Classifying polynomials and identity testing

MANINDRA AGRAWAL[1,*] and RAMPRASAD SAPTHARISHI[2]

[1] *Indian Institute of Technology, Kanpur*
[2] *Chennai Mathematical Institute, and Indian Institute of Technology, Kanpur*
e-mail:

One of the fundamental problems of computational algebra is to classify polynomials according to the hardness of computing them. Recently, this problem has been related to another important problem: *Polynomial identity testing*. Informally, the problem is to decide if a certain succinct representation of a polynomial is zero or not. This problem has been extensively studied owing to its connections with various areas in theoretical computer science.

Several efficient randomized algorithms have been proposed for the identity testing problem over the last few decades but an efficient deterministic algorithm is yet to be discovered. It is known that such an algorithm will imply hardness of computing an explicit polynomial. In the last few years, progress has been made in designing deterministic algorithms for restricted circuits, and also in understanding why the problem is hard even for small depth.

In this article, we survey important results for the polynomial identity testing problem and its connection with classification of polynomials.

1. Introduction

The interplay between mathematics and computer science demands algorithmic approaches to various algebraic constructions. The area of computational algebra addresses precisely this. The most fundamental objects in algebra are polynomials and it is a natural idea to classify the polynomials according to their 'simplicity'. The algorithmic approach suggests that the simple polynomials are those that can be computed easily. The ease of computation is measured in terms of the number of arithmetic operations required to compute the polynomial. This yields a very robust definition of simple (and hard) polynomials that can be studied analytically.

It is worth remarking that the number of terms in a polynomial is not a good measure of its simplicity. For example, consider the polynomials

$$(1 + x_1)(1 + x_2) \cdots (1 + x_n)$$

and

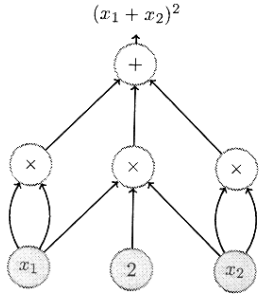
$$x_1 x_2 \ldots x_n.$$

The former has $2^n - 1$ more terms than the later, however, both are almost equally easy to describe as well as compute.

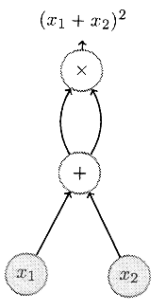
We use *arithmetic circuits* (formally defined in Section 1.1) to represent the computation of a polynomial. This also allows us to count the number of operations required in computation. Given below are a few example:

*Research supported by J C Bose Fellowship FLW/DST/CS/20060225 and IBM Fellowship DON/IBM/CSE/20080179

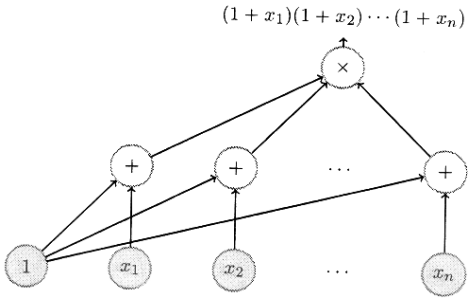
Keywords.



Example 1



Example 2



Example 3

Note that the first two circuits above, although different, compute the same polynomial. The *complexity* of a polynomial is defined as the *size* (the number of operations) of the smallest arithmetic circuit that computes the polynomial.

Using this definition, we can now classify polynomials according to their complexity. What would be the class of 'simple' polynomials? For this, we need to define the intuitive notion of 'polynomials that can be computed easily'. Following standard ideas from computer science, we call any polynomial over n variables that can be computed using at most $n^{O(1)}$ operations an easy polynomial (see next section for the explanation of $O(\cdot)$ notation). Strictly speaking, this definition is valid only for an infinite family of polynomials that contains one polynomial over n variables for each $n > 0$. Any single polynomial can always be computed using $O(1)$ operations rendering the whole exercise meaningless. However, often we will omit to explicitly mention the infinite family to which a polynomial belongs when talking about its complexity; the family would be obvious.

The class VP is the class of polynomial families that are easy in the above sense. The polynomials in VP are essentially represented by the *determinant* polynomial: the determinant of an $n \times n$ matrix whose entries are affine linear combinations of variables. It is known that determinant polynomial belongs to the class VP [1] and any polynomial in VP over n variables can be written as a determinant polynomial of a $m \times m$ matrix with $m = n^{O(\log n)}$ [2].

This provides a excellent classification of easy polynomials. Hence, any polynomial that cannot be written as a determinant of a small sized matrix is not easy. A simple counting argument shows that there exist many such polynomials. However, proving an explicitly given polynomial to be hard has turned out to be a challenging problem which has not been solved yet. In particular, the *permanent* polynomial, the permanent of a matrix with affine linear entries, is believed to be very hard to compute – requiring $2^{\Omega(n)}$-size circuits for an $n \times n$ matrix in general. However, there is no proof yet of this. It is not even known if it requires $\Omega(n^3)$ operations!

A general way of classifying a given polynomial is to design an algorithm that, given a polynomial in the form of one specific arithmetic circuit as input, outputs the smallest size arithmetic circuit computing the same polynomial. Such an algorithm is easy to design: given an arithmetic circuit C as input, the algorithm runs through all circuits smaller than C and checks if any of these computes the same polynomial as C (this check can be performed easily as we discuss in the next paragraph). However, this algorithm is not efficient: it will take exponential time (in the size of the input circuit C) to find the classification of a polynomial. No efficient algorithm for this is known; further, it is believed that *no efficient algorithm exists* for this problem.

A closely related problem that occurs above is to check if two given arithmetic circuits C and D compute the same polynomial. The problem is equivalent to asking if the circuit $C - D$ is the zero polynomial or not. This problem of checking if a circuit computes the zero polynomial is called *polynomial identity testing* (PIT). It turns out that this problem is easy to solve algorithmically. We give later several randomized polynomial time algorithms for solving it. Moreover, in a surprising connection, it has been found that if there is a deterministic polynomial time algorithm for solving PIT, then certain explicit polynomials are hard to compute [3,4]! Therefore, the solution to PIT problem has a key role in our attempt to computationally classify polynomials. In this article, we will focus on this connection between PIT and polynomial classification.

We now formally define arithmetic circuits and the identity testing problem.

1.1 *Problem definition*

Let $\mathbb{N}$ and $\mathbb{Q}$ denote the set of natural and rational numbers, respectively.

Definition 1.1 (Order notation). *Given two functions f and g, $f, g : \mathbb{N} \mapsto \mathbb{N}$, we write $f = O(g)$ if there exist constants $c, n_0 > 0$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$. We write $g = \Omega(f)$ if $f = O(g)$.*

Some examples are: $10 = O(1)$, $3n^2 + 7 = O(n^3)$, $n^3 = \Omega(n^2)$, $n^{22} = n^{O(1)}$, *etc.*

We shall fix an underlying field $\mathbb{F}$.

Definition 1.2 (Arithmetic circuits and formulas). *An* arithmetic circuit *is a directed acyclic graph with one sink (which is called the output gate). Each of the source vertices (which are called input nodes) are either labeled by a variable x_i or an element from an underlying field $\mathbb{F}$. Each of the internal nodes are labeled either by $+$ or $\times$ to indicate if it is an addition or multiplication gate, respectively.*

Such a circuit naturally computes a multivariate polynomial at every node. The circuit is said to compute a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ if the output node computes f.

If the underlying field $\mathbb{F} = \mathbb{Q}$, then a circuit is said to be monotone *if none of the constants are negative.*

An arithmetic circuit is a formula *if every internal node has out-degree* 1.

Without loss of generality, the circuit is assumed to be layered, with edges only between successive layers. Further, it is assumed it consists of alternating layers of addition and multiplication gates. A layer of addition gates is denoted by Σ and a layer of multiplications by Π.

