Polynomial Identity Testing and its relation to some Algebraic Problems.

Ilya Volkovich
Polynomial Identity Testing and its relation to some Algebraic Problems.

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Ilya Volkovich

Submitted to the Senate of the Technion — Israel Institute of Technology
Kislev 5772 Haifa December 2011
The research thesis was done under the supervision of Prof. Amir Shpilka in the Computer Science Department.

I would like to take this opportunity to express my deepest gratitude to my advisor, Prof. Amir Shpilka. Thank you for your wise guidance and your strong support all the way to this very moment.

The generous financial support of the Technion, the Israel Science Foundation (grant number 439/06), and the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575 is gratefully acknowledged.
Contents

Abstract 1

Abbreviations and Notations 2

1 Introduction 3
   1.1 Polynomial Identity Testing (PIT) 3
   1.2 Polynomial Factorization 6
   1.3 Circuit Reconstruction, Minimization and C-testing 6
   1.4 Objective and Results 8
   1.5 Related Works 9
   1.6 Organization 10
   1.7 Source 10

2 Basic Notations, Tools and Techniques 11
   2.1 Preliminaries 11
      2.1.1 Partial Derivatives 13
      2.1.2 Polynomials and Circuit Classes 15
      2.1.3 Some Useful Facts about Polynomials 16
   2.2 Generators and Hitting Sets for Arithmetic Circuits 16
      2.2.1 The Generator $G_k$ 18
   2.3 From PIT to Common Justifying Assignments 19
      2.3.1 From a Generator to a Justifying Set 22
   2.4 Black-box PIT Requires Large (extension) Fields 23

3 PIT for Read-Once Formulas and related circuit classes 24
   3.1 Introduction 25
      3.1.1 Our Results and Techniques 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2</td>
<td>Proof Technique</td>
<td>27</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Comparison to Previous Works</td>
<td>29</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Organization</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Our Model</td>
<td>30</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Read-Once Formulas and Read-Once Polynomials</td>
<td>30</td>
</tr>
<tr>
<td>3.2.2</td>
<td>The Gates-Graph of ROFs and ROPs</td>
<td>31</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Factors of ROPs</td>
<td>34</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Partial Derivatives of ROPs</td>
<td>37</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Multiplicative and 0-Justified ROPs</td>
<td>38</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Preprocessed Read-Once Polynomials</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>Black-Box PIT for Preprocessed Read-Once Polynomials</td>
<td>45</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Small Depth Preprocessed Alternating Read-Once Formulas</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>PIT for Sum of Preprocessed Read-Once Formulas</td>
<td>49</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Hardness of Representation</td>
<td>50</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Non Black-Box Identity Testing Algorithm for Sum of PROPs</td>
<td>55</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Black-Box Identity Testing Algorithm for Sum of PROPs</td>
<td>56</td>
</tr>
<tr>
<td>3.5</td>
<td>Depth-3 $\Sigma\Pi\Sigma(k)$ Circuits</td>
<td>57</td>
</tr>
<tr>
<td>3.5.1</td>
<td>New Black-Box PIT Algorithm for Depth-3 Circuits</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>PIT for Depth-4 Circuits</td>
<td>62</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>63</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Overview of the Proof of The Structure Theorem</td>
<td>65</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Overview of the Black-Box Algorithm</td>
<td>67</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Related Previous Results</td>
<td>68</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Organization</td>
<td>71</td>
</tr>
<tr>
<td>4.2</td>
<td>Preliminaries</td>
<td>71</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Sparsity of a Polynomial</td>
<td>71</td>
</tr>
<tr>
<td>4.2.2</td>
<td>The Operator $D_{\ell}$</td>
<td>74</td>
</tr>
<tr>
<td>4.3</td>
<td>Depth-4 Multilinear Circuits</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>The Sparsity Bound</td>
<td>76</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Lower Bound</td>
<td>83</td>
</tr>
<tr>
<td>4.5</td>
<td>Black-Box PIT for Multilinear $\Sigma\Pi\Sigma\Pi(k)$ Circuits</td>
<td>84</td>
</tr>
<tr>
<td>4.6</td>
<td>Non Black-Box PIT for Multilinear $\Sigma\Pi\Sigma\Pi(k)$ Circuits</td>
<td>88</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Overview</td>
<td>88</td>
</tr>
</tbody>
</table>
## 4.6.2 The Algorithm ............................................. 89

## 4.7 Missing Proofs ............................................. 90

## 5 Variable Disjoint Factors ................................ 92

### 5.1 Introduction ............................................... 93

#### 5.1.1 Polynomial Decomposition ............................. 93

#### 5.1.2 Our Results ............................................ 94

#### 5.1.3 Motivation .............................................. 95

#### 5.1.4 Proof Technique ....................................... 96

#### 5.1.5 Related Works .......................................... 97

#### 5.1.6 Organization ........................................... 98

### 5.2 Indecomposable Polynomials .............................. 98

### 5.3 Decomposition ............................................ 99

#### 5.3.1 Some Corollaries ....................................... 101

### 5.4 Extensions of Theorem 11 ................................. 103

### 5.5 Concluding remarks ..................................... 106

## 6 Read-Once Reconstruction and Testing .................... 107

### 6.1 Introduction ............................................... 108

#### 6.1.1 Comparison to Previous Works ....................... 112

#### 6.1.2 Organization ........................................... 113

### 6.2 Reconstruction of a PROF ............................... 114

#### 6.2.1 Reconstruction of a \( \bar{0} \)-justified PROF ...... 114

#### 6.2.2 Reconstruction of a General PROF ................... 120

### 6.3 Read-Once Testing ...................................... 123

#### 6.3.1 Generic Scheme ....................................... 124

#### 6.3.2 Read-Once Verification ............................... 125

#### 6.3.3 Proof of Theorem 15 .................................. 129

### 6.4 ROT for Specific Models ................................. 130

#### 6.4.1 Sparse Polynomials .................................. 130

#### 6.4.2 Depth-3 Circuits ..................................... 131

#### 6.4.3 Read-Once Testing for Sum of \( k \) PROFs .......... 132

### 6.5 PROF Graph-Related Algorithms ....................... 132

#### 6.5.1 Factoring a ROF ..................................... 133

#### 6.5.2 Counting the Number of Monomials in a PROP ...... 134

## 7 Conclusions and Open Questions .......................... 137
# List of Algorithms

1. Find Common Justifying Assignment ................................................. 20
2. PIT algorithm for sum of preprocessed read-once formulas ............. 55
3. Construction of a hitting set for $\Sigma\Pi\Sigma\Pi(k)$ circuits of size $s$ .... 87
4. Non-Black PIT algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits ........ 90
5. Finding variable partition ................................................................. 99
6. Reconstruct $\overline{0}$-justified PROF .............................................. 114
7. Learn the Preprocessing, $\text{var}_0(P)$ and a Witness ....................... 116
8. Construct The Gates Graph .............................................................. 116
9. Top Gate + Case .............................................................................. 118
10. Top Gate $\times$ Case ................................................................. 119
11. Non-Adaptive PROF Reconstruction ............................................ 123
12. Generic ROT Scheme ................................................................. 124
13. Verify $(C, \Phi)$ ........................................................................... 127
14. FactorROF($\Phi$) ........................................................................... 134
15. CountMonomials($v$) ...................................................................... 135
Abstract

In this thesis we study the fundamental problem of Polynomial Identity Testing (PIT). The problem is one of a few natural problems that have an efficient randomized algorithm (coRP) but lack deterministic ones. We present several deterministic algorithms for several important instances of PIT. In addition, we show connections between PIT and other problems sharing this virtue. In particular, we show that PIT is essentially equivalent to some restricted version of the polynomial factorization problem; and present a generic scheme that can be used to strengthen efficient PIT algorithms to yield efficient algorithms for testing whether a given polynomial can be computed by an arithmetic circuit of a specific form.
Abbreviations and Notations

\( \mathbb{F} \) — Algebraic Field
\( \mathbb{F}[x_1, \ldots, x_n] \) — The ring of \( n \)-variate polynomials over the field \( \mathbb{F} \)
\( \Sigma \Pi \Sigma(k) \) — The class of depth-3 circuits with top fan-in \( k \)
\( \Sigma \Pi \Sigma \Pi(k) \) — The class of depth-4 circuits with top fan-in \( k \)
\( (P)\text{ROF} \) — (preprocessed) read-once formula
\( (P)\text{ROP} \) — (preprocessed) read-once polynomial
\( \text{var}(P) \) — The set of variables the polynomial \( P \) depends on.
\( \mathcal{H} \) — A hitting set
\( \mathcal{C} \) — A circuit class
Chapter 1

Introduction

Arithmetic circuits are the most natural framework for computing polynomials. An arithmetic circuit in the variables \( X = \{ x_1, \ldots, x_n \} \), over the field \( \mathbb{F} \), is a labelled directed acyclic graph. The inputs (nodes of in-degree zero) are labelled by variables from \( X \) or by constants from the field. The internal nodes are labelled by \( + \) or \( \times \), computing the sum and product, resp., of the polynomials on the tails of incoming edges (subtraction is obtained using the constant \(-1\)). A formula is a circuit whose nodes have out-degree one (namely, a tree). The output of a circuit (formula) is the polynomial computed at the output node. The size of a circuit (formula) is the number of gates in it. The depth of the circuit (formula) is the length of a longest path between the output node and an input node.

It is important to note that all other implicit polynomial computation forms/models (such as: symbolic determinant, straight-line programs, etc.) can be efficiently transformed into arithmetic circuits.

In this thesis we deal with several algebraic problems that can be described in the language of arithmetic circuits.

1.1 Polynomial Identity Testing (PIT)

A central problem in algebraic complexity theory and algorithms design is the problem of Polynomial Identity Testing (PIT): given an arithmetic circuit \( C \) over a field \( \mathbb{F} \), with input variables \( x_1, x_2, \ldots, x_n \), determine whether \( C \) computes the identically zero polynomial. Numerous applications and
connections to other algorithmic and number theoretic problems further emphasize the significance of PIT. Among the examples are algorithms for finding perfect matchings in graphs [46, 48], primality testing [2], and many more. In addition, PIT also shows up in many fundamental results in complexity theory such as \(\text{IP} = \text{PSPACE}\) [47, 62] and the PCP theorem [8, 7].

PIT is one of the most basic and natural questions for which a very simple randomized solution is known: Schwartz and Zippel [61, 69] independently showed that if one evaluates the circuit at a randomly chosen point from a sufficiently large domain, then with high probability any non-zero circuit will evaluate to a non-zero value. It has been a long standing open question to derandomize the algorithm.

The main open question is to come up with an efficient (i.e. polynomial-time or at least sub-exponential-time) deterministic algorithm for the problem. Indeed, Kabanets and Impagliazzo [31] showed that any deterministic algorithm for identity testing implies super polynomial circuit lower bounds: either \(\text{NEXP} \not\subseteq \text{P}/\text{poly}\) or the Permanent has no polynomial size arithmetic circuits. Other connections between deterministic PIT algorithms and circuits lower bounds were given in [28, 18].

A very natural and often desirable setting to consider the PIT question is in the black-box model. The connection to lower bounds is even more natural and strong in this case. In the black-box setting, one is not given the full description of the circuit \(C\) but only allowed black-box (oracle) access to \(C\). The problem of derandomizing identity testing in this setting reduces to that of finding for every integer \(s\) an explicit set of points \(H \subseteq \mathbb{F}^n\) of size \(\text{poly}(s)\) such that any non-zero circuit of size \(s\) does not vanish on \(H\). We refer to such sets as hitting sets. The Schwartz-Zippel test, in fact, provides an exponential-size hitting set. Furthermore, applying standard probabilistic arguments one can show existence of “small” hitting sets. Interestingly, any explicit construction of a hitting set (for any class of circuits) immediately gives, via interpolation, an explicit multilinear polynomial\(^1\) that cannot be computed by that class of circuits [1].

In a recent surprising result by Agrawal and Vinay [3], it was shown that a complete derandomization of black-box identity testing for just depth-4 \(\Sigma\Pi\Sigma\Pi\) arithmetic circuits (for more details see Definition 4.3.1) already

\(^1\)A multilinear polynomial is a polynomial in which the individual degree of each variable is at most 1.
implies a near complete derandomization for the general PIT problem. More precisely, they showed that black-box identity testing for depth-4 (ΣΠΣΠ) arithmetic circuits implies exponential lower bounds for general arithmetic circuits, which in turn implies a quasi-polynomial-time algorithm for the general PIT problem. This makes black-box identity testing for even very low depth circuits a very rewarding pursuit!

In view of the difficulty in providing efficient deterministic PIT algorithms and the tight connection to lower bounds it is natural to study the PIT problem for models for which lower bounds are known. In particular, the recent results of [51, 52, 53, 55], giving lower bounds for multilinear circuits and formulas, suggest that efficient deterministic PIT algorithms for multilinear formulas may be at reach. Unfortunately, except for the models of multilinear depth-2 circuits and multilinear ΣΠΣ(k) circuits (see Definition 3.5.2) no such algorithm is known.² This difficulty motivates the study of restricted models of multilinear formulas in hope of gaining insight on the general case.

For a long time, black-box identity tests were only known for depth-2 circuits (equivalently circuits computing sparse polynomials) [11, 43, 45] (and references within). In light of the Agrawal-Vinay result, studying black-box identity testing for depth-3 and depth-4 circuits seems to be a very promising direction and line of attack for the general PIT problem. In recent years there has been a surge of results on black-box (and non-black-box) identity testing for some classes of depth-3 circuits such as depth-3 circuits with bounded top fan-in (also known as ΣΠΣ(k) circuits) [17, 41, 37, 40, 58, 9, 59, 60], and even some restricted classes of depth-4 circuits [57, 36, 9].

It is important to stress that PIT asks whether the resulting polynomial is identically zero as a formal sum of monomials and not just as a function. For example, the polynomial $x^2 - x$ is a nonzero polynomial over any field, although it represents the zero function over the field with 2 elements. Therefore, to avoid the aforementioned problem, in the setting of PIT we can always assume that the underlying field $\mathbb{F}$ is large enough, or that we have an access to some polynomially large extension field of $\mathbb{F}$. In Lemma 2.4.1 we observe that we cannot hope to achieve efficient black-box PIT

²The recent works of [29, 30, 36, 56, 5] that deal with richer classes of multilinear circuits use several ideas that appeared in the conference versions of this work ([64, 65]).
algorithms if we restrict ourselves to small (i.e. constant size) fields.

1.2 Polynomial Factorization

The polynomial factorization problem is another fundamental problem in algebraic complexity. We are given an arithmetic circuit computing a multivariate polynomial, over some field $\mathbb{F}$, and we have to compute its irreducible factors. Similarly to the case of PIT, this problem is considered in both the black-box and the non black-box settings. When the arithmetic circuit is explicitly given to us, we have to find arithmetic circuits for its irreducible factors (it is known that if the circuit is of size $s$ and the degree of the computed polynomial is at most $d$ then the irreducible factors have arithmetic circuits of size $\text{poly}(s, d)$ [32]). When we are only given black-box access to the circuit, we have to output black-boxes for its irreducible factors.

The polynomial factorization problem is one of the most studied questions in algebraic complexity and a large amount of research was devoted to finding efficient algorithms for it (see e.g. [21]). One well-known application is list decoding [67, 26]. Several randomized algorithms were designed [21, 33, 20] when the underlying polynomial is given as a list of monomials. In [32], Kaltofen gave an efficient probabilistic algorithm for computing the irreducible factors of polynomials given by arithmetic circuits. In [34] Kaltofen and Trager gave a polynomial time probabilistic algorithm for black-box factoring of multivariate polynomials. The question of whether there exist deterministic algorithms for the problem is an interesting and long standing open question (see [21, 39]). We note that even for the univariate case, a lot of effort was invested in trying to derandomize the factorization algorithm when the characteristic of $\mathbb{F}$ is large (see e.g. [21, 19, 39]).

1.3 Circuit Reconstruction, Minimization and $C$-testing

The reconstruction problem for arithmetic circuits is defined as follows. Given a black-box (oracle) access to a polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ computed by an arithmetic circuit from some circuit class $C$, output a circuit from $C$ that computes $P$. 
As mentioned earlier, a deterministic black-box PIT algorithm for a circuit class $C$ can be regarded as a “hitting” set. That is, a set of points $H$ such that if a circuit from $C$ evaluates to zero over $H$ then it must compute the zero polynomial. One of the main properties of such $H$ is that the values of a circuit from $C$ on $H$ can be used to uniquely describe that circuit, in the sense that two circuits that agree on $H$ must compute the same polynomial, as their difference evaluates to zero over $H$. Yet, such a set does not provide us with an efficient algorithm for reconstructing circuits from $C$. On the other hand, a deterministic reconstruction algorithm can be used as a black-box PIT algorithm.

Another important problem is the circuit (or formula) minimization problem: Given a circuit $C$ (explicitly or via black-box) find a minimal circuit (or formula) $\hat{C}$ that computes the same polynomial (that is $\hat{C} \equiv C$). In a similar fashion to various optimization problems, the problem is also considered in its decision version: Given a circuit $C$ decide whether there exists a smaller circuit (or formula) that computes the same polynomial. In the more general form the problem can be stated as follows: Let $\mathcal{C}$ be a circuit class. Given a circuit $C$ determine whether there exists $\hat{C} \in \mathcal{C}$ such that $C \equiv \hat{C}$ (and, if the answer is “yes” output it). We refer to this problem as the $C$-testing (and reconstruction) problem. It can be easily seen that the minimization problem can be formulated as a sequence of $C$-testing problems, by choosing $\mathcal{C}_1, \mathcal{C}_2, \ldots$ to be arithmetic circuits of monotonically decreasing sizes.

In the Boolean world the decision version of the minimization problem is known to be in $\Sigma_2 = \text{NP}^{\text{NP}}$ and is also an NP-hard problem (even when the formula is a 3-CNF). Using the Schwartz-Zippel lemma [69, 61] it can be easily shown that the decision version of the arithmetic minimization problem is in $\text{NP}^{\text{RP}} \subseteq \text{MA}$: given an arithmetic circuit $C$, “guess” a circuit $\hat{C}$ of a smaller size and then check whether $C \equiv \hat{C}$. Moreover, the problem is, in fact, known to be one of a few natural MA problems. Unfortunately, similarly to the PIT problem, for the majority of the natural circuit classes (i.e. $C =$ multilinear circuits, multilinear formals, depth-3 circuits, etc.) not much is known on the $C$-testing problem. For example coming up with an efficient “Sparsity-testing” (i.e. determine whether a given circuit $C$ computes a sparse polynomial) is a long standing open question posed by von zur Gathen [68].
1.4 Objective and Results

In this work we study the problem of PIT for several restricted circuit classes, and its relation to other algebraic problems. In particular, we present efficient deterministic PIT algorithms for the following circuit classes:

- For sum of \( k \) read-once formulas we give \( n^{O(\log n+k)} \) and \( n^{O(k)} \) time black-box and non black-box PIT algorithms, respectively. Those are the first sub-exponential PIT algorithms for this circuit class.

- For depth-3 \( \Sigma\Pi\Sigma(k) \) circuits we give \( (nd)^{O(k^2 \log d)} \) time black-box PIT algorithm for circuits of degree \( d \) and \( n^{O(k)} \) time black-box PIT algorithm for multilinear circuits. \(^3\)

- For multilinear depth-4 \( \Sigma\Pi\Sigma\Pi(k) \) circuits we give \( n^{O(k)} \cdot s^{O(k^3)} \) time black-box PIT algorithm, when \( s \) is the size of the circuit. This is the first polynomial-time PIT algorithms for this circuit class.

As mentioned above, PIT is one of a few problems that have efficient randomized algorithms but lack of deterministic ones. Several results [28, 31, 18] suggest that, in some sense, PIT is the most general problem in this class. We initiated a study of the relation between PIT and other problems sharing that property. A natural domain of such problems is the domain of algebraic problems.

In this work we show a connection between PIT and polynomial factorization. Specifically, an efficient algorithm for polynomial factorization implies an efficient PIT algorithm. Conversely, an efficient PIT algorithm implies an efficient algorithm for “partial” factorization (for a more precise definition see Section 5.1.1). For circuit classes computing multilinear polynomials we obtained a complete (up to a polynomial factor) equivalence between the problems. This result holds in both black-box and non black-box settings.

We show a similar equivalence between PIT and reconstruction of read-once formulas. In other words, an efficient black-box PIT algorithm for read-once formulas implies an efficient reconstruction algorithm for read-once formulas and vice versa. As a result, we obtain the first sub-exponential

\(^3\)Recently, this result was subsumed by Saxena & Seshadhri [60] who gave a \( \text{poly}(n) \cdot d^{O(k)} \) time black-box PIT algorithm for this circuit class.
(in fact, quasi-polynomial) reconstruction algorithm for read-once formulas, by plugging-in our PIT algorithm.

Furthermore, we show that if we have an efficient PIT algorithm for a circuit class \( \mathcal{C} \), that satisfies some “nice” closure properties, then we can also solve the read-once testing (ROT) problem for that class efficiently. That is, given a circuit \( C \in \mathcal{C} \) determine whether it can be computed by a read-once formula.

### 1.5 Related Works

Following our line of work a progress has been made in the study of richer classes of multilinear circuits. Karnin et al. [36] gave the first sub-exponential (in fact, quasi-polynomial) PIT algorithm for multilinear depth-4 \( \Sigma \Pi \Sigma \Pi (k) \) circuits. Their result was in the black-box setting and applied several ideas and techniques from the current work. The black-box PIT algorithm for multilinear depth-4 \( \Sigma \Pi \Sigma \Pi (k) \) circuits, presented in the current work (Chapter 4) was partially inspired by [36], although it was attained via a different approach.

Jansen et. al. [29, 30] showed how to generalize the PIT algorithms for sums of read-once formula to get PIT algorithms, of roughly the same running time, for a slightly larger circuit class: sums of read-once Algebraic Branching Programs.

Another major step in this direction was made in the recent work of Anderson et al. [5]. In their work they extended the techniques presented in the current work (in particular, the hardness of representation approach), along with some techniques from [36], to handle multilinear read-\( k \) formulas.

Note that multilinear read-\( k \) formulas are a strictly stronger class than sums of \( k \) ROFs. Similarly to our work, their results yield polynomial and quasi-polynomial time PIT algorithms in the black-box and the non black-box settings, respectively, for constant \( k \). In fact, their results also hold for a somewhat broader model.

In a follow-up paper Gupta et al. [25] devised a randomized reconstruction algorithm for multilinear depth-4 \( \Sigma \Pi \Sigma \Pi (2) \) circuits.

\footnote{read-\( k \) formulas are arithmetic formulas in which each variable can appear at most \( k \) times.}
1.6 Organization

The work is organized as follows: In Chapter 2 we present basic notations, tools and techniques that will be used throughly in the work. We study the class of Read-Once formulas in Chapter 3. In the same chapter we give PIT algorithm for read-once formulas and related classes, as well as for depth-3 circuits. Next, in Chapter 4 we investigate depth-4 circuits to the end of presenting a PIT algorithm for that circuit class. The relation between PIT and other problems is given in Chapters 5 and 6. In Chapter 5 we demonstrate the relation between PIT and polynomial factorization. In Chapter 6 we give a new reconstruction algorithm for read-once formulas and show the relation between PIT and read-once testing.

1.7 Source

The work described in the thesis is based on the following publications:


Chapter 2

Basic Notations, Tools and Techniques

In this chapter we give the basic notation, tools and techniques common to the entire manuscript. In several occasions we strongly leverage on the connection between PIT, justifying assignments and generator (see Definitions below) to achieve our results.

2.1 Preliminaries

For a positive integer $n$ we denote $[n] = \{1, \ldots, n\}$. For a graph $G = (V, E)$ we denote with $G^c$ the complement graph. That is $G^c = (V, E')$ such that for $i \neq j \in V$ we have that $(i, j) \in E$ if and only if $(i, j) \notin E'$.

Let $F$ be a field and denote by $\overline{F}$ its algebraic closure. For a polynomial $P(x_1, \ldots, x_n)$, a variable $x_i$ and a field element $\alpha$ we denote by $P|_{x_i=\alpha}$ the polynomial resulting from setting $x_i = \alpha$. The following definitions are for polynomials $P, Q \in F[x_1, \ldots, x_n]$ and an assignment $\overline{a} \in F^n$. We say that $P$ depends on $x_i$ if there exist $\overline{a} \in F^n$ and $b \in F$ such that

$$P(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) \neq P(a_1, a_2, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n).$$

We denote $\text{var}(P) \triangleq \{x_i \mid P \text{ depends on } x_i\}$. Sometimes, we will abuse notations and consider the set $\text{var}(P)$ as a set of indices (i.e. $\{i \mid P \text{ depends on } x_i\}$) rather than variables. Intuitively, $P$ depends on $x_i$ if $x_i$ “appears” when $P$ is listed as a sum of monomials. We say that two polynomials are variable-
disjoint whenever they are defined on disjoint sets of variables. Given a subset $I \subseteq [n]$ and an assignment $\bar{a} \in \mathbb{F}^n$ we define $P|_{x_I=\bar{a}_I}$ to be the polynomial resulting from setting $x_i = a_i$ for every $i \in I$. In particular

$$\text{var}(P|_{x_I=\bar{a}_I}) \subseteq \{x_i \mid i \in [n] \setminus I\}.$$ 

We say that $P$ divides $Q$, or equivalently $Q$ is divisible by $P$, and denote it by $P \mid Q$ if there exists a polynomial $h \in \mathbb{F}[x_1, \ldots, x_n]$ such that $Q = P \cdot h$. Otherwise, we say that $P$ does not divide $Q$ (or $Q$ is not divisible by $P$) and denote it by $P \not\mid Q$. Given the notion of divisibility we define the gcd of a set of polynomials in the natural way.

We can conclude that by setting a variable of $P$ to some field element we, obviously, eliminate the dependence of $P$ on this variable, however we may also eliminate the dependence of $P$ on other variables and thus lose more information than intended. For the purposes of identity testing and reconstruction we cannot allow losing any information as it may affect our final answer. We now define a lossless type of an assignment. Similar definitions were given in [27] and [14].

**Definition 2.1.1** (Justifying assignment). Given an assignment $\bar{a} \in \mathbb{F}^n$ we say that $\bar{a}$ is a justifying assignment of $P$ if for each subset $I \subseteq \text{var}(P)$ we have that

$$\text{var}(P|_{x_I=\bar{a}_I}) = \text{var}(P) \setminus I. \quad (2.1)$$

We say that $\bar{a}$ is a weakly-justifying assignment of $P$ if the above it true for $|I| = 1$. We say that $\bar{a}$ is a common justifying assignment for a set of polynomials $\{P_m\}_{m \in [k]}$ if $\bar{a}$ is a justifying for each $P_m$.

Though this definition is very intuitive, ensuring that a given assignment $\bar{a}$ satisfies condition 2.1 for every $I \subseteq \text{var}(P)$ seems to be a computationally hard problem. The following observation provides an alternative and more useful definition:

**Observation 2.1.2.** An assignment $\bar{a} \in \mathbb{F}^n$ is a justifying assignment of $P$ if and only if condition 2.1 holds for every subset $I$ of size $|\text{var}(P)| - 1$.

We can, as well, define justification as a property of polynomials.

**Definition 2.1.3.** We say that a polynomial $P$ is $\bar{a}$-justified if $\bar{a}$ is a justifying assignment of $P$. Similarly we define the term weakly-$\bar{a}$-justified.
For convenience we will concentrate on $\bar{0}$-justified polynomials. The following lemma shows that in a sense we can do so w.l.o.g.

**Lemma 2.1.4.** Let $\bar{a} \in \mathbb{F}^n$ and let $P(\bar{x})$ be a (weakly) $\bar{a}$-justified polynomial. Then

$$P(\bar{x} + \bar{a}) \triangleq P(x_1 + a_1, x_2 + a_2, \ldots, x_n + a_n)$$

is a (weakly) $\bar{0}$-justified polynomial, in addition, $P(\bar{x} + \bar{a}) \equiv 0$ if and only if $P(\bar{x}) \equiv 0$.

**Definition 2.1.5.** We denote \(\text{var}_0(P) \triangleq \{x_i \mid P|_{x_i=1} = \bar{0}_{[n]\{i\}} \text{ depends on } x_i\}\).

Clearly, \(\text{var}_0(P) \subseteq \text{var}(P)\). We get equality only for $\bar{0}$-justified polynomials and vice versa.

**Lemma 2.1.6.** A polynomial $P$ is $\bar{0}$-justified iff \(\text{var}(P) = \text{var}_0(P)\).

### 2.1.1 Partial Derivatives

The concept of a partial derivative of a multivariate polynomial and its properties (for example: $P$ depends on $x_i$ if and only if $\frac{\partial P}{\partial x_i} \neq 0$) are well-known and well-studied for continuous domains (such as $\mathbb{R}$ and $\mathbb{C}$). Here we use a well-known variant for polynomials over arbitrary fields. Namely, the discrete partial derivatives. Discrete partial derivatives will play a major role in the analysis of our algorithms.

**Definition 2.1.7.** Let $P(\bar{x})$ be an $n$ variate polynomial over a field $\mathbb{F}$. We define the discrete partial derivative of $P(\bar{x})$ with respect to $x_i$ as $\frac{\partial P}{\partial x_i} = P|_{x_i=1} - P|_{x_i=0}$.

Notice that if $P$ is a multilinear polynomial then this definition coincides with the “analytical” one when $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. The following lemma is easy to verify and we will use it implicitly from now on.

**Lemma 2.1.8.** The following properties hold for any multilinear polynomial $P$.

- $P$ depends on $x_i$ if and only if $\frac{\partial P}{\partial x_i} \neq 0$.

- $\frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i} \left( \frac{\partial P}{\partial x_j} \right) = \frac{\partial^2 P}{\partial x_j \partial x_i}$.
• $\forall i \neq j \frac{\partial P}{\partial x_i}|_{x_j=a} = \frac{\partial}{\partial x_i} (P|_{x_j=a})$

• $\bar{a} \in \mathbb{F}^n$ is a justifying assignment for $P$ if and only if $\forall i \in \text{var}(P)$ it holds that $\frac{\partial P}{\partial x_i}(\bar{a}) \neq 0$.

The following lemma shows that when dealing with multilinear polynomials the usual properties of partial derivatives continue to hold for discrete partial derivatives as well.

**Lemma 2.1.9.** Let $P, G, Q$ be multilinear polynomials. Then the following derivation rules hold (with the appropriate implicit restrictions)

1. **Sum Rule.** If $Q = P + G$ then $\frac{\partial Q}{\partial x_i} = \frac{\partial P}{\partial x_i} + \frac{\partial G}{\partial x_i}$.

2. **Product Rule.** If $Q = P \cdot G$ then either $\frac{\partial P}{\partial x_i} \equiv 0$ or $\frac{\partial G}{\partial x_i} \equiv 0$ holds. Hence, $\frac{\partial Q}{\partial x_i} = \frac{\partial P}{\partial y} \cdot G + P \cdot \frac{\partial G}{\partial x_i}$.

3. **Chain Rule.** Let $Q(y, \bar{z})$ be a polynomial such that $P(\bar{x}, \bar{z}) \equiv Q(G(\bar{x}), \bar{z})$. Then $\frac{\partial P}{\partial x_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial G}{\partial x_i}$. Notice that since $Q$ is a multilinear polynomial, $\frac{\partial Q}{\partial y}$ does not depend on $y$.

Note that these properties do not hold for general polynomials. For example, when $P(x) = x^2 - x$ we get that $\frac{\partial P}{\partial x} = P|_{x=1} - P|_{x=0} \equiv 0$. Thus, in order to handle general polynomials we need the following extension.

**Definition 2.1.10.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. The directed partial derivative of $P$ w.r.t. $x_i$ and a direction $\alpha \in \mathbb{F}$ is defined as $\frac{\partial P}{\partial x_i} \alpha = P|_{x_i=\alpha} - P|_{x_i=0}$.

We can now define the notion of a witness.

**Definition 2.1.11.** We say that $0 \neq \alpha \in \mathbb{F}$ is a witness for $x_i$ in $P$ if $\frac{\partial P}{\partial x_i} \alpha \neq 0$ or $x_i \not\in \text{var}(P)$. The vector $\bar{\alpha} \in \mathbb{F}^n$ is a witness of $P$ if each $\alpha_i$ is a witness for $x_i$ in $P$.

The following proposition is immediate. It gives a sufficient condition for an assignment to be justifying for $P$.

**Proposition 2.1.12.** Let $\bar{\alpha} \in \mathbb{F}^n$ be a witness for $P$. Then (for each $i$) $P$ depends on $x_i$ if and only if $\frac{\partial P}{\partial x_i} (\bar{\alpha}) \neq 0$. Furthermore, let $\bar{a} \in \mathbb{F}^n$ be such that for every $x_i \in \text{var}(P)$ it holds that $\frac{\partial P}{\partial x_i}(\bar{a}) \neq 0$ (i.e. $\bar{a}$ is a nonzero of $\frac{\partial P}{\partial x_i}$). Then $\bar{a}$ is a justifying assignment of $P$. 

14
Proof. The first statement is just a restating of the definition of a witness. The proof of the second statement follows by observing that first computing \( \frac{\partial P}{\partial \alpha_i x_i} \) and then evaluating it on \( \bar{a} \) is the same as first setting \( x_j = a_j \) for all \( j \neq i \) and then computing the discrete derivative.

The next lemma shows that a sufficiently large field contains many witnesses.

**Lemma 2.1.13.** Let \( P(\bar{x}) \) be a polynomial with individual degrees bounded by \( d \) and let \( W \subseteq \mathbb{F} \) be a subset of size \( d + 1 \) (we assume that \( |\mathbb{F}| > d \)). Then \( W^n \) contains a witness for \( P \).

**Proof.** Note that for each variable we can find the witness separately as they are uncorrelated. Consider \( i \in [n] \) and define \( \varphi_i(\bar{x},w) \triangleq \frac{\partial P}{\partial x_i} \) (recall Definition 2.1.10). In this way we obtain a set of polynomials in the variables \( \bar{x} \) and \( w \) with individual degrees bounded by \( d \). Thus, \( \alpha \) is a witness for \( x_i \) in \( P \) if and only if \( \varphi_i(\bar{x},\alpha) \not\equiv 0 \) or \( x_i \notin \text{var}(P) \). If \( x_i \notin \text{var}(P) \) then any \( \alpha \in W \) is a witness. Otherwise, \( \varphi_i(\bar{x},w) \) is a nonzero polynomial of degree \( d \) in \( w \) and therefore \( W \) contains a nonzero assignment of it (see e.g. Lemma 2.1.16). We can repeat the same reasoning for every \( i \in [n] \).

**Definition 2.1.14.** For a non-empty subset \( I \subseteq [n] \), \( I = \{i_1, \ldots, i_{|I|}\} \), we define the iterated partial derivative with respect to \( I \) as \( \partial_I P \triangleq \frac{\partial^{(|I|)} P}{\partial x_{i_1} \partial x_{i_2} \partial x_{i_3} \cdots \partial x_{i_{|I|}}} \).

### 2.1.2 Polynomials and Circuit Classes

Let \( \mathcal{C} \) be a circuit class. We shall associate with it the polynomials that can be computed by its circuits. We make the following notations that will be later used.

**Definition 2.1.15.**

1. We say that the circuit class \( \mathcal{C} \) contains the polynomial \( P \) if \( P \) can be computed by some circuit \( C \) from \( \mathcal{C} \). We denote it by \( P \in \mathcal{C} \).

2. We say that the circuit class \( \mathcal{C}_1 \) contains the circuit class \( \mathcal{C}_2 \) if it contains all its polynomials (i.e. \( P \in \mathcal{C}_1 \implies P \in \mathcal{C}_2 \)). We denote it by \( \mathcal{C}_1 \subseteq \mathcal{C}_2 \).
3. For a circuit class $\mathcal{C}$ we define the (discrete) Directed Partial Derivatives of $\mathcal{C}$ as:
$$\partial \mathcal{C} \triangleq \left\{ \frac{\partial P}{\partial x_i} \mid P \in \mathcal{C}, i \in [n], \alpha \in \mathbb{F} \right\}.$$ 

4. A circuit class $\mathcal{C}$ is closed under partial derivatives if $\partial \mathcal{C} \subseteq \mathcal{C}$.

5. For a circuit class $\mathcal{C}$ we define the linear closure of $\mathcal{C}$ as:
$$\mathcal{L}(\mathcal{C}) \triangleq \{ \alpha \cdot P + \beta \mid P \in \mathcal{C}, \alpha, \beta \in \mathbb{F} \}.$$ 

2.1.3 Some Useful Facts about Polynomials

We conclude this section with two well-known facts concerning polynomials.

**Lemma 2.1.16.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. Suppose that for every $i \in [n]$ the individual degree of $x_i$ is bounded by $d_i$, and let $S_i \subseteq \mathbb{F}$ be such that $|S_i| > d_i$. Then, for $S = S_1 \times S_2 \times \cdots \times S_n$ it holds that $P \equiv 0$ iff $P|_S \equiv 0$.

A proof can be found in [4].

**Lemma 2.1.17 (Gauss).** Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n, y]$ be a nonzero polynomial and $g \in \mathbb{F}[x_1, \ldots, x_n]$ such that $P|_{y=g(\bar{x})} \equiv 0$ then $y - g(\bar{x})$ is an irreducible factor of $P$ in the ring $\mathbb{F}[x_1, x_2, \ldots, x_n, y]$.

2.2 Generators and Hitting Sets for Arithmetic Circuits

In this section, we formally define the notion of a generator for a circuit class, describe a few of their useful properties and give the connection to hitting sets. Intuitively, a generator $\mathcal{G}$ for a circuit class $\mathcal{C}$, is a function that stretches $t$ independent variables into $n \gg t$ dependent variables that can be plugged into any polynomial $P \in \mathcal{C}$ without causing it to vanish. Recall that a hitting set $\mathcal{H} \subseteq \mathbb{F}^n$ for a circuit class $\mathcal{C}$ is a set such that for any nonzero polynomial $P \in \mathcal{C}$, there exists $\bar{a} \in \mathcal{H}$, such that $P(\bar{a}) \neq 0$. In identity testing, generators and hitting sets play the same role. Given a generator one can easily construct a hitting set by evaluating the generator on a large enough set of points. Conversely, given a hitting set $\mathcal{H}$ it is easy to construct a generator by taking a low degree curve through $\mathcal{H}$.
Definition 2.2.1. A polynomial mapping \( \mathcal{G} = (\mathcal{G}^1, \ldots, \mathcal{G}^n) : \mathbb{F}^t \rightarrow \mathbb{F}^n \) is a generator for the circuit class \( \mathcal{C} \) if for every nonzero \( n \)-variate polynomial \( P \) computed by \( \mathcal{C} \) it holds that \( P(\mathcal{G}) \not\equiv 0 \).

In other words, the polynomial, resulting from setting \( x_i = \mathcal{G}^i \) for every \( i \in [n] \), is a nonzero polynomial. A generator can also be viewed as a map that contains a hitting set for \( \mathcal{C} \) in its image. That is, for every nonzero \( P \in \mathcal{C} \) there exists \( \bar{a} \in \text{Im}(\mathcal{G}) \) such that \( P(\bar{a}) \neq 0 \) (where \( \text{Im}(\mathcal{G}) \triangleq \mathcal{G}(\mathbb{F}^t) \)). All our black-box PIT algorithms are, in fact, generators for some (relatively) small \( t \). The following is an immediate and an important property of generators.

Observation 2.2.2. Let \( P = P_1 \cdot P_2 \cdot \ldots \cdot P_k \) be a product of nonzero polynomials \( P_i \in \mathcal{C} \) and let \( \mathcal{G} \) be a generator for \( \mathcal{C} \). Then \( P(\mathcal{G}) \not\equiv 0 \).

We now describe an efficient way for constructing a generator for a circuit class \( \mathcal{C} \) from a hitting set \( \mathcal{H} \) for \( \mathcal{C} \). The construction is performed by passing a low degree curve through \( \mathcal{H} \) using polynomial interpolation: Choose an arbitrary subset \( V \subseteq \mathbb{F} \) of size \( n \) and set \( t \triangleq \lceil \log_n |\mathcal{H}| \rceil \). Clearly, \( |\mathcal{H}| < n^t \). Denote \( \mathcal{H} = \{ \bar{a}^1, \bar{a}^2, \ldots, \bar{a}^{|\mathcal{H}|} \} \) where \( \bar{a}^i = (a^i_1, a^i_2, \ldots, a^i_n) \). Let \( \varphi : V^t \rightarrow \{1, 2, \ldots, |\mathcal{H}|\} \subseteq \mathbb{N} \) be some surjection. We define the functions \( h_i(\bar{y}) : \mathbb{F}^t \rightarrow \mathbb{F} \) to be the interpolation polynomial of the \( i \)-th coordinates of the vectors in \( \mathcal{H} \). That is, \( h_i(\bar{y}) \) is a \( t \)-variate polynomial, of degree at most \( n - 1 \) in each variable, such that for every \( \bar{b} \in V^t \) we have that \( h_i(\bar{b}) = a_i^{\varphi(\bar{b})} \).

Finally, let \( h(\bar{y}) : \mathbb{F}^t \rightarrow \mathbb{F}^n \) be defined as \( h(\bar{y}) \triangleq (h_1(\bar{y}), h_2(\bar{y}), \ldots, h_n(\bar{y})) \). From the construction it is clear that \( \mathcal{H} \subseteq \text{Im}(h) \). We thus get that \( h \) is a generator for \( \mathcal{C} \).

Lemma 2.2.3. The procedure described above in time \( \text{poly}(n, |\mathcal{H}|) \) constructs a map \( h(\bar{y}) : \mathbb{F}^t \rightarrow \mathbb{F}^n \) with individual degrees bounded by \( n - 1 \), which is a generator for the circuit class \( \mathcal{C} \).

Proof. Let \( P \in \mathcal{C} \) be a nonzero polynomial. From the definition of \( \mathcal{H} \) there exists \( \bar{a} \in \mathcal{H} \) such that \( P(\bar{a}) \neq 0 \). As \( \mathcal{H} \subseteq \text{Im}(h) \) it follows that \( \bar{a} \in \text{Im}(h) \) and consequently \( P(h(\bar{y})) \neq 0 \). The claim regarding the degree follows form the construction of \( h_i \)-s. In addition, note that the \( h_i \)-s can be computed in time polynomial in \( |\mathcal{H}| \) using simple interpolation.

This Lemma was used in [36] to establish a generator for multilinear sparse polynomials.
Lemma 2.2.4 (Lemma 2.7 in [36]). For every $m \geq 1$ there exists a generator $S_m \triangleq (S^1_m, S^2_m, \ldots, S^n_m) : \mathbb{F}^t \to \mathbb{F}^n$ for $m$-sparse multilinear polynomials where the individual degree of each $S^i_m$ is bounded by $n-1$ and $q(n,m) = O(\log n^m)$.

We now describe the obvious way of obtaining a hitting set from a generator.

Lemma 2.2.5. Let $G = (G^1, \ldots, G^n) : \mathbb{F}^t \to \mathbb{F}^n$ be a generator for a circuit class $C$ such that the individual degrees of the $G^i$-s are bounded by $\Delta$. Let $W \subseteq \mathbb{F}$ be of size $nd\Delta$. Then, $H \triangleq G(W)$ is a hitting set, of size $|H| \leq (nd\Delta)^t$, for polynomials $P \in C$ of individual degrees at most $d$.

Proof. Let $P \in C$ be a nonzero polynomial with individual degrees at most $d$. By definition, $P(G)$ is a nonzero $t$-variate polynomial with individual degrees bounded by $nd\Delta$. Lemma 2.1.16 implies that $P(G)|_W \not\equiv 0$. Equivalently, $P|_H \not\equiv 0$. Finally, note that $|H| \leq |W|^t \leq (nd\Delta)^t$.

2.2.1 The Generator $G_k$

In this section we define a map that will be one of the main ingredients in our PIT algorithms. We start with some notations. The Hamming weight of a vector $\bar{a} \in \mathbb{F}^n$ is defined as: $w_H(\bar{a}) \triangleq \{|i| \mid a_i \neq 0\}$, that is, the number of its nonzero coordinates. For a set $0 \subseteq W \subseteq \mathbb{F}$ and $k \leq t$ we define $A^t_k(W)$ to be the set of all vectors in $W^t$ with Hamming weight at most $k$, i.e. the set of vectors that have at most $k$ nonzero coordinates. Formally:

$$A^t_k(W) \triangleq \{ \bar{a} \in W^t \mid w_H(\bar{a}) \leq k \}.$$ 

It is easy to see that $|A^t_k(W)| = \sum_{j=0}^{k} \binom{t}{j} (|W| - 1)^j = (t \cdot |W|)^{O(k)}$. Throughout the entire paper we fix a set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}$ of $n$ distinct elements. Recall that we are allowed to assume that $|\mathbb{F}| > n$ as we are allowed to use elements from an appropriate extension field.

