j^{(b)})$, and so by induction if we know that $f(\pi_1^{(b)},\ldots,\pi_j^{(b)},x_{j+1},\ldots,x_n) \not\equiv 0$, then also $f(\pi_1^{(b)},\ldots,\pi_{j+1}^{(b)},x_{j+2},\ldots,x_n) \not\equiv 0$. Note that $\pi_i^{(b_1)}$ and $\pi_i^{(b_2)}$ are conjugates, for any b_1 and b_2.

Since the $\pi_i^{(b)}$'s are irrational numbers, Chen and Kao proposed the following randomized algorithm. Choose b uniformly at random (i.e., choose random conjugates) and let $q_{i,j}$ be the rounding of $\sqrt{p_{i,j}}$ at the ℓ-th position after the radix point in binary representation (ℓ is the parameter that determines the error). Set $a_i^{(b)} = \sum_{j=1}^{k_i}(-1)^{b_{i,j}}q_{i,j}$. Accept if and only if $f(a_1^{(b)},\ldots,a_n^{(b)}) \neq 0$. Note that the algorithm uses $\sum_i k_i = \sum_i \lceil \log(r_i+1)\rceil$ random bits.

We now give the idea behind the analysis. We will bound from below the value of $f(a_1^{(b)},\ldots,a_n^{(b)})$ as a function of r,ℓ, and n. Galois theory also tells us that the product, over all choices of b, of $f(\pi_1^{(b)},\ldots,\pi_n^{(b)})$ is a nonzero integer. In other words, as long as f is nonzero,

$$\left|\prod_b f^{(b)}\right| \geq 1, \tag{4.2}$$

where $f^{(b)} = f(\pi_1^{(b)},\ldots,\pi_n^{(b)})$. The idea now is to show that each $|f^{(b)}|$ is not too large, which implies that most of the multiplicands cannot be too small in absolute value. Consequently, as $a_i^{(b)}$ is not "too far" from $\pi_i^{(b)}$, the same conclusion holds for $f(a_1^{(b)},\ldots,a_n^{(b)})$, implying that it is nonzero with high probability. We now give a more formal explanation.

By the prime number theorem, $\sqrt{p_{i,j}} \leq O(\sqrt{nk\log(nk)})$. Hence, regardless of b, $|\pi_i^{(b)}| \leq O(k\sqrt{nk\log(nk)})$. As f has degree at most r in each variable, it has at most n^{r+1} monomials. Since the bit length of each coefficient is at most L, we get that for every b,

$$|f^{(b)}| = |f(\pi_1^{(b)},\ldots,\pi_n^{(b)})| \leq n^{r+1}\cdot\left(O(k\sqrt{nk\log(nk)})\right)^r \cdot 2^L \overset{\text{def}}{=} 2^m.$$

Equation (4.2) thus implies that, for every $t > 0$, the fraction of b's for which $|f^{(b)}| \leq 2^{-t}$ is at most $m/(m+t)$. In other words, the fraction of b's for which $|f^{(b)}| > 2^{-t}$ is at least $t/(m+t)$. We now show that for every such b, and an appropriate choice of ℓ, it holds that $f(a_1^{(b)},\ldots,a_n^{(b)}) \neq 0$. Indeed, as $|q_{i,j} - p_{i,j}| \leq 2^{-\ell}$, we have that for each b, $|a_i^{(b)} - \pi_i^{(b)}| \le k2^{-\ell}$. Using the estimate $|x^e - y^e| \le |x - y| \cdot e \cdot \max(x,y)^{e-1}$, we get that

$$\left| f(a_1^{(b)},\ldots,a_n^{(b)}) - f(\pi_1^{(b)},\ldots,\pi_n^{(b)}) \right| \le 2^m \cdot 2^{-\ell+\log r} \le^{(*)} 2^{-t-1},$$

where inequality $(*)$ holds when $\ell \ge m + \log r + t + 1$. For such an ℓ, we thus have

$$\left| f(a_1^{(b)},\ldots,a_n^{(b)}) \right| > \left| f(\pi_1^{(b)},\ldots,\pi_n^{(b)}) \right| - 2^{-t-1} \ge 2^{-t-1} > 0.$$

To better understand the parameters, for $\ell = m/\varepsilon + \log r + 1$ and $t = \ell - m - \log r - 1$, we have that the error probability is at most $m/(m+t) = \epsilon$. The number of random bits used is $\sum_{i=1}^n \lceil \log(r_i + 1) \rceil$ and the running time it takes to "prepare" the inputs $\{a_i^{(b)}\}$ is polynomial in n and in $\ell \le O((L + r\log(n\log r))/\varepsilon)$. □

The algorithm seems to heavily rely on $\mathbb{Q}$ being the underlying field. Lewin and Vadhan [89] showed how to extend it to the case of finite fields as well (in fact their algorithm works over any field). The main idea is to find the "correct" finite field analogs of the notions used by Chen and Kao [34]: irreducible polynomials instead of prime numbers, infinite power series instead of square roots, and truncation modulo x^ℓ instead of truncating after the ℓ-th bit. Since the algorithm and its analysis are not very different than that of Chen and Kao, we refer the reader to Ref. [89] for a proof.

Theorem 4.5 ([89]). Let $f \in \mathbb{F}[x_1,\ldots,x_n]$ be of degree at most r_i in x_i over some finite field $\mathbb{F}$. Then there is a randomized algorithm that, for every $\epsilon > 0$, decides correctly, with probability at least $1 - \epsilon$, whether $f \equiv 0$, using $\sum_{i=1}^n \lceil \log(r_i + 1) \rceil$ random bits and running time $T = \mathrm{poly}(rn/\varepsilon)$. This algorithm can work in the black-box model, given access to a polynomial size extension field of $\mathbb{F}$ (unless $\mathbb{F}$ is already large enough).

4.2.3 The Agrawal–Biswass Algorithm

Assume that Φ is an arithmetic circuit computing a polynomial f of individual degrees smaller than r. Consider the substitution $x_i = y^{r^i}$.

It is not hard to see that $f(x_1,\ldots,x_n) \equiv 0$ iff $f_y \overset{\mathrm{def}}{=} f(y^r,\ldots,y^{r^n}) \equiv 0$. The idea of Agrawal and Biswass [3] is to use the Chinese Remainder Theorem in order to reduce the degree of f_y. Specifically, they find a relatively small set of low degree co-prime polynomials, $\{g_i(y)\}_{i\in I}$, and check whether $f_y \equiv 0 \mod g_i$, for each $i \in I$. The Chinese Remainder Theorem implies that when $\sum_i \deg(g_i) > \deg(f_y)$ it holds that $f \equiv 0 \Leftrightarrow \forall\ i \in I\ \ f_y \equiv 0 \mod g_i$. As the g_i's are of low degree, it is relatively easy to verify whether $f_y \equiv 0 \mod g_i$ when Φ is given to us. Agrawal and Biswass in fact do something more sophisticated. They give a set of polynomials that are not co-prime, but rather the least common multiple of any large enough subset of them (say, of size at least $\epsilon|I|$) has high degree. Their randomized algorithm simply picks one g_i at random, and checks whether $f_y \equiv 0 \mod g_i$. This gives an algorithm whose error is at most ϵ, as the following theorem shows.

Theorem 4.6 ([3]). Let Φ be an arithmetic circuit of size s computing a polynomial f of degree at most r_i in x_i, for every i. For every $\epsilon > 0$, there is a randomized algorithm that, with probability at least $1-\epsilon$, decides correctly whether $f \equiv 0$. The algorithm uses $\lceil \sum_{i=1}^{n} \log r_i \rceil$ random bits and works in time $\mathrm{poly}(n, s, 1/\epsilon, \log(q))$, when the field is of size q. For fields of characteristic zero, a similar result holds, with a dependence on the bit length of the coefficients instead of the field size.

4.3 PIT and Lower Bounds: Hardness-Randomness Tradeoffs

The hardness-randomness paradigm in computer science (roughly) says that hard functions exist iff one can derandomize any (polynomial time) randomized algorithm. The intuition being that if a function is hard to compute then its output looks random (to an efficient algorithm) and vice versa. A prominent example is Impagliazzo and Wigderson's proof that if E requires exponential size Boolean circuits then $\mathsf{BPP} = \mathsf{P}$ [66]. In other words, they showed that if there are hard (explicit) functions then randomness is not essential for computation. As for the other direction, Impagliazzo et al. [65] showed that derandomizing promise-BPP will imply that NEXP $\not\subset$ P/poly. In recent years, similar results were obtained for arithmetic circuits. We shall now review these results and sketch their proofs.

In [69] Kabanets and Impagliazzo showed that derandomizing PIT implies arithmetic lower bounds for NEXP. More precisely, they proved the following theorem.

Theorem 4.7 ([69]). The following three assumptions cannot be simultaneously true.

(1) NEXP $\subseteq$ P/poly.
(2) Permanent is computable by polynomial size arithmetic circuits over $\mathbb{Z}$.
(3) There is a (non-deterministic[5]) sub-exponential time algorithm for PIT.

Stated differently, derandomizing PIT implies either a Boolean lower bound for NEXP or an arithmetic lower bound for permanent. We shall later see a result in the reverse direction as well, namely, that lower bounds for arithmetic circuits imply derandomization of PIT. It is an interesting example where a lower bound gives an upper bound (PIT algorithm) and vice versa. Before sketching the proof, we describe one high-level part of it. Assuming that permanent has small circuits and that we can solve PIT efficiently and deterministically, we can obtain an efficient non-deterministic *uniform* algorithm for permanent: simply guess a small circuit for permanent and then verify its correctness, using the self-reducibility of permanent and the PIT algorithm. As permanent is hard for the polynomial hierarchy, we do not expect it to have an efficient non-deterministic algorithm.

Proof. [Sketch] Consider the Boolean language 0-1-Perm comprising all pairs (M,v), where M is an $n \times n$ matrix with 0/1 entries, v is a string of $O(n\log n)$ bits representing an integer, and $\mathrm{PERM}(M) = v$. Assume for a contradiction that all three assumptions hold. We will reach a contradiction by showing that these assumptions imply that

[5] Recall that PIT is in co-RP, but is not known to be in NSUBEXP.

$\mathsf{co\text{-}NEXP} \subset \mathsf{NSUBEXP} = \cap_{\epsilon>0}\mathsf{NTIME}(2^{\mathsf{n}^{\epsilon}})$, which is false (this can be shown by a diagonalization argument).

Valiant's theorem that 0-1-Perm is $\#\mathsf{P}$-complete [139] implies that if 0-1-Perm is in $\mathsf{NSUBEXP}$ then $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{NSUBEXP}$. In Ref. [65] it was shown that $\mathsf{NEXP} \subseteq \mathsf{P/poly}$ implies that $\mathsf{NEXP} = \mathsf{co\text{-}NEXP} = \mathsf{MA}$. By applying Toda's theorem that $\mathsf{PH} \subset \mathsf{P}^{\#\mathsf{P}}$ [136], it thus follows, under our assumptions, that if 0-1-Perm is in $\mathsf{NSUBEXP}$ then $\mathsf{co\text{-}NEXP} \subset \mathsf{P}^{\#\mathsf{P}} \subset \mathsf{NSUBEXP}$.

It therefore suffices to show that if there is a (non-deterministic) sub-exponential time algorithm for PIT and permanent is computable by polynomial size arithmetic circuits over $\mathbb{Z}$, then 0-1-Perm is in $\mathsf{NSUBEXP}$. The idea is to use the self-reducibility of permanent. For an $n \times n$ matrix $X = (x_{i,j})$, denote by X_j the sub-matrix obtained by deleting the first row and j-th column from X. Thus, $\mathrm{PERM}(X) = \sum_{j=1}^{n} x_{1,j} \cdot \mathrm{PERM}(X_j)$. Now, assuming that permanent has polynomial size arithmetic circuits, we can guess such a circuit Φ_n using non-determinism. Given such a guess Φ_n, construct, for every $1 \leq m \leq n-1$, a circuit Φ_m computing the permanent of an $m \times m$ matrix by substituting zero-one values to the appropriate variables in Φ_n. Now, for every $1 < m \leq n$, write the identity

$$\Phi_m(X^{(m)}) = \sum_{j=1}^{m} x_{1,j}^{(m)} \cdot \Phi_{m-1}(X_j^{(m)}),$$

where $X^{(m)}$ is an $m \times m$ matrix of variables. In addition, consider the identity $\Phi_1(x) = x$. It is not difficult to show that if all these n identities hold, then Φ_n indeed computes permanent. Using the sub-exponential time PIT algorithm, we can verify that all these identities hold, and thus guarantee that Φ_n computes permanent. This shows that 0-1-Perm is in $\mathsf{NSUBEXP}$, which is what we wanted to prove. $\square$

This proof holds in the black-box model as well as in the white-box one. For the case of black-box algorithms, Heintz and Schnorr, and later Agrawal, proved a more straightforward result [2, 56]. The idea behind the proof is that given a small hitting set, we can find a nonzero polynomial that vanishes on all the points of the set. This immediately implies that this polynomial has high circuit complexity.

Theorem 4.8 ([2, 56]). Let $T : \mathbb{N} \to \mathbb{N}$ be a monotone increasing function. If there is a black-box deterministic PIT algorithm that runs in time $T(s)$ for arithmetic circuits of size s, then there is an n-variate polynomial whose coefficients are computed in PSPACE that requires arithmetic circuits of size at least $T^{-1}(\exp(n))$, where $\exp(n)$ means c^n with $c > 1$ and T^{-1} is the inverse function of T.

For example, if $T(s) = \mathrm{poly}(s)$, we get an exponential lower bound for the size of arithmetic circuits, and if $T(s) = \exp(\mathrm{poly}(\log s))$, the lower bound is of the form $\exp(n^{\varepsilon})$.

Proof. [Sketch] Let $\mathcal{H}$ be the hitting set generated by the algorithm. Clearly, $|\mathcal{H}| \leq T(s)$. Using simple interpolation, we can compute in PSPACE the coefficients of a nonzero polynomial $f(y_1,\ldots,y_n)$, with $n = O(\log T(s))$, satisfying $f(a) = 0$ for every $a \in \mathcal{H}$. It is clear that f cannot be computed by a size s circuit, as any circuit of size at most s that vanishes on all points of $\mathcal{H}$ is the zero polynomial. The circuit-size of f is thus at least $s = T^{-1}(\exp(n))$, as required. $\square$

This proof highlights, in a more direct way, the relation between PIT and proving circuit lower bounds. This result, however, does not imply lower bounds for permanent as we are only guaranteed a lower bound for a polynomial whose coefficients are computed in PSPACE. It is an interesting question whether from any black-box PIT algorithm one can get a lower bound for a polynomial in VNP.

Open Problem 17. Assume that an arithmetic circuit class $\mathcal{C}$ has a PIT algorithm that runs in time $\mathrm{poly}(s)$ for circuits of size s. Does the permanent require super-polynomial circuits from $\mathcal{C}$?

So far we have seen that PIT algorithms imply lower bounds. Next we shall see the other direction, that lower bounds for arithmetic circuits imply efficient deterministic PIT algorithms. The first such result was proved by Kabanets and Impagliazzo [69].

Theorem 4.9 ([69]). Let $\mathbb{F}$ be a large enough field (of size at least some polynomial in n) and $s : \mathbb{N} \to \mathbb{N}$ be a monotone increasing function. Assume that a multilinear m-variate polynomial $f(x_1,\ldots,x_m)$ cannot be computed by size $s(m)$ arithmetic circuits over $\mathbb{F}$. Then, there is a deterministic black-box PIT algorithm for arithmetic circuits in n variables, of polynomial degree over $\mathbb{F}$, that runs in time $\exp((s^{-1}(\text{poly}(t)))^2)$ for size $t = t(n) \geq n$ circuits.

To make sense of the parameters, consider the case when we have an $\exp(n)$ lower bound. Then, we get a PIT algorithm for that runs in time $\exp((\log n)^2)$ for $\text{poly}(n)$-size circuits. If the lower bound is $\exp(n^{\epsilon})$, then we get an $\exp(\text{poly}(\log n))$-time PIT algorithm for $\text{poly}(n)$-size circuits. In general, if the lower bound is super-polynomial in n, then the PIT algorithm will be sub-exponential.

Proof. The proof is similar in nature to the proof of Nisan and Wigderson [99] and in particular uses an arithmetic analog of the so-called NW-generator. The next lemma both defines and guarantees the existence of NW-designs. The proof of the lemma can be found in Ref. [99].

Lemma 4.10. Let n,m be integers such that $n < 2^m$. There exists a family of sets $S_1,\ldots,S_n \subset [k]$ with $k = O(m^2/\log(n))$ so that (1) for each $i \in [n]$ it holds that $|S_i| = m$, and (2) for every $i \neq j$ in $[n]$ we have that $|S_i \cap S_j| \leq \log(n)$. This family of sets can be constructed deterministically in time $\text{poly}(n, 2^k)$. Such a family is called an *NW-design*.

Given such an NW-design and the "hard" polynomial f, we construct the hitting set as follows. Let $m = m(t)$ be such that $s(m) > t^{O(1)}$, the exact value of the $O(1)$ in the exponent will be determined later. Consider an NW-design $S_1,\ldots,S_n$ with parameters m and n (note that $n < 2^m$, as f is multilinear and hence s is at most exponential). Define the map $G : \mathbb{F}^k \to \mathbb{F}^n$ as $G(y) = (f(y|_{S_1}),\ldots,f(y|_{S_n}))$, where $y|_{S_i}$ is the restriction of y to the entries in S_i, namely, $y|_{S_i} = (y_{j_1}, y_{j_2},\ldots,y_{j_m})$ with $S_i = \{j_1 < j_2 < \cdots < j_m\}$.

We now show that for any size t circuit Φ, it holds that $\Phi \equiv 0$ iff $\Phi(G) \equiv 0$. One direction is trivial. The other direction is proved using the "hybrid" method. Assume for a contradiction that $\Phi(G) \equiv 0$ but $\Phi \not\equiv 0$. Then, there exists $i \in [n]$ such that

$$\Phi(f(y|_{S_1}),\ldots,f(y|_{S_i}),x_{i+1},\ldots,x_n) \not\equiv 0 \quad \text{but}$$
$$\Phi(f(y|_{S_1}),\ldots,f(y|_{S_{i+1}}),x_{i+2},\ldots,x_n) \equiv 0.$$

As $\mathbb{F}$ is large enough, by fixing the values of $x_{i+2},\ldots,x_n$ to appropriate field elements $a_{i+2},\ldots,a_n$, we get that there is a circuit Ψ in $k+1$ variables such that

$$\Psi(y,z) \stackrel{\text{def}}{=} \Phi(f(y|_{S_1}),\ldots,f(y|_{S_i}),z,a_{i+2},\ldots,a_n) \not\equiv 0 \quad \text{but}$$
$$\Psi(y,f(y|_{S_{i+1}})) \equiv 0.$$

Without loss of generality, assume that $S_{i+1} = [m]$. We can further fix values $b_{m+1},\ldots,b_k$ to $y_{m+1},\ldots,y_k$ such that

$$\Psi'(y_1,\ldots,y_m,z) \stackrel{\text{def}}{=} \Psi(y_1,\ldots,y_m,b_{m+1},\ldots,b_k,z) \not\equiv 0.$$

Here is the point where we use the design property: as for every $j < i+1$ we have $|S_j \cap S_{i+1}| \le \log n$, each of the polynomials $f(y|_{S_j})\big|_{y_{m+1}=b_{m+1},\ldots,y_k=b_k}$ has a circuit of size $\mathrm{poly}(n)$ (it is a multilinear polynomial with $\le \log n$ variables). Therefore, Ψ' has a circuit of size $t + \mathrm{poly}(n)$. Since

$$\Psi'(y_1,\ldots,y_m,z) \not\equiv 0 \quad \text{but} \quad \Psi'(y_1,\ldots,y_m,f(y_1,\ldots,y_m)) \equiv 0,$$

we get that $z - f(y_1,\ldots,y_m)$ is a factor of Ψ'. The following factoring result by Kaltofen [71] now tells us that $f(y_1,\ldots,y_m)$ has a circuit of size $\mathrm{poly}(t(n) + \mathrm{poly}(n)) = t^{O(1)}$.

Theorem 4.11. There is a probabilistic polynomial-time algorithm that gets as input an arithmetic circuit of size s computing a polynomial $g \in \mathbb{F}[x_1,\ldots,x_n]$ of total degree at most r and outputs, with probability at least 3/4, integers $e_1,e_2,\ldots,e_{r'} > 0$ and circuits $\Phi_1,\Phi_2,\ldots,\Phi_{r'}$, each of size at most $\mathrm{poly}(s,r,\log|\mathbb{F}|)$, computing irreducible polynomials $h_1,h_2,\ldots,h_{r'}$ such that $g = \prod_{i=1}^{r'} h_i^{e_i}$. In case the characteristic q of $\mathbb{F}$

divides any e_i, i.e., $e_i = q^{k_i} e'_i$ with e'_i not divisible by q, the algorithm returns e'_i instead of e_i, and the corresponding arithmetic circuit computes $h_i^{q^{k_i}}$ instead of h_i. For $\mathbb{F} = \mathbb{Q}$, the size of the produced arithmetic circuits is at most $\text{poly}(s,r,L)$, where L is the size of the maximal coefficient in g.

The polynomial $f(y_1,\dots,y_m)$ thus has an arithmetic circuit of size $t^{O(1)}$. The lower bound on f implies that $t^{O(1)} > s(m)$, but this contradicts our choice of m. We conclude that $\Phi \equiv 0$ iff $\Phi(G) \equiv 0$. As f and Φ have degrees $\text{poly}(n)$, we get that $\Phi(G)$ has degree at most $\text{poly}(n)$. Thus, if we evaluate $\Phi(G)$ on all points in A^k, where $A \subseteq \mathbb{F}$ is a set of size $\text{poly}(n)$, we get that $\Phi \equiv 0$ iff $\Phi(G(A^k)) = \{0\}$. This gives a hitting set of size $\text{poly}(n)^k = \text{poly}(n)^{m^2/\log n} = \exp((s^{-1}(t^{O(1)}))^2)$. □

Using the ideas of the theorem above and of Theorems 2.11 and 4.8, Agrawal and Vinay proved that a polynomial size hitting set for depth-4 circuits gives rise to a quasi-polynomial size hitting set for general arithmetic circuits [5].

Theorem 4.12 ([5]). If for every s there is an efficient way to construct a hitting set of size $\text{poly}(s)$ for $\Sigma\Pi\Sigma\Pi$ arithmetic circuits of size s, then in time $\exp(\log^2(s))$ one can construct a hitting set for general arithmetic circuits of size s.

Proof. [Sketch] The idea is that if we have a polynomial size hitting set for depth-4 circuits, then Theorem 4.8 guarantees a polynomial that requires exponential size depth-4 circuits. Theorem 2.11 and the discussion following it imply that this polynomial actually requires exponential size general arithmetic circuits. By Theorem 4.9, this implies the existence of a quasi-polynomial size hitting set for general circuits. □

The following question is thus as interesting as the general PIT question.

Open Problem 18. Derandomize the black-box PIT problem for $\Sigma\Pi\Sigma\Pi$ arithmetic circuits.

The result of Agrawal and Vinay requires an $\exp(n)$ lower bound for depth-4 circuits in order to obtain exponential lower bounds for general circuits and thus derandomize PIT. But, what can we conclude if we only have lower bounds of the form $\exp(n^{\varepsilon})$ for depth-4 circuits, or even just super-polynomial lower bounds? What if we have lower bounds for a larger depth? A close inspection of the proof of Theorem 4.9 shows that the use of Kaltofen's factoring algorithm requires the lower bound to hold for general arithmetic circuits and not just for bounded depth circuits (this is the case since Kaltofen's factoring algorithm does not preserve depth). A partial answer was given in Ref. [44] where it was shown that exponential lower bounds for depth-d circuits imply quasi-polynomial time deterministic algorithms for PIT of depth-$(d-5)$ circuits, assuming that the polynomial computed by the circuit has individual degrees at most $\mathrm{poly}(\log n)$. The idea behind this result is to replace Kaltofen's factoring algorithm with the following root finding result that basically says that if $z - f(x_1,\ldots,x_n)$ is a factor of $\Psi(x_1,\ldots,x_n,z)$, where Ψ is a small depth-d circuit with a small degree in z, then f can be computed by a small arithmetic circuit of depth at most $d+3$.

Theorem 4.13. Let n,s,r,m,t,d be integers such that $s \geq n$. Let $\mathbb{F}$ be a field which has at least $\max\{mt+1, r+1\}$ elements. Let $g(x_1,\ldots,x_n,z) \in \mathbb{F}[x_1,\ldots,x_n,z]$ be a nonzero polynomial with $\deg(g) \leq t$ and $\deg_z(g) \leq r$ (i.e., the degree of z in g is at most r) that has a depth-d arithmetic circuit of size s over $\mathbb{F}$. Let $f(x_1,\ldots,x_n) \in \mathbb{F}[x_1,\ldots,x_n]$ be a polynomial with $\deg(f) \leq m$ such that $g(x_1,\ldots,x_n,f(x_1,\ldots,x_n)) \equiv 0$. Then $f(x_1,\ldots,x_n)$ can be computed by a circuit of size $\mathrm{poly}(s,m^r)$ and depth $d+3$ over $\mathbb{F}$.

We do not prove this theorem here but only give the idea behind the proof. Newton's method (also known as Hensel's lifting lemma) provides a general way for finding roots of a given function. In our case, we apply this method to the polynomial g with the intention of finding the root f of g. The proof follows by showing that, with some careful manipulation, Newton's method actually implies that the complexity of f is not much larger than that of g.

Given the theorem above one can follow the proof of Theorem 4.9 to conclude the following analog for the case of bounded depth circuits.

Theorem 4.14 ([44]). Let d be an integer, $\epsilon > 0$ a real number and $\mathbb{F}$ a finite field. Let $f \in \mathbb{F}[x_1,\ldots,x_n]$ be an explicit polynomial of degree $\mathrm{poly}(n)$ such that any depth-d circuit computing it has size at least $2^{n^{\epsilon}}$. Then, the black-box PIT problem for polynomial-size depth-$(d-5)$ circuits, having individual degrees at most $\mathrm{poly}(\log n)$, can be solved deterministically in time $\exp(\mathrm{poly}(\log n))$.

It is an intriguing question whether one can factor polynomials, or even just compute factors that are linear in some variable, without increasing the depth by much.

Open Problem 19. Let $f(x_1,\ldots,x_n,z)$ be computed by a depth-d size s circuit. Is it true that any factor of f can be computed by a depth-$O(d)$ size $\mathrm{poly}(s)$ circuit? What about factors that are linear in z?

The results that were described in this section give tight relations between PIT and lower bounds. In Section 3.6 we saw that quasi-polynomial lower bounds are known for multilinear formulas, and that for constant-depth multilinear formulas exponential lower bounds are known. Unfortunately, the results of Refs. [44, 69] require lower bounds for general circuits and not just for multilinear circuits. We therefore would like to understand whether one can use lower bounds for weak models to obtain PIT algorithms (for the corresponding weak models). As we shall see in Section 4.8, Shpilka and Volkovich [126, 127] use a lower bound for sums of read-once formulas, to obtain a PIT algorithm for that model. It is a very interesting question to understand whether this can be achieved for other models as well.

Open Problem 20. Is there a way to transform lower bounds for multilinear circuits to PIT algorithms for some model of multilinear circuits or formulas? What if we have a (slightly) super-polynomial lower bound for arithmetic formulas, can we use it to get a sub-exponential time PIT algorithm for arithmetic formulas?