Some important parameters of an arithmetic circuit are the following:

- *Size*: the number of gates in the circuit
- *Depth*: the longest path from a leaf gate to the output gate
- *Degree*: the syntactic degree of the polynomial computed at the output gate. This is computed recursively at every gate in the most natural way (max of the degrees of children at an addition gate, and the sum of the degrees at a multiplication gate).

  This needn't be the degree of the polynomial computed at the output gate (owing to cancellations) but this is certainly an upper bound.

A circuit evaluating a polynomial provides a succinct representation of the polynomial. For instance, in Example 3, though the polynomial has 2^n terms, we have a circuit size $O(n)$ computing the polynomial. The PIT problem is deciding if a given succinct representation is zero or not.

Also, a circuit of size s can potentially compute a polynomial of exponential degree. But usually in identity testing, it is assumed that the degree of the polynomial is $O(n)$, where n is the number of variables. Most interesting polynomials, like the determinant or permanent, satisfy this property.

Problem 1.3 (Polynomial identity testing). *Given an arithmetic circuit C with input variables $x_1, \ldots, x_n$ and constants taken from a field $\mathbb{F}$, check if the polynomial computed is identically zero.*

The goal is to design a deterministic algorithm for PIT that runs in time polynomial in n, size of C and $|\mathbb{F}|$. A much stronger algorithm is one that doesn't look into the structure of the circuit at all, but just evaluates it at chosen input points. Such an algorithm that just uses the circuit as a 'black box' is hence called a *black-box algorithm*.

1.2 *Current status*

A likely candidate of a *hard* polynomial is the *permanent* polynomial. It is widely believed that it requires circuits of exponential size, but this is still open. However, progress has been made in restricted settings. Raz and Yehudayoff [5] showed that monotone circuits for permanent require exponential size. Nisan and Wigderson [6] showed that 'homogeneous' depth 3 circuits for the $2d$-th symmetric polynomial requires $\left(\frac{n}{4d}\right)^{\Omega(d)}$ size. Shpilka and Wigderson [7] showed that depth 3 circuits for determinant or permanent over $\mathbb{Q}$ require quadratic size. Over finite fields, Grigoriev and Karpinsky [8] showed that determinant or permanent required exponential sized depth 3 circuit.

As for PIT, the problem has drawn significant attention due to its role in various fields of theoretical computer science. Besides being a natural problem in algebraic computation, identity testing has found applications in various fundamental results like Shamir's $\mathsf{IP} = \mathsf{PSPACE}$ [9], the PCP theorem [10], etc. Many other important results, such as the AKS primality test [11], check if some special polynomials are identically zero or not. Algorithms for graph matchings [12] and multivariate polynomial interpolation [13] also involve identity testing. Another promising role of PIT is its connection to the question of 'hardness of polynomials'. It is known that strong algorithms for PIT can be used to construct polynomials that are *very hard* [3,4].

There is a score of randomized algorithms proposed for PIT. The first randomized polynomial time algorithm for identity testing was given by Schwartz and Zippel [14,15]. Several other *randomness-efficient* algorithms [16–19] came up subsequently, resulting in a significant improvement in the number of random bits used.

However, despite numerous attempts, a deterministic polynomial time algorithm has remained unknown. Nevertheless, important progress has been made both in the designing of deterministic algorithms for special circuits, and in the understanding of why a general deterministic solution could be hard to get.

Kayal and Saxena [20] gave a deterministic polynomial time identity testing algorithm for depth 3 ($\Sigma\Pi\Sigma$) circuits with constant top fan-in (the top addition gate has only constantly many children). When the underlying field is $\mathbb{Q}$, this was further improved to a black-box algorithm by Kayal and Sharaf [21]. Saxena gave a polynomial time algorithm for a restricted form of depth 3 circuits called 'diagonal circuits'. As such, no polynomial time PIT algorithm is known for general depth 3 circuits.

Most of the progress made appears to stop at around depth 3. A 'justification' behind the hardness of PIT even for small depth circuits was provided recently by Agrawal and Vinay [22]. They showed that a deterministic polynomial time black-box identity test for depth 4 ($\Sigma\Pi\Sigma\Pi$) circuits would imply a quasi-polynomial ($O(n^{\log n})$) time deterministic PIT algorithm for any circuit computing a polynomial of *low degree*[1]. Thus, PIT for depth 4 circuits over a field is *almost* the general case.

Thus we see that the non-trivial case for identity testing starts with depth 3 circuits; whereas circuits of depth 4 are *almost* the general case. A natural first step to design an algorithm for PIT of $\Sigma\Pi\Sigma$ circuits. In this article, we survey some important results in identity testing and possible approaches to attack depth 3 circuits.

1.3 *Organization*

Section 2 discusses the connection between PIT and circuit lower bounds. Section 3 looks at PIT and lower bounds for depth 4 and why this is almost the general case. In Section 4, we look at some randomized algorithms for PIT and in Section 5 we sketch some deterministic algorithms for special cases.

2. Connecting PIT to lower bounds

The polynomial identity testing problem is very closely connected to lower bounds in the arithmetic settings. A well studied polynomial in this context is the symbolic *permanent* polynomial.

$$\mathsf{perm}_n(x_{11}, x_{12}, \ldots, x_{1n}, x_{21}, \ldots, x_{nn}) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i\sigma(i)}.$$

Though the polynomial is very close to the determinant polynomial[2], it is widely believed that they are very different in complexity. It is a long standing open problem to show that the permanent polynomial requires exponential sized arithmetic circuits.

The quest for an explicit polynomial with large circuit complexity is one for 'hardness' in terms of circuits. Polynomial identity testing on the other hand is a quest for 'easiness'. While the two seem orthogonal, they are indeed very closely related. Kabanets and Impagliazzo [3] showed that 'subexponential' algorithms for identity testing does in fact yield circuit lower bounds.

Theorem 2.1 ([3]). *If PIT can be solved in polynomial time, or even in $\bigcap_{\varepsilon>0} \mathsf{NTIME}(n^{\varepsilon})$, then one of the following statements is true:*

- $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$
- *Permanent requires super-polynomial sized arithmetic circuits.* □

As a partial converse, they also show that explicit circuit lower bounds give quasi-polynomial algorithms for PIT.

Theorem 2.2 ([3]). *Let $\{q_m\}_{m\geq 1}$ be a family of multilinear polynomials over $\mathbb{F}$ computable in exponential time and that cannot be computed by subexponential sized arithmetic circuits. Then identity testing of low degree polynomials can be solved in time $n^{O(\log n)}$.* □

Dvir, Shpilka and Yehudayoff [32] proved similar results in the constant depth domain.

Theorem 2.3 ([32]). *If PIT of depth d circuits can be solved in polynomial time, or even in $\bigcap_{\varepsilon>0} \mathsf{NTIME}(n^{\varepsilon})$, then one of the following statements is true:*

- $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$
- *Permanent requires super-polynomial sized depth d arithmetic circuits.* □

Theorem 2.4 ([32]). *Let $\{q_m\}_{m\geq 1}$ be a family of multilinear polynomials over $\mathbb{F}$ computable in exponential time and that cannot be computed by subexponential sized arithmetic circuits of depth d.*

[1] A polynomial is said to have low degree if its degree is less than the size of the circuit.

[2] In the determinant polynomial, each term in the summation has a sign associated which is determined by the permutation σ.

Then identity testing of depth $d' = d - \delta$, for some absolute constant δ, circuits computing low degree polynomials can be solved in time $n^{O(\text{polylog}\, n)}$. □

2.1 *Black-box PITs and lower bounds*

Black-box algorithms for PIT imply much stronger implications in terms of circuit lower bounds. Agrawal showed a 'hardness-randomness' dichotomy in this setting, using the following definition of *pseudorandom generators* for arithmetic circuits that captures black-box PITs.

