THEORY OF COMPUTING, Volume 10 (18), 2014, pp. 465–514
www.theoryofcomputing.org

On Reconstruction and Testing of Read-Once Formulas

Amir Shpilka* Ilya Volkovich*

Received June 18, 2012; Revised July 18, 2014; Published December 26, 2014

Abstract: 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 each 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 $T_i(x_i)$. We obtain a deterministic non-adaptive reconstruction algorithm for PROFs, that is, an algorithm that, given black-box access to a PROF, constructs a PROF computing the same polynomial. The running time of the algorithm is $(nd)^{\mathcal{O}(\log n)}$ for PROFs of individual degrees at most d. To the best of our knowledge our results give the first subexponential-time deterministic reconstruction algorithms for ROFs.

Another question that we study is the following generalization of the polynomial identity testing (PIT) problem. Given an arithmetic circuit computing a polynomial $P(\bar{x})$, decide whether there is a PROF computing $P(\bar{x})$, and find one if one exists. 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 ROT problem.

An extended abstract appeared in the Proceedings of the 40th Annual Symposium on Theory of Computing (STOC 2008), pages 507-516 [40].

*Research supported by the Israel Science Foundation (grant number 339/10) and the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575.

ACM Classification: F.2.2

AMS Classification: 68Q25

Key words and phrases: read-once formulas, identity testing, reconstruction, arithmetic circuits, bounded-depth circuits

© 2014 Amir Shpilka and Ilya Volkovich
Licensed under a Creative Commons Attribution License (CC-BY) DOI: 10.4086/toc.2014.v010a018

In this work we show that most previous PIT algorithms be strengthened to yield algorithms for the ROT problem. In particular 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 sum of k ROFs. The running time of the ROT algorithm is essentially the same as the running time of the corresponding PIT algorithm for the class.

The main tool in most of our results is a new connection between PIT and reconstruction of ROFs. 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.

Contents

1 Introduction

Let $\mathbb{F}$ be a field and $\mathcal{C}$ a class of arithmetic circuits. The *reconstruction* problem for the class $\mathcal{C}$ is defined as follows. Given black-box (i. e., oracle) access to a polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, computable by an arithmetic circuit from the class $\mathcal{C}$, construct a circuit $C \in \mathcal{C}$ that computes P. A reconstruction algorithm is *efficient* if the number of queries it makes to the polynomial and its running time are polynomial in the size of the representation of P in the class $\mathcal{C}$. The reconstruction problem can be considered as the algebraic analog of the learning problem. A special case of the problem is the *interpolation* problem, where the goal is to find the monomial representation of the circuit. This can be seen to be the reconstruction problem when $\mathcal{C}$ is the class of depth-2 circuits (see, e. g., [29]).

The reconstruction problem is tightly connected to the Polynomial Identity Testing (PIT for short) problem, in which we need to determine whether a given arithmetic circuit $C(\bar{x})$ computes the zero[1] polynomial. There are two common models for studying the PIT problem. In the *white-box* model the algorithm is given access to the circuit itself. In particular, the algorithm can take into account the underlying graph of the circuit, etc. In the *black-box* model, the algorithm is not given access to the circuit and instead it can only query the value of the circuit on different inputs. Thus, a deterministic black-box PIT algorithm for a circuit class $\mathcal{C}$ has to compute a *hitting-set* for that class, 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. Note that if $\mathbb{F}$ is a small field, e. g., $\mathbb{F} = \mathbb{F}_2$, then a hitting set does not necessarily exist. Indeed, over $\mathbb{F}_2$ we have that $x^2 = x$, but they are not equal as polynomials. Thus, to construct a hitting set we may need to use inputs from a polynomially large extension field. One of the main properties of a hitting set $\mathcal{H}$ is that the values of any circuit from $\mathcal{C}$ on $\mathcal{H}$ describe the polynomial computed by that circuit uniquely, in the sense that two circuits that agree on $\mathcal{H}$ must compute the same polynomial since their difference[2] evaluates to zero over $\mathcal{H}$. Yet, such a set does not provide us with 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. For more information on the reconstruction problem and on PIT we refer the reader to the survey [44].

Several works have shown that efficient deterministic PIT algorithms imply lower bounds for arithmetic circuits [19, 21, 1]. Thus, the deterministic 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 [29], depth-3 circuits with bounded top fan-in [39, 24], non-commutative formulas [6, 28] and multilinear depth-4 circuits with top fan-in 2 [17]. The motivation to study restricted models is two fold. One reason is that many restricted models are interesting by themselves. Another reason is that we hope to gain insight from restricted models for the general problem.

The focus of this paper is on models related to arithmetic read-once 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. A *preprocessed ROF* (PROF for short) is a ROF in which we are allowed to replace each variable x_i with a univariate

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

[2]In fact, this statement requires an additional assumption that given C_1, C_2 in $\mathcal{C}$ the circuit $C_1 - C_2$ is in the class as well. But this assumption holds true for most natural classes.

polynomial $T_i(x_i)$. Although read-once formulas form a very restricted model of computation, they received a lot of attention in both the Boolean and the algebraic models. Over the Boolean domain, [22, 4, 11] gave efficient learning algorithms for several classes of Boolean read-once formulas. In the algebraic model, [18, 10, 8, 9] gave efficient *randomized* reconstruction algorithms for ROFs. However, prior to this work, no deterministic sub-exponential time reconstruction algorithm for ROFs was known. This state of affairs implies that if we want to give efficient deterministic algorithms for bounded-depth multilinear circuits or for multilinear formulas then we should first try to find such algorithms for ROFs. In view of this, we focus in this work on ROFs and, as a result, give the first deterministic sub-exponential time reconstruction algorithm for PROFs.

1.1 Our results

Our results require that the underlying field is of polynomial size. In case that $|\mathbb{F}|$ is too small we make queries to a polynomial-sized extension field. This is common to many (all) PIT algorithms, see discussions in [29, 44]. The running time of our algorithms is polynomial in $\log|\mathbb{F}|$ when $\mathbb{F}$ is finite, and is polynomial in the bit-size of the field elements appearing in the formula, for general fields. We now state our results formally.

1.1.1 Reconstruction of preprocessed read-once formulas

Theorem 1.1. *There is a deterministic $(nd)^{\mathcal{O}(\log n)}$ time algorithm that given black-box access to a PROF Φ, on n variables and individual degrees (of the preprocessing) at most d, over a field $\mathbb{F}$, reconstructs Φ, i. e., it constructs a PROF that computes the same polynomial. If $|\mathbb{F}| \leq n^2 d$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than $n^2 d$.*

Moreover, there is an algorithm that performs this task in a non-adaptive manner, that is, when each query to the black-box is independent of the results of the previous ones, with roughly the same running time.

Theorem 1.2. *There is a deterministic $(nd)^{\mathcal{O}(\log n)}$ time algorithm that given black-box access to a PROF Φ, on n variables and individual degrees (of the preprocessing) at most d, over a field $\mathbb{F}$, reconstructs Φ. Moreover, the algorithm queries Φ on a fixed set of points of size $(nd)^{\mathcal{O}(\log n)}$. If $|\mathbb{F}| \leq n^3 d$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than $n^3 d$.*

We also re-establish the result of [10] that gives a randomized polynomial-time reconstruction algorithm for ROFs. In fact, we show that a similar approach will work for PROFs as well.

Theorem 1.3. *There is a polynomial-time randomized algorithm that given black-box access to a PROF Φ, on n variable and individual degrees (of the preprocessing) at most d, reconstructs Φ with high probability. If $|\mathbb{F}| \leq 4n^2 d$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than $4n^2 d$.*

1.1.2 Read-once testing

In addition to reconstruction of preprocessed read-once formulas we are also interested in a generalization of the problem that we call the *read-once testing problem* (ROT for short).[3]

Problem 1.4 (ROT)**.** Given a circuit C (maybe as a black-box) computing some polynomial $P(\bar{x})$, decide whether P can be computed by a PROF, and if the answer is positive then compute a PROF for it.

Similarly to the PIT problem, there is an efficient randomized algorithm for the ROT problem. Indeed, by the results of [10] (or Theorem 1.3), there is a randomized algorithm for reconstructing a given black-box ROF. Thus, given access to the circuit C we can run the reconstruction algorithm to get a candidate PROF and then, invoking the Schwartz-Zippel randomized identity testing algorithm [46, 38], we can check whether the ROF that we computed earlier, computes the same polynomial as the one in the black-box. This results in a two-sided error algorithm. A simpler, one-sided error algorithm follows for the recent result of [45], which provides a characterization of polynomials computed by ROFs.

The main question is then whether we can find an efficient deterministic algorithm for the ROT problem. Clearly, this is a generalization of the PIT problem, as the zero polynomial is computable by a ROF. We show that if we are given an efficient deterministic PIT algorithm for a circuit class $\mathcal{C}$, that satisfies some "nice" closure properties, then we can also solve the ROT problem for that class efficiently. Moreover, the reduction works both in the black-box and in the white-box models, depending on the PIT algorithm at hand.

An argument similar to the above one shows that an efficient randomized reconstruction algorithm for a circuit class $\mathcal{C}$ implies an efficient randomized algorithm for the corresponding "$\mathcal{C}$-testing" problem (i. e., checking whether a given arithmetic circuit can be implemented within the class $\mathcal{C}$). Yet, no such connection is known in the deterministic setting. 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 (see, e. g., [15]).

Our main result for ROT is given in the following theorem.

Theorem 1.5. *Let $\mathcal{C}$ be a class of arithmetic circuits. Denote by $\mathcal{L}(\mathcal{C})$ the class of all circuits of the form $\alpha \cdot C + \beta$ for $\alpha, \beta \in \mathbb{F}$ and $C \in \mathcal{C}$ (see Definition 2.18). Denote with $\mathcal{C}_V$ 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 $\mathcal{L}(\mathcal{C})$ and C_2, C_3 and C_4 are pairwise variable disjoint. Suppose that there is a deterministic PIT algorithm A *that runs in time $T(n,d,s)$, when given access (white-box or black-box) to a circuit in n-variables, of size s and degree d, that belongs to $\mathcal{C}_V$. Then, there is a deterministic algorithm* B*, that uses* A *as a subroutine, such that when given access (white-box or black-box) to an n-variate circuit of size s and degree d from $\mathcal{C}$, solves the ROT problem in time $\mathrm{poly}(n,d,s,T(n,d,s))$. If $|\mathbb{F}| \leq nd$ then the algorithm may make queries from an extension field of size $> nd$. Similarly, if the given PIT algorithm puts an additional requirement on the size of the field (a lower bound) then we allow our algorithm to make queries from an appropriate extension field of $\mathbb{F}$.*

[3]It may be more accurate to denote this problem as *preprocessed read-once testing and reconstructing*, but for brevity we use read-once testing.