As we shall see in Section 4.5, Raz and Shpilka [106] obtained a polynomial time (white-box) PIT algorithm for noncommutative formulas. Their proof does not transform the lower bound of Nisan [98], discussed in Section 3.4, to a PIT algorithm but rather relies on the intuition gained in the proof of the lower bound to design the algorithm. Therefore, we may hope that even if there is no clear way of transforming lower bounds for multilinear formulas into PIT algorithms, that we can still use the intuition gained in the lower bounds proofs to design such algorithms.

Open Problem 21. Give a deterministic sub-exponential time identity testing algorithm for multilinear formulas.

4.4 Sparse Polynomials

Sparse polynomials are polynomials having a small number of monomials. We will be mostly interested in the case when the number of monomials m is polynomial in the number of variables. In other words, we will be interested in polynomials that have polynomial size $\Sigma\Pi$ circuits. This model is of course the simplest and "most explicit" way for computing and representing a polynomial: the polynomial is given by its list of coefficients. As such, the polynomial identity testing problem for this model attracted a lot of research and many algorithms were devised for it, see Refs. [86, 20] and references therein.

In this section we give the proof of Klivans and Spielman [86] for identity testing of sparse polynomials. The main idea is to reduce the multivariate problem to a univariate problem. Namely, to generate from each sparse polynomial f a univariate polynomial $\hat{f}$ and then check whether $\hat{f} \equiv 0$.

Assume that the unknown polynomial is $f(x_1,\ldots,x_n)$ and that its (total) degree is smaller than r. Consider the substitution $x_i = y^{r^i}$. Any two different monomials of f are mapped to different powers of y, and so $\hat{f}(y) \overset{\mathrm{def}}{=} f(y^r,\ldots,y^{r^n}) \equiv 0$ iff $f \equiv 0$. Therefore, all that we have to do now is to check whether a univariate polynomial is identically zero or not. This can be easily achieved by querying $\hat{f}$ on $\deg(\hat{f})$ many points. The problem with this approach is that $\deg(\hat{f})$ can be exponentially large. Recall that this is the same problem that faced Agrawal and Biswass in Section 4.2.3. Here, Klivans and Spielman handle this by picking some prime $p = O(\max(r, mn+1))$ and making the substitution $x_i = y^{r^i \bmod p}$. The problem now is that different monomials in f can be mapped to the same monomial in $\hat{f}$ and can therefore cancel each other. To overcome this difficulty, they consider all substitutions of the form $x_i = y^{k^i \bmod p}$ for all integers $k \in [mn+1]$. They show that for at least one of these k's, at least one monomial of f will be uniquely mapped to a monomial of $\hat{f}$, and this will give that $f \equiv 0$ iff $\hat{f} \equiv 0$, with the additional property $\deg(\hat{f}) < rp$. By evaluating $\hat{f}$ in rp many points we can check whether $f \equiv 0$. We now give the formal details of the proof.

Theorem 4.15 ([86]). Let f be a nonzero n-variate polynomial of individual degrees smaller than r, that has (at most) m monomials over $\mathbb{F}$. Let p be a prime larger than $\max(r, mn+1)$. Then, there is some $k \in [mn+1]$ such that the univariate polynomial $\hat{f}(y) \overset{\mathrm{def}}{=} f(y, y^{k^1 \bmod p}, \ldots, y^{k^{n-1} \bmod p})$ is not identically zero, and of degree at most $rp - 1$. Hence, if $S \subset \mathbb{F}$ is of size at least rp then evaluating f on all points of the form $(y, y^{k^1 \bmod p}, \ldots, y^{k^{n-1} \bmod p})$ for $y \in S$ is a hitting set for f. In particular, when m and r are polynomial in n we get a deterministic polynomial time PIT algorithm by testing this for every $k \in [mn+1]$.

Proof. Let $f = \sum_{j=1}^{m} c_j x^{r(j)}$, where $r(j) = (r(j)_1,\ldots,r(j)_n)$ and $x^{r(j)} = x_1^{r(j)_1} x_2^{r(j)_2} \cdots x_n^{r(j)_n}$. Consider the substitution $x_i = y^{k^i \bmod p}$. The monomial $x^{r(j)}$ is mapped to $y^{\sum_{i=1}^{n} r(j)_i (k^{i-1} \bmod p)}$. We will show that for some choice of $k \in [mn+1]$, there exists $j \in [m]$ such that the monomial $x^{r(j)}$ is mapped uniquely to $y^{\sum_{i=1}^{n} r(j)_i (k^{i-1} \mod p)}$, which implies that $\hat{f}(y) \not\equiv 0$. To see this, observe that if $x^{r(j)}$ and $x^{r(j')}$ are mapped to the same y monomial, then $\sum_{i=1}^{n} r(j)_i (k^{i-1} \mod p) = \sum_{i=1}^{n} r(j')_i (k^{i-1} \mod p)$. The last equality implies that for any such pair $j \neq j'$, it holds that $\sum_{i=1}^{n} r(j)_i k^{i-1} \overset{\text{mod p}}{=} \sum_{i=1}^{n} r(j')_i k^{i-1}$. As $r < p$, it holds that $r(j) \neq r(j')$ modulo p. Thus, a bad "event" can happen only if k is a root of the nonzero polynomial $g_{j,j'}(z) \overset{\text{def}}{=} \sum_{i=1}^{n} (r(j)_i - r(j')_i) z^{i-1}$ over $\mathbb{F}_p$. Fix some j_0 such that $c_{j_0} \neq 0$. There are m different polynomials $g_{j_0,j}(z)$. As $\deg(g_{j_0,j}(z)) \leq n-1$, it follows that there are at most mn different values of k that can be roots of any of these polynomials. Since $mn + 1 < p$, there exists some $k \in [mn+1]$ that is not a root of any of the $g_{j_0,j}(z)$'s. For this k, we get that $x^{r(j_0)}$ was mapped uniquely to some y monomial. □

The bit length of the points on which we evaluate f can be as large as $m \log m$, however, using the Chinese Remainder Theorem this can be reduced to $O(\log nr)$. We omit the details of this reduction.

As mentioned above, there were other deterministic PIT algorithms for sparse polynomials prior to Ref. [86]. The main difference between the various algorithms is the bit length of the points on which one evaluates the polynomial and the exact size of the hitting set constructed. We refer the interested reader to Ref. [86] where these issues are discussed.

In [20] a slight twist of the above result was considered in which we do not have a polynomial bound on the degree of f but instead we are assured that f (which is sparse) is computed by a size s circuit (that does not necessarily have small depth). This implies that $\deg(f) \leq 2^s$. Note that the size of the hitting set constructed in Theorem 4.15 is polynomial in the degree, which is not efficient when $s = \text{poly}(n)$. Bläser et al. gave a deterministic PIT algorithm, with running time $\text{poly}(m,n,s)$, for such circuits [20]. The algorithm is black-box over $\mathbb{Z}$ and white-box over finite fields. Their black-box algorithm basically evaluates the polynomial on points of the form $(k^{r^0}, \ldots, k^{r^{n-1}}) \mod p$ for several different primes p and all points $k \in \mathbb{F}_p$.[6]

[6] In fact, the size of the hitting set over $\mathbb{Z}$ is polynomial in L, which is an upper bound on the bit length of the coefficients of the underlying polynomial.

4.5 Noncommutative Formulas

In Ref. [106] a deterministic, polynomial time, white-box algorithm for PIT of noncommutative formulas was given. Besides the model of sparse polynomials, this is the only "complete" model of computation for which polynomial time deterministic PIT algorithms are known. This algorithm also plays a role in Saxena's [116] PIT algorithm for the so-called diagonal circuits, see Section 4.7.1.

The algorithm of Raz and Shpilka [106] relies on the fact, shown in Section 3.4, that a noncommutative formula can be simulated efficiently by an ABP. The algorithm, in fact, works also for the more general model of noncommutative ABP's. We first give a sketch of the argument. Given an ABP $\mathcal{A}$, computing a polynomial $f(x_1,\ldots,x_n)$, we can construct, for every monomial of the form x_ix_j (i may be equal to j), a new ABP that computes the part of f containing all monomials that "start" with x_ix_j. Since we are working in the noncommutative polynomial ring, $\mathcal{A}$ computes the zero polynomial iff all the new ABPs compute the zero polynomial as well. We now explain how to merge the new ABPs to a single ABP $\hat{\mathcal{A}}$, which has one less level, such that $\hat{\mathcal{A}}$ is identically zero iff $\mathcal{A}$ is identically zero. $\hat{\mathcal{A}}$ is constructed by removing from $\mathcal{A}$ the vertices of level 1 and the edges adjacent to them, and then connecting the source directly to level 2. The technical part is the labeling of the new edges, going out of the source, with the "correct" linear functions (recall that every edge is labeled with a linear form). The naive way of replacing each term of the form x_ix_j with a new variable $y_{i,j}$ does not work, as in this way the number of variables introduced after k steps can be as large as n^{2k}. The main point in the argument is that the number of new variables is never larger than $O(s+n)$, and this is achieved by the "correct" labeling. We now give a more formal explanation.

Let $\mathcal{A}$ be an ABP with levels $0,\ldots,d$. Denote by m_1 and m_2 the number of vertices in levels 1 and 2 of $\mathcal{A}$, respectively. Denote by v_{in} the source of $\mathcal{A}$ and by v_{out} the sink of $\mathcal{A}$. Let $v_1,\ldots,v_{m_2}$ be the vertices in level 2 of $\mathcal{A}$. Let $\mathcal{A}(v,v_{\text{out}})$ be the polynomial that is computed by the ABP with the vertex v as source and v_{out} as sink. By definition,

$$\mathcal{A}(v_{\text{in}},v_{\text{out}}) = \sum_{i\in[m_2]} \mathcal{A}(v_{\text{in}},v_i)\cdot\mathcal{A}(v_i,v_{\text{out}}).$$

For every $i \in [m_2]$, we can express $\mathcal{A}(v_{\text{in}}, v_i)$ as

$$\mathcal{A}(v_{\text{in}}, v_i) = \sum_{1 \le k \le n} \alpha_{i,k} {x_k}^2 + \sum_{1 \le j < k \le n} \beta_{i,j,k} x_j x_k + \sum_{1 \le j < k \le n} \gamma_{i,j,k} x_k x_j.$$

We thus have

$$\begin{aligned} \mathcal{A}(v_{\text{in}}, v_{\text{out}}) = & \sum_{1 \le k \le n} {x_k}^2 \sum_{i \in [m_2]} \alpha_{i,k} \cdot \mathcal{A}(v_i, v_{\text{out}}) \\ & + \sum_{1 \le j < k \le n} x_j x_k \sum_{i \in [m_2]} \beta_{i,j,k} \cdot \mathcal{A}(v_i, v_{\text{out}}) \\ & + \sum_{1 \le j < k \le n} x_k x_j \sum_{i \in [m_2]} \gamma_{i,j,k} \cdot \mathcal{A}(v_i, v_{\text{out}}). \end{aligned}$$

The next claim now follows, as we are working in the noncommutative polynomial ring.

Claim 4.16. $\mathcal{A}$ computes the zero polynomial iff for every $1 \le k \le n$,

$$\sum_{i \in [m_2]} \alpha_{i,k} \cdot \mathcal{A}(v_i, v_{\text{out}}) \equiv 0,$$

and for every $1 \le j < k \le n$,

$$\sum_{i \in [m_2]} \beta_{i,j,k} \cdot \mathcal{A}(v_i, v_{\text{out}}) \equiv \sum_{i \in [m_2]} \gamma_{i,j,k} \cdot \mathcal{A}(v_i, v_{\text{out}}) \equiv 0.$$

For simplicity of notation, define the following vectors

$$\begin{aligned} &\alpha_k = (\alpha_{1,k}, \alpha_{2,k}, \ldots, \alpha_{m_2,k}), && 1 \le k \le n, \\ &\beta_{j,k} = (\beta_{1,j,k}, \beta_{2,j,k}, \ldots, \beta_{m_2,j,k}), && 1 \le j < k \le n, \\ &\gamma_{j,k} = (\gamma_{1,j,k}, \gamma_{2,j,k}, \ldots, \gamma_{m_2,j,k}), && 1 \le j < k \le n, \quad \text{and} \\ &a = (\mathcal{A}(v_1, v_{\text{out}}), \mathcal{A}(v_2, v_{\text{out}}), \ldots, \mathcal{A}(v_{m_2}, v_{\text{out}})). \end{aligned}$$

The claim tells us that $\mathcal{A}(v_{\text{in}}, v_{\text{out}}) \equiv 0$ iff for every $1 \le j < k \le n$,

$$\langle \alpha_k, a \rangle \equiv \langle \beta_{j,k}, a \rangle \equiv \langle \gamma_{j,k}, a \rangle \equiv 0,$$

where $\langle \cdot, \cdot \rangle$ is the inner product form. Consider $V = \text{span}\{\alpha_k, \beta_{j,k}, \gamma_{j,k}\}_{j,k}$. Stated differently, $\mathcal{A}(v_{\text{in}}, v_{\text{out}}) \equiv 0$ iff $a \perp V$. Let

$d = \dim(V) \leq m_2$ and $u_1, \ldots, u_d$ be a basis for V. Note that as we work in the white-box model, we can easily compute $\alpha_k, \beta_{j,k}, \gamma_{j,k}$ and hence we can efficiently compute a basis of V. Denote $u_j = (u_{j,1}, \ldots, u_{j,m_2})$. For $i \in [m_2]$, define the linear form

$$\ell_i(y_1, \ldots, y_d) = \sum_{j \in [d]} u_{j,i} y_j.$$

We thus have

$$\hat{\mathcal{A}} \overset{\text{def}}{=} \sum_{i \in [m_2]} \ell_i \cdot \mathcal{A}(v_i, v_{\text{out}}) = \sum_{j \in [d]} y_j \sum_{i \in [m_2]} u_{j,i} \cdot \mathcal{A}(v_i, v_{\text{out}}) = \sum_{j \in [d]} y_j \langle u_j, a \rangle,$$

which implies

$$\hat{\mathcal{A}} \equiv 0 \quad \Leftrightarrow \quad \forall\, j \in [d] \quad \langle u_j, a \rangle \equiv 0 \quad \Leftrightarrow \quad \mathcal{A} \equiv 0.$$

Notice that $\hat{\mathcal{A}}$ can be computed by an ABP with $d - 1$ levels in the following manner. Remove all the vertices in level 1 from $\mathcal{A}$. For every $i \in [m_2]$, connect v_i to the source with an edge labeled by ℓ_i. We now give a bound on the number of operations needed to construct $\hat{\mathcal{A}}$, given $\mathcal{A}$. With $O(m_1 \cdot m_2 \cdot n^2)$ operations we can compute the vectors $\{\alpha_k, \beta_{j,k}, \gamma_{j,k}\}_{j,k}$. We can compute the ℓ_i's with $O((n^2 + m_2) \cdot n^2 \cdot m_2)$ operations, using Gaussian elimination. Thus, using $O(m_2 \cdot n^4 + m_2^2 \cdot n^2 + m_1 \cdot m_2 \cdot n^2)$ operations we can transform $\mathcal{A}$ to $\hat{\mathcal{A}}$.

Theorem 4.17 ([106]). Let $\mathcal{A}$ be an ABP of size s, then we can deterministically verify whether $\mathcal{A} \equiv 0$ in time $O(s^5 + s \cdot n^4)$.

Proof. The size of $\mathcal{A}$ is $s = m_1 + \cdots + m_d$, where m_i is the size of level i. By the above procedure, we can reduce $\mathcal{A}$ to an ABP with two levels in d steps. After the i-th step, we have an ABP in at most $n + m_{i+1}$ variables (where $m_d = 1$). The reduction, therefore, runs in time

$$O\left(\sum_{i=1}^{d-1} m_{i+1} \cdot (m_i + n)^4 + (m_{i+1})^2 \cdot (m_i + n)^2 \right.$$
$$\left. + m_i \cdot m_{i+1} \cdot (m_i + n)^2 \right) = O(s^5 + s \cdot n^4).$$

As we can efficiently verify whether an ABP with two levels is identically zero or not, the result follows. □

This algorithm runs in polynomial time but it works only in the white-box setting. It is thus an interesting question to find a black-box algorithm for the problem. Any black-box algorithm will need to make assignments from a noncommutative domain, as otherwise it will make a mistake, e.g., on the polynomial $xy - yx$ which is nonzero when x and y do not commute. In the next section we present randomized noncommutative PIT algorithms.

4.5.1 Randomized Noncommutative PIT Algorithms

The first such algorithm was given by Bogdanov and Wee [23], that presented a noncommutative analog of the Schwartz–Zippel algorithm. A nontrivial extension is required as, e.g., the polynomial $xy - yx$ is nonzero in the noncommutative world but vanishes whenever the inputs come from a commutative domain.

Theorem 4.18 ([23]). There exists a black-box, randomized identity testing algorithm for the class of noncommutative degree d n-variate polynomials that uses $O(d \log n \log(d \log n/\epsilon))$ random bits and succeeds with probability $1 - \epsilon$.

The algorithm is based on a classical theorem of Amitsur and Levitzki saying that over any field $\mathbb{F}$ the matrix algebra $\mathbb{F}^{k\times k}$ does not satisfy any polynomial identity of degree less than $2k$ [7]. In other words, for every nonzero polynomial $g(x_1, \ldots, x_n)$ of degree smaller than $2k$, there exist $k \times k$ matrices $A_1, \ldots, A_n$ such that $g(A_1, \ldots, A_n)$ is nonzero. For example, for $xy - yx$, we have

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} - \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}.$$

The high-level idea behind the Bogdanov–Wee algorithm is the following. Let $f(x_1, \ldots, x_n)$ be a noncommutative polynomial of degree r. Evaluate f on random $r \times r$ matrices over $\mathbb{F}$ (we need to assume that $\mathbb{F}$ is large enough, otherwise we pick matrices over an extension field). After this substitution, f computes an $r \times r$ matrix in which each entry is a degree r polynomial in the entries of the n matrices. If f is not a polynomial identity for $\mathbb{F}^{r\times r}$, then some entry of f computes a nonzero polynomial of degree at most r in the entries of the matrices that we picked. In particular, if we pick the matrices at random then by the usual commutative version of the Schwartz–Zippel lemma f will compute a nonzero matrix with high probability. Furthermore, we may obtain a lower bound for this probability via the commutative Schwartz–Zippel lemma. While this is the main idea behind the algorithm, Bogdanov and Wee use another trick in order to optimize parameters. Roughly, they pick the random matrices from a linear subspace where some entries of the matrices are fixed to zero, and the remaining entries are chosen randomly from some subset $T \subseteq \mathbb{F}$.

In Ref. [13] a randomized algorithm based on the so-called *isolation lemma* and that uses ideas from automata theory was given. To explain their approach, we first recall some standard automata theory (see, e.g., Ref. [59]). Fix a deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ that takes inputs from Σ^*, where Q is the set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, and q_0 and q_f are the initial and final states, respectively. For each letter $w \in \Sigma$, let $\delta_w : Q \to Q$ be defined by $\delta_w(q) = \delta(q, w)$. Define the *transition matrix* M_w to be the $|Q| \times |Q|$ matrix given by $M_w(q, q') = 1$ if $\delta_w(q) = q'$ and $M_w(q, q') = 0$ otherwise. For a word $\bar{w} = w_1 w_2 \ldots w_k \in \Sigma^*$, we define $M_{\bar{w}} = M_{w_1} \cdots M_{w_k}$, and if $\bar{w}$ is the empty string then $M_{\bar{w}}$ is the identity matrix. Let $\delta_{\bar{w}}$ denote the natural extension of the transition function to $\bar{w}$ (if $\bar{w}$ is the empty string then $\delta_{\bar{w}}$ is simply the identity function). As before, we have $M_{\bar{w}}(q, q') = 1$ iff $\delta_{\bar{w}}(q) = q'$. Specifically, $M_{\bar{w}}(q_0, q_f) = 1$ iff $\bar{w}$ is accepted by the automaton A.

The authors of Ref. [13] use automata to get a randomized PIT algorithm in the following way. Let Φ be the noncommutative circuit that we wish to test. Let $A = (Q, \Sigma, \delta, q_0, q_f)$ be a finite automaton over the alphabet $\Sigma = \{x_1, \ldots, x_n\}$ and define the matrices $M_{x_i} \in \mathbb{F}^{|Q|\times|Q|}$ as before. Consider the matrix M_{out} obtained by evaluating Φ on the input $(M_{x_1}, \ldots, M_{x_n})$. The crucial observation is that M_{out} is determined completely by the polynomial f computed by Φ, and the structure of Φ itself is otherwise irrelevant. In particular, M_{out} is always zero when $f \equiv 0$. Consider what happens when Φ computes a single monomial, i.e., $f(x_1, \ldots, x_n) = c \cdot x_{j_1} \cdots x_{j_r}$, where c is a field element. In this case, the output matrix M_{out} is the matrix $c \cdot M_{x_{j_1}} \cdots M_{x_{j_r}} = c \cdot M_{\bar{w}}$, for $\bar{w} = x_{j_1} \cdots x_{j_r}$. Specifically, $M_{\text{out}}(q_0, q_f)$ is zero when A rejects $\bar{w}$, and equals c when A accepts $\bar{w}$. In general, suppose that Φ computes the polynomial $f = \sum_{i \in [t]} c_i \cdot m_i$, where $m_i = \prod_{k=1}^{r_i} x_{i_k}$. Let $\bar{w}_i$ denote the string representing the monomial m_i. Finally, let $S(f, A) = \{i \in [t] : A \text{ accepts } \bar{w}_i\}$. It is not hard to see that $M_{\text{out}}(q_0, q_f) = \sum_{i \in S(f,A)} c_i$. Thus, when evaluating Φ on the input $(M_{x_1}, \ldots, M_{x_n})$ we obtain a matrix that reflects how the automaton acts on f, when viewed as the set of strings corresponding to its monomials. In particular, if A is an automaton that accepts exactly one string that corresponds to a monomial of f, then $M_{\text{out}}(q_0, q_f)$ is the coefficient of that monomial in f. In order to use this observation we recall the isolation lemma of Mulmuley et al. [97].

Lemma 4.19. Let U be a universe of size u and $\mathcal{F}$ be any family of subsets of U. Let $W : U \to [2u]$ denote a weight assignment function to elements of U. Then, when W is picked uniformly at random, the probability that there exists a unique minimum-weight set in $\mathcal{F}$ is at least 1/2 (the weight of a set is the sum of the weights of its elements).

To obtain a "noncommutative Schwartz–Zippel algorithm" we do the following. Let f be a noncommutative polynomial of degree r. Consider the universe $U_t = [t] \times [n]$ with $t \in [r]$. We identify each monomial $m = x_{i_1} \cdots x_{i_t}$ with the set $S_m = \{(1, i_1), (2, i_2), \ldots, (t, i_t)\} \subseteq U_t$. Consider the family $\mathcal{F}$ of all sets S_m such that the monomial m has a nonzero coefficient in f. The isolation lemma tells us that if we assign random weights from $[2tn]$ to the elements of U_t then with probability at least 1/2 there is a unique minimum-weight set in $\mathcal{F}$. In Ref. [13] Arvind and Mukhopadhyay construct a family of small automata $\{A_{\mathrm{w},t} : \mathrm{w} \in [2nr], t \in [r]\}$ such that the automaton $A_{\mathrm{w},t}$ accepts precisely all the strings m of length t such that the weight of S_m is w. Thus, the automaton corresponding to the minimum weight accepts only one string (monomial). Now, for each automaton $A_{\mathrm{w},t}$, plug the n matrices that it defines to f as inputs in the way described above. If the output matrix of some automaton is nonzero, then the algorithm declares $f \not\equiv 0$, otherwise it declares $f \equiv 0$. By the isolation lemma, with probability at least $1/2$ the algorithm is correct. Repeating this procedure several times, with "fresh" weight function at each run, we can reduce the error as we like.

We end this section by stating two open questions.

Open Problem 22. Derandomize black-box PIT for noncommutative arithmetic formulas.

Another interesting problem is to derandomize PIT for noncommutative *circuits*, even in the white-box setting. Currently no sub-exponential time deterministic algorithm is known. We note that similar to the general PIT question, derandomizing PIT for noncommutative circuits is equivalent to proving lower bounds in the sense of [69]. Specifically, Arvind et al. [14] proved the following.

Theorem 4.20. **([14])** If PIT for noncommutative circuits can be done in deterministic sub-exponential time, then either $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$ or the noncommutative permanent does not have polynomial-size noncommutative circuits.

The proof is essentially the same as the proof of Theorem 4.7 and is left to the readers. Thus, derandomizing PIT of noncommutative circuits can be viewed as a first step toward proving super-polynomial lower bound for noncommutative circuits (recall Open Problem 10).

Open Problem 23. Derandomize PIT for noncommutative arithmetic circuits.

Finally, we note that while for general circuits Kabanets and Impagliazzo showed that lower bounds imply PIT algorithm (Theorem 4.9), no such result is known for noncommutative circuits.

Open Problem 24. Obtain an analog of Theorem 4.9 for noncommutative circuits. I.e., show that an exponential lower bound for noncommutative circuits computing a multilinear polynomial, implies a sub-exponential time deterministic black-box PIT algorithm for noncommutative circuits.

4.6 Depth-3 Circuits

Theorem 4.12 shows that black-box derandomization of $\Sigma\Pi\Sigma\Pi$ circuits is almost equivalent to black-box derandomization of PIT for general circuits. Currently only the case of depth-2 is solved, as we saw in Section 4.4. The next step is, therefore, depth-3 circuits. In the last few years this model attracted a lot of research but still the question of derandomizing PIT for depth-3 circuits remains open. We note that for depth-3 circuits the interesting case is $\Sigma\Pi\Sigma$ circuits and not $\Pi\Sigma\Pi$ circuits, as the latter follows by the depth-2 case.

Open Problem 25. Give a deterministic sub-exponential time PIT algorithm for $\Sigma\Pi\Sigma$ circuits.

As the general question is still open, research has focused on restricted depth-3 circuits, in hope of gaining a better understanding of the general case. Klivans and Spielman [86] raised the problem of giving a deterministic white-box PIT algorithm for $\Sigma\Pi\Sigma$ circuits, even when the top fan-in is three. Namely, when there are only three multiplication gates. This question was first solved in Ref. [43] who gave a white-box quasi-polynomial time deterministic PIT algorithm for $\Sigma\Pi\Sigma(k)$ circuits (depth-3 circuits with top fan-in k). This result was significantly improved by Kayal and Saxena [81] who gave an $n^{O(k)}$ time white-box PIT algorithm for $\Sigma\Pi\Sigma(k)$ circuits. Consequently, Arvind and Mukhopadhyay [12] gave a somewhat simpler algorithm of the same running time for the problem. In Ref. [76] it was shown how to generalize the approach of [43] to the black-box setting, thus giving the first quasi-polynomial time black-box algorithm for the problem. This result was later improved by Saxena and Seshadhri [117] who gave a better quasi-polynomial-time-algorithm over finite fields, and by Kayal and Saraf [80] who gave a polynomial-time algorithm, of running time $n^{k^{O(k)}}$, over $\mathbb{Q}$ and $\mathbb{R}$. Very recently, Saxena and Seshadhri [118] obtained improved algorithms for both finite fields, where the algorithm runs in quasi-polynomial time, and for $\mathbb{Q}$ and $\mathbb{R}$, where it runs in time $n^{O(k^2)}$. The algorithms of [80, 117, 118] are the same as the original algorithm of Karnin and Shpilka [76], the difference is that they improve the parameters in the main lemma of Dvir and Shpilka [43], on which the analysis of Karnin and Shpilka [76] relies.