Definition 2.5 ([4] Pseudorandom generators for arithmetic circuits). *Let $\mathbb{F}$ be a field and $\mathcal{C}$ be a class of low degree arithmetic circuits over $\mathbb{F}$. A function $f : \mathbb{N} \longrightarrow (\mathbb{F}[y])^*$ is a $s(n)$-pseudorandom generator against $\mathcal{C}$ if*

- *$f(n) = (p_1(y), p_2(y), \ldots, p_n(y))$, where each $p_i(y)$ is a univariate polynomial over $\mathbb{F}$, whose degree is bounded by $s(n)$ and computable in time polynomial in $s(n)$*
- *For any arithmetic circuit $C \in \mathcal{C}$ of size n,*

$$C(x_1, \ldots, x_n) = 0 \quad \text{if and only if}$$
$$\times\, C(p_1(y), p_2(y), \ldots, p_n(y)) = 0.$$

It is clear that given a $s(n)$-pseudorandom generator f against $\mathcal{C}$, we can solve the PIT problem for circuits in $\mathcal{C}$ in time $(s(n))^{O(1)}$ by just evaluating the univariate polynomial. A polynomial time derandomization is obtained if $s(n)$ is $n^{O(1)}$ and such generators are called *optimal pseudorandom generators*. The following lemma shows that existence of pseudorandom generators give lower bounds.

Theorem 2.6 ([4]). *Let $f : \mathbb{N} \longrightarrow (\mathbb{F}(y))^*$ be a $s(n)$-pseudorandom generator against a class $\mathcal{C}$ of arithmetic circuit computing a polynomials of degree at most n. If $n \cdot s(n) \leq 2^n$, then there is a multilinear polynomial computed in $2^{O(n)}$ time that cannot be computed by $\mathcal{C}$.* □

This, coupled with Theorem 2.2, show that explicit 'hard' polynomials and pseudorandom generators go hand in hand.

3. Chasm at depth 4

The previous section shows that strong lower bounds can be obtained by giving a suitable deterministic algorithm for PIT. However, attempts to obtain such an algorithm have failed so far. At present, we know deterministic algorithms for only restricted kind of depth three PITs (we will see this in section 5). In fact, as already observed above, direct attempts to obtain lower bounds for certain polynomials also stop at depth three. This seems to suggest that we are very far away from obtaining lower bounds for arbitrary depth circuits. However, a recent depth reduction result by Agrawal and Vinay [22] shows that the gap is not that large. Informally, their result states that exponential sized circuits do not gain anything if the depth is beyond 4. Formally, the main result can be stated as follows:

Theorem 3.1 ([22]). *If a polynomial $P(x_1, \ldots, x_n)$ of degree $d = O(n)$ can be computed by an arithmetic circuit of size $2^{o(d + d \log \frac{n}{d})}$, it can be computed by a depth 4 circuit of size $2^{o(d + d \log \frac{n}{d})}$ as well.*

It is a simple observation that any polynomial $p(x_1, \ldots, x_n)$ of degree d has at most $\binom{n+d}{d}$ monomials and hence can be trivially computed by a $\Sigma\Pi$ circuit of size $\binom{n+d}{d} = 2^{O(d + d \log \frac{n}{d})}$. Hence, the above theorem implies that if we have subexponential lower bounds for depth 4 circuits, we have subexponential lower bounds for any depth!

Corollary 3.2. *Let $p(x_1, \ldots, x_n)$ be a multivariate polynomial. Suppose there are no $2^{o(n)}$ sized depth 4 arithmetic circuits that can compute p. Then there is no $2^{o(n)}$ sized arithmetic circuit (of arithmetic depth) that can compute p.* □

These results have very strong implications on PITs for depth 4 circuits.

Proposition 3.3. *If there is a PIT algorithm for depth 4 circuit running in deterministic polynomial time, then there is a PIT algorithm for any general circuit computing a low degree polynomial running in deterministic $2^{o(n)}$ time.* □

Proof. Given any circuit computing a low degree polynomial, we can convert it to a depth 4 circuit of size $2^{o(n)}$. Further, this conversion can be done in time $2^{o(n)}$ as well. Therefore, a polynomial time PIT algorithm for depth 4 would yield a $2^{o(n)}$ algorithm for general circuits. □

If the PIT on depth 4 circuits was black-box, then we get stronger results for general circuits.

Theorem 3.4 ([22]). *If there is a deterministic* black-box *PIT algorithm for depth 4 circuit running in polynomial time, then there is a deterministic $n^{O(\log n)}$ algorithm for PIT on general circuits computing a low degree polynomial.* □

Proof. Suppose there does indeed exist an optimal pseudorandom generator against depth 4 circuits. By theorem 2.6 we know that we have a

subexponential lower bound in depth 4 circuits for a family of multilinear polynomials $\{q_m\}$. By corollary 3.2 we know that this implies a subexponential lower bound for $\{q_m\}$ in arithmetic circuits of any depth. To finish, theorem 2.2 implies such a family $\{q_m\}$ can be used to give a $n^{O(\log n)}$ algorithm for PIT. □

Therefore, in essence, solving PIT for depth 4 circuits or proving lower bounds for depth 4 circuits would translate to general circuits as well.

We now present a sketch of the proof of Theorem 3.1. The depth reduction is achieved in two stages. The first stage reduces the depth to $O(\log d)$ by the construction of Allender, Jiao, Mahajan and Vinay [33]. Using a careful analysis of this reduction, the circuit is further reduced to a depth 4 circuit.

3.1 *Reduction to depth $O(\log d)$*

Given as input is a circuit C computing a polynomial $p(x_1, \ldots, x_n)$ of degree $d = O(n)$. Without loss of generality, we can assume that the circuit is layered with alternative layers of addition and multiplication gates. Further, we shall assume that each multiplication gate has exactly two children.

Computing degrees

Though the polynomial computed by the circuit is of degree less than d, it could be possible that the intermediate gates compute larger degree polynomials which are somehow canceled later. However, we can make sure that each gate computes a polynomial of degree at most d. Further, we can label each gate by the formal degree of the polynomial computed there.

Each gate g_i of the circuit is now replaced by $d+1$ gates $g_{i0}, g_{i1}, \ldots, g_{id}$. The gate g_{is} would compute the degree s homogeneous part of the polynomial computed at g_i.

If g_0 was an addition gate with $g_0 = h_1 + h_2 + \cdots + h_k$, then we set $g_{0i} = h_{0i} + \cdots + h_{ki}$ for each i. If g_0 was a multiplication gate with two children h_1 and h_2, we set $g_{0i} = \sum_{j=0}^{i} h_{1j} h_{2(i-j)}$.

Thus, every gate is naturally labeled by its degree. As a convention, we shall assume that the degree of the left child of any multiplication gate is smaller than or equal to the degree of the right child.

If the tree was 'balanced', that is, the two children of every multiplication gate have roughly the same degree, then the tree would have depth at most $O(\log d)$. Else, we just need to reorient the tree properly by digging deep enough where the degree is halved. This is done through *proof trees*.

Evaluation through proof trees

A *proof tree* rooted at a gate g is a sub-circuit of C that is obtained as follows:

- start with the sub-circuit in C that has gate g at the top and computes the polynomial associated with gate g,
- for every addition gate in this sub-circuit, retain only one of the inputs to this gate and delete the other input lines,
- for any multiplication gate, retain both the inputs.

A simple observation is that a single proof tree computes one monomial of the formal expression computed at g. And the polynomial computed at g is just the sum of the polynomial computed at every proof tree rooted at g. We shall denote the polynomial computed by a proof tree T as $p(T)$.

For every gate g, define $[g]$ to be the polynomial computed at gate g. Also, for every pair of gates g and h, define $[g,h] = \sum_T p(T,h)$ where T runs over all proof trees rooted at g with h occurring on its rightmost path and $p(T,h)$ is the polynomial computed by the proof tree T when the last occurrence of h is replaced by the constant 1. If h does not occur on the right most path, then $[g,h]$ is zero. The gates of the new circuits are $[g]$, $[g,h]$ and $[x_i]$ for gates $g, h \in C$ and variables x_i. We shall now describe the connections between the gates.