Definition 2.2.6. For every $i \in [n]$ let $u_i(w) : \mathbb{F} \to \mathbb{F}$ be the $i$-th Lagrange Interpolation polynomial for the set $A$. Namely, $u_i(w)$ is a degree $n-1$ polynomial satisfying: $u_i(\alpha_j) = 1$ when $j = i$ and $u_i(\alpha_j) = 0$ when $j \neq i$. For every $i \in [n]$ and $k \geq 1$ we define $G^i_k(y_1, \ldots, y_k, z_1, \ldots, z_k) : \mathbb{F}^{2k} \to \mathbb{F}$ as

$$G^i_k(y_1, \ldots, y_k, z_1, \ldots, z_k) \triangleq \sum_{j=1}^{k} u_i(y_j) \cdot z_j.$$
Finally, let $G_k(y_1, \ldots, y_k, z_1, \ldots, z_k) : \mathbb{F}^{2k} \to \mathbb{F}^n$ be defined as

$$G_k(y_1, \ldots, y_k, z_1, \ldots, z_k) \triangleq \left( G_k^1, G_k^2, \ldots, G_k^n \right) = 
\left( \sum_{j=1}^{k} u_1(y_j) \cdot z_j, \sum_{j=1}^{k} u_2(y_j) \cdot z_j, \ldots, \sum_{j=1}^{k} u_n(y_j) \cdot z_j \right).$$

In other words, each coordinate $i$ is a sum of $k$ distinct copies of the $u_i(y) \cdot z$. We also define $G_0^i \equiv 0$ for every $i \in [n]$. The following simple observation plays an important role in our algorithms.

**Observation 2.2.7.** Denote with $\bar{e}_i \in \{0, 1\}^n$ the vector that has 1 in the $i$-th coordinate and 0 elsewhere. Let $k \geq 0$. Then, $G_{k+1} = G_k + \sum_{i=1}^{n} u_i(y_{k+1}) \cdot z_{k+1} \cdot \bar{e}_i$. Hence, for every $\alpha_m \in A$ we have that $G_{k+1}|_{y_{k+1} = \alpha_m} = G_k + z_{k+1} \cdot \bar{e}_m$. Hence, for every $W \subseteq \mathbb{F}$ it holds that $A_k^n(W) \subseteq \text{Im}(G_k)$.

### 2.3 From PIT to Common Justifying Assignments

In this section we give an algorithm (Algorithm 1) that, using a PIT algorithm for a circuit class $C'$ such that $\partial C \subseteq C'$ (recall Definition 2.1.15), efficiently and deterministically finds a common justifying assignment for a set of polynomials from $C$.

Before presenting the algorithm we explain the intuition behind it. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial with individual degrees bounded by $d$. What we are after is a vector $(a_1, \ldots, a_n) \in \mathbb{F}^n$ such that if $P$ depends on $x_i$ then the polynomial $P(a_1, \ldots, a_{i-1}, x_i, a_{i+1}, \ldots, a_n)$ depends on $x_i$ as well. Our approach will be to consider a witness for $P$, $\bar{a} \in \mathbb{F}^n$, and look for $\bar{a} \in \mathbb{F}^n$ such that if $\frac{\partial P}{\partial a_i} \neq 0$ then also $\frac{\partial P}{\partial a_i}(\bar{a}) \neq 0$ (see Proposition 2.1.12). For this purpose we consider the polynomials \( \left\{ g_i = \frac{\partial P}{\partial a_i}, x_i \right\}_{i \in [n]} \) and look for a vector which is their common nonzero. We do so in a manner similar to finding a satisfying assignment to a CNF formula given a SAT oracle, i.e. variable by variable. In each step $1 \leq j \leq n$ we find $a_j$ such that for any $i \neq j$ it holds $g_i|_{x_j = a_j} \neq 0$. As the degree of each $x_j$ in each $g_i$ is bounded by $d$, there are at most $d$ ‘bad’ possible values for
Namely, for the majority of the values of \(a_j\), we have that \(g_i|_{x_j=a_j} \neq 0\). Hence, if we check enough values then we should find some \(a_j\) that is good for all \(g_i\)'s. To verify that \(g_i|_{x_j=a_j} \neq 0\) we use the PIT algorithm for \(C'\). Once we found \(a_j\) we set \(x_j = a_j\) to get a new set of polynomials \(g_i\)'s and continue to step \(j + 1\). We now give the algorithm and its analysis.

Algorithm 1 Find Common Justifying Assignment

**Input:** Circuits \(C_1, \ldots, C_k\) from \(C\) computing \(P_1, \ldots, P_k\) with individual degrees bounded by \(d\),

A subset \(V \subseteq \mathbb{F}\) be of size \(|V| = knd\),

Access to a PIT algorithm for \(C'\) such that \(\partial C \subseteq C'\).

**Output:** A Common Justifying Assignment \(\bar{a}\) for \(P_1, P_2, \ldots, P_k\)

1. Find \(\{\bar{\alpha}_m\}_{m \in [k]}\) such that \(\bar{\alpha}_m\) is a witness for \(P_m\) (see Lemma 2.3.1)
2. For \(i \in [n], m \in [k]\) set \(g_i^m = \frac{\partial P_m}{\partial a_m^i x_i}\)

We describe an iteration \(j \in [n]\) for finding the value of \(a_j\) in \(\bar{a}\):

3. for \(j = 1 \ldots n\) do
4. Find \(c_j \in V\) such that for every \(m \in [k]\) and \(i \neq j \in [n]\); if \(g_i^m \neq 0\) then \(g_i^m|_{x_j = c_j} \neq 0\).
5. For every \(m \in [k]\) and \(i \neq j \in [n]\), set \(g_i^m \leftarrow g_i^m|_{x_j = c_j}\).
6. Set \(a_j \leftarrow c_j\)

**Lemma 2.3.1.** Let \(\mathbb{F}\) be a field of size \(|\mathbb{F}| > d\). Let \(P\) be a polynomial, with individual degrees bounded by \(d\), computed by a circuit class \(C\) over \(\mathbb{F}\). Let \(C'\) be a circuit class such that \(\partial C \subseteq C'\). Then there is an algorithm that when given access to \(P\) (either explicitly or via black-box, depending on the PIT algorithm for \(C'\)) computes \(\text{var}(P)\) and outputs a witness \(\bar{\alpha}\) for \(P\) in time \(O(nd \cdot T_{C'})\), where \(T_{C'}\) is the running time of the PIT algorithm for \(C'\).

**Proof.** The proof of Lemma 2.1.13 gives a simple algorithm for finding witnesses. Indeed, consider \(\varphi_i(\bar{x}, w) \triangleq \frac{\partial P}{\partial a_i x_i}\). Clearly \(\varphi_i \in C'\). Note that if \(x_i \in \text{var}(P)\) then \(\varphi_i(\bar{x}, w) \neq 0\). Pick \(d + 1\) different elements \(v_0, \ldots, v_d \in \mathbb{F}\) and for each of them check, using the PIT algorithm for \(C'\), whether \(\varphi_i(\bar{x}, v_i) \neq 0\). Let \(\alpha_i\) be the first \(v_j\) for which \(\varphi_i(\bar{x}, v_j) \neq 0\) (if no such \(j\) exists then take \(\alpha_i = 0\)). Set \(\bar{\alpha} = (\alpha_1, \ldots, \alpha_n)\). Lemma 2.1.13 implies that \(\bar{\alpha}\) is the required witness. The claim regarding the running time is clear.
Note that the same approach also determines, for each variable $x_i$, whether the polynomial depends on $x_i$ or not. Therefore, the algorithm can be used to compute $\text{var}(P)$ as well.

We now give the analysis of the second step of the algorithm.

**Lemma 2.3.2.** Let $F$ be a field of size $|F| > knd$ and let $V \subseteq F$ be of size $|V| = knd$. Let $\{P_m\}_{m \in \mathbb{K}}$ be a set of polynomials with individual degrees bounded by $d$ that are computed by circuits from $C$. Let $C'$ be a circuit class such that $\partial C \subseteq C'$. Then, Algorithm 1 returns a common justifying assignment $\bar{a}$ for $\{P_m\}_{m \in \mathbb{K}}$ in time $O(n^3k^2d \cdot T_{C'})$, where $T_{C'}$ is the running time of the PIT algorithm for $C'$.

**Proof.** We show that each iteration $j \in [n]$ succeeds, and that the algorithm outputs a justifying assignment. In order to succeed in $j$-th step the algorithm must find $c_j \in V$ that is good for every $g_i^m \neq 0$. Namely, for every $m$ and $i \neq j$ if $g_i^m \neq 0$ then $g_i^m |_{x_j=c_j} \neq 0$. Note, that $g_i^m$-s are polynomials with individual degrees bounded by $d$ and hence, by Lemma 2.1.17, each $g_i^m$ has at most $d$ roots of the form $x_j = c_j$. Therefore, there are at most $kd(n-1)$ ‘bad’ values of $c_j$ (i.e. values for which there exist $m$ and $i \neq j$ with $g_i^m \neq 0$ and $g_i^m |_{x_j=c_j} \equiv 0$). Consequently, $V$ contains at least one ‘good’ $c_j$. From the definition $g_i^m \in C'$, therefore we can use the supplied PIT algorithm for $C'$ to find such $c_j$. Notice that before the first iteration $g_i^m \neq 0$ iff $x_i \in \text{var}(P_m)$. In addition, for each $i, j \in [n]$ if $g_i^m$ is nonzero before the $j$-iteration then it remains nonzero after that iteration. We conclude that after the $n$-th iteration is (successfully) completed we have that for every $m \in \mathbb{K}$ and $x_i \in \text{var}(P_m)$ it holds that $\frac{\partial P_m}{\partial x_i}(\bar{a}) = g_i^m \neq 0$. This follows from the definition of the $g_i^m$-s and the fact that in each iteration we set $x_j = a_j$ for every $g_i^m$. Thus, by Proposition 2.1.12 $\bar{a}$ is indeed a common justifying assignment.

Next, we analyze the running time. By Lemma 2.3.1 finding $\{\bar{a}^m\}_{m \in \mathbb{K}}$ requires $O(knd)$ PIT checks. The computation of $\{g_i^m\}_{i \in [n], m \in \mathbb{K}}$ can be done in $O(nk)$ time. The execution of each iteration $j$ requires for each $c \in V$ to perform $k(n-1)$ PIT checks, thus in every iteration we preform at most $k(n-1) \cdot |V| < n^2k^2d$ PIT checks. Therefore, we do at most $O(n^3k^2d + knd)$ PIT checks during the execution. Hence the total running time of the algorithm is $O(n^3k^2d \cdot T_{C'})$, where $T_{C'}$ is the cost of every PIT check for a circuit in $C'$.

$\square$

21
2.3.1 From a Generator to a Justifying Set

Algorithm 1 shows how to find a common justifying assignment for a set of polynomials in an adaptive manner, even if the PIT for $C'$ is in the black-box setting. In this section we give a non-adaptive version of the algorithm. More precisely, given a generator (a black-box PIT algorithm) $G$ for $C'$ (satisfying $\partial C \subseteq C'$) we construct a $(k,d)$-justifying set for $C$. Namely, a set of elements $J_{k,d} \subseteq \mathbb{F}^n$ that contains a common justifying assignment of any set of $k$ polynomials with individual degrees bounded by $d$, that are computed by $C$. The construction is performed by evaluating the generator on many points (assuming that $\mathbb{F}$ is large enough). In particular, we show that $\text{Im}(G)$ contains a common justifying assignment for any set of polynomials computed by $C$.

Lemma 2.3.3. Let $\{P_m(\bar{x})\}_{m \in [k]}$ be a set of $k$ polynomials over $\mathbb{F}[x_1, \ldots, x_n]$ computed by circuits from $C$ with individual degrees bounded by $d$. Let $C'$ be (another) circuit class such that $\partial C \subseteq C'$. Let $G = (G^1, \ldots, G^n) : \mathbb{F}^t \rightarrow \mathbb{F}^n$ be a generator for $C'$ such that the individual degrees in each $G^i$ are bounded by $\Delta$. Let $V \subseteq \mathbb{F}$ be of size $|V| = 2kn^2d\Delta$. Then $J_{k,d} \overset{\Delta}{=} G(V^t)$ contains a common justifying assignment for $P_1, \ldots, P_k$ (that is, $J_{k,d}$ is a $(k,d)$-justifying set for $C$).

Proof. By Lemma 2.1.13 $\mathbb{F}^n$ contains witnesses $\{\bar{\alpha}^m\}_{m \in [k]}$ for $\{P_m(\bar{x})\}_{m \in [k]}$. For $i \in [n]$ and $m \in [k]$ define $g_i^m \overset{\Delta}{=} \frac{\partial P_m}{\partial x_i^m}(\bar{G})$. From the definition of $C'$ and the generator, we get that if $\frac{\partial P_m}{\partial x_i^m}(\bar{G}) \neq 0$ then $g_i^m \neq 0$. Consider the polynomial $g \overset{\Delta}{=} \prod_{i,m \mid g_i^m \neq 0} g_i^m$. It follows that $g$ is a nonzero $t$-variate polynomial of degree at most $nk \cdot nd\Delta$ in each variable. Lemma 2.1.16 implies that $g|_{V^t} \neq 0$. Equivalently, there exists $\bar{\gamma} \in V^t$ such that for any $i \in [n]$ and $m \in [k]$, if $g_i^m \neq 0$ then $g_i^m(\bar{\gamma}) \neq 0$. Now, let $i \in [n]$ and $m \in [k]$ be such that $x_i \in \text{var}(P_m)$. Then $\frac{\partial P_m}{\partial x_i^m}(\bar{G}) \neq 0$ and thus $g_i^m \overset{\Delta}{=} \frac{\partial P_m}{\partial x_i^m}(\bar{G}) \neq 0$. From the choice of $\bar{\gamma}$ we obtain that $\frac{\partial P_m}{\partial x_i^m}(\bar{G}(\bar{\gamma})) = g_i^m(\bar{\gamma}) \neq 0$ and hence $\bar{a} = G(\bar{\gamma})$ is a justifying assignment of every $P_m$ (recall Proposition 2.1.12). Finally, note that $|J_{k,d}| \leq (2kn^2d\Delta)^t$.

Note that we do not need to know the $\bar{\alpha}^m$s to guarantee that $J_{k,d}$ has
the required properties, but rather just know that such $\bar{\alpha}^m$-s exist. The following is an immediate corollary of the proof.

**Corollary 2.3.4.** Let $C, C'$ and $G$ be as in Lemma 2.3.3 and let $k \geq 1$. Then $\text{Im}(G)$ contains a common justifying assignment for any set of $k$ polynomials computed by $C$.

### 2.4 Black-box PIT Requires Large (extension) Fields

In this section we show that “small” hitting sets require access to “large” (extension) fields.

**Lemma 2.4.1.** Let $\mathbb{F}$ be a field of size $q$. For every $\bar{a} \in \mathbb{F}^n$ define $P_{\bar{a}}(\bar{x}) \triangleq \prod_{i=1}^{n} (x_i - a_i)$. Let $\mathcal{H} \subseteq \mathbb{F}^n$ be a hitting set for the circuit class $C = \{ P_{\bar{a}}(\bar{x}) \}_{\bar{a} \in \mathbb{F}^n}$. Then $|\mathcal{H}| = 2^{\Omega(n/q)}$.

**Proof.** By the definition of $\mathcal{H}$, each $P_{\bar{a}}$ must be hit by at least one point $\bar{b} \in \mathcal{H}$, that is $P_{\bar{a}}(\bar{b}) \neq 0$. Observe that $P_{\bar{a}}(\bar{b}) \neq 0$ iff for each $i \in [n]$ it holds that $a_i \neq b_i$. Consequently, every $\bar{b}$ hits exactly $(q - 1)^n$ functions $P_{\bar{a}}$ and thus $\mathcal{H}$ must contain at least $\frac{q^n}{(q-1)^n} \geq \exp(n/q)$ points.

Indeed, we cannot hope for sub-exponential size hitting sets if we restrict ourselves to constant-size fields. Furthermore, if we wish to attain polynomial size hitting sets then we must assume/allow access to an extension field of size $\Omega(n/\log n)$.
Chapter 3

PIT for Read-Once Formulas and related circuit classes

An arithmetic read-once formula (ROF for short) is a formula (a circuit whose underlying graph is a tree) in which the operations are $\{+, \times\}$ and such that every input variable labels at most one leaf. A preprocessed ROF (PROF for short) is a ROF in which we are allowed to replace each variable $x_i$ with a univariate polynomial $T_i(x_i)$. In this chapter we study the problems of designing deterministic identity testing algorithms for models related to preprocessed ROFs. Our main result gives PIT algorithms for the sum of $k$ preprocessed ROFs, of individual degrees at most $d$ (i.e. each $T_i(x_i)$ is of degree at most $d$), that run in time $(nd)^{O(k)}$ in the non black-box setting and in time $(nd)^{O(k+\log n)}$ in the black-box setting. We also obtain better algorithms when the formulas have a small depth that lead to an improvement on the best PIT algorithm for multilinear depth-3 $\Sigma\Pi\Sigma(k)$ circuits.

Our main technique is to prove a hardness of representation result, namely, a theorem showing a relatively mild lower bound on the sum of $k$ PROFs. We then use this lower bound in order to design our PIT algorithm.

The results of this chapter are based on the works [64, 65].
3.1 Introduction

In this chapter we study the polynomial identity testing problem for several models based on read-once formulas, which form a a restricted model of multilinear formulas. An arithmetic read-once formula (ROF for short) is a formula (a circuit in which the fan-out of every gate is at most 1) in which the operations are \{+, \times\} and such that every input variable labels at most one leaf. Read-once formulas can be thought of as the simplest form of multilinear formulas. Although ROFs form a very restricted model of computation they have received a lot of attention both in the Boolean world \([35, 6, 15]\) and in the algebraic world \([27, 14, 12, 13]\). However, no deterministic sub-exponential time black-box PIT algorithm for arithmetic ROF was known prior to this work. We give the first sub-exponential (in fact, quasi-polynomial) time deterministic PIT algorithms for (sums of) read-once arithmetic formulas in the black-box and non black-box settings. Besides being a relaxation of the general model of multilinear formulas, another motivation for our work is to better understand recent results on depth-3 circuits. It is not difficult to see that a multilinear depth-3 \(\Sigma\Pi\Sigma(k)\) circuit is a sum of \(k\) read-once formulas of a very restricted form (i.e. each multiplication gate is a ROF). Thus, our work can be seen as a (significant) generalization and extension of previous results for multilinear \(\Sigma\Pi\Sigma(k)\) circuit \([17, 41, 37, 58]\) (although, the focus of those results was not on the multilinear case).

3.1.1 Our Results and Techniques

Our black-box PIT algorithms use the notion of generators. A generator for a circuit class \(C\) is a mapping \(G: \mathbb{F}^t \rightarrow \mathbb{F}^n\), such that for any nonzero polynomial \(P\), computed by a circuit from \(C\), it holds that the composition \(P(G)\) is nonzero as well. By considering the image of \(G\) on \(W^t\), where \(W \subseteq \mathbb{F}\) is of polynomial size, we obtain a hitting set for \(C\) (more details given in Section 2.2). We now state our results. We begin with our main results: PIT algorithms for the sum of \(k\) PROFs.

**Theorem 1.** Given a black-box access to \(\Phi = \Phi_1 + \ldots + \Phi_k\), where the \(\Phi_i\)-s are preprocessed read-once formulas in \(n\) variables, with individual degrees at most \(d\), there is a deterministic algorithm that checks whether \(\Phi \equiv 0\).
The running time of the algorithm is \((nd)^{O(\log n + k)}\).

In fact, we design a generator \(G : \mathbb{F}^{O(\log n + k)} \to \mathbb{F}^n\), for sum of \(k\) PROFs. When the formulas are given to us explicitly we can achieve a more efficient PIT algorithm.

**Theorem 2.** Given \(k\) preprocessed read-once formulas in \(n\) variables, \(\Phi_1, \ldots, \Phi_k\) with individual degrees at most \(d\), there is a deterministic algorithm that checks whether \(\Phi_1 + \ldots + \Phi_k \equiv 0\). The running time of the algorithm is \((nd)^{O(k)}\).

In fact, our main results are obtained by reducing the problem from the case of general \(k\) to the case \(k = 1\). This reduction is carried out via hardnes of representation approach. This technique enables us to transform a mild lower bound for a very structured polynomial into a PIT algorithm for sum of (preprocessed) ROFs.

For the purpose of the reduction, we first design a PIT algorithm for a single PROF. Note that when a PROF is (explicitly) given to us, we can determine whether it computes the zero polynomial in time \(O(n)\) by a simple traversal over the formula. For the black-box case we prove the following theorem.

**Theorem 3.** Given black-box access to a preprocessed read-once formula \(\Phi\) in \(n\) variables, with individual degrees at most \(d\), there is a deterministic algorithm that checks whether \(\Phi \equiv 0\). The running time of the algorithm is \((nd)^{O(\log n)}\).

The intuition for the proof is that if a PROF is not zero then it can be written as combination of two smaller PROFs, such that one of them depends on at most \(n/2\) variables. Using this observation we design a generator \(G : \mathbb{F}^{O(\log n)} \to \mathbb{F}^n\) such that \(\Phi(G) \neq 0\). Applying the same techniques we design a different generator for the sum of PROFs of a small depth (see Definition 3.3.4), which yields a better running time PIT algorithm.

**Theorem 4.** Given a black-box access to \(\Phi = \Phi_1 + \ldots + \Phi_k\), where the \(\Phi_i\)-s are preprocessed read-once formulas in \(n\) variables of depth at most \(D\), with individual degrees at most \(d\), there is a deterministic algorithm that checks whether \(\Phi \equiv 0\). The running time of the algorithm is \((nd)^{O(D+k)}\).
As a corollary we obtain an \( n^{O(k)} \) time PIT algorithm for multilinear \( \Sigma \Pi \Sigma(k) \) circuits (a multilinear \( \Sigma \Pi \Sigma(k) \) circuit can be considered as a sum of \( k \) ROFs of depth-2), which gives the best known running-time algorithm for this circuit class. Moreover, our algorithm also holds for the more general case of preprocessed multilinear \( \Sigma \Pi \Sigma(k) \) circuits (see Section 3.5 for the definition).

**Theorem 5.** Let \( C \) be a preprocessed multilinear \( \Sigma \Pi \Sigma(k) \) circuit with individual degrees bounded by \( d \). Then there is a deterministic black-box PIT algorithm for \( C \) that runs in time \( (nd)^{O(k)} \).

We note that, unlike most of previous works on black-box \( \Sigma \Pi \Sigma(k) \) circuits [17, 37, 40, 58, 59], this result does not rely on bounds on the rank of zero \( \Sigma \Pi \Sigma(k) \) circuits (see Section 3.5). In addition to the multilinear case, we obtain a new PIT algorithm (by constructing an appropriate generator) for general \( \Sigma \Pi \Sigma(k) \) circuits that has (roughly) the same running time as the algorithm obtained from the results of [37, 59].

**Theorem 6.** Let \( C \) be a \( \Sigma \Pi \Sigma(k) \) circuit, of degree \( d \), over \( \mathbb{F} \). There is a deterministic black-box PIT algorithm for \( C \) that runs in time \( (nd)^{O(k^2 \log d)} \). In the case when \( \mathbb{F} = \mathbb{R} \) or \( \mathbb{F} = \mathbb{Q} \) the the running time is \( (nd)^{O(k^2)} \).

Finally, we show that it is possible to generalize the previous theorems to the case where we have a read-\( r \) sum of PROFs. That is, every variable appears in at most \( r \) of the formulas (see Definition 3.4.8).

**Theorem 7.** Let \( \Phi = \Phi_1 + \ldots + \Phi_k \) be a read-\( r \) sum of PROFs. Then there is an \( (nd)^{O(\log n + r)} \) time deterministic black-box PIT algorithm for \( \Phi \). If in addition we are guaranteed that the \( \Phi_i \)-s are depth-\( D \) PROFs then there is an \( (nd)^{O(D + r)} \) time deterministic black-box PIT algorithm for \( \Phi \). In the non black-box setting there is an \( (nd)^{O(r)} \) deterministic PIT algorithm for \( \Phi \).

### 3.1.2 Proof Technique

The high level idea behind the algorithms for sum of ROFs is the following. We first consider the case of a single ROF. Instead of constructing a hitting

---

1 Recently, [60] obtained an improved algorithm that has a better running time than all previous algorithms. For the multilinear case the running time of the obtained algorithm matches the one given in the current paper.
set we will construct a generator. Intuitively, a generator is a polynomial in a few variables of a not too high degree such that composing it with any nonzero ROF results in a nonzero polynomial. The basic observations that leads to the construction of the generator is that every ROF is a sum or a product of two smaller ROFs that are defined over disjoint sets of variables. Our generator will have the property that it will be able to leave \( \log n \) variables, of our choice, “alive” while still behaving like a “generator” on the other variables (see Definition 2.2.6 and Observation 2.2.7).

The main technical difficulty is turning a PIT algorithm for a single ROF into a PIT algorithm for a sum of ROFs. For this we use a technique we call hardness of representation. The basic idea is the following. We first prove that if \( F_1, \ldots, F_k \) are ROFs that have a certain technical property (i.e. they are all weakly-\( \bar{0} \)-justified, see Definition 2.1.1) then \( F_1 + \ldots + F_k = \prod_{i=1}^{n} x_i \) implies that \( k \geq n/3 \). From this it follows that if \( k < n/3 \) and \( F_1, \ldots, F_k \) are all weakly-\( \bar{0} \)-justified such that \( F_1 + \ldots + F_k \neq 0 \) then there must exist \( x_i \) such that \( (F_1 + \ldots + F_k)_{x_i=0} \neq 0 \). Using this we show that the set of all vectors that have at most \( k \) nonzero coordinates is a hitting set for such sums of ROFs. The problem, of course, is that in the general case we are not guaranteed to be given weakly-\( \bar{0} \)-justified ROFs. To overcome this we show how, using the PIT for a single ROF, we can find a relatively small set of inputs \( J_{k,d} \) such that for some \( \bar{a} \in J_{k,d} \) the formulas \( F_1(\bar{x} + \bar{a}), \ldots, F_k(\bar{x} + \bar{a}) \) are all weakly-\( \bar{0} \)-justified (see Lemma 2.3.3).

We believe that the general intuition behind the technique may be useful elsewhere: First, one needs to find a polynomial that has “nice” irreducible factors but that is still “hard” to compute. E.g., in our case the polynomial was \( \prod_{i=1}^{n} x_i \) and we could not represent it as a sum of a few (weakly-\( \bar{0} \)-justified) ROFs. Then, one constructs a generator that, in some sense, contains many zeros of any of the irreducible factors. The idea being that if a circuit vanishes when composed with this generator then it will basically vanish on the zero set of each of the irreducible factors of our special polynomial. This will imply that the polynomial is a factor of the circuit which, by the hardness result, should lead to a contradiction (see Theorems 3.4.2 and 3.4.4 in Section 3.4.1).
3.1.3 Comparison to Previous Works

Arithmetic read-once formulas received a lot of attention in the context of learning theory and exact learning algorithms were given to them. We shall now discuss the different variants that were studied and highlight the differences from our work.

In [27] learning algorithms for arithmetic read-once formulas that use membership and equivalence queries were given. A membership query to a ROF $\Phi(\bar{x})$ is simply a query that asks for the value of $\Phi(\bar{x})$ on a specific input. An equivalence query on the other hand, gives the oracle a certain hypothesis, $\Psi(\bar{x})$, and the oracle answers “equal” if $\Phi \equiv \Psi$ or returns an input $\bar{\alpha}$ such that $\Phi(\bar{\alpha}) \neq \Psi(\bar{\alpha})$. Following [27], other works gave learning algorithms for various extensions of read-once formulas [14, 12, 13]. All those works (including the original work [27]) give randomized learning algorithms.

Our results are also related to the model of depth-3 $\Sigma\Pi\Sigma(k)$ circuits. This model was extensively studied in recent years [17, 41, 37, 58, 38, 40, 9, 59, 60] as it stands between the simpler depth-2 case and the depth-4 case that, when studying lower bounds and polynomial identity testing, is (almost) as hard as the general case [3]. Prior to this work the best known black-box PIT algorithm for degree $d$ $\Sigma\Pi\Sigma(k)$ circuits had running time of $\text{poly}(n) \cdot d^{O(k^3 \log d)}$ for the general case and of $n^{O(k)}$ for the multilinear case [37, 58]. Both results were obtained via the rank-bound (see Section 3.5). We improve the algorithm for the multilinear case and obtain an $n^{O(k)}$ algorithm that also works in the preprocessed case. Our PIT algorithm uses a different technique than previous approaches. In addition, applying a recent result of [58] we obtain a new PIT algorithm for the general case with (roughly) the same running time $(nd)^{O(k^3 \log d)}$.

Note that Theorem 5 actually gives a PIT for a restricted class of depth-4 circuits. At the time of the initial publication of our work this was the first black-box PIT algorithm for a restricted class of depth-4 circuits. Other known PIT algorithms for depth-4 circuits were non black-box and covered only other very special cases. Arvind and Mukhopadhyay [9] gave a polynomial time PIT algorithm for the case that $k = O(1)$ and the additional requirement that each linear function depends on a constant number of variables. Saxena [57] gave a polynomial time PIT algorithm for the case where each linear product consists of a constant number of linear functions (but
3.1.4 Organization

This chapter is organized as follows. We give the formal definition of our model (Section 3.2) along with some important technical properties of the ROFs that will be later used. In the following sections we prove our theorems: The new black-box PIT algorithm for a single PROF (Theorem 3) is given in Section 3.3. This section also contains a PIT algorithm for bounded-depth PROFs (Theorem 4 for the case $k = 1$). In Section 3.4 we consider sums of PROFs and prove Theorems 1, 2, 4 and 7. All those results are based on the hardness of representation approach that is given in Section 3.4.1. Finally, in Section 3.5 we consider depth-3 circuit. We introduce the model and give PIT algorithms for those circuits and special cases of depth-4 circuits, proving Theorems 5 and 6.

3.2 Our Model

In this section we discuss our computational model. We first consider the basic model of read-once formulas and cover some of its main properties. Then, we introduce the model of preprocessed-read-once formulas and give its corresponding properties.

3.2.1 Read-Once Formulas and Read-Once Polynomials

Most of the definitions that we give in this section are from [27], or some small variants. We start by formally defining the notions of a read-once formula and a read-once polynomial.

**Definition 3.2.1.** An arithmetic read-once formula (ROF for short) $\Phi$ over a field $\mathbb{F}$ in the variables $\bar{x} = (x_1, \ldots, x_n)$ is a binary tree whose leaves are labelled with the input variables and whose internal nodes are labelled with the arithmetic operations $\{+, \times\}$ and with a pair of field elements $^2(\alpha, \beta) \in \mathbb{F}^2$. Each input variable can label at most one leaf. The computation is performed in the following way. A leaf labelled with the variable $x_i$ and with $(\alpha, \beta)$ computes the polynomial $\alpha \cdot x_i + \beta$. If a node $v$ is labelled with the

---

^2 This is a slightly more general model than the usual definition of read-once formulas.
operation $\text{op}$ and with $(\alpha, \beta)$, and its children compute the polynomials $\Phi_{v_1}$ and $\Phi_{v_2}$ then the polynomial computed at $v$ is $\Phi_v = \alpha \cdot (\Phi_{v_1} \text{ op } \Phi_{v_2}) + \beta$. We say that a ROF $\Phi$ is non-degenerate if it depends on all the variables appearing in it.

A polynomial $P(\bar{x})$ is a read-once polynomial (ROP for short) if it can be computed by a read-once formula. Clearly, ROPs form a subclass of multilinear polynomials.

A ROF is called a multiplicative ROF if it has no addition gates. A polynomial computed by a multiplicative ROF is called a multiplicative ROP. Note that because we allow gates to apply linear functions on the results of their operations, the output of a multiplicative ROF can be more than just a monomial.

**Example 3.2.2.** The polynomial $(5x_1 \cdot x_2 + 1) \cdot ((-x_3 + 2) \cdot (2x_4 - 1) + 5)$ has a multiplicative ROF.

### 3.2.2 The Gates-Graph of ROFs and ROPs

In this section we define the important notion of gates-graph of a ROF and a ROP. A similar notion was defined in [27], but here we define it in a slightly more general form.

Given a graph of computation of a ROF it is natural to define the first common gate of a subset of the gates. A similar notion was presented in [14].

**Definition 3.2.3.** Let $V \subseteq \text{var}(\Phi)$, $|V| \geq 2$ be a subset of the input variables of a ROF $\Phi$. We define the first common gate of $V$, $\text{fcg}(V)$, to be the first gate in the graph of computation of $\Phi$ common to all the paths from the inputs of $V$ to the root of the formula.

We note, that $\text{fcg}(V)$ is in fact the least common ancestor of the nodes in $V$ when looking at the formula as a tree. We now define for every read-once formula several new graphs that are related to it. The notion of a $t$-graph of a ROF (where $t$ is a type of a gate) was introduced and studied in [27] for various types of gates (such as threshold gates, modular gate, division gates etc.). We shall focus on $+$ and $\times$ gates as these are the only gates appearing in our ROF’s. We now give the definition of a $t$-graph that is slightly different from the one in [27].
**Definition 3.2.4 (t-graph).** Let $\Phi$ be a ROF in the input variables $\text{var}(\Phi)$. Let $t \in \{+, \times\}$ be a possible label of an internal node. The $t$-graph of the formula $\Phi$ is an undirected graph whose vertex set is $\text{var}(\Phi)$ and its edges are defined as follows: there is an edge $(i, j)$ if and only if the arithmetic operation that labels the gate $\text{fcg}(x_i, x_j)$ is $t$. We denote this graph with $G^t_{\Phi}$.

The following simple lemma (whose basic form can be found in [27]) gives some basic properties of $t$-graphs. We state it here (mainly) for alternating ROF's. We omit the proof as it is implicit in the proof of Lemma 3 in [27].

**Lemma 3.2.5.** The graphs satisfy the following relations.

1. Let $P$ be a ROP and let $\Phi$ and $\Psi$ be two ROFs computing $P$. Then $G^\times_{\Phi} = G^\times_{\Psi}$ and $G^+_{\Phi} = G^+_{\Psi}$. In particular, the graphs $G^\times_P \Delta G^\times_{\Phi}$ and $G^+_{\Phi} \Delta G^+_{\Psi}$ are well defined. As a result, we shall focus on the graphs of the ROPs from now on instead of the graphs of the ROFs computing them.

2. For every ROP $P$ we have $G^\times_P = (G^+_P)^c$. That is, the addition and the multiplication graphs of each ROP $P$ are simply complements to each other.

3. For every ROP $P$ exactly one of $G^\times_P$ and $G^+_P$ is disconnected. More precisely, the type of the top gate of $P$ is $t$ if and only if $G^t_P$ is connected. Alternatively, the type of the top gate of $P$ is $t$ if and only if $(G^t_P)^c$ is disconnected.

We define the gates-graph of a ROF $\Phi$ as $G^\Delta_{\Phi} = G^\times_{\Phi}$. Similarly, we define $G_P$ the gates-graph of a ROP $P$. The proof of the following claim is not difficult.

**Lemma 3.2.6.** Let $P$ be a ROP. Then $(i, j)$ is an edge in the gates-graph $G_P$ if and only if $x_i \cdot x_j$ appears in some monomial of $P$. Moreover, if $I$ is a clique in $G_P$ then there is some monomial in $P$ containing $\prod_{i \in I} x_i$.

The following is an algebraic version of the above lemma. It also provides a more algebraic definition of the gates-graph of a ROP $P$. 

32
Lemma 3.2.7. Let $P(x_1, \ldots, x_n)$ be a ROP. Then $(i, j)$ is an edge in the gates-graph $G_P$ if and only if $\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0$. Moreover, for $I \subseteq [n]$ we have that $\partial_I P \neq 0$ if and only if $I$ is a clique in $G_P$.

Next, we bring two examples of family of multilinear polynomials that are not ROPs.

Example 3.2.8. The polynomial $P(x_1, x_2, \ldots, x_n) = x_1 x_2 + x_2 x_3 + \cdots + x_{n-1} x_n$, $n \geq 4$ is not a ROP. Indeed, assume the contrary. Then by Lemma 3.2.6 we get that both the +-graph and the ×-graph are connected for this polynomial. This contradicts Lemma 3.2.5.

Example 3.2.9. For every $n \geq 3$, the polynomial $P(x_1, x_2, \ldots, x_n) = \sum_{i \neq j \in [n]} x_i x_j$ is not a ROP. Notice that $G_P$ is complete graph (a clique) while $\partial_{[n]} P \equiv 0$. In contradiction to Lemma 3.2.7.

The following lemma is also an easy corollary of Lemma 3.2.5.

Lemma 3.2.10 (ROP Structural Lemma). Every ROP $P(\bar{x})$ such that $|\text{var}(P)| \geq 2$ can be presented in exactly one of the following forms:

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$
2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$

where $P_1$ and $P_2$ are non-constant variable disjoint ROPs and $c$ is a constant.

The following is another simple claim regarding representations of ROFs.

Lemma 3.2.11. Let $P(\bar{x})$ be a ROP and $v$ a node in a ROF $\Phi$ computing $P$. Denote by $\Phi_v(\bar{x})$ the polynomial that is computed by $v$. Then there exists a polynomial $Q(y, \bar{x})$ such that $Q(\Phi_v(\bar{x}), \bar{x}) \equiv P(\bar{x})$ and, in addition, $\Phi_v$ and $Q$ can be computed by variable-disjoint ROFs.

Proof. Consider $\Phi$’s graph of computation. Denote with $\Psi$ the sub-formula whose top gate is $v$. Let $\varphi$ be the rest of the graph. The output of $\Psi$ is wired as one of the inputs of $\varphi$. We denote this input by $y$. We define $Q$ to be the polynomial computed by $\varphi$. Consequently, $Q(\Phi_v(\bar{x}), \bar{x}) \equiv P(\bar{x})$ and $\Phi_v, Q$ are variable-disjoint ROPs as they are computed by different parts of the same ROF. \qed
3.2.3 Factors of ROPs

In this section we study the properties of factors of ROPs. It can be easily seen that every factor of a ROP is a ROP itself. We give another proof to that fact that can be use to devise an linear-time factorization algorithm for explicitly given ROFs.

To handle factorization we are going to use the multilinear commutator. Previously, a more general definition was used (see Definition 5.4.5). Yet, in this section we are going to restrict ourselves to the multilinear setting.

**Definition 3.2.12.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial and let $i, j \in [n]$. We define the multilinear commutator between $x_i$ and $x_j$ as

$$\Delta_{ij} P \triangleq P|_{x_i=1,x_j=1} \cdot P|_{x_i=0,x_j=0} - P|_{x_i=1,x_j=0} \cdot P|_{x_i=0,x_j=1}$$

The following lemma gives a crucial property of the commutator.

**Lemma 3.2.13.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a multilinear polynomial and let $i, j \in \text{var}(P)$. Then $P$ is $(x_i, x_j)$-decomposable if and only if $\Delta_{ij}$ \equiv 0.

The proof is similar to the proof of Lemma 5.4.6. In addition, we have the following observation:

**Observation 3.2.14.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a multilinear polynomial and let $c \in \mathbb{F}$ be a constant. Then $\Delta_{ij}(P + c) = \Delta_{ij}P + c \cdot \frac{\partial^2 P}{\partial x_i \partial x_j}$.

We can prove an important property of the ROPs.

**Lemma 3.2.15.** Every factor of a ROP is a ROP.

**Proof.** Let $P(\bar{x}) = h_1(\bar{x}) \cdot h_2(\bar{x})$ be a reducible ROP and its (not necessarily irreducible) non-constant factors, respectively. In particular, $|\text{var}(P)| \geq 2$. According to Lemma 3.2.10 $P$ can be in exactly one of the following two forms:

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Notice that the $h_1, h_2$ are variable disjoint. Consequently, there exist $x_i, x_j$ and such that $x_i \in \text{var}(h_1) \cap \text{var}(P_1)$ and $x_j \in \text{var}(h_2) \cap \text{var}(P_2)$. Otherwise all the variables of $P$ will be in the same factor. Now, by considering $\frac{\partial^2 P}{\partial x_i \partial x_j}$ we obtain on one hand

$$\frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial^2}{\partial x_i \partial x_j}(P_1 + P_2) \equiv 0$$

since $P_1$ and $P_2$ are variable-disjoint. On
the other hand \( \frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial h_1}{\partial x_i} \cdot \frac{\partial h_2}{\partial x_j} \). As \( \frac{\partial h_1}{\partial x_i}, \frac{\partial h_2}{\partial x_j} \) and are non-zero, we reach a contradiction.

2. \( P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c \). As previously, there exist \( x_i, x_j \) such that \( x_i \in \text{var}(h_1) \cap \text{var}(P_1) \) and \( x_j \in \text{var}(h_2) \cap \text{var}(P_2) \). This time we consider \( \Delta_{ij} P \). On one hand \( \Delta_{ij} P = \Delta_{ij} (h_1 \cdot h_2) \equiv 0 \). However, on the other hand, by Observation 3.2.14 \( \Delta_{ij} P = c \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j} \). This implies that \( c = 0 \). It follows that \( P_1 \) and \( P_2 \) are factors of \( P \) and that \( P_1 \) and \( P_2 \) are ROPs. A simple induction completes the proof.

\( \square \)

The above proof has an addendum: it allows us to extend the notion of factorization/reducibility. We can strengthen Lemma 3.2.10.

**Observation 3.2.16.** Let \( P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c \), \( P(\bar{x}) = P'_1(\bar{x}) \cdot P'_2(\bar{x}) + c' \) be a ROP \( P \) and its two representations, (according to the second case of Lemma 3.2.10), respectively. Then \( c = c' \). In other words, the constant \( c \) is unique.

Given the above it can be meaningful to discuss the factors of \( P - c \) for a ROP \( P \). The above observation on the uniques of factorization gives rise to the following definition.

**Definition 3.2.17.** Let \( P \in \mathbb{F}[x_1, \ldots, x_n] \) be a ROP. We say that \( P \) is reducible up to a constant if there exists a constant \( c \in \mathbb{F} \) such that \( P - c \) is reducible.

We define the graph \( G_{P}^{FACT} \) to be an undirected graph whose vertex set is \( \text{var}(P) \) and the edge \( (i, j) \in G_{P}^{FACT} \) iff \( P \) can be written as \( P = h \cdot g + c \) when \( x_i \in \text{var}(h) \setminus \text{var}(g) \), \( x_j \in \text{var}(g) \setminus \text{var}(h) \) and \( c \) is constant.

Given the above observation \( G_{P}^{FACT} \) is well-defined. We would like to tie reducibility up to a constant to the commutator. The problem is that \( \Delta_{ij} P \equiv 0 \) only when the constant \( c = 0 \). To circumvent this problem will apply the commutator on some partial derivatives of \( P \). We get the following results.

**Lemma 3.2.18.** Let \( P \) be a ROP and let \( i \neq j \in \text{var}(P) \) such that \( (i, j) \notin G_{P}^{FACT} \). Then for every \( k \neq i, j \) it holds that \( \Delta_{ij} \left( \frac{\partial P}{\partial x_k} \right) \equiv 0 \).
Proof. Since \((i,j) \notin G_P^{FACT}\) the polynomial \(P\) can be written as \(P = P_1 \cdot P_2 + c\), when \(x_i \in \text{var}(P_1)\) and \(x_j \in \text{var}(P_2)\). W.l.o.g \(k \in \text{var}(P_1)\). Then \(\frac{\partial P}{\partial x_k} = \frac{\partial P_1}{\partial x_k} \cdot P_2\) and hence \(\Delta_{ij} \left( \frac{\partial P}{\partial x_k} \right) \equiv 0\). \(\square\)

Lemma 3.2.19. Let \(P\) be a \(\bar{0}\)-justified ROP and let \(i \neq j\) such that \(i \sim j\) in \(G_P^{FACT}\) but \(i \not\sim j\) in \(G_P^*\). Then there exists \(k \neq i,j\) such that \(P_{i,j,k} \triangleq P|_{x[i] \backslash (i,j,k)} = \bar{0}_{[n] \backslash (i,j,k)}\) can be represented in the form \(P_{i,j,k} = P'_1 \cdot P'_2 + c'\) such that:

1. \(\text{var}(P'_1) = \{x_i, x_j\}\)
2. \(\text{var}(P'_2) = \{x_k\}\)
3. \(\Delta_{ij} P'_1 \neq 0\) - note that \(\Delta_{ij} P'_1\) is a constant

Proof. First, we note that \(G_P^*\) is disconnected, therefore by Lemmas 3.2.5 and 3.2.10 we can write \(P = P_1 \cdot P_2 + c\). As \(i \sim j\) in \(G_P^{FACT}\) we can assume w.l.o.g that \(x_i, x_j \in \text{var}(P_1)\) and \(\Delta_{ij} P_1 \neq 0\). Furthermore, it must be the case that \(i \not\sim j\) in \(G_P^*\) (as \(G_P^*\) is a subgraph of \(G_P^{FACT}\)). Therefore, by a repeating a previous reasoning we can write \(P_1 = Q_1 \cdot Q_2 + b\) or equivalently \(P = (Q_1 \cdot Q_2 + b) \cdot P_2 + c\).