4.6.1 White-Box Algorithms

The following theorem of Kayal and Saxena [81] gives the best white-box algorithm for $\Sigma\Pi\Sigma(k)$ circuits.

Theorem 4.21([81]). There is a deterministic poly(n, r^k)-time white-box PIT algorithm for $\Sigma\Pi\Sigma(k)$ circuits of degree r.

We shall present the ideas behind the algorithm of Kayal and Saxena without giving the full details. We start by a simplified version of the algorithm. Let Φ be a given $\Sigma\Pi\Sigma(k)$ circuit of degree r. Assume that we can find $(r+1)$ co-prime linear functions $\ell_1,\ldots,\ell_{r+1}$ that appear in Φ. In this case $\Phi \equiv 0 \mod \prod_{i\in[r+1]} \ell_i$ iff $\Phi \equiv 0$, as $\deg(\Phi) < \deg(\prod_{i\in[r+1]} \ell_i)$. In other words, $\Phi \equiv 0$ iff for every ℓ_i it holds that $\Phi|_{\ell_i=0} \equiv 0$. (imagine that $\ell_i = x_i$ and in this case $\Phi|_{\ell_i=0}$ is the circuit obtained by substituting $x_i = 0$. The general case is handled similarly.) Since each ℓ_i appears in Φ, the circuit $\Phi|_{\ell_i=0}$ is a $\Sigma\Pi\Sigma(k-1)$ circuit. This gives rise to an inductive argument: find "enough" co-prime linear functions that appear in the circuit and check all the restriction of the circuit to the linear functions. The base case for the induction is $k = 2$ which is trivial to check, as a $\Sigma\Pi\Sigma(2)$ circuit is zero iff its two multiplication gates have the same factorization (with a different sign). Thus, if we could find such a set of linear functions, we would be done.

In the general case, however, we may not be able to find such a set. For example, this will occur if all linear functions appear with high multiplicity. Kayal and Saxena circumvent this difficulty by working modulo powers of linear functions and performing computations in the ring $\mathbb{F}[x_1,\ldots,x_n]/\langle x_1^{e_1},\ldots,x_r^{e_r}\rangle$. To better understand their approach, let us first consider the case $k = 3$. It is not difficult to see (e.g., using a greedy argument) that if the multiplication gates are not identical, and no linear function appears in all three gates, then we can find co-prime *powers* of linear functions $\ell_1^{e_1},\ldots,\ell_m^{e_m}$ such that each $\ell_i^{e_i}$ divides exactly one of the multiplication gates and $e_1 + \cdots + e_m > r$. The algorithm now checks whether $\Phi \equiv 0 \mod \ell_i^{e_i}$ for every i. Let us consider what happens when we set $\ell_1 = 0$. For simplicity, assume without loss of generality that $\ell_1 = x_1$. The circuit $\Phi \mod x_1^{e_1}$ is a $\Sigma\Pi\Sigma(2)$ circuit in the ring $\mathbb{F}[x_1,\ldots,x_n]/\langle x_1^{e_1}\rangle$. We now repeat the same procedure for this circuit, that is, find a set of co-prime powers of linear functions (that are also co-prime to x_1), each dividing one of the two remaining gates, such that their product is of high degree. By doing so we reduced PIT for $\Sigma\Pi\Sigma(3)$ circuits to $O(r^2)$ instances of the problem of verifying that a given multiplication gate is zero in the ring $\mathbb{F}[x_1,\ldots,x_n]/\langle x_1^{e_1},x_2^{e_2}\rangle$. This can be done by observing that a product of linear functions is zero modulo $\langle x_1^{e_1},x_2^{e_2}\rangle$ iff the product of all the factors that only involve x_1,x_2 is zero in this ring, which can be verified in time $\text{poly}(n,e_1 \cdot e_2)$.

To extend the proof to higher values of k, one has to make sure that the Chinese Remainder Theorem also holds for rings of the form $\mathbb{F}[x_1,\ldots,x_n]/\langle x_1^{e_1},\ldots,x_r^{e_r}\rangle$, which is what Kayal and Saxena [81] did. This enables an inductive argument since the same argument as above can be used to reduce PIT of $\Sigma\Pi\Sigma(k)$ circuits to at most $r + 1$ instances of PIT of $\Sigma\Pi\Sigma(k-1)$ circuits over rings "similar" to $\mathbb{F}[x_1,\ldots,x_n]/\langle x_1^{e_1}\rangle$. This description also shows that the running time is $\text{poly}(n,r^k)$.

4.6.2 Black-Box Algorithms

We now explain the main ideas behind [43, 76, 80, 117, 118]. All these works are based on the following structural theorem from [43]. The improvements of [80, 117, 118] are based on improving the parameters in the theorem. Before stating the theorem we define the notions of simple and minimal circuits.

Definition 4.1. A $\Sigma\Pi\Sigma(k)$ circuit $\Phi = \sum_{i \in [k]} \Psi_i$, where each Ψ_i is a $\Pi\Sigma$ circuit, is *minimal* if there is no nonempty set $I \subsetneq [k]$ such that $\sum_{i \in I} \Psi_i \equiv 0$. The circuit is *simple* if no linear function is a factor of all the multiplication gates, that is, if $\gcd(\Psi_1, \ldots, \Psi_k) = 1$.

Clearly, one can create an identically zero $\Sigma\Pi\Sigma(k)$ circuit by adding together several smaller depth-3 circuits that are identically zero, or by multiplying a zero $\Sigma\Pi\Sigma(k)$ circuit by some linear function. The following theorem of Dvir and Shpilka [43] shows that if an identically zero circuit was not constructed in this way, i.e., the circuit is simple and minimal, then it depends on a few linear functions. To state the theorem we need the following definitions.

Definition 4.2. Let $\Phi = \sum_{i \in [k]} \Psi_i$ be a $\Sigma\Pi\Sigma(k)$ circuit. Denote $\Psi_i = \prod_{j \in [r_i]} \ell_{i,j}$, where each $\ell_{i,j}$ is a linear function. The *degree* of Φ is $\deg(\Phi) = \max_{i \in [k]} r_i$. The *rank* of Φ is $\mathrm{rank}(\Phi) = \dim(\mathrm{span}\{\ell_{i,j} : i \in [k], j \in [r_i]\})$.

The theorem below is the structural theorem of Dvir and Shpilka [43] with the improved bounds of Saxena and Seshadhri [118].

Theorem 4.22 ([43, 80, 117, 118]). For every field $\mathbb{F}$, there exists a function $R(k,r) = R_{\mathbb{F}}(k,r)$ such that if Φ is a simple and minimal $\Sigma\Pi\Sigma(k)$ circuit of degree r over $\mathbb{F}$ that computes the zero polynomial, then $\mathrm{rank}(\Phi) \leq R(k,r)$. If $\mathbb{F}$ is a finite field then $R(k,r) = O(k^2 \log r)$. If $\mathbb{F} = \mathbb{R}$ or $\mathbb{Q}$ then $R(k,r) = k^2$.

The interesting point about the theorem is that the upper bound on the rank does not depend on n. Roughly, when a $\Sigma\Pi\Sigma$ circuit is simple, minimal, and computes the zero polynomial, it cannot involve too many linearly independent linear functions. This property will play a major role in all the black-box PIT algorithms as well as in the reconstruction algorithms given in Section 5.4. We now sketch the proof of the theorem. The actual proof is quite long and is different for finite fields and for $\mathbb{R}, \mathbb{Q}$. We only explain the case $k = 3$ over finite fields and over the reals.

Proof. [Sketch for $k = 3$] Let $\Phi = \Psi_1 + \Psi_2 + \Psi_3$ be a circuit computing the zero polynomial with multiplication gates Ψ_1, Ψ_2, and Ψ_3. Assume without loss of generality that $\deg(\Psi_1) = \deg(\Psi_2) = \deg(\Psi_3) = r$ and that all the linear forms in Φ are homogeneous. Indeed, we can replace $\Psi_i = \prod_{j\in[r_i]}(a_{i,j,0} + \sum_{k\in[n]} a_{i,j,k}x_k)$ by $\Psi'_i = y^{r'_i}\prod_{j\in[r_i]}(a_{i,j,0}y + \sum_{k\in[n]} a_{i,j,k}x_k)$, where y is a new variables and r'_i is such that $\deg(\Psi'_i) = r$. Denote $R = \operatorname{rank}(\Phi)$. Then, there exist R linearly independent linear functions $\ell_1,\ldots,\ell_R$ in Φ. By applying an invertible linear transformation, we can assume without loss of generality that $\ell_i = x_i$ for every $i \in [R]$. Consider the circuit $\Phi|_{x_i=0}$ for some $i \in [R]$. Clearly, $\Phi|_{x_i=0} \equiv 0$. As $\ell_i = x_i$ appears in Φ, some product gate becomes zero in $\Phi|_{x_i=0}$. Observe that neither of the other two product gates will become zero after setting $x_i = 0$. Indeed, if x_i divides both Ψ_1 and Ψ_2 then, as $\Psi_3 = -\Psi_1 - \Psi_2$, we get that x_i divides Ψ_3 as well. But, since Φ is simple this is a contradiction. We thus have that $\Phi|_{x_i=0}$ is a zero $\Sigma\Pi\Sigma(2)$ circuit. This is possible only if the two product gates multiply the same linear functions, up to multiplication by constants. The variable x_i thus induces a partial matching of the linear functions in the circuit. This matching contains r pairs of linear functions such that for every pair (ℓ, ℓ') in the matching, the two linear functions ℓ and ℓ' belong to two different multiplication gates and $\ell|_{x_i=0} = \alpha \cdot \ell'|_{x_i=0}$ for some nonzero constant $\alpha \in \mathbb{F}$. Denote with M_i the matching induced by x_i. The next claim gives more information about these pairs.

Claim 4.23. For every (ℓ, ℓ') in M_i it holds that $x_i \in \operatorname{span}\{\ell, \ell'\}$.

Proof. Denote $\ell = \sum_{j\in[n]} a_j x_j$ and $\ell' = \sum_{j\in[n]} b_j x_j$. Since $\ell|_{x_i=0} = \alpha \cdot \ell'|_{x_i=0}$, it holds that $a_j = \alpha \cdot b_j$ for every $j \neq i$ in $[n]$. Since the two linear forms ℓ, ℓ' are linearly independent, as Φ is simple, it must hold that $a_i \neq \alpha \cdot b_i$. Hence, $x_i = (\ell - \alpha \cdot \ell')/(a_i - \alpha \cdot b_i)$. □

Claim 4.23 tells us that every pair (ℓ, ℓ') in M_i spans x_i. We also have that all the matchings $\{M_i\}_{i\in[R]}$ are contained in a set of at most $3r$ linear functions, and that each matching contains r pairs. Standard arguments from information theory (e.g., Theorem 1.2 of Dvir and

Shpilka [43] concerning *locally decodable codes*) now show that, regardless of the field, $3r \geq \exp(R)$. In other words, $R = O(\log r)$. This proved the claimed result for finite fields.

The argument above works for any field, but when $\mathbb{F} = \mathbb{R}, \mathbb{Q}$ one can get a better bound. To obtain the improved bound we use a slightly different point of view, as conjectured in Ref. [43] and first proved in Ref. [80]. Given a linear function $\ell(x) = \sum_{i \in [n]} a_i \cdot x_i$ we map it to the line $\varphi(\ell)$ in $\mathbb{R}^n$ passing through $a = (a_1, \ldots, a_n)$ and the origin. Given a simple $\Sigma\Pi\Sigma(3)$ circuit computing the zero polynomial, consider the three multi-sets of lines corresponding to each of the three multiplication gates. Let $H \subset \mathbb{R}^n$ be a hyperplane such that each of the lines defined above intersects H in exactly one point (a random H has this property). For a linear function ℓ from the circuit, denote this intersection point by $h(\ell) = \varphi(\ell) \cap H$. This gives a multi-set of points in H. Color all these points by three colors, according to the different multiplication gate that they came from (every point receives exactly one color, as Φ is simple). As the circuit is zero, similar to the proof of Claim 4.23, for every ℓ and ℓ' that belong to two different multiplication gates, there is some ℓ'' in the third gate such that $\ell'' \in \operatorname{span}(\ell, \ell')$. This implies that the point $h(\ell'')$ belongs to the line passing through $h(\ell)$ and $h(\ell')$. The three colored sets thus have the property that every line that contains points from two sets must also contain a point from the third set. By [45] this implies that the points belong to a three-dimensional subspace of $\mathbb{R}^n$. It follows that, originally, the linear function in the circuit span a vector space of dimension at most 4. □

We note that the result of Edelstein and Kelly [45] is a colored version of the famous Sylvester–Gallai theorem, that shows that if we have a collection of points in the plane such that any line passing through two points contains a third point then the points must be collinear.

The proof above only holds for the case $k = 3$. The extension to larger values of k is by induction [43, 80, 117, 118]. We do not give further details of the proofs, which are based on the arguments sketched above.

Theorem 4.22 enables us to explain the black-box algorithm of Karnin and Shpilka [76].

Theorem 4.24 ([76]). There is a deterministic $n^{O(R(k,r))}$-time black-box PIT algorithm for $\Sigma\Pi\Sigma(k)$ circuits of degree r.

Before giving the proof we need the following definition. Let $\Phi = \sum_{i\in[k]} \Psi_i$ be a $\Sigma\Pi\Sigma(k)$ circuit. Let $g = \gcd(\Psi_1,\ldots,\Psi_k)$ be the greatest common divisor of all the multiplication gates. Clearly, g is a product of linear functions. The *simplification* of Φ is

$$\mathrm{sim}(\Phi) = \Phi/g.$$

In words, it is the circuit obtained by deleting g from each Ψ_i. Specifically, $\mathrm{sim}(\Phi)$ is a simple $\Sigma\Pi\Sigma(k)$ circuit.

Proof. [Sketch] Denote $R = R(k,r)$. The first step is to construct a set of $m = \mathrm{poly}(n)$ linear transformations $\mathcal{T} = \{T_i : \mathbb{F}^{2R} \to \mathbb{F}^n\}_{i\in[m]}$ such that for every vector space V, of linear functions from $\mathbb{F}^n$ to $\mathbb{F}$, it holds that all but a polynomially small fraction of the transformations T in $\mathcal{T}$ satisfy that

$$\dim(V \circ T) = \min(\dim(V), 2R),$$

where $V \circ T = \mathrm{span}\{\ell \circ T : \ell \in V\}$ and $\circ$ is composition. Intuitively, for any such V, for most of the transformations T in $\mathcal{T}$, the linear functions in V remain as linearly independent as possible when composed with T. Indeed, $\dim(V \circ T)$ is never larger than $\min(\dim(V), 2R)$. Gabizon and Raz [47] constructed a set $\mathcal{T}$ with the required properties (see Ref. [76] for the exact translation of the construction of Gabizon and Raz [47] to our setting). We omit the details of this construction. For a $\Sigma\Pi\Sigma(k)$ circuit Φ and $T \in \mathcal{T}$, denote by $\Phi \circ T$ the $\Sigma\Pi\Sigma(k)$ circuit in $2R$ variables $y = (y_1,\ldots,y_{2R})$ obtained by substituting x_ℓ by $(Ty)_\ell$ in Φ.

Assume the existence of such a set $\mathcal{T}$. Let $\Phi = \sum_{i\in[k]} \Psi_i$ be a $\Sigma\Pi\Sigma(k)$ circuit where $\Psi_i = \prod_{j\in[r_i]} \ell_{i,j}$. For every nonempty $I \subseteq [k]$, let $\Phi_I = \sum_{i\in I} \Psi_I$. Define V_I as the span of all linear functions that occur in $\mathrm{sim}(\Phi_I)$. In addition, for every pair of linear functions ℓ, ℓ' in $\mathrm{sim}(\Phi)$, consider the space $W_{\ell,\ell'} = \mathrm{span}\{\ell,\ell'\}$. By the union bound, $\mathcal{T}$ contains a transformation T such that for every V_I and every such $W_{\ell,\ell'}$, it holds that $\dim(V_I \circ T) = \min(\dim(V_I), 2R)$ and

$\dim(W_{\ell,\ell'} \circ T) = \min(\dim(W_{\ell,\ell'}), 2R)$. We now show that $\Phi \equiv 0$ iff $\Phi \circ T \equiv 0$. As a first step we prove a claim showing that if the rank of $\mathrm{sim}(\Phi_I \circ T)$ is small then so is the rank of $\mathrm{sim}(\Phi_I)$.

Claim 4.25. For every nonempty I, we have $\mathrm{rank}(\mathrm{sim}(\Phi_I \circ T)) = \min(\mathrm{rank}(\mathrm{sim}(\Phi_I)), 2R)$.

Proof. We first show that $\gcd(\Phi_I \circ T) = (\gcd(\Phi_I)) \circ T$. Namely, that no new linear functions were added to the greatest common divisor. Assume toward a contradiction that $\gcd(\Phi_I \circ T) \neq (\gcd(\Phi_I)) \circ T$. It must be the case that for some ℓ that occurs in Φ_I but does not occur in $\gcd(\Phi_I)$, the linear form $\ell \circ T$ belongs to all the product gates in $\Phi_I \circ T$. Therefore, there is some ℓ' in Φ_I such that ℓ and ℓ' are linearly independent but $\ell \circ T$ and $\ell' \circ T$ are linearly dependent. This contradicts the choice of T, as $\dim(W_{\ell,\ell'} \circ T) < 2$. A similar argument shows that for every ℓ in $\gcd(\Phi_I)$ the linear form $\ell \circ T$ is nonzero. We thus get that $\mathrm{sim}(\Phi_I \circ T) = \mathrm{sim}(\Phi_I) \circ T$. The result now follows by the choice of T, when considering $V_I \circ T$. □

The following lemma is the final step before describing the algorithm.

Lemma 4.26. $\Phi \equiv 0$ iff $\Phi \circ T \equiv 0$.

Proof. Assume that $\Phi \circ T \equiv 0$. Our goal is to show that in this case $\Phi \equiv 0$ (the other direction is trivial). Let $[k] = I_1 \cup I_2 \cup \ldots \cup I_c$ be the unique partition satisfying that each of the circuits $\Phi_{I_j} \circ T$ is minimal and zero. Consider $\mathrm{sim}(\Phi_{I_j} \circ T)$. It is a simple, minimal and zero depth-3 circuit, and so by Theorem 4.22 it holds that $\mathrm{rank}(\mathrm{sim}(\Phi_{I_j} \circ T)) \leq R$. Claim 4.25 implies that $\mathrm{rank}(\mathrm{sim}(\Phi_{I_j})) = \mathrm{rank}(\mathrm{sim}(\Phi_{I_j}) \circ T)$. In other words, the rank of $\mathrm{sim}(\Phi_{I_j})$ does not change after composition with T. Therefore, we also have that $\mathrm{sim}(\Phi_{I_j}) \equiv 0$, because T is "an invertible transformation" between $\mathrm{sim}(\Phi_{I_j})$ and $\mathrm{sim}(\Phi_{I_j} \circ T)$. Specifically, $\Phi = \sum_{j=1}^{c} \Phi_{I_j} \equiv 0$. □

To obtain a PIT algorithm, using such a "good" T, all that we have to do is to check whether $\Phi \circ T$ is identically zero. As $\Phi \circ T$ is a degree r polynomial in $2R$ variables, by evaluating it on all points in S^{2R}, where $S \subseteq \mathbb{F}$ is a set of size $r+1$, we can test whether it is zero or not. The PIT algorithm, therefore, simply checks whether for all $T \in \mathcal{T}$ it holds that $(\Phi \circ T)(S^{2R}) = \{0\}$, and decides accordingly. $\square$

When the underlying $\Sigma\Pi\Sigma(k)$ circuit is multilinear, namely, every product gate in the circuit computes a multilinear polynomial, a stronger result can be proved. Indeed, notice that the rank of a simple multilinear $\Sigma\Pi\Sigma(k)$ circuit of degree r is at least r. This follows since all the linear functions in any one of its multiplication gates have disjoint support and are therefore linearly independent. Combining this with the bound $R(k,r) \le O(k^3 \log r)$, we get that $r \le O(k^3 \log r)$ and hence $r \le O(k^3 \log k)$ and $R(k,r) = O(k^3 \log k)$. There is no clear way, however, to use this improved rank bound for multilinear circuits, as in the proof of Theorem 4.24, since after composing a multilinear circuit with a linear transformation we do not necessarily get a multilinear circuit. In Ref. [76] a set of transformations $\mathcal{T}$ that has the required properties and that also keeps the circuit multilinear, was constructed. Using this set $\mathcal{T}$, Karnin and Shpilka [76] obtained an $n^{\exp(k^2)}$-time black-box PIT algorithm for multilinear $\Sigma\Pi\Sigma(k)$ circuits. This result was later improved in Ref. [127] that gave an $n^{O(k)}$-time algorithm. Interestingly, the result of Ref. [127] does not rely on the rank bound, but rather uses their PIT algorithm for sums of read-once formulas. We discuss this result in Section 4.8 (Theorem 4.43). In spite of all the work the following question is still open.

Open Problem 26. Give a deterministic sub-exponential time PIT algorithm for multilinear $\Sigma\Pi\Sigma$ circuits.

In Section 4.5 we saw a white-box PIT algorithm for noncommutative formulas. A special type of formulas that can be thought of as a model of noncommutative formulas is the model of *set-multilinear* depth-3 circuits [100]. Recall that a set-multilinear depth-3 circuit has the following structure. The variables X are given as the union of r disjoint sets $X = X_1 \cup X_2 \cup \cdots \cup X_r$. The circuit is a depth-3 circuit in the variables X with the additional property that every product gate has the form $\Psi = \prod_{i=1}^{r} \ell_i(X_i)$, each ℓ_i is a linear form in X_i. In words, each product gate multiplies a linear function in X_1 with a linear function in X_2 with a linear function in X_3 and so forth. Clearly, every monomial that appears in the polynomial computed by such a circuit contains exactly one variable from each of the sets $X_1, \ldots, X_r$. It is not difficult to see that such circuits compute the same polynomial also in a noncommutative world. Namely, they do not "use" commutativity during the computation (as we always multiply a linear function in X_1 with a linear function in X_2, etc.). Stated differently, depth-3 set-multilinear circuits form a restricted sub-model both of multilinear depth-3 circuits and of noncommutative formulas. As a first step toward understanding Open problem 26 one can try to solve the following problem, for which we already saw a white-box algorithm.

Open Problem 27. Give a deterministic sub-exponential time black-box PIT algorithm for set-multilinear $\Sigma\Pi\Sigma$ circuits.

Another sub-model of depth-3 circuits that was considered is the so-called model of *diagonal* circuits, defined by Saxena [116]. A diagonal depth-3 circuit is a depth-3 circuit in which the number of *different* linear functions in each multiplication gate is bounded by some number k, and the degree is polynomial. As a comparison, in $\Sigma\Pi\Sigma(k)$ circuits we bound the top fan-in by k and here we do not restrict the top fan-in but rather restrict the number of "different" children of each product gate. Saxena showed how to adapt the noncommutative PIT algorithm of Raz and Shpilka [106] to this scenario as well, thus obtaining a polynomial time white-box algorithm when $k = O(1)$. We explain his approach in Section 4.7.1 where we discuss a generalization of this model, namely, *diagonal depth-4* circuits.

4.7 Depth-4 Circuits

Theorem 4.12 tells us that derandomization of PIT of depth-4 circuits implies a quasi-polynomial time derandomization of PIT of general arithmetic circuits. It is, therefore, not surprising that very little is known even for restricted sub-models of depth-4 circuits. We briefly describe the sub-models for which such algorithms are known and then present the algorithms. In what follows we say that a polynomial is s-sparse if it has at most s monomials. Recall that a $\Sigma\Pi\Sigma\Pi$ circuit of size s has the form

$$\begin{aligned}\Phi &= \sum_{i=1}^{k} \Psi_i, \quad \text{where} \\ \Psi_i &= \prod_{j=1}^{r_i} P_{i,j}(x_1,\ldots,x_n), \quad \text{and each } P_{i,j} \text{ is an } s\text{-sparse polynomial.}\end{aligned} \tag{4.3}$$

We consider two different restrictions of this model. The first, defined by Saxena, is that of *diagonal* depth-4 circuits [116]. In a nutshell, these are circuits in which the number of different $P_{i,j}$'s in Ψ_i is small (constant, mostly), and each $P_{i,j}$ is a sum of univariate polynomials, $P_{i,j}(x) = \sum_{m=1}^{n} g_{i,j,m}(x_m)$. Saxena showed how to reduce the PIT problem of diagonal circuits to that of (a generalized version of) noncommutative formulas, thus obtaining a polynomial time white-box PIT algorithm for a certain range of parameters. We discuss this result in Section 4.7.1. Another model that was recently considered is multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits. In this model each of the k product gates Ψ_i computes a multilinear polynomial. In other words, the $P_{i,j}$'s are multilinear polynomials, and all the factors of Ψ_i are variable disjoint. For this model, Karnin et al. [75] gave a quasi-polynomial time black-box PIT algorithm (for $k = \mathrm{polylog}(n)$). We describe this result in Section 4.7.2. In Ref. [12] a white-box PIT algorithm for another restricted version of $\Sigma\Pi\Sigma\Pi$ circuits was given. The circuits have to be restricted in the following way. The top fan-in is constant, i.e., $k = O(1)$, each $P_{i,j}$ depends only on $c = O(1)$ variables, and each x_m appears in at most ck different $P_{i,j}$'s. Under these assumptions (in fact, some more assumptions are needed), Arvind and Mukhopadhyay [12] gave a polynomial-time algorithm. However, we shall later see that the black-box algorithm of Karnin et al. [75] can be adapted to work in this case as well. Very recently, Saraf and Volkovich [115] were able to obtain a polynomial time black-box algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits, for a constant k, thus improving the result of [75]. We shortly discuss this result in Section 4.7.2.

4.7.1 Diagonal Circuits

A diagonal depth-4 circuit has the following form: $\Phi = \sum_{i=1}^{k} \Psi_i$, each product gate Ψ_i is of the form $\Psi_i = \prod_{j=1}^{r_i} P_{i,j}^{e_{i,j}}$, and each $P_{i,j}$ is a sum of univariate polynomials, $P_{i,j} = \sum_{m=1}^{n} g_{i,j,m}(x_m)$. Saxena [116] gave a white-box algorithm for such circuits using the approach described in Section 4.5.

Theorem 4.27 ([116]). There is a deterministic algorithm that, given as input a diagonal depth-4 circuit Φ as above, runs in time $\mathrm{poly}(nk, \max_{i\in[k]}(1+e_{i,1})\cdot(1+e_{i,2})\cdots(1+e_{i,r_i}))$ and decides correctly whether $\Phi \equiv 0$.