Note that C_1 may share variables with C_2, C_3 and C_4. The requirement that we have a PIT for the class $\mathcal{C}_V$ may seem odd at first, but it is explained by the way our algorithm works: it first constructs a candidate ROF and then makes several tests of the form $C_1 \overset{?}{=} C_2 + C_3 \times C_4$, thinking of C_2, C_3 and C_4 as "disjoint parts" of a ROF C_1, which is a restriction of the ROF that we constructed at the first step. For more details see the discussion in Section 6.2.

We note that for most of the circuit classes $\mathcal{C}$ for which we have efficient deterministic PIT algorithms, we also have such algorithms for their "closure," $\mathcal{C}_V$. As a corollary we get that all previous PIT algorithms can be generalized to yield algorithms for ROT. We demonstrate this point in the next theorem.

Theorem 1.6. *Let F be a multilinear read-k formula (i. e., a multilinear formula in which each variable appears at most k times) over $\mathbb{F}[x_1, x_2, \ldots, x_n]$. Then:*

- *There is a deterministic algorithm that given white-box access to F solves the ROT problem for F in time $n^{k^{\mathcal{O}(k)}}$.*
- *There is a deterministic algorithm that given black-box access to F solves the ROT problem for F in time $n^{k^{\mathcal{O}(k)} + \mathcal{O}(k \log n)}$.*

If $|\mathbb{F}| \leq n^2$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than n^2.

This result generalizes both the black-box and the white-box PIT algorithms for multilinear read-k formulas of [3].

Our next result strengthens many reconstruction algorithms that were obtained for the model of sparse polynomials ([7, 16, 29] to name just a few). The result uses the black-box PIT algorithm for sparse polynomials of [29], yet any of the above algorithms can be used instead. As was communicated to us by Lisa Hellerstein [20], for the case where P is a multilinear polynomial a similar result was proved in [30].

Theorem 1.7. *There is a deterministic polynomial time algorithm that given black-box access to an n-variate polynomial P, of degree d with m monomials (i. e., a polynomial computed by a depth-2 arithmetic circuit with m multiplication gates), solves the ROT problem for P. The running time of the algorithm is a polynomial in n, m and d. If $|\mathbb{F}| \leq (nd)^6$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than $(nd)^6$.*

Our next result solves the ROT problem for depth-3 and depth-4 circuits, with bounded top fan-in (see formal definition in Sections 7.2 and 7.3). It generalizes the results of [12, 27, 26, 25, 33, 36] and others, who gave deterministic PIT algorithms for these models. The theorem uses the black-box PIT algorithm for $\Sigma\Pi\Sigma(k)$ circuits of [36].

Theorem 1.8. *There is a deterministic algorithm that given black-box access to a depth-3 circuit C, of top fan-in k (i. e., a $\Sigma\Pi\Sigma(k)$ circuit), solves the ROT problem for the polynomial computed by the circuit. The running time of the algorithm is $\mathrm{poly}(n) \cdot d^{\mathcal{O}(k^2)}$, where n is the number of variables and d is the degree of the circuit. If $|\mathbb{F}| \leq dnk^2$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than dnk^2.*

Our next result uses the black-box PIT algorithm for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits of [33] and obtains ROT algorithms for this class.

Theorem 1.9. *There is a deterministic algorithm that given black-box access to a multilinear depth-4 circuit C, of top fan-in k (i. e., a $\Sigma\Pi\Sigma\Pi(k)$ circuit), solves the ROT problem. The running time of the algorithm is $(ns)^{\mathcal{O}(k^6)}$, where n is the number of variables and s is the size of the circuit. If $|\mathbb{F}| \le n^2$ then the algorithm may make queries from an extension field of $\mathbb{F}$ of size larger than n^2.*

We can also extend our results to hold for non-commutative formulas, for which [32] gave a polynomial time white-box PIT algorithm and [14, 13] gave quasi-polynomial time black-box PIT algorithms. However, extending our results to the non-commutative case requires redoing many steps of the algorithm, without introducing any new ideas, and so we omit the proof. We nevertheless write some algorithms in a way that allows one to apply them in non-commutative models as well (by making sure we do not change the order of left and right children of a multiplication gate, etc.).

1.2 Related work

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

In [18] algorithms for learning ROFs, 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[4] 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 [10] 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. The learning algorithm of [10] can reconstruct, with high probability, ROFs 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 and use) are given in advance to the algorithm. This result was later generalized in [8] who gave a randomized reconstruction algorithm for ROFs that use addition, multiplication, division and exponentiation gates.

In this work we give deterministic reconstruction algorithms, introduce the ROT problem, show the relation between PIT and ROT and, as a result, obtain several deterministic ROT algorithms. Our reconstruction algorithms diverge from previous work in that our algorithms are deterministic, we do not need to use randomness at all.[5] We also study the more general problem of ROT. To the best of our knowledge, this is the first time that ROT is studied. The main difference between the reconstruction problem and the ROT problem is that in the reconstruction setting we are guaranteed that there exists a PROF for the given circuit, whereas in the ROT problem we first have to check whether the polynomial can be computed by a PROF and only then we can try and find such a formula. The first step is the main difficulty in designing ROT algorithms.

Finally, we discuss the relation to our previous papers [40, 41]. The core ideas behind this work appeared as an extended abstract in [40]. There, the ROT problem was formally introduced and shown to

[4]In the learning theory literature, the word oracle is used to describe the function in the black-box and so we use this terminology here too.

[5]Equivalence queries can be viewed as randomized queries, as they allow one to solve PIT efficiently.

be computationally equivalent to the PIT problem. That is, it was shown that a circuit class $\mathcal{C}$, satisfying some closure properties, has an efficient deterministic ROT algorithm if and only if it has an efficient deterministic PIT algorithm. That paper also gave the first deterministic sub-exponential time PIT algorithm for sum of ROFs. Later, in [41], a more efficient deterministic PIT algorithm for sum of ROFs was obtained. That work also extended our previous results to PROFs. In conclusion, the basic ideas behind our reconstruction and ROT algorithms come from the paper [40], and the improved PIT algorithm of [41] is only used to improve the running time of the algorithms (in the black-box model).

In this paper we only consider reconstruction and testing of PROFs. We defer our results concerning PIT algorithms to another paper ([43][6]) to make this work more focused.

1.3 Techniques

To explain our approach we first need the notion of the *gate-graph* of a ROF. This is a graph on the set of variables, in which two variables are connected if and only if their first common ancestor gate is a multiplication gate (see Definition 3.3).

Our reconstruction algorithms use, at a high level, the gate-by-gate reconstruction technique of [18, 10]. The idea is to first first learn the gate-graph of the ROF. Once we have the gate-graph, we can learn the top gate of the formula by, roughly, checking whether the graph is connected or disconnected. We can thus reconstruct the tree of the formula recursively. In the process we also learn the "fine details," namely, we reconstruct the constants in the circuit. For reconstructing PROFs, we first learn the underlying preprocessing and then proceed with the reconstruction algorithm for the underlying ROF.

Compared to the techniques of [18, 10] our algorithm does not require the ability to compute square roots of field elements, which is assumed to be the case in [10]. The algorithm of [10] used randomness to gain access to so called *justifying assignments*. Instead, we use PIT algorithms for ROF in order to deterministically compute justifying assignments. Furthermore, we introduce some new ingredients that allow us to extend the algorithm to the case of PROFs.

The second part of the paper is devoted to the ROT problem with the main idea being "Doveryai, no Proveryai."[7] Given (white-box or black-box) access to an input circuit C, we first apply our reconstruction algorithm and construct a PROF Φ, that supposedly computes the same polynomial as C. Recall that our reconstruction algorithm works in the black-box model so we can pretend that C is a PROF and run the algorithm on C. If the algorithm fails then we know that the polynomial computed by C cannot be computed by a PROF. However, if the reconstruction does not fail then we are not done yet. At this point we have a PROF Φ and we have to test whether $C \equiv \Phi$. We do so by going over Φ gate-by-gate, mimicking the reconstruction algorithm. Since we know Φ, we can set variables to constants and obtain access to sub-formulas of Φ. Making appropriate queries to the circuit C, we can compare the restricted Φ and the restricted C. We can thus check if "locally" C and Φ agree. It is at this point that we need access to a PIT algorithm for circuits of the form $C_1 + C_2 + C_3 \times C_4$ as in Theorem 1.5. A more detailed explanation of the verification procedure is given in Section 6.2.

[6]This is the full version of the paper [41].

[7]"Trust, but Verify"—a translation of a Russian proverb which became a signature phrase of Ronald Reagan during the cold war.

1.4 Subsequent work

Recently, in [45] a simple characterization of polynomials computable by ROFs was given. Those polynomials are refereed to as read-once polynomials (ROP for short), see Definition 3.1. Roughly speaking, it is shown that a polynomial P is a ROP if and only if all its restrictions to three variables are ROPs. In other words, a polynomial P can be computed by a ROF if and only if it can be "locally" computed by ROFs. This result is obtained by borrowing and extending several techniques from the current paper.

1.5 Organization

The paper is organized as follows. In Sections 2 and 3 we give the basic definitions and notation. In Section 4 we give some basic algorithms for ROFs. Then, in Section 5 we prove Theorems 1.1, 1.2 and 1.3. Next, in Section 6 we show how to convert efficient deterministic PIT algorithms to efficient deterministic ROT algorithms and prove Theorem 1.5. We conclude the paper (Section 7) by showing some corollaries of Theorem 1.5, thus proving Theorems 1.6, 1.7, 1.8 and 1.9.

2 Preliminaries

For a positive integer n we denote $[n] = \{1,\dots,n\}$. Given a graph $G = (V,E)$ we denote with G^c the complement graph.