The proof is carried out by induction on \(m = |\text{var}(P)|\). The above implies that \(m \geq 3\) and that the claim holds for \(m = 3\). Assume that \(m \geq 4\). There are a few cases to consider:

1. \(|\text{var}(P_2)| \geq 2\). Let \(x_\ell \in \text{var}(P_2)\). Consider \(P' \triangleq P|_{x_\ell=0} = P_1 \cdot P_2|_{x_\ell=0} + c\). Note that \(P'\) is \(\bar{0}\)-justified ROP, \(P_2|_{x_\ell=0} \neq 0\), \(i \sim j\) in \(G_P^{FACT}\) but \(i \not\sim j\) in \(G_P^*\) and \(|\text{var}(P')| < |\text{var}(P)|\). Therefore, we can apply the induction hypothesis on \(P'\) to obtain \(P'_{i,j,k} = P'_1 \cdot P'_2 + c'\) with the required properties. Observe that \(P_{i,j,k} = P'_{i,j,k}\).

2. \(x_i \in \text{var}(Q_1), x_j \in \text{var}(Q_2)\). As \(P_1 = Q_1 \cdot Q_2 + b\) and \(\Delta_{ij} P_1 \neq 0\) it must the case that \(b \neq 0\). In addition, as w.l.o.g \(|\text{var}(P_2)| = 1\) and \(|\text{var}(Q_1)| + |\text{var}(Q_2)| + |\text{var}(P_2)| = |\text{var}(P)| \geq 4\) it follows that w.l.o.g \(|\text{var}(Q_1)| \geq 2\). Now, let \(x_\ell \neq x_i \in \text{var}(Q_1)\). Consider \(P' \triangleq P|_{x_\ell=0} = (Q_1|_{x_\ell=0} \cdot Q_2 + b) \cdot P_2 + c\). As \(b \neq 0\) and \(Q_1\) is a \(\bar{0}\)-justified ROP we can conclude that (as previously) \(x_i \in \text{var}(Q_1)\) and that \(P'\) satisfies the required properties, hence we can apply the induction hypothesis to obtain \(P'_{i,j,k} = P'_1 \cdot P'_2 + c'\). As previously, note that \(P_{i,j,k} = P'_{i,j,k}\).
3. w.l.o.g. $x_i, x_j \in \text{var}(Q_1)$. The case is carried out in the same manner as case 1.

Corollary 3.2.20. Let $P$ be a $\bar{0}$-justified ROP and let $i \neq j$ be such that $i \sim j$ in $G_P^{FACT}$ but $i \not\sim j$ in $G_P^+$. Then there exists $k \neq i, j$ such that

$$\Delta_{ij} \left( \frac{\partial P}{\partial x_k} \right)(\bar{0}) \neq 0.$$ 

Proof. Let $k$ be as promised by Lemma 3.2.19. Then

$$\Delta_{ij} \left( \frac{\partial P}{\partial x_k} \right)(\bar{0}) = \Delta_{ij} \left( \frac{\partial P_{i,j,k}}{\partial x_k} \right) = \Delta_{ij} P_1' \neq 0.$$ 

3.2.4 Partial Derivatives of ROPs

In this section we list some important properties of the partial derivatives of ROPs. Clearly, a partial derivative of a multilinear polynomial is a multilinear polynomial. In particular, a partial derivative of a ROP is a multilinear polynomial as well. The following lemma gives a stronger statement.

Lemma 3.2.21. A partial derivative of a ROP is a ROP.

Proof. Let $P$ be a ROP and $i \in [n]$. We prove the claim by induction on $k = |\text{var}(P)|$. For $k = 0, 1$ the claim is trivial. For $k \geq 2$ we get by Lemma 3.2.10 that $P$ can be in one of two forms.

Case 1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Since $P_1$ and $P_2$ are variable-disjoint we can assume w.l.o.g. that $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i}$. In addition, $|\text{var}(P_1)| < |\text{var}(P)|$ and so by the induction hypothesis we get that $\frac{\partial P}{\partial x_i}$ is a ROP.

Case 2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$. Again we assume w.l.o.g. that $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i} \cdot P_2$. As before, $\frac{\partial P_1}{\partial x_i}$ is a ROP. Since $P_1$ and $P_2$ are variable-disjoint and $P_2$ is a ROP, we have that $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i} \cdot P_2$ is a ROP as well.

We now give two useful properties of the derivatives of ROPs.

Observation 3.2.22. Let $P$ be a ROP and let $x_i, x_j \in \text{var}(P)$. Let $\Phi$ be a ROF computing $P$ and $v = \text{fpg}\{x_j, x_i\}$. Then $v$ is a multiplication gate iff

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0.$$ 

37
Lemma 3.2.23 (2-nd Derivative Lemma). Let \( P(x_1, x_2, \ldots, x_n) \) be a ROP such that \( \frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0 \) and \( \frac{\partial^2 P}{\partial x_i \partial x_j} \big|_{x_k = \alpha} \equiv 0 \) for three different indices \( i, j, k \) and \( \alpha \in \mathbb{F} \). Then, \( \frac{\partial P}{\partial x_i} \big|_{x_k = \alpha} \equiv 0 \) or \( \frac{\partial P}{\partial x_j} \big|_{x_k = \alpha} \equiv 0 \).

Proof. First, notice that \( x_j, x_i \in \text{var}(P) \). Let \( \Phi \) be a ROF computing \( P \) and \( v = \text{fcg}\{x_j, x_i\} \). As \( \frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0 \), it follows that \( v \) is a multiplication gate. Let \( R(\bar{x}) \) be the polynomial computed by \( v \) in \( \Phi \). From Lemma 3.2.11 there exists a ROP \( Q(y, \bar{x}) \) such that \( Q(R(\bar{x}), \bar{x}) \equiv P(\bar{x}) \). Clearly, \( x_j, x_i \in \text{var}(R) \setminus \text{var}(Q) \). By the definition of \( \text{fcg} \) and Lemma 3.2.10 we obtain that \( R(\bar{x}) \) can be presented as \( P_1 \cdot P_2 + c \) where \( x_i \in \text{var}(P_1), x_j \in \text{var}(P_2) \) and \( c \in \mathbb{F} \). By the chain rule (Lemma 2.1.9):

\[
\frac{\partial P}{\partial x_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial R}{\partial x_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2 \quad \text{and} \quad \frac{\partial P}{\partial x_j} = \frac{\partial Q}{\partial y} \cdot \frac{\partial R}{\partial x_j} = \frac{\partial Q}{\partial y} \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j} .
\]

Consequently, \( \frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial^2 Q}{\partial y \partial x_i} \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j} \). This implies that:

\[
\frac{\partial P}{\partial x_i} \big|_{x_k = \alpha} \cdot \frac{\partial P}{\partial x_j} \big|_{x_k = \alpha} = \left( \frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2 \right) \big|_{x_k = \alpha} \cdot \left( \frac{\partial Q}{\partial y} \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j} \right) \big|_{x_k = \alpha} = \\
\left( \frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j} \right) \big|_{x_k = \alpha} \cdot \left( \frac{\partial Q}{\partial y} \cdot P_1 \cdot P_2 \right) \big|_{x_k = \alpha} = \\
\frac{\partial^2 P}{\partial x_i \partial x_j} \big|_{x_k = \alpha} \cdot \left( \frac{\partial Q}{\partial y} \cdot P_1 \cdot P_2 \right) \big|_{x_k = \alpha} \equiv 0.
\]

In particular, either \( \frac{\partial P}{\partial x_i} \big|_{x_k = \alpha} \equiv 0 \) or \( \frac{\partial P}{\partial x_j} \big|_{x_k = \alpha} \equiv 0 \) holds.

As a corollary we obtain a simple example of multilinear polynomial which is not a ROP.

Example 3.2.24. The polynomial \( P(x_1, x_2, x_3) = x_1 x_2 x_3 + x_1 + x_2 \) is not a ROP. To see this apply Lemma 3.2.23 on \( P \) with the parameters \( i = 1, j = 2, k = 3, \alpha = 0 \).

3.2.5 Multiplicative and \( \bar{0} \)-Justified ROPs

Recall that multiplicative ROFs are ROFs with no addition gates. Observation 3.2.22 provides an algebraic characterization of ROPs computed by such ROFs (i.e. multiplicative ROPs).
Lemma 3.2.25. A ROP $P$ is a multiplicative ROP iff for any two variables $x_i \neq x_j \in \text{var}(P)$ we have that $\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0$.

From now on, whenever we discuss multiplicative ROPs we shall use the property described in the claim as an alternative definition. The following lemma is a generalization of Lemma 3.2.21 to weakly-0-justified ROPs.

Lemma 3.2.26. A partial derivative of a weakly-0-justified ROP is a weakly-0-justified ROP.

Proof. Let $P$ to be a weakly-0-justified ROP. From Lemma 3.2.21 it is enough to show that the partial derivatives of $P$ are weakly-0-justified. Assume for a contradiction that for some $i \in [n]$, we have that $\frac{\partial P}{\partial x_i}$ is not weakly-0-justified. That is, there exist some $j,k \in [n]$ such that $\frac{\partial P}{\partial x_j}$ depends on $x_k$ however $\frac{\partial P}{\partial x_k}|_{x_k=0}$ does not. In other words, we have that: $\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0$ and $\frac{\partial^2 P}{\partial x_i \partial x_j}|_{x_k=0} \equiv 0$. Note that $\{x_i, x_j\} \subseteq \text{var}(P)$. Lemma 3.2.23 implies that either $\frac{\partial P}{\partial x_k}|_{x_k=0} \equiv 0$ or $\frac{\partial P}{\partial x_j}|_{x_k=0} \equiv 0$ holds. On the other hand, $\{x_i, x_j\} \subseteq \text{var}(P|_{x_k=0})$ since $P$ is a weakly-0-justified ROP and hence $\frac{\partial P}{\partial x_i}|_{x_k=0} \neq 0$ and $\frac{\partial P}{\partial x_j}|_{x_k=0} \neq 0$, in contradiction.

It is also possible to extend this proof for 0-justified ROPs and to get a following property:

Lemma 3.2.27. Let $P$ be a 0-justified ROP. Then $\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0$ iff $\frac{\partial^2 P}{\partial x_i \partial x_j}(\bar{0}) \neq 0$.

In a similar (and simpler) way, we observe the following.

Lemma 3.2.28. Every factor of a weakly-0-justified ROP is a weakly-0-justified ROP.

Proof. Let $P = h_1 \cdot h_2$ be a ROP, where $h_1$ and $h_2$ are two multilinear polynomials. As $P$ is multilinear, $h_1$ and $h_2$ must be variable-disjoint. Therefore, we can write $P(\bar{x}, \bar{y}) = h_1(\bar{x}) \cdot h_2(\bar{y})$. As $P$ is weakly-0-justified so are $h_1$ and $h_2$. Furthermore, it holds that $h_1(\bar{0}), h_2(\bar{0}) \neq 0$. Consequently, $h_1(\bar{x}) = P(\bar{x}, \bar{0})/h_2(\bar{0})$ is a weakly-0-justified ROP and so is $h_2(\bar{y})$.

We note that the same proof shows that a factor of any ROP is also a ROP. We conclude this section with a very useful property of multiplicative ROPs.
Lemma 3.2.29. Let $P$ be a weakly-$\bar{0}$-justified multiplicative ROP with $|\text{var}(P)| \geq 2$. Then, for every $x_i \in \text{var}(P)$ there exists $x_j \in \text{var}(P)$ satisfying the following properties:

1. $\frac{\partial P}{\partial x_j} = (x_i - \alpha) \cdot h_j(\bar{x})$ for some $\alpha \neq 0 \in \mathbb{F}$ and $h_j(\bar{x})$ (in particular, $\frac{\partial P}{\partial x_j}|_{x_i=\alpha} \equiv 0$).

2. $h_j(\bar{x})$ is a weakly-$\bar{0}$-justified ROP with $\text{var}(h_j) = \text{var}(P) \setminus \{x_i, x_j\}$.

3. There exists at most one element $\beta \neq \alpha \in \mathbb{F}$ such that $P|_{x_i=\beta}$ is not weakly-$\bar{0}$-justified.

Proof. Let $\Phi$ be a multiplicative ROF computing $P$. As $|\text{var}(\Phi)| = |\text{var}(P)| \geq 2$, $\Phi$ has at least one gate. Let $v$ be the unique entering gate\footnote{The entering gate of $x_i$ is the neighbor of the leaf labelled by $x_i$.} of $x_i$. We denote by $R(\bar{x})$ the ROP computed by $v$. Assume w.l.o.g that $\text{var}(R) = \{x_1, x_2, \ldots, x_{i-1}, x_i\}$. By Lemma 3.2.11 there exists some ROP $Q(y, x_{i+1}, \ldots, x_n)$ such that $Q(R(x_1, x_2, \ldots, x_i, x_{i+1}, \ldots, x_n) \equiv P(x_1, x_2, \ldots, x_n)$. Since $v$ is a multiplication gate (recall that $\Phi$ is a multiplicative ROF) and is the entering gate of $x_i$, we get, in a similar manner to Lemma 3.2.10, that $R$ can be written as $R(\bar{x}) = (x_i - \alpha) \cdot H(\bar{x}) + c$ for some ROP $H(\bar{x})$ such that $\text{var}(H) \neq \emptyset$ and $x_i \notin \text{var}(H)$. By the chain rule, for every $x_j \in \text{var}(H)$ it holds that:

$$\frac{\partial P}{\partial x_j} = \frac{\partial Q}{\partial y} \cdot \frac{\partial R}{\partial x_j} = \left(\frac{\partial Q}{\partial y} \cdot (x_i - \alpha) \cdot \frac{\partial H}{\partial x_j}\right) = (x_i - \alpha) \cdot h_j(\bar{x}) \quad (3.1)$$

where $h_j(\bar{x}) \triangleq \frac{\partial Q}{\partial y} \cdot \frac{\partial H}{\partial x_j}$. As $\frac{\partial P}{\partial x_j}$ is a ROP so is $h_j$. Since $P$ is a weakly-$\bar{0}$-justified ROP we get by Lemma 3.2.26 that $\frac{\partial P}{\partial x_j}$ is a weakly-$\bar{0}$-justified ROP as well. By Lemma 3.2.28 $h_j(\bar{x})$ is a weakly-$\bar{0}$-justified ROP and $\alpha \neq 0$. This completes the proof of Properties 1 and 2.

Now suppose that for some $\beta \neq \alpha \in \mathbb{F}$ the polynomial $P|_{x_i=\beta}$ is not weakly-$\bar{0}$-justified. We will show that the value of $\beta$ is uniquely defined. By definition there exist some variables $x_t, x_k \neq x_i \in \text{var}(P)$ such that setting $x_k = 0$ affects the dependence of $P|_{x_i=\beta}$ on $x_t$. Equivalently, $\frac{\partial P}{\partial x_t}|_{x_i=\beta} \neq 0$ but $\left(\frac{\partial P}{\partial x_t}|_{x_i=\beta}\right)|_{x_k=0} \equiv 0$. In particular, $\frac{\partial P}{\partial x_t} \neq 0$ which implies $\frac{\partial P}{\partial x_t}|_{x_k=0} \neq 0$ since $P$ is weakly-$\bar{0}$-justified. In other words, setting $x_i = \beta$ affects the
dependence of $P|_{x_k=0}$ on $x_\ell$. We consider two cases.

**Case 1:** $x_\ell \in \text{var}(H)$ (i.e. $1 \leq \ell \leq i-1$): By Equation (3.1) it holds that $\frac{\partial P}{\partial x_\ell}|_{x_k=0} = (x_i - \alpha) \cdot h_\ell|_{x_k=0}$ and hence $\frac{\partial P}{\partial x_\ell}|_{x_k=0, x_i=\beta} = (\beta - \alpha) \cdot h_\ell|_{x_k=0}$. As $\beta - \alpha \neq 0$ we conclude that this case is impossible.

**Case 2:** $x_\ell \in \text{var}(Q)$ (i.e. $i+1 \leq \ell \leq n$): Here we have that $\frac{\partial Q}{\partial x_\ell}|_{x_k=0} = \frac{\partial P}{\partial x_\ell}|_{y=R(\bar{x})}$. On the one hand we have that $\frac{\partial Q}{\partial x_\ell}|_{x_k=0, y=(R|_{x_k=0})} \neq 0$ since $\partial Q/\partial x_\ell|_{x_k=0, y=(R|_{x_k=0})} = \frac{\partial P}{\partial x_\ell}|_{x_k=0, x_i=\beta} \equiv 0$. On the other hand,

$$\frac{\partial Q}{\partial x_\ell}|_{x_k=0, y=(R|_{x_k=0}, x_i=\beta)} = \frac{\partial P}{\partial x_\ell}|_{x_k=0, x_i=\beta} \equiv 0.$$

Therefore, from Lemma 2.1.17 we conclude that $y - R|_{x_k=0, x_i=\beta}$ is a factor of $\frac{\partial Q}{\partial x_\ell}|_{x_k=0}$. Since $\text{var}(R) \cap \text{var}(Q) = \emptyset$ and $Q$ is a multilinear polynomial it follows that there must exist (exactly one) $\gamma \in \mathbb{F}$ such that $R|_{x_k=0, x_i=\beta} = \gamma$ (otherwise $R$ introduces variables that do not appear in $Q$). Recall that $R(\bar{x}) = (x_i - \alpha) \cdot H(\bar{x}) + c$. Thus, $H|_{x_k=0} = \frac{\gamma - c}{\beta - \alpha}$ (since $\beta - \alpha \neq 0$ and $x_i \notin \text{var}(H)$). I.e., $H|_{x_k=0}$ is constant. As $H$ is a non-constant polynomial, it must the case that $x_k \in \text{var}(H)$. Finally, notice that since $P$ is a weakly-0-justified polynomial then it must be the case that $\text{var}(H) = \{x_k\}$ (otherwise, if there is $x_m \neq x_k \in \text{var}(H)$ then $\frac{\partial P}{\partial x_m}|_{x_k=0} \equiv 0$ as $\frac{\partial H}{\partial x_m}|_{x_k=0} \equiv 0$ in contradiction). We conclude that $H$ is a univariate polynomial in $x_k$ and that the value of $\beta$ is uniquely defined by $\alpha, \gamma, c$ and $H$, which, in turn, are uniquely defined by $P$. \hfill \square

### 3.2.6 Preprocessed Read-Once Polynomials

In this section we extend the model of ROFs by allowing a preprocessing step of the input variables. While the basic model is read-once in its variables, the extended model can be considered as read-once in univariate polynomials.

**Definition 3.2.30.** A preprocessing is a transformation $T(\bar{x}) : \mathbb{F}^n \to \mathbb{F}^n$ of the form $T(\bar{x}) \triangleq (T_1(x_1), T_2(x_2), \ldots, T_n(x_n))$ such that each $T_i$ is a non-
constant univariate polynomial. We say that a preprocessing is standard if in addition to the above each \( T_i \) satisfies \( T_i(0) = 0 \).

Notice that preprocessings do not affect the PIT problem in the non black-box setting as for every \( n \)-variate polynomial \( P(\bar{y}) \) it holds that \( P(\bar{y}) \equiv 0 \) if and only if \( P(T(\bar{x})) \equiv 0 \). We now give a formal definition and list some immediate properties.

**Definition 3.2.31.** A preprocessed arithmetic read-once formula (PROF for short) over a field \( \mathbb{F} \) in the variables \( \bar{x} = (x_1, \ldots, x_n) \) is a binary tree whose leafs are labelled with non-constant univariate polynomials \( T_1(x_1), T_2(x_2), \ldots, T_n(x_n) \) (all together forming a preprocessing) and whose internal nodes are labelled with the arithmetic operations \( \{+, \times\} \) and with a pair of field elements \( (\alpha, \beta) \in \mathbb{F}^2 \). Each \( T_i \) can label at most one leaf. The computation is performed in the following way. A leaf labelled with the polynomial \( T_i(x_i) \) and with \( (\alpha, \beta) \) computes the polynomial \( \alpha \cdot T_i(x_i) + \beta \). If a node \( v \) is labelled with the operation \( \text{op} \) and with \( (\alpha, \beta) \), and its children compute the polynomials \( \Phi_{v_1} \) and \( \Phi_{v_2} \) then the polynomial computed at \( v \) is \( \Phi_v = \alpha \cdot (\Phi_{v_1} \text{op} \Phi_{v_2}) + \beta \).

A polynomial \( P(\bar{x}) \) is a Preprocessed Read-Once Polynomial (PROP for short) if it can be computed by a preprocessed read-once formula. A Decomposition of a polynomial \( P \) is a couple \( Q(\bar{z}), T(\bar{x}) \) such that \( P(\bar{x}) = Q(T(\bar{x})) \) when \( Q \) is a ROP and \( T \) is a preprocessing. A Standard Decomposition is as above with the additional requirement that \( T \) is a standard preprocessing. An immediate consequence from the definition is that each PROP admits a decomposition.

**Lemma 3.2.32.** Every PROP \( P \) admits a standard decomposition.

**Proof.** Let \( (Q, T) \) be a decomposition of \( P \) consider the shifted polynomials:

\[
Q'(\bar{z}) \triangleq Q(\bar{z} + T(\bar{0})) = (z_1 + T_1(0), z_2 + T_2(0), \ldots, z_n + T_n(0))
\]

\[
T'_i(x_i) \triangleq T_i(x_i) - T_i(0), \ T'(\bar{x}) \triangleq (T'_1(x_1), T'_2(x_2), \ldots, T'_n(x_n)).
\]

It is easy to verify that \( (Q', T') \) is a standard decomposition of \( P \). \( \square \)

In order to handle PROP we define a generalization of the commutator.
Definition 3.2.33. Let \( P \in \mathbb{F}[x_1, \ldots, x_n] \) be a polynomial, a vector \( \bar{\alpha} \in \mathbb{F}^n \) and \( i, j \in [n] \). We define the directed commutator between \( x_i \) and \( x_j \) as
\[
\Delta_{ij}^\alpha P \overset{\Delta}{=} P|_{x_i=\alpha_i, x_j=\alpha_j} \cdot P|_{x_i=0, x_j=0} - P|_{x_i=\alpha_i, x_j=0} \cdot P|_{x_i=0, x_j=\alpha_j}.
\]

The following lemma summarizes the properties or PROP with respect to other operators.

Lemma 3.2.34. Let \( (Q(\bar{z}), T(\bar{x})) \) be a PROP and its standard decomposition, respectively. And let \( \bar{\alpha} \in \mathbb{F}[x_1, \ldots, x_n] \). Then the following properties hold:

- \( \frac{\partial P}{\partial x_i} \equiv 0 \iff \frac{\partial Q}{\partial z_i} \equiv 0 \) and \( \Delta_{ij}^\alpha P \equiv 0 \iff \Delta_{ij}^\alpha Q \equiv 0 \).
- \( P \) is \( \bar{0} \)-justified if and only if \( Q \) is \( \bar{0} \)-justified.
- \( \Delta_{ij}^\alpha P = \Delta_{ij}^\alpha Q|_{\bar{z}=T(\bar{x})} \cdot T_i(\alpha_i) \cdot T_j(\alpha_j) \).
- If \( \bar{\alpha} \) is a witness for \( P \) then \( \frac{\partial P}{\partial x_i} \equiv 0 \iff \frac{\partial Q}{\partial z_i} \equiv 0 \) and \( \Delta_{ij}^\alpha P \equiv 0 \iff \Delta_{ij}^\alpha Q \equiv 0 \).

Since the above properties trivially hold, we will use them implicitly. The following two lemmas are the PROPs analogs of Lemmas 3.2.10 and 3.2.21.

Lemma 3.2.35 (PROP Structural Lemma). Every PROP \( P(\bar{x}) \) such that \( |\text{var}(P)| \geq 2 \) can be presented in exactly one of the following forms:

1. \( P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x}) \)
2. \( P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c \)

where \( P_1 \) and \( P_2 \) are non-constant, variable-disjoint PROPs and \( c \) is a constant.
Lemma 3.2.36. A partial derivative of a PROP is a PROP.

The following lemma exhibits yet another important property of PROFs. Note that for characteristic zero fields the claim holds for every polynomial.

Lemma 3.2.37. Let $P$ be a PROP and $G = (G^1, \ldots, G^n) : \mathbb{F}^t \to \mathbb{F}^n$ be such that $P(G)$ is a non-constant polynomial. Then there exists $x_m \in \text{var}(P)$ such that $P(G^1, \ldots, G^{m-1}, x_m, G^{m+1}, \ldots, G^n)$ (the polynomial resulting from setting $x_i = G^i$ for every $i \neq m$) depends on $x_m$.

Proof. We prove the claim by induction on $k = |\text{var}(P)|$. Clearly, $k \geq 1$. We also note that for $k = 1$ the claim is trivial. For $k \geq 2$ we get by Lemma 3.2.35 that $P$ can be in a one of two forms.

Case 1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Since $(P_1 + P_2)(G)$ is a non-constant polynomial we get that w.l.o.g. $P_1(G)$ is a non-constant polynomial. In addition, $|\text{var}(P_1)| < |\text{var}(P)|$ and so by the induction hypothesis we get that there exists $x_m \in \text{var}(P_1)$ such that $P_1(G^1, \ldots, G^{m-1}, x_m, G^{m+1}, \ldots, G^n)$ depends on $x_m$. As $P_1$ and $P_2$ are variable-disjoint we obtain that $P(G^1, \ldots, G^{m-1}, x_m, G^{m+1}, \ldots, G^n)$ depends on $x_m$ as well.

Case 2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$. Again we assume w.l.o.g. that $P_1(G)$ is a non-constant polynomial and $P_2(G) \neq 0$. As before, there exists $x_m \in \text{var}(P_1)$ such that $P_1(G^1, \ldots, G^{m-1}, x_m, G^{m+1}, \ldots, G^n)$ depends on $x_m$, and from variable-disjointedness and the fact that $P_2(G) \neq 0$ we obtain that $P(G^1, \ldots, G^{m-1}, x_m, G^{m+1}, \ldots, G^n)$ depends on $x_m$ as well. \qed

The next example demonstrates that the claim is not true for general polynomials over fields with finite characteristics.

Example 3.2.38. Let $\mathbb{F}$ be a field of characteristic $p$. Consider $Q(x_1, \ldots, x_{p+1}) = \sum_{i=1, i \neq \bar{x}}^{p+1} x_j$. Note that $Q(y, y, \ldots, y) = (p+1) \cdot y^p = y^p$ is a non-constant polynomial, while for every $m$ we get that $Q(y, \ldots, y, x_m, y, \ldots, y) = p \cdot x_m \cdot y^{p-1} + y^p = y^p$ which does not depend on $x_m$. 

44
3.3 Black-Box PIT for Preprocessed Read-Once Polynomials

In this section we give a black-box PIT algorithm for PROPs, thus proving Theorem 3. The main idea is to convert a PROP $P$, that has many variables, each of low degree, into a polynomial $P'$ with a smaller number of variables while maintaining a reasonable degree, such that $P' \equiv 0$ if and only if $P \equiv 0$. In fact, we construct a low-degree generator for PROPs.

**Lemma 3.3.1.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero PROP with $|\text{var}(P)| \leq 2^t$, for some $t \geq 0$. Then $P(G_{t+1}) \neq 0$. Moreover, if $P$ is non-constant then so is $P(G_{t+1})$ (recall Definition 2.2.6).

**Proof.** We prove the claim by induction on $|\text{var}(P)|$. For $|\text{var}(P)| = 0$ the claim is trivial. Assume that $|\text{var}(P)| \geq 2$ (i.e. $t \geq 1$). By Lemma 3.2.35 we get that $P$ can be in a one of two forms.

**Case 1.** $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Since $P_1$ and $P_2$ are variable-disjoint we can assume w.l.o.g. that $|\text{var}(P_1)| \leq |\text{var}(P)|/2$ (in particular $|\text{var}(P_1)| < |\text{var}(P)|$). By the induction hypothesis we see that $P_1(G_t) \neq 0$ is a non-constant polynomial. Lemma 3.2.37 implies that there exists a variable $x_m \in \text{var}(P_1)$ such that even after setting $x_i = G_i^t$ for all other $i$’s, $P_1$ still depends on $x_m$. As $x_m \notin \text{var}(P_2)$ we obtain that $P(G_t^1, \ldots, G_t^m-1, x_m, G_t^{m+1}, \ldots, G_t^n)$ depends on $x_m$ as well. By Observation 2.2.7, $P(G_{t+1})|_{G_{t+1}^i = a_m} = P(G_t^1, \ldots, G_t^{m-1}, G_t^m + z_{t+1}, G_t^{m+1}, \ldots, G_t^n)$. Since $z_{t+1}$ only appears in the $m$-th coordinate it follows that $P(G_{t+1})|_{G_{t+1}^i = a_m}$ depends on $z_{t+1}$. Hence, $P(G_{t+1})$ is a non-constant polynomial and in particular $P(G_{t+1}) \neq 0$.

**Case 2.** $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$. As $P_1, P_2$ are non-constant and variable disjoint we have that $1 \leq |\text{var}(P_1)|, |\text{var}(P_2)| < |\text{var}(P)| \leq 2^t$. Hence, we can apply the induction hypothesis on both $P_1$ and $P_2$. As $P(G_{t+1}) = P_1(G_{t+1}) \cdot P_2(G_{t+1}) + c$, we see that $P(G_{t+1})$ is a non-constant polynomial (since $P_1(G_{t+1}), P_2(G_{t+1})$ are non-constant as well).

**Theorem 3.3.2.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero PROP with individual degrees bounded by $d$ that depends on at most $t$ variables$^4$. Then, there exists

---

$^4$Clearly, $t \leq n$ but we choose this more general statement.
an explicit set $\mathcal{H}$ of size $|\mathcal{H}| = (\mathcal{O}(\log t))$ such that $P|_{|\mathcal{H}} \not\equiv 0$.

Proof. Denote $\ell = \lceil \log_2 t \rceil + 1$. By Lemma 3.3.1 we get that $P(G_\ell) \not\equiv 0$. The proof follows from Lemma 2.2.5. Note that $|\mathcal{H}| \leq (n^2d)^{2\ell} = (nd)^{\mathcal{O}\log t}$. \qed

In particular, since every PROP depends on at most $n$ variables, we obtain an $(nd)^{\mathcal{O}(\log n)}$ time black-box PIT algorithm for PROPs, thus proving Theorem 3.

Remark 3.3.3. Lemma 3.3.1 shows that $G_\ell$ (for the appropriate value of $\ell$) is a generator for PROPs, (that is $P(G_\ell) \not\equiv 0$) regardless of the degree of $P$. It can be also shown that $G_\ell$ is a generator for a more general model - arithmetic read-once formulas with operations $\{+,\times,/\}$ (addition, multiplication, division). We leave this extension to the readers.

3.3.1 Small Depth Preprocessed Alternating Read-Once Formulas

In this section we use similar ideas to construct generators for PROPs computed by small depth formulas. When considering small depth (preprocessed) read-once formulas we allow the tree to have unbounded fan-in (and not just fan-in 2 as in the usual definition). Moreover, we allow small depth PROFs to use generalized multiplication gates. A generalized multiplication gate on the inputs $(x_1,\ldots,x_k)$ is allowed to compute any multiplicative ROP in its input variables.

Definition 3.3.4. An alternating read-once formula (AROF) over a field $\mathbb{F}$ in the variables $\bar{x} = (x_1,\ldots,x_n)$ is a tree, of unbounded fan-in, whose leaves are labelled with the input variables and whose internal nodes are labelled with either $+$ or MUL. Each input variable can label at most one leaf. Every leaf and every $+$ gate are labelled with two field elements $(\alpha,\beta) \in \mathbb{F}^2$. In addition, any children of a MUL ($+$) gate is either a leaf or a $+$ (MUL) gate. The computation is performed in the following way. A leaf labelled with the variable $x_i$ and with $(\alpha,\beta)$ computes the polynomial $\alpha x_i + \beta$. If a node $v$, of fan-in $k$, is labelled with $+$ and $(\alpha,\beta)$ and its children compute the polynomials $\Phi_{v_1},\ldots,\Phi_{v_k}$ then the polynomial computed at $v$ is $\Phi_v = \alpha \cdot (\sum_{i=1}^{k} \Phi_{v_i}) + \beta$. If $v$ is labelled with MUL then it computes a multiplicative ROP in its input variables. That is, if $v$ is labelled with the multiplicative

46
ROP $\Psi$, and its children compute the polynomials $\Phi_{v_1}, \ldots, \Phi_{v_k}$, then the output of $v$ will be the polynomial $\Phi_v = \Psi(\Phi_{v_1}, \ldots, \Phi_{v_k})$. The depth of an AROF is defined as the depth of its tree. In other words, the length of the longest path from a leaf to the root.

A preprocessed alternating read-once formula (P-AROF for short) is an AROF $\Phi$ whose leaves are labelled with non-constant univariate polynomials $T_1(x_1), T_2(x_2), \ldots, T_n(x_n)$ (namely, a preprocessing) and the computation is performed as before (in a similar manner to Definition 3.2.31).

**Example 3.3.5.** The polynomial computed in Example 3.2.2 has an AROF of depth 1 that contains a single MUL gate.

**Definition 3.3.6.** For a PROP $P \in \mathbb{F}[x_1, \ldots, x_n]$ we define $\text{depth}(P)$ to be the depth of the shallowest P-AROF computing it.

In fact, it can be shown that all the non-degenerate P-AROFs computing the same PROP have the same depth. We now give the analog of Lemmas 3.2.35 and 3.2.36 for the case of P-AROFs.

**Lemma 3.3.7.** Every PROP $P(x)$ with $|\text{var}(P)| \geq 2$ of depth $D$ can be presented in exactly one of the following forms: $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x}) + \ldots + P_k(\bar{x})$ or $P(\bar{x}) = f(P_1(\bar{x}), P_2(\bar{x}), \ldots, P_k(\bar{x}))$, where the polynomials $\{P_j(\bar{x})\}_{j \in [k]}$ are non-constant, variable-disjoint PROPs of depth at most $D - 1$, and $f$ is a multiplicative ROP.

The proof is similar to the proof of Lemma 3.2.10 so we omit it.

**Lemma 3.3.8.** A partial derivative of a PROP $P(\bar{x})$ of depth $D$ is a PROP of depth at most $D$.

**Proof.** Let $P$ be a PROP of depth $D$, $x_i \in \text{var}(P)$ and $\alpha \in \mathbb{F}$. We prove the lemma by induction on $m = |\text{var}(P)|$. For $m = 0, 1$ the claim is trivial. For $m \geq 2$ we get by Lemma 3.3.7 that $P$ can be in one of two forms.

**Case 1.** $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x}) + \ldots + P_k(\bar{x})$. In this case we get that since the $P_j$-s are variable-disjoint PROPs we can assume w.l.o.g that $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i}$. In addition, $|\text{var}(P_1)| < |\text{var}(P)|$. By the induction hypothesis we get that $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i}$ is a PROP of depth at most $D - 1$.
Case 2. \( P(\bar{x}) = f(P_1(\bar{x}), P_2(\bar{x}), \ldots, P_k(\bar{x})) \), where \( f \) is a multiplicative ROP in \( \{y_1, y_2, \ldots, y_k\} \). Assume w.l.o.g that \( x_i \in \text{var}(P_1) \). By the chain rule we get that \( \frac{\partial P}{\partial x_i} = \frac{\partial f}{\partial y_1}(P_1, \ldots, P_k) \cdot \frac{\partial P_1}{\partial x_i} \). As \( f \) is a multiplicative ROP, we get that \( \frac{\partial f}{\partial y_1} \) is a multiplicative ROP in the variables \( y_2, \ldots, y_k \). In addition, our induction hypothesis implies that \( \frac{\partial P_1}{\partial x_i} \) is a PROP of depth at most \( D - 1 \) (as the depth of \( P_1 \) is at most \( D - 1 \)). As the \( P_j \)-s are variable disjoint it follows that \( \frac{\partial P}{\partial x_i} = \frac{\partial f}{\partial y_1}(P_1, \ldots, P_k) \cdot \frac{\partial P_1}{\partial x_i} \) is a PROP of depth at most \( D \). 

We now give a generator for small depth P-AROFs. The idea is to ‘reduce’ the depth of the formula level by level. In a P-AROF each pair of adjacent levels consists of + and MULT gates. To reduce a + gate, we use Lemma 3.2.37. To reduce a MULT gate, we use the following lemma. Note that in the proof of Lemma 3.3.1 we made an implicit use of Lemma 3.3.9 for the case \( k = 2 \).

**Lemma 3.3.9.** Let \( Q(x_1, \ldots, x_k) : \mathbb{F}^k \to \mathbb{F} \) be a non-constant multiplicative ROP and \( h_1(\bar{y}), \ldots, h_k(\bar{y}) \) be non-constant polynomials. Then \( Q(h_1, \ldots, h_k) \) is a non-constant polynomial.

*Proof.* The proof follows immediately by a simple induction on the structure of the multiplicative ROF for \( Q \). We just notice that the top gate is \( \times \) and by induction the children are non-constant and so their product is non-constant. The base case of the induction is trivial. 

Finally, we can state the depth-version of Lemma 3.3.1.

**Lemma 3.3.10.** Let \( P \in \mathbb{F}[x_1, \ldots, x_n] \) be a non-constant PROP of depth \( \le D \). Then \( P(G_{D+1}) \) is a non-constant polynomial (in particular \( P(G_{D+1}) \neq 0 \)).

*Proof.* We prove the claim by induction on depth \( P \). For depth \( P = 0 \) we get that \( |\text{var}(P)| \le 1 \) and the proof is trivial. Now assume that depth \( P \ge 1 \). This implies \( |\text{var}(P)| \ge 2 \). By Lemma 3.3.7, \( P \) can be written in exactly one of the following two forms.

**Case 1.** \( P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x}) + \ldots + P_k(\bar{x}) \), where the polynomials \( P_j(\bar{x}) \) are non-constant variable-disjoint PROPs of depth at most \( D - 1 \): By the induction hypothesis we see that \( P_1(G_D) \) is a non-constant polynomial. By Lemma 3.2.37 there is a variable \( x_m \in \text{var}(P_1) \) such that even after setting
\( x_i = G_i \) for all other \( i \)'s, \( P \) still depends on \( x_m \). As \( x_m \notin \text{var}(P_j) \) for \( 2 \leq j \leq k \) it follows that \( P(G_D, \ldots, G_{m-1}, x_m, G_{m+1}, \ldots, G_n) \) depends on \( x_m \) as well. By Observation 2.2.7 we get that \( P(G_{D+1}) |_{y_{D+1} = \alpha_n} = P(G_D, \ldots, G_{m-1}, G_m + z_{D+1}, G_{m+1}, \ldots, G_n) \). As \( z_{D+1} \) only appears in the \( m \)-th coordinate it follows that \( P(G_{D+1}) |_{y_{D+1} = \alpha_n} \) depends on \( z_{D+1} \). Therefore, \( P(G_{D+1}) \) is a non-constant polynomial and in particular \( P(G_{D+1}) \neq 0 \).

**Case 2.** \( P(\bar{x}) = f(P_1(\bar{x}), P_2(\bar{x}), \ldots, P_k(\bar{x})) \), where the polynomials \( P_j(\bar{x}) \) are non-constant variable-disjoint PROPs of depth at most \( D - 1 \), and \( f \) is a multiplicative ROP. By applying the induction hypothesis on each \( P_j \) we get that \( P_j(G_{D+1}) \) is a non-constant polynomial, for every \( j \in [k] \). As \( P(G_{D+1}) = f(P_1(G_{D+1}), P_2(G_{D+1}), \ldots, P_k(G_{D+1})) \) it follows from Lemma 3.3.9 that \( P(G_{D+1}) \) is a non-constant polynomial.

We now give an analog of Theorem 3.3.2 that clearly implies Theorem 4, for the case \( k = 1 \). The proof is immediate from Lemma 3.3.10.

**Theorem 3.3.11.** Let \( P \in \mathbb{F}[x_1, \ldots, x_n] \) be a nonzero PROP with individual degrees bounded by \( d \) and depth at most \( D \). Then, there exists an explicit set \( \mathcal{H} \) of size \( |\mathcal{H}| = (nd)^{O(D)} \) such that \( P|_{\mathcal{H}} \neq 0 \).

### 3.4 PIT for Sum of Preprocessed Read-Once Formulas

In this section we prove Theorems 1, 2, 4 and 7. Specifically, we are given \( k \) PROPs \( \{F_m\}_{m \in [k]} \) and we have to find whether they sum to zero or not. In other words, let \( F = F_1 + \ldots + F_k \). The problem is to decide whether \( F \equiv 0 \). Our algorithm for the problem has two steps. First we find a common justifying assignment to \( F_1, \ldots, F_k \) using Algorithm 1. Once we have a common justifying assignment we can assume w.l.o.g. that all the input formulas are \( 0 \)-justified (see Proposition 2.1.4). In the second step we simply verify that \( F \) vanishes on a relatively small set of vectors, each of Hamming weight at most \( 3k \). Theorem 3.4.4 then guarantees that \( F \equiv 0 \). In the black-box version of the algorithm we construct a generator that simulates this process. We now present our main technical contribution.
3.4.1 Hardness of Representation

The main tool in our proof is Theorem 3.4.1 that shows that we cannot represent $\mathcal{P}_n \triangleq \prod_{i=1}^{n} x_i$ as a sum of less than $\frac{1}{3} 3^n \bar{0}$-justified ROPs. We call this approach a *hardness of representation* approach as the proof is based on the fact that a simple polynomial cannot be represented by a sum of a ‘small’ number of $\bar{0}$-justified ROPs. Then, using this preliminary result, we prove a stronger hardness of representation theorem (Theorem 3.4.2) for PROPs. Namely, we show that every nonzero polynomial that has $\mathcal{P}_n$ as a factor, cannot be written as a sum of at most $\frac{n}{3} 0$-justified PROPs. For completeness we give a simple representation of $\mathcal{P}_n$ as a sum of $n 0$-justified ROPs, showing the near optimality of our bound.

**Theorem 3.4.1.** $\mathcal{P}_n(\bar{x})$ cannot be represented as sum of $k \leq \frac{n}{3}$ weakly-$\bar{0}$-justified ROPs.

**Proof.** Let $\{ F_m(\bar{x}) \}_{m \in [k]}$ be $k$ weakly-$\bar{0}$-justified ROPs over $\mathbb{F}[x_1, \ldots, x_n]$. We prove the claim by induction on $k$. For $k = 0, 1$ the claim follows from the definition of weak-$\bar{0}$-justification. We now assume that $k \geq 2$ and that $n \geq 3k$. We shall assume for a contradiction that $\sum_{m=1}^{k} F_m = \mathcal{P}_n$. The idea of the proof is to eliminate “many” ROPs at the cost of a “not too many” variables. Specifically, we find a small set of (indices of) input variables $J \subseteq [n-1]$ and a constant $\alpha \neq 0 \in \mathbb{F}$ such that after we take a partial derivative with respect to all of the variables in $J$ and set $x_n = \alpha$ (that is we consider the ROPs $\{ \partial_j F_m|_{x_n=\alpha} \}_{m \in [k]}$) we eliminate “many” $F_m$-s in a way that the rest of the ROPs remain weakly-$\bar{0}$-justified. We thus get a representation of $\partial_j \mathcal{P}_n|_{x_n=\alpha} = \alpha \cdot \mathcal{P}_n$ (for a relatively large $\hat{n}$) as a sum of a ‘small’ number of weakly-$\bar{0}$-justified ROPs. Then we 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 $\frac{\partial^2 F_m}{\partial x_i \partial x_j} \equiv 0$ (namely, $F_m$ does not contain $x_i \cdot x_j$ in any of its monomials). Assume w.l.o.g. that $i = n-1, j = n$ and $m = k$. By considering the partial derivatives with respect to $\{x_n, x_{n-1}\}$ we see that $\sum_{m=1}^{k-1} \frac{\partial^2 F_m}{\partial x_n \partial x_{n-1}} = \mathcal{P}_{n-2}$. It may be the case that more than one $F_m$ vanishes when we take a partial
derivative w.r.t. \( \{x_n, x_{n-1}\} \), however they cannot all vanish simultaneously (as \( P_n \) contains \( x_n \cdot x_{n-1} \)). By Lemma 3.2.26 we get that the polynomials \( \left\{ \frac{\partial^2 F_m}{\partial x_n \partial x_{n-1}} \right\} \) are weakly-0-justified ROPs. Hence, we obtain a representation of \( P_{n-2} \) as a sum of 0 < \( k \leq k - 1 \) weakly-0-justified ROPs such that 0 < 3\( k \leq 3(k - 1) = 3k - 3 < n - 2 \) which contradicts the induction hypothesis.