Proof. [Sketch] While the theorem holds over any field, we shall sketch the argument only for fields of characteristic zero. The idea behind Saxena's algorithm is to consider a "dual form" for each product gate. This dual form will enable us to view the computation as done in a noncommutative world. For a power series $f(x_1,\ldots,x_n,z,z_1,\ldots,z_m)$, denote by $[z^e z_1^{e_1}\cdots z_m^{e_m}]f$ the coefficient of $z^e z_1^{e_1}\cdots z_m^{e_m}$ in f, which is a polynomial in $x_1,\ldots,x_n$. Consider a product gate $\Psi = \prod_{j=1}^{r} P_j^{e_j}$ with $P_j = \sum_{m=1}^{n} g_{j,m}(x_m)$. Let $e = e_1 + \cdots + e_r$. By considering the Taylor expansion of the exponential function, $\exp(\xi) = \sum_{\ell=0}^{\infty} \xi^\ell/\ell!$, and its truncated Taylor expansion, $E_e(\xi) = 1 + \xi + \xi^2/2! + \cdots + \xi^e/e!$, we observe that

$$\begin{aligned}(e_1!e_2!\cdots e_r!)^{-1}\Psi &= [z^e z_1^{e_1}\cdots z_r^{e_r}]\exp(P_1 z_1 z)\cdots\exp(P_r z_r z)\\ &= [z^e z_1^{e_1}\cdots z_r^{e_r}]\prod_{m=1}^{n}\exp\left(\sum_{j=1}^{r} g_{j,m}(x_m) z_j z\right)\\ &= [z^e z_1^{e_1}\cdots z_r^{e_r}]\prod_{m=1}^{n} E_e\left(\sum_{j=1}^{r} g_{j,m}(x_m) z_j z\right).\end{aligned} \tag{4.4}$$

Equation (4.4) is of polynomials of degree ne in z. We can therefore interpolate and obtain the coefficient of z^e. In other words, there exist

$\alpha_0,\ldots,\alpha_{ne}$ and $\beta_0,\ldots,\beta_{ne}$ in $\mathbb{F}$ such that

$$[z^e z_1^{e_1}\cdots z_r^{e_r}]\prod_{m=1}^{n} E_e\left(\sum_{j=1}^{r} g_{j,m}(x_m)z_j z\right)$$
$$=\sum_{\ell=0}^{ne}\beta_\ell\cdot[z_1^{e_1}\cdots z_r^{e_r}]\prod_{m=1}^{n} E_e\left(\sum_{j=1}^{r} g_{j,m}(x_m)z_j\alpha_\ell\right).$$

Let $\mathcal{I}=\langle z_1^{e_1+1},\ldots,z_r^{e_r+1}\rangle$ be the ideal generated by $z_1^{e_1+1},\ldots,z_r^{e_r+1}$ in the ring of polynomials. Set, $\mathcal{R}=\mathbb{F}[z_1,\ldots,z_r]/\mathcal{I}$. It follows that for some field elements $\{\beta'_\ell\}$ we have that

$$\Psi\cdot z_1^{e_1}\cdots z_r^{e_r}\equiv\sum_{\ell=0}^{ne}\beta'_\ell\prod_{m=1}^{n} E_e\left(\sum_{j=1}^{r} g_{j,m}(x_m)z_j\alpha_\ell\right)\quad\text{over}\,\mathcal{R}.\tag{4.5}$$

Indeed, the coefficient of any monomial other than $z_1^{e_1}\cdots z_r^{e_r}$ in the RHS of Equation (4.5) must involve some $z_i^{e_i+1}$ and is thus zero in $\mathcal{R}$. Equation (4.5) is called the *dual form* of a product gate in diagonal circuits. Observe that it expresses the multiplication gate as a sum of ne multiplication gates, where each is a product of univariate polynomials. This structure explains why the algorithm for noncommutative formulas can be applied to this setting.

Now, given our circuit $\Phi=\sum_{i=1}^{k}\Psi_i$ with $\Psi_i=\prod_{j=1}^{r_i}P_{i,j}^{e_{i,j}}$, denote $\mathcal{I}_i=\langle z_{i,1}^{e_{i,1}+1},\ldots,z_{i,r_i}^{e_{i,r_i}+1}\rangle$ and $\mathcal{R}_i=\mathbb{F}[z_{i,1},\ldots,z_{i,r_i}]/\mathcal{I}_i$. The dual form of the product gates Ψ_i is

$$\Psi_i\cdot z_{i,1}^{e_{i,1}}\cdots z_{i,r_i}^{e_{i,r_i}}\equiv\sum_{\ell=0}^{ne_i}\beta_{i,\ell}\prod_{m=1}^{n} E_{e_i}\left(\sum_{j=1}^{r_i} g_{i,j,m}(x_m)z_{i,j}\alpha_\ell\right)\quad\text{over}\,\mathcal{R}_i.$$

We would like to work over a single ring $\mathcal{R}$, and so we define the ideal $\mathcal{I}$ to be generated by the following relations:

(1) For every $i\in[k]$ and $j\in[r_i]$, the relation $z_{i,j}^{e_{i,j}+1}=0$.
(2) For every $i\neq i'$ in $[k]$, $j\in[r_i]$ and $j'\in[r_{i'}]$, the relation $z_{i,j}\cdot z_{i',j'}=0$.
(3) For every $i,i'\in[k]$, the relation $z_{i,1}^{e_{i,1}}\cdots z_{i,r_i}^{e_{i,r_i}}=z_{i',1}^{e_{i',1}}\cdots z_{i',r_{i'}}^{e_{i',r_{i'}}}$.

Let $\mathcal{R} = \mathbb{F}[z_{1,1},\ldots,z_{k,r_k}]/\mathcal{I}$. A standard calculation reveals that $\mathcal{R}$ is an algebra of dimension $\sum_{i=1}^{k}(1+e_{i,1})\cdots(1+e_{i,r_i}) - 2(k-1)$. By the discussion above, we get that over $\mathcal{R}$

$$\begin{aligned}\Phi \cdot z_{1,1}^{e_{1,1}}\cdots z_{1,r_1}^{e_{1,r_1}} &= \sum_{i=1}^{k}\Psi_i \cdot z_{i,1}^{e_{i,1}}\cdots z_{i,r_i}^{e_{i,r_i}} \\ &= \sum_{i=1}^{k}\sum_{\ell=0}^{ne_i}\beta_{i,\ell}\prod_{m=1}^{n}E_{e_i}\left(\sum_{j=1}^{r_i} g_{i,j,m}(x_m)z_{i,j}\alpha_\ell\right) \quad \text{over}\,\mathcal{R}.\end{aligned} \tag{4.6}$$

As Equation (4.6) can be represented as $\sum_{i=1}^{k}\sum_{\ell=0}^{ne_i} f_{i,\ell,1}(x_1)\cdots f_{i,\ell,n}(x_n)$, we can view it as a noncommutative polynomial in $x_1,\ldots,x_n$. In Section 4.5 we described the PIT algorithm of Raz and Shpilka [106] for noncommutative formulas, that basically runs in time polynomial in the size of the formula, which in our case can be thought of as $nk \cdot \max_i e_i$. It turns out that this algorithm can be extended to work over the algebra $\mathcal{R}$ at the cost multiplying the running time by a polynomial in $\dim(\mathcal{R})$. This gives a PIT algorithm for diagonal depth-4 circuits with running time $\mathrm{poly}(nk, \max_i(1+e_{i,1})\cdots(1+e_{i,r_i}))$. In particular, if $r_i = O(1)$ then the algorithm runs in time polynomial in the size of the circuit. □

Theorem 4.27 tells us that there is a model of ΣΠΣΠ circuits with unbounded top fan-in for which a polynomial time white-box PIT algorithm is known. This comes at the cost of bounding the number of different factors that each multiplication gate has and the requirement that each $P_{i,j}$ is the sum of univariate polynomials. Besides diagonal depth-4 circuits, only noncommutative depth-4 formulas give a submodel of depth-4 circuits with unbounded top fan-in, for which efficient (white-box) PIT algorithms are known.

4.7.2 Multilinear ΣΠΣΠ(k) Circuits

We now describe the idea and proof of the PIT algorithm of Karnin et al. [75] for multilinear ΣΠΣΠ(k) circuits.

Theorem 4.28 ([75]). There is an algorithm that in time $n^{O(k^6 \log(k) \log^2 s)}$ constructs a hitting set for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits of size s in n variables.

To explain the proof we first recall that a multilinear $\Sigma\Pi\Sigma\Pi$ circuit Φ has the following structure. Each product gate is of the form $\Psi_i = \prod_{j=1}^{r_i} P_{i,j}$ and for every $j \neq j'$ in $[r_i]$, the polynomials $P_{i,j}, P_{i,j'}$ are defined on disjoint sets of variables. The PIT algorithm of Karnin et al. [75] builds on a reduction from identity testing of multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits to identity testing of a special type of such circuits, circuits for which for every i,j it holds that $|\text{var}(P_{i,j})| \leq |\text{var}(\Phi)|/c$, where $\text{var}(P)$ is the set of variables that appear in P and c is some parameter. We call such circuits *c-compressed* circuits. The main ingredient in the proof is the following lemma on the structure of multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits. For a set of (indices of) variables $I \subseteq [n]$, let $m(I) = \prod_{i \in I} x_i$ be the product of the variables in I.

Theorem 4.29. Let P be a nonzero n-variate polynomial computable by a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s. Let $c > 0$ be a parameter. Then there exists $I \subseteq [n]$ of size $|I| \leq 2 \log n \cdot \log s \cdot kc$ for which the following holds: Let $P = \sum_{T \subseteq I} m(T) P_T$ be the expansion of P with respect to the variables in I. In particular, each P_T is a polynomial in the variables (whose indices are) in $[n] \setminus I$. There exists a set $T \subseteq I$ such that: (1) P_T is a nonzero polynomial, and (2) $P_T = Q \cdot H$, where Q is a product of s-sparse polynomials and H is computable by a simple,[7] c-compressed $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s.

We first give the proof of the theorem and then explain how it can be used to derive a PIT algorithm.

Proof. Given a product gate Ψ_i, write $\Psi_i = A_i \cdot B_i$ where A_i is the product of all the $P_{i,j}$'s such that $|\text{var}(P_{i,j})| > \text{var}(P)/2c$ and B_i is the

[7] The definition of simple here is the same: a $\Sigma\Pi\Sigma\Pi(k)$ circuit $\Phi = \sum_{i=1}^{k} \Psi_i$ is *simple* if $\gcd(\Psi_1, \Psi_2, \ldots, \Psi_k) = 1$.

product of the other factors. As Ψ_i is multilinear, the number of factors in A_i is at most $2c$. Therefore, since $P_{i,j}$ is s-sparse, we get that A_i is s^{2c}-sparse. We would like to somehow "remove" the A_i part, as the next lemma hints at.

Lemma 4.30. Let $f \in \mathbb{F}[x_1,\ldots,x_n]$ be a multilinear s-sparse polynomial. Then, there exists a set of variables $I \subseteq \mathrm{var}(f)$ of size $|I| \leq \log s$ such that when writing $f = \sum_{T \subseteq I} m(T) f_T$, it holds that for some $T \subseteq I$, the polynomial f_T is a monomial.

Proof. We prove the lemma by induction on the size of $\mathrm{var}(f)$. Consider a variable $x \in \mathrm{var}(f)$ that does not divide f (if all variables divide f then f is a monomial and we are done). Write $f = xg + h$. There is a polynomial $f' \in \{g,h\}$ that is $s/2$-sparse. Induction, therefore, implies that a set of size at most $\log(s/2) = \log s - 1$ with the required property, with respect to f', exists. Add x to this set and observe that it satisfies the claim. □

The lemma tells us that there is some set of variables I and $T \subset I$ such that after taking a partial derivative with respect to the variables in T and setting the variables in $I \setminus T$ to zero, f becomes a (nonzero) monomial. With this in mind, let us continue the proof. Pick a set I_i for each A_i, according to Lemma 4.30, and let $I' = \bigcup_{i=1}^{k} I_i$. Specifically, $|I'| \leq k \cdot \log s^{2c} = 2ck \log s$. In addition, there is some $T' \subseteq I'$ such that after taking a partial derivative with respect to the variables in T' and setting the variables in $I' \setminus T'$ to zero, all the Ψ_i's become products of s-sparse polynomials, each of which depends on at most $|\mathrm{var}(P)|/2c$ variables, and the resulting polynomial is still nonzero (this can be seen, e.g., by considering the expansion of P w.r.t. I'). It seems that we are done, but this is not quite the case. Denote by $\tilde{P}_1$ the polynomial $\partial_{T'}(P)|_{x_{I' \setminus T'}=0}$, where $\partial_{T'}(P)$ is the partial derivative of P with respect to all variables in T' and $|_{x_{I' \setminus T'}=0}$ means substituting $x_i = 0$ for all $i \in I' \setminus T'$. The circuit computing $\tilde{P}_1$ is not necessarily simple and so we let Q_1 be the greatest common divisor of all the product gates of $\partial_{T'}(P)|_{x_{I' \setminus T'}=0}$ and set $P_1 = \tilde{P}_1/Q_1$. Clearly, the circuit for P_1 is simple. Now, it may be the case that $|\mathrm{var}(P_1)| < |\mathrm{var}(P)|/2$ and so P_1

is not c-compressed. If this is indeed the case then we simply repeat the process for P_1. This can happen at most $\log n$ times as the number of variables shrinks by a factor of at least two after each "unsuccessful" round. Now, we are basically done. We obtained a set I of size at most $2\log n \cdot \log s \cdot kc$ (which is the union of all sets I' constructed in the different stages) and a set $T \subset I$ so that the following holds. Let Q be the greatest common divisor of all the product gates of $\partial_T(P)|_{x_{I\setminus T}=0}$. Denote $H = \partial_T(P)|_{x_{I\setminus T}=0}/Q$. Then H is both c-compressed and simple as claimed. □

Theorem 4.29 suggests the following algorithmic approach. Start by constructing a hitting set for c-compressed circuits. Then, "guess" the set I guaranteed by the lemma (going over all possibilities requires quasi-polynomial time). Assume that we have the "correct" I. Theorem 4.29 guarantees that for some $T \subseteq I$, the coefficient[8] of $m(T)$ is a (nonzero) simple, c-compressed circuit. Therefore, by trying each of the possible substitutions to the variables in $[n] \setminus I$, from the hitting set for c-compressed circuits, we are guaranteed that for some substitution the resulting polynomial (in the I variables) will be nonzero. Now, all that we have to do is to verify in a black-box manner whether a multilinear polynomial in the $|I|$ variables is nonzero and this can be done in time $\exp(|I|)$, which is quasi-polynomial in n.

To conclude, our goal is to come up with a hitting set for c-compressed circuits. As we shall soon see the hitting set for c-compressed $\Sigma\Pi\Sigma\Pi(k)$ circuits is based on a generator for multilinear $\Sigma\Pi\Sigma\Pi(k-1)$ circuits. Thus, our construction is recursive in nature.

The idea behind the PIT algorithm for c-compressed circuits is that such circuits "contain" a nonzero multilinear depth-3 circuit. In fact, we prove that there is a small, but not too small, set J of variables and an assignment, coming from a hitting set for sparse polynomials, to the variables not in J so that after we make this assignment, we are left with a nonzero multilinear depth-3 circuit. To identify this "good" set of variables we use the following simple lemma (the proof is by a straightforward greedy argument).

[8] The coefficient is in fact a product of sparse polynomials and a c-compressed circuit, but this does not change the picture much.

Lemma 4.31. Let $P \in \mathbb{F}[x_1,\dots,x_n]$ be computed by a $\Sigma\Pi\Sigma\Pi(k)$ multilinear c-compressed circuit $\Phi = \sum_{i=1}^{k}\prod_{j=1}^{r_i} P_{i,j}$. Then there exists a set $J \subseteq \mathrm{var}(P)$ of size $|J| \geq c/k$ such that for every i,j we have $|J \cap \mathrm{var}(P_{i,j})| \leq 1$.

Let J be the set guaranteed by the lemma. Think of c as small so we can actually "guess" J (we can always make J smaller if it is too large). We shall try to fix the variables in $[n] \setminus J$ so that the resulting polynomial is computed by a nonzero multilinear $\Sigma\Pi\Sigma(k)$ circuit. For this end we shall use generators (see Section 4.1 for the definition) for sparse polynomials. We construct a generator for c-compressed circuits by induction on the top fan-in, k. As mentioned above, the inductive argument will in fact assume that we already have a generator for multilinear $\Sigma\Pi\Sigma\Pi(k-1)$ circuits.

The base case of the induction is $k = 1$. It is easy to see that a generator for s-sparse polynomials is also a generator for this case. Indeed, such a circuit is a product of s-sparse polynomials, and Observation 4.1 guarantees that a generator for s-sparse polynomials is also a generator for a product of s-sparse polynomials. By Theorem 4.15 and Lemma 4.1 we get that there exists a generator $\mathcal{S}_{2s^2} : \mathbb{F}^t \to \mathbb{F}^n$ for the class of $(2s^2)$-sparse polynomials with $t = O(\log s + \log n)$. We now assume that, for some parameter t_{k-1}, we have a mapping $\mathcal{G}_{k-1} : \mathbb{F}^{t_{k-1}} \to \mathbb{F}^n$ that is a generator for sparse polynomials as well as for $\Sigma\Pi\Sigma\Pi(k-1)$ circuits of size s (recall Observation 4.3), and show how to construct a generator for c-compressed $\Sigma\Pi\Sigma\Pi(k)$ circuits of size s. Denote by $R(k)$ a number larger than the rank of every minimal and simple multilinear $\Sigma\Pi\Sigma(k)$ circuit that computes the zero polynomial. The discussion following the proof of Theorem 4.24 tells us that $R(k) = O(k^3 \log k)$. In the following lemma we show that if $c = kR(k)$ then when we restrict the variables in $[n] \setminus J$ to $\mathcal{G}_{k-1}$, we obtain a nonzero polynomial.

Lemma 4.32. Let $k \geq 2$ and $0 \not\equiv P \in \mathbb{F}[x_1,\dots,x_n]$ be computed by a simple multilinear $(k \cdot R(k))$-compressed $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s. In addition, let $\mathcal{G}_{k-1}$ be a generator for multilinear $\Sigma\Pi\Sigma\Pi(k-1)$ circuits of size s as well as for $(2s^2)$-sparse polynomials. Then, there is a set

$J \subseteq \mathrm{var}(P)$ of size $R(k)$ so that (recall the notations in Section 4.1)

$$P \circ \mathcal{G}_{k-1}^{\mathrm{var}(P)\setminus J} \not\equiv 0.$$

In words, after substituting the generator for the variables in $[n] \setminus J$, the resulting circuit remains nonzero.

Proof. Let $\Phi = \sum_{i=1}^{k} \prod_{j=1}^{r_i} P_{i,j}$ be a simple multilinear $(k \cdot R(k))$-compressed $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s computing P. If Φ is not minimal, then P can be computed by a $\Sigma\Pi\Sigma\Pi(k-1)$ circuit of size s and we are done. So we can assume without loss of generality that Φ is minimal. Let J be a set promised by Lemma 4.31 and let $\bar{J} = [n] \setminus J$. We can further assume without loss of generality that $|J| = k \cdot R(k)/k = R(k)$, by removing arbitrary indices from J if required. Via a reduction to depth-3 circuits, we describe how to find an assignment for $x_{\bar{J}} = (x_j)_{j \in \bar{J}}$ such that the restriction of P to that assignment in nonzero. For every $P_{i,j}$ in Φ, there is at most one variable x_ℓ so that $x_\ell \in J \cap P_{i,j}$. Let $f_{i,j}, g_{i,j}$ be such that $P_{i,j} = f_{i,j} x_\ell + g_{i,j}$ (if no such x_ℓ exists then $P_{i,j}$ is a constant). Note that two such polynomials P_{i_1,j_1}, P_{i_2,j_2} have a common factor iff $Q_{i_1,j_1,i_2,j_2} \stackrel{\mathrm{def}}{=} f_{i_1,j_1} \cdot g_{i_2,j_2} - f_{i_2,j_2} \cdot g_{i_1,j_1} \equiv 0$. Let $\mathcal{Q}$ be the set of all such nonzero Q_{i_1,j_1,i_2,j_2}'s. The following lemma gives a sufficient condition that a given assignment to $x_{\bar{J}}$ results in a simple, minimal, and nonzero depth-3 circuit. Let $\Phi_1, \ldots, \Phi_{2^k-2}$ be the proper sub-circuits of Φ, excluding the empty circuit. They are all $\Sigma\Pi\Sigma\Pi(k-1)$ circuits of size at most s.

Lemma 4.33. Let

$$\hat{\Phi} = \prod_{i=1}^{2^k-2} \Phi_i \cdot \prod_{Q \in \mathcal{Q}} Q.$$

Let $a \in \mathbb{F}^n$ be such that $\hat{\Phi}|_{x_{\bar{J}}=a_{\bar{J}}} \not\equiv 0$. Then $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is a simple and minimal multilinear $\Sigma\Pi\Sigma(k)$ circuit.

We will show that $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is nonzero after the proof of the lemma.

Proof. The circuit $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is minimal, since all of the sub-circuits of Φ are factors of $\hat{\Phi}$ and so if one of them is zero then so is $\hat{\Phi}|_{x_{\bar{J}}=a_{\bar{J}}}$. Due to the same reason, no $P_{i,j}$ is reduced to zero. By the choice of J, the size of $\mathrm{var}(P_{i,j})$ is at most one, which implies that $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is a $\Sigma\Pi\Sigma(k)$ circuit and it is clearly multilinear. We now show that $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is simple. For two polynomials f,g, write $f \nsim g$ if f/g is not a field element. Let $P_{i_1,j_1} \nsim P_{i_2,j_2}$ in Φ. If $\mathrm{var}(P_{i_1,j_1}|_{x_{\bar{J}}=a_{\bar{J}}}) \neq \mathrm{var}(P_{i_2,j_2}|_{x_{\bar{J}}=a_{\bar{J}}})$ then $P_{i_1,j_1}|_{x_{\bar{J}}=a_{\bar{J}}} \nsim P_{i_2,j_2}|_{x_{\bar{J}}=a_{\bar{J}}}$. If, on the other hand, $\mathrm{var}(P_{i_1,j_1}|_{x_{\bar{J}}=a_{\bar{J}}}) = \mathrm{var}(P_{i_2,j_2}|_{x_{\bar{J}}=a_{\bar{J}}})$ then as $Q_{i_1,j_1,i_2,j_2}|_{x_{\bar{J}}=a_{\bar{J}}} \not\equiv 0$ it follows that $P_{i_1,j_1}|_{x_{\bar{J}}=a_{\bar{J}}} \nsim P_{i_2,j_2}|_{x_{\bar{J}}=a_{\bar{J}}}$. As Φ is simple, we can thus conclude that $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is simple. □

We return to the proof of Lemma 4.32. The polynomial $\hat{\Phi}$ is a product of $(2s^2)$-sparse polynomials and $\Sigma\Pi\Sigma\Pi(k-1)$ circuits of size s each. Therefore, $\hat{\Phi}|_{x_{\bar{J}}=\mathcal{G}_{k-1}^{\bar{J}}} \not\equiv 0$. Lemma 4.33 thus implies that there exists $a \in \mathrm{Im}(\mathcal{G}_{k-1})$ for which $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is a simple, minimal, and multilinear $\Sigma\Pi\Sigma(k)$ circuit. The circuit $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ contains $R(k)$ variables (the previous proof shows that all the variables in J "survived") and any linear function appearing in it contains only one variable. The rank of $\Phi|_{x_{\bar{J}}=a_{\bar{J}}}$ is hence $R(k)$. By the definition of $R(k)$, this implies that $P \circ \mathcal{G}_{k-1}^{\bar{J}} \not\equiv 0$. □

We now show how to construct $\mathcal{G}_k : \mathbb{F}^{t_k} \to \mathbb{F}^n$ using $\mathcal{G}_{k-1}$. The construction uses the following two statements that were proved above. First, Theorem 4.29 implies that it suffices for $\mathcal{G}_k$ to satisfy the following property: for every subset I of the variables of size $|I| \leq 2\log n \cdot \log s \cdot k \cdot (kR(k))$, we can leave the variables in I "alive" while applying a generator for both $(kR(k))$-compressed circuits and for sparse polynomials to the rest of the variables. Indeed, the polynomial P_T given by Theorem 4.29 is a product of a $(kR(k))$-compressed circuit with sparse polynomials. Second, Lemma 4.32 applied to P_T tells us that there is a set $J \subseteq \mathrm{var}(P)$ of size $R(k)$ so that $P_T \circ \mathcal{G}_{k-1}^{\mathrm{var}(P)\setminus J} \not\equiv 0$.

Concluding, the only property that we need $\mathcal{G}_k$ to have is that it can keep "alive" any subset of the variables of size at most "$|I| + |J|$" $\leq 2\log n \cdot \log s \cdot k^2 R(k) + R(k) \leq 3\log n \cdot \log s \cdot k^2 R(k)$ while applying $\mathcal{G}_{k-1}$ to the rest of the variables. A generator that keeps variables "alive" was constructed in Ref. [127] in the context of PIT of sums of read-once formulas. Specifically, Shpilka and Volkovich [127] construct a map G_t from $\mathbb{F}^{2t}$ to $\mathbb{F}^n$ so that its image contains all points $a \in \mathbb{F}^n$ with at most t nonzero entries. In the next section we describe this construction. Joining all the different ingredients of this section together we get Theorem 4.28.

Theorem 4.34. Let $P \in \mathbb{F}[x_1,\ldots,x_n]$ be a nonzero polynomial computed by a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size s. Let G_t is the generator that keeps variables "alive," constructed in Section 4.7.2.1 below. Then for every $t \geq 3k^3R(k)\log(s)\log(n)$ it holds that $P(G_t(y,z) + \mathcal{S}_{2s^2}(w)) \not\equiv 0$, where y,z,w are vectors of new variables.

Proof. The proof is by induction on k. For $k = 1$, the polynomial P is a product of $(2s^2)$-sparse polynomials. By definition of $\mathcal{S}_{2s^2}$ and Observations 4.1 and 4.4, we get that $P(G_t + \mathcal{S}_{2s^2}) \not\equiv 0$, as claimed. Assume that $k \geq 2$. Let $I, J \subseteq [n]$ be the subsets guaranteed by Theorem 4.29 and Lemma 4.32, respectively, and set $U = I \cup J$. By the induction hypothesis, the mapping $G_m(y,z) + \mathcal{S}_{2s^2}(w)$ with $m = \lceil 3(k-1)^3R(k-1)\log(s)\log(n)\rceil$ is a generator for both $\Sigma\Pi\Sigma\Pi(k-1)$ circuits and for $(2s^2)$-sparse polynomials. From Theorem 4.29 and Lemma 4.32 (see the discussion above) it follows that there is a point $a \in \left(G_m^{[n]\setminus U} + \mathcal{S}_{2s^2}^{[n]\setminus U}\right)$ so that $P(a) \neq 0$. Since $|U| \leq 3k^2R(k)\log(s)\log(n) \leq t - m$, Observation 4.4 below implies

$$\mathrm{Im}\left(G_m^{[n]\setminus U} + \mathcal{S}_{2s^2}^{[n]\setminus U}\right) \subseteq \mathrm{Im}\left(G_m^{[n]\setminus U} + \mathcal{S}_{2s^2}\right) \subseteq \mathrm{Im}\left(G_t + \mathcal{S}_{2s^2}\right)$$

and thus $a \in \mathrm{Im}\left(G_t + \mathcal{S}_{2s^2}\right)$, as claimed. □

Theorem 4.34 shows the existence of a generator for $\Sigma\Pi\Sigma\Pi(k)$ circuits. To conclude the proof of Theorem 4.28 we observe the following facts: First, the degree of every coordinate of G_t and of $\mathcal{S}_{2s^2}$ is $n-1$. Furthermore, the generator $G_t(y,z) + \mathcal{S}_{2s^2}(w)$ has "seed-length" $O(k^3R(k)\log(s)\log(n) + \log(s) + \log(n))$. Therefore, Lemma 4.1 implies that there is a hitting set of size $n^{O(k^3R(k)\log(s)\log(n))} = n^{O(k^6\log k\log^2(s))}$ as claimed.