For a polynomial $P(x_1,\dots,x_n)$, a variable x_i and a field element α we denote with $P|_{x_i=\alpha}$ the polynomial resulting from setting $x_i = \alpha$. We say that P *depends* on x_i if there exist assignments $\bar{a},\bar{b} \in \overline{\mathbb{F}}^n$, which differ only on the i-th coordinate, such that $P(\bar{a}) \neq P(\bar{b})$. We denote

$$\mathrm{var}(P) \triangleq \{x_i \mid P \text{ depends on } x_i\}.$$

Given a subset $I \subseteq [n]$ we say that P is *defined on I* if $\mathrm{var}(P) \subseteq I$. Intuitively, P depends on x_i if x_i "appears" when P is listed as a sum of monomials. On the other hand, a constant function $P \equiv c$ is defined on every subset of $[n]$.

Definition 2.1. 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 substituting a_i to x_i, for every $i \in I$. In particular $P|_{x_I=\bar{a}_I}$ is defined on $[n] \setminus I$.

Observation 2.2. Let $J \subseteq I \subseteq \mathrm{var}(P)$ be subsets of $\mathrm{var}(P)$. Then for every assignment $\bar{a} \in \mathbb{F}^n$ it must be the case that $\mathrm{var}(P|_{x_I=\bar{a}_I}) \subseteq \mathrm{var}(P|_{x_J=\bar{a}_J}) \subseteq \mathrm{var}(P) \setminus J$.

Note that by substituting a value to a variable of P we, obviously, eliminate the dependence of P on that variable. This, however, may also eliminate the dependence of P on other variables and thus lose more information than intended. When we wish to reconstruct a formula, we cannot allow losing any information as it would affect our final answer. We now define a "lossless" type of an assignment. Similar definitions were given in [18] and [10], but we repeat the definitions here to ease the reading of the paper (we also slightly change some of the definitions).

Definition 2.3 (Justifying assignment)**.** We say that an assignment $\bar{a} \in \mathbb{F}^n$ is a *justifying assignment* for a polynomial P if for every subset $I \subseteq \text{var}(P)$ we have that

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

We say that a polynomial P is *$\bar{a}$-justified* if $\bar{a}$ is a justifying assignment for P. A set $\mathcal{J} \subseteq \mathbb{F}^n$ is called a *justifying set* for a circuit class $\mathcal{C}$, if it contains a justifying assignment for every polynomial P computed by a circuit from $\mathcal{C}$.

The next proposition shows that in order to verify whether P is $\bar{a}$-justified it is enough to consider only subsets I of size $|\text{var}(P)|-1$, rather than all subsets of $\text{var}(P)$.

Proposition 2.4. *An assignment $\bar{a} \in \mathbb{F}^n$ is a justifying assignment for P if and only if equation* (2.1) *holds for every subset I of size* $|\text{var}(P)|-1$.

Proof. Let $J \subseteq \text{var}(P)$. If $J = \text{var}(P)$ then, obviously, $\text{var}(P|_{x_J=\bar{a}_J}) = \emptyset = \text{var}(P) \setminus J$. Otherwise, let $x \in \text{var}(P) \setminus J$. Consider the set $I = \text{var}(P) \setminus \{x\}$. From the definition: $|I| = |\text{var}(P)|-1$ and $J \subseteq I$, and thus (2.1) holds for I. Combining with Observation 2.2 we obtain that

$$\{x\} = \text{var}(P) \setminus I = \text{var}(P|_{x_I=\bar{a}_I}) \subseteq \text{var}(P|_{x_J=\bar{a}_J}) \subseteq \text{var}(P) \setminus J\,.$$

Hence, $x \in \text{var}(P|_{x_J=\bar{a}_J})$. As this holds for every $x \in \text{var}(P) \setminus J$, it follows that $\text{var}(P) \setminus J \subseteq \text{var}(P|_{x_J=\bar{a}_J})$, and hence the two sets are equal. The second direction of the claim is trivial. □

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 2.5. *Let $\bar{a} \in \mathbb{F}^n$ and let $P(\bar{x})$ be an $\bar{a}$-*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 $\bar{0}$-justified polynomial. In addition, $P(\bar{x}+\bar{a}) \equiv 0$ if and only if $P(\bar{x}) \equiv 0$.

Definition 2.6 ($\bar{0}$-justified closure)**.** We denote

$$\text{var}_0(P) \triangleq \left\{ x_i \;\middle|\; P|_{x_{[n]\setminus\{i\}}=\bar{0}_{[n]\setminus\{i\}}} \text{ depends on } x_i \right\}.$$

That is, $\text{var}_0(P)$ is the maximal set of variables such that P is $\bar{0}$-justified with respect to this set. Clearly, $\text{var}_0(P) \subseteq \text{var}(P)$ and equality holds iff P is $\bar{0}$-justified.

Lemma 2.7. *A polynomial P is $\bar{0}$-justified iff* $\text{var}(P) = \text{var}_0(P)$.

The *Hamming weight* of a vector $\bar{a} \in \mathbb{F}^n$ is defined as: $\text{wt}(\bar{a}) \triangleq |\{i \mid a_i \neq 0\}|$. That is, the number of non-zero coordinates.

Definition 2.8. For a set $0 \in W \subseteq \mathbb{F}$ and $k \le t$ we define $\mathcal{A}^t_k(W)$ to be the set of all vectors in W^t with Hamming weight at most k.

An immediate conclusion from the definition is that for a finite set W (that contains 0), it holds that

$$\left|\mathcal{A}^t_k(W)\right| = \sum_{j=0}^{k} \binom{t}{j} \cdot (|W|-1)^j = (t \cdot (|W|-1))^{\mathcal{O}(k)}\,.$$

2.1 Partial derivatives

The concept of a *partial derivative* of a multivariate function and its properties (for example, P depends on x_i if and only if $\frac{\partial P}{\partial x_i} \not\equiv 0$) are well-known and well-studied for continuous domains (such as, $\mathbb{R}$, $\mathbb{C}$ etc.). This concept can be extended to polynomials over arbitrary fields, by defining the *discrete* partial derivative. Discrete partial derivatives play an important role in the analysis of our algorithms.

Definition 2.9. Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial. We define the *discrete partial derivative* of P, with respect to x_i, as

$$\frac{\partial P}{\partial x_i} \triangleq P|_{x_i=1} - P|_{x_i=0}\,.$$

Notice that if P is a multilinear polynomial then this definition coincides with the formal definition for polynomials. The following lemma is easy to verify.

Lemma 2.10. *The following properties hold for a multilinear polynomial P:*

- *P depends on x_i if and only if* $\dfrac{\partial P}{\partial x_i} \not\equiv 0$.
- $\dfrac{\partial P}{\partial x_i}$ *does* not *depend on x_i (in particular* $\dfrac{\partial^2 P}{\partial x_i^2} \equiv 0$*).*
- $\dfrac{\partial^2 P}{\partial x_i \partial x_j} = \dfrac{\partial}{\partial x_i}\left(\dfrac{\partial P}{\partial x_j}\right) = \dfrac{\partial^2 P}{\partial x_j \partial x_i}$.
- $\forall i \neq j,\ \dfrac{\partial P}{\partial x_i}|_{x_j=a} = \dfrac{\partial}{\partial x_i}(P|_{x_j=a})$.
- *$\bar{a} \in \mathbb{F}^n$ is a justifying assignment for P if and only if for every $i \in \mathrm{var}(P)$ it holds that* $\dfrac{\partial P}{\partial x_i}(\bar{a}) \not\equiv 0$.

Since the above properties trivially hold, we shall use them implicitly throughout the paper. We note that these basic properties do not hold for non-multilinear polynomials. For example, when $P(x) = x^2 - x$ we get that $\frac{\partial P}{\partial x} \equiv 0$. Thus, to handle general polynomial we need the following extension.

Definition 2.11. Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial. The *directed* partial derivative of P, with respect to x_i and direction $\alpha \in \mathbb{F}$, is defined as

$$\frac{\partial P}{\partial_\alpha x_i} \triangleq P|_{x_i=\alpha} - P|_{x_i=0}\,.$$

Definition 2.12. For a non-empty subset $I \subseteq [n]$, $I = \left\{i_1, \ldots, i_{|I|}\right\}$, we define the iterated partial derivative with respect to I in the following way:

$$\partial_I P \triangleq \frac{\partial^{|I|} P}{\partial x_{i_1} \partial x_{i_2} \partial x_{i_3} \cdots \partial x_{i_{|I|}}}\,.$$

2.2 Commutator

We now formally introduce one of our main reconstruction tools. Previously it was defined and used in [42] for the purpose of polynomial factorization. Here we define a simpler, "multilinear" version of the commutator since we are only going to apply it to multilinear polynomials. It is not hard to see that both our definition and the commutator of [42] have the same properties when restricted to multilinear polynomials.

Definition 2.13. Let $P \in \mathbb{F}[x_1,x_2,\ldots,x_n]$ be a polynomial and let $i,j \in [n]$. We define the *commutator* between x_i and x_j as

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

Definition 2.14. Let $P \in \mathbb{F}[x_1,x_2,\ldots,x_n]$ be a polynomial, $c \in \mathbb{F}$, and $i \neq j \in [n]$. We say that P is (x_i,x_j)-*separated mod* c if P can be written as $P = h \cdot g + c$, where $x_i \in \mathrm{var}(h) \setminus \mathrm{var}(g)$ and $x_j \in \mathrm{var}(g) \setminus \mathrm{var}(h)$. We say that P is (x_i,x_j)-*separated mod* $\mathbb{F}$ if there exists a field element $c \in \mathbb{F}$ such that P is (x_i,x_j)-separated mod c.

We next show that the commutator actually tells us whether a polynomial is separated or not.

Lemma 2.15 (Lemma 4.6 of [42]). *Let $P \in \mathbb{F}[x_1,x_2,\ldots,x_n]$ be a multilinear polynomial and let $x_i \neq x_j \in \mathrm{var}(P)$. Then P is (x_i,x_j)-separated mod 0 if and only if $\Delta_{ij}P \equiv 0$.*

The next observation follows from the definition of the commutator.

Observation 2.16. Let $P,R \in \mathbb{F}[x_1,x_2,\ldots,x_n]$ be multilinear polynomials with $x_i,x_j \notin \mathrm{var}(R)$. Then

$$\Delta_{ij}(P+R) = \Delta_{ij}P + R \cdot \frac{\partial^2 P}{\partial x_i \partial x_j} .$$

The following is an immediate corollary of Lemma 2.15 and Observation 2.16, which extends Lemma 2.15.