**Case 2:** For every \( i \neq j \in [n] \) and \( m \in [k] \) we have that \( \frac{\partial^2 F_m}{\partial x_i \partial x_j} \neq 0 \). Thus, by Lemma 3.2.25 we get that the polynomials \( \{F_m\}_{m \in [k]} \) are multiplicative ROPs. In addition, for every \( m \in [k] \) we have that \( \text{var}(F_m) = [n] \). In particular, \( |\text{var}(F_m)| \geq 6 \). Lemma 3.2.29 implies that \( \forall m \in [k] \) there exist \( j_m \in [n] \), \( \alpha_m \neq 0 \in \mathbb{F} \) and a ROP \( h_m(\bar{x}) \) such that \( \frac{\partial F_m}{\partial x_{j_m}} = (x_n - \alpha_m)h_m(\bar{x}) \). Let \( A = \{\alpha_m \mid m \in [k]\} \). Note that \( 0 \notin A \) as \( P \) is weakly-0-justified. For every \( \alpha \in A \) denote

\[
E_\alpha \triangleq \{ m \in [k] \mid \alpha_m = \alpha \}
\]

and

\[
B_\alpha \triangleq \{ m \in [k] \mid \alpha_m \neq \alpha \text{ and } F_m|_{x_n=\alpha} \text{ is not weakly-0-justified} \}.
\]

Intuitively, \( E_\alpha \) is set of the ROPs that can be eliminated by setting \( x_n = \alpha \) and \( B_\alpha \) is set of (‘bad’) ROPs that will become non weakly-0-justified upon the aforementioned setting and thus require a special treatment. From the definition of \( A \) we have that \( |E_\alpha| \geq 1 \) and \( \sum_{\alpha \in A} |E_\alpha| = k \). More specifically, the \( E_\alpha \)'s form a partition of \([k]\). Similarly, Lemma 3.2.29 implies that for each \( \alpha \neq \alpha' \in A \) the sets \( B_\alpha \) and \( B_{\alpha'} \) are disjoint (since for every ROP there exists at most one bad value \( \beta \) of \( x_n \) and therefore \( \sum_{\alpha \in A} |B_\alpha| \leq k \). Hence, there exists \( \alpha_0 \in A \) such that \( |B_{\alpha_0}| \leq |E_{\alpha_0}| \). Let \( I = E_{\alpha_0} \cup B_{\alpha_0} \subseteq [k] \) and \( J = \{j_m \mid m \in I\} \subseteq [n] \). In addition, \( 1 \leq |J| \leq |I| \leq |E_{\alpha_0}| + |B_{\alpha_0}| \leq 2|E_{\alpha_0}| \) and \( n \notin J \). Consider the following ROPs for every \( m \in [k] \): \( F'_m \triangleq \partial_J F_m \). The \( F'_m \)'s have the following properties:

1. By Lemma 3.2.26 we get that every \( F'_m \) is a weakly-0-justified ROP.

2. For every \( m \in I \) we have that \( F'_m = (x_n - \alpha_m)h'_m(\bar{x}) \) for some ROP
Indeed, as \( j_m \in J \) we have that

\[
F'_m = \partial J F_m = \partial J \{j_m\} \frac{\partial F_m}{\partial x_{j_m}} = \partial J \{j_m\} ((x_n - \alpha_m)h_m(\bar{x}))
\]

\[
= (x_n - \alpha_m) \cdot \partial J \{j_m\} h_m(\bar{x}).
\]

3. For every \( m \in I \) we have that \( h'_m(\bar{x}) \) is a weakly-\( \bar{0} \)-justified ROP (this follows from Lemma 3.2.28 and the previous two properties).

For \( m \in [k] \) consider the following ROPs:

\[
F''_m \overset{\Delta}{=} \partial J F_m \mid x_n = \alpha_0 = F'_m \mid x_n = \alpha_0.
\]

Based on the above we can conclude that:

- For every \( m \in E_{\alpha_0} \) it holds that \( F''_m = (\alpha_0 - \alpha_m)h'_m(\bar{x}) \equiv 0 \) (by definition of \( E_{\alpha_0} \) we have that \( \alpha_m = \alpha_0 \)).

- For every \( m \in B_{\alpha_0} \) we have that \( F''_m = (\alpha_0 - \alpha_m)h'_m(\bar{x}) \) is a nonzero weakly-\( \bar{0} \)-justified ROP. Notice that in contrary to \( F_m \), the structure of \( F'_m \), and the fact that it is weakly-\( \bar{0} \)-justified, guarantees that it remains weakly-\( \bar{0} \)-justified when setting \( x_n = \alpha_0 \).

- For \( m \in [k] \setminus I \) the definitions of \( E_{\alpha_0} \) and \( B_{\alpha_0} \) guarantee that \( F_m \mid x_n = \alpha_0 \) is a weakly-\( \bar{0} \)-justified ROP. Lemma 3.2.26 implies that the same holds for \( F''_m = \partial J (F_m \mid x_n = \alpha_0) \) as well. Note that in this case it is also possible that \( F''_m \equiv 0 \).

Thus, \( F''_m \equiv 0 \) for \( m \in E_{\alpha_0} \) and \( F''_m \) is a weakly-\( \bar{0} \)-justified ROP for \( m \in [k] \setminus E_{\alpha_0} \). W.l.o.g. let us assume that \( J = \{\hat{n} + 1, \hat{n} + 2, \ldots, n - 2, n - 1\} \) for some \( \hat{n} \). We get that \( \sum_{m=1}^{k} F''_m = \partial J P_n \mid x_n = \alpha_0 = \alpha_0 \cdot P_{\hat{n}} \). That is, we found a representation of \( \alpha_0 \cdot P_{\hat{n}} \) as a sum of weakly-\( \bar{0} \)-justified ROPs, where \( |E_{\alpha_0}| \) of the ROPs are zeros. Notice that \( 2 |E_{\alpha_0}| \geq |J| = (n - 1) - \hat{n} \) and \( |E_{\alpha_0}| \geq 1 \). Therefore, we have found a representation of \( \alpha_0 \cdot P_{\hat{n}} \) as a sum of \( 0 \leq \hat{k} < k \) weakly-\( \bar{0} \)-justified ROPs such that

\[
0 \leq 3\hat{k} \leq 3(k - |E_{\alpha}|) = 3k - 3 |E_{\alpha}| \leq n - 3 |E_{\alpha}| \leq \hat{n} + 1 - |E_{\alpha}| \leq \hat{n}.
\]

By our induction hypothesis we get that \( \alpha_0 = 0 \), which is a contradiction (recall that \( \alpha_0 \in A \) and \( 0 \notin A \)). Hence, \( P_{\hat{n}} \) cannot be represented as a sum of less than \( \frac{3}{2} \) weakly-\( \bar{0} \)-justified ROPs. This completes the proof of Theorem 3.4.1. \( \square \)
We now generalize the hardness of representation theorem to the case of PROPs.

**Theorem 3.4.2.** Let \( g(\bar{x}) \not\equiv 0 \) be an arbitrary polynomial. Then, the polynomial \( g(\bar{x}) \cdot P_n(\bar{x}) \) cannot be represented as sum of \( k \) weakly-\( \bar{0} \)-justified PROPs for \( k \leq \frac{n}{3} \).

**Proof.** Let \( \{F_m(\bar{x})\}_{m \in [k]} \) be \( k \) weakly-\( \bar{0} \)-justified PROPs with individual degrees bounded by \( d \) over \( F \) and let \( \{(Q_m(\bar{z}), T_m(\bar{x}))\}_{m \in [k]} \) be their standard decompositions. Recall (Lemma 3.2.34) that \( \{Q_m(\bar{z})\}_{m \in [k]} \) are weakly-\( \bar{0} \)-justified ROPs. Denote \( T^m_i(x_i) = \sum_{j=1}^{d} \alpha_{j,i,m} \cdot x^j_i \). Assume that \( F(\bar{x}) \triangleq \sum_{m=1}^{k} F_m(\bar{x}) = g(\bar{x}) \cdot P_n(\bar{x}). \) Let \( c \cdot \prod_{i=1}^{n} x_i^{e_i} \) be some (nonzero) monomial appearing in \( g(\bar{x}) \). It follows that the monomial \( A = c \cdot \prod_{i=1}^{n} x_i^{e_i+1} \) appears in \( F \). Recall that a standard preprocessing can be chosen up to multiplicative constants. Therefore, we can assume w.l.o.g. that for each \( i \in [n] \) and \( m \in [k] \) we have that \( \alpha_{e_i+1,i,m} = 1 \). Now, since the \( Q_m \)-s are multilinear polynomials we obtain that

\[
c \cdot P_n(x_1^{e_1+1}, x_2^{e_2+1}, \ldots, x_n^{e_n+1}) = c \cdot \prod_{i=1}^{n} x_i^{e_i+1} = A = \sum_{m=1}^{k} Q_m(x_1^{e_1+1}, x_2^{e_2+1}, \ldots, x_n^{e_n+1})
\]

and consequently \( \sum_{m=1}^{k} Q_m(\bar{z}) = c \cdot P_n(\bar{z}) \). By Theorem 3.4.1 it follows that \( n > 3k \). \( \square \)

To complete the picture we show that over a large field (\(|F| > n\)) the polynomial \( P_n(\bar{x}) \) can be represented as a sum of \( n \) \( \bar{0} \)-justified ROPs.

**Lemma 3.4.3.** Let \( F \) be a field with more than \( n \) elements. Then the polynomial \( P_n(\bar{x}) \) can be represented as a sum of \( n \) \( \bar{0} \)-justified ROPs.

**Proof.** Let \( A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq F \setminus \{0\} \) be a subset of \( n \) distinct, nonzero elements. For every \( i \in [n] \) let \( u_i(w) \) be the \( i \)-th Lagrange Interpolation polynomial over \( A \) (see Definition 2.2.6). Let \( \varphi(\bar{x}, t) = (x_1 + t)(x_2 + \ldots) \)
\( t \cdot \cdots (x_n + t) - t^n \). Since the degree of \( t \) in \( \varphi(\bar{x}, t) \) is \( n - 1 \) we get \( \varphi(\bar{x}, t) = \sum_{m=1}^{n} u_m(t) \cdot \varphi(\bar{x}, \alpha_m) \) (i.e., interpolate \( \varphi(\bar{x}, t) \) as a degree \( n - 1 \) polynomial in \( t \)). Consequently, \( \mathcal{P}_n(\bar{x}) = \varphi(\bar{x}, 0) = \sum_{m=1}^{n} u_m(0) \cdot \varphi(\bar{x}, \alpha_m) = \sum_{m=1}^{n} F_m(\bar{x}) \)

where \( F_m(\bar{x}) \equiv u_m(0) \cdot \varphi(\bar{x}, \alpha_m) \) are \( \bar{0} \)-justified ROPs. This completes the proof. \( \square \)

Next, we show that if \( \sum_{m=1}^{k} F_m(\bar{x}) \), a sum of \( \bar{0} \)-justified PROPs, vanishes on a certain small set then this sum is zero.

**Theorem 3.4.4.** Let \( \{F_m(\bar{x})\}_{m \in [k]} \) be \( \bar{0} \)-justified PROPs over \( \mathbb{F} \) with individual degrees bounded by \( d \). Let \( W \subseteq \mathbb{F} \) be a subset of size \( 5^d + 1 \) such that \( 0 \in W \). Let \( F(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x}) \). Then \( F \equiv 0 \) if and only if \( F|_{A_{3k}(W)} \equiv 0 \) (recall the definition in Section 2.2.1).

**Proof.** If \( F \equiv 0 \) then the claim is clear. For the other direction we apply induction on \( n \). Our base case is when \( n \leq 3k \). In this case \( F \) is a polynomial in \( n \leq 3k \) variables of degree at most \( d \) in each variable and therefore by Lemma 2.1.16 we get that \( F|_{A_{3k}(W)} \equiv 0 \) implies that \( F \equiv 0 \). We now assume that \( n > 3k \geq 3 \). Let \( \ell \in [n] \). Consider the restriction of the \( F_m \)-s and \( F \) to the subspace \( x_\ell = 0 \). We now show that the required conditions hold for \( F' \equiv F|_{x_\ell=0} \) and \( \{F'_m \equiv F_m|_{x_\ell=0}\}_{m \in [k]} \) as well. Indeed, the \( \{F'_m\}_{m \in [k]} \) are \( \bar{0} \)-justified PROPs with individual degrees bounded by \( d \). Moreover, \( F'|_{A_{3k}^{n-1}(W)} = F'|_{A_{3k}(W)} \equiv 0 \). From the induction hypothesis we conclude that \( F|_{x_\ell=0} = F' \equiv 0 \) and therefore \( x_\ell \) is a factor of \( F \) (see Lemma 2.1.17). As this holds for every \( \ell \in [n] \) we get that \( \mathcal{P}_n(\bar{x}) \) divides \( F(\bar{x}) \) or equivalently \( F(\bar{x}) = g(\bar{x}) \cdot \mathcal{P}_n(\bar{x}) \) for some \( g(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n] \). It follows that \( g(\bar{x}) \cdot \mathcal{P}_n(\bar{x}) \) is a sum of \( k \) \( \bar{0} \)-justified PROPs. As \( n > 3k \) we get by Theorem 3.4.2 that we must have that \( g(\bar{x}) \equiv 0 \). Hence \( F = g \cdot \mathcal{P}_n \equiv 0 \). This completes the proof of the theorem. \( \square \)

The following is an immediate corollary from Theorem 3.4.4 and Observation 2.2.7.

---

\(^{5}\)We implicitly assume that \(|\mathbb{F}| > d\).
Corollary 3.4.5. In the settings of Theorem 3.4.4 let $\bar{a}$ be a common justifying assignment for the PROPs $F_1, \ldots, F_k$. Then $F(\bar{x}) \equiv 0$ iff $F(\bar{x} + \bar{a})|_{A_{3k}^d(W)} \equiv 0$ and hence $F(\bar{x}) \equiv 0$ iff $F(G_{3k} + \bar{a}) \equiv 0$.

In the next section we show how to get, from a PIT for a single PROF, a common justifying assignment for several PROPs (Algorithm 1). Using this and Corollary 3.4.5 we will get our PIT algorithms.

3.4.2 Non Black-Box Identity Testing Algorithm for Sum of PROPs

In this section we prove Theorem 2. For the algorithm we assume that $|F| > knd$, where $d$ is the bound on the individual degrees of the PROFs.

Algorithm 2 PIT algorithm for sum of preprocessed read-once formulas

\textbf{Input:} PROFs $F_1(\bar{x}), \ldots, F_k(\bar{x})$ with individual degrees bounded by $d$

\textbf{Output:} “true” iff $F(\bar{x}) \Delta \equiv F_1(\bar{x}) + \cdots + F_k(\bar{x}) \equiv 0$

1: Choose $W \subseteq F$ a subset of size $d + 1$, such that $0 \in W$.
2: Find a common justifying assignment $\bar{a}$ for $F_1(\bar{x}), \ldots, F_k(\bar{x})$ \{using Algorithm 1\}.
3: Return “true” if and only if $F(\bar{x} + \bar{a})|_{A_{3k}^d(W)} \equiv 0$.

Lemma 3.4.6. Algorithm 2 runs in time $(nd)^{O(k)}$ and correctly determines whether $F \equiv 0$.

\textbf{Proof.} We start by showing the correctness of the algorithm. If the algorithm does not return “true” then $F(\bar{x} + \bar{a})$ evaluates to a nonzero value which implies that $F(\bar{x} + \bar{a}) \neq 0$ and hence $F(\bar{x}) \neq 0$. If, on the other hand, the algorithm outputs “true”, then $F(\bar{x} + \bar{a})|_{A_{3k}^d(W)} \equiv 0$, where $\bar{a}$ is common justifying assignment for the PROPs $F_1(\bar{x}), \ldots, F_k(\bar{x})$. Corollary 3.4.5 now implies that $F(\bar{x}) \equiv 0$.

To analyze the running time we first recall that given a PROF (explicitly) we can determine whether it computes the zero polynomial in time $O(n)$ by a simple traversal over the formula. Therefore, finding a common justifying assignment $\bar{a}$ for the formulas requires time $O(n^4k^2d)$ (set $T_{C'} = O(n)$ in Lemma 2.3.2). Verifying that $F(\bar{x} + \bar{a})|_{A_{3k}^d(W)} \equiv 0$ re-
quires at most $|A_{3k}^n(W)| \cdot kn$ time. Hence, the running time is at most $kn \cdot (nd)^{O(k)} = (nd)^{O(k)}$ (see Section 2.2.1).

Theorem 2 is an immediate corollary of Lemma 3.4.6.

### 3.4.3 Black-Box Identity Testing Algorithm for Sum of PROPs

In this section we prove Theorems 1, 4 and 7. The idea is to give a generator that, in some sense, simulates Algorithm 2. Specifically, a generator whose image contains a common justifying assignment and the set $A_{3k}^n(W)$ (for an appropriate $W$). For that purpose we use Corollary 2.3.4 of Lemma 2.3.3 that shows how to obtain a justifying set from a generator. Thus, we actually show how to construct a generator for sum of $k$ PROPs from a generator for a single PROF.

**Theorem 3.4.7.** Let $F_1(\vec{x}), \ldots, F_k(\vec{x})$ be PROPs computed by a circuit class $C^g$ such that $F(\vec{x}) \triangleq F_1(\vec{x}) + \ldots + F_k(\vec{x}) \neq 0$. Let $C'$ be a circuit class such that $\partial C \subseteq C'$ and let $\mathcal{G} = (\mathcal{G}^1, \ldots, \mathcal{G}^n) : \mathbb{F}^t \rightarrow \mathbb{F}^n$ be a generator for $C'$. Then $F(G + G_{3k}) \neq 0$. That is, the map $G + G_{3k} : \mathbb{F}^{t+6k} \rightarrow \mathbb{F}^n$, obtained by component-wise addition, is a generator for sums of $k$ PROPs.

**Proof.** By Corollary 2.3.4 there exists $\vec{\gamma} \in \mathbb{F}^t$ such that $\vec{a} = \mathcal{G}(\vec{\gamma})$ is a common justifying assignment of $F_1, \ldots, F_k$. Now, by Corollary 3.4.5 we get: $F(G(\vec{\gamma}) + G_{3k}) \neq 0$. In particular, $F(G + G_{3k}) \neq 0$.

Note that claim holds regardless of the degrees of the PROPs. Using Theorem 3.4.7 and Lemma 2.2.5 we prove Theorems 1, 4 and 7.

**Proof of Theorem 1.** From Lemma 3.3.1 we get that for $\ell = \lceil \log_2 n \rceil + 1$ the mapping $G_\ell : \mathbb{F}^{2\ell} \rightarrow \mathbb{F}^n$ is a generator for PROFs. Lemma 3.2.36 implies that PROFs are closed under partial derivatives. Hence, by Theorem 3.4.7 we get that the mapping $G_{\ell+3k}$ is a generator for sum of $k$ PROFs. The hitting set produced by Lemma 2.2.5 is of size $|\mathcal{H}| = (n^2d)^{(6k+2\ell)} = (nd)^{O(k+\log n)}$.

The next case is when all the $F_m$-s are bounded depth PROFs.

---

Note that we can let $C$ be the class of PROFs, however we give the more general statement in order to apply it for models for which we have a more efficient generator than the one for PROFs.
Proof of Theorem 4. Lemma 3.3.10 implies that the mapping $G_\ell : \mathbb{F}^{2\ell} \to \mathbb{F}^n$, for $\ell = D + 1$, is a generator for depth-$D$ PROFs. By Lemma 3.3.8, this circuit class is closed under partial derivatives. Therefore, it follows from Theorem 3.4.7 that $G_{\ell+3k}$ is a generator for sum of $k$ PROFs of depth at most $D$. Lemma 2.2.5 now gives a hitting set of size $|\mathcal{H}| = (n^2d)^{(6k+2\ell)} = (nd)^{(D+k)}$.

The last result in this vein is a black-box PIT algorithm for the case where the black-box holds a sum of PROFs that is a read-r (i.e., every variable appears in at most $r$ PROFs).

Definition 3.4.8. Let $\{F_m\}_{m \in [m]}$ be PROFs. We say that $F = \sum_{m=1}^k F_m$ is a read-r sum if for each $i \in [n]$ there are at most $r$ functions $F_m$ that depend on $x_i$. In other words, each variable is read at most $r$ times in $F$.

We can easily extend a PIT algorithm for sum of $r$ PROFs to a PIT algorithm for a read-r sum with the following observation.

Observation 3.4.9. Let $F$ be a read-r sum. Then $\frac{\partial F}{\partial x_i}$ is a sum of (at most) $r$ PROFs, for every $i \in [n]$ and each $\alpha \in \mathbb{F}$.

Proof of Theorem 7. Given a read-r sum $F$ we can, by Lemma 2.3.1, find $\text{var}(F)$. Observation 3.4.9 implies that we can use Theorems 1, 2, and 4 as the corresponding PIT algorithm.

3.5 Depth-3 $\Sigma\Pi\Sigma(k)$ Circuits

In this section we give a new black-box PIT algorithm for depth-3 circuits based on the hardness of representation approach. We also derive a new PIT algorithm for multilinear depth-3 circuits and a special case of depth-4 circuits based on Theorem 4.

Definition 3.5.1. A linear function over a field $\mathbb{F}$ is a polynomial of the form $L(\bar{x}) = \sum_{i=1}^n b_i x_i + b_0$, where $\forall i b_i \in \mathbb{F}$. A polynomial $P(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ is a linear product if it is a product of linear functions: $P(\bar{x}) = \prod_j L_j(\bar{x})$ where each $L_j(\bar{x})$ is a linear function.
Definition 3.5.2. A depth-$3$ $\Sigma \Pi \Sigma (k)$ circuit $C$ computes a polynomial of the form
\[ C(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x}) = \sum_{m=1}^{k} d_m \prod_{j=1}^{L} L_{m,j}(\bar{x}), \]
where each $L_{m,j}(\bar{x})$ is a linear function. The $F_m$-s are the multiplication gates of the circuit. Note that the $F_m$-s are, in fact, linear products. We denote by $\Sigma \Pi \Sigma (k,d)$ a $\Sigma \Pi \Sigma (k)$ circuit in which each multiplication gate has degree at most $d$, i.e. $d_m \leq d$ for every $m$. A multilinear $\Sigma \Pi \Sigma (k)$ circuit is a $\Sigma \Pi \Sigma (k)$ circuit in which each $F_m$ is a multilinear polynomial. Note, that in this case the degree is at most $n$. Furthermore, in the multilinear case each $F_m$ is a ROP.

As before, we shall also consider preprocessed $\Sigma \Pi \Sigma (k)$ circuits that form a special subclass of depth-4 circuits. The definition is similar to the way PROFs are generated from ROFs.

Definition 3.5.3. A preprocessed linear function is a polynomial of the form $L(\bar{x}) = \sum_{i=1}^{n} T_i(x_i)$, where each $T_i(x_i)$ is a univariate polynomial. A polynomial $F(\bar{x})$ is a preprocessed linear product if it is a product of preprocessed linear functions. A Preprocessed $\Sigma \Pi \Sigma (k)$ computes a polynomial of the form: $C(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x})$, where the $F_m$-s are preprocessed linear products.

As a corollary of Theorem 4 we obtain a PIT algorithm for preprocessed multilinear depth-3 circuits (Theorem 5). Indeed, in a multilinear $\Sigma \Pi \Sigma (k)$ circuit each multiplication gate is a depth-2 ROP. Therefore, a preprocessed multilinear $\Sigma \Pi \Sigma (k)$ circuit is actually a sum of $k$ depth-2 PROPs. Having this in mind we can apply the results of Section 3.4 (i.e. Theorems 3.4.2 and 3.4.4, and Theorem 4).\footnote{Theorem 3.4.2 is tight for multilinear depth-3 circuits since Lemma 3.4.3, in fact, gives a representation of $P_n$ as a sum of $n$ 0-justified linear products and a constant. By a slightly more sophisticated argument one can get rid of the constant.} Thus, Theorem 5 is in fact an immediate corollary.

Proof of Theorem 5. By the above discussion, a preprocessed multilinear $\Sigma \Pi \Sigma (k)$ circuit is a sum of $k$ depth-2 PROPs with the same individual degrees. The result follows from Theorem 4. \qed
We now describe a new PIT algorithm for general \( \Sigma \Pi \Sigma (k) \) circuits. Before presenting our algorithm we give several notations (originally defined in [17]) and discuss related results.

**Definition 3.5.4.** Let \( C(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x}) \) be a \( \Sigma \Pi \Sigma (k) \) circuit. We say that \( C \) is minimal if no subset of the multiplication gates sums to zero. We define gcd\((C)\) as the linear product of all the non-constant linear functions that belong to all the \( F_m \)-s. I.e. gcd\((C) = \gcd(F_1, \ldots, F_k)\). We say that \( C \) is simple if gcd\((C) = 1\). The simplification of \( C \), denoted by sim\((C)\), is defined as \( C / \gcd(C) \). Note that if \( C \) is a \( \Sigma \Pi \Sigma (k,d) \) then so is sim\((C)\).

Let rank\((C)\) be defined as the dimension of the span of the linear functions in \( C \) when viewed as \((n+1)\)-dimensional vectors over \( \mathbb{F}^{n+1} \).

In [17] it was proved that the rank of a \( \Sigma \Pi \Sigma (k) \) circuit computing the identically zero polynomial cannot be too large. Specifically, there exists a non-decreasing function \( R(k,d) \) such that the dimension of the linear space spanned by all the linear functions in a simple and minimal circuit is at most \( R(k,d) \). Originally, \( R(k,d) \) was shown to be \( 2^{O(k^2)(\log d)^{k-2}} \). Recently, this bound was improved in [40, 58, 59]. We give now the best known upper bounds.

**Theorem 3.5.5 ([59]).** There exists an non-decreasing function \( R(k,d) \) such that \( \text{rank}(C) < R(k,d) \) for any simple and minimal \( \Sigma \Pi \Sigma (k,d) \) circuit \( C \) over a field \( \mathbb{F} \). Furthermore, \( R(k,d) = O(k^2 \log d) \) for any field. If \( \mathbb{F} = \mathbb{R} \) or \( \mathbb{F} = \mathbb{Q} \) then \( R(k,d) = O(k^2) \).

The heart of all existing black-box PIT algorithm for \( \Sigma \Pi \Sigma (k) \) circuits was the result of [37] that gave a PIT algorithm whose running time depends exponentially on \( R(k,d) \), in particular equals to \( \text{poly}(n) \cdot d^{R(k,d)} \). As a result, [59] got the following corollary (which is weaker than the recent result of [60]).

**Theorem 3.5.6 ([59]).** There is a deterministic black-box PIT algorithm for \( \Sigma \Pi \Sigma (k,d) \) circuits over \( \mathbb{F} \) that runs in time \( \text{poly}(n) \cdot d^{R(k,d)} = \text{poly}(n) \cdot d^{O(k^2 \log d)} \). If \( \mathbb{F} = \mathbb{R} \) or \( \mathbb{F} = \mathbb{Q} \), then the algorithm runs in time \( \text{poly}(n) \cdot d^{O(k^2)} \).

\(^8\)The result of [60], giving a poly\((n,d^k)\) time black-box PIT algorithm, applies the same generator of [37] but relies on a completely different analysis. [60] appeared after the initial publication of our work.

59
On the other hand, in the non black-box setting there is a poly$(n,d^k)$ time PIT algorithm.

**Theorem 3.5.7** ([41]). There is a deterministic non black-box PIT algorithm for $\Sigma\Pi\Sigma(k,d)$ circuits that runs in time poly$(n,d^k)$.

### 3.5.1 New Black-Box PIT Algorithm for Depth-3 Circuits

In this section we give a different black-box PIT algorithm for $\Sigma\Pi\Sigma(k,d)$ circuits based on the recent results of [58, 59] using our hardness of representation approach. For this we will use a result of [58] that generalizes Theorem 3.5.5, giving an upper bound on the rank of the linear factors of a polynomial that is computed by a simple, minimal and nonzero $\Sigma\Pi\Sigma(k,d)$ circuit.\(^9\)

**Definition 3.5.8.** Let $P(\bar{x}) = h_1(\bar{x}) \cdot h_2(\bar{x}) \cdots h_t(\bar{x})$ be a nonzero polynomial and its irreducible factors, respectively. We denote by $\text{Lin}(P)$ the set of (non-constant) linear factors of $P$. Formally, $\text{Lin}(P) \triangleq \{h_i \mid h_i \text{ is a linear factor of } P\}$.

**Lemma 3.5.9** ([58]). Let $P(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial computed by a simple, minimal and nonzero $\Sigma\Pi\Sigma(k,d)$ circuit. Then, $\text{rank}(\text{Lin}(P)) < R(k,d)$.

We now give an efficient PIT algorithm using techniques from Section 3.4.

**Lemma 3.5.10.** The mapping $G_1(y_1, z_1)$ is a generator for preprocessed linear products.

**Proof.** From Observation 2.2.2 it is sufficient to show that the claim holds for a single non-constant preprocessed linear function $L(\bar{x}) = \sum_{i=1}^{n} T_i(x_i)$. From the definition, there exists $j$ such that $T_j(x_j)$ is a non-constant polynomial. As $L(G_1(\alpha_j, z_1)) = \sum_{i \neq j} T_i(0) + T_j(z_1)$ (see Observation 2.2.7) we get that $L(G_1(y_1, z_1))$ is a non-constant polynomial. \(\square\)

\(^9\)This result appears only in [58]. However, the proof of [59] can be easily extended to this case as well.
**Observation 3.5.11.** Let $F(\bar{x}) = \prod_j L_j(\bar{x})$ be a (preprocessed) linear product with $j \geq 2$. Then $F$ is $\bar{0}$-justified iff $L_j(\bar{0}) \neq 0$ for each $j$.

Lemma 3.5.9 allows us to establish a hardness of representation theorem and an analog of Theorem 3.4.4. for $\Sigma \Pi \Sigma(k,d)$ circuits.

**Theorem 3.5.12.** Let $g(\bar{x}) \neq 0$ be an arbitrary polynomial. Then, the polynomial $g(\bar{x}) \cdot \mathcal{P}_n(\bar{x})$ cannot be represented as a sum of $k$ $\bar{0}$-justified linear products of (total) degree $d$ when $n > R(k,d)$.

**Proof.** Assume that $C = \sum_{m=1}^{k} F_m$ computes $g(x) \cdot \mathcal{P}_n$. Furthermore, assume, w.l.o.g., that $C$ is minimal and that $n \geq 2$. As all the linear functions in $C$ are $\bar{0}$-justified, Observation 3.5.11 implies that $x_i \notin \gcd(C)$, for any $i \in [n]$. Therefore, $\gcd(C, \mathcal{P}_n) = 1$. Consequently, if we consider the simplification of $C$ we get $C' \triangleq \text{sim}(C) = g'(x) \cdot \mathcal{P}_n(x)$, where $g' \neq 0$. That is, $g' \cdot \mathcal{P}_n$ is computed by a simple, minimal, nonzero circuit $C'$ ($C'$ remains minimal after the simplification). Hence, for every $i \in [n]$, $x_i$ is a linear factor of $C'$. Lemma 3.5.9 implies that $n \leq \text{rank}(\text{Lin}(g' \cdot \mathcal{P}_n)) \leq R(k,d)$, as required. \qed

As before, the following theorems follow immediately from Theorem 3.4.4.

**Theorem 3.5.13.** Let $C(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x})$, be of total degree at most $d$, where each $F_m(\bar{x})$ is a $\bar{0}$-justified linear product over $\mathbb{F}$. Let $W \subseteq \mathbb{F}$ be of size $d + 1$, where $0 \in W$. Then $C \equiv 0$ iff $C|_{\mathcal{A}_{R(k,d)}(W)} \equiv 0$.

Finally, in a similar fashion to the proof of Theorem 3.4.7, we obtain the following theorem that implies Theorem 6.

**Theorem 3.5.14.** Let $C$ be a $\Sigma \Pi \Sigma(k,d)$ circuit over $\mathbb{F}$ then $C(G_{R(k,d)+1}) \neq 0$.

Lemma 2.2.5 gives a hitting set for such circuits of size $|\mathcal{H}| = (nd)^{O(R(k,d))} = (nd)^{O(k^2 \log d)}$. When $\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{Q}$ we get $|\mathcal{H}| = (nd)^{O(R(k,d))} = (nd)^{O(k^2)}$. (Recall Theorem 3.5.5). Note, that we achieve roughly the same running times as in Theorem 3.5.6.
Chapter 4

PIT for Depth-4 Circuits

We study the problem of identity testing for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits, i.e. multilinear depth-4 circuits with fan-in $k$ at the top $+$ gate. We give the first polynomial-time deterministic identity testing algorithm for such circuits. Our results also hold in the black-box setting.

The running time of our algorithm is $(ns)^{O(k^3)}$, where $n$ is the number of variables, $s$ is the size of the circuit and $k$ is the fan-in of the top gate. The importance of this model arises from [3], where it was shown that derandomizing black-box polynomial identity testing for general depth-4 circuits implies a derandomization of polynomial identity testing (PIT) for general arithmetic circuits. Prior to our work, the best PIT algorithm for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits [36] ran in quasi-polynomial-time, with the running time being $n^{O(k^6 \log(k) \log^2(s))}$.

We obtain our results by showing a strong structural result for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits that compute the zero polynomial. We show that under some mild technical conditions, any gate of such a circuit must compute a sparse polynomial. We then show how to combine the structure theorem with a result by Klivans and Spielman [43], on the identity testing for sparse polynomials, to yield the full result.

The results of this chapter are based on the work [56].
4.1 Introduction

In the current chapter, we study multilinear depth-4 $\Sigma\Pi\Sigma\Pi(k)$ circuits. Very recently Karnin et al. [36] gave the first deterministic subexponential-time (in fact, quasi-polynomial-time) PIT algorithm for this model. Their result was in the black-box setting. We further investigate this class of circuits and give the first deterministic polynomial-time black-box PIT algorithm for it. Our approach is quite different from that taken in [36], and we believe that the techniques might be useful to understand other, more general classes of circuits as well.

Following the same approach as in [3], it can be shown that derandomizing black-box identity testing for multilinear depth-4 ($\Sigma\Pi\Sigma\Pi$) circuits implies an exponential lower bound for general multilinear arithmetic circuits. Getting explicit lower bounds is one of the biggest challenges of complexity theory and has been the focus of much research. So far, the best known lower bounds are: $\Omega(n^{4/3}/\log^2 n)$ for multilinear circuits due to Raz et al. [53], and $n^{\Omega(\log n)}$ for multilinear formulas due to Raz [52]. It is an interesting open question to improve any of those bounds. All the above makes the study of PIT for depth-4 circuits, even in the multilinear case, a really interesting and challenging open question.

We now define the model of multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits formally. Similar definitions were given in [36], however we repeat them for the sake of completeness. A depth-4 circuit has 4 layers of alternating $(+, \times)$ gates and it computes a polynomial of the form

$$C(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{k} F_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}$$

where $k$ is the fan-in of the top $\Sigma$ gate and $d_i$ are the fan-ins of the $\Pi$ gates at the second level. We refer to $F_i$-s as the multiplication gates and $P_{ij}$-s are the polynomials computed at the third level of the circuit (which is a $\Sigma\Pi$ component). This implies that if the size of $C$ is $s$ then, clearly, each $P_{ij}$ in $C$ can be computed by a depth-2 ($\Sigma\Pi$) circuit of size at most $s$. Such polynomials are known as s-sparse $\Sigma\Pi$ circuit of size at most $s$ non-zero monomials (see e.g [11, 43, 45]). In other words, each $P_{ij}$ is s-sparse. We define $\gcd(C) \triangleq \gcd(F_1, \ldots, F_k)$, that is, the gcd of the set of
polynomials computed by the multiplication gates. We say that $C$ is simple if $\text{gcd}(C) = 1$. A $\Sigma\Pi\Sigma\Pi(k)$ circuit is called minimal if for every proper subset $\emptyset \subsetneq A \subsetneq [k]$, the corresponding subcircuit $C_A \triangleq \sum_{i \in A} F_i$ of $C$ is non-zero. Multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits are circuits in which the fan-in of the top $\Sigma$ gate is a constant $k$ and each multiplication gate $F_i$ computes a multilinear polynomial. We say that a polynomial is $s$-dense if it contains more than $s$ monomials.

The main technical contribution in the proof of both black-box and non-black-box identity testing algorithms for $\Sigma\Pi\Sigma\Pi(k)$ circuits is a new structural theorem for identically zero multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits. We refer to it as the Sparsity Bound. This result can be viewed as a natural (though unsuspected) generalization of the previously shown structural theorems for depth-3 $\Sigma\Pi\Sigma(k)$ circuits known as the Rank Bound (see Section 4.1.3). Our result lends optimism to the hope that similar structural results should also hold for general $\Sigma\Pi\Sigma\Pi(k)$ circuits (without the restriction of multilinearity).

At a very high level, we show that the only way a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit can completely cancel itself out and compute the zero polynomial is that the circuit must not be computing very “complex” polynomials. In particular, we show that in any simple and minimal multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit that computes the identically zero polynomial, the polynomials computed at the multiplication gates must be sparse.

**Theorem 8** (Sparsity Bound for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits). Let $k \geq 2$ and let $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x})$ be a simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size $s$ computing the zero polynomial. Then each $F_i$ is $s^{O(k^2)}$-sparse.

One way to interpret the theorem is as follows: for a fixed $k$ the sparsity of each multiplication gate in a simple and minimal, identically zero multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit is at most polynomially large in the size of the circuit. Note, that for general circuits this sparsity can be exponentially large. Later on (Section 4.4.1) we show a lower bound on the multiplication gate’s sparsity indicating that our result is nearly optimal. Once we have the structure theorem we exploit it to design PIT algorithms in both black-box and non-black-box settings, thus proving the following theorems.

**Theorem 9** (Black-Box PIT for $\Sigma\Pi\Sigma\Pi(k)$ circuits). Let $k, n, s$ be integers. There is an explicit set $\mathcal{H}$ of size $n^{O(k)} \cdot s^{O(k^3)}$, that can be constructed in
time $n^{O(k)} \cdot s^{O(k^3)}$, such that the following holds. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial computed by a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size $s$ on $n$ variables. Then $P|_H \not\equiv 0$.

In our construction we heavily use the black-box PIT algorithm of [43] for sparse polynomials as a subroutine. Using their PIT algorithm we show how to make any “non sparse” circuit into a “somewhat sparse” circuit. Our structure theorem then guarantees that within this process, we do not inadvertently end up making a non-zero circuit into a zero circuit. Once we have a “somewhat sparse” non-zero circuit, we use the above PIT algorithm coupled with some techniques from [36] to find a non-zero evaluation point for it, and hence get a black-box identity tester. In the non black-box setting (e.g. when the circuit is given to us explicitly) we get a slight improvement in the running time.

**Theorem 10** (Non Black-Box PIT for $\Sigma\Pi\Sigma\Pi(k)$ circuits). Let $k, n, s$ be integers. Given a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit $C$ of size $s$ computing a polynomial over $\mathbb{F}[x_1, \ldots, x_n]$ there exists an algorithm that runs in time $\text{poly}(n) \cdot s^{O(k^2)}$ and determines whether $C \equiv 0$.

### 4.1.1 Overview of the Proof of The Structure Theorem

As mentioned earlier, our algorithm is based on a new structure theorem for simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits $C = \sum_{i=1}^{k} F_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}$. We now give an overview of the proof of the structural theorem. We wish to find an upper bound on the sparsity (i.e. the number of non-zero monomials) of the polynomials computed by the multiplication gates ($F_i$-s). At a high level, our strategy will be to set a multiplication gate to zero (thus obtaining a circuit with less multiplication gates) and then to use an inductive argument on the resulting circuit. To do so we will find a partial zero assignment $\bar{a}$ to some $P_{ij}$ (i.e. $P_{ij}(\bar{a}) = 0$) and substitute it into $C$. Let $C' = \sum_{i=1}^{k} F'_i$ be the resulting circuit obtained by the substitution of $\bar{a}$. First of all, note that $C'$ may not satisfy the conditions of the inductive claim. In other words, the substitution may compromise either simplicity or minimality of the circuit (or both of them). Furthermore, note that a

---

1In fact, our construction works with any black-box PIT algorithm for sparse polynomials.
partial substitution may decrease the sparsity of the multiplication gates. That is, for some $i \in [k]$ the sparsity of $F'_i$ might be much smaller than the sparsity of $F_i$. The main issue here is that it is not clear how to bound this difference in sparsity accurately enough. Hence, any upper bound obtained on the sparsity of the gates of $C'$ may not provide us with any useful information on the sparsity of the gates in the original circuit $C$. This makes the offered strategy problematic.

Nevertheless, we show that we can still work around these problems. Our main idea behind effectively bounding the sparsity is that, instead of identifying a single $P_{ij}$ and setting it to zero, we will go over all $P_{ij}$-s and set them to zero one at a time. Once we select a $P_{ij}$ to set to zero, we will look for a zero assignment of $P_{ij}$ that preserves certain special properties of $C$. More specifically, we will find a zero assignment of $P_{ij}$ such that after the substitution, the resulting circuit $C'$ is simple and minimal, and in addition, the aforementioned decrease in sparsity is brought to a minimum. In order to find such an assignment we construct a polynomial $\Phi$ such that the above conditions (e.g. “simplicity”, “minimality” and “sparsity difference minimization”) can be formulated in terms of being a non-zero assignment of $\Phi$. Consequently, the desired assignment $\bar{a}$ will be a zero assignment of $P_{ij}$ which is simultaneously also a non-zero assignment of $\Phi$. However, such an assignment may not even exist (for example if $P_{ij}$ is a factor of $\Phi$). To handle this problem we introduce a new technique. In fact, we settle on finding a zero assignment $\bar{a}$ of $P_{ij}$ which is “almost good”, paying some “small price” for it. The main issue turns out to be the estimation of the sparsity difference that results from the partial substitution (i.e. $\text{Sparsity}(F_i) - \text{Sparsity}(F'_i)$). For this purpose we use Shearer’s Lemma (see Lemma 4.2.5). To apply the lemma, we map each multilinear polynomial $P$ to a family of sets corresponding to $P$’s non-zero monomials (see Definition 4.2.1). Note that the lemma suggests that the more distinct partial substitutions we have, the tighter is the bound. This is the reason why we combine the information received from all different partial substitutions to $C$ (i.e. by trying out all the $P_{ij}$).
Our black-box algorithm is based on the Sparsity Bound. For the moment, let us ignore the issues of simplicity and minimality. These do create issues which will have to be addressed, but just to understand the motivation for the algorithm suppose that we were given a simple and minimal $\Sigma \Pi \Sigma \Pi(k)$ circuit $C$. There can be two cases: either (i) each multiplication gate of $C$ is sparse (e.g. the circuit is “sparse”) or (ii) $C$ has a dense multiplication gate (e.g. the circuit is “dense”). In case (i) the polynomial computed by $C$ is sparse (as a sum of a small number of sparse polynomials) therefore one can invoke a PIT algorithm for sparse polynomials to check if $C \equiv 0$. In case (ii) the Sparsity Bound implies that $C \not\equiv 0$. This observation gives rise to a non black-box PIT algorithm: Compute the sparsity of each gate separately. If there is a dense gate, then conclude that $C \not\equiv 0$. Otherwise, check whether the monomials cancel each other out. Again, the conditions of simplicity and minimality create some issues, but they can be dealt with (for more details see Section 4.6).

The black-box setting is trickier, since we do not get to “see” the circuit, and hence cannot carry out the procedure described above. Indeed, the main problem is to decide in which of the cases we are. Our strategy will be to walk on the edge between the two cases. Given a non-zero circuit $C$ we are going to gradually reduce the sparsities of the gates, step-by-step, until we reach case (i), while preserving the properties of $C$ (simplicity and minimality). In each step the sparsities of gates will reduce by a “small” factor. Through the entire process the Sparsity Bound will guarantee that $C$ remains non-zero as long as we are in case (ii). Now, let us consider the “edge step” that is, the last reduction step before we reach case (i). We claim that in this step the circuit is dense, but not “too dense”. On one hand, we are still in case (ii) where the Sparsity Bound guarantees that $C \not\equiv 0$. But on the other hand, a reduction by a “small” factor makes the circuit sparse. Hence, we can conclude that the circuit is non-zero and “somewhat sparse”. To make the argument a bit more formal, consider a non-zero $\Sigma \Pi \Sigma \Pi(k)$ circuit $C$ of size $s$. If $C$ is $s^{O(k^2)}$-sparse, then we are done. Otherwise, let $\bar{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$ be an assignment such that for every $1 \leq t \leq n$ the circuit $C(a_1, \ldots, a_t, x_{t+1}, \ldots, x_n)$ - resulting from setting $x_j = a_j$ for $1 \leq j \leq t$ - is simple and minimal. In particular, $F_i(\bar{a}) \neq 0$ for each $1 \leq i \leq k$, and hence
each $F_i(\bar{a})$ is 1-sparse. Let $0 \leq t \leq n$ be the largest index such that the circuit $C(a_1, \ldots, a_t, x_{t+1}, \ldots, x_n)$ has an $s^{O(k^2)}$-dense multiplication gate. Such an index exists by a hybrid argument. From the Sparsity Bound, we get that $C(a_1, \ldots, a_t, x_{t+1}, \ldots, x_n) \neq 0$. One the other hand, from the choice of $t$ we get that each multiplication gate in $C(a_1, \ldots, a_{t+1}, x_{t+2}, \ldots, x_n)$ is $s^{O(k^2)}$-sparse. Observe that since each $F_i$ is multilinear all the $P_{ij}$-s in it must be variable disjoint. Thus, setting $x_{t+1} = a_{t+1}$ can affect at most one $P_{ij}$ in each $F_i$, and hence reduce the sparsity of each $F_i$ by a factor of at most $s$ (recall that each $P_{ij}$ is $s$-sparse and that $P_{ij}(\bar{a}) \neq 0$). Consequently, each multiplication gate in $C(a_1, \ldots, a_t, x_{t+1}, \ldots, x_n)$ is $s \cdot s^{O(k^2)}$-sparse. A priori we do not know what this index $t$ is. However, it turns out not to matter, since we can just run over all possible indices.