Very recently, Saraf and Volkovich improved the result of Theorem 4.28 [115]. In fact, they gave a black-box algorithm of running time $n^{\tilde{O}(k^2)}$. The main idea behind their proof is an extension of Theorem 4.22 to the case of multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits. Specifically, note that if a depth-3 multilinear multiplication gate has rank R, then, because of multilinearity, it computes an n^R-sparse polynomial. By replacing "rank" with "sparsity" Saraf and Volkovich [115] were able to show that if a simple and minimal multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit computes the zero polynomial then any of its multiplication gates computes a sparse polynomial. Using ideas similar to those of Klivans and Spielman [86] (see the proof of Theorem 4.15) they managed to gradually reduce the sparsity of the output polynomial. Applying their structural theorem they were able to prove that the output polynomial does not become zero in this process, thus obtaining an efficient way of transforming a nonzero polynomial computed by a $\Sigma\Pi\Sigma\Pi(k)$ circuit to a sparse polynomial. When the polynomial is sparse enough they simply run the [86] PIT test for sparse polynomial.

Theorem 4.35 ([115]). There is a deterministic $n^{O(k^2 \log k)}$ time black-box PIT algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits.

We mention an obvious next step question.

Open Problem 28. Give a sub-exponential time PIT algorithm for $\Sigma\Pi\Sigma\Pi(k)$ circuits.

4.7.2.1 The Generator of [127]

In this section we construct a map from $\mathbb{F}^{2t}$ to $\mathbb{F}^n$ with the following property: Its image contains all points $a \in \mathbb{F}^n$ with at most t nonzero entries. Throughout the entire section we fix a set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}$ of n distinct elements.[9]

[9] We assume w.l.o.g. that $|\mathbb{F}| > n$ as we are allowed to use elements from an appropriate extension field.

Definition 4.3. For every $i \in [n]$ let $u_i(w) : \mathbb{F} \to \mathbb{F}$ be the i-th Lagrange Interpolation polynomial for the set A. That is, each $u_i(w)$ is polynomial of degree $n-1$ satisfying $u_i(\alpha_j) = 1$ if $i = j$ and zero otherwise. For every $i \in [n]$ and $t \geq 1$ define $G_t^i(y_1,\ldots,y_t,z_1,\ldots,z_t) : \mathbb{F}^{2t} \to \mathbb{F}$ as

$$G_t^i(y_1,\ldots,y_t,z_1,\ldots,z_t) \overset{\text{def}}{=} \sum_{j=1}^{t} u_i(y_j) \cdot z_j \, .$$

Finally, let $G_t(y_1,\ldots,y_t,z_1,\ldots,z_t) : \mathbb{F}^{2t} \to \mathbb{F}^n$ be defined as

$$\begin{aligned} G_t(y_1,\ldots,y_t,z_1,\ldots,z_t) &\overset{\text{def}}{=} (G_t^1, G_t^2, \ldots, G_t^n) \\ &= \left(\sum_{j=1}^{t} u_1(y_j) \cdot z_j, \sum_{j=1}^{t} u_2(y_j) \cdot z_j, \ldots, \sum_{j=1}^{t} u_n(y_j) \cdot z_j \right). \end{aligned}$$

The following is an immediate observation that was used in the proof of Theorem 4.34.

Observation 4.4. The following properties hold:

(1) For every $t \geq 1$, it holds $G_t(\bar{y}, \bar{0}) \equiv 0$.
(2) Denote with $\bar{e}_i \in \{0,1\}^n$ the vector that has 1 in the i-th coordinate and 0 elsewhere. Then

$$G_{t+1} = G_t + \sum_{i=1}^{n} u_i(y_{t+1}) \cdot z_{t+1} \cdot \bar{e}_i.$$

Hence, for every $t \geq 1$ and $\alpha_m \in A$ we have that $G_{t+1}|_{y_{t+1}=\alpha_m} = G_t + z_{t+1} \cdot \bar{e}_m$.
(3) Let $t, m \in \mathbb{N}$, $I \subseteq [n]$ and $|I| \leq t$. Then, it holds that

$$\operatorname{Im}\left(G_t^{[n]\setminus I}\right) \subseteq \operatorname{Im}(G_{t+m}).$$

Property (4.4) is the one showing that G_t can keep any t variables "alive".

4.8 Read-Once Formulas

As mentioned above, in spite of the known lower bounds for multilinear formulas, there is no PIT algorithm even for multilinear $\Sigma\Pi\Sigma$ formulas (Open Problem 26). Due to this lack of progress, the model of read-once formulas (ROF) — a restricted model of multilinear formulas — was considered. Roughly, an ROF is an arithmetic formula in which every variable labels at most one leaf. It is clear that every ROF is also a multilinear formula. We now give the formal definition.

Definition 4.4. A *read-once arithmetic formula* (ROF) over a field $\mathbb{F}$ in the variables $\{x_1,\ldots,x_n\}$ is a binary tree as follows. Like in a formula, the leaves are labeled by variables and the internal nodes by $\{+,\times\}$. In addition, every node is labeled by a pair of field elements $(\alpha,\beta) \in \mathbb{F}^2$. Each input variable labels at most one leaf. The computation is performed in the following way. A leaf labeled by x_i and (α,β) computes $\alpha x_i + \beta$. If a node v is labeled by $\star \in \{+,\times\}$ and (α,β) and its children compute the polynomials f_1 and f_2, then v computes $\alpha(f_1 \star f_2) + \beta$.

We say that f is a *read-once polynomial* (ROP) if it can be computed by an ROF. An ROF that does not contain addition gates is called a *multiplicative ROF*. In a similar way, define *multiplicative ROP*.

Clearly, not every polynomial is an ROP. For example, in Ref. [126] it is shown that $x_1x_2 + x_2x_3 + x_1x_3$ is not an ROP.

It is not hard to see that there is a linear time white-box PIT algorithm for ROFs. The best deterministic black-box PIT algorithm for the problem, however, runs in time $n^{O(\log n)}$ [127]. The following theorem of Shpilka and Volkovich [127] gives such a black-box PIT algorithm for sums of a small number of ROFs.

Theorem 4.36 ([127]). There is a deterministic algorithm for deciding whether a polynomial $f = f_1 + \cdots + f_k$, where $f_1,\ldots,f_k$ are ROPs in $\{x_1,\ldots,x_n\}$ and $k \leq n/3$, is zero or not. The algorithm runs in time $n^{O(k)}$ in the white-box model and in time $n^{O(k+\log n)}$ in the black-box model.

The proof given by Shpilka and Volkovich [127] has three parts. First, they prove the theorem for $k = 1$. Second, they prove a hardness of representation result, which is basically a lower bound for sum of read-once formulas that have a certain property. Third, they show how to combine the PIT for a single ROP with the hardness of representation result to obtain the PIT for sums of ROPs.

We start with the proof for the case $k = 1$. Shpilka and Volkovich [127] proved that the mapping $G_{\log n}$ constructed in Section 4.7.2 is a generator for ROFs.

Theorem 4.37 ([127]). Let f be an ROP. If $f \not\equiv 0$ then $f \circ G_t$ is not constant for $t = \lceil \log n \rceil + 1$. Hence, if $S \subseteq \mathbb{F}$ is a set of size $n^2 + 1$, then $f \equiv 0$ iff $f \circ G_t$ is zero on S^{2t}. This gives a black-box PIT algorithm of running time $n^{4 \log n + O(1)}$ for ROFs.

Proof. The proof is by induction on the number of variables n. The simple idea behind the induction is that the two children of the output node compute two variable-disjoint ROPs. Hence, at least one of them contains at most $n/2$ variables. We therefore need to find a generator that does not cause too many cancellations between variable-disjoint ROPs. Since G_t allows us to keep $\Omega(t)$ variables "alive," we can show that it has the required property.

The case $n \leq 2$ is easily verified. For $n > 2$, consider the structure of the ROF computing f. If $f = \alpha(f_1 \times f_2) + \beta$ then the induction hypothesis immediately implies the claim for f. So we can assume $f = f_1 + f_2$ and without loss of generality $|\mathrm{var}(f_1)| \leq n/2$. By the induction hypothesis $f_1 \circ G_{t-1}$ is not constant. Since f is an ROF one can prove (by induction on the size of the ROF) that there exists some variable x_m such that $f_1(G_{t-1}^1, \ldots, G_{t-1}^{m-1}, x_m, G_{t-1}^{m+1}, \ldots, G_{t-1}^n)$ depends on x_m.[10] Thus, by Property 4.4 of Observation 4.4, $f_1 \circ G_t|_{y_t = a_m}$ depends on z_m, where y_t, z_m are variables of G_{t+1}. As x_m does not belong to $\mathrm{var}(f_2)$

[10] Surprisingly, this is not true for general polynomials, e.g., consider $g(x_1, x_2, x_3) = x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3$ over $\mathbb{F}_2$. Substituting $x_1 = x_2 = x_3 = y$ gives the polynomial $g(y, y, y) = y$. However, $g(x_1, y, y) = y$ and so it does not depend on x_1 and similarly for any other x_i.

(since f is an ROF) we get that $f \circ G_t|_{y_t=a_m} = f_1 \circ G_t|_{y_t=a_m} + f_2 \circ G_r|_{y_t=a_m}$ depends on z_m and therefore $f \circ G_t$ is not constant. In particular, $f \circ G_t$ is a nonzero polynomial of degree at most n^2 in each variable. Evaluating it on S^{2t} gives zero if and only if f is zero. $\square$

We now proceed to the second step. We first define the notion of a *justifying assignment* that in short can be described as an assignment that "preserves dependencies." This notion was first defined in Ref. [28] for the aim of reconstructing read-once formulas. Justifying assignments turn out to be a very important tool for dealing with ROFs.

Definition 4.5. We say that an assignment $a \in \mathbb{F}^n$ is a *justifying assignment* of a polynomial f if for every subset $I \subseteq \text{var}(f)$ we have that $\text{var}(f|_{x_I=a_I}) = \text{var}(f) \setminus I$. We also say that f is a-justified in this case. We say that an assignment a is a *weakly-justifying assignment* of f if the condition above holds when $|I| = 1$.

The following is a simple observation.

Observation 4.5. A multilinear polynomial f is a-justified iff for every $x_i \in \text{var}(f)$ it holds that[11] $\partial_{x_i}(f)(a) \neq 0$.

The following useful claim was proved in Ref. [127]. It basically shows how to use PIT for a given circuit class in order to construct a common justifying assignment for a set of polynomials constructed by circuits in this class. Here we prove the claim for the special case of ROFs, taking into account Theorem 4.37.

Lemma 4.38. Let $f_1, \ldots, f_k$ be n-variate ROPs. Let $T \subseteq \mathbb{F}$ be of size $|T| = kn^3 + 1$. Then $\mathcal{H}_n^k \overset{\text{def}}{=} G_t(T^{2t})$, for $t = \lceil \log n \rceil + 1$, contains a common justifying assignment for $f_1, \ldots, f_k$. Given the ROFs computing the f_i's, we can find this common justifying assignment in time $\text{poly}(n, k)$.

[11] In Ref. [126] the notion of discrete partial derivative was used. The *discrete partial derivative* of f with respect to x_i is defined as $f|_{x_i=1} - f|_{x_i=0}$. Notice that for multilinear polynomials this is the same as $\partial_{x_i}(f)$ and so we will use our notations instead.

Proof. For $i \in [n], j \in [k]$, set $g_i^j \stackrel{\text{def}}{=} \partial_{x_i}(f_j) \circ G_t$. As a partial derivative of an ROP is also an ROP, Theorem 4.37 implies that if $\partial_{x_i}(f_j) \not\equiv 0$ then $g_i^j \not\equiv 0$. Consider the polynomial

$$g \stackrel{\text{def}}{=} \prod_{i,j \mid g_i^j \not\equiv 0} g_i^j.$$

It is a nonzero $(2t)$-variate polynomial of degree at most kn^3 in each variable. As T is large enough, $g|_{T^{2t}} \not\equiv 0$. Equivalently, there exists $a \in T^{2t}$ so that if $g_i^j \not\equiv 0$ then $g_i^j(a) \neq 0$. Let $i \in [n], j \in [k]$ be such that $x_i \in \text{var}(f_j)$. Then $\partial_{x_i}(f_j) \not\equiv 0$ and hence $g_i^j \not\equiv 0$, which implies $\partial_{x_i}(f_j)(G_t(a)) = g_i^j(a) \neq 0$. By Observation 4.5 it follows that $G_t(a)$ is a justifying assignment for every f_j.

In the white-box case we can actually use the $O(n)$ time white-box PIT algorithm for ROFs in order to obtain a common justifying assignment. The idea is to find the entries of the point a one by one, using the PIT algorithm for the g_i^j's. For more details, see Ref. [126]. □

Lemma 4.38 tells us that there is a subset $\mathcal{H}_n^k \subset \mathbb{F}^n$ of size $|\mathcal{H}_n^k| = \text{poly}(k^{\log n}, n^{\log n})$ that contains a common justifying assignment for every set of k ROPs. Assume that $a = (a_1, \ldots, a_n) \in \mathbb{F}^n$ is such an assignment for $f_1, \ldots, f_k$. By shifting the variables $x_i \leftarrow x_i + a_i$ we can assume that the zero vector, $\bar{0} \in \mathbb{F}^n$, is justifying for all the f_i's. The following theorem of Shpilka and Volkovich [127] shows a lower bound for a sum of $\bar{0}$-justified read-once formulas.

Theorem 4.39. The monomial $\mathcal{P}_n = x_1 \cdot x_2 \cdots x_n$ cannot be computed as a sum of $k \leq n/3$ read-once formulas that are weakly-$\bar{0}$-justified.

The polynomial $x_1 \cdot x_2 \cdots x_n$ is an ROP that is not weakly-$\bar{0}$-justified. It is not difficult to see that using a simple interpolation (for the polynomial $f(y) = \prod_{i=1}^n (y + x_i)$), one can represent $\mathcal{P}_n$ as a sum of n read-once formulas that are weakly-$\bar{0}$-justified. Thus, the theorem above is almost tight.

Proof. The proof is by induction on k. The cases $k \in \{0,1\}$ follow by definition. We now assume that $k \geq 2$ and that $n \geq 3k$. Let $f_1,\dots,f_k$ be weakly-$\bar{0}$-justified ROPs over $\mathbb{F}[x_1,\dots,x_n]$. Assume toward a contradiction that $\sum_{i\in[k]} f_i = \mathcal{P}_n$. The idea of the proof is to eliminate a "large" number of ROPs at the cost of a "small" number of variables. Specifically, we find a small set of (indices of) input variables $J \subseteq [n-1]$ and a nonzero constant $\alpha \in \mathbb{F}$ so that if we take a partial derivative with respect to all of the variables in J and substitute $x_n = \alpha$ then we eliminate many f_i's in a way that the rest of the ROPs remain weakly-$\bar{0}$-justified. We thus get a representation of $\partial_J(\mathcal{P}_n)|_{x_n=\alpha} = \alpha \cdot \mathcal{P}_{n'}$, with a relatively large n', as a sum of a small number of weakly-$\bar{0}$-justified ROPs. We then use the induction hypothesis to reach a contradiction. We now proceed with the proof. There are two cases to consider.

Case 1: There exist $i \neq j$ in $[n]$ and m in $[k]$ such that $\partial_{x_i}\partial_{x_j}(f_m) \equiv 0$, namely, f_m does not contain $x_i x_j$ in any of its monomials. Assume without loss of generality that $i = n-1, j = n$, and $m = k$. It thus holds that $\sum_{i\in[k-1]} \partial_{x_n}\partial_{x_{n-1}}(f_i) = \mathcal{P}_{n-2}$. It may be the case that more than one f_m vanishes when we take a partial derivative with respect to x_n, x_{n-1}, however they cannot all vanish as $\mathcal{P}_n$ contains $x_n x_{n-1}$. Lemma 4.40 below (whose simple proof is omitted) guarantees that the polynomials $\partial_{x_n}\partial_{x_{n-1}}(f_1),\dots,\partial_{x_n}\partial_{x_{n-1}}(f_k)$ are also weakly-$\bar{0}$-justified ROPs.

Lemma 4.40. A partial derivative of a weakly-$\bar{0}$-justified ROP is a weakly-$\bar{0}$-justified ROP.

We thus obtain a representation of $\mathcal{P}_{n-2}$ as a sum of $0 < k' \leq k-1$ weakly-$\bar{0}$-justified ROPs such that $0 < 3k' \leq 3k-3 < n-2$, a contradiction.

Case 2: For every $i \neq j$ in $[n]$ and m in $[k]$ we have that $\partial_{x_i}\partial_{x_j}(f_m) \not\equiv 0$. In this case the ROFs for $f_1,\dots,f_k$ do not contain any $+$ gate, that is, they are multiplicative ROPs. The following lemma of Shpilka and Volkovich [127] is intuitive when thinking about the tree structure of a multiplicative ROF, yet the proof is rather technical and so we omit it.

Lemma 4.41. Let g be a multiplicative ROP with $|\mathrm{var}(g)| \geq 2$. Then, for every $x_i \in \mathrm{var}(g)$ there exists $x_j \in \mathrm{var}(g)$ such that $\partial_{x_j}(g) = (x_i - \alpha)h$ for some $\alpha \in \mathbb{F}$ and an ROP h such that $\mathrm{var}(h) = \mathrm{var}(g) \setminus \{x_i, x_j\}$ If, in addition, g is weakly-$\bar{0}$-justified then so is h. Moreover, in this case $\alpha \neq 0$ and there is at most one element $\beta \neq \alpha \in \mathbb{F}$ such that $g|_{x_i=\beta}$ is not weakly-$\bar{0}$-justified.

Lemma 4.41 implies that for every $m \in [k]$ there exist $j_m \in [n]$, a nonzero $\alpha_m \in \mathbb{F}$, and an ROP h_m such that $\partial_{x_{j_m}}(f_m) = (x_n - \alpha_m)h_m$. Let $A = \{\alpha_m \mid m \in [k]\}$. Clearly, $0 \notin A$. For every $\alpha \in A$, let $E_\alpha = \{m \in [k] \mid \alpha_m = \alpha\}$ and $B_\alpha = \{m \in [k] | \alpha_m \neq \alpha \text{ and } f_m|_{x_n=\alpha} \text{ is not weakly-}\bar{0}\text{-justified}\}$. Intuitively, E_α is set of the ROPs that can be eliminated by substituting $x_n = \alpha$ and B_α is set of ("bad") ROPs that will become non-weakly-$\bar{0}$-justified upon the substitution and thus require a special treatment. From the definition of A we have that $|E_\alpha| \geq 1$ and $\sum_{\alpha \in A} |E_\alpha| = k$. In fact, the E_α's form a partition of $[k]$. Lemma 4.41 implies that for each $\alpha \neq \alpha'$ in A the sets B_α and $B_{\alpha'}$ are disjoint (since for every ROP there exists at most one "bad" value of x_n) and therefore $\sum_{\alpha \in A} |B_\alpha| \leq k$. Hence, there exists $\alpha_0 \in A$ such that $|B_{\alpha_0}| \leq |E_{\alpha_0}|$ and $0 < |E_{\alpha_0}|$. Let $I = E_{\alpha_0} \cup B_{\alpha_0} \subseteq [k]$ and $J = \{j_m \mid m \in I\} \subseteq [n]$. By choice, $1 \leq |J| \leq |I| \leq |E_{\alpha_0}| + |B_{\alpha_0}| \leq 2|E_{\alpha_0}|$ and $n \notin J$. For every $m \in [k]$, define $f'_m = \partial_J(f_m)$, where $\partial_J(\cdot)$ is the partial derivative with respect to $\{x_j : j \in J\}$. We note the following properties: By Lemma 4.40, every f'_m is a weakly-$\bar{0}$-justified ROP. Secondly, for every $m \in I$ we have that $f'_m = (x_n - \alpha_m)h'_m$ for some ROP $h'_m(\bar{x})$. Indeed, as $j_m \in J$ we have that

$$\begin{aligned} f'_m &= \partial_J(f_m) = \partial_{J \setminus \{j_m\}}(\partial_{\{j_m\}}(f_m)) = \partial_{J \setminus \{j_m\}}((x_n - \alpha_m)h_m) \\ &= (x_n - \alpha_m) \cdot \partial_{J \setminus \{j_m\}} h_m. \end{aligned}$$

In addition, for every $m \in I$ we have that h'_m is a weakly-$\bar{0}$-justified ROP. This follows from the fact that a partial derivative of a weakly-$\bar{0}$-justified ROP is also a weakly-$\bar{0}$-justified ROP and the previous two properties.

For $m \in [k]$ consider the ROP $f''_m = f'_m|_{x_n=\alpha_0}$. Based on the above we can conclude the following: (1) For every $m \in E_{\alpha_0}$, since $\alpha_m = \alpha_0$, it holds that $f''_m = (\alpha_0 - \alpha_m)h'_m \equiv 0$. (2) For every $m \in B_{\alpha_0}$ we have that $f''_m = (\alpha_0 - \alpha_m)h'_m$ is a nonzero weakly-$\bar{0}$-justified ROP. In contrast to f_m, the structure of f'_m guarantees that it remains weakly-$\bar{0}$-justified when substituting $x_n = \alpha_0$. (3) For $m \in [k] \setminus I$ the definitions of E_{α_0} and B_{α_0} guarantee that $f_m|_{x_n=\alpha_0}$ is a weakly-$\bar{0}$-justified ROP. Lemma 4.40 implies that the same holds for f''_m. In this case it is also possible that $f''_m \equiv 0$.

Concluding, we get that $f''_m \equiv 0$ for $m \in E_{\alpha_0}$ and f''_m is a weakly-$\bar{0}$-justified ROP for $m \in [k] \setminus E_{\alpha_0}$. Without loss of generality, assume that $J = \{n'+1, n'+2, \ldots, n-2, n-1\}$ for some n'. Thus $\sum_{m\in[k]} f''_m = \partial_J(\mathcal{P}_n|_{x_n=\alpha_0}) = \alpha_0 \cdot \mathcal{P}_{n'}$. That is, we found a representation of $\alpha_0 \cdot \mathcal{P}_{n'}$ as a sum of weakly-$\bar{0}$-justified ROPs, where at least $|E_{\alpha_0}|$ of the ROPs are zeros. As $2|E_{\alpha_0}| \geq |J| = (n-1) - n'$ and $|E_\alpha| \geq 1$, we found a representation of $\alpha_0 \cdot \mathcal{P}_{n'}$ as a sum of $0 \leq k' < k$ weakly-$\bar{0}$-justified ROPs such that

$$0 \leq 3k' \leq 3(k - |E_\alpha|) = 3k - 3|E_\alpha| \leq n - 3|E_\alpha| \leq n' + 1 - |E_\alpha| \leq n'.$$

By induction we get that $\alpha_0 = 0$, which is a contradiction as $\alpha_0 \in A$ and $0 \notin A$. □

Using this hardness of representation result we get the claimed PIT for sum of ROFs.

Proof. [Proof of Theorem 4.36] Let $A_n^k = \{w \in \{0,1\}^n : \#$ of ones in w is at most $k\}$. Let $\mathcal{H}_n^k$ be the set constructed in Lemma 4.38. Define $\mathcal{J}_n^k = A_n^k + \mathcal{H}_n^k = \{v + u : v \in A_n^k \text{ and } u \in \mathcal{H}_n^k\}$. We now show that if $f = \sum_{i\in[k]} f_i \not\equiv 0$ then $f|_{\mathcal{J}_n^k} \not\equiv 0$. Indeed, let $\rho \in \mathcal{H}_n^k$ be a common justifying assignment to the f_i's. We will show that $\rho + A_n^k$ contains a nonzero assignment for f. By shifting the variables ($x_i \leftarrow x_i + \rho_i$) it is enough to prove the following claim.

Claim 4.42. Let $f = \sum_{i\in[k]} f_i \not\equiv 0$ be a sum of $\bar{0}$-justified ROPs. Then $f|_{A_n^k} \not\equiv 0$.

Proof. The proof is by induction on the number of variables in f. If $|\mathrm{var}(f)| \leq k$ then, since f is a multilinear polynomial, evaluating it on all zero-one assignments tells us whether it is identically zero or not. If $|\mathrm{var}(f)| > k$, on the other hand, consider $g_i = f|_{x_i=0}$ for each variable x_i. Each g_i is a $\bar{0}$-justified ROP. By the induction hypothesis, evaluating g_i on $A^k_{n-1} \subset A^k_n$ tells us whether g_i is identically zero or not. If one of the g_i's is not zero then so is f and we are done. If all the g_i's are zero then either $f \equiv 0$ or the polynomial $\mathcal{P}_n$ divides f. Indeed, if $f \not\equiv 0$ but $g_i \equiv 0$ then x_i is a factor of f. But now Theorem 4.39 tells us that if $f = \sum_{i\in[k]} f_i = c \cdot \mathcal{P}_n$ for some constant $c \in \mathbb{F}$, then $k > n/3$, which is a contradiction. □

The black-box identity testing algorithm now follows immediately: simply evaluate f on all points in $\mathcal{J}^k_n$. The running time is $|\mathcal{J}^k_n| = n^{O(k+\log n)}$. The algorithm in the white-box case is based on the same approach but is faster, due to the white-box part of Lemma 4.38 saying that we can find a common justifying assignment in time $\mathrm{poly}(n,k)$; having a common justifying assignment a at hand, we simply evaluate f on $A^k_n + a$. □

An interesting aspect of the proof of Theorem 4.36 is that it uses a lower bound for a very weak model (sum of weakly-$\bar{0}$-justified ROFs) in order to obtain a PIT algorithm. In addition, the lower bound is proved for a very easy polynomial, $\mathcal{P}_n$, which is difficult only when we consider weakly-$\bar{0}$-justified ROFs. Furthermore, the way that the lower bound is used is very different from the way suggested in Theorem 4.9. Specifically, the hitting set is constructed by considering the zero set of the "hard" polynomial $\mathcal{P}_n$. This zero set is very simple and very structured (it is the union of the halfspaces $x_i = 0$), and it can be shown that if f vanishes on all points in the hitting set then $\mathcal{P}_n$ is a factor of f. The lower bound is then invoked to imply that $f = 0$. It is an intriguing question whether this approach can be adapted to other scenarios to yield a new way of transforming a lower bound to a PIT algorithm.

Theorem 4.36 gives a quasi-polynomial time black-box algorithm for sum of k ROFs. No polynomial time black-box algorithm is known when $k = O(1)$. In particular the following question is still open.

Open Problem 29. Give a polynomial time black-box PIT algorithm for read-once formulas.

In Ref. [127] it was shown that (a slightly modified version of) Theorem 4.36 actually holds for the more general case of *preprocessed* read-once formulas. In this model we can replace x_i with a univariate polynomial $p_i(x_i)$. When we consider a sum of k ROFs, we can have a different polynomial $p_i^j(x_i)$ for each f_j. If $\deg(p_i^j) < r$ for all i, j then two modifications are required in order to get the PIT algorithm. The first is to slightly change the size of T in Lemma 4.38 to $O(kn^3r)$. The second is to replace A_n^k by the set of all vectors having at most k nonzero coordinates and taking values in S, where $S \subset \mathbb{F}$ can be taken to be any set of size $r+1$. These changes give a PIT algorithm for preprocessed ROFs of running time $(nr)^{O(k+\log n)}$ (some more work is required to prove that these changes indeed give the required result).