Firstly, $[g] = \sum_i [g, x_i][x_i]$. Also, if g is an addition gate with children $g_1, \ldots, g_k$, then $[g,h] = \sum_i [g_i, h]$. If g is a multiplication gate, it is a little trickier. If the rightmost path from g to h consists of just addition gates, then $[g,h] = [g_L]$, the left child of g. Otherwise, for any fixed rightmost path, there must be at least a unique intermediate multiplication gate p on this path such that

$$\deg(p_R) \leq \frac{1}{2}(\deg g + \deg h) \leq \deg p.$$

Since there could be rightmost paths between g and h, we just run over all gates p that satisfy the above equation. Then, $[g,h] = \sum_p [g,p][p_L][p_R,h]$. We want to ensure that the degree of each child of $[g,h]$ is at most $(\deg(g) - \deg(h))/2$.

- $\deg([g,p]) = \deg(g) - \deg(p) \leq \frac{1}{2}(\deg g - \deg h)$
- $\deg([p_R,h]) = \deg(p_R) - \deg(h) \leq \frac{1}{2}(\deg(g) - \deg(h))$
- $\deg(p_L) \leq \deg(p) \leq \frac{1}{2}\deg(g)$
  Also, $\deg(p_L) \leq \deg(p_L) + \deg(p_R) - \deg(h) \leq \deg(g) - \deg(h)$.

Unfortunately, p_L's degree has not dropped by a factor of 2 and hence this doesn't directly give the

depth reduction. However, we know that $\deg(p_L) \leq \deg(g)/2$. By expanding the gate p_L further, we can obtain an expression of the form

$$[g,h] = \sum [g,p][p_{L,j},q][q_L][q_R,x_i][p_R,h]$$

each of the children have degree at most $(\deg(g) - \deg(h))/2$. This completes the description of the new circuit. It is clear that the depth of the circuit is $O(\log d)$ and the fan-in of multiplication gates is 6. The size of the new circuit is polynomial bounded by size of C.

3.2 *Reduction to depth* 4

We now construct an equivalent depth 4 circuit from the reduced circuit. Let t be a parameter that will be appropriately fixed. The circuit is cut into two parts: the top has exactly t layers of multiplication gates and the rest of the layers belonging to the bottom. Let $g_1, \ldots, g_k$ (where $k \leq S$) be the output gates at the bottom layer. Thus, we can think of the top half as computing a polynomial P_{top} in new variables $y_1, \ldots, y_k$ and each of the g_i computing a polynomial P_i over the input variables. The polynomial computed by the circuit equals

$$P_{\text{top}}(P_1(x_1,\ldots,x_n), P_2(x_1,\ldots,x_n), \ldots, P_k(x_1,\ldots,x_n)).$$

Since the top half consists of t levels of multiplication gates, and each multiplication gate has at most 6 children, $\deg(P_{\text{top}})$ is bounded by 6^t. And since the degree drops by a factor of two across multiplication gates, we also have $\deg(P_i) \leq \frac{d}{2^t}$. Expressing each of these as a sum of product, we have a depth 4 circuit computing the same polynomial. The size of this circuit is

$$\binom{S+6^t}{6^t} + S\binom{n+\frac{d}{2^t}}{\frac{d}{2^t}}.$$

The parameter t can be chosen appropriately to make the size $2^{o(d+d\log\frac{n}{d})}$, as theorem 3.1 claimed.

4. Randomized Algorithms for PIT

Though deterministic algorithms for PIT have remained elusive, a number of randomized solutions are available. Quite an extensive study has been made on reducing the number of random bits. In this section, we inspect a few of them, starting with the oldest, simplest and the most natural test.

4.1 *The Schwarz–Zippel test*

The Schwarz–Zippel test is the oldest algorithm for PIT. The idea is the following: if the polynomial computed is non-zero then the value of the polynomial cannot be zero at too many places. This intuition is indeed true.

Lemma 4.1 ([14,15]). *Let $p(x_1,\ldots,x_n)$ be a non-zero polynomial over $\mathbb{F}$ of total degree d. Let S be any subset of $\mathbb{F}$ and let $a_1,\ldots,a_n$ be chosen independently from S with the uniform distribution. Then, $\mathsf{Pr}[p(a_1,a_2,\ldots,a_n)=0] \leq \frac{d}{|S|}$.* □

To get the error less than ε, we need a set S that is as large as $\frac{d}{\varepsilon}$. Hence, the total number of random bits required would be $n \cdot \log\frac{d}{\varepsilon}$. If the field $\mathbb{F}$ is not large enough, then we may move to an appropriate extension field.

4.2 *Chen–Kao: Evaluating at irrationals*

The Chen–Kao test works on circuits computing an integer polynomial and can be thought of as a 'partial derandomization' of the Schwarz–Zippel test. The main idea can be described as follows:

Suppose we consider only univariate integer polynomials, can we find out a single point on which *all* non-zero univariate integer polynomial evaluate to non-zero values? Indeed, if we evaluate it at some transcendental number like π; $p(\pi) = 0$ for an integer polynomial if and only if $p = 0$. More generally, if we can find suitable irrational that is not a root of any degree d polynomial, we can use that point to evaluate and test if a given degree d polynomial is zero or not. This is the basic idea in Chen–Kao's paper [16].

However, it is infeasible to actually evaluate at irrational points since they have infinitely many bits to represent them. Chen and Kao worked with approximations of the numbers, and introduced randomness to make their algorithm work with high probability.

4.2.1 *Algebraically d-independent numbers*

The goal is to design an identity test for all n-variate polynomials whose degree in each variable is less than d. The following definition is precisely what we want for the identity test.

Definition 4.2 (Algebraically *d*-independence). *A set of number $\{\pi_1,\ldots,\pi_n\}$ is said to be algebraically d-independent over $\mathbb{F}$ if there exists no polynomial relation $p(\pi_1,\ldots,\pi_n)=0$ over $\mathbb{F}$ with the degree of $p(x_1,\ldots,x_n)$ in each variable bounded by d.*

It is clear that if we can find such a set of numbers then this is a single point that would be non-zero at all non-zero polynomials with degree bounded by d. The following lemma gives an explicit construction of such a point.

Lemma 4.3. *Set $k = \log(d+1)$ and $K = nk$. Let $p_{11}, p_{12}, \ldots, p_{1k}, p_{2k}, \ldots, p_{nk}$ be first K distinct primes and let $\pi_i = \sum_{j=1}^{k} \sqrt{p_{ij}}$. Then $\{\pi_1, \ldots, \pi_n\}$ is algebraically d-independent.* □

As remarked earlier, it is not possible to actually evaluate the polynomial at these irrational points. Instead we consider approximations of these irrational points and evaluate them. However, we can no longer have the guarantee that all non-zero polynomials will be non-zero at this truncated value. Chen and Kao solved this problem by introducing randomness in the construction of the π_i.

It is easy to observe that Lemma 4.3 is true even if each $\pi_i = \sum_{j=1}^{k} \alpha_{ij}\sqrt{p_{ij}}$, where each $\alpha_{ij} = \pm 1$. Randomness is introduced by setting each α_{ij} to ± 1 independently and uniformly at random, and then we evaluate the polynomial at the π_i truncated to ℓ decimal places.

Chen and Kao showed that if we want the error to be less than ε, we would have to choose $\ell \geq d^{O(1)} \log n$. Randomness used in this algorithm is for choosing the α_{ij}'s and hence $n \log d$ random bits are used[3]; this is independent of ε! Therefore, to get better accuracy, we don't need to use a single additional bit of randomness but just need to look at better approximations of π_i's.

4.2.2 *Chen–Kao over finite fields*

Though the algorithm that is described seems specific to polynomials over $\mathbb{Q}$, they can be extended to finite fields as well. Lewin and Vadhan [17] showed how to use the same idea over finite fields. Instead of using square root of prime numbers in $\mathbb{Q}$, they use square roots of irreducible polynomials. The infinite decimal expansion is paralleled by the infinite power series expansion of the square roots, with the approximation to ℓ decimal places replaced by taking residues modulo x^ℓ.