In all this discussion, we hid many technical issues under the rug such as the simplicity and minimality of the original circuits and how to find such an assignment $\bar{a}$. We note that in the black-box setting minimality comes for free since we can assume w.l.o.g that the black-box contains a minimal circuit. To handle simplicity and find an assignment $\bar{a}$ as above we use the method of generators (see Section 2.2). In [36] this method has been already used to work around the aforementioned problems and to ensure that all the steps of the proof go through smoothly.

4.1.3 Related Previous Results

Since the vast majority of the work on low depth circuits focused on circuits of depths 2 and 3, and since our results can be viewed as an extension of this body of work, we formally define depth-3 circuits and some related and relevant notions. A depth-3 $\Sigma\Pi\Sigma(k)$ circuit $C$ of degree $d$ computes a polynomial of the form

$$C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} L_{ij}(\bar{x})$$

where the $L_{ij}(\bar{x})$-s are linear functions: $L_{ij}(\bar{x}) = \sum_{t=1}^{n} a_{ij}^t x_t + a_{ij}^0$ with $a_{ij}^t \in \mathbb{F}$, and $d_i \leq d$. Note that $L_{ij}$-s are irreducible polynomials. A multilinear $\Sigma\Pi\Sigma(k)$ circuit has the additional requirement that each $F_i$ is a multilinear polynomial. We refer to the $F_i$-s as the multiplication gates of the circuit.
A subcircuit of $C$ is defined as a sum of a subset of the multiplication gates in $C$. Let $\gcd(C) \overset{\Delta}{=} \gcd(F_1, F_2, \ldots, F_k)$. We say that a circuit is simple if $\gcd(C) = 1$. We say that a circuit is minimal if no proper subcircuit of $C$ computes the zero polynomial. Define the rank of $C$, denoted by $\text{rank}(C)$, as the rank of its linear functions, viewed as $(n+1)$-dimensional vectors over $\mathbb{F}^{n+1}$. Formally: $\text{rank}(C) \overset{\Delta}{=} \dim(\text{span}\{L_{ij}\}_{i \in [k], j \in [d_i]})$. Clearly, $\Sigma \Pi \Sigma(k)$ circuits are a restricted case of $\Sigma \Pi \Sigma \Pi(k)$ circuits.

The first non black-box [17] and almost all the black-box PIT algorithms for $\Sigma \Pi \Sigma(k)$ circuits [37, 40, 58, 59] were designed based on the following structural property of $\Sigma \Pi \Sigma(k)$ known as the Rank Bound.

**Lemma 4.1.1.** There exists an increasing function $R(k, d)$ such that if $C$ is a simple and minimal $\Sigma \Pi \Sigma(k)$ circuit of degree $d$ computes the zero polynomial then $\text{rank}(C) < R(k, d)$.

Up until recently, all the resulting black-box PIT algorithms for $\Sigma \Pi \Sigma(k)$ circuits were based on the black-box PIT algorithm of [37] that admits a running time of $\text{poly}(n) \cdot d^{O(R(k, d))}$. The recent result of [60], giving a $\text{poly}(n) \cdot d^{O(k)}$ time black-box PIT algorithm, was obtained by a different approach. In the non black-box setting Kayal & Saxena [41] presented a $\text{poly}(n) \cdot d^{O(k)}$ time PIT algorithm by another methodology.

The first result of Dvir & Shpilka [17] gave an upper bound of $R(k, d) = 2^{O(k^2) \log k^2 - d}$ over general fields. It was later improved by Saxena & Seshadhri [58, 59] to $R(k, d) = O(k^2 \log d)$. Based on an example of [41] they, however, also illustrated a limitation of this approach by exhibiting a lower bound of $\text{rank}(C) = \Omega(k \log d)$ over finite fields, implying that the best black-box PIT algorithm for finite fields achieved via this approach will be quasi-polynomial-time for constant values of $k$. However, over the field of reals $\mathbb{R}$ the bound was significantly improved by Kayal & Saraf [40] to $k^{O(k)}$ and later on by Saxena & Seshadhri [59] to $R(k) = O(k^3)$. Note that there is no dependence on $d$, thus implying a polynomial-time algorithm.

For multilinear $\Sigma \Pi \Sigma(k)$ circuits the best upper bound of $R_{\text{ML}}(k) = O(k^2 \log k)$ for general fields was shown in [59]. Yet, the PIT algorithm with the best running time of $n^{O(k)}$ obtained in Theorem 5 does not rely on the Rank Bound.

Much less is understood about depth-4 circuits. The existing determin-
istic PIT algorithms had covered only very restricted classes of those circuits \cite{57, 65, 9}. The first PIT algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits was given by Karnin et al. in \cite{36}. The algorithm is in the black-box setting and has a running time of $n^{O(k^3 \cdot R_{ML}(k) \cdot \log^2 s)}$, when $s$ is the size of the circuit and $R_{ML}(k)$ is the multilinear Rank Bound (see above). This implies a quasi-polynomial-time algorithm for constant values of $k$. The main ingredient of the algorithm is a new structural theorem for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits suggesting that there is a non-zero multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit "hiding" in every non-zero multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit. The idea is to "search" for the hidden $\Sigma\Pi\Sigma(k)$ circuit using the multilinear Rank Bound. This search is what makes the algorithm quasi-polynomial.

The Rank Bound relies strongly on the properties of linear functions. That is, one can define a linear space spanned by the circuit components and benefit from its structure. This notion is absent when moving to depth-4 circuits. In the current paper we suggest a natural generalization of this notion - the Sparsity Bound. The idea is to bound the sparsities of the polynomials computed by each multiplication gate in terms of the sparsities of the circuit components ($P_{ij}$-s). This approach can be seen as an extension of the Rank Bound approach. The following lemma demonstrates this point and, in fact, can be seen as a “sanity check”.

**Lemma 4.1.2.** Let $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} L_{ij}(\bar{x})$ be a simple and minimal, multilinear $\Sigma\Pi\Sigma(k)$ circuit computing the zero polynomial. Let $s$ denote the maximal sparsity of a $L_{ij}$ appearing in $C$. Then each $F_i$ is $s^{R_{ML}(k)}$-sparse.

**Proof.** Fix $i \in [k]$ and consider $F_i = \prod_{j=1}^{d_i} L_{ij}(\bar{x})$. By definition, each $F_i$ is $s^{d_i}$-sparse. As $F_i$ is a multilinear polynomial, it must be the case that all the $L_{ij}$-s of it are variable-disjoint and hence linearly independent. Therefore, by the Rank Bound: $d_i = \dim \left( \operatorname{span}\{L_{ij}\}_{j \in [d_i]} \right) \leq \operatorname{rank}(C) < R_{ML}(k)$ and whence $F_i$ is $s^{R_{ML}(k)}$-sparse.

Finally, we note that in the light of the connections between deterministic PIT and circuit lower bounds, the recent results of \cite{51, 52, 53, 55}, showing lower bounds for multilinear circuits and formulas, suggest that efficient identity testers for multilinear formulas may be at reach. Indeed, a major
progress in this question has been made in the recent of work of Anderson et al. [5]. In this work, multilinear read-
$k$ formulas were studied, resulting in polynomial and quasi-polynomial time PIT algorithms in the black-box and
the non black-box settings, respectively, for constant values of $k$. In fact, they have studied a broader model - multilinear read-$k$ formulas in which each leaf (variable) can be replaced by a sparse polynomial. The model is
referred to as “sparse-substituted” and it extends the model considered here. However, the resulting black-box PIT algorithm for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits admits a quasi-polynomial running time.

4.1.4 Organization

We start by some basic definitions and notation in Section 4.2. In Section 4.3 we formally introduce our model and give some related definition. In Section 4.4, we prove our structural theorem (Theorem 8) and exhibit a
lower bound. We give our main result - a black-box PIT algorithm for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits, in Section 4.5, thus proving Theorem 9. Finally, in Section 4.6 we consider a non black-box PIT algorithm for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits, proving Theorem 10.

4.2 Preliminaries

4.2.1 Sparsity of a Polynomial

In this section we formally define the notion “sparsity of a polynomial”. We also show how to upper bound the sparsity of a given polynomial via the sparsities of its different partial substitutions. For simplicity we concentrate on multilinear polynomials, however the definitions can be generalized to all
polynomials. A multilinear polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ can be (uniquely) written as

$$P = \sum_{A \subseteq [n]} \alpha_A \cdot X^A$$  \hspace{1cm} (4.1)

where $\alpha_A \in \mathbb{F}$ (the coefficients) and $X^A$ denotes $\prod_{i \in A} x_i$.

\footnote{read-$k$ formulas are arithmetic formulas in which each variable can appear at most $k$
times.}
Definition 4.2.1. We define the characteristic set of a multilinear polynomial $P$ as $\chi_P \triangleq \{ A \mid A \subseteq [n], \alpha_A \neq 0 \}$. The sparsity of $P$ is defined as the number of (the non-zero) monomials of $P$ and denoted by $\|P\|$. Clearly, $\|P\| = |\chi_P|$. For the purposes of connecting the sparsity of a given polynomial with the sparsities of its partial substitutions we extend these notions. For every $I \subseteq [n]$ we define $\chi_{P|I} \triangleq \{ A \setminus I \mid A \subseteq [n], \alpha_A \neq 0 \}$ and $\|P||I\triangleq |\chi_{P|I}|$.

The following is immediate from the definitions and will be used implicitly.

Corollary 4.2.2. Let $P,Q \in \mathbb{F}[x_1,\ldots,x_n]$ be variable-disjoint polynomials and let $I \subseteq [n]$. Then $\|P||I\leq\|P\|$ and $\|P \cdot Q||I = \|P||I \cdot \|Q||I$.

Intuitively, $\chi_{P|I}$ captures the distinct monomials left after we “erase” the variables of $\bar{x}_I$. In fact, the same effect is achieved by setting these variables to field elements. However, different substitutions may lead to different results, as monomials may cancel out. For example: $\|(x_1x_3 + x_1x_2x_3)|_{x_3=1}\| = 2 = \|x_1x_3 + x_1x_2x_3\|_{[3]}$, however $\|(x_1x_3 + x_1x_2x_3)|_{x_3=0}\| = 0$. It is not hard too see that a random assignment to the variables of $\bar{x}_I$ should not lead to any unwanted cancellations, and thus achieve the same effect in sparsity as “erasing” the variables of $I$.

We show that $\|P||I\$ is, in fact, attained by the substitution that maximizes the sparsity of the resulting polynomial. We do so by formulating this condition (of not creating any unwanted cancellations and hence maximizing the sparsity) as that of being a non-zero assignment to a certain polynomial $\Psi_P$. Observe that a random assignment would indeed satisfy this condition. Before giving the proof, we need some more definitions.

Definition 4.2.3. Let $P \in \mathbb{F}[x_1,\ldots,x_n]$. For $B,I \subseteq [n]$ such that $B \cap I = \emptyset$ we define:

$$P_{B,I} \triangleq \sum_{J \subseteq I} \alpha_{B \cup J} \cdot X^J \text{ and } \Psi_P \triangleq \prod_{P_{B,I} \neq 0} P_{B,I}.$$  

Intuitively, $P_{B,I}$ represents the monomials of $P$ that could be affected by setting the variables in $\bar{x}_I$ to field elements. $\Psi_P$ captures all the non-zero $P_{B,I}$-s.

Lemma 4.2.4 (Max sparsity Condition). Let $I \subseteq [n]$ and let $\bar{a} \in \mathbb{F}^n$ be such that $\Psi_P|_{\bar{x}_I=\bar{a}} \neq 0$. Then $\|P|_{\bar{x}_I=\bar{a}}\| = \max_{\bar{b} \in \mathbb{F}^n} \|P|_{\bar{x}_I=\bar{b}}\| = \|P||I\$.
Proof. Clearly, \(|P|_{x_I=\bar{a}_I}|| \leq \max_{b \in \mathbb{F}^k} |P|_{x_I=b_I}|| \leq |P||_I. We now show that
|P||_I \leq |P|_{x_I=\bar{a}_I}||. In fact, we show that \(\chi_{P|_I} \subseteq \chi(P|_{x_I=\bar{a}_I})\). Let \(P|_{x_I=\bar{a}_I} = \sum_{B \subseteq [n]\backslash I} \beta_B \cdot X^B\) be the representation of \(P|_{x_I=\bar{a}_I}\) as in From 4.1. Observe that \(\beta_B = P_{B,I}|_{x_I=\bar{a}_I}\). Now, let \(B \in \chi_{P|_I}\). By the definition of \(\chi_{P|_I}\) there exists \(A \subseteq [n]\) such that \(B = A \backslash I\) and \(\alpha_A \neq 0\). Set \(J = A \backslash B\). Note that \(J \subseteq I\). This implies that \(P_{B,I} \neq 0\) as the coefficient of \(X^J\) in \(P_{B,I}\) is \(\alpha_{B,J} = \alpha_A \neq 0\). From the choice of \(\bar{a}\), it holds that \(\beta_B = P_{B,I}|_{x_I=\bar{a}_I} \neq 0\) which implies that \(B \in \chi(P|_{x_I=\bar{a}_I})\).

As mentioned earlier, the main technical task will be to bound \(|F||_I\) in terms of the different \(|F||_I\)-s. For this purpose we use Shearer’s Lemma (see e.g. [16]).

Lemma 4.2.5 (Shearer). Let \(n \in \mathbb{N}\) and let \(I_1, \ldots, I_m \subseteq [n]\) be subsets of \([n]\) such that every element of \(i \in [n]\) is contained in at least \(k\) of \(I_1, \ldots, I_m\). Let \(\mathcal{F}\) be a collection of subsets of \([n]\) and let \(\mathcal{F}_j = \{A \cap I_j \mid A \in \mathcal{F}\}\) for \(j \in [m]\). Then we have \(|\mathcal{F}|^k \leq \prod_{j=1}^m |\mathcal{F}_j|\).

The following corollary of Shearer’s Lemma connects the sparsity of a polynomial with the sparsities of its different partial substitutions.

Corollary 4.2.6. Let \(F \in \mathbb{F}[x_1, \ldots, x_n]\) be a polynomial, \(d \geq 2\) and \(I_1, I_2, \ldots, I_d \subseteq [n]\) disjoint sets. Then \(|F|^{d-1} \leq \prod_{j=1}^d |F||_{I_j}\).

Proof. Set \(F = \chi_F\) and \(\mathcal{F}_j = \{A \cap I_j \mid A \in \mathcal{F}\}\), and apply Shearer’s lemma. Note that since the \(I_j\)-s are disjoint, every \(i \in [n]\) appears in every complement set \(\bar{I}_j\), except at most one.

We will use the results from Section 5.3.1 that gives an efficient factorization algorithm for sparse multilinear polynomials. In particular we can see that a factor of a multilinear \(s\)-sparse polynomial is also a multilinear \(s\)-sparse polynomial.

Lemma 4.2.7 (Corollary from Corollary 5.3.3). Given a multilinear polynomial \(P \in \mathbb{F}[x_1, \ldots, x_n]\) there is a poly\((n, |P|)\) time deterministic algorithm that outputs the irreducible factors, \(h_1, \ldots, h_k\) of \(P\). Furthermore, \(|h_1|| \cdot |h_2|| \cdot \ldots \cdot |h_k|| = |P|\).
The following is a simple property of the sparsity of the gcd of several polynomials. For a proof see Appendix 4.7.

**Observation 4.2.8.** Let \( \{F_i\}, \{G_i\} \subseteq \mathbb{F}[x_1,\ldots,x_n] \) be such that \( F_i, G_i \not\equiv 0 \). Then

\[
\|\gcd(F_1 \cdot G_1, F_2 \cdot G_2,\ldots,F_k \cdot G_k)\| \leq \|\gcd(F_1,F_2,\ldots,F_k)\| \cdot \|G_1\| \cdot \|G_2\| \cdots \|G_k\|.
\]

**4.2.2 The Operator \( D_\ell \)**

This operator was defined and used in [36] for the purpose of finding a non-zero assignment of a polynomial, that preserves certain properties. In this paper we extend the usage of this operator to finding zero assignments of polynomials. This task of finding zero assignments turns out to be much trickier. In this section we formally define the operator and list some properties that immediately follow (and will be used later).

**Definition 4.2.9.** For \( \ell \in [n] \) let \( D_\ell(P,Q) \) be the polynomial defined as follows:

\[
D_\ell(P,Q)(\bar{x}) \triangleq \left| \begin{array}{c} P \parallel P|_{x_\ell=0} \\ Q \parallel Q|_{x_\ell=0} \end{array} \right| (\bar{x}) = (P \cdot Q|_{x_\ell=0} - P|_{x_\ell=0} \cdot Q)(\bar{x}).
\]

Note that \( D_\ell \) is a bilinear transformation. The following lemma lists several useful properties of \( D_\ell \) that are easy to verify.

**Lemma 4.2.10.** Let \( P,Q,R \in \mathbb{F}[x_1,\ldots,x_n] \) be multilinear polynomials and let \( \ell \in [n] \). Then the following properties hold:

1. \( D_\ell(R + Q,P) = D_\ell(R,P) + D_\ell(Q,P) \).
2. Let \( R \) be such that \( \ell \not\in \text{var}(R) \) then \( D_\ell(R \cdot Q,P) = R \cdot D_\ell(Q,P) \).
3. \( D_\ell(Q,P) = -D_\ell(P,Q) \).
4. Let \( i \neq \ell \) then \( D_\ell(P|_{x_i=\alpha},Q|_{x_i=\alpha}) = (D_\ell(P,Q))|_{x_i=\alpha} \).
5. Let \( \alpha,\beta \in \mathbb{F} \) then \( \|D_\ell(Q,\alpha \cdot x_\ell + \beta)\| \leq \|Q\| \).
6. Let \( P \) be irreducible and let \( \ell \in \text{var}(P) \) then \( D_\ell(Q,P) \equiv 0 \) iff \( P \mid Q \).
The last property can be easily generalized to yield a condition for a set of polynomials to have a (non-trivial) gcd. This condition was implicitly used in [36]. For the sake of completeness we give a proof in Appendix 4.7.

**Lemma 4.2.11.** Let $F_1, F_2, \ldots, F_k \in \mathbb{F}[x_1, \ldots, x_n]$ be multilinear polynomials. Then $\text{gcd}(F_1, F_2, \ldots, F_k) \neq 1$ iff there exists $\ell \in \text{var}(F_1)$ such that $D_\ell(F_i, F_1) \equiv 0$ for every $i \in [k]$.

### 4.3 Depth-4 Multilinear Circuits

In this section, we formally present the model of depth-4 multilinear circuits and some related definitions. Similar definitions were given in [36].

**Definition 4.3.1.** A multilinear depth-4 $\Sigma\Pi\Sigma\Pi(k)$ circuit $C$ has four layers of alternating $\Sigma$ and $\Pi$ gates (the top $\Sigma$ gate is at level one) and it computes a polynomial of the form

$$C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}(\bar{x})$$

where the $P_{ij}(\bar{x})$-s are multilinear polynomials computed by the last two layers of $\Sigma\Pi$ gates of the circuit and are the inputs to the $\Pi$ gates at the second level. In addition, each multiplication gate $F_i$ computes a multilinear polynomial.

The requirement that the $F_i$-s compute multilinear polynomials implies that for each $i \in [n]$ the polynomials $\{P_{ij}\}_{j \in [d_i]}$ are variable-disjoint. Note that if the circuit is of size $s$ then each $P_{ij}$ is $s$-sparse. For every $A \subseteq [k]$ we define the subcircuit $C_A$ of $C$ as $C_A \triangleq \sum_{i \in A} F_i$. We define $\text{gcd}(C) \triangleq \text{gcd}(F_1, \ldots, F_k)$. We say that the circuit $C$ is **simple** if $\text{gcd}(C) = 1$. We define the **simplification** of $C$ to be $\text{sim}(C) \triangleq C / \text{gcd}(C)$. Note that $\text{sim}(C)$ is a simple $\Sigma\Pi\Sigma\Pi(k)$ circuit. We say that the circuit $C$ is **minimal** if no proper subcircuit of $C$ computes the zero polynomial. That is, for every $\emptyset \subsetneq A \subsetneq [k]$ it holds that $C_A \neq 0$. For $P \in \mathbb{F}[x_1, \ldots, x_n]$ we say that the circuit $C$ is $P$-**minimal** if no proper subcircuit of $C$ is divisible by $P$.

Lemma 4.2.7 implies that all the irreducible factors of the $P_{ij}$-s are $s$-sparse. Moreover, if $C$ is given to us explicitly those factors can be computed
efficiently. Consequently, we can assume w.l.o.g that all the $P_{ij}$-s are irreducible.

### 4.4 The Sparsity Bound

In this section we prove the main technical result (Theorem 8) an upper bound on the sparsity of the multiplication gates in a simple and minimal, multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuit $C$ computing the zero polynomial. To complete the picture we also give a lower bound on the gate’s sparsity.

We first present an outline of the proof. As mentioned earlier, we assume w.l.o.g that all $P_{ij}$-s are irreducible and use the circuit size $s$ to bound their sparsities (i.e. $\|P_{ij}\| \leq s$). The proof is by induction on $k$ (the fan-in).

**Step 1:** We show that for every $\emptyset \subsetneq A \subsetneq [k]$, $\gcd(C_A)$ is $s^{5(k-|A|+1)^2}$-sparse. We do so by “embedding” $\gcd(C_A)$ into a circuit with a smaller $k$ and applying the inductive argument. In particular, we conclude that for every $i \neq j$ it holds that $\gcd(F_i, F_j)$ is $s^{5(k-1)^2}$-sparse.

**Step 2:** Applying the previous argument, we conclude for each $1 \leq i \leq k-1$ there are “many” $P_{kj}$-s that do not divide $F_i$. More specifically, if all the gates of $C$ are $s^{5k^2}$-sparse, then we are done. Otherwise, w.l.o.g $F_k$ is $s^{5k^2}$-dense. Let $1 \leq i \leq k-1$. Write $F_k = F'_k \cdot \gcd(F_i, F_k)$. By definition, $F'_k$ is the product of all the $P_{kj}$-s that do not divide $F_i$. As $F_k$ is $s^{5k^2}$-dense and $\gcd(F_i, F_k)$ is $s^{5(k-1)^2}$-sparse we get that $F'_k$ is $s^{5k^2-5(k-1)^2} = s^{10k-5}$-dense. Finally, we observe that since each $P_{kj}$ is $s$-sparse, $F'_k$ must be a product of at least $10k - 5$ of them.

**Step 3:** For every $1 \leq i \leq k-1$ and $j$ such that $P_{kj}$ does not divide $F_i$ we find a zero assignment $\bar{a}$ of $P_{kj}$ that preserves certain properties of $C$ (see discussion in Section 4.1.1). In particular, $\bar{a}$ maximizes the sparsity of $F'_i$ - the polynomial resulting from a substitution of $\bar{a}$ into $F_i$. Afterwards, using the inductive argument we obtain a “good” estimation for the sparsity of $F'_i$. Formally, we show that $F'_i$ is $s^{5(k-1)^2+k+19}$-sparse. This analysis is the heart of the proof of the sparsity bound.

**Step 4:** Based on the information collected in Step 3 we use Shearer's
Lemma to show that for every $1 \leq i \leq k - 1$, $F_i$ is $s^{5k^2-1}$-sparse. As $-F_k = F_1 + F_2 + \ldots + F_{k-1}$ we can upper bound the sparsity of $F_k$ by the sum of the sparsities of $F_i$'s, for $1 \leq i \leq k - 1$. We conclude that $F_k$ is $s^{5k^2-1} \cdot (k - 1) < s^{5k^2}$-sparse, thus reaching a contradiction to our assumption. The “large” number of distinct partial substitutions used in Shearer’s Lemma makes our bound nearly optimal.

We now give a more formal statement and then prove Theorem 8.

**Theorem 4.4.1 (The Sparsity Bound).** There exists an non-decreasing function $\varphi(k,s)$ such that if $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x})$ is a simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size $s$ computing the zero polynomial, then for each $i \in [k]$ it holds that $\|F_i\| \leq \varphi(k,s)$ and $\varphi(k,s) \leq s^{5k^2}$.

**Proof.** The proof is by induction on $k$. The base case is $k = 2$. Note that in this case $C$ must be of the form $C = \alpha - \alpha$ for some $\alpha \in \mathbb{F}$. Therefore, $\|F_1\| = \|F_2\| = 1$. Assume that $k \geq 3$.

We first state and prove some lemmas that will be useful for the proof. Note that all lemmas are proven as part of the inductive argument we apply in the proof of the theorem. We start by showing an upper bound on the sparsity of the gcd of any subcircuit of a simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit computing the zero polynomial. We do so by ‘embedding’ the gcd as a multiplication gate into a $\Sigma\Pi\Sigma\Pi$ circuit of a smaller fan-in. Informally, we do this be setting all the variables that do not appear in the gcd to field elements in such a way that the simplicity and minimality of the circuit is preserved (and hence induction can be applied). By setting the variables, the subcircuit corresponding to the gcd collapses to a single multiplication gate which is just a scaled version of the gcd term. In fact, setting those variables to *random* field elements would work with high probability if the underlying field is large enough. Below we present a formal argument.

**Lemma 4.4.2.** Let $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x})$ be a simple and minimal, multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit of size $s$ computing the zero polynomial and let $G \overset{\Delta}{=} \gcd(F_1, F_2, \ldots, F_t)$ for some $2 \leq t \leq k - 1$. Then $\|G\| \leq \varphi(k-t+1,s)$. 


Proof. Denote $V = [n] \setminus \text{var}(G)$ and $F_i = f_i \cdot G$ for $i \in [t]$. Observe that \text{var}(f_i) \subseteq V$ for each $f_i$. We will show now that there exists a partial assignment $\bar{a} \in \mathbb{F}^n$ to $\bar{x}_V$ that preserves the properties of minimality and the simplicity of the circuit. For that purpose, recalling Definition 4.2.9, we define the polynomial:

$$\Phi = \prod_{\emptyset \subseteq A \subseteq [k]} C_A \cdot \prod_{\ell,j : D_{\ell}(F_i,F_1) \neq 0} D_{\ell}(F_i,F_1).$$

Let $\bar{a} \in \mathbb{F}^n$ such that $\Phi|_{\bar{x}_V=\bar{a}_V} \neq 0$. Let $F_i' \triangleq F_i|_{\bar{x}_V=\bar{a}_V}$ for $i \in [k]$. Consider $C' = C|_{\bar{x}_V=\bar{a}_V} \triangleq \sum_{i=1}^k F_i'$. From the definition of $\bar{a}$ it follows that $C'$ is minimal since $C'|_A = C|_{\bar{x}_V=\bar{a}_V} \neq 0$ for every $\emptyset \subseteq A \subseteq [k]$. Next, we argue that $C'$ is simple. As $C$ is simple we have that $\gcd(F_1,F_2,\ldots,F_k) = 1$. Therefore, Lemma 4.2.11 implies that for every $\ell \in \text{var}(F_1)$ there exists $i \in [k]$ such that $D_{\ell}(F_i,F_1) \neq 0$. From the choice of $\bar{a}$ we obtain that for every $\ell \in \text{var}(F_1)$ there exists $i \in [k]$ such that $D_{\ell}(F_i',F_1') \neq 0$, and thus, by the second direction of Lemma 4.2.11 $\gcd(F_1',F_2',\ldots,F_k') = 1$. Now, set $H_1 \triangleq \sum_{i=1}^t F_i'$ and $H_i \triangleq F_{t+i-1}'$ for $2 \leq i \leq k-t+1$. Recall that $\text{var}(f_i) \subseteq V$ for each $i \in [t]$, therefore

$$H_1 = \sum_{i=1}^t F_i|_{\bar{x}_V=\bar{a}_V} = \sum_{i=1}^t (f_i \cdot G)|_{\bar{x}_V=\bar{a}_V} = \left( \sum_{i=1}^t f_i|_{\bar{x}_V=\bar{a}_V} \right) \cdot G = \alpha \cdot G$$

for some $\alpha \in \mathbb{F}$. We now can define $\hat{C} \triangleq \sum_{i=1}^{k-t+1} H_i = \alpha \cdot G + \sum_{i=t+1}^k F_i'$ - the circuit obtained from $C'$ by joining together the first $t$ summands. Indeed, we embedded $G$ into a circuit with a smaller fan-in. We argue $\hat{C}$ satisfies the required properties so we can apply the induction hypothesis. First, note that $\hat{C} \equiv 0$. The minimality of $\hat{C}$ follows from the minimality of $C'$. In particular, $\alpha \cdot G = H_1 = C'|_F \neq 0$ and thus $\alpha \neq 0$. Finally, observe that

$$\gcd(\hat{C}) = \gcd(\alpha \cdot G,F_{t+1}',\ldots,F_k') = \gcd(F_1',\ldots,F_t',F_{t+1}',\ldots,F_k') = \gcd(C')$$

and hence $\hat{C}$ is simple. We can conclude that $\hat{C}$ is a simple and minimal, $\Sigma \Pi \Sigma \Pi (k-t+1)$ circuit of size $s$ computing the zero polynomial. In addition,
note that $2 \leq k - t + 1 \leq k - 1$. Consequently, we can apply the induction hypothesis on $\hat{C}$. We obtain that $\|G\| = \|H_1\| \leq \varphi(k - t + 1, s)$. 

Next is a technical lemma that will allow us to decrease, in some sense, the top fan-in of the circuit. This step is required in order to use the inductive argument. Recall that a $P$-minimal circuit is one where no proper subcircuit is divisible by $P$.

**Lemma 4.4.3.** Let $C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x})$ be a multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuit computing the zero polynomial and let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a factor of $F_k$ (i.e. $P | F_k$) such that $P \not| F_1$. Then there exists a set $A \subseteq [k]$ of size $2 \leq |A| \leq k - 1$ such that the following holds: $1 \in A$, the subcircuit $C_A(\bar{x}) = \sum_{i \in A} F_i(\bar{x})$ is $P$-minimal and $P \mid C_A$.

**Proof.** As $C$ computes the zero polynomial we have that $P | C$. Therefore, $C$ can be partitioned into subcircuits that are minimal w.r.t. this property. Formally, there exists a partition $\bigcup_i A_i = [k]$, $A_i \cap A_j = \emptyset$ such that for every $i$ the subcircuit $C_{A_i}$ is $P$-minimal and $P \mid C_{A_i}$. Let w.l.o.g. $A_1$ be such that $1 \in A_1$. It is only left to show that $2 \leq |A_1| \leq k - 1$. First of all, note that since $P | F_k$ there must be $A_j = \{k\}$ for some $j \neq i$ and hence $|A_1| \leq k - 1$. For the second condition, note that since $P \not| F_1$ it must be the case that $A_1 \neq \{1\}$ and hence $|A_1| \geq 2$. 

Finally, we give the heart of our argument. The lemma shows how to transform a given simple and $P$-minimal circuit $C$, such that $P \mid C$, into a simple and minimal circuit of a smaller fan-in, computing the zero polynomial. As before, the transformation is carried out by finding a partial assignment $\bar{a}$ to $\bar{x}_{\text{var}(P)}$ that preserves the minimality and the simplicity of $C$. However, the most important property of $\bar{a}$ is that it maximizes the sparsity of the circuit, resulting upon the partial substitution into $\bar{x}_{\text{var}(P)}$. This fact allows us to apply Shearer’s Lemma. It can be easily seen that $P(\bar{a}) = 0$. To find the required assignment we present a new technique (see Sec 4.2.2).

**Lemma 4.4.4.** Let $2 \leq t \leq k - 1$. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-constant, irreducible, multilinear polynomial and let $C(\bar{x}) = \sum_{i=1}^{t} F_i(\bar{x})$ be a simple and
P-minimal, multilinear $\Sigma\Pi\Sigma\Pi(t)$ circuit of size $s$ such that $P \mid C$. Then 
$$\|F_1\|_{\text{var}(P)} \leq \varphi(t, s) \cdot s^t.$$  

Proof. Our goal is to upperbound $\|F_1\|_{\text{var}(P)}$. As previously, we wish to do so by an appropriate embedding. What we would like to do is to fix the variables in $\text{var}(P)$ in a way that will make $P$ evaluate to zero, and at the same time, will result in a simple and minimal circuit. This way the circuit would have fewer multiplication gates, and we could apply induction. Unfortunately, this scenario might not be possible to implement. We will look to approximate it paying a ‘small price’.

Pick $\ell \in \text{var}(P)$. For each $i \in [t]$ we can write $F_i = H_i \cdot Q_i$ such that $\|Q_i\| \leq s$ and $\ell \not\in \text{var}(H_i)$: If $\ell \in \text{var}(F_i)$ set $Q_i$ to be the irreducible factor of $F_i$ that depends on $\ell$, otherwise set $Q_i = 1$. Now, recalling Definition 4.2.9, consider $C' = D_\ell(C, P) \triangleq \sum_{i=1}^{t} D_\ell(F_i, P)$. By Lemma 4.2.10

$$C' = \sum_{i=1}^{t} H_i \cdot D_\ell(Q_i, P) \equiv 0.$$  

In addition, note that $C'$ is minimal since $C'_A = D_\ell(C_A, P) \neq 0$ for every $\emptyset \subsetneq A \subsetneq [t]$ from the $P$-minimality of $C$ and Lemma 4.2.10. By definition, the polynomials $H_i$ and $Q_i$ are variable-disjoint. However, this might not be the case for $P$ and $H_i$. Consequently, $C'$ might be non-multilinear. Furthermore, $\|D_\ell(Q_i, P)\|$ might be large. In order to resolve the aforementioned problems we will use a partial assignment $\bar{a}$ with properties similar to the ones in Lemma 4.4.2. Let $V = \text{var}(P) \setminus \{\ell\}$. We recall Definition 4.2.3 and define:

$$\Phi = \prod_{\emptyset \subseteq A \subseteq [t]} C'_A \cdot \prod_{j,i : D_j(F_i, F_1) \neq 0} D_j(F_i, F_1) \cdot \Psi_{H_1}.$$  

Let $\bar{a} \in \mathbb{F}^n$ such that $\Phi|_{\bar{x}_V = \bar{a}_V} \neq 0$. Set $H'_i \triangleq H_i|_{\bar{x}_V = \bar{a}_V}$ and $Q'_i \triangleq D_\ell(Q_i, P)|_{\bar{x}_V = \bar{a}_V}$ for $i \in [t]$. Consider $C'' = C'|_{\bar{x}_V = \bar{a}_V} \triangleq \sum_{i=1}^{k} H'_i \cdot Q'_i.$ By a reasoning similar to Lemma 4.4.2 we obtain that $C''$ is minimal and that $\gcd(H'_1, H'_2, \ldots, H'_t) = 1$. By Lemma 4.2.10 for each $i \in [t]$ we have:

$$Q'_i = D_\ell(Q_i|_{\bar{x}_V = \bar{a}_V}, P|_{\bar{x}_V = \bar{a}_V}) = D_\ell(Q_i|_{\bar{x}_V = \bar{a}_V}, \alpha P \cdot x_i + \beta P)$$  

80
for some $\alpha_P, \beta_P \in \mathbb{F}$ and hence $\|Q_i\| \leq \|Q_i\| \leq s$. Consequently, we get that $C''$ is a minimal, multilinear $\Sigma\Pi\Sigma\Pi(t)$ circuit of size $s$ computing the zero polynomial. In addition, from the choice of $\bar{a}$ and Lemma 4.2.4 we obtain (recall that $\ell \notin \text{var}(H_1)$):

$$\|H_1\| = \|H_1\|_{\text{var}(P)} = \|H_1\|_{\text{var}(P) \setminus \{\ell\}} = \|H_1\|_{\text{var}(P)}.$$ 

By the induction hypothesis applied on the simplification $\text{sim}(C'')$ of $C''$:

$$\|H'_i\| \cdot \|Q'_1\| / \gcd(C'') \leq \varphi(t, s).$$

While by Observation 4.2.8

$$\|\gcd(C'')\| \leq \|\gcd(H'_1, \ldots, H'_t)\| \cdot \|Q'_1\| \cdot \ldots \cdot \|Q'_t\| \leq s^{t-1} \cdot \|Q'_1\|.$$ 

Putting the above together we obtain:

$$\|F_1\|_{\text{var}(P)} = \|H_1\|_{\text{var}(P)} \cdot \|Q_1\|_{\text{var}(P)} \leq \|H'_1\| \cdot s \leq \varphi(t, s) \cdot \frac{\|\gcd(C'')\|}{\|Q'_1\|} \cdot s \leq \varphi(t, s) \cdot s^t.$$ 

We now return to the proof Theorem 4.4.1.
Assume for a contradiction (and w.l.o.g) that $\|F_k\| > s^{5k^2}$. We will show that this implies $\|F_i\| \leq s^{5k^2-1}$ for $1 \leq i \leq k-1$. As $\sum_{i=1}^{k} F_i = 0$ we would obtain that

$$\|F_k\| = \left\| \sum_{i=1}^{k-1} F_i \right\| \leq \sum_{i=1}^{k-1} \|F_i\| < (k-1) \cdot s^{5k^2-1} < s^{5k^2}$$

thus leading us to a contradiction.

For the sake of simplicity, we show that the claim holds for $i = 1$ (i.e. $\|F_1\| \leq s^{5k^2-1}$). Note however, the same proof can be repeated for every $1 \leq i \leq k-1$ due to the symmetry of the circuit. We first show that there are “many” (irreducible) $P_{kj}$-s such that $P_{kj} \parallel F_1$. Let w.l.o.g. $P_{kj}$-s are irreducible. For that purpose we define $F'_k \overset{\Delta}{=} F_k / \gcd(F_1, F_k)$. Let w.l.o.g. $P_{k1}, P_{k2}, \ldots, P_{kd}$ be its irreducible factors, that
is, \( F'_k = P_{k1} \cdot P_{k2} \cdots P_{kd} \). By definition, each such \( P_{kj} \) divides \( F_k \) and does not divide \( F_1 \). By Lemma 4.4.2 \( \|\gcd(F_1, F_k)\| \leq s^{5(k-1)^2} \) and hence:

\[
s^d \geq \|P_{k1} \cdot P_{k2} \cdots P_{kd}\| = \|F'_k\| = \frac{\|F_k\|}{\|\gcd(F_1, F_k)\|} \geq s^{5k^2-5(k-1)^2}
\]

implying that \( d \geq 5k^2 - 5(k - 1)^2 = 10k - 5 \). Fix some some \( j \in [d] \) and consider \( P_{kj} \). By Lemma 4.4.3 there exists a set \( A \subseteq [k] \) of size \( 2 \leq |A| \leq k - 1 \) such that \( 1 \in A \), the sub-circuit \( C_A(\bar{x}) \) is \( P_{kj} \)-minimal, and \( P_{kj} \mid C_A \). Assume w.l.o.g that \( A = \{1,2,\ldots,t\} \) when \( t = |A| \). Let \( G \triangleq \gcd(F_1, F_2, \ldots, F_i) \). By Lemma 4.4.2.

\[
\|G\| \leq \varphi(k-t+1, s) \leq s^{5(k-t+1)^2}.
\]

From the \( P_{kj} \)-minimality of \( C_A \) we have that for each \( i \in [t] \) \( P_{kj} \not\parallel F_i \) and hence \( P_{kj} \not\parallel G \). Consequently, we obtain that \( \sim(C_A) \triangleq \sum_{i=1}^{t} F_i/G \) is a simple and \( P_{kj} \)-minimal, multilinear \( \Sigma\Pi\Sigma\Pi(t) \) circuit such that \( P_{kj} \mid \sim(C_A) \) (as the simplification does not affect the \( P_{kj} \)-minimality). Thus, by Lemma 4.4.4:

\[
\|F_1/G\|_{\text{var}(P_{kj})} \leq \varphi(t, s) \cdot s^t \leq s^{5t^2+t}.
\]

Putting together:

\[
\|F_1\|_{\text{var}(P_{kj})} \leq \|F_1/G\|_{\text{var}(P_{kj})} \cdot \|G\| \leq s^{5(k-t+1)^2+t+5k^2} \leq s^{5(k-1)^2+k+19}.
\]

Recall that this inequality holds for every \( P_{kj} \) when \( 1 \leq j \leq 10k - 5 \leq d \). In addition, recall that \( P_{kj} \)-s are variable-disjoint polynomials (as factors of a multilinear polynomial). Hence, we can upper bound \( \|F_1\| \) using Corollary 4.2.6.

\[
\|F_1\| \leq \left( \prod_{j=1}^{10k-5} \|F_1\|_{\text{var}(P_{kj})} \right)^{\frac{1}{10k-6}} \leq s^{\left(5(k-1)^2+k+19\right) \cdot \frac{10k-5}{10k-6}} < s^{5k^2-1}.
\]

As it was previously stated, the above inequality holds, in fact, for every \( F_i \)

\footnote{For \( k \geq 3 \) and \( 2 \leq t \leq k - 1 \) it holds that: \( 5(k-t+1)^2+t+5k^2 \leq 5(k-1)^2+k+19 \).
\footnote{For \( k \geq 3 \) it holds that: \( (5(k-1)^2+k+19) \cdot \frac{10k-5}{10k-6} < 5k^2-1 \).}
when $1 \leq i \leq k - 1$. Therefore, since $F_k = \sum_{i=1}^{k-1} F_i$ it holds that $\|F_k\| < s^{5k^2}$.

In conclusion we obtain that $\|F_i\| \leq s^{5k^2}$ for each $i \in [k]$. \hfill \Box

### 4.4.1 Lower Bound

To give a complete picture, we show that lower bound of $\varphi(k, s) = s^{\Omega(k)}$ on the sparsity of the multiplication gates in a simple and minimal, multilinear $\Sigma^\ell \Pi^\ell \Sigma^\ell \Pi^\ell (k)$ circuit, in terms of the size of the circuit. Our lower bound is over sufficiently large fields and it suggests that our result is near optimal. More specifically, for every $\ell \geq 2$ we construct a simple and minimal, multilinear $\Sigma^\ell \Pi^\ell \Sigma^\ell \Pi^\ell (k)$ circuit of size $s = \text{poly}(k)$, computing the zero polynomial with $s^{\Omega(k)}$-dense multiplication gates, when $k = \Omega(\sqrt{\ell})$. In fact, our $\Sigma^\ell \Pi^\ell \Sigma^\ell \Pi^\ell (k)$ circuit is a $\Sigma^\ell \Pi^\ell \Sigma^\ell (k)$. Our construction is carried out in two steps. At the first step, we construct a multilinear $\Sigma^\ell \Pi^\ell \Sigma^\ell (\ell)$ circuit $C$ with distinct linear functions and show that it computes the zero polynomial. The distinct linear functions imply that every subcircuit of $C$ is simple. At the second step, we consider one of $C'$s minimal subcircuits - $C'$ and use our upper bound to show that $C'$ has a “large” fan-in. This example is similar to the one in Lemma 3.4.3.

**Lemma 4.4.5.** For every $\ell \geq 2$ there exist $k = \Omega(\sqrt{\ell})$ and a simple and minimal, multilinear $\Sigma^\ell \Pi^\ell \Sigma^\ell \Pi^\ell (k)$ circuit $C = \sum_{i=1}^{k} F_i$ of size $s = \text{poly}(k)$, computing the zero polynomial such that $\|F_i\| = s^{\Omega(k)}$ for every $i \in [k]$. (Assuming that $|F| \geq \ell + 2$).

**Proof.** Let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_{\ell+1}\} \subseteq F \setminus \{0\}$ be a subset of $\ell + 1$ distinct non-zero elements. For every $i \in [\ell + 1]$ let $u_i(w)$ be the $i$-th Lagrange Interpolation Polynomial over $A$. Let $R : F^{\ell + 1} \to F$ be defined as: $R(\bar{x}, y) = \prod_{i=1}^{\ell} (x_{i,1} + x_{i,2} + \ldots + x_{i,\ell} + y)$. Since the degree of $y$ in $R(\bar{x}, y)$ is $\ell$ we get that $R(\bar{x}, y) = \sum_{i=1}^{\ell+1} u_i(y) \cdot R(\bar{x}, \alpha_i)$, by interpolating $R(\bar{x}, y)$ as a degree $\ell$ polynomial in $y$. Now consider:

$$C \triangleq \sum_{i=1}^{\ell+1} u_i(0) \cdot R(\bar{x}, \alpha_i) - R(\bar{x}, 0)$$

83
By definition, \( C \) is a multilinear \( \Sigma \Pi \Sigma \Pi(\ell + 2) \) circuit computing the zero polynomial. Let \( C' = \sum_{i=1}^{k} F_i \) be a minimal subcircuit of \( C \) computing the zero polynomial. Note that for distinct \( \alpha_i \)-s the multiplication gates \( (F_i \text{-s}) \) contain distinct linear functions and that \( u_i(0) \neq 0 \) for every \( i \). Consequently, \( C' \) is a simple and minimal, multilinear \( \Sigma \Pi \Sigma \Pi(k) \) circuit computing the zero polynomial, with \( 2 \leq k \leq \ell + 2 \). Let \( s \) denote the size of \( C \). Clearly, \( s = \text{poly}(k, \ell) = \ell \Theta(1) \). Now, let \( i \in [k] \). On one hand, we have that \( \|F_i\| \geq \ell = s^{\Omega(\ell)} \). On the other hand, by Theorem 8 \( \|F_i\| = s^{O(k^2)} \). This implies that \( \|F_i\| = s^{\Omega(k)} \) and that \( k = \Omega(\sqrt{\ell}) \), which in turn implies that \( s = \text{poly}(k) \), as required.

\[ \square \]

### 4.5 Black-Box PIT for Multilinear \( \Sigma \Pi \Sigma \Pi(k) \) Circuits

In this section we give an efficient black-box PIT algorithm for multilinear \( \Sigma \Pi \Sigma \Pi(k) \) circuits. We do so by constructing a generator for such circuits, which gives us a small hitting set. We start by describing the construction. Intuitively, we construct a family of mappings \( H_{\ell,m} \) such that the image of \( H_{\ell,m} \) contains all vectors which are obtained as a concatenation of a prefix of a vector from \( \text{Im}(H_{\ell-1,m}) \) and a suffix of a vector from \( S_m \) (a generator for \( m \)-sparse multilinear polynomials. See Lemma 2.2.4). The correctness of our construction relies on the Sparsity Bound \( \varphi(k,s) \), as defined in Theorem 4.4.1.