Another model that was considered in Ref. [127] is that of small depth read-once formulas. An interesting point in the definition is that such formulas have alternating levels of + gates and MUL gates, where an MUL gate can compute any *multiplicative* ROF in its inputs. This generalizes the usual notion of bounded depth formulas in which we have alternating levels of + and $\times$. The authors of Ref. [127] give an $n^{O(d+k)}$-time black-box PIT algorithm for the sum of k depth-d ROFs. Since multilinear $\Sigma\Pi\Sigma(k)$ circuits are actually a sum of k ROFs, this gives the currently best PIT algorithm for them.[12]

Theorem 4.43 ([127]). There is an $n^{O(k)}$-time black-box PIT algorithm for multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Jansen et al. [67] showed how to generalize Theorem 4.36 to get a PIT algorithm for a sum of k read-once arithmetic branching programs of roughly the same running time. The proof is similar to the proof of Theorem 4.36. In particular it is based on first solving the case $k=1$ and then proving a hardness of representation theorem using essentially the same ideas. The interested reader is referred to [67] for details.

Very recently, Anderson et al. [8] obtained a deterministic PIT algorithm for read-k formulas. Observe that this model contains as

[12] The theorem also holds if we consider preprocessed multilinear $\Sigma\Pi\Sigma(k)$ circuits, which form a sub-model of depth-4 circuits.

sub-models multilinear depth-3 and depth-4 formulas as well as sum of ROFs. The PIT algorithm of Anderson et al. [8] runs in time $n^{k^{O(k)}}$ in the white-box model and in time $n^{k^{O(k)}\cdot\log n}$ in the black-box model. While for the sub-models mentioned above the running time is worse than those given in Theorems 4.43, 4.28 and 4.36, the algorithm of Anderson et al. [8] works for a more general model. Interestingly, the approach of Anderson et al. combines the technique developed in the proofs of Theorems 4.28 and 4.36. The basic idea is to apply partial derivatives (using the technique of Theorem 4.36) to the formula and show that if we take the "correct" set of partial derivatives then the formula becomes a sum of c-compressed formulas. I.e., formulas that each of their factors depends on a relatively small number of variables. At this point, the algorithm reduces each factor to a linear form by fixing the values of most variables, using ideas similar to the proof of Theorem 4.28. Finally, using the rank bound for multilinear $\Sigma\Pi\Sigma(k)$ circuits (Theorem 4.22) the authors prove that the resulting depth-3 formula is nonzero. We stress that while this is the basic scheme, the proof is of course more complicated and does not use previous techniques in a straightforward manner.

Theorem 4.44 ([8]). There is a deterministic PIT algorithm for read-k multilinear formulas. The algorithm runs in time $n^{k^{O(k)}}$ in the white-box model and in time $n^{k^{O(k)}\cdot\log n}$ in the black-box model.

4.9 Relation to Other Problems

In this section we discuss the relation between PIT and other problems. We start by showing the relation to the problem of polynomial factorization. We then discuss the problem of read-once testing which is a generalization of the PIT problem.

4.9.1 Polynomial Factorization

Consider the following approach for PIT of multilinear formulas. Start by making the formula into a read-once formula, that is, a formula in which each variable labels at most one leaf. This can be done by replacing the j-th occurrence of x_i with a new variable $x_{i,j}$. then, check whether this formula is zero or not. If it is zero then the original formulas was also zero and we are done. Otherwise start replacing each $x_{i,j}$ with x_i. After each replacement we would like to verify that the resulting formula is still not zero. When replacing $x_{i,j}$ with x_i, we get zero iff the linear function $x_i - x_{i,j}$ is a factor of the formula that we have at hand. Thus, we somehow have to find a way of verifying whether a linear function is a factor of a multilinear formula. Furthermore, as we start with a read-once formula for which PIT is known (Section 4.8), we can assume that we know many inputs on which the formula is not zero. One may hope that before replacing $x_{i,j}$ with x_i we somehow managed to obtain inputs that will enable us to verify whether $x_i - x_{i,j}$ is a factor of the formula or not. This, of course, is not formal, but it does show the importance of understanding how to factor a multilinear formula given a PIT algorithm, or even just some (structured) set of nonzero inputs. As the different factors of a multilinear formula are variable disjoint this motivates the study of factorization of polynomials to variable-disjoint factors. Henceforth, following Shpilka and Volkovich [128], we refer to this problem as the *decomposition* problem.

Let $X = \{x_1, \ldots, x_n\}$ be the set of variables. For a set $I \subseteq [n]$, denote by X_I the set of variables whose indices belong to I. A polynomial $f(X)$ is said to be *decomposable* if it can be written as $f(X) = g(X_I) \cdot h(X_{[n]\setminus I})$ for some $I \subseteq [n]$ of size $0 < |I| < n$. The *indecomposable* factors of a polynomial $f(X)$ are polynomials $g_1(X_{I_1}), \ldots, g_k(X_{I_k})$ such that the I_j's are disjoint sets of indices, $f(X) = g_1(X_{I_1}) \cdot g_2(X_{I_2}) \cdots g_k(X_{I_k})$ and the g_i's are indecomposable. It is not difficult to see that every polynomial has a unique factorization to indecomposable factors, up to multiplication by field elements. The problem of polynomial decomposition is defined in the following way: Given an arithmetic circuit from some circuit class $\mathcal{M}$ computing a polynomial f, we have to output circuits for each of the indecomposable factors of f. If we only have a black-box access to f then we have to output a black-box for each of the indecomposable factors of f. We refer to this as black-box polynomial decomposition. Clearly, finding the indecomposable factors of a polynomial f is an easier task than finding all of the irreducible factors of f. However, for the natural class of multilinear polynomials these two problems are in fact the same.

The problem can be considered in its decision version as well. That is, given an arithmetic circuit computing a multivariate polynomial decide whether the polynomial is decomposable or not. In the decision version the algorithm just has to answer "yes" or "no" and is not required to find the decomposition.

Many randomized algorithms are known for factoring multivariate polynomials in the black-box and white-box settings (see, e.g., Refs. [72, 146, 147]). These algorithms clearly imply randomized algorithms for the decomposition problem. Similar to the case of PIT, it is a long-standing open question whether there is an efficient deterministic algorithm for factoring multivariate polynomials (see Refs. [79, 147]). Moreover, there is no known deterministic algorithm for the decision version of the problem and not even for the simpler case of multilinear polynomials. The authors of Ref. [128] showed that this is not a coincidence. Namely, they show that PIT and decomposing are closely related tasks. One direction is quite easy — showing that deterministic decomposing (even in its decision version) implies deterministic PIT. In what follows $\mathcal{M}$ is some fixed model of arithmetic circuits, for example, $\mathcal{M}$ can be the class of multilinear formulas, depth-4 circuits and so forth.

Observation 4.6. Assume that there is a deterministic algorithm for the decision version of the polynomial decomposition problem, that is, an algorithm that when given access (explicit or via a black-box) to a size s degree r circuit $C \in \mathcal{M}$ runs in time $T(s,r)$ and outputs "yes" iff the polynomial computed by C is decomposable. Then there is a deterministic algorithm that runs in time $O(T(s+2,r))$ and solves the PIT problem for size s degree r circuits from $\mathcal{M}$.

Proof. Let C be an arithmetic circuit. Consider $C' \overset{\text{def}}{=} C + y \cdot z$ where y,z are new variables. Note that C' is decomposable iff $C \not\equiv 0$ (in addition, C' is multilinear iff C is). □

The other direction is more involved and in fact we may need a PIT algorithm for a slightly larger class of circuits than the circuit that we wish to decompose. Given a circuit class $\mathcal{M}$, the circuit class $\mathcal{M}_V$ is defined as follows. The class $\mathcal{M}_V$ contains all circuits of the form $C_1 + C_2 \times C_3$ where $C_1, C_2, C_3 \in \mathcal{M}$ and C_2, C_3 are variable disjoint.

Theorem 4.45 ([128]). Assume that there is a deterministic algorithm that when given access (explicit or via a black-box) to a circuit $C \in \mathcal{M}_V$, of size $3s$ and degree r, runs in time $T(s,r)$ and decides whether $C \equiv 0$. Then, there is a deterministic algorithm that when given access (explicit or via a black-box) to a size s degree r circuit C' from $\mathcal{M}$, runs in time $O(n^3 \cdot r \cdot T(s,r))$ and decomposes C'. Moreover, each decomposable factor belongs to $\mathcal{M}$ and is of size at most s.

The theorem says that in order to find the decomposable factors of circuits from $\mathcal{M}$ it suffices to have a PIT algorithm for a slightly larger family of circuits $\mathcal{M}_V$. In fact, for most natural circuit classes, $\mathcal{M}_V$ is the same class as $\mathcal{M}$, e.g., when $\mathcal{M}$ is the class of multilinear formulas. On the other hand, when $\mathcal{M}$ is the class of ROFs then the theorem requires a PIT algorithm for the sum of two ROFs. It is an interesting question to prove a similar result using a PIT for $\mathcal{M}$ itself and not for a larger class.

Proof. Let f be a polynomial that is computed by a size s degree r circuit $C \in \mathcal{M}$. The proof actually gives an algorithm that returns a partition of $[n]$ to $I_1, \ldots, I_k$ such that the decomposition of f is $f = h_1(X_{I_1}) \cdot \cdots \cdot h_k(X_{I_k})$ for some indecomposable polynomials $h_1, \ldots, h_k$. We call the partition $\mathcal{I} = \{I_1, \ldots, I_k\}$ the *variable-partition* of f. The algorithm has the following four steps.

(1) Find a justifying assignment a to f.
(2) Recursively, find the variable-partition $\mathcal{I}'$ of $f|_{x_n = a_n}$.
(3) For every set $I \in \mathcal{I}'$, use the PIT to check whether $C(a) \cdot C \equiv C|_{x_I = a_I} \cdot C|_{x_{[n]\setminus I} = a_{[n]\setminus I}}$. If this is the case then add I to $\mathcal{I}$. If it is not the case, then move to the next I. At the end, add the remaining elements to $\mathcal{I}$. Namely, $\mathcal{I} \leftarrow \mathcal{I} \cup \{[n] \setminus \cup_{I \in \mathcal{I}} I\}$.
(4) For every $I \in \mathcal{I}$, let $h_I = f|_{x_{[n]\setminus I} = a_{[n]\setminus I}}$ and $\alpha = f(a)^{1-|\mathcal{I}|}$. Output $f = \alpha \prod_{I \in \mathcal{I}} h_I$.

We now explain why the algorithm works. Assume that the first step works, that is, that we found a justifying assignment a for f (this is given to us by assumption). The proof is by induction on n. The case $n = 1$ is trivial as a univariate polynomial is indecomposable. Now assume that $n > 1$. Let $f = h_1(X_{I_1}) \cdot \dots \cdot h_{k-1}(X_{I_{k-1}}) \cdot h_k(X_{I_k})$ be the decomposition of f, where $\mathcal{I} = \{I_1, \dots, I_k\}$ is its variable-partition. Assume without loss of generality that $n \in I_k$. Consider $f' = f|_{x_n = a_n}$. It holds that $f' = h_1 \cdot \dots \cdot h_{k-1} \cdot g_1 \cdot g_2 \cdot \dots \cdot g_\ell$ where the g_i's are the indecomposable factors of $h_k|_{x_n=a_n}$. Denote by $\mathcal{I}_k = \{I_{k,1}, \dots, I_{k,\ell}\}$ the variable-partition of $h_k|_{x_n=a_n}$. As a is a justifying assignment of f, we obtain that $\mathrm{var}(f') = \{x_i : i \in [n-1]\}$. From the uniqueness of the decomposition and by the induction hypothesis, the recursive application of the algorithm on f' returns the variable-partition of f', that is, $\mathcal{I}' = \{I_1, \dots, I_{k-1}, I_{k,1}, \dots, I_{k,\ell}\}$.

Lemma 4.46. Let $F \in \mathbb{F}[x_1, \dots, x_n]$ be a polynomial and let $a \in \mathbb{F}^n$ be a justifying assignment for it. Then for every $I \subseteq [n]$ of size $0 < |I| < n$, we have $F(a) \cdot F \equiv F|_{x_I = a_I} \cdot F|_{x_{[n]\setminus I} = a_{[n]\setminus I}}$ iff I is a disjoint union of sets from the variable-partition of F.

Proof. Assume that equality holds. Since F is a-justified, both $F|_{x_I=a_I}$ and $F|_{x_{[n]\setminus I}=a_{[n]\setminus I}}$ are nonzero, and hence $F(a) \neq 0$. Consequently, if we define $g(X_{[n]\setminus I}) = F|_{x_I=a_I}$ and $h(X_I) = (F|_{x_{[n]\setminus I}=a_{[n]\setminus I}})/F(a)$, we obtain that $F = g(X_{[n]\setminus I}) \cdot h(X_I)$. The uniqueness of the decomposition of F thus implies that I is a disjoint union of sets from the variable-partition of F.

To prove the other direction notice that we can write $F \equiv g(X_{[n]\setminus I}) \cdot h(X_I)$ for two polynomials g and h. This implies $F|_{x_I=a_I} \equiv g(X) \cdot h(a)$ and similarly $F|_{x_{[n]\setminus I}=a_{[n]\setminus I}} \equiv g(a) \cdot h(X)$. Hence, $F(a) \cdot F \equiv g(a) \cdot h(a) \cdot g(X) \cdot h(x) \equiv F|_{x_I=a_I} \cdot F|_{x_{[n]\setminus I}=a_{[n]\setminus I}}$. □

The lemma tells us that the third and fourth steps of the algorithm indeed find the decomposition of f. To finish the proof, we now analyze the running time of the algorithm. An analog of Lemma 4.38 shows that, given a PIT algorithm for circuits of the form $C_{x_i=1} - C_{x_i=0}$, that runs in time $T(s,r)$, when $C \in \mathcal{M}$ is of size s and degree r, we can find a justifying assignment for C in time $O(n^3 \cdot r \cdot T(s,r))$. We omit the proof of this extension and the interested reader can find it in Ref. [126]. Once we have a justifying assignment it is not difficult to see that the recursion runs in time $O(n^2 \cdot T(s,r))$. Thus, the total running time can be bounded from above by $O(n^3 \cdot r \cdot (T(s,r)))$. □

We note two interesting aspects of Theorem 4.45. The first is that the complexity of a decomposable factor of Φ is the same as the complexity of Φ. I.e., even when Φ comes from a restricted family of circuits we are guaranteed that its decomposable factors belong to the same restricted family. This is not known for the case of general factors, see, e.g., Open Problem 19. The second interesting aspect is that the theorem transforms a PIT algorithm for a weak class to decomposition of a slightly weaker class. In general, similar to Theorem 4.9, one can expect that a strong lower bound for arithmetic circuits will lead to strong derandomization results. However, it is not clear that a lower bound for a weak circuit class (like the one that follows from assuming the existence of a polynomial time black-box PIT algorithm, see Theorem 4.8) can help in derandomization, even for weaker classes.

4.9.2 Read-Once Testing

Consider the following generalization of the PIT problem. Given a polynomial, either explicitly by a circuit or as a black-box, decide whether it computes an ROP. Namely, decide whether there is a read-once formula computing it. Following [126] we refer to this problem as the *read-once testing* problem (ROT for short). Surprisingly enough, Shpilka and Volkovich [126] proved that this problem is (roughly) equivalent to PIT. That is, there is a deterministic algorithm for PIT iff there is a deterministic algorithm for ROT.

As in Theorem 4.45 we need a PIT algorithm for a slightly larger circuit class than the one we wish to do ROT for. In what follows we assume that $\mathcal{M}$ is some circuit class such that there is a deterministic PIT algorithm for the circuit class $\mathcal{M}_V$ (defined as before).

Theorem 4.47. There is a deterministic algorithm that, given an n-variate size s degree r circuit $C \in \mathcal{M}$, runs in time poly

$(n,r,s,T(s,r))$, where $T(s,r)$ is the cost of a single PIT algorithm for size s degree r circuits from $\mathcal{M}_V$, and outputs "yes" iff C is an ROP.

Proof. [Sketch] The algorithm consists of the following steps.

(1) Find a justifying assignment a for the polynomial computed by C. This can be done in time $O(n^3 \cdot r \cdot T(s,r))$ by a generalized version of Lemma 4.38.
(2) Reconstruct an ROF from the justifying assignment a. In Ref. [55] it was shown that given a black-box access and a justifying assignment for a polynomial f, one can construct, in time poly(n), a read-once formula Φ such that if f is an ROP then Φ computes f. At this stage we have Φ and all we have to do is verify whether C and Φ compute the same polynomial.
(3) Verify that $C \equiv \Phi$. The idea is to recursively ensure that every gate v of Φ computes the same polynomial as a "corresponding" restriction of C. To give a rough sketch, consider v, the root of Φ. Denote $\Phi = \alpha \cdot (\Phi_{v_1} \star \Phi_{v_2}) + \beta$, where v_1, v_2 are the two children of v and $\star \in \{+, \times\}$. Assume that the variables of v_i are I_i. The two sets I_1 and I_2 are disjoint. Consider the circuit $C_1 = C|_{x_{I_2}=a_{I_2}}$. Similarly define C_2, and the ROFs Φ_1 and Φ_2. Recursively verify that $C_i \equiv \Phi_i$. The only thing left is to verify that indeed $C \equiv \alpha \cdot (C_1 \star C_2) + \beta$. This can be done as we have a PIT algorithm for circuits in $\mathcal{M}_V$ which is exactly what we need. This step requires $O(n \cdot T(s,r))$ time. □

4.10 Concluding Remarks

In this chapter we surveyed most of the known deterministic identity testing algorithms. We saw that the main question is to find black-box algorithms for depth-4 circuits. We also saw that no algorithm is known even for $\Sigma\Pi\Sigma$ circuits and that current techniques can deal with circuits with a bound on either (1) the top fan-in or (2) the number of times a variable is read or (3) the number of different functions appearing in a multiplication gate. The only exception to this is the white-box algorithm of Raz and Shpilka [106] that handles the case of set-multilinear $\Sigma\Pi\Sigma$ circuits. We think that the first step toward a better understanding of $\Sigma\Pi\Sigma$ circuits (and hopefully $\Sigma\Pi\Sigma\Pi$ circuits) is answering Open Problem 27. Namely, giving an efficient deterministic black-box identity testing algorithm for set-multilinear $\Sigma\Pi\Sigma$ circuits. Another possible approach to $\Sigma\Pi\Sigma$ circuits is obtaining PIT algorithms for the symmetric model discussed at the end of Section 3.5: decide whether a given polynomial f of the form $f = \sigma_{r,m}(\ell_1,\ldots,\ell_m)$ is equivalently zero, where $\sigma_{r,m}$ is the degree r elementary symmetric polynomial in m variables and the ℓ_i's are linear functions in $x_1,\ldots,x_n$. This question is open even in the white-box model.

Open Problem 30. Give a deterministic PIT algorithm for polynomials of the form $f = \sigma_{r,m}(\ell_1,\ldots,\ell_m)$.

At the end of the chapter we saw a connection between PIT and polynomial decomposition. It is interesting to understand the relation between problems in RP. In particular it may be the case that derandomizing PIT implies derandomization of the general polynomial factorization problem. Note, however, that a decomposable factor of a circuit Φ has essentially the same complexity as Φ, even when Φ is, say, a sum of read-once formulas or a bounded depth circuit, etc. In the case of general factorization no such result is known, see Open Problem 19.

5

Reconstruction of Arithmetic Circuits

The *reconstruction problem* for arithmetic circuits is defined as follows. We are given a black-box "holding" an arithmetic circuit, that belongs to a predetermined family of circuits, e.g., bounded depth circuits, multilinear circuits, read-once formulas, etc., and that computes some multivariate polynomial f. We are allowed to ask for the value of f on inputs of our choice. We then have to come up with an arithmetic circuit computing the same polynomial of roughly the same complexity as the black-box circuit. The goal is to minimize the number of queries and running time required for constructing the circuit.

A reconstruction algorithm is efficient if its running time is polynomial in the circuit complexity of the black-box polynomial. This problem is the algebraic analog of the *learning with membership and equivalence queries problem* from computational learning theory of Boolean functions.[1] In the model of membership and equivalence queries, the learner can query the function on inputs of her choice (membership queries). In addition whenever the learner constructs a hypothesis she can ask for a counterexample, if one exists (an equivalence query). Namely, an input x such that the hypothesis

[1] For a good background on computational learning theory, see Ref. [83].

and the underlying function disagree on. Eventually the learner has to come up with a hypothesis that computes the function exactly. In the arithmetic setting equivalence queries can be replaced by membership queries, as by picking an input at random we can tell any two difference polynomials apart.

A closely related notion is that of interpolation of a polynomial. In the interpolation problem the goal of the learner is to find the coefficients of the underlying polynomial using membership queries. While this problem is closely related to the reconstruction problem it is not exactly the same. Note, however, that for sparse polynomials ($\Sigma\Pi$ circuits) interpolation and reconstruction are the same problem.

In the Boolean world, the PAC learning model is more often considered than the model of learning with membership and equivalence queries. In the PAC model the learner gets examples of the form $(x, f(x))$ where the input x is drawn from some unknown distribution $\mathcal{D}$ on inputs. She then has to come up with a hypothesis that agrees with f on most inputs, according to $\mathcal{D}$. In the arithmetic world, however, we believe that the reconstruction problem is more natural to study than the PAC learning one. One reason is that it seems natural to think of the input for a polynomial as coming from the uniform distribution and not from some "strange" distribution that PAC learning algorithms have to deal with. Another reason is that any two polynomials are far from each other, so any "reasonable" learning algorithm should compute the polynomial exactly.

It is clear that PIT algorithms are strongly related to reconstruction algorithms. E.g., a black-box algorithm provides a *test set* for the underlying class of circuits, that is, a set of points that distinguishes any two different circuits from the class. In other words, black-box PIT algorithms provide a set of points with the property that evaluating a circuit from the underlying class on all the points in the set, completely determines the circuit. The problem remains: "how to reconstruct the circuit from the values that it obtains on a test set?"

Currently, reconstruction algorithms are known for depth-2 circuits, see Refs. [18, 49, 51, 86] and references within, for depth-3 circuits with bounded top fan-in [77, 125] and for set-multilinear noncommutative formulas [17, 85]. It is an interesting question whether there is a generic way of transforming a black-box PIT algorithm to a reconstruction algorithm. On the face of it the answer should be negative as a random set should be a good test set and there is no clear way of using a random set of points to generate circuits. However, by considering PIT algorithms of a certain type, like the ones suggested by Agarwal [2] and Shpilka and Volkovich [127], it seems possible that one would be able to transform a successful PIT algorithm to a reconstruction algorithm. Consider, for example, the case of read-once formulas (ROFs). Although a weak model, it received a lot of attention in the learning community and several randomized learning algorithms were devised for it Refs. [26, 27, 28, 55]. In Shpilka and Volkovich [126, 127] a PIT algorithm for *sums* of read-once formulas was given. Consequently, Shpilka and Volkovich [127] obtained a deterministic quasi-polynomial time learning algorithms for read-once formulas. However, the problem of learning a sum of read-once formulas, even the sum of two read-once formulas, is still open.

Open Problem 31. Give sub-exponential reconstruction algorithm for the sum of a small number of read-once arithmetic formulas.

Although there is a clear relation between PIT and reconstruction, it may be the case that we will have randomized reconstruction algorithms without a corresponding deterministic PIT algorithm. For example, in Refs. [26, 27, 28, 55] randomized polynomial time reconstruction algorithms for ROFs were given, much before the designed of PIT algorithms for it. In addition, the PIT algorithm of Shpilka and Volkovich [127], that was discovered later, only implies a quasi-polynomial time reconstruction algorithm due to the size of the hitting set. We note however that the algorithm of [127] has the advantage of being deterministic whereas the previous algorithms [26, 27, 28, 55] are all randomized.

We now explain the main ideas behind some of the known reconstruction algorithms and discuss some known hardness results.

5.1 Hardness of Reconstruction

By experience, reconstruction algorithms exist only for classes for which strong lower bounds are known. It is thus natural to ask whether efficient reconstruction algorithms imply strong lower bounds for the underlying circuit class. If we could show such a result then we will immediately get that for classes for which lower bounds are hard to prove (like depth-4 circuits), one cannot expect to obtain an efficient reconstruction algorithm. In the Boolean world such results were first proved by Valiant [142] (for more on hardness of learning see Ref. [87] and references within). In the arithmetic world much less is known. This is mostly due to the fact that most cryptographic assumptions that were used to prove hardness of learning do not have counterparts in the arithmetic world (recall the discussion in Section 3.9). Nevertheless, recently some hardness of reconstruction results were proved for arithmetic circuits as well.

In Ref. [87], Klivans and Sherstov proved that assuming the hardness of the shortest vector problem (SVP) for *quantum* algorithms, no *PAC* learning algorithm exists for $\Sigma\Pi\Sigma$ circuits. The $f(n)$-SVP question is the problem of computing an $f(n)$ multiplicative approximation to the length of the shortest vector in a given lattice in $\mathbb{R}^n$. Regev constructed a new public-key cryptosystem based on the assumption that $\tilde{O}(n^{1.5})$-SVP is hard for quantum computers [113]. Klivans and Sherstov observed that the decrypting function for Regev's public-key encryption scheme can be computed by small $\Sigma\Pi\Sigma$ circuits. Using a lemma of Kearns and Valiant [82] that basically says that if a decryption function of a secure public-key cryptosystem can be computed by a circuit class $\mathcal{C}$ then $\mathcal{C}$ does not have efficient learning algorithms, Klivans and Sherstov [87] concluded that no efficient PAC learning algorithms exist for $\Sigma\Pi\Sigma$ circuits (assuming the security of Regev's public-key cryptosystem).

Theorem 5.1 ([87]). Assume that polynomial-size $\Sigma\Pi\Sigma$ arithmetic circuits are PAC-learnable in polynomial time. Then there is a polynomial-time quantum solution to $\tilde{O}(n^{1.5})$-SVP.

While this theorem provides a hardness of PAC learning result for $\Sigma\Pi\Sigma$ circuits, it does not say much about the general reconstruction problem. The main difference between these problems is that in the case of reconstruction we are allowed to ask for the value of the circuits on inputs of our choice. Furthermore, in the reconstruction problem we usually consider inputs coming from the uniform distribution. On the other hand, in the PAC model the goal is to learn the circuit using random examples that come from some unknown distribution. Specifically, the adversary is allowed to choose a distribution that is supported on a "strange" set of inputs. Thus, giving a PAC learning algorithm that works for any distribution may be a more difficult task than reconstructing the circuit.

Fortnow and Klivans [46] show hardness of learning results for arithmetic circuits and formulas using ideas similar to those of Kabanets and Impagliazzo [69].

Theorem 5.2 ([46]). Let $\mathcal{C}$ be a family of polynomial size arithmetic formulas. Assume that $\mathcal{C}$ is exactly learnable from membership and equivalence queries in polynomial time and that the hypothesis of the learner is an arithmetic formula. Then there exists a polynomial $f \in \mathsf{EXP}^{\mathsf{RP}}$ such that $f \notin \mathcal{C}$. If we allow both $\mathcal{C}$ and the hypothesis to be arithmetic circuits then there exists an $f \in \mathsf{ZPEXP}^{\mathsf{RP}}$ such that $f \notin \mathcal{C}$.

When saying $f \in \mathsf{EXP}^{\mathsf{RP}}$, etc. we mean that there is an $\mathsf{EXP}^{\mathsf{RP}}$ machine that computes f (when the input is given in bits).