Lewin and Vadhan result achieves more or less the exact same parameters as in Chen–Kao and involves far less error analysis as it works over a finite field. They also present another algorithm that works over integers by considering approximations over p-adic numbers, i.e. solutions modulo p^ℓ. This again has the advantage that little error analysis is required.

[3] In fact, if the degree in x_i is bounded by d_i, then $\sum_i \log(d_i + 1)$ random bits would be sufficient.

4.3 *Agrawal–Biswas: Chinese remaindering*

Agrawal and Biswas [18] presented a new approach to identity testing via Chinese remaindering. This algorithm works in randomized polynomial time in the size of the circuit and also achieves the time-error tradeoff as in the algorithm by Chen and Kao. This algorithm can be made to work over all fields but we present the case when it is a finite field $\mathbb{F}_q$.

The algorithm proceeds in two steps. The first is a deterministic conversion to a univariate polynomial of exponential degree. The second part is a novel construction a sample space of 'almost coprime' polynomials that is used for Chinese remaindering.

4.3.1 *Univariate substitution*

Let $f(x_1, \ldots, x_n)$ be the polynomial given as a circuit of size s. Let the degree of f in each variable be less than d. The first step is a well-known conversion to a univariate polynomial of exponential degree that maps distinct monomials to distinct monomials. The following substitution achieves this

$$x_i = y^{d^i}.$$

Claim 4.4. *Under this substitution, distinct monomials go to distinct monomials.*

Denote the univariate polynomial thus produced by $P(x)$ and let the degree be D. We now wish to test whether this univariate polynomial is non-zero. This is achieved by picking a polynomial $g(x)$ from a suitable sample space and doing all computations in the circuit modulo $g(x)$ and return zero if the polynomial is zero modulo $g(x)$.

Suppose these $g(x)$'s came from a set such that the lcm of any ε fraction of them has degree at least D, then the probability of success would be $1 - \varepsilon$. One way of achieving this is to choose a very large set of mutually coprime polynomials say $\{(x - \alpha) : \alpha \in \mathbb{F}_q\}$. But if every epsilon fraction of must have an lcm of degree D, then the size of the sample space must be at least $\frac{D}{\varepsilon}$. This might force us to go to an extension field of $\mathbb{F}_q$ and thus require additional random bits. Instead, Agrawal and Biswas construct a sample space of polynomials that share very few common factors between them which satisfies the above property.

4.3.2 *Polynomials sharing few factors*

We are working with the field $\mathbb{F}_q$ of characteristic p. For a prime number r, let $Q_r(x)$ denote the r-th cyclotomic polynomial, i.e

$Q_r(x) = 1 + x + \cdots + x^{r-1}$. Let $\ell \geq 0$ be a parameter that would be fixed soon. For a sequence of bits $b_0, \ldots, b_{\ell-1} \in \{0,1\}$ and an integer $t \geq \ell$, define

$$A_{b,t}(x) = x^t + \sum_{i=0}^{\ell-1} b_i x^i$$

$$T_{r,b,t}(x) = Q_r(A_{b,t}(x)).$$

The space of polynomials consists of $T_{r,b,t}$ for all values of b, having suitably fixed r and t.

Lemma 4.5. *Let r be a prime such that $r \neq p$ and r does not divide any of $q-1$, $q^2-1, \ldots, q^{\ell-1}-1$ and let t be a fixed parameter. Then, the l cm of any K polynomials from the set $\{T_{r,b,t}(x)\}_b$ has degree at least $K \cdot t$.* □

The algorithm is now straightforward:

1. Set parameters $\ell = \log D$ and $t = \max\left\{\ell, \frac{1}{\varepsilon}\right\}$.
2. Let r is chosen as the smallest prime that doesn't divide any of $p, q-1, q^2-1, \ldots, q^{\ell-1}-1$.
3. Let $b_0, b_1, \ldots, b_{\ell-1}$ be randomly and uniformly chosen from $\{0,1\}$.
4. Compute $P(x)$ modulo $T_{r,b,t}(x)$ and accept if and only if $P(x) = 0 \bmod T_{r,b,t}(x)$.

Since $P(x)$ was obtained from a circuit of size S, we have $D \leq 2^s$. It is easy to see that the algorithm runs in time $\mathsf{poly}\left(s, \frac{1}{\varepsilon}, q\right)$, uses $\log D$ random bits and is correct with probability at least $1 - \varepsilon$.

Remark. Saha [23] observed that there is a deterministic algorithm to find an irreducible polynomial $g(x)$ over $\mathbb{F}_q$ of degree roughly d in $\mathsf{poly}(d, \log q)$ time. Therefore, by going to a suitable field extension, we may even use a sample space of coprime polynomials of the form $x^t + \alpha$ and choose $t = \frac{1}{\varepsilon}$ to bound the error probability by ε. This also uses only $\log D$ random bits and achieves a slightly better time complexity.

4.4 *Klivans–Spielman: Random univariate* substitution

All the previous randomized algorithms use $\Omega(n)$ random bits. It is easy to see that identity testing for n-variate polynomials of total degree bounded by d needs $\Omega(d \log n)$ random bits. For polynomials with m monomials, one can prove a lower bound of $\Omega(\log m)$. Klivans and Spielman [19] present a randomized identity test that uses $O(\log(mnd))$ random bits which works better than the earlier algorithms if m is subexponential.

The idea is to reduce the given multivariate polynomial $f(x_1, \ldots, x_n)$ to a univariate polynomial whose degree is not too large. This reduction will be randomized and the resulting univariate polynomial would be non-zero with probability $1 - \varepsilon$ if the polynomial was non-zero to begin with.

One possible approach is to just substitute $x_i = y^{r_i}$ for each i where r_i's are randomly chosen in a suitable range. This indeed works due to the following lemma. The original version of the *isolation lemma* was by Mulmuley, Vazirani and Vazirani. Their lemma was extended by Chari, Rohatgi and Srinivasan [24], and the parameters improved by Klivans and Spielman [19].

Lemma 4.6 (Isolation lemma [25,24,19]). *Let $\mathcal{F}$ be a family of distinct linear forms $\{\sum_{i=1}^{n} c_i x_i\}$ where each c_i is an integer less than C. If each x_i is randomly set to a value in $\{1, \ldots, Cn/\varepsilon\}$, then with probability at least $1-\varepsilon$ there is a unique linear form of minimum value.* □

The isolation lemma gives a simple randomized algorithm for identity testing of polynomials whose degree in each variable is bounded by d: make the substitution $x_i = y^{r_i}$ for $r_i \in \{1, \ldots, dn/\varepsilon\}$. Each monomial $x_1^{d_1} \ldots x_n^{d_n}$ is now mapped to $y^{\sum d_i r_i}$. Since r_i's are chosen at random, there is a unique monomial which has least degree and hence is never canceled.

However, the number of random bits required is $n \log\left(\frac{dn}{\varepsilon}\right)$. Klivans and Spielman use a different reduction to univariate polynomials which uses $O\left(\log\left(\frac{mnd}{\varepsilon}\right)\right)$ random bits where m is the number of monomials.

4.4.1 *Reduction to univariate polynomials*

Let t be a parameter that shall be fixed later. Pick a prime p larger than t and d. The reduction picks a k at random from $\{0, \ldots, p-1\}$ and makes the following substitution:

$$x_i = y^{a_i}, \quad \text{where} \quad a_i = k^i \bmod p.$$

Lemma 4.7. *Let $f(x_1, \ldots, x_n)$ be a non-zero polynomial whose degree is bounded by d. Then, each monomial of f is mapped to different monomials under the above substitution, with probability at least $\left(1 - \frac{m^2 n}{t}\right)$.* □

If we want the error to be less than ε, then choose $t \geq \frac{m^2 n}{\varepsilon}$. This would make the final degree of the polynomial bounded by $\frac{m^2 nd}{\varepsilon}$ on which we can use a Schwarz–Zippel test by going to a large enough extension. Klivans and Spielman deal with this large degree (since it depends on m) by using the isolation lemma. We now sketch the idea.