Corollary 2.17. *Let $P \in \mathbb{F}[x_1,x_2,\ldots,x_n]$ be a multilinear polynomial and let $x_i \neq x_j \in \mathrm{var}(P)$. Let $c \in \mathbb{F}$ be a field element. Then P is (x_i,x_j)-separated mod c if and only if*

$$\Delta_{ij}P = c \cdot \frac{\partial^2 P}{\partial x_i \partial x_j} .$$

2.3 Polynomials and circuit classes

We shall associate with a circuit class $\mathcal{C}$ the set of polynomials that can be computed by circuits from $\mathcal{C}$. The following notation will be useful for us.

Definition 2.18.

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:
$$\partial\mathcal{C} \triangleq \left\{ \frac{\partial P}{\partial_\alpha x_i} \,\middle|\, P \in \mathcal{C}, i \in [n], \alpha \in \mathbb{F} \right\}.$$

4. We say that a circuit class $\mathcal{C}$ is *closed under partial derivatives* if $\partial\mathcal{C} \subseteq \mathcal{C}$.

5. The *linear closure* of a circuit class $\mathcal{C}$ is $\mathcal{L}(\mathcal{C}) \triangleq \{\alpha \cdot P + \beta \mid P \in \mathcal{C} \text{ and } \alpha, \beta \in \mathbb{F}\}$.

3 Our model

In this section we discuss our computational model. We first consider the basic model of ROFs and cover some of its main properties. Then, we introduce the model of PROFs and give the corresponding basic properties.

3.1 ROFs and ROPs

Most of the definitions that we give in this section, or small variants of them, are from [18]. We start by formally defining the notions of a ROF and a ROP.

Definition 3.1. An *arithmetic read-once formula* (ROF for short) Φ, in the variables $\bar{x} = (x_1, \ldots, x_n)$, over a field $\mathbb{F}$, is a binary tree whose leafs are labelled with the input variables and whose internal nodes are labelled with the arithmetic operations $\{+, \times\}$ and with a pair of field elements[8] $(\alpha, \beta) \in \mathbb{F}^2$. 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 (α, β) computes the polynomial $\alpha \cdot x_i + \beta$. If a node v is labelled with the operation $\mathrm{op} \in \{+, \times\}$ and with (α, β), and its children compute the polynomials Φ_{v_1} and Φ_{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 Φ is *non-degenerate* if the polynomial that it computes depends on all the variables appearing in it Φ. For the sake of completeness, we denote the non-degenerate ROF computing the constant field element β by the special CONST gate CN_β.

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

3.2 The gate-graph of ROFs and ROPs

In this section we define the important notion of *gate-graph* of a ROF and a ROP. A similar notion was defined in [18], 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 gates. A similar notion was presented in [10].

[8]This is a slightly more general model than the usual definition of read-once formulas.

Definition 3.2. Let $V \subseteq \mathrm{var}(\Phi)$, $|V| \geq 2$, be a subset of the input variables of a ROF Φ. We define the *first common gate* of V, $\mathrm{fcg}(V)$, to be the first gate in the graph of computation of Φ common to all the paths from the inputs of V to the root of the formula.

We note that $\mathrm{fcg}(V)$ is in fact the least common ancestor of the nodes in V when viewing the formula as a tree. We now define, for every ROF, 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 [18] 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 ROFs. We now give a definition of a t-graph that is slightly different from the one in [18].

Definition 3.3 (t-graph)**.** Let Φ be a ROF in the input variables $\mathrm{var}(\Phi)$. Let $t \in \{+, \times\}$ be a possible label of an internal node. The t-graph of the formula Φ, denoted G^t_Φ, is an undirected graph whose vertex set is $\mathrm{var}(\Phi)$ and whose edges are defined as follows: (i, j) is an edge if and only if the arithmetic operation that labels the gate $\mathrm{fcg}(x_i, x_j)$ is t.

The following simple lemma (whose basic form can be found in [18]) gives some basic properties of t-graphs. We omit the proof as it is implicit in the proof of Lemma 3 in [18].

Lemma 3.4. *The following properties are satisfied by t-graphs.*

1. *Let P be a ROP and let Φ and Ψ be two non-degenerate ROFs computing P. Then $G^\times_\Phi = G^\times_\Psi$ and $G^+_\Phi = G^+_\Psi$. In particular, the graphs $G^\times_P \triangleq G^\times_\Phi$ and $G^+_P \triangleq G^+_\Phi$ are well defined. Consequently, the t-graph of a ROP is a well defined notion.*

2. *For every ROP P we have $G^\times_P = \left(G^+_P\right)^c$. 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 connected. More precisely, the type of the top gate of P is t if and only if G^t_P is connected.*

We define the *gate-graph* of a ROF Φ as $G_\Phi \triangleq G^\times_\Phi$. Similarly, we define G_P to be the gate-graph of a ROP P.

Lemma 3.5. *Let P be a ROP. Then (i, j) is an edge in the gate-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 $\prod_{i \in I} x_i$.*

Proof. The first part is immediate from the definition. The second part is easily proved by induction on $|I|$. □

The following is an algebraic version of the above lemma. It also provides a more algebraic definition of the gate-graph of a ROP P. The proof is similar to the proof of Lemma 3.5.

Lemma 3.6. *Let $P(x_1, \ldots, x_n)$ be a ROP. Then (i, j) is an edge in the gate-graph G_P if and only if*

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0\,.$$

Moreover, for $I \subseteq [n]$ we have that $\partial_I P \not\equiv 0$ if and only if I is a clique in G_P.

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

Example 3.7. For $n \geq 4$, the polynomial $P(x_1, x_2, \ldots, x_n) = x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n$ is not a ROP. Indeed, form Lemmas 3.4 and 3.5 we get that both the $+$-graph and the $\times$-graph are connected for this polynomial. From Lemma 3.4 we get that P is not a ROP.

Example 3.8. 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. Indeed, notice that G_P is complete graph (a clique) while $\partial_{[n]} P \equiv 0$. Lemma 3.6 implies that P is not a ROP.

The following lemma is also an easy corollary of Lemma 3.4.

Lemma 3.9 (ROP Structural Lemma)**.** *Every ROP $P(\bar{x})$ with $|\mathrm{var}(P)| \geq 2$ can be presented in* exactly *one of the following two forms:*

1. *$P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$ and*
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.

We end this section by providing two more useful properties related to partial derivatives of ROPs.

Lemma 3.10. *A partial derivative of a ROP is a ROP. Moreover, a partial derivative of a $\bar{0}$-justified ROP is also a $\bar{0}$-justified ROP.*

The proof is very similar to the proof of Lemma 5.12 in [43]. For completeness we give it in Appendix A.

Lemma 3.11. *Let P be a $\bar{0}$-justified ROP. Then*

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0 \qquad \textit{if and only if} \qquad \frac{\partial^2 P}{\partial x_i \partial x_j}(\bar{0}) \neq 0\,.$$

Proof. By Lemma 3.10, $\frac{\partial P}{\partial x_i}$ is a $\bar{0}$-justified ROP. Consequently, by the definition,

$$\frac{\partial}{\partial x_j}\left(\frac{\partial P}{\partial x_i}\right) \not\equiv 0 \qquad \text{if and only if} \qquad \left(\frac{\partial}{\partial x_j}\left(\frac{\partial P}{\partial x_i}\right)\right)(\bar{0}) \neq 0\,. \qquad \square$$

3.3 Factors of ROPs

In this section we study properties of factors of ROPs. We start by proving that every factor of a ROP is also a ROP. Although this is an easy claim to prove, our proof has the advantage that it can be used to devise a linear-time factorization algorithm for ROFs in the white-box model (see Algorithm 1 in Section 4.2.1).

Lemma 3.12. *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, where h_1 and h_2 are (not necessarily irreducible) non-constant factors. In particular, $|\mathrm{var}(P)| \geq 2$. According to Lemma 3.9 P has exactly one of the following two forms:

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Notice that h_1 and h_2 are variable disjoint. Consequently, we can assume w. l. o. g. that there exist x_i, x_j such that $x_i \in \mathrm{var}(h_1) \cap \mathrm{var}(P_1)$ and $x_j \in \mathrm{var}(h_2) \cap \mathrm{var}(P_2)$. 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 the 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 $\frac{\partial h_1}{\partial x_i}$ and $\frac{\partial h_2}{\partial x_j}$ are non-zero, we get a contradiction.

2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$. As previously, we can assume w. l. o. g. that there exist x_i, x_j such that $x_i \in \mathrm{var}(h_1) \cap \mathrm{var}(P_1)$ and $x_j \in \mathrm{var}(h_2) \cap \mathrm{var}(P_2)$. This time we consider $\Delta_{ij}P$. On the one hand $\Delta_{ij}P = \Delta_{ij}(h_1 \cdot h_2) \equiv 0$. On the other hand, by Observation 2.16,
$$\Delta_{ij}P = c \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j}.$$
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. □

As a corollary of the proof we deduce the following observation.

Observation 3.13. 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 two representations of a ROP P. Then $c = c'$. In other words, the constant c is unique.