We assume that \( |F| > n \) as we are allowed to use elements from an appropriate extension field. Throughout the entire section we fix a set \( A = \{\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq F \) of \( n + 1 \) distinct elements.

**Definition 4.5.1.** For every \( i \in [n] \) let \( U_i(y) : F \to F \) be defined as the degree \( n \) polynomial satisfying: \( U_i(\alpha_j) = 1 \) if \( j \geq i \) and 0 otherwise. For every \( \ell \geq 1 \) and \( m \geq 1 \) we define: for \( i \in [n] \) \( H_{\ell,m}^i(\bar{w}_1, \ldots, \bar{w}_\ell, y_1, \ldots, y_\ell) : F^{q^{\ell+\ell}} \to F \) as

\[
H_{\ell,m}^i(\bar{w}_1, \ldots, \bar{w}_\ell, y_1, \ldots, y_\ell) \triangleq H_{\ell-1,m}^i(\bar{w}_1, \ldots, \bar{w}_{\ell-1}, y_1, \ldots, y_{\ell-1}) \cdot U_i(y_\ell) + S_{m}^i(\bar{w}_\ell) \cdot (1 - U_i(y_\ell)).
\]

and
\[ H_{\ell,m}(\bar{w}_1, \ldots, \bar{w}_\ell, y_1, \ldots, y_\ell) : \mathbb{R}^{q\ell+\ell} \to \mathbb{R}^n \text{ as } H_{\ell,m} \triangleq \left( H_{\ell,m}^1, H_{\ell,m}^2, \ldots, H_{\ell,m}^n \right). \]

For the sake of completeness we set \( H_{0,m}^i \equiv 0 \). We will use the following immediate but crucial observation:

**Observation 4.5.2.** For every \( 0 \leq t \leq n \), it holds that

\[ H_{\ell,m}|_{y_t} = \alpha_t = \left( H_{\ell-1,m}^1, \ldots, H_{\ell-1,m}^t, S_{m+1}^t, \ldots, S_m^n \right) \]

and hence, for every \( \bar{a} \in \text{Im} \left( H_{\ell-1,m} \right) \) and \( \bar{b} \in \text{Im} \left( S_m \right) \) it holds that

\[ (a_1, \ldots, a_t, b_{t+1}, \ldots, b_n) \in \text{Im} \left( H_{\ell,m} \right). \]

In particular, \( \text{Im} \left( H_{\ell-1,m} \right) \cup \text{Im} \left( S_m \right) \subseteq \text{Im} \left( H_{\ell,m} \right) \).

To use the construction, we first show that every “non sparse” circuit can be “shrunk” into a “somewhat sparse” circuit.

**Lemma 4.5.3.** Let \( M \geq 1 \) and let \( C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}(\bar{x}) \) be a multilinear \( \Sigma \Pi \Sigma \Pi(k) \) circuit of size \( s \) such that \( \max_i \| F_i \| > M \). Let \( \bar{a} \in \mathbb{F}^n \) such that \( F_i(\bar{a}) \neq 0 \) for each \( i \in [k] \). Then there exists \( 0 \leq t \leq n-1 \) such that \( M < \max_i \| F_i|_{\bar{x}[t]=\bar{a}[t]} \| \leq M \cdot s \)

**Proof.** We apply the hybrid argument since \( \max_i \| F_i \| > M \) and \( \max_i \| F_i|_{\bar{x}[n]=\bar{a}[n]} \| \leq 1 \leq M \). Let \( 0 \leq t \leq n-1 \) be the maximal index such that \( \max_i \| F_i|_{\bar{x}[t]=\bar{a}[t]} \| > M \). From the choice of \( t \) we have that \( \max_i \| F_i|_{\bar{a}[t+1]=\bar{a}[t+1]} \| \leq M \). For the remaining condition, note that for each \( i \in [k] \) the polynomial \( F_i|_{\bar{x}[t+1]=\bar{a}[t+1]} \) is obtained from \( F_i|_{\bar{x}[t]=\bar{a}[t]} \) by fixing the value of \( x_{t+1} \) to \( a_{t+1} \). As \( F_i \) is multilinear and \( F_i(\bar{a}) \neq 0 \) this fixation can affect at most one \( P_{ij} \) in it, and hence, reduce the maximal sparsity by a factor of at most \( \| P_{ij} \| \leq s \). \( \square \)

Next, we use our structure theorem to guarantees that in the process of shrinking, we do not inadvertently end up making a non-zero circuit into a zero circuit, thus allowing an inductive step.

**Lemma 4.5.4.** Let \( k \geq 2 \) and \( Q \not\equiv 0 \in \mathbb{F}[x_1, \ldots, x_n] \) be a polynomial computed by a simple and minimal, multilinear \( \Sigma \Pi \Sigma \Pi(k) \) circuit \( C(\bar{x}) = \)
\[
\sum_{i=1}^{k} F_i(\bar{x}) \text{ of size } s. \text{ In addition, let } \mathcal{G}_{k-1} \text{ be a generator for } \Sigma \Pi \Sigma \Pi(k-1) \text{ circuits of size } s \text{ and } (2s^2)\text{-sparse polynomials. Then there exists } \bar{a} \in \text{Im}(\mathcal{G}_{k-1}) \text{ and } 0 \leq t \leq n-1, \text{ such that } Q' \triangleq Q|_{x[t]=\bar{a}[t]} \text{ is a non-zero, } (\varphi(k,s) \cdot s^2)\text{-sparse polynomial.}
\]

**Proof.** If \(\max_i \|F_i\| \leq \varphi(k,s)\), then clearly \(\|Q\| \leq \varphi(k,s) \cdot k \leq \varphi(k,s) \cdot s^2\).

Suppose \(\max_i \|F_i\| > \varphi(k,s)\). We define the following polynomial:

\[
\Phi = \prod_{\emptyset \subset A \subseteq [k]} C_A \cdot \prod_{\ell:i : D_\ell(F_i,F_1) \neq 0} D_\ell(F_i,F_1)
\]

From the properties of \(D_\ell\) (Lemma 4.2.10) we get that all multiplicands of \(\Phi\) are either \((2s^2)\)-sparse polynomials or \(\Sigma \Pi \Sigma \Pi(k-1)\) circuits. Therefore, by Observation 2.2.2 we have that \(\Phi(\mathcal{G}_{k-1}) \neq 0\) and consequently, there exists \(\bar{a} \in \text{Im}(\mathcal{G}_{k-1})\) such that \(\Phi(\bar{a}) \neq 0\). By definition, \(F_i(\bar{a}) \neq 0\) for each \(i \in [k]\).

Thus, by Lemma 4.5.3 there exists \(0 \leq t \leq n-1\) such that

\[
\varphi(k,s) < \max_i \|F_i|_{x[t]=\bar{a}[t]}\| \leq \varphi(k,s) \cdot s
\]

Consider the circuit \(C' \triangleq C|_{x[t]=\bar{a}[t]} = \sum_{i=1}^{k} F_i|_{x[t]=\bar{a}[t]}\). By a reasoning similar to Lemma 4.4.2 we obtain that \(C'\) is simple and minimal. We now argue that \(C' \neq 0\). Assume the contrary. By Theorem 4.4.1 we get that \(\max_i \|F_i|_{x[t]=\bar{a}[t]}\| \leq \varphi(k,s)\), which leads us to a contradiction. Finally, note that \(\|Q'\| \leq \varphi(k,s) \cdot s \cdot k \leq \varphi(k,s) \cdot s^2\).

Having the above, we use a generator to work around the problems raised by lack of simplicity or minimality. This idea has been previously used in [36].

**Theorem 4.5.5** (\(H_{k,m}\) is a Generator). Let \(k \geq 1, s \geq 2\). Let \(Q \neq 0 \in \mathbb{F}[x_1, \ldots, x_n]\) be a polynomial computed by a multilinear \(\Sigma \Pi \Sigma \Pi(k)\) circuit \(C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}(\bar{x})\) of size \(s\). Then for every \(m \geq \varphi(k,s) \cdot s^2\) it holds that \(Q(H_{k,m}) \neq 0\).

**Proof.** We apply induction on \(k\). For \(k = 1\) note that \(Q\) is a product of \(s\)-sparse polynomials. Thus, the claim follows from Observations 4.5.2 and

\[86\]
2.2.2, and properties of \( S_m \) (Lemma 2.2.4). Assume that \( k \geq 2 \). We will argue that we can assume w.l.o.g that \( C \) is simple and minimal. Suppose \( C \) is not minimal. Then there exists a \( \Sigma \Pi \Sigma \Pi(k') \) circuit \( C' \) with \( k' < k \) computing \( Q \). Therefore, by the induction hypothesis it holds that \( Q(H_{k',m}) \neq 0 \); and hence, by Observation 4.5.2 we get that \( Q(H_{k,m}) \neq 0 \). Now, suppose \( C \) is not simple. As previously, we assume w.l.o.g that \( P_{ij} \) are irreducible and write: \( Q \equiv G \cdot C' \), when \( G = \gcd(C) \) and \( C' = \text{sim}(C) \). By definition, \( G \) is a product of \( s \)-sparse polynomials and thus, as previously, \( G(H_{k',m}) \neq 0 \). Therefore, it is sufficient to show that \( C'(H_{k',m}) \equiv 0 \). Recall that \( \text{sim}(C) \) is a simple \( \Sigma \Pi \Sigma \Pi(k) \) circuit. Due to the above, we can assume w.l.o.g that \( C \) is simple and minimal. By the induction hypothesis \( H_{k-1,m} \) is a generator for \( \Sigma \Pi \Sigma \Pi(k - 1) \) circuits of size \( s \). In addition, it is a generator for \( (2s^2) \)-sparse polynomials (Observation 4.5.2). By Lemma 4.5.4 there exists \( \bar{a} \in \text{Im}(H_{k-1,m}) \) and \( 0 \leq t \leq n - 1 \), such that \( Q' \equiv Q|_{x_i=a_i} \) is a non-zero, \( (\varphi(k,s) \cdot s^2) \)-sparse polynomial. Being such, there exists \( \bar{b} \in \text{Im}(S_m) \) for which \( Q'(\bar{b}) \neq 0 \), or equivalently \( Q(a_1, \ldots, a_t, b_{t+1}, \ldots, b_n) = Q'(\bar{b}) \neq 0 \). Observation 4.5.2 completes the proof.

We conclude by giving an explicit construction a hitting set for polynomials \( Q \) computed by \( \Sigma \Pi \Sigma \Pi(k) \) circuits. The idea is that \( Q(H_{k,m}) \) is a non-zero polynomial depending on a small number of variables \( r = q \cdot k + k \), with individual degrees less than \( n^2 + 1 \). Consequently, it is sufficient to evaluate \( Q \) on the set \((V^q)^k \times V^k\).

**Algorithm 3** Construction of a hitting set for \( \Sigma \Pi \Sigma \Pi(k) \) circuits of size \( s \).

**Input:** \( n,k,s \geq 1 \).

**Output:** A hitting set \( \mathcal{H} \)

1. Choose a subset \( V \subseteq \mathbb{F} \) be of size \( |V| = n^2 + 1 \)
2. Set \( \mathcal{H} \equiv H_k, \varphi(k,s), s^2 \) \( (V^q)^k \times V^k \)

**Theorem 4.5.6.** Given \( n, s, k \) as input, Algorithm 3 runs in time \( n^{O(k)} \cdot s^{O(k^3)} \) and outputs \( \mathcal{H} \) of size \( n^{O(k)} \cdot s^{O(k^3)} \), which is a hitting set for \( n \)-variate polynomials that can be computed by multilinear \( \Sigma \Pi \Sigma \Pi(k) \) circuits of size \( s \).

**Proof.** Let \( Q \neq 0 \in \mathbb{F}[x_1, \ldots, x_n] \) be a polynomial computed by a multilinear
ΣΠΣΠ(\(k\)) circuit of size \(s\). Let \(\mathcal{H}\) be the set given by Algorithm 3. By Theorem 4.5.5 we get that \(Q\left(\mathcal{H}_k, \varphi(k, s)\cdot s^2\right)\) is a non-zero polynomial depending on \(r = q \cdot k + k\) variables, with individual degrees less than \(n^2 + 1\) (the degrees of \(y\)'s and \(w\) are at most \(n\) and \(Q\) is multilinear). Consequently, Lemma 2.1.16 implies that \(Q|_{\mathcal{H}} \not\equiv 0\). For the size of \(\mathcal{H}\). Recall Lemma 2.2.4 and that \(\varphi(k, s) \leq s^{5k^2}\). We obtain: \(|\mathcal{H}| \leq |V|^{k+q-k} = n^{O(k)} \cdot \left(n^{O(k^2 \log n} \cdot s\right)^k = n^{O(k)} \cdot s^{O(k^3)}\).

4.6 Non Black-Box PIT for Multilinear ΣΠΣΠ(\(k\)) Circuits

In this section we give a non black-box PIT algorithm for multilinear ΣΠΣΠ(\(k\)) circuits. This running time of the algorithm is slightly better than the running of the black-box one.

4.6.1 Overview

Given Theorem 8 it is fairly easy to check if a simple and minimal, multilinear ΣΠΣΠ(\(k\)) circuit \(C\) computes the zero polynomial. More specifically, let \(C = \sum_{i=1}^{k} F_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}\) and let \(s\) denote the size of \(C\). First, compute the sparsity of each \(F_i\) by noting that \(\|F_i\| = \prod_{j=1}^{d_i} \|P_{ij}\|\). Now, if there exists \(i \in [k]\) for which \(\|F_i\| > s^{5k^2}\) then by Theorem 8 \(C \not\equiv 0\). Otherwise, \(C\) computes a \((k \cdot s^{5k^2})\)-sparse polynomial and hence we can check if \(C \equiv 0\) by computing its monomial expansion. However, an arbitrary multilinear ΣΠΣΠ(\(k\)) circuit \(C\) may be neither minimal nor simple. The idea is first to transform the circuit \(C\) into a simple and minimal circuit \(C'\) such that \(C \equiv 0\) iff \(C' \equiv 0\), and afterwards apply the procedure described above. In fact, we are going to “simplify” and “minimize” \(C\). The “minimization” is carried out by (recursively) checking if all the proper subcircuits of \(C\) compute the zero polynomial. For the “simplification” of \(C\), suppose that all \(P_{ij}\) were irreducible. Under this assumption, each multiplication gate \(F_i\) contains among its \(P_{ij}\)'s the exact same list of irreducible polynomials (up to multiplication by a field element) forming the gcd of \(C\) (if gcd(\(C\)) \(\not\equiv 1\)). Therefore, we can erase those polynomials and obtain a simple circuit. Although the irreducibility of the \(P_{ij}\)'s is a valid assumption for the analysis,
this might not be the case in the given circuit. We handle this scenario by factorizing each $P_{ij}$ to its irreducible factors. The factorization is carried out using Lemma 4.2.7. Note that factorization does not affect the sparsity of $F_i$s and can only decrease the size of the circuit.

### 4.6.2 The Algorithm

We now present the algorithm. First, we start with an algorithm that simplifies a given $\Sigma \Pi \Sigma \Pi(k)$ circuit.

**Lemma 4.6.1.** There is a deterministic algorithm that when given as input a multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuit $C$ of size $s$ on $n$ variables runs in time $\text{poly}(n,s)$ and outputs a simple $\Sigma \Pi \Sigma \Pi(k)$ circuit $C'$ of size $s$ such that $C \equiv 0$ iff $C' \equiv 0$.

**Proof.** Given $C = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P_{ij}$ we factorize each $P_{ij}$ to its irreducible factors in time $\text{poly}(n,s)$ using the algorithm in Lemma 4.2.7 (recall that $\|P_{ij}\| \leq s$). We then replace each $P_{ij}$ by the product of its irreducible factors, obtaining the circuit by $C_{\text{red}} = \sum_{i=1}^{k} \prod_{j=1}^{d_i} P'_{ij}$. Observe that the size of $C_{\text{red}}$ is at most $s$. As there are at most $s$ different $P_{ij}$-s, the total running time of this step is $\text{poly}(n,s)$. By definition, all $P'_{ij}$-s are irreducible. Therefore each multiplication gate of $C_{\text{red}}$ contains among its $P'_{ij}$-s the exact same set of irreducible polynomials (up to multiplication by a field element) forming the gcd of $C_{\text{red}}$. Let $C'$ be the circuit resulting from erasing the mentioned set of polynomials. Clearly, $C'$ is a simple $\Sigma \Pi \Sigma \Pi(k)$ circuit of size $s$. The polynomials computed by $C$ and $C'$ differ by a multiplicative factor of gcd($C_{\text{red}}$). Consequently, $C \equiv 0$ if and only if $C' \equiv 0$. \qed

Finally, we present our PIT algorithm for multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuits, thus proving Theorem 9.

**Lemma 4.6.2.** Given a multilinear $\Sigma \Pi \Sigma \Pi(k)$ circuit $C$ of size $s$ on $n$ variables Algorithm 4 runs in time $\text{poly}(n) \cdot s^{O(k^2)}$ and outputs “true” if and only if $C \equiv 0$.

**Proof.** We begin the correctness analysis by induction on $k$. For $k = 1$ the claim is clear. Let $k \geq 2$. Assume the correctness for smaller values of $k$. By Lemma 4.6.1 $C \equiv 0$ iff $C' \equiv 0$. If there exists $\emptyset \subsetneq A \subsetneq [k]$ such that $C'_A \equiv 0$ then, clearly $C' \equiv 0$ iff $C'_{[k]\setminus A} \equiv 0$. Otherwise, $C'$ is simple and...
Algorithm 4 Non-Black PIT algorithm for multilinear ΣΠΣΠ(\(k\)) circuits

**Input:** ΣΠΣΠ(\(k\)) circuit \(C\) of size \(s\)

**Output:** “true” iff \(C \equiv 0\)

1: if \(k = 1\) then
2: Return “true” iff \(F_1 \equiv 0\)
3: Compute \(C' = \sum_{i=1}^{k} F'_i\) \{using Lemma 4.6.1.\}
4: for all \(\emptyset \subsetneq A \subseteq [k]\) do
5: if \(C'_{A} \equiv 0\) then
6: Return “true” iff \(C'_{[k]\setminus A} \equiv 0\) \{Recursive call.\}
7: for \(i = 1\) to \(k\) do
8: Compute \(\|F'_i\|\)
9: if \(\|F'_i\| > s^{5k^2}\) then
10: Return “false”
11: Return “true” iff \(C' \equiv 0\), by computing the monomial expansion of \(C'\).

minimal, multilinear ΣΠΣΠ(\(k\)) circuit of size \(s\) with \(k \geq 2\). By Theorem 4.4.1 if \(C' \equiv 0\) then \(\|F'_i\| \leq s^{5k^2}\) for every \(i \in [k]\). Therefore, if there exists \(i \in [k]\) such that \(\|F'_i\| > s^{5k^2}\), then \(C \neq 0\). The last step is correct by its definition.

Time complexity: By Lemma 4.6.1 \(C'\) can be computed in time \(\text{poly}(n, s)\). Next, as \(\|F'_i\| \leq 2^n\) for each \(i \in [k]\) the computation of \(\|F'_i\|\) can be done in time \(\text{poly}(n, s)\). Finally, note that the last line is reached only in the case that every \(F'_i\) is \(s^{5k^2}\)-sparse. Therefore, the polynomial computed by \(C'\) in this case has at most \(k \cdot s^{5k^2}\) monomials. Consequently, this step takes \(\text{poly}(n) \cdot s^{O(k^2)}\) time. Putting all together we obtain the following recurrent relation: \(T(k, n, s) \leq 2^{k-1} \cdot T(k-1, n, s) + \text{poly}(n) \cdot s^{O(k^2)}\) and hence \(T(k, n, s) = \text{poly}(n) \cdot s^{O(k^2)}\).

4.7 Missing Proofs

For the sake of completeness we give here some of the missing proofs.

**Proof of Lemma 4.2.11.** For the first direction, assume \(\gcd(F_1, F_2, \ldots, F_k) \neq 1\). Then, by definition, there exists an irreducible polynomial \(P\) such that \(P \mid F_i\) for every \(i \in [k]\). Equivalently, we can write \(F_i = F'_i \cdot P\). Pick
\( \ell \in \text{var}(P) \). Clearly, \( \ell \in \text{var}(F_1) \) and hence \( \ell \notin \text{var}(F'_i) \) since the factors of a multilinear polynomial are variable-disjoint. From Lemma 4.2.10 we get that \( D_\ell(F_1, F_1) = D_\ell(F'_i \cdot P, F'_i \cdot P) = F'_i \cdot F'_i \cdot D_\ell(P, P) \equiv 0 \). For the second direction, let \( \ell \in \text{var}(F_1) \). We can write \( F_1 = F'_i \cdot P \), when \( P \) denotes the irreducible factor of \( F_1 \) that depends on \( \ell \). From the statement \( F'_i \cdot D_\ell(F_i, P) = D_\ell(F_i, F_1) \equiv 0 \). As \( F'_i \neq 0 \), we get that \( D_\ell(F_i, P) \equiv 0 \) for every \( i \in [k] \). It follows from Lemma 4.2.10 that \( P \mid F_i \) for every \( i \in [k] \), or equivalently \( P \mid \gcd(F_1, F_2, \ldots, F_k) \). In particular this implies \( \gcd(F_1, F_2, \ldots, F_k) \neq 1 \) and completes the proof.

**Proof Observation 4.2.8.** Denote \( R = \gcd(F_1 \cdot G_1, F_2 \cdot G_2, \ldots, F_k \cdot G_k) \), \( G_0 = \gcd(F_1, F_2, \ldots, F_k) \) and consider \( P \in \mathbb{F}[x_1, \ldots, x_n] \) an irreducible factor of \( R \). By definition \( P \mid F_i \cdot G_i \) for each \( i \in [k] \). Therefore, for each \( i \in [k] \) it must be the case that either \( P \mid F_i \) or \( P \mid G_i \) holds. Now, if \( P \mid F_i \) for all \( i \)-s, then by definition \( P \mid G_0 \). Otherwise, there must exist \( j \in [k] \) such that \( P \mid G_j \). Consequently, \( P \mid G_0 \cdot G_1 \cdot G_2 \cdot \ldots \cdot G_k \). Since the above holds for every irreducible factor of \( R \) we obtain that \( R \mid G_0 \cdot G_1 \cdot G_2 \cdot \ldots \cdot G_k \). Equivalently, there exists \( Q \) such that \( R \cdot Q = G_0 \cdot G_1 \cdot G_2 \cdot \ldots \cdot G_k \) and hence \( \|R\| \leq \|G_0\| \cdot \|G_1\| \cdot \ldots \cdot \|G_k\| \).

We also show that this inequality is tight. Let \( \Phi \triangleq \prod_{i=1}^k (x_i + 1) \). Take:
\( G_i = (x_i + 1), F_i = \Phi / G_i \). Then \( \gcd(F_1 \cdot G_1, F_2 \cdot G_2, \ldots, F_k \cdot G_k) = \Phi = G_1 \cdot G_2 \cdot \ldots \cdot G_k \cdot \gcd(F_1, F_2, \ldots, F_k) \). Note that \( \gcd(F_1, F_2, \ldots, F_k) = 1 \).
Chapter 5

Variable Disjoint Factors

We say that a polynomial \( f(x_1, \ldots, x_n) \) is *indecomposable* if it cannot be written as a product of two polynomials that are defined over disjoint sets of variables. The polynomial decomposition problem is defined to be the task of finding the indecomposable factors of a given polynomial. Note that for multilinear polynomials, factorization is the same as decomposition, as any two different factors are variable disjoint.

In this chapter we show that the problem of derandomizing polynomial identity testing is essentially equivalent to the problem of derandomizing algorithms for polynomial decomposition. More accurately, we show that for any reasonable circuit class there is a deterministic polynomial time (black-box) algorithm for polynomial identity testing of that class if and only if there is a deterministic polynomial time (black-box) algorithm for factoring a polynomial, computed in the class, to its indecomposable components.

An immediate corollary is that polynomial identity testing and polynomial factorization are equivalent (up to a polynomial overhead) for multilinear polynomials. In addition, we observe that derandomizing the polynomial decomposition problem is equivalent, in the sense of Kabanets and Impagliazzo [31], to proving arithmetic circuit lower bounds to \( \text{NEXP} \).

Our approach uses ideas that showed that the polynomial identity testing problem for a circuit class \( \mathcal{C} \) is essentially equivalent to the problem of deciding whether a circuit from \( \mathcal{C} \) computes a polynomial that has a read-once arithmetic formula.

The results of this chapter are based on the works [66].
5.1 Introduction

In this chapter we study the relation between two fundamental algebraic problems, polynomial identity testing and polynomial factorization. We show that the tasks of giving deterministic algorithms for polynomial identity testing and for a variant of the factorization problem (that we refer to as the polynomial decomposition problem) are essentially equivalent. For background see Sections 1.1 and 1.2.

5.1.1 Polynomial Decomposition

Let $X = (x_1, \ldots, x_n)$ be the set of variables. For a set $I \subseteq [n]$ denote with $X_I$ the set of variables whose indices belong to $I$. A polynomial $f$, depending on $X$, is said to be decomposable if it can be written as $f(X) = g(X_S) \cdot h(X_{[n] \setminus S})$ for some $\emptyset \subseteq S \subseteq [n]$. The indecomposable factors of a polynomial $f(X)$ are polynomials $f_1(X_{I_1}), \ldots , f_k(X_{I_k})$ such that the $I_j$-s are disjoint sets of indices, $f(X) = f_1(X_{I_1}) \cdot f_2(X_{I_2}) \cdots f_k(X_{I_k})$ and the $f_i$’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 an arithmetic circuit class $C$ 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 $f$. Clearly, finding the indecomposable factors of a polynomial $f$ is an easier task than finding all the irreducible factors of $f$. It is not hard to see though, that for the natural class of multilinear polynomials the two problems are the same. We also consider the decision version of the polynomial decomposition problem: Given an arithmetic circuit computing a multivariate polynomial decide whether the polynomial is decomposable or not. Note that 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 non black-box models (see the surveys in [21, 33, 20]). These algorithms also solve the decomposition problem. However, it is a long standing open question whether there is an efficient deterministic algorithm for factoring multivariate polynomials (see [21, 39]).
Moreover, there is no known deterministic algorithm even for the decision version of the problem (that is defined analogously). Furthermore, even for the simpler case of factoring multilinear polynomials (which is a subproblem of polynomial decomposition) no deterministic algorithms are known.

In this work we (essentially) show equivalence between the PIT and polynomial decomposition problems. Namely, we prove that for any (reasonable) circuit class \( C \), it holds that \( C \) has a polynomial time deterministic PIT algorithm if and only if it has a polynomial time deterministic decomposition algorithm. The result holds both in the black-box and the non black-box models. That is, if the PIT for \( C \) is in the black-box model then deterministic black-box decomposition is possible and vice versa, and similarly for the non black-box case.

### 5.1.2 Our Results

We now formally state our results. We give them in a very general form as we later apply them to very restricted classes such as depth-3 circuits, read-once formulas etc. We shall say that \( C \) is an \((n, s, d)\)-circuit if it is an \( n \)-variate arithmetic circuit of size \( s \) with individual degrees bounded by \( d \).

**Theorem 11 (Main).** Let \( C \) be a class of arithmetic circuits, defined over a field \( \mathbb{F} \). Consider circuits of the form \( C = C_1 + C_2 \times C_3 \), where the \( C_i \)-s are \((n, s, d)\) circuits from \( C \) and, \( C_2 \) and \( C_3 \) are defined over disjoint sets of variables.\(^1\) Assume that there is a deterministic algorithm that when given access (explicit or via a black-box) to such a circuit \( C \) runs in time \( T(s, d) \) and decides whether \( C \equiv 0 \). Then, there is a deterministic algorithm that when given access (explicit or via a black-box) to an \((n, s, d)\) circuit \( C' \in C \),\(^2\) runs in time \( O(n^3 \cdot d \cdot T(s, d)) \) and outputs the indecomposable factors, \( H = \{h_1, \ldots, h_k\} \), of the polynomial computed by \( C' \). Moreover, each \( h_i \) is in \( C \) and \( \text{size}(h_i) \leq s \).

The other direction is, in fact, very easy and is given by the following observation.

**Observation 5.1.1.** Let \( C \) be a class of arithmetic circuits. Assume that there is an algorithm that when given access (explicit or via a black-box) to

---

\(^1\)This requirement seems a bit strange but we need it in order to state our results in the most general terms.

\(^2\)\( C' \in C \) denotes that the circuit \( C' \) is from \( C \).
an \((n, s, d)\) circuit \(C \in \mathcal{C}\) runs in time \(T(s, d)\) and outputs “true” iff the polynomial computed by \(C\) is decomposable. Then, there is a deterministic algorithm that runs in time \(O(T(s + 2, d))\) and solves the PIT problem for size \(s\) circuits from \(\mathcal{C}\).

As mentioned above, the irreducible factors of multilinear polynomials are simply their indecomposable factors. Hence we obtain the following corollary. We give here a slightly informal statement. The full statement is given in Section 5.3.1.

**Corollary 5.1.2** (informal). Let \(\mathcal{C}\) be an arithmetic circuit class computing multilinear polynomials. Then, up to a polynomial overhead, the deterministic polynomial identity testing problem and the deterministic factorization problem for circuits from \(\mathcal{C}\) are equivalent, in both the black-box the and non-black-box models.

We also obtain some extensions to the results above. The first result (Theorem 5.4.1) shows how to get a non-adaptive decomposition from a PIT algorithm (Theorem 11 gives an adaptive algorithm). To prove it we need a stronger PIT algorithm than the one used in the proof of Theorem 11. The second extension (Theorem 5.4.3) gives an algorithm deciding whether for a given polynomial \(f\) there are two variables \(x_i, x_j\) such that \(f(X) = f_1(X_{[n]\setminus \{i\}}) \cdot f_2(X_{[n]\setminus \{j\}})\). This can be thought of as a generalization of Theorem 11. Finally, we obtain a connection (Corollary 5.4.7) between the decomposition problem and lower bounds in the sense of Kabanets and Impagliazzo. The statements and proofs of these results are given in Section 5.4.

### 5.1.3 Motivation

The motivation for this work is twofold. First, the most obvious motivation is that we think that the problem of connecting the complexity of PIT and polynomial factorization is very natural. Another motivation is to better understand the PIT problem for multilinear formulas.\(^3\) Although lower bounds are known for multilinear formulas \([49, 50, 53, 54]\), we do not have an efficient PIT algorithm even for depth-3 multilinear formulas. Consider

\(^3\)A multilinear formula is a formula in which every gate computes a multilinear polynomial, see \([49]\).
the following approach towards PIT of multilinear formulas. Start by making the formula a read-once formula. I.e. a formula in which every variable labels at most one leaf. This can be done by replacing, for each \(i\) and \(j\), the \(j\)-th occurrence of \(x_i\) with a new variable \(x_{i,j}\). Now, using PIT algorithm for read-once formulas (Theorems 1, 2), 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 back each \(x_{i,j}\) with \(x_i\). After each replacement we would like to verify that the resulting formula is still not zero. Notice that when replacing \(x_{i,j}\) with \(x_i\) we get zero if and only if the linear function \(x_i - x_{i,j}\) is a factor of the formula at hand. Thus, we somehow have to find a way of verifying whether a linear function is a factor of a multilinear formula. Notice that as we start with a read-once formula for which PIT is known, we can assume that we know many inputs on which the formula does not vanish. 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 and only gives a sketch of an idea, but it shows the importance of understanding how to factor multilinear formulas given a PIT algorithm. As different factors of multilinear formulas are variable disjoint this motivates the study of polynomial decomposition.

5.1.4 Proof Technique

It is not difficult to see that efficient algorithms for polynomial decomposition imply efficient algorithms for PIT. Indeed, notice that \(f(x_1, \ldots, x_n) \equiv 0\) if and only if \(f(x_1, \ldots, x_n) + y \cdot z\), where \(y\) and \(z\) are two new variables, is decomposable (in which case \(y\) and \(z\) are its indecomposable factors). Hence, an algorithm for polynomial decomposition (even for the decision version of the problem) gives rise to a PIT algorithm.

The more interesting direction is obtaining a decomposition algorithm given a PIT algorithm (this is what the idea sketched in Section 5.1.3 requires). Note that if \(f(X) = f_1(X_{I_1}) \cdot \ldots \cdot f_k(X_{I_k})\) is the decomposition of \(f\) and if we know the sets \(I_1, \ldots, I_k\) then using the PIT algorithm we can easily obtain circuits for the different \(f_i\)’s. Indeed, if \(\bar{a} \in \mathbb{F}^n\) is such that \(f(\bar{a}) \neq 0\) then, for some constant \(\alpha_j\), \(f_j(X_{I_j}) = \alpha_j \cdot f|_{\bar{a}[n]\setminus I_j}(X)\), where \(\bar{a}[n]\setminus I_j\) is the assignment that assigns values to all the variables except those whose index
belongs to $I_j$. Now, given a PIT algorithm we can use it to obtain such $ar{a}$ in a manner similar to finding a satisfying assignment to a CNF formula given a SAT oracle. Consequently, finding the partition $I_1, \ldots, I_k$ of $[n]$ is equivalent to finding the indecomposable factors (assuming that we have a PIT algorithm).

We present two approaches for finding the partition. The first is by induction: Using the PIT algorithm we obtain an assignment $\bar{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$ that has the property that for every $j \in [n]$ it holds that $f$ depends on $x_j$ if and only if $f|_{\bar{a}[\{j\}]}$ depends on $x_j$. Following [27, 14, 64, 65] we call $\bar{a}$ a justifying assignment of $f$. Given a justifying assignment $\bar{a}$, we find, by induction, the indecomposable factors of $f|_{x_n=a_n}$. Then, using simple algebra we recover the indecomposable factors of $f$ from those of $f|_{x_n=a_n}$. This is the idea behind the proof of Theorem 11.

The second approach is used in the proofs of Theorems 5.4.1 and 5.4.3 (the extensions to Theorem 11). Note that the variables $x_i$ and $x_j$ belong to the same set $I$ in the partition if and only if $\Delta_{ij} f \triangleq f \cdot f|_{x_i=y, x_j=w} - f|_{x_i=y, x_j=y}$, $f|_{x_j=w} \not\equiv 0$, when $y$ and $w$ are two new variables. Using this observation we obtain the partition by constructing a graph $G$ on the set of vertices $[n]$ in which $i$ and $j$ are neighbors iff $\Delta_{ij} f \not\equiv 0$. The sets of the partition are exactly the connected components of $G$. In fact, $\Delta_{ij} f$ can be used to obtain some additional information on the reducibility of $f$ that we use in order to prove Theorem 5.4.3.

The main difference between the two approaches is the model for which we need the PIT algorithm. For example the second approach does not work for the case of (sums of) read-once formulas.

### 5.1.5 Related Works

A line of works that is related to our results is that of Kabanets and Impagliazzo [31] and of [18]. There it was shown that the question of derandomizing the PIT problem is closely related to the problem of proving lower bounds for arithmetic circuits (Corollary 5.4.7, given in Section 5.4, is an analogous result). These results use the fact that factors of small arithmetic circuits can also be computed by small arithmetic circuits. This gives another con-
nection between PIT and polynomial factorization, although a less direct one.

The results of [31] relate PIT to arithmetic lower bounds for NEXP. However, these lower bounds are not strong enough and do not imply that derandomization of PIT gives derandomization of other algebraic problems. Similarly, the results of [1] show that polynomial time black-box PIT algorithms give rise to exponential lower bounds for arithmetic circuits which in turn, using ideas a-la [31], may give quasi-polynomial time derandomization of polynomial factorization.\(^5\) However, this still does not guarantee polynomial time derandomization as is achieved in this work.

5.1.6 Organization

In Section 5.2 we formally introduce indecomposable polynomials. In Section 5.3 we prove our main result and some corollaries. The extensions to Theorem 11 are given in Section 5.4.

5.2 Indecomposable Polynomials

**Definition 5.2.1.** We say that a polynomial \( f \in \mathbb{F}[x_1,\ldots,x_n] \) is indecomposable if it is non-constant and cannot be represented as the product of two (or more) non-constant variable disjoint polynomials. Otherwise, we say that \( f \) is decomposable.

Clearly decomposability is a relaxation of irreducibility. For example, \((x + y + 1)(x + y - 1)\) is indecomposable but is not irreducible. Also note that any univariate polynomial is indecomposable. The following lemma is easy to prove.

**Lemma 5.2.2** (Unique decomposition). Let \( f \in \mathbb{F}[x_1,\ldots,x_n] \) be a non-constant polynomial. Then \( f \) has a unique (up to multiplication by field elements) factorization to indecomposable factors.

*Proof.* Let \( f = f_1 \cdots f_m \) be the factorization of \( f \) to irreducible polynomials. Define a graph on \( \{1,\ldots,m\} \) as follows: connect \( i \) and \( j \) if there is a variable \( x_\ell \) such that both \( f_i \) and \( f_j \) depend on \( x_\ell \). For every connected component

\(^5\) We use the word ‘may’ as it is not immediate how to derandomize the factorization problem using lower bounds for arithmetic circuits.
I of the graph let \( h_I = \prod_{i \in I} f_i \). It is not difficult to see that in this way we get a factorization of \( f \) to indecomposable factors (the \( h_I \)-s) and that this factorization is unique (up to multiplication by field elements).

**Observation 5.2.3.** Let \( f \) be a multilinear polynomial. Then \( f \) is indecomposable if and only if \( f \) is irreducible. In particular, if \( f(\bar{x}) = f_1(\bar{x}) \cdot f_2(\bar{x}) \cdot \ldots \cdot f_k(\bar{x}) \) is the decomposition of \( f \), then the \( f_i \)-s are \( f \)'s irreducible factors.

### 5.3 Decomposition

In this section we give the proof of Theorem 11. Algorithm 5 shows how to find the indecomposable factors for a polynomial computed by \( C \) using the PIT algorithm. In fact, the algorithm returns a partition \( I = \{I_1, \ldots, I_k\} \) of \([n] \) such that the decomposition of \( f \) is

\[
\begin{align*}
\text{Algorithm 5 Finding variable partition} \\
\text{Input: An (n,s,d)-circuit C from a circuit class C, a justifying assignment } \bar{a} \text{ for C, and access to a PIT algorithm as in the statement of Theorem 11.} \\
\text{Output: A variable-partition } I \\
1: \text{Set } I = \emptyset, \ J = [n] \text{ (I will be the partition that we seek).} \\
2: \text{Set } x_n = a_n \text{ and recursively compute the variable-partition of } C' = C|_{x_n = a_n}. \text{ Let } I' \text{ be the resulting partition (note that when } n = 1 \text{ then we just return } I = \{\{1\}\}). \\
3: \text{For every set } I \in I' \text{ check whether } C(\bar{a}) \cdot C \equiv C|_{x_I = \bar{a}_I} \cdot C|_{x_{[n] \setminus I} = \bar{a}_{[n] \setminus I}}. \text{ If this is the case then add } I \text{ to } I \text{ and set } J \leftarrow J \setminus I. \text{ Otherwise, move to the next } I. \\
4: \text{Finally, add the remaining elements to } I. \text{ Namely, } I \leftarrow I \cup \{J\}. \\
\end{align*}
\]

The following lemma gives the analysis of the algorithm and its correctness.

**Lemma 5.3.1.** Let \( C \) be an (\( n, s, d \))-circuit from \( C \) such that \( \text{var}(C) = [n] \). Assume there exists a PIT algorithm as in the statement of Theorem 11. Let
Let \( f \delta \) be a justifying assignment of \( C \). Then given \( C \) and \( f \delta \) Algorithm 5 outputs a variable-partition \( \mathcal{I} \) for the polynomial computed by \( C \). The running time of the algorithm is \( \mathcal{O}(n^2 \cdot T(s, d)) \), where \( T(s, d) \) is as in the statement of Theorem 11.

Proof. The proof of correctness is by induction on \( n \). For the base case \( (n = 1) \) we recall that a univariate polynomial is an indecomposable polynomial. Now assume that \( n > 1 \) and we recall that a univariate polynomial is an indecomposable polynomial.

Proof. Assume that equality holds. Then, as \( \bar{a} \) is a justifying assignment of \( C \) we obtain that \( \text{var}(C') = [n - 1] \). From the uniqueness of the decomposition (Lemma 5.2.2) and by the induction hypothesis we get that, when running on \( C' \), the algorithm returns \( \mathcal{I}' = \{I_1, \ldots, I_{k-1}, I_{k, \ell}\} \). The next lemma shows that the algorithm indeed returns the variable-partition \( \mathcal{I} \).

Lemma 5.3.2. Let \( f(\bar{x}) \in \mathbb{F}[x_1, \ldots, x_n] \) be a polynomial and let \( \bar{a} \in \mathbb{F}^n \) be a justifying assignment of \( f \). Then \( I \subseteq [n] \) satisfies that \( f(\bar{a}) \cdot f \equiv f|_{x_I=\bar{a}_I} \cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \) if and only if \( I \) is a disjoint union of sets from the variable-partition of \( f \).

Proof. Assume that equality holds. Then, as \( \bar{a} \) is a justifying assignment of \( f \) we have that \( f|_{x_I=\bar{a}_I} \cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \neq 0 \) and hence \( f(\bar{a}) \neq 0 \). Consequently, if we define \( h(X_I) \equiv f|_{x_I=\bar{a}_I} \cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \) and \( g(X_{[n]\setminus I}) \equiv f|_{x_I=\bar{a}_I} \cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \) then we obtain that \( f(\bar{x}) = h(X_I) \cdot g(X_{[n]\setminus I}) \). The result follows by uniqueness of decomposition.

To prove the other direction notice that we can write \( f(\bar{x}) \equiv h(X_I) \cdot g(X_{[n]\setminus I}) \) for two polynomials \( h \) and \( g \). We now have that, \( f|_{x_I=\bar{a}_I} \equiv h(\bar{a}) \cdot g(X_{[n]\setminus I}) \) and similarly \( f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \equiv h(X_I) \cdot g(\bar{a}) \). Hence, \( f(\bar{a}) \cdot f \equiv h(\bar{a}) \cdot g(\bar{a}) \cdot h(X_I) \cdot g(X_{[n]\setminus I}) \equiv f|_{x_I=\bar{a}_I} \cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \). This concludes the proof of Lemma 5.3.2. \( \square \)

By the lemma, each \( I_j \) \((j < k)\) will be added to \( \mathcal{I} \) whereas no \( I_{k,j} \) will be added to it. Eventually we will have that \( J = I_k \) as required. To finish the proof of Lemma 5.3.1 we now analyze the running time of the algorithm. The following recursion is satisfied, where \( t(n, s, d) \) is the running time of the algorithm on input an \( (n, s, d) \)-circuit \( C \in \mathcal{C} \): \( t(n, s, d) = t(n -
The proof of Theorem 11 easily follows.

Proof of Theorem 11. We first note that the assumed PIT algorithm also works for circuits in $\partial C$, when $C$ is an $(n, s, d)$ circuit from $\mathcal{C}$. Therefore, by Lemma 2.3.2 we have an algorithm that finds a justifying assignment $\overline{a}$, as well as computes $\mathrm{var}(C)$. This requires $O(n^3 \cdot d \cdot T(s,d))$ time. Once $\mathrm{var}(C)$ is known we can assume w.l.o.g that $\mathrm{var}(C) = [n]$. Lemma 5.3.1 guarantees that Algorithm 5 returns a variable-partition $\mathcal{I}$ in time $O(n^2 \cdot T(s,d))$. At this point we can define, for every $I \in \mathcal{I}$ the polynomial $h_I \triangleq C|_{x[n]\backslash I=\overline{a}[n]\backslash I}$. It is now clear that for $\alpha = C(\overline{a})^{1-|\mathcal{I}|}$ we have that $C = \alpha \prod_{I \in \mathcal{I}} h_I$ is the decomposition of $C$. Moreover, note that from the definition, each $h_i$ belongs to $\mathcal{C}$ and has size at most $s$. The total running time can be bounded from above by $O(n^3 \cdot d \cdot T(s,d))$. 

To complete the equivalence between polynomial decomposition and PIT we provide a short proof of Observation 5.1.1.

Proof of Observation 5.1.1. Let $C$ be an arithmetic circuit. Consider $C' \triangleq C + y \cdot z$ where $y, z$ are new variables. Clearly, $C'$ is decomposable iff $C \not\equiv 0$ (we also notice that $C'$ is multilinear iff $C$ is).

5.3.1 Some Corollaries

An immediate consequence of Theorem 11 is that there are efficient algorithms for polynomial decomposition in circuit classes for which efficient PIT algorithms are known. The proof of the following corollary is immediate given the state of the art PIT algorithms.