4.4.2 *Degree reduction (a sketch)*

The previous algorithm described how a multivariate polynomial can be converted to a univariate polynomial while still keeping each monomial separated. Now we look at a small modification of that construction that uses the isolation lemma to isolate a single non-zero monomial, if present.

The earlier algorithm made the substitution $x_i = y^{a_i}$ for some suitable choice of a_i. Let us assume that each a_i is a q bit number and let $\ell = O(\log(dn))$. The modified substitution is the following:

1. Pick k at random from $\{0, \ldots, p-1\}$ and let $a_i = k^i \bmod p$.
2. Represent a_i in base 2^ℓ as

$$a_i = b_{i0} + b_{i1}2^\ell + \cdots + b_{i\left(\frac{q}{\ell}-1\right)}2^{\left(\frac{q}{\ell}-1\right)\ell}.$$

3. Pick $r_0, \ldots, r_{\frac{q}{\ell}-1}$ values independently and uniformly at random from a small range $\{1, \ldots, R\}$.
4. Make the substitution $x_i = y^{c_i}$ where $c_i = b_{i0}\, r_0 + b_{i1}r_1 + \cdots + b_{i\left(\frac{q}{\ell}-1\right)}r_{\frac{q}{\ell}-1}$.

After this substitution, each monomial in the polynomial is mapped to a power of y, where the power is a linear function over r_i's.

Claim 4.8. *Assume that the choice of k is a positive candidate in lemma 4.7. Then, under the modified substitution, different monomials are mapped to exponents that are different linear functions of the r_i's.* □

Therefore, each exponent of y in the resulting polynomial is a distinct linear function of the r_i's. It is a simple calculation to check that the coefficients involved are $\mathsf{poly}(n, d)$ and we can choose our range $\{1, \ldots, R\}$ appropriately to make sure that the isolation lemma guarantees a unique minimum value linear form with high probability. This means that the exponent of least degree will be contributed by a unique monomial and hence the resulting polynomial is non-zero. The degree of the resulting polynomial is $\mathsf{poly}\left(n, d, \frac{1}{\varepsilon}\right)$ and the entire reduction uses only $O\left(\log\left(\frac{mnd}{\varepsilon}\right)\right)$ random bits.

5. Deterministic algorithms for PIT

In this section we look at deterministic algorithms for PIT for certain restricted circuits. As mentioned above, progress has been made only for restricted versions of depth 3 circuits. Hopefully, some of the techniques developed here would also be useful for designing a deterministic algorithm for depth four PITs.

Easy cases

Depth 2 circuits can only compute 'sparse' polynomials, i.e. polynomials with few monomials in them. PIT of polynomials, where the number of monomials is a polynomial in n can be solved efficiently. The following observation can be directly translated to a polynomial time algorithm.

Observation 5.1. *Let $p(x_1, \ldots, x_n)$ be a non-zero polynomial whose degree in each variable is less than d and the number of monomials is n. Then there exists an $r \leq (mn \log d)^2$ such that*

$$p(1, y, y^d, \ldots, y^{d^{n-1}}) \neq 0 \bmod y^r - 1.$$

Several black-box tests have also been devised for depth 2 circuits but considerable progress has been made for restricted depth 3 circuits as well.

The case when root is a multiplication gate is easy to solve. This is because the polynomial computed by a $\Pi\Sigma\Pi$ circuit is zero if and only if one of the addition gates computes the zero polynomial. Therefore, the problem reduces to depth 2 circuits. Thus, the non-trivial case is $\Sigma\Pi\Sigma$ circuits. PIT for general $\Sigma\Pi\Sigma$ circuits is still open but polynomial time algorithms are known for restricted versions.

5.1 *The Kayal–Saxena test*

Let C be a $\Sigma\Pi\Sigma$ circuit over n variables and degree d such that the top addition gate has k children. For the sake of brevity, we shall refer to such circuits as $\Sigma\Pi\Sigma(n, k, d)$ circuits. Kayal and Saxena [20] presented a $\mathsf{poly}(n, d^k, |\mathbb{F}|)$ algorithm for PIT. Hence, for the case when the top fan-in is bounded, this algorithm runs in polynomial time.

5.1.1 *The idea*

By fixing an ordering of the variables, let $\preceq$ be the induced total order on the monomials. For any polynomial g, let $\mathrm{LM}(g)$ denote the leading monomial of g.

Let C be the given $\Sigma\Pi\Sigma(n, k, d)$ circuit that computes a polynomial f. Therefore, $f = T_1 + \cdots + T_k$ where each $T_i = \prod_{j=1}^{d} L_{ij}$ is a product of the linear forms L_{ij}'s. We can assume without loss of generality that

$$\mathrm{LM}(T_1) \succeq \mathrm{LM}(T_2) \succeq \cdots \succeq \mathrm{LM}(T_k).$$

If f is zero then the coefficient of the $\mathrm{LM}(f)$ must be zero, and this can be checked easily. Further, if we are able to show that $f \equiv 0 \bmod T_1$, then $f = 0$. And this would be done by induction on k.

Suppose T_1 consists of distinct linear forms. Then, by the Chinese remainder theorem, $f \equiv 0 \bmod T_1$ if and only if $f \equiv 0 \bmod L_{1i}$ for each i. To check if $f \equiv 0 \bmod L$ for some linear form L, we replace L by the variable x_1 and transform the rest to make it an invertible transformation. Thus the equation reduces to the form $f \bmod x_1 \in \frac{\mathbb{F}[x_1,\ldots,x_n]}{x_1} = \mathbb{F}[x_2,\ldots,x_n]$. The polynomial $f \bmod x_1$ is now a $\Sigma\Pi\Sigma(n-1,k-1,d)$ circuit and using induction, can be checked if it is zero. Repeating this for every L_{1i}, we can check if $f = 0 \bmod T_1$, and hence check if it is identically zero or not.

This method fails if T_1 happens to have repeated factors. For example, if $T_1 = x_1^5 x_2^3$, we should instead be checking if $f \bmod x_1^5$ and $f \bmod x_2^3$ are zero or not. Here $f \bmod x_1^5 \in \frac{\mathbb{F}[x_1,\ldots,x_n]}{x_1^5} = \left(\frac{\mathbb{F}[x_1]}{x_1^5}\right)[x_2,\ldots,x_n]$ is a polynomial over a *local ring*, not a field. Thus, in the recursive calls, the computations would be over a local ring rather than over the field. Therefore, we need to make sure that Chinese remaindering works over local rings; Kayal and Saxena showed that it indeed does.

We are now set to look at the identity test.

5.1.2 *The identity test*

Let C be a $\Sigma\Pi\Sigma$ arithmetic circuit, with top fan-in k and degree d computing a polynomial f. The algorithm is recursive where each recursive call decreases k but increases the dimension of the base ring (which is $\mathbb{F}$ to begin with).

Input:

The algorithm takes three inputs:

- A local ring $\mathcal{R}$ over a field $\mathbb{F}$ with the maximal ideal $\mathcal{M}$ of R presented in its basis form. The initial setting is $\mathcal{R} = \mathbb{F}$ and $\mathcal{M} = \langle 0 \rangle$.
- A set of k coefficients $\langle \beta_1, \ldots, \beta_k \rangle$, with $\beta_i \in \mathcal{R}$ for all i.
- A set of k terms $\langle T_1, \ldots, T_k \rangle$. Each T_i is a product of d linear functions in n variables over $\mathcal{R}$. That is, $T_i = \prod_{j=1}^{d} L_{ij}$.

Output:

Let $p(x_1,\ldots,x_n) = \beta_1 T_1 + \cdots, \beta_k T_k$. The output, $\mathrm{ID}(\mathcal{R}, \langle \beta_1, \ldots, \beta_k \rangle, \langle T_1, \ldots, T_k \rangle)$ is YES if and only if $p(x_1,\ldots,x_n) = 0$ in $\mathcal{R}$.