3.4 Commutators of ROPs

In general, a commutator $\Delta_{ij}P$ of a polynomial P may be a more complicated polynomial than the original P. In this section we show that a commutator of a ROP has a nice structure. For our future purposes we consider the commutator between two variables x_i, x_j for which
$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0.$$

Lemma 3.14. *Let $P(\bar{x}) \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a ROP and let $i \neq j \in \operatorname{var}(P)$ be such that*

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0\,.$$

Then there exist variable disjoint ROPs $H(\bar{x})$ and $R(\bar{x}, y)$ such that:

1. *$P(\bar{x}) = R(\bar{x}, H(\bar{x}))$,*
2. *$\Delta_{ij}P = R(\bar{x}, 0) \cdot \dfrac{\partial^2 P}{\partial x_i \partial x_j}$,*
3. *H is (x_i, x_j)-separated mod 0.*

Proof. Let Φ be a ROF for P and let $v = \operatorname{fcg}(x_i, x_j)$ in Φ. Take $R'(\bar{x}, y)$ to be the ROP computed by Φ when we replace the subtree rooted at v with a new variable y, and let H' be the ROP computed by Φ_v. We can write $P(\bar{x}) = R'(\bar{x}, H'(\bar{x}))$. As v is a multiplication gate (by Lemma 3.6), Lemma 3.9 implies that H' is of the form $H'(\bar{x}) = h_1(\bar{x}) \cdot h_2(\bar{x}) + c$ for $c \in \mathbb{F}$, where $x_i \in \operatorname{var}(h_1)$ and $x_j \in \operatorname{var}(h_2)$. Define

$$H(\bar{x}) \triangleq H'(\bar{x}) - c \qquad \text{and} \qquad R(\bar{x}, y) \triangleq R'(\bar{x}, y + c)\,.$$

Observe that $P(\bar{x}) = R'(\bar{x}, H'(\bar{x})) = R(\bar{x}, H(\bar{x}))$ and that H is (x_i, x_j)-separated mod 0. This proves items 1 and 3 of the claim. For the remaining item, observe that we can write $R(\bar{x}, y) = R_y(\bar{x}) \cdot y + R_0(\bar{x})$, where

$$R_y = \frac{\partial R}{\partial y} \qquad \text{and} \qquad R_0 = R|_{y=0}\,.$$

By Observation 2.16 and the chain rule:

$$\Delta_{ij}(P) = \Delta_{ij}(R_y \cdot H + R_0) = R_0 \cdot \frac{\partial^2 (R_y \cdot H)}{\partial x_i \partial x_j} = R_0 \cdot R_y \cdot \frac{\partial^2 H}{\partial x_i \partial x_j} = R(\bar{x}, 0) \cdot \frac{\partial^2 P}{\partial x_i \partial x_j}\,,$$

as claimed. □

Unlike the partial derivative operator (see Lemma 3.10) the commutator does not preserve the property of being a $\bar{0}$-justified ROP. We show, however, that to some extent, the assignment $\bar{0}$ remains interesting when considering commutators of $\bar{0}$-justified ROPs. Specifically, it acts nicely on the ROP $R(\bar{x}, y)$ that was constructed in Lemma 3.14.

Lemma 3.15. *Let $R(\bar{x}, y) \in \mathbb{F}[x_1, x_2, \ldots, x_n, y]$ be a ROP with $|\operatorname{var}(R)| = n + 1$ such that:*

1. *there exists $\ell \in [n]$ such that $\dfrac{\partial R}{\partial x_\ell}(\bar{x}, 0) \not\equiv 0$, and*
2. *for each $i \in [n]$ we have that $\dfrac{\partial R}{\partial x_i}(\bar{0}, y) \not\equiv 0$.*

Then there exists $k \in [n]$ such that $\dfrac{\partial R}{\partial x_k}(\bar{0}, 0) \neq 0$.

Proof. By induction on n. Let $n = 1$. Clearly, $k = 1$. Since R is a multilinear polynomial we get that in this case

$$\frac{\partial R}{\partial x_1}(0,0) = \frac{\partial R}{\partial x_1}(x_1,0) \neq 0\,.$$

Now suppose that $n \geq 2$. By Lemma 3.9, R can be presented in one of the following forms:

Case 1: $R(\bar{x},y) = R_1(\bar{x},y) + R_2(\bar{x})$. Pick $x_k \in \mathrm{var}(R_2)$. As $y \notin \mathrm{var}(R_2)$. We have that:

$$\frac{\partial R}{\partial x_k}(\bar{0},0) = \frac{\partial R_2}{\partial x_k}(\bar{0},0) = \frac{\partial R_2}{\partial x_k}(\bar{0},y) \neq 0\,.$$

Case 2: $R(\bar{x}) = R_1(\bar{x},y) \cdot R_2(\bar{x}) + c$, where $c \in \mathbb{F}$. We have two subcases to consider.

Case 2a: There exists $x_j \in \mathrm{var}(R_1)$ such that $\frac{\partial R_1}{\partial x_j}(\bar{x},0) \not\equiv 0$. In particular, $|\mathrm{var}(R_1) \setminus \{y\}| \geq 1$. For each $x_i \in \mathrm{var}(R_1)$ we have that:

$$\frac{\partial R_1}{\partial x_i}(\bar{0},y) \cdot R_2(\bar{0}) = \frac{\partial R}{\partial x_i}(\bar{0},y) \not\equiv 0 \implies \frac{\partial R_1}{\partial x_i}(\bar{0},y) \not\equiv 0\;,\; R_2(\bar{0}) \neq 0\,.$$

Consequently, we can apply the induction hypothesis to R_1 and get that there exists $x_k \in \mathrm{var}(R_1)$ such that

$$\frac{\partial R_1}{\partial x_k}(\bar{0},0) \neq 0\,.$$

To finish the proof of this case, observe that

$$\frac{\partial R}{\partial x_k}(\bar{0},0) = \frac{\partial R_1}{\partial x_k}(\bar{0},0) \cdot R_2(\bar{0}) \neq 0\,.$$

Case 2b: For each $x_j \in \mathrm{var}(R_1)$ we have that

$$\frac{\partial R_1}{\partial x_j}(\bar{x},0) \equiv 0\,.$$

This implies that $R_1(\bar{x},0) \equiv b$ for some $b \in \mathbb{F}$. In particular, $x_\ell \in \mathrm{var}(R_2)$. Consequently,

$$b \cdot \frac{\partial R_2}{\partial x_\ell}(\bar{x}) = R_1(\bar{x},0) \cdot \frac{\partial R_2}{\partial x_\ell}(\bar{x}) = \frac{\partial R}{\partial x_\ell}(\bar{x},0) \not\equiv 0 \implies b \neq 0\,,$$

and

$$R_1(\bar{0},y) \cdot \frac{\partial R_2}{\partial x_\ell}(\bar{0}) = \frac{\partial R}{\partial x_\ell}(\bar{0},y) \not\equiv 0 \implies \frac{\partial R_2}{\partial x_\ell}(\bar{0}) \neq 0\,.$$

Hence,

$$\frac{\partial R}{\partial x_\ell}(\bar{0},0) = R_1(\bar{0},0) \cdot \frac{\partial R_2}{\partial x_\ell}(\bar{0}) = b \cdot \frac{\partial R_2}{\partial x_\ell}(\bar{0}) \neq 0\,.$$

Taking $k = \ell$ completes the proof. □

We can now show that in a $\bar{0}$-justified ROP the $\bar{0}$ assignment can, in some sense, preserve the separation property.

Lemma 3.16. *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a $\bar{0}$-justified ROP with $|\mathrm{var}(P)| \geq 3$ and $x_i \neq x_j \in \mathrm{var}(P)$. Then, P is (x_i, x_j)-separated mod $\mathbb{F}$ iff for every $k \in \mathrm{var}(P) \setminus \{x_i, x_j\}$ it holds that*

$$P|_{x_{[n]\setminus\{i,j,k\}}=\bar{0}_{[n]\setminus\{i,j,k\}}}$$

is (x_i, x_j)-separated mod $\mathbb{F}$.

Proof. We begin by noting that since

$$\mathrm{var}\left(P|_{x_{[n]\setminus\{i,j,k\}}=\bar{0}_{[n]\setminus\{i,j,k\}}}\right) = \{x_i, x_j, x_k\},$$

the first direction of the claim immediately follows. For the other direction, observe that

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0.$$

Therefore, we can apply Lemma 3.14 to obtain ROPs $H(\bar{x})$ and $R(\bar{x}, y)$ such that

$$P(\bar{x}) = R(\bar{x}, H(\bar{x})) \qquad \text{and} \qquad \Delta_{ij} P = R(\bar{x}, 0) \cdot \frac{\partial^2 P}{\partial x_i \partial x_j}.$$

Now, assume for a contradiction that P is not (x_i, x_j)-separated mod $\mathbb{F}$. We would like to apply Lemma 3.15 to reach a contradiction. We first show that $R(\bar{x}, y)$ satisfies the condition of that lemma.

Note that $R(\bar{x}, 0)$ must be a non-constant polynomial. Indeed, if it was constant then we would get that

$$\Delta_{ij} P = R(\bar{x}, 0) \cdot \frac{\partial^2 P}{\partial x_i \partial x_j} = c \cdot \frac{\partial^2 P}{\partial x_i \partial x_j},$$

in contradiction to Corollary 2.17. Thus, there exists $x_\ell \in \mathrm{var}(R)$ such that

$$\frac{\partial R}{\partial x_\ell}(\bar{x}, 0) \not\equiv 0.$$

Let $x_m \in \mathrm{var}(R)$. As P is $\bar{0}$-justified, we have that

$$\frac{\partial R}{\partial x_m}(\bar{0}, H(\bar{0})) = \frac{\partial P}{\partial x_m}(\bar{0}) \neq 0,$$

hence

$$\frac{\partial R}{\partial x_m}(\bar{0}, y) \not\equiv 0.$$

We have thus shown that R satisfies the conditions of Lemma 3.15. Consequently, there exists x_k such that

$$\frac{\partial R}{\partial x_k}(\bar{0}, 0) \neq 0.$$

To simplify notation, assume without loss of generality that $k = 1$, i. e., assume that

$$\frac{\partial R}{\partial x_1}(\bar{0}, 0) \neq 0 .$$

By Corollary 2.17 there must exist $c \in \mathbb{F}$ such that

$$\Delta_{ij}\left(P|_{x_{[n]\setminus\{1,i,j\}}=\bar{0}_{[n]\setminus\{1,i,j\}}}\right) = c \cdot \frac{\partial^2 P}{\partial x_i \partial x_j}(x_1, \bar{0}) .$$

On the other hand,

$$\Delta_{ij}\left(P|_{x_{[n]\setminus\{1,i,j\}}=\bar{0}_{[n]\setminus\{1,i,j\}}}\right) = R(x_1, \bar{0}, 0) \cdot \frac{\partial^2 P}{\partial x_i \partial x_j}(x_1, \bar{0}) .$$

As

$$\frac{\partial^2 P}{\partial x_i \partial x_j}(x_1, \bar{0}) \not\equiv 0$$

(by Lemma 3.11) it must be the case that $R(x_1, \bar{0}, 0) \equiv c$ in contradiction to

$$\frac{\partial R}{\partial x_1}(\bar{0}, 0) \neq 0 .$$

Therefore, P must be (x_i, x_j)-separated mod $\mathbb{F}$. □

3.5 PROPs

In this subsection 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 to be read-once in univariate polynomials.