Corollary 5.3.3. Let $f(\overline{x})$ be a polynomial. We obtain the following algorithms.

1. If $f$ has degree $d$ and $m$ monomials then there is a polynomial time (in $m, n, d$) black-box algorithm for computing the indecomposable factors of $f$ (this is the circuit class of sparse polynomials, see e.g. [43]).

\footnote{It is not difficult to compute $\mathrm{var}(C)$ given Lemma 2.3.2 and in fact it is implicit in the proof of the theorem.}

101
2. If $f$ is computed by a depth-3 circuit with top fan-in $k$ (i.e. a $\Sigma\Pi\Sigma(k)$ circuit, see [17]) and degree $d$ then there is an $(nd)^{O(k^2)}$ non black-box algorithm for computing the indecomposable factors of $f$ (see [41]). In the black-box model there is an $n^{O(k^2 \log d)}$ time algorithm over finite fields and an $(nd)^{O(k)}$ time algorithm over $\mathbb{Q}$, for the task (see [59]).

3. If $f$ is computed by sum of $k$ Preprocessed Read-Once arithmetic formulas of individual degrees at most $d$, then there is an $(nd)^{O(k^2)}$ non black-box algorithm for computing the indecomposable factors of $f$ and an $(nd)^{O(k^2 + \log n)}$ black-box algorithm for the problem.

We now prove Corollary 5.1.2. We first give a more formal statement, again, in full generality, so that it can be applied to restricted models of arithmetic circuits as well.

**Corollary 5.1.2 restated:** Let $C$ be an arithmetic circuit class computing multilinear polynomials. Assume that there is a deterministic PIT algorithm that runs in time $T(s)$ when given as input a circuit of the form $C = C_1 + C_2 \times C_3$, where all the $C_i$-s $\in C$ are $n$-variate circuits of size $s$ and $C_2$ and $C_3$ are variable disjoint. Then, there is a deterministic algorithm that when given access (explicit or via a black-box) to an $n$-variate circuit $C' \in C$, of size $s$, runs in time $\text{poly}(n, T(s))$ and outputs the irreducible factors, $h_1, \ldots, h_k$, of the polynomial computed by $C'$. Moreover, each $h_i$ can be computed by a size $s$ circuit from $C$.

Conversely, assume there is a deterministic factoring algorithm that runs in time $T(s)$ when given as input a size $s$ circuit from $C$ (or even just a deterministic algorithm for the corresponding decision problem). Then $C$ has a PIT algorithm, for size $s$ circuits, of running time $O(T(s + 2))$.

In particular, if one of the problems has a polynomial time algorithms, namely $T(s) = \text{poly}(s)$, then so does the other. The two directions hold both in the black-box and non black-box models.

**Proof.** The claim is immediate from Theorem 11 and Observations 5.1.1 and 5.2.3. \qed
5.4 Extensions of Theorem 11

In this section we present some extensions to Theorem 11. The first is a non-adaptive polynomial decomposition algorithm. Note that the algorithm given in Theorem 11 is adaptive in nature. To obtain the non-adaptivity we require a stronger PIT algorithm.

**Theorem 5.4.1.** Let $C$ be a class of arithmetic circuits, defined over a field $F$. Assume that there is a hitting set $H$ (i.e. a black box PIT algorithm) for all circuits $C$ of the form $C = C_1 \times C_2 + C_3 \times C_4$, where the $C_i$-s are size $s$ circuits from some circuit class $C$. Then, there is a deterministic algorithm that when given access to a black-box containing a circuit $C' \in C$ of size $s$ runs in time $O(n^2 \cdot |H|)$ and outputs the indecomposable factors, $h_1, \ldots, h_k$ of the polynomial computed by $C'$. As before, each $h_i \in C$ and size$(h_i) \leq s$. In addition, the algorithm works in a non-adaptive fashion.

The second result concerns the problem of testing semi-decomposability.

**Definition 5.4.2.** Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. We say that $f$ is $(x_i, x_j)$-decomposable if $f$ can be written as $f = g \cdot h$ for polynomials $g$ and $h$ such that $i \in \text{var}(g) \setminus \text{var}(h)$ and $j \in \text{var}(h) \setminus \text{var}(g)$. We say that $f$ is semi-decomposable if there exists a pair $x_i \neq x_j$ such that $f$ is $(x_i, x_j)$-decomposable.

**Theorem 5.4.3.** Let $C$ be a class of arithmetic circuits, defined over a field $F$. Assume that there is a deterministic algorithm that when given access (explicit or black-box) to a circuit $C = C_1 \times C_2 + C_3 \times C_4$, where the $C_i$-s are $(n, s, d)$ circuits, runs in time $T(s, d)$ and decides whether $C \equiv 0$. Then, there is a deterministic algorithm that when given access (explicit or black-box) to an $(n, s, d)$ circuit $C' \in C$, runs in time $O(n^2 \cdot T(s, d))$ and decides whether the polynomial computed by $C'$ is semi-decomposable. In addition, if the answer is ‘yes’ then the algorithm outputs a pair of indices $i \neq j$ for which $f$ can be written as $f(X) = f_1(X_{[n]\setminus\{i\}}) \cdot f_2(X_{[n]\setminus\{j\}})$.

We note that in a contrary to polynomial decomposition, $(x_i, x_j)$-decomposability does not pose any requirements on other variables (besides $x_i$ and $x_j$). In that sense, semi-decomposability is a stronger notion and thus it is closer than polynomial decomposition to the general factorization problem.
Corollary 5.4.4. $f$ is $(x_i,x_j)$-decomposable if and only if it does not have an irreducible factor that depends on both $x_i$ and $x_j$.

The following definition and lemma explain how to check whether $f$ is semi-decomposable.

**Definition 5.4.5.** Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial and let $i, j \in [n]$. We define the commutator between $x_i$ and $x_j$ as
\[
\Delta_{ij} f = f \cdot f|_{x_i=y,x_j=w} - f|_{x_i=y} \cdot f|_{x_j=w}.
\]

**Lemma 5.4.6.** Let $i, j \in \text{var}(f)$ then $f$ is $(x_i,x_j)$-decomposable if and only if $\Delta_{ij} f \equiv 0$.

**Proof.** If $f = g \cdot h$ as above then the lemma follows by a simple substitution. For the other direction, assume that $f \cdot f|_{x_i=y,x_j=w} = f|_{x_i=y} \cdot f|_{x_j=w}$. From the uniqueness of factorization we can divide both sides of the equation by $f|_{x_i=y,x_j=w}$ and obtain a representation of the form $f = g \cdot h$ where $\text{var}(g) \subseteq \text{var}(f|_{x_j=w})$ and $\text{var}(h) \subseteq \text{var}(f|_{x_i=y})$. As $f$ does not depend on $y$ nor on $w$ then neither does $g \cdot h$. On the other hand, $f$ depends on both $x_i$ and $x_j$ and therefore so does $g \cdot h$. Since $j \not\in \text{var}(f|_{x_j=w})$ it follows that $j \not\in \text{var}(g)$. This implies that $j \in \text{var}(h)$ and consequently $j \in \text{var}(h) \setminus \text{var}(g)$. In a similar manner we obtain that $i \in \text{var}(g) \setminus \text{var}(h)$. This completes the proof.

The proof of Theorem 5.4.3 follows immediately from Lemma 5.4.6. We now give the proof of Theorems 5.4.1.

**Proof of Theorem 5.4.1.** In a similar fashion to Theorem 11 we first find the variable-partition $I_1, \ldots, I_k$ corresponding to the indecomposable factors of $f$ and then use a non-zero assignment to compute the factors themselves. The main difference is in the way in which we find the partition. We construct a graph $G$ with $[n]$ as the set of vertices, connecting $i$ and $j$ iff $\Delta_{ij} C' \not\equiv 0$. We can check this by evaluating $\Delta_{ij} C'$ on the set $\mathcal{H}$. It follows easily from Corollary 5.4.4 that the sets of the partition are exactly the connected components of $G$. Hence, the partition can be found in time $O(n^2 \cdot |\mathcal{H}|)$. By evaluating $C'$ on the points in $\mathcal{H}$ we can find a nonzero assignment to $C'$. We can now construct the indecomposable factors as explained above. Note that all queries to $C'$ are non-adaptive.
Using Theorem 5.4.1, instead of Theorem 11 we obtain a non-adaptive version of Corollary 5.3.3, of roughly the same running time, for the classes of \( m \)-sparse polynomials and \( \Sigma \Pi \Sigma(k) \) circuits. Similarly, we can obtain algorithms for testing semi-decomposability for those circuit classes (by applying Theorem 5.4.3). However, we can not use this approach for the class of sums of preprocessed read-once formulas. This is due to the stronger PIT algorithms needed for Theorems 5.4.1 and 5.4.3 that are currently not known for sums of read-once formulas.

We conclude this section with another corollary. In [31] Kabanets and Impagliazzo proved that PIT can be derandomized if and only if \( \text{NEXP} \subseteq \text{P}/\text{Poly} \). Later, [18] observed a similar result for bounded depth circuits. By combining their result with our Observation 5.1.1 we obtain the following corollary.

**Corollary 5.4.7.** The following three assumptions cannot be simultaneously true.

1. \( \text{NEXP} \subseteq \text{P}/\text{Poly} \),

2. Permanent is computable by polynomial size (bounded-depth) arithmetic circuits over \( \mathbb{Q} \),

3. There is a subexponential time algorithm for polynomial decomposition of (bounded-depth) arithmetic circuits over \( \mathbb{Z} \).

**Proof.** In [31], Kabanets and Impagliazzo proved that the following three conditions cannot be true simultaneously for arithmetic circuits (the bounded depth version was later observed in [18]).

1. \( \text{NEXP} \subseteq \text{P}/\text{Poly} \),

2. Permanent is computable by polynomial size (bounded-depth) arithmetic circuits over \( \mathbb{Q} \),

3. There is a subexponential time algorithm for PIT of (bounded-depth) arithmetic circuits over \( \mathbb{Z} \).

The claim now follows by combining this with Observation 5.1.1. For the bounded depth case we notice that if \( C \) is a depth-\( D \) circuit then \( C + y \cdot z \) is a circuit of depth \( \max(D, 2) \), and so we can apply Observation 5.1.1 here too. \( \Box \)
5.5 Concluding remarks

We showed a strong relation between PIT and polynomial decomposition. As noted, for multilinear polynomials, decomposition is the same as factorizing. Thus, for multilinear polynomials PIT and factorization are equivalent up to a polynomial overhead. It is an interesting question whether such a relation holds for general polynomials. Namely, whether PIT is equivalent to polynomial factorization.

We note that in restricted models it may be the case that a polynomial and one of its factors will have a different complexity. For example, consider the polynomial

\[ f(x_1, \ldots, x_k) = \prod_{i=1}^{k} (x_i^k - 1) + \prod_{i=1}^{k} (x_i - 1) = \prod_{i=1}^{k} (x_i - 1) \cdot \left( \prod_{i=1}^{k} (x_i^{k-1} + \ldots + 1) + 1 \right). \]

Then \( f \) has \( 2^{k+1} - 1 \) monomials, but one of its irreducible factors has \( k^k \) monomials. Thus, for \( k = \log n \) we can compute \( f \) as a sparse polynomial, but some of its factors will not be sparse (the fact that \( f \) has only \( \log n \) variables is not really important as we can multiply \( f \) by \( x_{\log n+1} \cdot \ldots \cdot x_n \) and still have the same problem). Thus, it is also interesting to understand whether it is even possible to have some analog of our result for the factorization problem in restricted models. This question touches of course the interesting open problem of whether the depth of a factor can increase significantly with respect to the depth of the original polynomial.
Chapter 6

Read-Once Reconstruction and Testing

An arithmetic read-once formula (ROF for short) is a formula (a circuit whose underlying graph is a tree) in which the operations are \{+, \times\} and such that every input variable labels at most one leaf. A preprocessed ROF (PROF for short) is a ROF in which we are allowed to replace each variable \(x_i\) with a univariate polynomial \(T_i(x_i)\). In this chapter we study the problems of giving deterministic reconstruction algorithms for preprocessed ROFs. We obtain a deterministic reconstruction algorithms for preprocessed read-once formulas. The running time of the algorithm is \((nd)^{O(\log n)}\) for preprocessed read-once formulas of individual degrees at most \(d\). We also extend the algorithm to operate in a deterministic, non-adaptive manner. To the best of our knowledge our results give the first sub-exponential time reconstruction algorithms for ROFs.

Another question that we study is the following generalization of the polynomial identity testing problem. Given an arithmetic circuit computing a polynomial \(P(\bar{x})\), decide whether there is a PROF computing \(P(\bar{x})\). If there is such a formula then output it. Otherwise output “No”. We call this question the read-once testing problem (ROT for short). Previous (randomized) algorithms for reconstruction of ROFs imply that there exists a randomized algorithm for the read-once testing problem. In this work we show that most previous algorithms for polynomial identity testing can be strengthen to yield algorithms for the read-once testing problem. In particu-
lar we give ROT algorithms for the following circuit classes: Depth-2 circuits (circuits computing sparse polynomials), Depth-3 circuits with bounded top fan-in and sums of $k$ ROFs. The running time of the ROT algorithm is essentially the same running time as the corresponding PIT algorithm for the class.

The main tool in most of our results is a new connection between polynomial identity testing and reconstruction of read-once formulas. Namely, we show that in any model that satisfies some “nice” closure properties (for example, a partial derivative of a polynomial computed by a circuit in the model, can also be computed by a circuit in the model) and that has an efficient deterministic polynomial identity testing algorithm, we can also answer the read-once testing problem.

The results of this chapter are based on the works [64, 65].

6.1 Introduction

Let $\mathbb{F}$ be a field. The reconstruction problem for arithmetic circuits is defined as follows. Given a black-box (oracle) access a polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ computed by an arithmetic circuit from some circuit class $\mathcal{C}$, output a circuit from $\mathcal{C}$ that computes $P$. A reconstruction algorithm is efficient if its number of queries and running time are polynomial in the size of the representation of $P$ in terms of the class $\mathcal{C}$. The reconstruction problem can be considered to be the algebraic analog of the learning problem. A partial case of reconstruction problem is the interpolation problem, where the goal is to find the monomial representation of the circuit. This can be seen as a reconstruction problem when the class $\mathcal{C}$ is the class of depth-2 (see e.g [43]).

The problem has a strong relation to the Polynomial Identity Testing (PIT) problem, in which we need to determine whether a given arithmetical circuit $C(\bar{x})$ computes the zero polynomial $^1$ A deterministic black-box PIT algorithm for a circuit class $\mathcal{C}$ can be viewed as a hitting set. That is, a set of points $\mathcal{H}$ such that if a circuit from $\mathcal{C}$ evaluates to zero over $\mathcal{H}$ then it must compute the zero polynomial. One of the main properties of such $\mathcal{H}$ is that the values of any circuit from $\mathcal{C}$ on $\mathcal{H}$ describe that circuit uniquely, in the

$^1$ As usual in this case, we ask whether $C(\bar{x})$ is the identically zero polynomial and not the zero function over $\mathbb{F}$. 

108
sense that two circuits that agree on $\mathcal{H}$ must compute the same polynomial, as their difference evaluates to zero over $\mathcal{H}$. Yet, such a set does not provide us an efficient algorithm for reconstructing circuits from $\mathcal{C}$. On the other hand, it is easy to see that a deterministic reconstruction algorithm can be used as a black-box PIT algorithm.

It was shown that giving efficient deterministic PIT algorithms implies lower bounds for arithmetic circuits [31, 1]. Consequently, we can conclude that reconstruction problem is at least as hard as proving circuit lower bounds. In view of the difficulty of the problem for general arithmetic circuits, research focused on restricted models such as: sparse polynomials [43], depth-3 circuits with bounded top fan-in [63, 38] and non-commutative formulas [10, 42], multilinear depth-4 circuits with top fan-in 2 [25].

The focus of this chapter is on models related to arithmetic read-once formulas (ROF for short). An arithmetic read-once formula is a formula (a circuit in which the fan-out of every gate is at most 1) in which the operations are $\{+, \times\}$ and such that every input variable labels at most one leaf. A preprocessed ROF (PROF for short) is a ROF in which we are allowed to replace each variable $x_i$ with a univariate polynomial $T_i(x_i)$. Although read-once formulas form a very restricted model of computation they received a lot of attention both in the boolean world [35, 6, 15] and in the algebraic world [27, 14, 12, 13]. However, no deterministic sub-exponential time reconstruction algorithm for arithmetic ROFs was known prior to this work. This sad state of affairs implies that if we want to give efficient algorithms for bounded-depth circuits or for multilinear formulas then we should first try to find algorithms for read-once formulas. In view of this we focus in this work on ROFs and, as a result give the first deterministic sub-exponential time reconstruction algorithm PROFs.

**Theorem 12.** Given a black-box (oracle) access to preprocessed read-one formula $\Phi$ on $n$ variables and individual degrees (of the preprocessing) at most $d$ there is a deterministic $(nd)^{O(\log n)}$ time algorithm that reconstructs $\Phi$. (that is, there is an algorithm that outputs a PROF computing the same polynomial). In case that $|\mathbb{F}|$ is too small we make queries to the black-box from a polynomial size extension field.

Moreover, there is an algorithm that preforms that task in a non-adaptive manner, that is, when each query to the black-box is independent of the
results of the previous ones.

**Theorem 13.** Given a black-box (oracle) access to preprocessed read-one formula \( \Phi \) on \( n \) variable and individual degrees (of the preprocessing) at most \( d \) there is a deterministic \((nd)^{O(\log n)}\) time algorithm that reconstructs \( \Phi \), moreover, the algorithm queries \( \Phi \) on a fixed set of point of size \((nd)^{O(\log n)}\).

In case that \( |F| \) is too small we make queries to the black-box from a polynomial size extension field.

We also re-establish the result of [27] that gives a randomized polynomial-time reconstruction algorithm for ROF. In fact, we show that a similar approach will work for PROFs as well.

**Theorem 14.** Given a black-box (oracle) access to preprocessed read-one formula \( \Phi \) on \( n \) variable and individual degrees (of the preprocessing) at most \( d \) there is a polynomial-time randomized algorithm that reconstructs \( \Phi \) w.h.p.

In addition to reconstruction of preprocessed read-once formulas we are interested in a generalization of the problem, that we call the **read-once testing problem** (ROT for short).

**Problem 6.1.1.** Given a circuit \( C \) (maybe as a black-box) computing some polynomial \( P(\bar{x}) \), decide whether \( P \) can be computed by an (arithmetic) preprocessed read-once formula, and if the answer is positive then compute a PROF for it.

This problem is a generalization of the PIT problem, as the zero polynomial is computable by a read-once formula. Moreover, given a read-once formula it is easy to check whether it computes the zero polynomial. Similarly to the PIT problem, there is an efficient randomized algorithm for the read-once testing problem. Indeed by the results of [27] (or Theorem 14) there is a randomized algorithm for reconstructing a read-one formula, given as a black-box, and then invoking the Schwartz-Zippel randomized identity testing algorithm [69, 61] we can check whether the read-one formula that we computed, computes the same polynomial as the one in the

---

\(^2\)It may be more accurate to denote this problem as **preprocessed read-once testing and reconstructing**, but for brevity we use read-once testing.
black-box. The main question is then whether we can find an efficient deterministic algorithm for the read-once testing problem. We show that if we have an efficient PIT algorithm for a circuit class $C$, that satisfies some “nice” closure properties, then we can also solve the read-once testing problem for the class efficiently.

If fact, the same argument shows that an efficient randomized reconstruction algorithm for a circuit class $C$ implies an efficient randomized algorithm for the corresponding “$C$-testing” problem (i.e. checking whether a given arithmetic circuit belongs to some specific class $C$). Yet, in light of the difficulty of the PIT problem, no such connection is known in the deterministic scenario. For example deterministic “Sparsity-testing” (i.e. determining whether a given circuit $C$ computes a sparse polynomial) is a long standing open question posed by von zur Gathen [68]. Formally we prove the following theorem.

**Theorem 15.** Let $C$ be a class of arithmetic circuits. Denote with $C_V$ the class of circuits that contains all circuits of the form

$$C_1 + C_2 + C_3 \times C_4$$

where the $C_i$’s are circuits from $\mathcal{L}(C)$ \footnote{$\mathcal{L}(C)$ is the class of all circuits of the form $\alpha \cdot C + \beta$ for $\alpha, \beta \in F$ and $C \in C$. See Definition 2.1.15.} and $C_2, C_3$ and $C_4$ are variable disjoint. Assume that there is a deterministic PIT algorithm that runs in time $T(n, d, s)$, when given access (explicit or via a black-box) to a circuit in $n$-variables, of size $s$ and degree $d$, that belongs to $C_V$. Then there is a deterministic algorithm that runs in time $\text{poly}(n, d, s, T(n, d, s))$ that when given access (explicit or via a black-box) to an $n$-variate circuit of size $s$ and degree $d$ from $C$, solves the ROT problem.

Note that for most circuit classes for which we have efficient PIT algorithms, we also have efficient PIT algorithms for their “closure”, $C_V$. As a corollary we get that all previous PIT algorithms can be generalized to yield algorithms for ROT. We demonstrate this point by obtaining the following results:

**Theorem 16.** Let $F = F_1 + \ldots + F_k$ be a sum of $k$ preprocessed read-once formulas over $F[x_1, \ldots, x_n]$ with individual degrees bounded by $d$. Then:
- There is a deterministic algorithm that given the PROFs $F_i$'s explicitly solves the ROT problem for $F$ in time $\mathcal{O}(k)$.  
- There is a deterministic algorithm that given a black-box access to $F$ solves the ROT problem for $F$ in time $\mathcal{O}(k+\log n)$.  

The following result strengthens many reconstruction algorithms that were given for the model of sparse polynomials ([11, 23, 43] to name just a few). We are not sure however whether our result is new, but we give it anyway (see last sentence of paragraph 4 on page 707 of [14] that refers to [44]).

**Theorem 17.** Given a black-box access to a polynomial $P$ of degree $d$ with $m$ monomials (i.e. a polynomial computed by a depth-2 arithmetic circuit with $m$ multiplication gates) there is a polynomial (in $n, m, d$) time algorithm that solves the ROT problem for $P$.  

Our next result solves the ROT problem for depth-3 circuits with bounded top fan-in. This result generalizes the results of [17, 41, 37, 40, 60] and others, who gave deterministic identity testing algorithms for this model.

**Theorem 18.** Given a black box holding a depth-3 circuit $C$ of degree $d$ with $k$ multiplication gates (i.e. a $\Sigma\Pi\Sigma(k, d)$ circuit) there is an $\mathcal{O}(d^{O(k^2)})$ time algorithm that solves the ROT problem.

### 6.1.1 Comparison to Previous Works

Read-once arithmetic formulas were studied before in the context of learning theory and exact learning algorithms were given to them. We shall now discuss the different models in which it was studied and highlight the differences from our work.

In [27] learning algorithms for read-once arithmetic formulas that use membership and equivalence queries were given. A membership query to a ROF $\Phi(\bar{x})$ is simply a query that asks for the value of $\Phi(\bar{x})$ on a specific input. An equivalence query on the other hand, gives the oracle a certain hypothesis, $h(\bar{x})$, and the oracle either answers “equal” if $\Phi \equiv h$ or returns an input $\bar{\alpha}$ such that $\Phi(\bar{\alpha}) \neq h(\bar{\alpha})$. In this language, our results use only membership queries.
In [14] a different approach was taken. They considered randomized learning algorithms that only use membership queries. It is not difficult to see that using randomness one does not need to use equivalence queries any more. The learning algorithm of [14] can reconstruct, with high probability, arithmetic read-once formulas that also use division gates (and not just \(+, \times\) gates). They also considered a model in which justifying assignments (a notion that we later define) are given in advance to the algorithm. This result was later generalized by [12] who gave a randomized reconstruction algorithm for read-once formulas that use additions, multiplications, divisions and exponentiations.

We use some of the techniques of [27, 14] in our read-once testing algorithms. Specifically, we use their ideas in order to construct a candidate PROF that \textit{may} compute the same polynomial as the one computed by the given circuit (recall Problem 6.1.1). The main difficulty is then to verify deterministically that this PROF indeed computes the same polynomial as the underlying circuit. Indeed, the main difference between the reconstruction problem and the read-once testing problem is that in the reconstruction setting we are guaranteed that there exists a PROF for the given circuit, so if we use the results of [27, 14] we are guaranteed that we have the correct PROF, whereas in the read-once testing setting we first have to check whether the polynomial can be computed by a PROF and only then we can try and find the formula. Thus, when we use the reconstruction algorithm we get a PROF and then we still have to verify that it does compute the same polynomial as the given circuit (or black-box). This may seem like a small point, however this is exactly the reason that originally we could not use the results of [41] to get a polynomial time black-box algorithm in Theorem 18, and instead settle for the results of [17, 37] due to the additional structure they guarantee. \footnote{The original paper predated the paper of [60] that gives polynomial-time black-box PIT algorithm for depth-3 circuits. In this version of the paper we use the new PIT algorithm.}

### 6.1.2 Organization

In Section 6.2 we prove Theorems 12, 13 and 14. Next, in Section 6.3 we show how to convert PIT algorithms to ROT algorithms and prove Theorem 15. In Section 6.4 we show how to use Theorem 15 to prove Theorems 16, 17
and 18. Section 6.5 concludes the paper with several basic algorithms for PROFs (some of them already known but we give them for completeness).

6.2 Reconstruction of a PROF

In this section we discuss the problem of PROF reconstruction: given a black-box (oracle) access to a PROP $P$ we wish to construct a PROF $\Phi$ such that $\Phi \equiv P$. We start by describing a reconstruction for the case of $\bar{0}$-justified PROPs. An arbitrary PROP need not to be $\bar{0}$-justified, however it can be made $\bar{0}$-justified via a proper shifting (see Lemma 2.1.4). We show how to compute this shifting efficiently, thus establishing reconstruction algorithms in three scenarios: randomized, deterministic adaptive and deterministic non-adaptive.

Let $d$ be a bound on the individual degrees of the PROP in question. We assume the $|\mathbb{F}| > d$, otherwise we allowed to the query the polynomial on some extension field. This assumption is necessary even for the case of univariate polynomial interpolation. Throughout this section we fix a set $W \subseteq \mathbb{F}$ of $d + 1$ distinct elements with the condition that $0 \in W$.

6.2.1 Reconstruction of a $\bar{0}$-justified PROF

Let $P$ be a $\bar{0}$-justified PROP with individual degrees (of the preprocessing) bounded by $d$. Theorem 3.4.4 shows that $P$ is uniquely defined by its values on the set $A^0(W)$ (recall the definition in Section 2.2.1). This implies that $P$ could be reconstructed given those values. Yet, in general the reconstruction algorithm may not be efficient. In this section we give an efficient reconstruction algorithm for $\bar{0}$-justified PROPs, which queries the polynomial on the set $A^0_3(W)$ only.

**Algorithm 6** Reconstruct $\bar{0}$-justified PROF

**Input:** $\bar{0}$-justified PROP $P$ given as a black-box (oracle) access

**Output:** PROF $\Phi$ such that $\Phi \equiv P$.

1: Learn the Preprocessing and $\text{var}_0(P)$
2: Construct the gates-graph of $P$
3: Construct a PROF $\Phi$ by recursively constructing its sub-formulas
The following lemma is summarizes the algorithm.

**Lemma 6.2.1.** Given an oracle access to a \( \bar{0} \)-justified PROP \( P \in \mathbb{F}[x_1, \ldots, x_n] \) with individual degrees bounded by \( d \), there exists an algorithm that constructs a (non-degenerate) PROF \( \Phi \) such that \( \Phi \equiv P \) by querying \( P \) on the set \( A^\bar{0}_n(W) \) only. The running time of the algorithm is \( \text{poly}(n, d) \). A high level description of the algorithm is given in Algorithm 6.

We now describe deeply each of the steps of the algorithm.

### Learning the Preprocessing and \( \text{var}_0(P) \)

Given an access to a PROP \( P(\bar{x}) = Q(T(\bar{x})) \) the first step is to identify (learn) the preprocessing \( T(\bar{x}) \) and the set \( \text{var}_0(P) \). Lemma 3.2.32 suggests that a preprocessing can be defined up a multiplicative an additive constants. We will output a standard preprocessing. For that purpose we consider

\[
P_i(x_i) \triangleq P(0, \ldots, 0, x_i, 0, \ldots, 0) \quad \text{for} \quad i \in [n]
\]

that is, \( P_i(x_i) \) is the univariate polynomial resulting from setting all variables besides \( x_i \) to 0. The next observation is immediate in the light of Lemma 3.2.32. Recall that for every \( \bar{0} \)-justified polynomial \( P \) it holds that \( \text{var}_0(P) = \text{var}(P) \).

**Observation 6.2.2.** For each \( i \in [n] \) it holds that \( P_i(x_i) = a_i \cdot T_i(x_i) + b_i \) for some \( a_i, b_i \in \mathbb{F} \), in addition \( a_i \neq 0 \) iff \( x_i \in \text{var}_0(P) \).

For the sake of future steps we also find a witness \( \bar{\alpha} \) for \( P \). The following summarizes this step:

**Lemma 6.2.3.** Given a black-box access to a \( \bar{0} \)-justified PROP \( P \) in time \( \text{poly}(n, d) \) Algorithm 7 outputs a standard preprocessing \( T(\bar{x}) \) of \( P \), the set \( \text{var}_0(P) \) and a witness \( \bar{\alpha} \) for \( P \). In addition, the algorithm does so by querying \( P \) on the set \( A^\bar{0}_1(W) \) only.

**Proof.** First of all note that as the degree \( P_i \) is bounded by \( d \), the set \( A^\bar{0}_1(W) \) contains enough points to interpolate it. It is easy to compute a standard \( T_i \) from \( P_i \). From Observation 6.2.2 \( x_i \in \text{var}_0(P) \) iff the resulting \( T_i \) is a non-constant polynomial, thus for every non-constant \( T_i \) there exists \( \alpha_i \in W \) such that \( T_i(\alpha_i) \neq 0 \). All the above actions can be carried out in time \( \text{poly}(n, d) \) using the queries of \( P \) on the set \( A^\bar{0}_1(W) \) only. \(\square\)
Algorithm 7 Learn the Preprocessing, \( \text{var}_0(P) \) and a Witness

**Input:** a 0-justified \( P(\bar{x}) = Q(T(\bar{x})) \) given as a black-box access,

**Output:** A standard preprocessing \( T(\bar{x}) = (T_1(x_1), \ldots, T_n(x_n)) \), \( \text{var}_0(P) \), a witness \( \bar{\alpha} \) for \( P \).

1. for \( i \in [n] \) do
2. Interpolate \( P_i(x_i) \) as a univariate polynomial of degree \( d \)
3. Compute \( T_i \) from \( P_i \)
4. If \( T_i \) is non-constant find \( \alpha_i \in W \) such that \( T_i(\alpha_i) \neq 0 \)
5. Compute \( \text{var}_0(P) \)
6. \( P \leftarrow P|_{x[n]\text{\textbackslash var}_0(P) = 0[n]\text{\textbackslash var}_0(P)} \)

As \( P \) is 0-justified we can set \( P \overset{\Delta}{=} P|_{x[n]\text{\textbackslash var}_0(P) = 0[n]\text{\textbackslash var}_0(P)} \) and assume w.l.o.g that \( \text{var}(P) = \text{var}_0(P) = [n] \), by renaming the variables. If \( P \) is constant or univariate, (i.e. \( |\text{var}_0(P)| \leq 1 \)) we are done. Otherwise, \( P \) contains at least one gate, and we continue to the next step of Algorithm 6.

After this step we, in some sense, will be able to “forget” about the preprocessing and reconstruct \( P \) as if it were a (non-preprocessed) ROP. In fact, the witness \( \bar{\alpha} \) and Lemma 3.2.34 will allow us to access the backbone ROF of \( P \).

Constructing the Gates Graph

This step will allow us to recursively unfold the structure of the backbone ROF of a given PROP. As it was mentioned earlier, after the previous step we in some sense, can treat \( P \) as if there were no preprocessing involved. (i.e. “assume” that \( T_i(x_i) = x_i \)). In particular, the preprocessing does not affect the gates graph. Thus, we define the gates graph \( G_P \) of a PROP to be the gates graph of its backbone ROP. We will use Lemma 3.2.7 in order to construct the graph.

Algorithm 8 Construct The Gates Graph

**Input:** a 0-justified PROP \( P(\bar{x}) \) given as a black-box access

**Output:** \( G_P \)

For each pair \( i \neq j \in [n] \) add \( (i, j) \) to \( G_P \) iff \( \frac{\partial^2 P}{\partial x_i \partial x_j}(\bar{0}) \neq 0. \)
Lemma 6.2.4. Given a black-box access to a $0$-justified PROP $P$ in time $O(n^2)$ Algorithm 8 constructs $G_P$ - the gates graph of the $P$, by querying $P$ on the set $A^2_0(W)$ only.

Proof. Let $P(\bar{x}) = Q(T(\bar{x}))$. Since $T(\bar{x})$ is standard Lemma 3.2.34 implies $Q$ is $0$-justified. Therefore, by Lemmas 3.2.7 and 3.2.27:

$$(i,j) \in G_P \iff \frac{\partial^2 Q}{\partial x_i \partial x_j} \neq 0 \iff \frac{\partial^2 Q}{\partial x_i \partial x_j}(\bar{0}) \neq 0$$

On the other hand, by Lemma 3.2.34

$$\frac{\partial^2 P}{\partial \alpha_i x_i \partial \alpha_j x_j}(\bar{0}) = \frac{\partial^2 Q}{\partial z_i \partial z_j} \bigg|_{z = T(\bar{0})} \cdot T_i(\alpha_i) \cdot T_j(\alpha_j) = T_i(\alpha_i) \cdot T_j(\alpha_j) \cdot \frac{\partial^2 Q}{\partial z_i \partial z_j}(\bar{0}).$$

As $T_i(\alpha_i), T_j(\alpha_j) \neq 0$ the claim follows. From the definition, for each $i, j$ the expression $\frac{\partial^2 P}{\partial \alpha_i x_i \partial \alpha_j x_j}(\bar{0})$ can be computed by querying $P$ on $A^2_0(W)$. \qed

Having the gates graph at hand, we can recursively reconstruct the formula gate-by-gate, according to the labelling of the top gate ($+$, $\times$) of the formula. According to Lemma 3.2.35 every PROP $P(\bar{x})$ that has at least one gate can be presented in exactly one of the two forms $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$ or $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$, depending of the labeling of the top gate. Therefore, in order to construct a PROF computing $P$ we will find a (possible) partition of the variables $L \cup R = \text{var}(P)$ such that $L = \text{var}(P_1)$ and $R = \text{var}(P_2)$, recursively construct formulas $\Phi_i \equiv P_i$ for $P_i$’s and finally “glue” them together to obtain a formula for $P$. See Section 3.2.2 for more info.

“$+$” Top Gate

In this case a partition can be obtained by considering the connected components of $G_P$. The idea is summarized in the following observation:

Observation 6.2.5. Let $Q(\bar{x})$ be a multilinear polynomial and let $L \cup R = \text{var}(Q)$ be a non-trivial partition of $\text{var}(Q)$. Then $Q$ can be represented in the form $Q(\bar{x}_L, \bar{x}_R) = Q_L(\bar{x}_L) + Q_R(\bar{x}_R)$ iff for each $i \in L, j \in R$ it holds that $\frac{\partial^2 Q}{\partial x_i \partial x_j} \equiv 0$.

This gives rise to the following algorithm:
Algorithm 9 Top Gate + Case
Input: 0-justified PROP $P(\bar{x})$ given as a black-box access,
Output: PROF $\Phi \equiv P$
1: Find a partition $L \cup R = \text{var}(P)$ on $G_P$ 
\{ $P$ can be regarded as $P(\bar{x}_L, \bar{x}_R)$ \}
2: $\Phi_L(\bar{x}_L) \leftarrow \text{Reconstruct}(P(\bar{x}_L, \bar{0}))$, $\Phi_R(\bar{x}_R) \leftarrow \text{Reconstruct}(P(\bar{0}, \bar{x}_R))$
3: $\Phi(\bar{x}_L, \bar{x}_R) \leftarrow \Phi_L(\bar{x}_L) + \Phi_R(\bar{x}_R) - P(\bar{0}, \bar{0})$

Lemma 6.2.6. Given a black-box access to a 0-justified PROP $P$ computed by a PROF with a Top Gate labelled as “+”, Algorithm 9 outputs a PROF $\Phi$ that computes $P$. The running time of the algorithm $\text{poly}(n)$ in addition to time required to reconstruct $\Phi_R(\bar{x}_R)$ and $\Phi_L(\bar{x}_L)$. The algorithm queries $P$ on the set $A^3_0(W)$ only.

Proof. Let $P(\bar{x}) = Q(T(\bar{x}))$. By Lemma 3.2.35 $P$ can be written as: $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$ and $G_P$ is disconnected, hence $L$ and $R$ form a non-trivial partition of $\text{var}(P)$. By Observation 6.2.5 we can assume w.l.o.g that $L = \text{var}(P_1)$ and $R = \text{var}(P_2)$, and hence $\Phi_L(\bar{x}_L) = P_1(\bar{x}_L) + P_2(\bar{0})$ and $\Phi_R(\bar{x}_R) = P_1(\bar{0}) + P_2(\bar{x}_R)$. The last follows since $P(\bar{0}) = P_1(\bar{0}) + P_2(\bar{0})$. 

“×” Top Gate

Let $P(\bar{x}) = Q(T(\bar{x}))$. The case of $\times$ is more subtle. A possible way to find a variable partition is by considering the connected components of $G^+_Q$ (see Definition 3.2.17). However, constructing $G^+_Q$ can seem like an involved task. Instead, we are going to construct another graph $G'_P$ that has the exact same connected components. First of all, note that $G^+_Q$ is a subgraph of $G^+_Q$. However, $G^+_Q$ provides us only a partial information regarding the actual partition. For example, for $Q = ((x_1+1)(x_2+1)+1)(x_3+1)+1$, $G^+_Q$ will be an empty graph. To tackle this problem we are going to extend $G^+_Q$ filling in the “missing” information. Note that $P$ and $Q$ essentially depend on the same sets of variables, thus we can abuse notation and say that $\text{var}(P) = \text{var}(Q)$.

Lemma 6.2.7. Let $P(\bar{x}) = Q(T(\bar{x}))$ be a 0-justified PROP and $T(\bar{x})$ a standard preprocessing. Take $i \neq j \in \text{var}(P)$. Then $i \sim j$ in $G^+_Q$ if and only if $i \sim j$ in $G'_P$, when the construction of $G'_P$ is specified in Algorithm 10.
**Algorithm 10** Top Gate × Case

**Input:** 0-justified PROP $P(\bar{x})$ given as a black-box access,
**Output:** PROF $\Phi \equiv P$

Construct the graph $G'_P$:

1: $G'_P \leftarrow G_Q$

2: For each $i \neq j \neq k \in \text{var}(P)$ add $(i, j)$ to $G'_P$ iff $\Delta_{ij}^\delta \left( \frac{\partial P}{\partial \alpha_k} \right)(\bar{0}) \neq 0$.

**Learning:**

1: Find a partition $L \cup R = \text{var}(P)$ on $G'_P$ { $P$ can be regarded as $P(\bar{x}_L, \bar{x}_R)$ }

2: Chose $i \in L, j \in R$

3: $c \leftarrow P(\bar{0}, \bar{0}) - \frac{\partial P}{\partial \alpha_i}(\bar{0}) \cdot \frac{\partial P}{\partial \alpha_j}(\bar{0}) \div \frac{\partial^2 P}{\partial \alpha_i \partial \alpha_j}(\bar{0})$

4: $\Phi_L(\bar{x}_L) \leftarrow \text{Reconstruct}(P(\bar{x}_L, \bar{0}) - c)$, $\Phi_R(\bar{x}_R) \leftarrow \text{Reconstruct}(P(\bar{0}, \bar{x}_R) - c)$

5: $\Phi(\bar{x}_L, \bar{x}_R) \leftarrow \frac{1}{P(\bar{0}, \bar{0})} \cdot \Phi_L(\bar{x}_L) \times \Phi_R(\bar{x}_R) + c$

**Proof.** Note that $Q$ is a 0-justified ROP. First, we show that $G'_P$ is a subgraph of $G^F_Q$. Let $(i, j) \in G'_P$. If $(i, j) \in G^+_Q$ then $(i, j) \in G^F_{PFACT}$ as $G^+_Q$ is a subgraph of $G^F_{QFACT}$. Otherwise, there exists $k \neq i, j$ such that

\[
\Delta_{ij} \left( \frac{\partial Q}{\partial \alpha_k} \right)(\bar{0}) = \Delta_{ij} \left( \frac{\partial Q}{\partial \alpha_k} \right) |_{\bar{z}=T(\bar{0})} \cdot T_i(\alpha_i) \cdot T_j(\alpha_j) \cdot T_k(\alpha_k) = \Delta_{ij}^\delta \left( \frac{\partial P}{\partial \alpha_k} \right)(\bar{0}) \neq 0.
\]

By Lemma 3.2.18 $(i, j)$ must be in $G^F_{QFACT}$. For the second direction, let $i \sim j$ be in $G^F_{QFACT}$. If $i \sim j$ in $G^+_Q$ then $i \sim j$ in $G'_P$ as $G^+_Q$ is a subgraph of $G'_P$. Now assume that $i \not\sim j$ in $G^+_Q$. By Corollary 3.2.20 there exists $k \neq i, j$ such that $\Delta_{ij} \left( \frac{\partial Q}{\partial \alpha_k} \right)(\bar{0}) \neq 0$. By repeating the previous argument we get that the edge $(i, j)$ will be added to $G'_P$ thus connecting $i$ and $j$. 

The next lemma summarizes the algorithm.

**Lemma 6.2.8.** Given a black-box access to a 0-justified PROP $P$ computed by a PROF with a Top Gate labelled as “×”, Algorithm 10 outputs a PROF $\Phi$ that computes $P$. The running time of the algorithm poly($n$) in addition
to time required to reconstruct $\Phi_R(\bar{x}_R)$ and $\Phi_L(\bar{x}_R)$. The algorithm queries $P$ on the set $A_3^3(W)$ only.

Proof. Similar to the proof of Lemma 6.2.6. Note that $\Delta_{ij}^{\delta} \left( \frac{\partial P}{\partial x_k x_k} \right)(\bar{0})$ can be computed given the values of $P$ on the set $A_3^3(W)$.

Conclusion

We can now prove Lemma 6.2.1.

Proof of Lemma 6.2.1. The proof is by induction on $\operatorname{var}_0(P) = \operatorname{var}(P)$. By Lemma 6.2.3 we can learn the set $\operatorname{var}_0(P)$ and handle the case when $|\operatorname{var}_0(P)| \leq 1$. Now, assume $|\operatorname{var}_0(P)| \geq 1$. This implies that any PROF computing $P$ must have at least one gate. Lemma 6.2.4 allow us to construct the Gates Graph, from which we can learn the labeling of the top gate (Lemma 3.2.5 Property 3). From this point on Lemmas 6.2.6 and 6.2.8 ensure the correctness of construction. We note that in each step we query $P$ on the set $A_3^3(W)$ only. The claim regarding the running time follows from previous lemmas.

6.2.2 Reconstruction of a General PROF

We have previously seen a reconstruction algorithm for $\bar{0}$-justified PROPs. Yet, an arbitrary PROP need not to be such. A natural question to ask is “what would happen if we executed that algorithm on a general (not necessarily $\bar{0}$-justified ) PROF”? “Will the outcome of such an execution be somewhat meaningful”? The following lemma provides answers to those questions. Intuitively, this can be seen as that the algorithm can only “hide” variables.

Lemma 6.2.9. Let $P$ a PROP with individual degrees bounded by $d$ given as a black-box access. Then the execution of algorithm described by Algorithm 6 will output a (non-degenerate) PROF $\Phi$ satisfying $\Phi \equiv P|_{x_n \mid \operatorname{var}_0(P) = \bar{0}_n \setminus \operatorname{var}_0(P)}$.

Proof. Denote $P' \triangleq P|_{x_n \mid \operatorname{var}_0(P) = \bar{0}_n \setminus \operatorname{var}_0(P)}$. Observe that $\operatorname{var}(P') = \operatorname{var}_0(P) = \operatorname{var}_0(P')$ hence $P'$ is a $\bar{0}$-justified PROP. To complete the proof note that the execution of the of the algorithm on $P$ and $P'$ will be the same.
An arbitrary PROP need not to be 0-justified, however it can be made 
0-justified via a proper shifting (see Lemma 2.1.4). Hence we have the 
following:

**Theorem 6.2.10.** Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a PROP with individual degrees 
bounded by $d$ given as a black-box access and let $\bar{a} \in \mathbb{F}^n$ be a justifying 
assignment of $P$. Then there exists an algorithm that constructs a (non- 
degenerate) PROF $\Phi$ such that $\Phi \equiv P$ by querying $P$ on the set $\bar{a} + A^\#(W)$ 
only. The running time of the algorithm is $\text{poly}(n, d)$.

**Proof.** Invoke the algorithm in Lemma 6.2.1 on $P'(\bar{x}) \triangleq P(\bar{x} + \bar{a})$. By 
Lemma 2.1.4 $P'$ is 0-justified PROP, therefore, the algorithm will return a 
PROF $\Phi$ such that $\Phi(\bar{x}) \equiv P'(\bar{x}) \triangleq P(\bar{x} + \bar{a})$. Output $\Phi(\bar{x} - \bar{a})$.