Algorithm:

Assume without loss of generality that $\mathrm{LM}(T_1) \succeq \cdots \succeq \mathrm{LM}(T_k)$.

Step 1: Check if the coefficient of $\mathrm{LM}(T_1)$ is a unit.

Step 2: (Single multiplication gate) If $k = 1$, we need to test if $\beta_1 T_1 = 0$ in $\mathcal{R}$. Check if $\beta_1 = 0$.

Step 3: (Checking if $p = 0 \bmod T_1$) Write T_1 as a product of coprime factors, where each factor is of the form

$$S = (l + m_1)(l + m_2)\cdots(l + m_t)$$

with $l \in \mathbb{F}[x_1,\ldots,x_n]$ and $m_i \in \mathcal{M}$ for all i.

For each such factor S, do the following:

Step 3.1: (Change of variables) With a suitable invertible linear transformation σ on the variables, make convert l to x_1.

Step 3.2: (Recursive calls) The new ring $\mathcal{R}' = R[x_1]/(\sigma(S))$ which is a local ring as well. For $2 \le i \le k$, the transformation σ might convert some of the factors of T_i to an element of $\mathcal{R}'$. Collect all such ring elements of $\sigma(T_i)$ as $\gamma_i \in \mathcal{R}'$ and write $\sigma(T_i) = \gamma_i T_i'$.

Recursively call $\mathrm{ID}(\mathcal{R}', \langle \beta_2\gamma_2, \ldots, \beta_k\gamma_k \rangle, \langle T_2', \ldots, T_k' \rangle)$. If the call returns NO, exit and output NO.

Step 4: Output YES.

It is fairly straight-forward to check that the algorithm is indeed right, and runs in time $\mathsf{poly}(n, d^k, |\mathbb{F}|)$. This completes the Kayal–Saxena identity test for $\Sigma\Pi\Sigma$ circuits with bounded top fan-in.

5.2 *Black-box algorithm for $\Sigma\Pi\Sigma(n,k,d)$ circuits over $\mathbb{Q}$*

The *rank approach* asks the following question: if C is a $\Sigma\Pi\Sigma$ circuit that indeed computes the zero polynomial, then how many variables does it *really* depend on? To give a reasonable answer, we need to assume that the given circuit is not 'redundant' in some ways.

Definition 5.2 (Minimal and simple circuits). *A $\Sigma\Pi\Sigma$ circuit $C = P_1 + \cdots + P_k$ is said to be* minimal *if no proper subset of $\{P_i\}_{1 \le i \le k}$ sums to zero.*

The circuit is said to be simple *there is no non-trivial common factor between all the P_i's.*

Definition 5.3 (Rank of a circuit). *For a given circuit $\Sigma\Pi\Sigma$ circuit, the* rank *of the circuit is the maximum number of independent linear functions that appear as a factor of any product gate.*

Suppose we can get an upper-bound R on the rank of any minimal and simple $\Sigma\Pi\Sigma(n,k,d)$ circuit computing the zero polynomial. Then we have a partial approach towards identity testing.

1. Without loss of generality, we may assume that the circuit is simple and minimal.
2. Compute the rank r of the circuit C.
3. If the $r < R$ is small, then the circuit is essentially a circuit on just R variables. We can check in d^R time if C is zero or not.
4. If the rank is larger than the upper-bound then the circuit computes a non-zero polynomial.

This was in fact the idea in Dvir and Shpilka's $n^{O(\log n)}$ algorithm [26] for $\Sigma\Pi\Sigma$ circuits of bounded top fan-in (before the algorithm by Kayal and Saxena [20]). Saxena and Seshadri recently showed rank upper bounds that are almost tight.

Theorem 5.4 ([27]). *Let C be a minimal, simple $\Sigma\Pi\Sigma(n,k,d)$ circuit that is identically zero. Then, rank $(C) = O(k^3 \log d)$. And there exist identities with rank $\Omega(k \log d)$.* □

Karnin and Shpilka showed how rank bounds can be turned into black-box identity tests. Using the bound by Saxena and Seshadri, this gave a $n^{O(\log n)}$ black-box test for depth 3 circuits with bounded top fan-in.

Theorem 5.5 ([28]). *Fix a field $\mathbb{F}$. Let $R(k,d)$ be an integer such that every minimal, simple $\Sigma\Pi\Sigma(n,k,d)$ circuit computing the zero polynomial has rank at most $R(k,d)$. Then, there is a black-box algorithm to test if a given $\Sigma\Pi\Sigma(n,k,d)$ circuit is zero or not, in deterministic time $\mathsf{poly}(d^{R(k,d)}, n)$.*

It was conjectured by Dvir and Shpilka [26] that $R(k,d)$ is a polynomial function of k alone. However, Kayal and Saxena [20] provided a counter-example over finite fields. The question remained if $R(k,d)$ is a function of k alone over $\mathbb{Q}$ or $\mathcal{R}$. This was answered in the affirmative by Kayal and Saraf [21] very recently.

Theorem 5.6 ([21]). *Every minimal, simple $\Sigma\Pi\Sigma(n,k,d)$ circuit with coefficients from $\mathcal{R}$ that computes the zero polynomial has rank bounded by $3^k((k+1)!) = 2^{O(k \log k)}$.*

This, along with theorem 5.5, gives a black-box algorithm for $\Sigma\Pi\Sigma$ circuits with bounded top fan-in.

Theorem 5.7 ([21]). *There is a deterministic black-box algorithm for $\Sigma\Pi\Sigma(n,k,d)$ circuits over $\mathbb{Q}$, running in time $\mathsf{poly}(d^{2^{O(k \log k)}}, n)$.*

5.3 *Saxena's test for diagonal circuits*

In this section we shall look at yet another restricted version of depth 3 circuits.

Definition 5.8. *A $\Sigma\Pi\Sigma$ circuit C is said to be* diagonal *if it is of the form*

$$C(x_1,\ldots,x_n) = \sum_{i=1}^{k} \ell_i^{e_i},$$

where ℓ_i is a linear function over the variables.

The idea is to reduce this problem to a PIT problem of a formula over non-commuting variables. In the setting of formulas over non-commuting variables, Raz and Shpilka [29] showed that PIT can be solved in deterministic polynomial time.

The reduction to a non-commutative formula is by a conversion to express a multiplication gate $(a_0 + a_1x_1 + \cdots a_nx_n)^d$ in a *dual* form:

$$(a_0 + a_1x_1 + \cdots a_nx_n)^d = \sum_j f_{j1}(x_1)f_{j2}(x_2)\ldots f_{jn}(x_n).$$

The advantage of using the expression on the RHS is that the variables can be assumed to be non-commuting. Therefore if the above conversion can achieved in polynomial time, then we have a polynomial algorithm for identity testing of diagonal circuits by just making this transformation and using the algorithm by Raz and Shpilka. Saxena provides a simple way to convert a multiplication gate to its dual. We present the case when $\mathbb{F}$ is a field of characteristic zero though it may be achieved over any field.

Lemma 5.9 ([30]). *Let $a_0,\ldots,a_n$ be elements of a field $\mathbb{F}$ of characteristic zero. Then, in $\mathsf{poly}(n,d)$ many field operations, we can compute univariate polynomials $f_{i,j}$'s such that*

$$(a_0 + a_1x_1 + \cdots a_nx_n)^d = \sum_{i=1}^{nd+d+1} f_{i1}(x_1)f_{i2}(x_2)\cdots f_{in}(x_n).$$ □

Once we obtain such a transformation, we can think of the transformed circuit as computing a non-commuting polynomial and employ the test by Raz and Shpilka. Thus, we get a deterministic polynomial time PIT for diagonal circuits.

The lemma can be extended to restricted forms of depth 4 circuits as well, giving the following theorem.