Definition 3.17. 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 each T_i is monic and satisfies $T_i(0) = 0$.

Notice that preprocessing does not affect the PIT problem in the white-box model 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 simple properties of PROFs.

Definition 3.18. A *preprocessed arithmetic read-once formula* (PROF for short), in the variables $\bar{x} = (x_1, \ldots, x_n)$, over a field $\mathbb{F}$, 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 (α, β) computes the polynomial $\alpha \cdot T_i(x_i) + \beta$. If a node v is labelled with the operation op $\in \{+, \times\}$ and with (α, β), and its children compute the polynomials Φ_{v_1} and Φ_{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 PROF. A *decomposition* of a polynomial P is a pair $(Q(\bar{z}),T(\bar{x}))$ such that $P(\bar{x})=Q(T(\bar{x}))$, where Q is a ROP and T is a preprocessing. We call Q the *backbone* of P. We let the gate-graph of P, G_P, be the gate-graph of its backbone Q. A *standard decomposition* is as above with the additional requirement that T is a standard preprocessing. An immediate consequence of the definition is that each PROP admits a decomposition. To provide additional intuition we start with a simple, yet important, lemma.

Lemma 3.19. *Every PROP P admits a standard decomposition. In addition, if $(Q(\bar{z}),T(\bar{x}))$ and $(Q'(\bar{z}),T'(\bar{x}))$ are two decompositions of a PROP $P(\bar{x})$, then there exist two vectors $\bar{a},\bar{b}\in\mathbb{F}^n$, with $a_i\neq 0$ for every $i\in[n]$, such that*

$$Q(\bar{z})=Q'(a_1\cdot z_1+b_1,\dots,a_n\cdot z_n+b_n)\,,$$
$$T'(\bar{x})=(a_1\cdot T_1(x_1)+b_1,\dots,a_n\cdot T_2(x_2)+b_n)\,.$$

Moreover, if (Q,T) is a decomposition of a PROP P, then any pair (Q',T') that satisfies the above conditions is also a decomposition of P.

Proof. Let (Q,T) be some decomposition of P and let $c_i\neq 0$ denote the leading coefficient of x_i in the polynomial $T_i(x_i)$ for $i\in[n]$ (c_i is well-defined since T_i is non-constant). Consider the shifted polynomials:

$$\tilde{Q}(\bar{z})\triangleq Q\left(c_1\cdot z_1+T_1(0),c_2\cdot z_2+T_2(0),\dots,c_n\cdot z_n+T_n(0)\right)\,,$$
$$\tilde{T}_i(x_i)\triangleq\frac{T_i(x_i)-T_i(0)}{c_i}\,,\qquad\text{and}$$
$$\tilde{T}(\bar{x})\triangleq\left(\tilde{T}_1(x_1),\tilde{T}_2(x_2),\dots,\tilde{T}_n(x_n)\right)\,.$$

It is easy to verify that $(\tilde{Q},\tilde{T})$ is a standard decomposition of P. Given the above argument, the proofs of the second and the third properties follow easily and so we omit them. □

To handle the preprocessed case we will need the following definitions:

Definition 3.20. Let $P(\bar{x})=Q(T(\bar{x}))\in\mathbb{F}[x_1,x_2,\dots,x_n]$ be a PROP in its standard decomposition. We say that $\bar{\alpha}\in\mathbb{F}^n$ is a *witness* for P if for every $i\in[n]$ it holds that $T_i(\alpha_i)\neq 0$.

Definition 3.21. Let $P\in\mathbb{F}[x_1,x_2,\dots,x_n]$ be a polynomial, $\bar{\alpha}\in\mathbb{F}^n$ and $i,j\in[n]$. We define the *α-directed commutator* between x_i and x_j as

$$\Delta_{ij}^{\bar{\alpha}}P\triangleq P|_{x_i=\alpha_i,x_j=\alpha_j}\cdot P|_{x_i=0,x_j=0}-P|_{x_i=\alpha_i,x_j=0}\cdot P|_{x_i=0,x_j=\alpha_j}\,.$$

The next two lemmas follow easily from the definition and from following the proof for the case of ROFs and so we omit their proofs.

Lemma 3.22. *Let P, $(Q(\bar{z}),T(\bar{x}))$ be a PROP and its standard decomposition, respectively. Then the following properties hold for any $\bar{\alpha}\in\mathbb{F}[x_1,x_2,\dots,x_n]$.*

- $\dfrac{\partial P}{\partial_{\alpha_i}x_i}=\dfrac{\partial Q}{\partial z_i}\Big|_{\bar{z}=T(\bar{x})}\cdot\dfrac{\partial T_i}{\partial_{\alpha_i}x_i}=\dfrac{\partial Q}{\partial z_i}\Big|_{\bar{z}=T(\bar{x})}\cdot T_i(\alpha_i).$
  In particular $\left(T_i(\alpha_i)\cdot\dfrac{\partial Q}{\partial z_i}(\bar{z}),T(\bar{x})\right)$ *is a standard decomposition of* $\dfrac{\partial P}{\partial_\alpha x_i}$.

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

The following is the PROP version of Lemma 3.9.

Lemma 3.23 (PROP Structural Lemma)**.** *Every PROP $P(\bar{x})$ such that $|\mathrm{var}(P)| \geq 2$ can be presented in* exactly *one of the following forms:*

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

where P_1, P_2 are non-constant, variable-disjoint PROPs and c is a constant.

4 Auxiliary algorithms

In this section we give some auxiliary algorithms to be used later.

4.1 From PIT to justifying assignments

We now show how to compute a justifying assignment from a PIT algorithm, following Section 4 in [43]. We shall consider a circuit class $\mathcal{C}$ (e. g., depth-3 circuits, ROFs, etc.) for which there exists another circuit class $\mathcal{C}'$ such that $\partial\mathcal{C} \subseteq \mathcal{C}'$ (recall Definition 2.18) and $\mathcal{C}'$ has an efficient deterministic PIT algorithm.[9]

Lemma 4.1 (Lemma 4.1 [43])**.** *Let $n, d \geq 1$. Assume $|\mathbb{F}| > nd$. Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial, with individual degrees bounded by d, that is computed by circuits from $\mathcal{C}$. Let $\mathcal{C}'$ be another (or the same) circuit class such that $\partial\mathcal{C} \subseteq \mathcal{C}'$. Suppose $\mathcal{C}'$ admits a deterministic PIT algorithm* A*. Then there exists a deterministic algorithm* B *that has black-box access to* A *and computes a justifying assignment $\bar{a}$ for P in time $\mathcal{O}(n^3 d \cdot t)$, where t is the running time of* A.

Lemma 4.1 gives an (adaptive) algorithm that computes a justifying assignment for any polynomial in $\mathcal{C}$, given access to a PIT algorithm for the class $\mathcal{C}'$. In order to obtain a non-adaptive reconstruction algorithm we will need a non-adaptive version of the lemma. More precisely, we require an algorithm that computes a *justifying set* for PROPs, i. e., a set that contains a justifying assignment for every PROP (recall Definition 2.3). In Section 4.1 of [43] it was also shown how to convert a black-box PIT algorithm into a justifying set of roughly the same size. In addition, [43] presented a black-box PIT algorithm for PROPs (see Lemma 5.12). Combining these results we get the following lemma:

Lemma 4.2. *Let $n, d \geq 1$. Assume $|\mathbb{F}| > n^3 d$. Then there exists a deterministic algorithm that, given n and d, runs in time $(nd)^{\mathcal{O}(\log n)}$ and computes a set, $\mathcal{J}_d$, of size $|\mathcal{J}_d| = (nd)^{\mathcal{O}(\log n)}$, which is a justifying set for PROPs, of individual degrees at most d, in $\mathbb{F}[x_1, x_2, \ldots, x_n]$.*

[9]Note that in most cases an identity testing algorithm for $\mathcal{C}$ can be slightly modified to yield an identity testing algorithm for $\partial\mathcal{C}$.

4.2 ROF graph-related algorithms

In this section we give two basic algorithms for PROFs in the white-box model. To slightly simplify the algorithms we shall assume w. l. o. g. that all the PROFs are non-degenerate. Notice that we can make this assumption since there is a trivial $\mathcal{O}(n)$ time white-box algorithm that can convert any ROF Φ to a non-degenerate ROF Φ' such that $\Phi \equiv \Phi'$.

For the analysis of the algorithms we will need the following easy observations. The first is that a non-degenerate ROF Φ computes the zero polynomial if and only if Φ is the "zero ROF," $\Phi = CN_0$ (the graph of computation of Φ is no other than CN_0). Secondly, in a non-degenerate ROF, for every gate-label (α, β), it holds that $\alpha \neq 0$.

We will use the following notation in our algorithms.

Definition 4.3. Let G be the graph of computation of a PROF. We think of G as a planar graph (and in particular the notions "left child" and "right child" of a node are well defined). Let v be a gate in the formula.

- Φ_v - denotes the formula rooted at v.
- v.var - denotes var(Φ_v). Note that v.var can be computed recursively.
- $v.\alpha$ - denotes the value of α labeling v.
- $v.\beta$ - denotes the value of β labeling v.
- v.Left - denotes the left child of v.
- v.Right - denotes the right child of v.
- v.Type - denotes the type of the arithmetic operation labeling v. (**IN** - Input, **CONST** - Constant Function gate (i. e., CN_α), $\times$ - Multiplication, **+** - Addition)

4.2.1 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 3.12. There we showed that a ROF Φ has non trivial factors if and only if its top gate is a multiplication gate and the additive constant β, of the top gate, is 0. Note that in this case Φ 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 1 computes a list $S = \{h_i\}$ of all irreducible factors of Φ_v (the polynomial computed by the node v) and a constant α 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 immediate from the proof of Lemma 3.12.