Proof. [Sketch] The idea of the proof is similar to the idea behind Theorem 4.7. Assume that there is a learning algorithm for $\mathcal{C}$, using membership and equivalence queries. Assume further that permanent is computable by a polynomial size circuit from $\mathcal{C}$. We shall show, using the self-reducibility of permanent, that this implies that permanent is in $\mathsf{ZPP}^{\mathsf{RP}}$. This is done by an iterative construction of circuits for PERM_m, permanent of $m \times m$ matrices, as follows. When $m = 1$ this is trivial. Assume now that we constructed a circuit for PERM_m. To construct a circuit for PERM_{m+1}, run the learning algorithm for $\mathcal{C}$ with PERM_{m+1} as the objective polynomial. The problem is, of course, to answer membership and equivalence queries. Using the circuit for PERM_m, we can easily answer membership queries for PERM_{m+1}. Equivalence queries can be simulated using randomness: since the constructed circuit is small, its degree is not too high, which implies that a random input will be a counterexample. We can thus learn a small circuit for PERM_n.

To conclude the proof we note the following. If permanent has polynomial size circuits from $\mathcal{C}$ and if $\mathsf{EXP} \subseteq \mathcal{C}$ (otherwise there is nothing to prove), then, as in the proof of Theorem 4.7, permanent is complete for EXP. Hence $\mathsf{EXP} \subseteq \mathsf{ZPP}^{\mathsf{RP}}$. By padding this implies that $\mathsf{EE} \subseteq \mathsf{ZPEXP}^{\mathsf{RP}}$ (where $\mathsf{EE} = \mathsf{DTIME}(2^{2^{\mathsf{O(n)}}})$). As EE contains functions with super-polynomial circuit complexity, so does $\mathsf{ZPEXP}^{\mathsf{RP}}$. $\square$

We end this section by asking whether a similar result can be proved for VNP.

Open Problem 32. Prove that if an arithmetic circuit class $\mathcal{C}$ is learnable then the permanent does not have polynomial size circuits from $\mathcal{C}$.

5.2 Interpolation of Sparse Polynomials

Similar to the PIT algorithms from Section 4.4, many different interpolation algorithms were designed for the class of sparse polynomials (see, e.g., Refs. [18, 51, 86] and references within). As before, we describe the approach of Klivans and Spielman [86]. The PIT algorithm given in Section 4.4 can be easily extended to an interpolation algorithm. Assume that the unknown polynomial is $f(x_1,\ldots,x_n)$ with $\deg(f) < r$. Recall the substitution $x_i = y^{k^i \mod p}$, for a large enough prime p. The proof of Theorem 4.15 shows that for some k, each monomial of f is mapped uniquely to a monomial of $\hat{f} \overset{\text{def}}{=} f(y, y^{k^1 \mod p}, \ldots, y^{k^{n-1} \mod p})$. By evaluating this polynomial on $\deg(\hat{f}) + 1$ points we can recover all the coefficients of f. It remains to find the monomials corresponding to each of the coefficients. To do so let $\gamma \notin \{0,1\}$ be some field element. Consider the polynomials $f_i \overset{\text{def}}{=} f(x_1,\ldots,x_{i-1},\gamma \cdot x_i, x_{i+1},\ldots,x_n)$. As before, we can interpolate the polynomials $\widehat{f_i} = f_i(y, y^{k^1 \mod p}, \ldots, y^{k^{n-1} \mod p})$. The important observation is that the monomials of $\widehat{f_i}$ are the same as the monomials of $\hat{f}$ and the only difference is that the coefficients of the monomials that resulted from monomials containing x_i changed. By comparing the coefficients of $\widehat{f_i}$

and those of $\hat{f}$, we can easily recover the monomials of f, thus obtaining the following theorem.

Theorem 5.3 ([86]). In time polynomial in n, m, r, and $\log(|\mathbb{F}|)$, we can output a test set of point that given the values of a degree r n-variate polynomial over $\mathbb{F}$ with at most m monomials at every point in the set, we can find the coefficients and monomials of the polynomial in polynomial time. If $\mathbb{F} = \mathbb{R}$, then every entry of each point in the set has bit length at most $O(\log(nr))$. When $\mathbb{F}$ is finite of size smaller than $(nr)^6$ the entries of the points come from the smallest algebraic extension field of $\mathbb{F}$ of size at least $(nr)^6$.

5.3 Learning via Partial Derivative

Beimel et al. [17] gave learning algorithms that use membership and equivalence queries for several classes of Boolean functions. The output of the algorithm is a *multiplicity automaton* (see definition below). They mainly considered Boolean function classes but they also applied their algorithms to the class of low degree polynomials over finite fields, and even to a model that can be thought of as set-multilinear depth-3 circuits. In Ref. [85] it was noticed that this learning algorithm can actually learn any class of arithmetic circuits that compute polynomials whose space of partial derivatives has low dimension. In this section we define the model of multiplicity automata, give the algorithm of Beimel et al. [17] and explain the connection to partial derivatives.

Definition 5.1. A multiplicity automaton $\mathcal{A}$ of size s over $\mathbb{F}$ consists of a vector $\gamma = (\gamma_1, \ldots, \gamma_s) \in \mathbb{F}^s$ and a set of matrices $\{A_\sigma\}_{\sigma \in \mathbb{F}}$, where each A_σ is an $s \times s$ matrix over $\mathbb{F}$. The output of $\mathcal{A}$ on input $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ is defined to be the first coordinate of the vector[2] $(\prod_{i=n}^{1} A_{x_i})\gamma$.

Intuitively, each matrix A_σ corresponds to the transition matrix of the automaton for the symbol $\sigma \in \mathbb{F}$. Iterative matrix multiplication

[2] We denote $A_{x_n} \cdot A_{x_{n-1}} \cdot \cdots \cdot A_{x_1}$ by $\prod_{i=n}^{1} A_{x_i}$.

keeps track of the weighted sum of paths from state i to state j. The first row of the iterated product corresponds to transition values starting from the initial state and γ determines the acceptance criteria. This definition can be extended to any input length and not just to n-tuples, but we restrict our discussions to inputs from $\mathbb{F}^n$.

Assume that a polynomial $f(x_1,\ldots,x_n)$ with individual degrees bounded by r can be computed by a multiplicity automaton of size s. Let $\alpha_0,\ldots,\alpha_r$ be distinct field elements. Let $A(z)$ be the unique $s \times s$ matrix whose entries are degree r polynomials in the variable z such that for every α_i it holds that $A(\alpha_i) = A_{\alpha_i}$. One can construct $A(z)$ by interpolation. Observe that this implies that $f(x_1,\ldots,x_n) = (A(x_n)\cdots A(x_1)\cdot\gamma)_1$. This representation immediately connects multiplicity automata to read-once oblivious algebraic branching programs. Specifically, f can be computed by an ABP of width s, such that the only variable labeling edges between the i-th level and the $i+1$'st level is x_i. An important notion related to multiplicity automata is that of Hankel matrices.

Definition 5.2. Let $f : \mathbb{F}^n \to \mathbb{F}$ be a polynomial. We construct a matrix H whose rows and columns are indexed by strings[3] in $\mathbb{F}^{\leq n}$ in the following way. For a string x, define $|x|$ to be the length of x, for example, $|01| = 2$. For two strings x,y, define the (x,y) entry of H to be $f(x \circ y)$ if $|x| + |y| = n$ and 0 otherwise. The resulting matrix H is called the *Hankel matrix* of f. Define H_k to be the k-th block of H, i.e., H_k is the sub-matrix of H define by all rows x so that $|x| = k$ and all columns y so that $|y| = n - k$.

The following key fact relates the rank of the Hankel matrix of a polynomial with the size of a multiplicity automaton computing it.

Theorem 5.4. The rank of the Hankel matrix of f (over $\mathbb{F}$) is equal to the size of the smallest multiplicity automaton computing f.

[3] The set $\mathbb{F}^{\leq n}$ is the set of strings of length at most n over the alphabet $\mathbb{F}$. For practical purposes, if $\mathbb{F}$ is too large then we can consider the Hankel matrix with respect to strings in $S^{\leq n}$ for some $S \subseteq \mathbb{F}$. As long as $|S| > \deg(f)$ the rest of the discussion remains the same.

Proof. [Sketch] We only prove one direction here as it will be the one used in our algorithms. Let H be the Hankel matrix of an n-variate polynomial f and assume that $\mathrm{rank}(H) = s$. Let $a_1, a_2, \ldots, a_s \in \mathbb{F}^{\leq n}$ be such that the rows indexed by them form a basis to the row space of H (we can assume that $f \neq 0$). Without loss of generality, assume that a_1 is the empty string, denoted $a_1 = \varepsilon$. Set $\gamma = (f(a_1), \ldots, f(a_s))$ where if $|a_i| \neq n$ then we set $f(a_i) = 0$. For every $\sigma \in \mathbb{F}$, define the matrix A_σ as follows: the i-th row of A_σ is the unique vector $(\alpha_1, \ldots, \alpha_s)$ satisfying $H_{a_i \circ \sigma} = \sum_{i \in [s]} \alpha_i \cdot H_{a_i}$, where H_w is the row of H indexed by w. One can now prove by induction on $|w|$ that for every $w \in \mathbb{F}^{\leq n}$ it holds that $(A_w \cdot \gamma)_i = f(a_i \circ w)$. □

The learning algorithm works in stages. In each stage it learns another row of the Hankel matrix that is independent of the rows learned so far. When a basis to the row space of the matrix is obtained the algorithm constructs a multiplicity automata that has the same Hankel matrix as the underlying polynomial, which is what we are after. More specifically, at the beginning of the k-th iteration, the algorithm holds a set of rows $X = \{x_1, \ldots, x_k\} \subseteq \mathbb{F}^{\leq n}$ and a set of columns $Y = \{y_1, \ldots, y_k\} \subseteq \mathbb{F}^{\leq n}$. For a string z, let $\hat{H}_z$ be the restriction of the row H_z to the k coordinates in Y. Given z and Y, the vector $\hat{H}_z$ can be computed using $k = |Y|$ membership queries. It will hold that $\hat{H}_{x_1}, \ldots, \hat{H}_{x_k}$ are linearly independent. Using these vectors the algorithm constructs a hypothesis h, in a manner similar to the proof of Theorem 5.4, and asks an equivalence query. A counterexample to h leads to adding a new element to both X and Y, in a way that preserves the above properties. This immediately implies that the number of iterations is bounded by $\mathrm{rank}(H)$. We shall now give a more detailed description of the algorithm. To simplify the algorithm we redefine f so that $f(\varepsilon) = 1$ (f remains zero on all other strings of length different than n).

(1) Set $x_1 = y_1 = \varepsilon$, $X = Y = \{\varepsilon\}$ and $k = 1$. At this point, $\hat{H}_\varepsilon$ has one coordinate which equals 1.
(2) Construct a hypothesis h, as in the proof of Theorem 5.4. Namely, set $\gamma = (f(x_1), \ldots, f(x_k))$. For every[4]

[4] Actually, if f has degree at most r in each variable then we only need to do so for all $\sigma \in S$ where $S \subseteq \mathbb{F}$ is of size $r + 1$.

$\sigma \in \mathbb{F}$, construct the matrix $\hat{A}_\sigma$ as follows: its i-th row is the coefficients of the vector $\hat{H}_{x_i \circ \sigma}$ when expressed as a linear combination of the vectors $\hat{H}_{x_1}, \ldots, \hat{H}_{x_k}$. This is possible as $\hat{H}_{x_1}, \ldots, \hat{H}_{x_k}$ are linearly independent vectors in $\mathbb{F}^k$. These matrices, together with γ, define a multiplicity automaton and our hypothesis h is the function computed by this automaton.

(3) Ask whether $h = f$. If the answer is YES then halt and output h. Otherwise the answer is NO and we obtain a counterexample z (this is the equivalence query). We now find a prefix $w \circ \sigma$ of z such that there exist constants $\alpha_1, \ldots, \alpha_k \in \mathbb{F}$ satisfying $\sum_{i \in [k]} \alpha_i \hat{H}_{x_i} = \hat{H}_w$ but for some $y \in Y$ it holds that $\sum_{i \in [k]} \alpha_i \hat{H}_{x_i}(\sigma \circ y) \neq \hat{H}_w(\sigma \circ y)$. Namely, the way in which the row w is spanned by $\hat{H}_{x_1}, \ldots, \hat{H}_{x_k}$, cannot be extended to the column $\sigma \circ y$. Now, set $X \leftarrow X \cup \{w\}$ and $Y \leftarrow Y \cup \{\sigma \circ y\}$. Repeat Step 2.

The correctness of the algorithm is implied by the following two claims that we leave to the reader. The proof of the first claim is by induction on the structure of z and it uses essentially the same kind of arguments as the proof of Theorem 5.4.

Claim 5.5. Let z be a counterexample to h found in Step 5.3 (i.e., $f(z) \neq h(z)$). Then, there exists a prefix $w \circ \sigma$ having the required properties.

The second claim follows by construction.

Claim 5.6. At each step the rows $\hat{H}_{x_1}, \ldots, \hat{H}_{x_k}$ are linearly independent.

Concluding, we obtain the following theorem.

Theorem 5.7 ([17]). Let $f : \mathbb{F}^n \to \mathbb{F}$ be a polynomial of individual degrees bounded by r such that $\mathrm{rank}(H(f)) = R$ over $\mathbb{F}$, where $H(f)$ is the Hankel matrix of f. Then, f is exactly learnable by the above algorithm in time poly(n, r, R) from equivalence and membership queries.

To explain the relation between Hankel matrix and partial derivatives we introduce some notations. For a function $f(x_1,\ldots,x_n)$ and $k \le n$, define $\partial_k(f) = \{\frac{\partial f}{\partial M} \mid M \text{ is a monomial in } x_1,\ldots,x_k\}$ and $\mathrm{rank}_k(f) = \dim(\mathrm{span}(\partial_k(f)))$.

Theorem 5.8 ([85]). Let $f(x_1,\ldots,x_n)$ be a degree r polynomial. Then for every $k \le n$, the k-th block of the Hankel matrix $H(f)$ of f admits $\mathrm{rank}(H_k(f)) \le \mathrm{rank}_k(f)$. If f is multilinear then equality holds.

The proof of the theorem is quite easy and relies on Taylor expansion of a polynomial and therefore we omit it. Combining Theorems 5.7 and 5.8 we get learning algorithms for several circuit classes.

Theorem 5.9 (Learnability of depth-3 circuits). Let f be computed by a set-multilinear $\Sigma\Pi\Sigma$ circuit with s product gates over n variables. Then f is learnable, from membership and equivalence queries, in time poly(n, s). If f is computed by a $\Sigma\Pi\Sigma$ circuit with s multiplication gates each of degree at most r then f is learnable in time poly$(n, 2^r, s)$.

The prove of the theorem is immediate once we notice the following. In the case of set-multilinear circuits, $\sum_{k\in[n]} \mathrm{rank}_k(f) \le ns$. In the case of general $\Sigma\Pi\Sigma$ circuits, $\sum_{k\in[n]} \mathrm{rank}_k(f) \le 2^r ns$.

Finally, we obtain a learning algorithm for set-multilinear noncommutative formulas (this generalizes the first case of Theorem 5.9).

Theorem 5.10. Let $f(X_1,\ldots,X_r)$ be a set-multilinear polynomial in the variables $X_1,\ldots,X_r$, the size of each $|X_i|$ is n. Assume that f is computed by a set-multilinear noncommutative formula of size s. Then f can be learned in time poly(n, r, s).

The proof follows since $\sum_{k\in[n]} \mathrm{rank}_k(f) \le ns$ (see the proof of Theorem 3.8 in Section 3.4).

5.4 Reconstruction of Depth-3 Circuits

So far we have seen learning algorithms for classes of circuits computing polynomials whose partial derivatives span a low dimensional space (sparse polynomials have this property as well). In this section we will give an example of a reconstruction algorithm for the class of $\Sigma\Pi\Sigma(k)$ circuits. This class can compute polynomials whose partial derivatives space has an exponentially large dimension. For example, the polynomial $f = (x_1 + x_2) \cdot (x_3 + x_4) \cdots (x_{n-1} + x_n)$ satisfies $\mathrm{rank}_k(f) \geq \binom{n/2}{k}$.

The reconstruction algorithm presented returns a $\Sigma\Pi\Sigma$ circuit. When the "original" circuit is multilinear, the algorithm outputs a multilinear $\Sigma\Pi\Sigma(k)$ circuit and runs in polynomial time. When the circuit is not multilinear, the algorithm returns a $\Sigma\Pi\Sigma$ circuit with quasi-polynomially many multiplication[5] gates and runs in quasi-polynomial time. This algorithm is very different from the previous ones and it heavily relies on Theorem 4.22.

Theorem 5.11 ([77]). Let f be an n-variate polynomial computed by a $\Sigma\Pi\Sigma(k)$ circuit of degree d over $\mathbb{F}$. Then there is a reconstruction algorithm for f, that runs in time $\mathrm{poly}(n) \cdot \exp(\log(|\mathbb{F}|) \cdot \log(d)^{O(k^3)})$, and outputs a $\Sigma\Pi\Sigma(K,d)$ circuit for f, where $K = \exp(\log(d)^k)$. When $|\mathbb{F}| = O(d^5)$ the algorithm may ask queries from a polynomial size algebraic extension field of $\mathbb{F}$.

In the case that f can be computed by a multilinear $\Sigma\Pi\Sigma(k)$ circuit, there is a reconstruction algorithm that runs in time $(n + |\mathbb{F}|)^{2^{O(k \log k)}}$ and outputs a multilinear $\Sigma\Pi\Sigma(k)$ circuits for it. When $|\mathbb{F}| < n^5$ The algorithm may ask queries from a polynomial size algebraic extension field of $\mathbb{F}$.

The rest of the section is organized as follows. First we will explain the idea behind the algorithm of Shpilka [125] that works for circuits with two multiplication gates ($k = 2$) and then the extension of Ref. [77] that works for the case of $k = O(1)$ multiplication gates.

[5] By generalizing the notion of a multiplication gate one can view the output of the algorithm as a sum of k generalized multiplication gates (see Ref. [77]).

5.4.1 The Case $k = 2$

Let us start with the simplest case $k = 1$. Here the circuit is simply a product of linear functions so all we need to do is to find its irreducible factors. This is easily done by using the black-box factoring algorithm of Kaltofen and Trager [73].

We would like to extend this approach for the case of two product gates. Obviously, we run into difficulties as the circuit itself can be irreducible. Furthermore, even if it is reducible then there is no reason that we could recover the linear functions from its irreducible factors. To overcome this difficulty [125] had the following idea. Let $\Phi = M_1 + M_2$ be the unknown circuit, each M_i is a product gate. Look for a linear function ℓ that is a factor of M_1 and not of M_2 (if such a function exists). Consider the circuit $\Phi|_{\ell=0}$. This circuit has exactly one multiplication gate $M_2|_{\ell=0}$ and so we can recover all its irreducible factors. Now, this still does not give us M_2 but rather only its restriction to the space $\{x : \ell(x) = 0\}$. To overcome this, Shpilka [125] actually looks for many different functions $\ell_1,\ldots,\ell_t$ that are linearly independent and divide M_1 and not M_2. The algorithm then computes $M_2|_{\ell_1=0},\ldots,M_2|_{\ell_t=0}$ and from these t different circuits reconstructs M_2. Once M_2 is known we can look at $\Phi - M_2$ and recover M_1 using factoring.

There are, of course, many problems with this approach. The first is ensuring that such linear functions even exist. If the rank of the circuit is small, for example, then they do not exist. The second problem is that even if such functions exist, we need to find them, and, after all, the space of linear functions is exponentially large. The third problem is that even if we overcome the first two obstacles, we still need to reconstruct M_2 from the different circuits $M_2|_{\ell_i=0}$.

We now explain how to overcome each of these difficulties. As the algorithm is very complicated and contains many details, we only sketch the ideas and avoid small (yet delicate) points that are dealt with in Ref. [125]. In what follows, to ease the reading, we assume that Φ is simple, namely, that its product gates have no common factors. The algorithm, of course, must deal with the non-simple case.

We first explain how the algorithm works when the rank of Φ is larger than $O(\log^2(n))$. We would like to find linear functions $\ell_1,\ldots,\ell_t$,

$t = O(\log n)$, such that all the ℓ_i's divide M_1 but none of them divides M_2. To do this, instead of considering the problem over $\mathbb{F}^n$, we restrict the inputs to the circuits to some random linear space $V \subset \mathbb{F}^n$ of dimension $\dim(V) = D$. It is not very difficult to show that with high probability, $\Phi|_V$ still has high rank and that the functions $\ell_i|_V$ still do not divide $M_2|_V$. Now, since the dimension of the space is rather small we can actually "guess"[6] the functions $\ell_1|_V, \ldots, \ell_t|_V$. Hence, we can assume that we have $\ell_1|_V, \ldots, \ell_t|_V$. We can now factor each of the circuits $M_2|_{V,\ell_1|_V=0}, \ldots, M_2|_{V,\ell_t|_V=0}$. At this point a new problem arises: how do we "glue" those gates together to get $M_2|_V$. Let us look more closely at this question. By applying an invertible linear transformation, we can assume without loss of generality that $\ell_i = x_i$ for every i. What we actually have is $M_2|_V$, except that for each $i \in [t]$ someone reveals all its linear factors but erases x_i from it. The consequence of setting $x_i = 0$ is that different linear functions can become identical after this setting. It is not clear how to combine the different factors of the circuits $M_2|_{V,x_i=0}$ together. Shpilka [125] overcomes this by showing that (for this choice of t) there must exist x_i and x_j and a linear factor L of $M_2|_V$ such that the multiplicity of L in $M_2|_V$ is equal to its multiplicity in $M_2|_{V,x_i=0}$ and to its multiplicity in $M_2|_{V,x_j=0}$. This means that by inspecting the factors of $M_2|_{V,x_i=0}$ and $M_2|_{V,x_j=0}$ we can find this function L (or some other function with the same property) as follows. By considering the restriction $L|_{x_i=0}$ we can recover the coefficients of x_j in L, and similarly by considering $L|_{x_j=0}$ we can recover the coefficients of x_i (this does not work when L depends only on x_i and x_j; this "bad" event happens with very small probability over the choice of V). Now we can remove L from $M_2|_V$ and recursively learn $M_2|_V/L$. The proof of the existence of such a function L is very similar to the proof of Theorem 4.22 so we skip it. Now, given $M_2|_V$ we can find $M_1|_V$ by factoring the polynomial $\Phi|_V - M_2|_V$.

At the end of the above argument we have the circuit $\Phi|_V$. The problem is of course finding the circuit Φ itself. Here we use the

[6] By "guess" we mean that we go over all possibilities, for each one we run the reconstruction algorithm, and at the end check which of the solutions that we found is the correct one using a PIT algorithm.

structural theorem (Theorem 4.22) to claim that the circuit that we found is unique. Namely, if $\Psi = N_1 + N_2$ is another $\Sigma\Pi\Sigma(2)$ circuit computing $\Phi|_V$ then either $N_1 = M_1|_V$ or $N_2 = M_1|_V$. This follows from the observation that $M_1|_V + M_2|_V - N_1 - N_2 \equiv 0$ and the fact that $\mathrm{rank}(\Phi|_V) \geq R(4,r)$. Knowing that the circuit is unique, we can "lift" it to $\mathbb{F}^n$ as follows. Repeat the learning algorithm and learn each of the circuits $\Phi|_{V_i}$, where $V_i = \mathrm{span}(V \cup \{e_i\})$ and e_i is the i-th basis vector. Denote $\Phi|_{V_i} = M_1^i + M_2^i$. The uniqueness of $\Phi|_V$ guarantees that we can simply "glue" together the linear functions in the different M_1^i's whose restriction gives the same linear function in $M_1|_V$ (as we picked V at random, different linear functions in Φ will be mapped to "far away" functions in $\Phi|_V$). This concludes the algorithm for the case that the rank of Φ is high. In this case the algorithm actually returns a $\Sigma\Pi\Sigma(2)$ circuit.

We now turn to the low rank case that has a different algorithm, but of roughly the same structure. As before we restrict the circuit to a random low dimensional space. Then we find a representation of $\Phi|_V$ as a polynomial in a small set of linear functions. This can be done, as the linear functions in the circuit have low rank. Assume that we found a representation $\Phi|_V = f(\ell_1,\ldots,\ell_\rho)$. What we do next is find a representation of the form $\Phi|_{V_i} = f(\ell_1^i,\ldots,\ell_\rho^i)$, where $\ell_j^i|_V = \ell_j$, and the V_i's are defined as before. The requirement $\ell_j^i|_V = \ell_j$ guarantees that the ℓ_j^i's are "consistent" with each other. It is not hard to prove that such a representation exists (given that $\rho = \mathrm{rank}(\Phi)$). Once we have the ℓ_j^i's, we can again combine them together and obtain a representation $\Phi = f(L_1,\ldots,L_\rho)$ for some linear functions. This is a very rough sketch and the interested reader is referred to [125] for more details. In the low rank case the algorithm does not return a $\Sigma\Pi\Sigma(2)$ circuit.

In the case of multilinear $\Sigma\Pi\Sigma(2)$ circuits, [125] gives a polynomial-time algorithm. The main difference is that the "critical" rank is now a constant and not poly-logarithmic. A delicate point is that the space V cannot be random as the restriction of a multilinear polynomial to a random subspace is usually not multilinear. Nevertheless, one can show that instead of choosing a random V it is enough to set variables to constants at random (leaving a constant number of variables "alive").

As the dimension of the resulting subspace is constant, the algorithm runs in polynomial time. It is also interesting that in this case the algorithm actually returns a multilinear $\Sigma\Pi\Sigma(k)$ circuit.

5.4.2 The Case of General k

We now explain the idea behind the algorithm of [77] for reconstructing $\Sigma\Pi\Sigma(k)$ circuits. Before we describe their algorithm let us explain the difficulties in extending the algorithm above for the case $k > 2$. The algorithm for $\Sigma\Pi\Sigma(2)$ circuits has the following three steps. (a) Restrict the circuit to a low dimensional space. (b) Either learn one multiplication gate at a time in the case of high rank, or find a representation of the circuit as a function in a small number of linear forms in the case of low rank. (c) "Lift" the circuit from the subspace to the whole space. Several problems occur when trying to make this approach work when we have many product gates. First of all, it may be the case that the circuit is of high rank, yet some of its multiplication gates share many common linear functions. For example, this can happen if the circuit is a sum of a high rank circuit with a low rank circuit. In this case it is not clear how to use the previous algorithm. Let us ignore this problem for now and consider the next difficulty. Previously, in the high rank case we learnt each product gate separately, by restricting the circuit to a subspace on which one of the gates vanishes. It is not clear that we can do the same when we have more than two gates. Finally, even if we do learn $\Phi|_V$, it is not obvious that we can lift it as it involves both "high rank" parts and "low rank" parts. Specifically, the representation of $\Phi|_V$ is not unique and therefore it is not clear that the circuits $\Phi|_{V_i}$ can be "glued" together (this is closely related to the first problem that we discussed).