Theorem 5.10 ([30]). *Given a circuit C over a field $\mathbb{F}$ with*

$$C = \sum_{i=1}^{k} L_{i1}^{e_{i1}} \cdots L_{is}^{e_{is}},$$

where each L_{ij} is a sum of univariate polynomials. We can test if C is identically zero or not in deterministic time $\mathsf{poly}(\mathrm{size}(C), \max_{i \le k} \{(1+e_{i1}) \cdots (1+e_{is})\})$. □

Though $(1+e_{i1}) \cdots (1+e_{is})$ could be exponential in general circuits, they perform well if the multiplication gates have 'few' distinct factors.

5.4 *Circuits over algebras*

A possible approach towards a deterministic polynomial time algorithm for general $\Sigma\Pi\Sigma$ circuits is to look at generalizations of PIT to other algebras (where the underlying constants come, not from $\mathbb{F}$, but an algebra over $\mathbb{F}$). Saha, Saptharishi and Saxena [31] showed that $\Pi\Sigma$ circuits over algebras are strongly connected to PIT for $\Sigma\Pi\Sigma$ circuits over fields.

Theorem 5.11 ([31]). *PIT for $\Sigma\Pi\Sigma$ circuits over fields is polynomial time equivalent to PIT for $\Pi\Sigma$ circuits over $\mathsf{U}_2(\mathbb{F})$, the algebra of 2×2 upper-triangular matrices.* □

The above theorem can also be re-written in terms of *algebraic branching programs* as follows:

Corollary 5.12. *PIT for $\Sigma\Pi\Sigma$ circuits over fields is polynomial time equivalent to PIT on restricted width 2 algebraic branching program.* □

But why should $\Pi\Sigma$ circuits over algebras be easier to attack than $\Sigma\Pi\Sigma$ circuits? Saha, Saptharishi and Saxena further showed that the problem is tractable if the underlying algebra was a constant dimensional commutative algebra over $\mathbb{F}$.

Theorem 5.13 ([31]). *PIT for $\Pi\Sigma$ circuits over k-dimensional commutative algebras can be solved in $O(n^k)$ time.* □

The proof decomposes uses a decomposition of the underlying algebra into local rings to reduce the problem into smaller subproblems. Perhaps similar mathematical tools can be used to attack $\mathsf{U}_2(\mathbb{F})$ and hence depth 3 circuits.

6. Conclusions

Finding a deterministic algorithm for PIT for depth 3 and 4 circuits remains a very challenging problem and is being investigated actively. There is a hope that in near future we would be able to find such algorithms and use them to show that permanent polynomial is hard to compute.

References

[1] Stuart Berkowitz J 1984 On computing the determinant in small parallel time using a small number of processors; *Inf. Process. Lett.* **18 (3)** 147–150.

[2] Leslie Valiant G 1979 Completeness classes in algebra; in *STOC* 249–261.

[3] Valentine Kabanets and Russell Impagliazzo 2003 Derandomizing polynomial identity tests means proving circuit lower bounds; in *STOC* 355–364.

[4] Manindra Agrawal 2005 Proving lower bounds via pseudo-random generators; in *FSTTCS* 92–105.

[5] Ran Raz and Amir Yehudayoff 2008 Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors; in *FOCS '08: Proc. 2008 49th Annual IEEE Symposium on Foundations of Computer Science* 273–282 (Washington, DC USA: 2008. IEEE Computer Soc.).

[6] Nisan N and Wigderson A 1995 Lower bounds on arithmetic circuits via partial derivatives; in *FOCS '95: Proc. 36th Annual Symposium on Foundations of Computer Science* 16 (Washington DC, USA: IEEE Computer Soc.).

[7] Amir Shpilka and Avi Wigderson 1999 Depth-3 arithmetic formulae over fields of characteristic zero; in *COCO '99: Proc. Fourteenth Annual IEEE Conference on Computational Complexity* 87 (Washington DC, USA: IEEE Computer Soc.).

[8] Dima Grigoriev and Marek Karpinski 1998 An exponential lower bound for depth 3 arithmetic circuits; in *STOC '98: Proc. thirtieth annual ACM symposium on theory of computing* 577–582 (New York, USA: ACM).

[9] Adi Shamir 1990 IP=PSPACE; in *FOCS* 11–15.

[10] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan and Mario Szegedy 1998 Proof verification and the hardness of approximation problems; *J. ACM* **45 (3)** 501–555.

[11] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena 2004 PRIMES is in P; *Ann. Math* **160 (2)** 781–793.

[12] László Lovász 1979 On determinants, matchings, and random algorithms; in *FCT* 565–574.

[13] Michael Clausen, Andreas Dress W M, Johannes Grabmeier and Marek Karpinski 1991 On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields; *Theor. Comput. Sci.* **84 (2)** 151–164.

[14] Jacob Schwartz T 1980 Fast probabilistic algorithms for verification of polynomial identities; *J. ACM* **27 (4)** 701–717.

[15] Richard Zippel 1979 Probabilistic algorithms for sparse polynomials; *EUROSAM* 216–226.

[16] Zhi-Zhong Chen and Ming-Yang Kao 1997 Reducing randomness via irrational numbers; in *STOC* 200–209.

[17] Daniel Lewin and Salil Vadhan P 1998 Checking polynomial identities over any field: Towards a derandomization? in *STOC* 438–447.

[18] Manindra Agrawal and Somenath Biswas 1999 Primality and identity testing via Chinese remaindering; in *FOCS* 202–209.

[19] Adam Klivans and Daniel Spielman A 2001 Randomness efficient identity testing of multivariate polynomials; in *STOC* 216–223.

[20] Neeraj Kayal and Nitin Saxena 2007 Polynomial identity testing for depth 3 circuits; *Computational Complexity* **16 (2)**.

[21] Neeraj Kayal and Shubhangi Saraf 2009 Blackbox polynomial identity testing for depth 3 circuits; *Electronic Colloquium on Computational Complexity (ECCC)*.

[22] Manindra Agrawal and Vinay V 2008 Arithmetic circuits: A chasm at depth four; in *FOCS* 67–75.

[23] Chandan Saha 2008 A note on irreducible polynomials and identity testing; (Manuscript) http://www.cse.iitk.ac.in/users/csaha/PID_CR.pdf.

[24] Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan 1995 Randomness-optimal unique element isolation with applications to perfect matching and related problems; *SIAM J. Comput.* **24 (5)** 1036–1050.

[25] Ketan Mulmuley, Umesh Vazirani V and Vijay Vazirani V 1987 Matching is as easy as matrix inversion; in *STOC '87: Proc. nineteenth annual ACM symposium on theory of computing* 345–354 (New York, USA: ACM).

[26] Zeev Dvir and Amir Shpilka 2005 Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits; in *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on theory of computing* 592–601 (New York, USA: ACM).

[27] Nitin Saxena and Seshadhri C 2009 An almost optimal rank bound for depth 3 identities; in *Conference on Computational Complexity*.

[28] Zohar Karnin S and Amir Shpilka 2008 Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in; in *CCC '08: Proc. 2008 IEEE 23rd Annual Conference on Computational Complexity* 280–291 (Washington DC, USA: IEEE Computer Soc.)

[29] Ran Raz and Amir Shpilka 2004 Deterministic polynomial identity testing in non-commutative models; in *IEEE Conference on Computational Complexity* 215–222.

[30] Nitin Saxena 2008 Diagonal circuit identity testing and lower bounds; in *ICALP '08: Proc. 35th international colloquium on automata, languages and programming Part I* 60–71 (Berlin, Heidelberg: Springer-Verlag.).

[31] Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena 2009 The power of depth 2 circuits over algebras. (Manuscript) http://arxiv.org/abs/0904.2058.

[32] Zeev Dvir, Amir Shpilka, and Amir Yehudayoff 2008 Hardness-randomness tradeoffs for bounded depth arithmetic circuits; in *STOC '08: Proc. 40th annual ACM symposium on theory of computing* 741–748 (New York, USA: ACM).

[33] Eric Allender, Jia Jiao, Meena Mahajan and Vinay V 1998 Non-commutative arithmetic circuits: depth reduction and size lower bounds; *Theor. Comput. Sci.* **209 (1–2)** 47–86.