Lemma 4.4. *Given a node v in the graph of computation of a non-constant ROF Φ, Algorithm 1 computes a pair (S, α), where S consists of ROFs for the irreducible factors of Φ_v and α is a constant satisfying $\Phi_v = \alpha \cdot \prod_{h \in S} h$. The running time of the algorithm is $\mathcal{O}(n)$.*

Algorithm 1 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 ($\Phi$ , 1)
3: if $v$.Type $= +$ then
4:     return ($\Phi$ , 1)
5: if $v.\beta \neq 0$ then
6:     return ($\Phi$ , 1)
   {Now $v$.Type $= \times$ and $\beta = 0$, thus $\Phi = \alpha \cdot \Phi_{v.\mathrm{Right}} \times \Phi_{v.\mathrm{Left}}$}
7: $(S_L , \alpha_L) \leftarrow$ FactorROF($\Phi_{v.\mathrm{Left}}$)
8: $(S_R , \alpha_R) \leftarrow$ FactorROF($\Phi_{v.\mathrm{Right}}$)
9: return (union($S_L$ , $S_R$) , $\alpha_L \cdot v.\alpha \cdot \alpha_R$)
```

4.2.2 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 deterministic algorithm for counting the number of monomials computed by a given PROF. The main idea is that once a non-constant monomial appears, it cannot 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 not we use an auxiliary function $\mathbf{NZ}(x)$ that is defined as

$$\mathbf{NZ}(x) = \begin{cases} 0 & x = 0, \\ 1 & \text{otherwise.} \end{cases}$$

The algorithm simply recurses on the left child and on the right child of the root and combines the results according to the different values of the constant terms.

Lemma 4.5. *Given a node v in the graph of computation of a PROF Φ, with individual degrees at most d, Algorithm 2 computes a pair (M,C), where M is the number of* ***non-constant*** *monomials of Φ_v and C is the constant term of Φ_v. The running time of the algorithm is $\tilde{O}(nd+n^2)$.*

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 the (α,β) labeling of the gates. 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: Φ_v computes a constant or a univariate polynomial. In this case we can simply count the number of non-constant monomials and set M to this value.

Algorithm 2 CountMonomials(v)

Input: The root v of a PROF.

Output: (M,C) where M is the number of the **non-constant** monomials of Φ_v,
C is the constant term of Φ_v.

```
1: if $v$.Type = IN OR $v$.Type = CONST then {$\Phi_v$ is a univariate or constant polynomial}
2:     $M \leftarrow$ The number of non-constant monomials in $\Phi_v$
3:     return $(M, v.\beta)$
   {Internal gate}
4: $(M_L, C_L) \leftarrow$ CountMonomials($v$.Left)
5: $(M_R, C_R) \leftarrow$ CountMonomials($v$.Right)
6: if $v$.Type = $\times$ then
7:     $C \leftarrow v.\alpha \cdot C_L \cdot C_R + v.\beta$
8:     $M \leftarrow (M_L + \mathbf{NZ}(C_L)) \cdot (M_R + \mathbf{NZ}(C_R)) - \mathbf{NZ}(C_L \cdot C_R)$
9:     return $(M, C)$
10: if $v$.Type = $+$ then
11:     $C \leftarrow v.\alpha \cdot (C_L + C_R) + v.\beta$
12:     $M \leftarrow M_L + M_R$
13:     return $(M, C)$
```

When v is not a leaf we recursively run the algorithm on the left child and on the right child. Denote with p_L and p_R the polynomials computed by the left child and the right child, respectively. We again analyze the different options for v.Type.

- v.Type = $\times$: Φ_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 $(M_L + \mathbf{NZ}(C_L)) \cdot (M_R + \mathbf{NZ}(C_R))$. Therefore, the number of non-constant monomials of $p_L \times p_R$ is $(M_L + \mathbf{NZ}(C_L)) \cdot (M_R + \mathbf{NZ}(C_R)) - \mathbf{NZ}(C_L \cdot C_R)$. From our observation this is exactly the number of non-constant monomials of Φ_v. The claim regarding C is trivial.

- v.Type = $+$: Φ_v computes a sum of two functions. $\Phi_v = v.\alpha \times (p_L + p_R) + v.\beta$ and, as before, the number of non-constant monomials of $p_L + p_R$ (and hence of Φ_v) is $M_L + M_R$. Again, the claim regarding C is trivial.

It is clear from the above analysis that the algorithm returns the correct answer. The claim regarding the running time follows easily from the fact that the algorithm is a simple graph traversal that requires $\mathcal{O}(n)$ time. As the individual degrees of the polynomial are bounded by d, computing the number of non-zero monomials for the leaves requires $\mathcal{O}(d)$ time. In addition, note that at each step we multiply two $n\log d$-bit numbers (since $0 \le M \le (d+1)^n$). Thus the total running time is $\tilde{\mathcal{O}}(nd + n^2)$. □

5 Reconstruction of a PROF

In this section we discuss the problem of PROF reconstruction: given black-box access to a PROP P we wish to construct a PROF Φ such that Φ computes P. We first give a reconstruction algorithm for the case of $\bar{0}$-justified PROPs. As an arbitrary PROP can be made $\bar{0}$-justified via a proper shifting (see Lemma 2.5), we conclude the general case by giving an algorithm for computing such a shift efficiently. We give three different reconstruction algorithms: a randomized algorithm, a deterministic algorithm and a non-adaptive deterministic algorithm, with some deterioration in the running times between the different models.

Let d be a bound on the individual degrees of the PROP in question. We assume that $|\mathbb{F}| > d$. If this is not the case then we allow the algorithm to make queries to the polynomial from an extension field of appropriate size. We note that such a requirement is necessary even for the simple task of univariate polynomial interpolation. Throughout this section we fix a set $W \subseteq \mathbb{F}$ of $d+1$ distinct elements such that $0 \in W$.

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

Let P be a $\bar{0}$-justified PROP with individual degrees bounded by d. In [43] (Theorem 7.4 specialized to $k = 2$) it was shown that P is uniquely determined by its values on the set $\mathcal{A}_6^n(W)$ (recall Definition 2.8). This implies that P can be reconstructed given those values. Yet, in principal, such an "information-theoretic" reconstruction algorithm need not be efficient. In this section we give an efficient deterministic reconstruction algorithm for $\bar{0}$-justified PROPs, which queries the polynomial on inputs from the set $\mathcal{A}_3^n(W)$.

Algorithm 3 Reconstruct $\bar{0}$-justified PROF

Input: Black-box access to a $\bar{0}$-justified PROP P.
Output: PROF Φ such that $\Phi \equiv P$.

```
1: Learn the Preprocessing and $\mathrm{var}_0(P)$
2: Construct the gate-graph of $P$
3: Construct a PROF $\Phi$ by recursively constructing its sub-formulas
```

Lemma 5.1. *There exists an algorithm that given black-box access to a $\bar{0}$-justified PROP*

$$P \in \mathbb{F}[x_1, x_2, \dots, x_n]\,,$$

with individual degrees bounded by d, constructs a (non-degenerate) PROF Φ, such that $\Phi \equiv P$. Furthermore, the algorithm only queries P on the set $\mathcal{A}_3^n(W)$. The running time of the algorithm is $\mathrm{poly}(n,d)$.

A high level description of the algorithm is given in Algorithm 3. We now describe in depth each step of the algorithm.

5.1.1 Learning the preprocessing and $\mathrm{var}_0(P)$

Given access to a PROP $P(\bar{x}) = Q(T(\bar{x}))$, the first step is to compute the standard preprocessing $T(\bar{x})$ and the set $\mathrm{var}_0(P)$. For that purpose, for every $i \in [n]$ we define

$$P_i(x_i) \triangleq P(0, \dots, 0, x_i, 0, \dots, 0)\,.$$

The next observation is immediate in the light of Lemma 3.19. Recall that for every $\bar{0}$-justified polynomial P it holds that $\mathrm{var}_0(P) = \mathrm{var}(P)$.

Observation 5.2. 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 $x_i \in \mathrm{var}_0(P)$.

For the sake of future steps we also compute a witness $\bar{\alpha}$ for P (recall Definition 3.20).

Algorithm 4 Learn the Preprocessing, $\mathrm{var}_0(P)$ and a Witness

Input: A $\bar{0}$-justified $P(\bar{x}) = Q(T(\bar{x}))$ given via black-box access.
Output: A standard preprocessing $T(\bar{x}) = (T_1(x_1), \ldots, T_n(x_n))$, $\mathrm{var}_0(P)$, 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 $\mathrm{var}_0(P)$
```

Lemma 5.3. *Given black-box access to a $\bar{0}$-justified PROP P, Algorithm 4 computes a standard preprocessing $T(\bar{x})$ of P, the set $\mathrm{var}_0(P)$ and a witness $\bar{\alpha}$ for P. The running time of the algorithm is $\mathrm{poly}(n,d)$. In addition, the algorithm does so by querying P on the set $\mathcal{A}_1^n(W)$ alone.*

Proof. First of all note that, as the individual degrees in P_i are bounded by d, the set $\mathcal{A}_1^n(W)$ contains enough points to interpolate P_i. It is easy to compute a standard T_i from P_i. From Observation 5.2, $x_i \in \mathrm{var}_0(P)$ 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 operations can be carried out in time $\mathrm{poly}(n,d)$ by querying P only on inputs from the set $\mathcal{A}_1^n(W)$. □

As P is $\bar{0}$-justified we can set

$$P \triangleq P|_{x_{[n]\setminus \mathrm{var}_0(P)} = \bar{0}_{[n]\setminus \mathrm{var}_0(P)}}$$

and, by renaming variables, assume w. l. o. g. that $\mathrm{var}(P) = \mathrm{var}_0(P) = [n]$. If P is constant or univariate (i. e., $|\mathrm{var}_0(P)| \leq 1$) then we are done. Otherwise, P contains at least one (addition or multiplication) gate, and we continue to the next step of Algorithm 3.

As we have learned the preprocessing, we will, in some sense, be able to "forget" about it and reconstruct P as if it was a (non-preprocessed) ROP. In fact, the witness $\bar{\alpha}$ and Lemma 3.22 allow us to access the backbone ROF of P.

5.1.2 Constructing the gate-graph

In this step we recursively unfold the structure of the backbone ROF of a given PROP. As mentioned above, after the previous step we can treat P as if there was no preprocessing involved (i. e., "assume" that $T_i(x_i) = x_i$). Recall that the gate-graph of a PROP P is defined to be the gate-graph of its backbone ROP.

We will use Lemma 3.6 in order to construct that graph.