We now explain how Karnin and Shpilka [77] handle these problems. The main idea is to define a more robust notion of a product gate. They define a distance function between gates $\Delta(M_1, M_2) \overset{\text{def}}{=} \text{rank}((M_1 + M_2)/\gcd(M_1, M_2))$. In words, the distance between two gates is the rank of their sum after we remove their common linear factors. As it turns out, Δ obeys the triangle inequality and so can be viewed as giving a metric for product gates. Furthermore, it also make sense to define (in the same manner) the function $\Delta(M_1,\dots,M_t)$. They use this metric to cluster product gates in a way that any two clusters are far. That is, they first partition $[k]$ to t sets $I_1,\dots,I_t$ and then define $\Phi_j = \sum_{i\in I_j} M_i$. Clearly, $\Phi = \sum_{j\in[t]} \Phi_j$. The important property that the Φ_j's satisfy is that $\mathrm{rank}(\Phi_j)$ is small but for any $j_1 \neq j_2$, $i_1 \in I_{j_1}$ and $i_2 \in I_{j_2}$ it holds that $\Delta(M_{i_1}, M_{i_2})$ is large. In words, the rank of the sum of the gates in each of the partitions is small (after removing their greatest common divisor) and any two clusters are far from each other. An important conclusion of this definition is that it guarantees uniqueness. Indeed, if $\Phi = \sum_{j\in J} \Psi_j$ is another representation of Φ as a sum of at most k far clusters, then it must be the case that $|J| = t$ and after permuting the indices, $\Phi_j = \Psi_j$. We call each Φ_j a generalized multiplication gate (with this definition each low rank circuit is a single generalized product gate). Another implication is that for a random subspace V (of polylog dimension), with high probability $\Phi|_V = \sum_{j\in[t]} \Phi_j|_V$ is the *unique* representation of $\Phi|_V$ as a sum of generalized multiplication gates. This property is very important because if we can learn $\Phi|_V$ then using the uniqueness we can hope to lift each Φ_j separately to $\mathbb{F}^n$ as was done in the case $k = 2$.

The second problem that we need to deal with is how to learn $\Phi|_V$. For this, Karnin and Shpilka [77] show that if we pick a subspace V of a high enough dimension ($\mathrm{polylog}(n)$ suffices) at random then there exists a *reconstruction tree* of depth at most k for $\Phi|_V$. A reconstruction tree is a tree whose nodes are labeled with subspaces of V having the following four properties. (1) The root is labeled with V. (2) If U is a child of W then U is a subspace of W of co-dimension one. (3) The subspaces in the children of each node span the subspace in the node. (4) There exists a generalized product gate Φ_j of Φ such that for each leaf U of the tree $\Phi_j|_U \not\equiv 0$, but all the other generalized product gates vanish on U. Now, if $\{U_i\}$ are the children of W then from $\{\Phi_j|_{U_i}\}$ we can reconstruct $\Phi_j|_W$, by going bottom up. Indeed, if U labels a leaf then we can reconstruct $\Phi_j|_U$ by simple factoring, as all the other gates vanish on U. Then, we reconstruct $\Phi_j|_U$ for all subspaces U that label nodes just above the leaves (using similar ideas to the case $k = 2$), and so forth. Eventually, we can reconstruct $\Phi_j|_V$. Then, we consider $\Phi|_V - \Phi_j|_V$ and repeat the same argument until we reconstruct the other generalized multiplication gates. As in the case $k = 2$, we find the adequate reconstruction trees by simply trying out all possibilities. As the dimension of V is poly-logarithmic and k is a constant going over all trees takes quasi-polynomial time.

Once we find $\Phi|_V$, using the fact that the representation is unique, we can apply the same ideas as in the case $k = 2$ to lift the representation to $\mathbb{F}^n$. It is not hard to show that this algorithm outputs a representation of Φ as a sum of at most k generalized multiplication gates.

We shall not elaborate more on this algorithm but rather explain why this can be viewed as a generalization of the algorithm for the case $k = 2$. When a $\Sigma\Pi\Sigma(2)$ circuit has a high rank then we view it as a sum of two clusters (each gate is clustered only with itself). On the other hand, in the low rank case we view the circuit as a single generalized multiplication gate and then learn this gate. Thus, the algorithm for reconstructing $\Sigma\Pi\Sigma(2)$ circuits either returns two clusters in the case of high rank or one cluster in the case of low rank.

In the case of multilinear $\Sigma\Pi\Sigma(k)$ circuits, similar arguments are used and again Karnin and Shpilka [77] manage to reconstruct such circuits in polynomial time (with k appearing in the exponent of course).

5.5 Concluding Remarks

In this chapter we discussed the reconstruction problem of arithmetic circuits. We saw some hardness results and described a few algorithms. We find the state of the art, however, to be not satisfactory in both directions. Specifically, we expect stronger hardness results to hold. E.g., that reconstruction implies lower bounds for more natural polynomials than those guaranteed by Theorem 5.2 (see, e.g., Open Problem 32). The situation with reconstruction algorithms is even more depressing. We believe that one should be able to reconstruct circuits in classes for which PIT algorithms are known. The problem of reconstructing sums of read-once formulas, Open Problem 31, is an interesting question in this direction. The special case of reconstructing read-once formulas is known but was not covered in this survey (the interested reader is referred to [26, 27, 28, 55, 126, 127]).

Acknowledgments

We thank Michael Forbes for his helpful comments that improved the presentation and Madhu Sudan for encouraging us to write this monograph.

References

[1] S. Aaronson, "Arithmetic natural proofs theory is sought," http://scottaaronson.com/blog/?p=336, 2008.

[2] M. Agrawal, "Proving lower bounds via pseudo-random generators," in *Proceedings of the 25th FSTTCS*, vol. 3821 of *LNCS*, pp. 92–105, 2005.

[3] M. Agrawal and S. Biswas, "Primality and identity testing via Chinese remaindering," *Journal of the ACM*, vol. 50, pp. 429–443, 2003.

[4] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in P," *Annals of Mathematics*, vol. 160, pp. 781–793, 2004.

[5] M. Agrawal and V. Vinay, "Arithmetic circuits: A chasm at depth four," in *Proceedings of the 49th Annual FOCS*, pp. 67–75, 2008.

[6] N. Alon, "Combinatorial nullstellensatz," *Combinatorics, Probability and Computing*, vol. 8, pp. 7–29, 1999.

[7] S. A. Amitsur and J. Levitzki, "Minimal identities for algebras," *Proceedings of the American Mathematical Society*, vol. 1, pp. 449–463, 1950.

[8] M. Anderson, D. van Melkebeek, and I. Volkovich, "Derandomizing polynomial identity testing for multilinear constant-read formulae," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 135, 2010. http://www.eccc.uni-trier.de/report/2010/189/.

[9] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[10] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *Journal of the ACM*, vol. 45, pp. 501–555, 1998.

[11] S. Arora and S. Safra, "Probabilistic checking of proofs: A new characterization of NP," *Journal of the ACM*, vol. 45, pp. 70–122, 1998.

[12] V. Arvind and P. Mukhopadhyay, "The monomial ideal membership problem and polynomial identity testing," in *Proceedings of the 18th ISAAC*, pp. 800–811, 2007.

[13] V. Arvind and P. Mukhopadhyay, "Derandomizing the isolation lemma and lower bounds for circuit size," in *APPROX-RANDOM*, pp. 276–289, 2008.

[14] V. Arvind, P. Mukhopadhyay, and S. Srinivasan, "New results on noncommutative and commutative polynomial identity testing," in *Proceedings of the 23rd Annual CCC*, pp. 268–279, 2008.

[15] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity*, vol. 1, pp. 3–40, 1991.

[16] W. Baur and V. Strassen, "The complexity of partial derivatives," *Theoretical Compter Science*, vol. 22, pp. 317–330, 1983.

[17] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio, "Learning functions represented as multiplicity automata," *Journal of the ACM*, vol. 47, pp. 506–530, 2000.

[18] M. Ben-Or and P. Tiwari, "A deterministic algorithm for sparse multivariate polynominal interpolation," in *Proceedings of the 20th Annual STOC*, pp. 301–309, 1988.

[19] M. Bläser, "A $5/2n^2$-lower bound for the rank of $n \times n$-matrix multiplication over arbitrary fields," in *Proceedings of the 40th Annual FOCS*, pp. 45–50, 1999.

[20] M. Bläser, M. Hardt, R. J. Lipton, and N. K. Vishnoi, "Deterministically testing sparse polynomial identities of unbounded degree," *Information Processing Letters*, vol. 109, pp. 187–192, 2009.

[21] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation.* Springer, 1997.

[22] M. Blum, A. K. Chandra, and M. N. Wegman, "Equivalence of free boolean graphs can be tested in polynomial time," *Information Processing Letters*, vol. 10, pp. 80–82, 1980.

[23] A. Bogdanov and H. Wee, "More on noncommutative polynomial identity testing," in *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pp. 92–99, 2005.

[24] A. Borodin, J. von zur Gathen, and J. E. Hopcroft, "Fast parallel matrix and GCD computations," *Information and Control*, vol. 52, pp. 241–256, 1982.

[25] M. R. Brown and D. P. Dobkin, "An improved lower bound on polynomial multiplication," *IEEE Transactions on Computers*, vol. 29, pp. 337–340, 1980.

[26] D. Bshouty and N. H. Bshouty, "On interpolating arithmetic read-once formulas with exponentiation," *Journal of Computer and System Sciences*, vol. 56, pp. 112–124, 1998.

[27] N. H. Bshouty and R. Cleve, "Interpolating arithmetic read-once formulas in parallel," *SIAM Journal on Computing*, vol. 27, pp. 401–413, 1998.

[28] N. H. Bshouty, T. R. Hancock, and L. Hellerstein, "Learning arithmetic read-once formulas," *SIAM Journal on Computing*, vol. 24, pp. 706–735, 1995.

[29] P. Bürgisser, "On the structure of valiant's complexity classes," *Discrete Mathematics & Theoretical Computer Science*, vol. 3, pp. 73–94, 1999.

[30] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*. Springer, 1997.

[31] J. Cai, X. Chen, and D. Li, "A quadratic lower bound for the permanent and determinant problem over any characteristic != 2," in *Proceedings of the 40th Annual STOC*, pp. 491–498, 2008.

[32] S. Chari, P. Rohatgi, and A. Srinivasan, "Randomness-optimal unique element isolation with applications to perfect matching and related problems," *SIAM Journal on Computing*, vol. 24, pp. 1036–1050, 1995.

[33] B. Chazelle, "A spectral approach to lower bounds with applications to geometric searching," *SIAM Journal on Computing*, vol. 27, pp. 545–556, 1998.

[34] Z. Chen and M. Kao, "Reducing randomness via irrational numbers," *SIAM Journal on Computing*, vol. 29, pp. 1247–1256, 2000.

[35] S. Chien and A. Sinclair, "Algebras with polynomial identities and computing the determinant," *SIAM Journal on Computing*, vol. 37, pp. 252–266, 2007.

[36] H. Cohn, R. D. Kleinberg, B. Szegedy, and C. Umans, "Group-theoretic algorithms for matrix multiplication," in *Proceedings of the 46th Annual FOCS*, pp. 379–388, 2005.

[37] H. Cohn and C. Umans, "A group-theoretic approach to fast matrix multiplication," in *Proceedings of the 44th Annual FOCS*, pp. 438–449, 2003.

[38] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[39] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progression," *Journal of Symbolic Computation*, vol. 9, pp. 251–280, 1990.

[40] L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM Journal on Computing*, vol. 5, pp. 618–623, 1976.

[41] R. A. DeMillo and R. J. Lipton, "A probabilistic remark on algebraic program testing," *Information Processing Letters*, vol. 7, pp. 193–195, 1978.

[42] Z. Dvir, "On matrix rigidity and locally self-correctable codes," in *Proceedings of the 25th Annual CCC*, pp. 291–298, 2010.

[43] Z. Dvir and A. Shpilka, "Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits," *SIAM Journal on Computing*, vol. 36, pp. 1404–1434, 2006.

[44] Z. Dvir, A. Shpilka, and A. Yehudayoff, "Hardness-randomness tradeoffs for bounded depth arithmetic circuits," *SIAM Journal on Computing*, vol. 39, pp. 1279–1293, 2009.

[45] M. Edelstein and L. M. Kelly, "Bisecants of finite collections of sets in linear spaces," *Canadanian Journal of Mathematics*, vol. 18, pp. 375–280, 1966.

[46] L. Fortnow and A. R. Klivans, "Efficient learning algorithms yield circuit lower bounds," *Journal of Computer System Science*, vol. 75, pp. 27–36, 2009.

[47] A. Gabizon and R. Raz, "Deterministic extractors for affine sources over large fields," *Combinatorica*, vol. 28, pp. 415–440, 2008.

[48] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, pp. 792–807, 1986.

[49] D. Grigoriev and M. Karpinski, "The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract)," in *Proceedings of the 28th Annual FOCS*, pp. 166–172, 1987.

[50] D. Grigoriev and M. Karpinski, “An exponential lower bound for depth 3 arithmetic circuits,” in *Proceedings of the 30th Annual STOC*, pp. 577–582, 1998.

[51] D. Grigoriev, M. Karpinski, and M. F. Singer, “Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields,” *SIAM Journal on Computing*, vol. 19, pp. 1059–1063, 1990.

[52] D. Grigoriev and A. A. Razborov, “Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields,” *Applicable Algebra in Engineering, Communication and Computing*, vol. 10, pp. 465–487, 2000.

[53] J. Håstad, “Almost optimal lower bounds for small depth circuits,” in *Proceedings of the 18th Annual STOC*, pp. 6–20, 1986.

[54] J. Håstad, “Tensor rank is np-complete,” *Journal of Algorithms*, vol. 11, pp. 644–654, 1990.

[55] T. R. Hancock and L. Hellerstein, “Learning read-once formulas over fields and extended bases,” in *Proceedings of the 4th Annual COLT*, pp. 326–336, 1991.

[56] J. Heintz and C. P. Schnorr, “Testing polynomials which are easy to compute (extended abstract),” in *Proceedings of the 12th annual STOC*, pp. 262–272, 1980.

[57] J. Heintz and M. Sieveking, “Lower bounds for polynomials with algebraic coefficients,” *Theoretical Computer Science*, vol. 11, pp. 321–330, 1980.

[58] S. Hoory, N. Linial, and A. Wigderson, “Expander graphs and their applications,” *Bulletin of the American Mathematical Society*, vol. 43, pp. 439–561, 2006.

[59] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Pearson Education, 2nd ed., 2000.

[60] P. Hrubeš, A. Wigderson, and A. Yehudayoff, “Non-commutative circuits and the sum-of-squares problem,” in *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 667–676, 2010.

[61] P. Hrubeš, A. Wigderson, and A. Yehudayoff, “Relationless completeness and separations,” in *Proceedings of the 25th Conference on Computational Complexity*, pp. 280–290, 2010.

[62] P. Hrubeš and A. Yehudayoff, “Arithmetic complexity in algebraic extensions,” *Manuscript*, 2009.

[63] P. Hrubeš and A. Yehudayoff, “Homogeneous formulas and symmetric polynomials,” *CoRR*, abs/0907.2621, 2009.

[64] P. Hrubeš and A. Yehudayoff, “Monotone separations for constant degree polynomials,” *Information Processing Letters*, vol. 110, pp. 1–3, 2009.

[65] R. Impagliazzo, V. Kabanets, and A. Wigderson, “In search of an easy witness: Exponential time vs. probabilistic polynomial time,” *Journal of Computer and System Sciences*, vol. 65, pp. 672–694, 2002.

[66] R. Impagliazzo and A. Wigderson, “P=BPP unless E has subexponential circuits: derandomizing the XOR lemma,” in *Proceedings of the 29th STOC*, pp. 220–229, 1997.

[67] M. Jansen, Y. Qiao, and J. Sarma, “Deterministic identity testing of read-once algebraic branching programs,” *CoRR*, abs/0912.2565, 2009.

[68] M. Jerrum and M. Snir, “Some exact complexity results for straight-line computations over semi-rings,” Technical Report CRS-58–80, University of Edinburgh, 1980.

[69] V. Kabanets and R. Impagliazzo, “Derandomizing polynomial identity tests means proving circuit lower bounds,” *Computational Complexity*, vol. 13, pp. 1–46, 2004.

[70] K. Kalorkoti, “A lower bound for the formula size of rational functions,” *SIAM Journal of Computing*, vol. 14, pp. 678–687, 1985.

[71] E. Kaltofen, “Factorization of polynomials given by straight-line programs,” in *Randomness in Computation*, vol. 5 of *Advances in Computing Research*, (S. Micali, ed.), pp. 375–412, 1989.

[72] E. Kaltofen, “Polynomial factorization: A success story,” in *ISSAC*, pp. 3–4, 2003.

[73] E. Kaltofen and B. M. Trager, “Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators,” *Journal of Symbolic Computation*, vol. 9, pp. 301–320, 1990.

[74] M. Kaminski, “A lower bound on the complexity of polynomial multiplication over finite fields,” *SIAM Journal on Computing*, vol. 34, pp. 960–992, 2005.

[75] Z. S. Karnin, P. Mukhopadhyay, A. Shpilka, and I. Volkovich, “Deterministic identity testing of depth 4 multilinear circuits with bounded top fan-in,” in *Proceedings of the 42nd Annual STOC*, pp. 649–658, 2010.

[76] Z. S. Karnin and A. Shpilka, “Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in,” in *Proceedings of the 23rd Annual CCC*, pp. 280–291, 2008.

[77] Z. S. Karnin and A. Shpilka, “Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in,” in *Proceedings of the 24th Annual CCC*, pp. 274–285, 2009.

[78] R. Karp, E. Upfal, and A. Wigderson, “Constructing a perfect matching is in random NC,” *Combinatorica*, vol. 6, pp. 35–48, 1, 1986.

[79] N. Kayal, “Derandomizing some number-theoretic and algebraic algorithms,” PhD thesis, Indian Institute of Technology, Kanpur, India, 2007.

[80] N. Kayal and S. Saraf, “Blackbox polynomial identity testing for depth 3 circuits,” in *Proceedings of the 50th Annual FOCS*, pp. 198–207, 2009.

[81] N. Kayal and N. Saxena, “Polynomial identity testing for depth 3 circuits,” *Computational Complexity*, vol. 16, pp. 115–138, 2007.

[82] M. J. Kearns and L. G. Valiant, “Cryptographic limitations on learning boolean formulae and finite automata,” *Journal of the ACM*, vol. 41, pp. 67–95, 1994.

[83] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory.* Cambridge, MA, USA: MIT Press, 1994.

[84] K. S. Kedlaya and C. Umans, “Fast modular composition in any characteristic,” in *Proceedings of the 49th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 146–155, 2008.

[85] A. Klivans and A. Shpilka, “Learning restricted models of arithmetic circuits,” *Theory of Computing*, vol. 2, pp. 185–206, 2006.

[86] A. Klivans and D. Spielman, “Randomness efficient identity testing of multivariate polynomials,” in *Proceedings of the 33rd Annual STOC*, pp. 216–223, 2001.

[87] A. R. Klivans and A. A. Sherstov, “Cryptographic hardness for learning intersections of halfspaces,” *Journal of Computer and System Sciences*, vol. 75, pp. 2–12, 2009.

[88] P. Koiran, “Arithmetic circuits: The chasm at depth four gets wider,” *CoRR*, abs/1006.4700, 2010.

[89] D. Lewin and S. Vadhan, “Checking polynomial identities over any field: Towards a derandomization?,” in *Proceedings of the 30th Annual STOC*, pp. 428–437, 1998.

[90] L. Lovasz, “On determinants, matchings, and random algorithms,” in *Fundamentals of Computing Theory*, (L. Budach, ed.), Akademia-Verlag, 1979.

[91] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, “Algebraic methods for interactive proof systems,” *Journal of the ACM*, vol. 39, pp. 859–868, 1992.

[92] T. Mignon and N. Ressayre, “A quadratic bound for the determinant and permanent problem,” *International Mathematics Research Notices*, vol. 79, pp. 4241–4253, 2004.

[93] P. Morandi, *Graduate Texts in Mathematics 167: Field and Galois Theory.* Springer-Verlag, New York, 1996.

[94] J. Morgenstern, “Note on a lower bound on the linear complexity of the fast Fourier transform,” *Journal of the ACM*, vol. 20, pp. 305–306, 1973.

[95] K. Mulmuley and M. A. Sohoni, “Geometric complexity theory i: An approach to the P vs. NP and related problems,” *SIAM Journal on Computing*, vol. 31, pp. 496–526, 2001.

[96] K. Mulmuley and M. A. Sohoni, “Geometric complexity theory ii: Towards explicit obstructions for embeddings among class varieties,” *SIAM Journal on Computing*, vol. 38, pp. 1175–1206, 2008.

[97] K. Mulmuley, U. Vazirani, and V. Vazirani, “Matching is as easy as matrix inversion,” *Combinatorica*, vol. 7, pp. 105–113, 1987.

[98] N. Nisan, “Lower bounds for non-commutative computation,” in *Proceedings of the 23rd Annual STOC*, pp. 410–418, 1991.

[99] N. Nisan and A. Wigderson, “Hardness vs. randomness,” *Journal of Computer System Sciences*, vol. 49, pp. 149–167, 1994.

[100] N. Nisan and A. Wigderson, “Lower bound on arithmetic circuits via partial derivatives,” *Computational Complexity*, vol. 6, pp. 217–234, 1996.

[101] R. Raz, “On the complexity of matrix product,” *SIAM Journal on Computing*, vol. 32, pp. 1356–1369, 2003.

[102] R. Raz, “Separation of multilinear circuit and formula size,” *Theory of Computing*, vol. 2, pp. 121–135, 2006.

[103] R. Raz, “Elusive functions and lower bounds for arithmetic circuits,” in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 711–720, 2008.

[104] R. Raz, "Multi-linear formulas for permanent and determinant are of super-polynomial size," *Journal of the ACM*, vol. 56, 2009.

[105] R. Raz, "Tensor-rank and lower bounds for arithmetic formulas," in *Proceedings of the 42nd Annual STOC*, pp. 659–666, 2010.

[106] R. Raz and A. Shpilka, "Deterministic polynomial identity testing in non commutative models," *Computational Complexity*, vol. 14, pp. 1–19, 2005.

[107] R. Raz, A. Shpilka, and A. Yehudayoff, "A lower bound for the size of syntactically multilinear arithmetic circuits," *SIAM Journal on Computing*, vol. 38, pp. 1624–1647, 2008.

[108] R. Raz and A. Yehudayoff, "Balancing syntactically multilinear arithmetic circuits," *Computational Complexity*, vol. 17, pp. 515–535, 2008.

[109] R. Raz and A. Yehudayoff, "Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors," in *Proceedings of the 49th Annual FOCS*, pp. 273–282, 2008.

[110] R. Raz and A. Yehudayoff, "Lower bounds and separations for constant depth multilinear circuits," *Computational Complexity*, vol. 18, pp. 171–207, 2009.

[111] A. A. Razboeov and S. Rudich, "Natural proofs," *Journal of Computer and System Sciences*, vol. 55, pp. 24–35, 1997.

[112] A. A. Razborov, "Lower bounds for the size of circuits with bounded depth with basis $\{\wedge, \oplus\}$," *Matematicheskie Zametki*, pp. 598–607, 1987. in Russian.

[113] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, 2009.

[114] H. J. Ryser, *Combinatorial Mathematics*, vol. 14. Carus Mathematical Monographs, 1963.

[115] S. Saraf and I. Volkovich, "Black-box identity testing of depth-4 multilinear circuits," *Manuscript*, 2010.

[116] N. Saxena, "Diagonal circuit identity testing and lower bounds," in *ICALP (1)*, pp. 60–71, 2008.

[117] N. Saxena and C. Seshadhri, "An almost optimal rank bound for depth-3 identities," in *Proceedings of the 24th Annual CCC*, pp. 137–148, 2009.

[118] N. Saxena and C. Seshadhri, "From Sylvester-Gallai configurations to rank bounds: Improved black-box identity test for deph-3 circuits," in *Proceedings of the 51st Annual FOCS*, pp. 21–30, 2010.

[119] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM*, vol. 27, pp. 701–717, 1980.

[120] A. Shamir, "IP = PSPACE," *Journal of the ACM*, vol. 39, pp. 869–877, 1992.

[121] E. Shamir and M. Snir, "Lower bounds on the number of multiplications and the number of additions in monotone computations," Technical Report RC-6757, IBM, 1977.

[122] V. Shoup and R. Smolensky, "Lower bounds for polynomial evaluation and interpolation problems," in *SFCS '91: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pp. 378–383, Washington, DC, USA: IEEE Computer Society, 1991.

[123] A. Shpilka, "Affine projections of symmetric polynomials," *Journal of Computer and System Sciences*, vol. 65, pp. 639–659, 2002.

[124] A. Shpilka, "Lower bounds for matrix product," *SIAM Journal on Computing*, vol. 32, pp. 1185–1200, 2003.

[125] A. Shpilka, "Interpolation of depth-3 arithmetic circuits with two multiplication gates," *SIAM Journal on Computing*, vol. 38, pp. 2130–2161, 2009.

[126] A. Shpilka and I. Volkovich, "Read-once polynomial identity testing," in *Proceedings of the 40th Annual STOC*, pp. 507–516, 2008.

[127] A. Shpilka and I. Volkovich, "Improved polynomial identity testing for read-once formulas," in *APPROX-RANDOM*, pp. 700–713, 2009.

[128] A. Shpilka and I. Volkovich, "On the relation between polynomial identity testing and finding variable disjoint factors," in *ICALP (1)*, pp. 408–419, 2010.

[129] A. Shpilka and A. Wigderson, "Depth-3 arithmetic circuits over fields of characteristic zero," *Computational Complexity*, vol. 10, pp. 1–27, 2001.

[130] R. Smolensky, "Algebraic methods in the theory of lower bounds for Boolean circuit complexity," in *Proceedings of the 19th Annual STOC*, pp. 77–82, 1987.

[131] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, pp. 354–356, 1969.

[132] V. Strassen, "Die berechnungskomplexiät von elementarsymmetrischen funktionen und von interpolationskoeffizienten," *Numerische Mathematik*, vol. 20, pp. 238–251, 1973.

[133] V. Strassen, "Vermeidung von divisionen," *The Journal für die Reine und Angewandte Mathematik*, vol. 264, pp. 182–202, 1973.

[134] V. Strassen, "Algebraic complexity theory," in *Handbook of Theoretical Computer Science*, vol. A: *Algorithms and Complexity (A)*, pp. 633–672, Elsevier and MIT Press, 1990.

[135] P. Tiwari and M. Tompa, "A direct version of Shamir and Snir's lower bounds on monotone circuit depth," *Information Processing Letters*, vol. 49, pp. 243–248, 1994.

[136] S. Toda, "PP is as hard as the polynomial time hierarchy," *SIAM Journal on Computing*, vol. 20, pp. 865–877, 1991.

[137] L. G. Valiant, "Graph-theoretic arguments in low-level complexity," in *Lecture notes in Computer Science*, vol. 53, pp. 162–176, Springer, 1977.

[138] L. G. Valiant, "Completeness classes in algebra," in *Proceedings of the 11th Annual STOC*, pp. 249–261, 1979.

[139] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, pp. 189–201, 1979.

[140] L. G. Valiant, "Negation can be exponentially powerful," *Theoretical Computer Science*, vol. 12, pp. 303–314, November 1980.

[141] L. G. Valiant, "Reducibility by algebraic projections," *L'Enseignement Mathematique*, vol. 28, pp. 253–268, 1982.

[142] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, pp. 1134–1142, 1984.

[143] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, "Fast parallel computation of polynomials using few processors," *SIAM Journal on Computing*, vol. 12, pp. 641–644, November 1983.

[144] E. Viola, "The sum of small-bias generators fools polynomials of degree," *Computational Complexity*, vol. 18, pp. 209–217, 2009.
[145] J. von zur Gathen, "Feasible arithmetic computations: Valiant's hypothesis," *Journal of Symbolic Computation*, vol. 4, pp. 137–172, 1987.
[146] J. von zur Gathen, "Who was who in polynomial factorization," in *ISSAC*, p. 2, 2006.
[147] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*. Cambridge University Press, 1999.
[148] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Symbolic and Algebraic Computation*, pp. 216–226, 1979.