Consequently, to extend the algorithm to handle a general PROP $P$ we first need to find a justifying assignment $\bar{a}$ for $P$ and then apply Theorem 
6.2.10. We implement this idea in three scenarios proving Theorems 12, 13 
and 14.

**Randomized PROF Reconstruction**

The simplest way to get an justifying assignment is to pick one a random. 
This idea was used in the previous works on ROF reconstruction ([27, 14]). 
We give it here for completeness. We first state Schwartz-Zippe lemma:

**Lemma 6.2.11** ([69, 61]). Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial 
with individual degrees bounded by $d$. Then $\Pr_{\bar{a} \in \mathbb{F}^n}[P(\bar{a}) = 0] \leq \frac{n d^2}{|\mathbb{F}|}$.

Since a justifying is simply a common non-zero assignment of several 
polynomials, we get the following corollary:

**Corollary 6.2.12.** Let $n, d \geq 1$. Assume that $|\mathbb{F}| > 4n^2d$. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial with individual degrees bounded by $d$. Then $\Pr_{\bar{a} \in \mathbb{F}^n}[\bar{a} \text{ is a justifying assignment of } P] \geq \frac{3}{4}$.

We can now prove Theorem 14.

**Proof of Theorem 14.** The algorithm will go as following: pick an assign- 
ment $\bar{a} \in \mathbb{F}^n$ at random and apply Theorem 6.2.10. The claim follows from 
the previous corollary and Theorem 6.2.10.
Deterministic PROF Reconstruction

We are now ready to prove Theorem 12.

Proof of Theorem 12. The algorithm will go as following: Given a PROP $P$ invoke the algorithm in Lemma 2.3.2 to obtain a justifying assignment $\bar{a}$ for $P$. By Lemma 3.2.36 PROPs are closed under derivation, hence we can use the PIT from Theorem 3. The running time will be $(nd)^{O(\log n)}$. Now, apply Theorem 6.2.10. The reconstruction will take poly$(n,d)$ time. Consequently, the whole execution takes $(nd)^{O(\log n)}$ time.

Non-Adaptive PROF Reconstruction

The reconstruction algorithm described by Algorithm 6 in non-adaptive as it queries the PROP on the set $A^n_0(W)$ only regardless of the results of those queries. Consequently, the randomized version of the algorithm is non-adaptive as well. On the other hand, the algorithm given in Lemma 2.3.2 is in fact adaptive: its subsequent queries to $P$ depend on the result of the previous (see Algorithm 1). This implies that the deterministic algorithm in the previous section is not adaptive.

Can we give a non-adaptive deterministic algorithm? The answer is “yes”. The idea is as follows: instead of finding a (single) justifying assignment and use it for reconstruction, we will attempt to reconstruct $P$ using every assignment in a justifying set for PROPs, that is a set that contains a justifying assignment for every PROP (see Lemma 2.3.3). We collect all the outcomes together (into a set $S$). As we have seen earlier (Lemma 6.2.9 and the preceding discussion) the set $S$ may contain many PROFs. The question now is, “how do we find the right one?”. As in general this set may be large, this task could seem as hard as finding a needle in a haystack. Lemma 6.2.9 suggests the following: it is enough to find a PROF with the largest var. (i.e. a PROF that depends of the largest number of variables).

Formally, we will be looking for $\Phi^M \in S$ such that for every $\Phi' \in S$ it holds: $|\text{var}(\Phi^M)| \geq |\text{var}(\Phi')|$. The subsequent algorithm and summarize this discussion and proves Theorem 13.

Proof of Theorem 13. The algorithm will follow the steps specified in Algorithm 11.

Running time: By Lemma 2.3.3 $|J_{1,d}| = (nd)^{O(\log n)}$. Other steps can be
Algorithm 11 Non-Adaptive PROF Reconstruction

\textbf{Input:} PROP $P$ with individual degrees bounded by $d$ given as a black-box access

\textbf{Output:} PROF $\Phi$ such that $\Phi \equiv P$.

1: Compute $J_{1,d}$ \{Using Lemmas 2.3.3 and 3.3.1\}
2: $S \leftarrow \emptyset$
3: \textbf{for all} $\bar{\gamma} \in J_{1,d}$ \textbf{do}
4: \hspace{1em} $\Phi \leftarrow \text{Reconstruct} P$ \{Using Theorem 6.2.10 \}
5: \hspace{1em} $S \leftarrow S \cup \Phi$
6: \hspace{1em} Find $\Phi' \in S$ with the maximal $\text{var}(\Phi')$, denote it with $\Phi_M$
7: \textbf{return} $\Phi_M$

carried out in polynomial time. Hence, the total running time is $(nd)^{O(\log n)}$.

\textbf{Correctness of the Algorithm:} First of all notice that since Algorithm 6 is non-adaptive (for a given $\bar{\gamma}$) so does Algorithm 11 (finding $\Phi_M$ does require querying the black-box). By Lemmas 3.2.36 and 2.3.3 there exists $\bar{a} \in J_{1,d}$ a justifying assignment of $P$, hence $S$ must contain $\Phi^*$ - the result the reconstruction of $P$ using $\bar{a}$. By Theorem 6.2.10 $\Phi^* \equiv P$ (in particular, $S$ is not empty, so $\Phi_M$ is well-defined). We will argue now that $\Phi_M \equiv \Phi^* \equiv P$. By Lemma 6.2.9 $\text{var}(\Phi_M) \subseteq \text{var}(P)$, on the other hand from definition of $\Phi_M$ it holds that $|\text{var}(\Phi_M)| \geq |\text{var}(\Phi^*)| = |\text{var}(P)|$. This implies that $\text{var}(\Phi_M) = \text{var}(P)$ and again, by Lemma 6.2.9 $\Phi_M \equiv P$. 

6.3 Read-Once Testing

In this section we study the relation between the \textit{polynomial identity testing} problem (PIT) and the \textit{(preprocessed) read-once testing} problem (ROT) and prove Theorem 15. We present a generic scheme that can be used to strengthen efficient PIT algorithms to yield efficient read-once testing algorithms. Then we use the scheme to obtain ROT algorithms for models for which PIT algorithms are known.
6.3.1 Generic Scheme

The idea behind the scheme is: “Doveriai, no Proveriai” 5. First, we give a PROP reconstruction algorithm for PROPs computed by circuits (based on Theorem 6.2.10). That is, an algorithm that given a circuit $C$ computing a PROP, outputs a PROF $\Phi$ such that $\Phi \equiv C$. Then, to preform the ROT, we assume that the given circuit $C$ computes a PROP and run the reconstruction algorithm based on this assumption. If the algorithm encounters an error or is unable to run correctly, then we conclude that our assumption was wrong (i.e. $C$ does not compute a PROP) and thus we stop and report a failure. Things are more complicated in the case of success, that is, when the algorithm does output a PROF. The problem is that we do not have a guarantee regarding the correctness of our assumption (that the circuit computes a PROP) and hence the correctness of its output. Moreover, for any circuit $C$ that computes a PROP there exist many circuits (computing different polynomials) “aliasing” $C$. Meaning, that an execution of our reconstruction algorithm on each such circuit $C' \neq C$ will succeed and yield a PROF $\Phi$ such that $\Phi \equiv C \neq C'$. Consequently, to complete the read-once testing we need to verify the correctness of the output. For this purpose we need a verification procedure (Algorithm 13). Algorithm 12 gives the generic scheme for ROT 6. The algorithm works both in the black-box and in the non black-box settings, depending on the PIT algorithm for $C$ at hand.

Algorithm 12 Generic ROT Scheme

**Input:** A (black-box holding a) circuit $C$ that belongs to $\mathcal{C}$.

**Output:** PROF $\Phi$ such that $\Phi \equiv C$, if $C$ computes a PROP, “failure” otherwise.

1: Acquire a justifying assignment $\bar{a}$ using Lemma 2.3.2.
2: Reconstruct a PROF $\Phi$ from $C$ using Theorem 6.2.10.
3: Verify that $\Phi \equiv C$ using Algorithm 13 (given in Subsection 6.3.2).

In Section 6.3.3 we give the proof of Theorem 15, that basically analyzes the running time of Algorithm 12 given Lemma 2.3.2 and, Theorems 6.2.10

---

5“Trust, but Verify” - a translation of a Russian proverb which became a signature phrase of Ronald Reagan during the cold war.

6In fact, in many cases we shall give “tailored” algorithms that are somewhat more efficient than the generic algorithm.
and 6.3.2. Then in Section 6.4 we go over some circuit classes that have PIT algorithms and give the corresponding ROT algorithms (as mentioned earlier, in some cases we shall give more efficient ROT algorithms that do not use the verification step of Algorithm 12).

6.3.2 Read-Once Verification

Read-Once Verification is testing whether a given circuit \( C \), from a certain circuit class \( \mathcal{C} \), and a given PROF \( \Phi \) compute the same polynomial. Note though, that while the verification might have the nature of a polynomial identity testing, it is somewhat a harder problem since it requires determining the equivalence of polynomials computed by circuits from two different circuit classes. We shall work under the assumption that \( \mathcal{C} \) has a PIT algorithm and the PROF \( \Phi \) is the output of the PROF reconstruction algorithm and thus is given to us explicitly (e.g. as a graph of computations). The circuit \( C \), on the other hand, may be given to us as a black-box, depending on the PIT algorithm for \( \mathcal{C} \). Clearly, if \( C - \Phi \in \mathcal{C} \) then the verification procedure is trivial (as \( \mathcal{C} \) has a PIT algorithm). The general case however is more complicated. We shall present a verification procedure that enables us to take care of the case where \( C - \Phi \not\in \mathcal{C} \). The idea behind the algorithm is to recursively ensure that every gate \( v \) of \( \Phi \) computes the same polynomial as a “corresponding” restriction of \( C \). To give a rough sketch, we first find a justifying assignment for \( \Phi \), \( \bar{a} \). Then we consider \( v \), the root of \( \Phi \). It has two sub formulas. W.l.o.g. assume that \( \Phi = \alpha \cdot (\Phi_{v_1} \text{ op } \Phi_{v_2}) + \beta \), where \( \Phi_{v_i} \) is the PROF computed at \( v_i \) and \( \text{op} \) is either + or ×. Assume that the variables of \( v_i \) are \( S_i \) (\( S_1 \) and \( S_2 \) are disjoint). Consider the circuit \( C_1 \) that equals to \( C \) after we substitute the corresponding values from \( \bar{a} \) to the variables in \( S_2 \). Similarly we define \( C_2 \), and the PROFs \( \Phi_1 \) and \( \Phi_2 \). We now recursively verify that \( C_i \equiv \Phi_i \). The only thing left now is to verify that indeed \( C \equiv \alpha \cdot (C_1 \text{ op } C_2) + \beta \). This basically reduces the verification problem to the problem of verifying that \( C \equiv C_1 \text{ op } C_2 \) where \( C_1 \) and \( C_2 \) compute variable disjoint PROPs and \( \text{op} \) is either + or ×. Note, that this is a PIT problem for a circuit class that is closely related to \( \mathcal{C} \), although slightly different (e.g. if \( \mathcal{C} \) is the class of \( \Sigma \Pi \Sigma(k) \) circuits, defined in Section

\footnote{In fact, in the algorithm it will be more convenient to have \( \Phi_i = \Phi_{v_i} \) and so we will slightly change the definition of \( C_i \).}
6.4.2, then we need a PIT algorithm for \( \Sigma \Pi \Sigma(\mathcal{O}(k^2)) \) circuits). Therefore, we shall assume that a slightly larger circuit class has an efficient PIT algorithm (e.g. a class containing \( C - C_1 \text{ op } C_2 \) for variable disjoint \( C_1 \) and \( C_2 \)). For this we make the following definition of a “verifying class”. The definition uses the notations given in Definition 2.1.15. Note, that for most circuit classes with efficient PIT algorithms, there exists an efficient PIT algorithm for a corresponding verifying class.

**Definition 6.3.1.** A circuit class \( \mathcal{C}_V \) is a (read-once) Verifying Class for a circuit class \( \mathcal{C} \) if \( C_1 + C_2 + C_3 \times C_4 \in \mathcal{C}_V \) when \( C_1, C_2, C_3, C_4 \in \mathcal{L}(\mathcal{C}) \) and \( C_2, C_3, C_4 \) are defined on disjoint sets of variables.

**Theorem 6.3.2.** Let \( C \) and \( \Phi \) be a circuit from a circuit class \( \mathcal{C} \), and a PROF correspondingly, such that \( \text{var}(C) = \text{var}(\Phi) \). In addition, let \( \mathcal{C}_V \) be a verifying class of \( \mathcal{C} \) and \( \bar{a} \) a justifying assignment of \( \Phi \) and \( C \). Then given \( C, \Phi \) and \( \bar{a} \) Algorithm 13 runs in time \( \mathcal{O}(nd + n \cdot t) \), when \( t \) is the cost of a single PIT algorithm for \( \mathcal{C}_V \), and outputs “true” if and only if \( C \equiv \Phi \).

**Proof.** **Running time:** We start by analyzing the running time. As described above, the algorithm performs a traversal over the tree of \( \Phi \). The PIT algorithm of \( \mathcal{C}_V \) is invoked once for every internal gate (multiplication, addition). On each input gate a single-variable query on \( d + 1 \) points is performed. Hence the total running time is \( \mathcal{O}(nd + n \cdot t) \) when \( t \) is the cost of a single PIT algorithm for \( \mathcal{C}_V \). We now prove the correctness of the algorithm.

**Execution:** We show that all the PIT calls that we make are “well defined”. From the first glance, we might expect “hazards” executing the lines which require an invocation of a PIT algorithm. That is, lines 9 and 14. We make the following observations. In each stage of the algorithm it holds:

1. \( C, C_L, C_R \in \mathcal{L}(\mathcal{C}) \) (when \( C_L, C_R \) defined). This follows (recursively) from the definitions of \( C_L, C_R \). \( C_L, C_R \) are defined as an application of a linear function on a restriction of \( C \).

2. \( \text{var}(C) = \text{var}(\Phi) \). By induction. For the base case we are guaranteed that \( \text{var}(C) = \text{var}(\Phi) \). For the step, consider the definition of the set \( L \) and \( R \). As \( \bar{a} \) is a justifying assignment of \( C \) we have that

\[
\text{var}(C_L) = \text{var}(C) \setminus R = \text{var}(\Phi) \setminus \text{var}(\Phi_{v, \text{Right}}) = \text{var}(\Phi_{v, \text{Left}})
\]
Algorithm 13 Verify \((C, \Phi)\)

Input: Circuit \(C\) from a circuit class \(C\),
Access to a PIT algorithm for \(C_V\),
Justifying assignment \(\bar{a}\) for \(\Phi\) and \(C\)

Output: “true” if \(C \equiv \Phi\) and “false” otherwise.

1: if \(v.\text{Type} = \text{IN}\) then \{(\(\Phi\) is a univariate polynomial\)}
2: \hspace{1em} Check that \(C \equiv \Phi\) \{As two univariate polynomials of degree at most \(d\)\}
3: \hspace{1em} \{Internal Gate\}
4: \hspace{1em} \(L \leftarrow \text{var}(\Phi_{v.\text{Left}})\)
5: \hspace{1em} \(R \leftarrow \text{var}(\Phi_{v.\text{Right}})\)
6: \hspace{1em} \{Multiplication Gate\}
7: \hspace{1em} \(C_L \leftarrow (C|_{x_R=\bar{a}_R - v.\beta}) / (v.\alpha \cdot \Phi_{v.\text{Left}}(\bar{a}))\)
8: \hspace{1em} \(C_R \leftarrow (C|_{x_L=\bar{a}_L - v.\beta}) / (v.\alpha \cdot \Phi_{v.\text{Right}}(\bar{a}))\)
9: \hspace{1em} \{Recursively\}
10: Check that \(C \cdot C(\bar{a}_L, \bar{a}_R) \equiv C_L \cdot C_R\) \{Using the PIT algorithm for \(C_V\)\}
11: \hspace{1em} \{Addition Gate\}
12: if \(v.\text{Type} = \times\) then
13: \hspace{1em} \(C_L \leftarrow (C|_{x_R=\bar{a}_R - v.\beta}) / (v.\alpha - \Phi_{v.\text{Left}}(\bar{a}))\)
14: \hspace{1em} \(C_R \leftarrow (C|_{x_L=\bar{a}_L - v.\beta}) / (v.\alpha - \Phi_{v.\text{Right}}(\bar{a}))\)
15: \hspace{1em} \{Recursively\}
16: \hspace{1em} \{Everything is OK\}
17: \hspace{1em} \{Recursively\}
18: \hspace{1em} \{Everyting is OK\}
19: \hspace{1em} \{Everything is OK\}
20: return true
and similarly var(C_R) = var(Φ_v,Right). It is only left to notice that the above result corresponds with the recursive invocation of the algorithm.

3. C_L and C_R are variable-disjoint Indeed,

\[ \text{var}(C_L) \cap \text{var}(C_R) = \text{var}(Φ_v,Left) \cap \text{var}(Φ_v,Right) = \emptyset \]

which follows from the definition of PROF.

From Observations 1 and 3 it follows that \( C - v.α \cdot (C_L \times C_R) - v.β \in C_V \) and \( C - v.α \cdot (C_L + C_R) - v.β \in C_V \). Hence, we can conclude that the identity tests in lines 9, 14 can be carried out using the PIT algorithm of \( C_V \). Consider line 2. In this case \( Φ \) is a univariate polynomial of degree at most \( d \). As \( \text{var}(C) = \text{var}(Φ) \) (Observation 2) we obtain that so is \( C - Φ \). Hence the test can be carried out by querying \( C \) and \( Φ \) on (at most) \( d + 1 \) points.

**Correctness:** We show that given a PIT algorithm for \( C_V \) the algorithm returns the correct answer. The correctness of the algorithm’s output can be proven by a simple induction. That is, the algorithm outputs “true” iff \( C \equiv Φ \). The base case (i.e. \( Φ \) is a univariate polynomial of degree at most \( d \)) is trivial. The correctness of the step follows from Lemmas 6.3.3 and 6.3.4 (and the correctness of the PIT algorithm of \( C_V \)). This completes the proof.

**Lemma 6.3.3.** Let \( C(\bar{x},\bar{y}) \) and \( Φ(\bar{x},\bar{y}) = α \cdot Φ_L(\bar{x}) \times Φ_R(\bar{y}) + β \) be two polynomials and let \( (\bar{x}_0,\bar{y}_0) \) be a justifying assignment of \( Φ \). Then \( C(\bar{x},\bar{y}) \equiv Φ(\bar{x},\bar{y}) \) if and only if the following conditions hold:

1. \( Φ_L(\bar{x}) = \frac{C(\bar{x}_0,\bar{y}_0)-β}{α \cdot Φ_R(\bar{y}_0)} \)
2. \( Φ_R(\bar{y}) = \frac{C(\bar{x}_0,\bar{y}_0)-β}{α \cdot Φ_L(\bar{x}_0)} \)
3. \( C(\bar{x},\bar{y}) = α \cdot \frac{C(\bar{x}_0,\bar{y}_0)-β}{α \cdot Φ_R(\bar{y}_0)} \times \frac{C(\bar{x}_0,\bar{y}_0)-β}{α \cdot Φ_L(\bar{x}_0)} + β \)

Notice that the conditions are well defined since \( (\bar{x}_0,\bar{y}_0) \) is a justifying assignment of \( Φ \) and hence \( Φ_L(\bar{x}_0), Φ_R(\bar{y}_0) \neq 0 \).
The additive version:

**Lemma 6.3.4.** Let $C(\bar{x}, \bar{y})$ and $\Phi(\bar{x}, \bar{y}) = \alpha \cdot (\Phi_\ell(\bar{x}) + \Phi_r(\bar{y})) + \beta$ be two polynomials and let $(\bar{x}_0, \bar{y}_0)$ be a justifying assignment of $\Phi$. Then $C(\bar{x}, \bar{y}) \equiv \Phi(\bar{x}, \bar{y})$ if and only if the following conditions hold:

1. $\Phi_\ell(\bar{x}) = \frac{C(\bar{x}, \bar{y}_0) - \beta}{\alpha} - \Phi_r(\bar{y}_0)$
2. $\Phi_r(\bar{y}) = \frac{C(\bar{x}_0, \bar{y}) - \beta}{\alpha} - \Phi_\ell(\bar{x}_0)$
3. $C(\bar{x}, \bar{y}) = \alpha \cdot (\frac{C(\bar{x}, \bar{y}_0) - \beta}{\alpha} - \Phi_r(\bar{y}_0) + \frac{C(\bar{x}_0, \bar{y}) - \beta}{\alpha} - \Phi_\ell(\bar{x}_0)) + \beta$

**Proof.** Both proofs are performed by simple substitutions. □

### 6.3.3 Proof of Theorem 15

**Proof of Theorem 15.** We will show that given $C$ and $C_V$ satisfying the required conditions we can successfully preform each phase of the “Generic ROT Scheme” described in Algorithm 12.

1. **Acquiring a Justifying Assignment**
   As $\partial C \subseteq C_V$ we can acquire a justifying assignment $\bar{a}$ using Lemma 2.3.2.
   The running time is $\mathcal{O}(n^3d \cdot T(n, d, s))$.

2. **Reconstructing PROF $\Phi$ from $C$**
   Given the justifying assignment $\bar{a}$ we can reconstruct $\Phi$ from $C$ using Theorem 6.2.10.
   The running time is $\text{poly}(n, d, s)$.

3. **Verifying that $\Phi \equiv C$**
   Notice that $C_V$ satisfies the conditions of Definition 6.3.1 and hence can serve a verifying class of $C$. This allows to invoke the verification procedure described in Algorithm 13.
   The running time is $\mathcal{O}(n^4d \cdot T(n, d, s))$.

The total running time is $\text{poly}(n, s, d, T(n, d, s))$. □
In the next section we use Theorem 15 and the Generic Scheme suggested in Algorithm 12 in order to get ROT algorithms for several restricted models for which PIT is known. Given our result it is necessary that a PIT algorithm is known for the class that we wish to get ROT algorithm for. However, we note that basically any “reasonable” PIT algorithm yields a ROT algorithm. For example, an immediate conclusion of Theorem 15 is that an efficient PIT algorithm for multilinear circuits (for both black-box and none black-box settings) implies an efficient read-once testing algorithm for multilinear circuits.

6.4 ROT for Specific Models

In this section we prove Theorems 16, 17 and 18 by applying the ROT scheme of Section 6.3 (Algorithm 12). We start with the case of sparse polynomials (Theorem 17).

6.4.1 Sparse Polynomials

An $m$-sparse polynomial is polynomial with at most $m$ (non-zero) monomials. Equivalently, it is a polynomial computed by a depth-2 circuit of size $m$. Sparse polynomials were deeply studied (see e.g. [11, 43, 45]) and, in fact, several polynomial time algorithms for both reconstruction and PIT were given, over sufficiently large fields (or extension fields). We now show how to ROT sparse polynomials. Note that in [14] it is mentioned that a polynomial time interpolation algorithm is known in the case that the given ROF is a sparse polynomial (see last sentence of paragraph 4 on page 707 of [14] that refers to [44]). We show another proof here which gives a stronger result (thus proving Theorem 17). We denote by $\Sigma \Pi(m, d)$ the class of $m$-sparse polynomials of degree $d$. In order to apply Algorithm 12 on $\Sigma \Pi(m, d)$ we shall need a PIT algorithm for $\Sigma \Pi(\text{poly}(m), \text{poly}(d))$. Several results are known. For simplicity we pick one of them given in [43].

**Lemma 6.4.1** ([43]). There is a deterministic black-box PIT algorithm for $m$-sparse $n$-variate polynomials of degree at most $d$ that runs in time $\text{poly}(m, n, d)$.
Note that $\Sigma\Pi(m^2 + 4m + 3, 2d)$ is a verifying class for $\Sigma\Pi(m, d)$ (see Definition 6.3.1). Hence, we can use the above lemma in conjunction with Algorithm 12 to prove Theorem 17. We now give an alternative proof of the theorem that follows the same lines as the scheme of Algorithm 12, however uses a different verification procedure.

**Proof of Theorem 17.** Let $P \in \Sigma\Pi(m, d)$ be a sparse polynomial given as a black-box. We follow the scheme outlined in Algorithm 12 noting that $\partial\Sigma\Pi(m, d) \subseteq \Sigma\Pi(m, d)$. We first run the algorithms suggested by Lemma 2.3.2 and Theorem 6.2.10 to obtain a candidate PROF $\Phi$. Of course, should any of the algorithms fail, we output “not PROF”. We now wish to verify that $\Phi \equiv P$. Originally, we could use Algorithm 13, but instead we use a different scheme. We first check that $\deg(\Phi) \leq d$ and then run Algorithm 15 to count the number of monomials in the PROP computed by $\Phi$ (denoted by $M$). Now, if $M > m$ or $\deg(\Phi) > d$ then, obviously, $\Phi \neq P$. Otherwise, we set $P' \triangleq \Phi - P$. Note that the resulting polynomial $P'$ has at most $2m$ monomials and hence $P' \in \Sigma\Pi(2m, d)$. Consequently, we can apply Lemma 6.4.1 to determine whether $P' \equiv 0$. Note that the PIT algorithm for $\Sigma\Pi(2m, d)$ is polynomial in $n, m, d$ (and is slightly more efficient than Algorithm 13 equipped with a PIT algorithm for $\Sigma\Pi(m^2 + 4m + 3, 2d)$).

6.4.2 Depth-3 Circuits

A depth-3 $\Sigma\Pi\Sigma(k)$ circuit $C$ of degree $d$ computes a polynomial of the form

$$C(\bar{x}) = \sum_{i=1}^{k} F_i(\bar{x}) = \sum_{i=1}^{k} d_i \prod_{j=1} L_{ij}(\bar{x})$$

where the $L_{ij}(\bar{x})$-s are linear functions: $L_{ij}(\bar{x}) = \sum_{t=1}^{n} a_{ij}^t x_t + a_{ij}^0$ with $a_{ij}^t \in \mathbb{F}$, and $d_i \leq d$. We denote by $\Sigma\Pi\Sigma(k, d)$ a $\Sigma\Pi\Sigma(k)$ in which each $F_i$ has degree at most $d$. Those circuits were a subject for a long line of study [17, 41, 37, 40, 58, 9, 59] yielding several efficient PIT algorithm. Yet, only recently a polynomial-time black-box PIT algorithm for this circuit class was given. For more info see Section 3.5.

**Lemma 6.4.2** ([60]). There is a deterministic black-box PIT algorithm for $n$-variate polynomials computed by $\Sigma\Pi\Sigma(k, d)$ circuits that runs in time

131
We now give the proof of Theorem 18.

Proof of Theorem 18. Observe that the class $\Sigma \Pi \Sigma(k^2 + 4k + 3, 2d)$ is a verifying class for $\Sigma \Pi \Sigma(k, d)$. Hence, we can invoke Algorithm 12 in conjunction with Lemma 6.4.2. Theorem 15 guarantees the correction. The running time is $\text{poly}(n) \cdot d^{O(k^2)}$.

6.4.3 Read-Once Testing for Sum of $k$ PROFs

PIT algorithms for sums of preprocessed read-once formulas were given in Theorems 1 and 2. We now show a generalization of those PIT algorithms to ROT algorithms, thus proving Theorem 16.

Proof of Theorem 16. As previously, let $F = F_1 + F_2 + \cdots + F_k$. We first apply Lemma 2.3.2 to obtain a justifying assignment for $F$. Lemma 3.2.36 implies that this circuit class is closed under partial derivatives, thus we can use the theorems themselves in the procedure.

Next, construct a candidate ROF $\Phi$ by applying Theorem 6.2.10. Given $\Phi$ we simply need to verify that $F = F_1 + F_2 + \cdots + F_k$ or, in other words, that $F_1 + F_2 + \cdots + F_k - \Phi \equiv 0$. This is exactly a PIT of sum of $k + 1$ PROFs, hence we can the theorems again to obtain the result.

6.5 PROF Graph-Related Algorithms

In this section we give two basic algorithms for PROFs. Some of the algorithms are small variants of already known algorithms. To slightly simplify the algorithms we assume w.l.o.g that all the PROFs are non-degenerate. We can make this assumption due to the fact that there is a trivial $O(n)$ algorithm that can convert any ROF $\Phi$ to a non-degenerate ROF $\Phi'$ such that $\Phi \equiv \Phi'$. In fact, this algorithm can also be used as a (non-black box) PIT algorithm for ROFs. We conclude this discussion with two trivial observations. The first is that a non-degenerate ROF $\Phi$ computes the zero polynomial if and only if $\Phi$ is the “zero ROF” i.e $\Phi = C_{v_0}$ (the graph of computation of $\Phi$ is no other than $C_{v_0}$). Secondly, in a non-degenerate ROF,
in all the labels \((\alpha, \beta)\) of the gates, we have that \(\alpha \neq 0\). We now give some notations that will be used in our algorithms.

**Notation 6.5.1.** Let \(G\) be the graph of computation of a PROF. We think of \(G\) as a planar graph (and in particular the notion of "left child" and "right child" of a node are well defined). Let \(v\) be a gate in the formula.

- \(\Phi_v\) - denotes the formula rooted at \(v\).
- \(v.\text{var}\) - denotes \(\text{var}(\Phi_v)\). Note, that it can be computed recursively.
- \(v.\alpha\) - denotes the value of \(\alpha\) labeling \(v\).
- \(v.\beta\) - denotes the value of \(\beta\) labeling \(v\).
- \(v.\text{Left}\) - denotes the left child of \(v\).
- \(v.\text{Right}\) - denotes the right child of \(v\).
- \(v.\text{Type}\) - denotes the type of the arithmetic operation labeling \(v\). (IN - Input, CONST - Constant Function (non-degenerate form), \(\times\) - Multiplication, \(+\) - Addition)

### 6.5.1 Factoring a ROF

We now give an algorithm for finding all the irreducible factors of a given ROF. The idea of the algorithm comes from the proof of Lemma 3.2.15. In the proof of the lemma we showed that a ROF \(\Phi\) has non trivial factors if and only if its top gate is a multiplication gate and the additive constant, \(\beta\), of the top gate is 0. Note, that in this case \(\Phi\) equals the product of the polynomials computed by its children. From this it is clear that by looking at the top gate and recursing on the left and right children we can find all the irreducible factors. Algorithm 14 returns a list \(S = \{h_i\}\) of the irreducible factors of \(\Phi_v\) (the polynomial computed by the node \(v\)) and a constant \(\alpha\) such that \(\Phi_v = \alpha \cdot \prod_i h_i\).

The following lemma gives the analysis of the algorithm. We omit the proof as it is similar to previous proofs.

**Lemma 6.5.2.** Given a node \(v\) in a graph of computations of a non-constant ROF \(\Phi\), Algorithm 14 returns a pair \((S, \alpha)\) where \(S\) is a list of (ROFs
Algorithm 14 FactorROF(Φ)

Input: Non-constant ROF Φ.

Output: (S, α) where S is a list of the irreducible factors of Φ, α a constant in the field.

1: if v.Type = IN then \{Univariate polynomial\}
2: return (Φ, 1)
3: if v.Type = + then
4: return (Φ, 1)
5: if \(v.\beta \neq 0\) then
6: return (Φ, 1)
7: \(\{\text{Now } v.\text{Type} = \times \text{ and } \beta = 0, \text{ thus } \Phi = \alpha \cdot \Phi_{v.\text{Right}} \times \Phi_{v.\text{Left}}\}\)
8: \((S_L, \alpha_L) \leftarrow \text{FactorROF}(\Phi_{v.\text{Left}})\)
9: \((S_R, \alpha_R) \leftarrow \text{FactorROF}(\Phi_{v.\text{Right}})\)
10: return \((\text{union}(S_L, S_R), \alpha_L \cdot v.\alpha \cdot \alpha_R)\)

computing) the irreducible factors of the polynomial Φ_v and α is a constant satisfying Φ_v = α \cdot \prod_{h \in S} h. The running time of the algorithm is \(O(n)\).

6.5.2 Counting the Number of Monomials in a PROP

The number of monomials in a polynomial \(P\) is the number of monomials with non-zero coefficients. We now give an efficient algorithm for counting the number of monomials computed by a given PROF. The main idea is that once a non-constant monomial appears, it can not be canceled later. Consequently, we only have to sum the number of non-constant monomials and keep track of the behavior of the constant term.

In order to determine whether a certain value is zero or non-zero we use an auxiliary function \(\Phi_{nz}(x)\) that is defined as

\[
\Phi_{nz}(x) = \begin{cases} 
0 & x = 0 \\
1 & \text{otherwise}
\end{cases}
\]

The algorithm simply recurses on the left child and right child of the root and combines the results according to the different values of the constant terms.

Lemma 6.5.3. Given a node \(v\) in the graph of computation of a PROF \(\Phi\),
Algorithm 15 CountMonomials(v)

Input: The root $v$ of a PROF.

Output: $(M, C)$ where

$M$ is the number of the non-constant monomials of $\Phi_v$,
$C$ is the constant term of $\Phi_v$.

1: if $v$.Type = IN OR $v$.Type = CONST then  {$\Phi_v$ is a univariate or constant polynomial}
2:    return $(\deg(\Phi_v) , v.\beta)$
   {Internal gate}
3: $(M_L, C_L) \leftarrow$ CountMonomials($v$.Left)
4: $(M_R, C_R) \leftarrow$ CountMonomials($v$.Right)
5: if $v$.Type = $\times$ then
6:    $C \leftarrow v.\alpha \cdot C_L \cdot C_R + v.\beta$
7:    $M \leftarrow (M_L + \Phi_{nz}(C_L)) \cdot (M_R + \Phi_{nz}(C_R)) - \Phi_{nz}(C_L \cdot C_R)$
8:    return $(M, C)$
9: if $v$.Type = $+$ then
10:   $C \leftarrow v.\alpha \cdot (C_L + C_R) + v.\beta$
11:   $M \leftarrow M_L + M_R$
12: return $(M, C)$
Algorithm 15 returns a pair \((M,C)\) when \(M\) is the (exact) number of the non-constant monomials of \(\Phi_v\) and \(C\) is the constant term of \(\Phi_v\). The running time of the algorithm is \(O(nd)\).

**Proof.** We start by proving the correctness of the algorithm. The proof is by induction on the structure of the formula. We first analyze what happens at the leaves and then move up. During the analysis we shall use the simple observation that the number of non-constant monomials is not affected by \((\alpha,\beta)\) labeling each gate. We denote with \((M_L,C_L)\) and \((M_R,C_R)\) the results returned from the left child and the right child, respectively. We consider the different possibilities for the operation labeling \(v\) (i.e. \(v\).Type).

- \(v\).Type = CONST or \(v\).Type = IN: \(\Phi_v\) computes a constant or a univariate polynomial. Clearly, \(M = \deg(\Phi_v)\) (when we take the degree of the zero polynomial as “0”).

When \(v\) is not a leaf there is recursive call of the function for to the left and right children. Denote with \(p_L\) and \(p_R\) the polynomials computed by the left child and right child, respectively. We again analyze the different options for \(v\).Type.

- \(v\).Type = \(\times\): \(\Phi_v\) computes a product of two functions, \(\Phi_v = v.\alpha \cdot p_L \times p_R + v.\beta\). The total number of monomials of \(p_L \times p_R\) is \((M_L + fnz(C_L)) \cdot (M_R + fnz(C_R))\). Therefore, the number of non-constant monomials of \(p_L \times p_R\) is \((M_L + fnz(C_L)) \cdot (M_R + fnz(C_R)) - fnz(C_L \cdot C_R)\). From our observation this is exactly the number of non-constant monomials of \(\Phi_v\). The claim regarding \(C\) is trivial.

- \(v\).Type = \(+\): \(\Phi_v\) computes a sum of two functions. \(\Phi_v = v.\alpha \times (p_L + p_R) + v.\beta\) then (similarly): the number of non-constant monomials of \(p_L \times p_R\) (and hence of \(\Phi_v\)) is \(M_L + M_R\). As previously, the claim regarding \(C\) is trivial.

It is clear from the above analysis the the algorithm returns a correct answer. The claim regarding the running time follows easily from the fact that the algorithm is a simple graph traversal that requires \(O(n)\) time. As the individual degrees of the polynomial bounded by \(d\) computing \(\deg(\Phi_v)\) for the leaves requires \(O(d)\) time. In addition, note that at each step we multiply two \(n \log d\) bits numbers (since \(0 \leq M \leq (d+1)^n\)). Thus the total running time is \(O(nd)\).

\(\square\)
Chapter 7

Conclusions and Open Questions

The general PIT problem is out of reach. Yet, in this work we saw several PIT algorithms for some restricted case of multilinear and bounded-depth circuits: sums of read-once formulas, depth-3 $\Sigma \Pi \Sigma (k)$ circuits and multilinear depth-4 $\Sigma \Pi \Sigma \Pi (k)$ circuits. In the light of existing strong connection between PIT and circuit lower bounds a further study of those classes of circuits is believed to be a promising direction since lower bounds for those classes are already known [22, 24, 50, 53, 54, 52]. Hopefully, applying and extending our techniques would lead us to efficient PIT algorithms for larger classes of circuits and would bring us closer to the resolution of the general question.

In addition, we exhibited connections between PIT and other problems that have efficient randomized algorithms but lack of deterministic ones (partial polynomial factorization, read-once testing). Following those results, it would be nice to examine this connection further. In particular, to obtain an equivalence PIT and the general factorization problem, or come up with an efficient “sparsity-testing” - as we already have efficient PIT algorithms for sparse polynomials.

In the recent years, several efficient reconstruction algorithms for restricted circuit classes, have seen light. For example: sparse polynomials [43], depth-3 $\Sigma \Pi \Sigma (k)$ circuits [63, 38] and others. Interestingly, many of them were, in fact, follow-ups of presented earlier black-box PIT algorithms.
In this work, we followed the same road with read-once formulas. Moreover, we have actually shown a way of generalizing a black-box PIT algorithm for read-once formulas into a reconstruction algorithm. Additionally, a recent result of Gupta et al. [25], that gives a reconstruction algorithm for multilinear depth-4 $\Sigma\Pi\Sigma\Pi(2)$ circuits, can be also seen as a follow-up result to our black-box PIT algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits.

However, no sub-exponential reconstruction algorithm (even non-deterministic) is known for the case of $k > 1$, while Theorem 1 (and actually, Theorem 3.4.4) suggests that a sum of read-once formulas is uniquely determined by its values on a “small” (quasi-polynomial) set of points. In view of the above, coming up with an efficient reconstruction algorithm for sum of $k > 1$ read-once formulas appears to be an interesting problem. The same is goes for the result of [25] for $k > 2$.

More generally, we should attempt to see a more global picture in the relation between PIT and reconstruction. In particular, whether there is a generic way to reconstruct a circuit class that admits an efficient PIT algorithm.
Bibliography


התוצאות שלהן:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

(Read-Once Polynomials)

- סוכנים של פולינומי הפולטים לישוב, קריאה חדית
- מונע למשוך 3
- מונעים פוליטייזרים יפים בצפוף 4

כמ众所 עליל, בודק זיהוי פולינומי היה השיטה העולה על אלגוריתם הסתברותי יעיל, אך לא אלגוריתם סותרי יעיל. לפרס, אנו שואלים את השאלת הנבואה: "האם ניתן לבדוק זיהוי פולינומי כמו הסתברותי בבלוקה הסתברותי בולדר?".

茲樂[HS80, KI04, DSY09] מ {*} ממספר בנתאות בתוכשו, זה אכז

בהתיה זו, אנוعودة להלך השאלת. אנו מציא קשירים בין בדיקת זיהוי פולינומי לבר

ביין פלור פולינומי. בפרט, אנו Marxism ci אלגוריתם עלי לפורק פולינומי זרו

אלגוריתם עלי בבריקת הזר פולינומי. מנגד, אלגוריתם עלי בבריקת הזר פולינומי

גזור אלגוריתם עלי לברק מתחש של פורק פולינומי. יזור הכרך, עבש פולינומי

פוליטייזרים יעיבר שקולות לחלוץ (עד דיכי פקוט פולינומי).

בנוסף, אנו מציא ק詈יאית של הדמה בין בדיקת זיהוי פולינומי לבר, בעיות השaukee עבש

מתלחת הפולינומי המגנינים לחשוש ע"", קריאה חדית (זרה המדרגה בגוף ההיתידה). התצבאה

מכך, אנו מקבלים אלגוריתם שלב פולינימי껄, הרארח יוס, עבש מחלקה זו.

לבסוף, אנו מציא בבריקת אלגוריתם עלי בבריקת הזר פולינומי עבש מתלחת מעגלים

מסיועות נינן, לבריקת עבשה ההאמ של מתלחת מחלקה משמש פולינום, הנסים ליווש ע"", קריאה

חדית.
A possible explanation of the gap is the connection between finding an efficient algorithm for a problem and finding lower bounds for arithmetic circuits.


\[ K_{04}, AGR_{05} \]

In particular, an efficient algorithm for checking isomorphism implies lower bounds for arithmetic circuits and/or Boolean functions.


\[ DS_{06}, K_{07}, K_{08}, K_{09}, S_{09}, A_{10}, S_{10}, S_{11} \]

In particular, lower bounds for arithmetic circuits and/or Boolean functions exist.


\[ S_{08}, K_{MSV_{10}}, A_{10} \]

One of the questions we can ask is: "Why did the research stop at depth 4?" and "What is the progress of the research?"

The answer to these questions was given in the thesis by Agarwal and Vinay [AV_{08}]

In the case of arithmetic circuits of depth 4, the problem is almost as difficult as the general problem of polynomial factorization.


\[ K_{AL_{89}}, K_{T_{90}}, G_{99}, K_{AL_{03}}, G_{AT_{06}} \]

However, the question of the existence of a deterministic efficient algorithm for the problem remains unanswered.


\[ \text{minimization and membership for arithmetic circuits} \]

- The problem of checking membership in arithmetic circuits:
  - Given an arithmetic circuit, decide if it computes the same polynomial as another arithmetic circuit.
  - In general, this problem can also be formulated as a decision problem: given an arithmetic circuit computing a polynomial, determine if there is another arithmetic circuit of a certain size, modulo, and so on...

All these problems are important in the field of computational algebra.

It is not difficult to see that an efficient algorithm for each of these problems implies an efficient algorithm for checking isomorphism.


\[ \text{NP} \to \text{NP} = \Sigma_{2}^{P} \]

It can be shown that the problems above are at least as hard as the general problem of the

Finally, the problem of checking membership in arithmetic circuits

\[ \sum^{2} = \text{NP}^{\text{NP}} \]

The result is obtained from the polynomial hierarchy.
In this thesis, we study one of the core problems in the theory of computer science—"the equality problem" (also known as Polynomial Identity Testing). The uniqueness of this problem lies in the fact that it is one of the few problems that admit randomized efficient algorithms (coRP), but on the other hand, it is found that a deterministic efficient algorithm for it has been developed.

In this work, we present several deterministic algorithms for several special cases of the problem, and we also examine connections between the problem and other similar problems. In particular, we show that the problem of checking the equality of polynomials is polynomially equivalent to the problem of factoring polynomials, and we also present a general framework for transforming efficient algorithms for the equality problem into algorithms for other problems.

To solve the equality problem, we use the model of arithmetic circuits: Arithmetic circuits are the standard model for computing polynomials with multiple variables over a field. In an arithmetic circuit with \( n \) variables over the field \( F \), we have a directed graph without cycles, with inputs labeled \( x_1, \ldots, x_n \), and outputs. The internal nodes represent arithmetic operations such as addition or multiplication. The circuit computes the polynomial of the circuit evaluated at the inputs.

The equality problem in this context is to check whether a given circuit computes the zero polynomial. A large number of connections to algorithmic and theoretical problems are highlighted by this important problem. For example, algorithms for finding perfect matching [LOV79, MVV87], primality testing [AKS04, PCP-AS98, ALM98], and others are mentioned as connections to this problem.

For the equality problem, we require only that we check the value of the polynomial at an arbitrary point. On the other hand, finding an algorithm efficient for this problem is an open question for over thirty years.

Technion - Computer Science Department - Ph.D. Thesis  PHD-2012-06 - 2012
ה המחקר נועש בהנחיית פרופסור אמר שפילקה בפקולטה למדעי המחשב.

אני רוצה לצלוח את זדנותו ונדיבותו של פרופסור אמיר שפילקה על הנחייתו המ쨔פה ותמיכתו המהירה בכל הדר氖.

אני מודה לטכניון, לארגון המדע הישראלי מענק מספר 439/06,(ו ליוכנות European Community’s Seventh Framework Programme מענק מפסר 257575 (FP7/2007-2013) על התמיכהتحدית והרבה לא_accepted כל הדרים.
בעיית בדיקת זהות פולינומיים окשראית
למספר בעיון אלגברהית

ת changer על מחקר

לשם مليים חלקי של הדרישות לקבלת התואר דוקטור לפילוסופיה

אילה וולקוביץ

הוגש לסנט
הטכניון - מכון הטכנולוגי לישראל
כסלו, תשמ"ב, חיפה, דצמבר 2011
בעיית בדיקת זהות פולינומיים והשיות
למספר בעיות אלגבריות

איליה ולקוביץ"