Algorithm 5 Construct The Gate-Graph

```
Input: a $\bar{0}$-justified PROP $P(\bar{x})$ given via black-box access, a witness $\bar{\alpha}$ for $P$.
Output: $G_P$.
    For each pair $i \neq j \in [n]$ add $(i,j)$ to $G_P$ iff $\frac{\partial^2 P}{\partial_{\alpha_i} x_i \, \partial_{\alpha_j} x_j}(\bar{0}) \neq 0$.
```

Lemma 5.4. *Given black-box access to a $\bar{0}$-justified PROP P Algorithm 5 runs in time $\mathcal{O}(n^2)$ and constructs G_P—the gate-graph of the P—by making queries to P from the set $\mathcal{A}_2^n(W)$ alone.*

Proof. Let $P(\bar{x}) = Q(T(\bar{x}))$. Since $T(\bar{x})$ is standard, Lemma 3.22 implies that Q is $\bar{0}$-justified. Therefore, by Lemmas 3.6 and 3.11:

$$(i,j) \in G_P \iff \frac{\partial^2 Q}{\partial z_i \partial z_j} \not\equiv 0 \iff \frac{\partial^2 Q}{\partial z_i \partial z_j}(\bar{0}) \neq 0\,.$$

On the other hand, Lemma 3.22 implies that

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

As $T_i(\alpha_i), T_j(\alpha_j) \neq 0$, correctness of the algorithm follows. From the definition of discrete partial derivative (Definition 2.11), 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 four points from $\mathcal{A}_2^n(W)$. □

Having the gate-graph at hand, we can recursively reconstruct the formula gate-by-gate, according to the labelling ($+$ or $\times$) of its top gate. By Lemma 3.23, 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 on the labeling of the top gate. Thus the high level view of step 3 of the reconstruction algorithm is as follows: first we find a (possible) partition of the variables $L \dot{\cup} R = \mathrm{var}(P)$ such that $L = \mathrm{var}(P_1)$ and $R = \mathrm{var}(P_2)$, then we recursively construct formulas $\Phi_i \equiv P_i$ and, finally, we “glue” them together to obtain a formula for P.

5.1.3 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 that follows from the definitions in Section 3.2.

Observation 5.5. Let $Q(\bar{x})$ be a multilinear polynomial and let $L \dot{\cup} R = \mathrm{var}(Q)$ be a non-trivial partition of $\mathrm{var}(Q)$. 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 every $i \in L$ and $j \in R$ it holds that

$$\frac{\partial^2 Q}{\partial x_i \partial x_j} \equiv 0\,.$$

This gives rise to the following algorithm:

Algorithm 6 Top Gate + Case

Input: $\bar{0}$-justified PROP $P(\bar{x})$ given via black-box access.
Output: PROF $\Phi \equiv P$.

```
1: Find a partition $L \dot{\cup} R = \mathrm{var}(P)$ of $G_P$, such that there is no edge between a vertex in $L$ and a vertex
   in $R$. $\{P$ can be regarded as $P(\bar{x}_L, \bar{x}_R) = P_1(\bar{x}_L) + P_2(\bar{x}_R)\}$
2: $\Phi_L(\bar{x}_L) \leftarrow \mathsf{Reconstruct}(P(\bar{x}_L, \bar{0}))$, $\Phi_R(\bar{x}_R) \leftarrow \mathsf{Reconstruct}(P(\bar{0}, \bar{x}_R))$
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 5.6. *Given black-box access to a $\bar{0}$-justified PROP P, computed by a PROF with a top gate labelled as "+," Algorithm 6 constructs a PROF Φ that computes P. The algorithm makes queries to P on inputs from the set $\mathcal{A}_2^n(W)$ and from the set queried during the* Reconstruct *procedure. The running time is* poly(n) *plus the time required by the* Reconstruct *procedure.*

Proof. Let $P(\bar{x}) = Q(T(\bar{x}))$. By Lemma 3.23, $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$ and G_P is disconnected. Hence, we can find a non-trivial partition $L \dot{\cup} R = \mathrm{var}(P)$, to two components that are not connected to each other. By Observation 5.5 we can assume w.l.o.g that $L = \mathrm{var}(P_1)$ and $R = \mathrm{var}(P_2)$, and hence $\Phi_L(\bar{x}_L) = P_1(\bar{x}_L) + P_2(\bar{0})$ and $\Phi_R(\bar{x}_R) = P_1(\bar{0}) + P_2(\bar{x}_R)$. Hence,

$$\Phi_L(\bar{x}_L) + \Phi_R(\bar{x}_R) - P(\bar{0}, \bar{0}) = P(\bar{x}) + P_1(\bar{0}) + P_2(\bar{0}) - P(\bar{0}, \bar{0}) = P(\bar{x}),$$

and so the last step gives the required formula. □

5.1.4 Top gate "×"

The case of a top $\times$ gate is slightly more subtle. Here $P(\bar{x}) = Q(T(\bar{x})) = Q_1(T(\bar{x})) \cdot Q_2(T(\bar{x})) + c$. Although c is a priori unknown, Observation 3.13 implies that a possible partition can be found by considering all j such that Q is (x_1, x_j)-separated mod $\mathbb{F}$. For this purpose we shall use Corollary 2.17. In general there is no efficient deterministic procedure to divide, or even check divisibility of, two multivariate polynomials. Lemma 3.16 suggests that in our setting this can be done by considering a set of three variables at a time. The structural result of Lemma 3.14 allows further simplification of the test. Note that P and Q essentially depend on the same set of variables, thus we can abuse notation and say that $\mathrm{var}(P) = \mathrm{var}(Q)$.

Lemma 5.7. *Given black-box access to a $\bar{0}$-justified PROP P, computed by a PROF with a top gate labelled as "×," Algorithm 7 constructs a PROF Φ that computes P. The running time of the algorithm is* poly(n) *plus the time required by the* Reconstruct *procedure. The algorithm queries P on inputs from the set $\mathcal{A}_3^n(W)$ and on the inputs queried by the* Reconstruct *procedure.*

Proof. Note that all the arithmetic operations in the algorithms are on polynomials of at most three variables. Thus, those operation can be carried out efficiently by making queries to P on inputs from the set $\mathcal{A}_3^n(W)$. We now prove that

$$L = \left\{ j \in [n] \;\middle|\; Q \text{ is } \textbf{not } (x_1, x_j)\text{-separated mod } \mathbb{F} \right\} \cup \{1\}.$$

The rest of the proof is similar to the proof of Lemma 5.6.

Algorithm 7 Top Gate $\times$ Case

Input: $\bar{0}$-justified PROP $P(\bar{x})$, given via black-box access, and a witness $\bar{\alpha}$.
Output: PROF $\Phi \equiv P$.

```
Finding $L$:
1: $L \leftarrow \{1\}$
2: for all $j \neq k \in \mathrm{var}(P) \setminus \{1\}$ do
3:     $P_{1,j,k} \leftarrow$ Interpolate $P|_{x_{[n]\setminus\{1,j,k\}}=\bar{0}_{[n]\setminus\{1,j,k\}}}$ {viewed as a three-variate polynomial}
4:     if $\frac{\partial^2 P_{1,j,k}}{\partial_{\alpha_1} x_1 \partial_{\alpha_j} x_j} \equiv 0$ OR $\deg\left(\Delta_{1j}^{\bar{\alpha}}(P_{1,j,k})\right) \neq \deg\left(\frac{\partial^2 P_{1,j,k}}{\partial_{\alpha_1} x_1 \partial_{\alpha_j} x_j}\right)$ then
5:         $L \leftarrow L \cup \{j\}$
Reconstructing the formula:
1: $R \leftarrow \mathrm{var}(P) \setminus L$ {$P$ can be regarded as $P(\bar{x}_L, \bar{x}_R) = P_1(\bar{x}_L) \cdot P_2(\bar{x}_R) + c$}
2: Chose $i \in L, j \in R$
3: $c \leftarrow P(\bar{0},\bar{0}) - \left(\frac{\partial P}{\partial_{\alpha_i} x_i}(\bar{0}) \cdot \frac{\partial P}{\partial_{\alpha_j} x_j}(\bar{0})\right) \div \left(\frac{\partial^2 P}{\partial_{\alpha_i} x_i \partial_{\alpha_j} x_j}(\bar{0})\right)$
4: $\Phi_L(\bar{x}_L) \leftarrow \mathsf{Reconstruct}(P(\bar{x}_L,\bar{0}) - c)$, $\Phi_R(\bar{x}_R) \leftarrow \mathsf{Reconstruct}(P(\bar{0},\bar{x}_R) - c)$
5: $\Phi(\bar{x}_L,\bar{x}_R) \leftarrow \frac{1}{P(\bar{0},\bar{0}) - c} \cdot \Phi_L(\bar{x}_L) \times \Phi_R(\bar{x}_R) + c$
```

($\subseteq$) Let $j \in L \setminus \{1\}$. Then there exists $k \in \mathrm{var}(P) \setminus \{1, j\}$ such that

$$\frac{\partial^2 P_{1,j,k}}{\partial_{\alpha_1} x_1 \partial_{\alpha_j} x_j} \equiv 0 \text{ or } \deg\left(\Delta_{1j}^{\bar{\alpha}}(P_{1,j,k})\right) \neq \deg\left(\frac{\partial^2 P_{1,j,k}}{\partial_{\alpha_1} x_1 \partial_{\alpha_j} x_j}\right)$$

for $P_{1,j,k} = P|_{x_{[n]\setminus\{1,j,k\}}=\bar{0}_{[n]\setminus\{1,j,k\}}}$. In the former case, by Lemma 3.11,

$$\left.\frac{\partial^2 Q}{\partial z_1 \partial z_j}\right|_{\bar{z}=T(\bar{x})} \cdot T_1(\alpha_1) \cdot T_j(\alpha_j) = \frac{\partial^2 P}{\partial_{\alpha_1} x_1 \partial_{\alpha_j} x_j} \equiv 0$$

and hence Q is **not** (x_1, x_j)-separated mod $\mathbb{F}$. The latter case is equivalent to

$$\deg\left(\Delta_{1j}(Q(x_1, x_j, x_k, \bar{0}))\right) \neq \deg\left(\frac{\partial^2 Q(x_k, \bar{0})}{\partial x_i \partial x_j}\right)$$

which implies that

$$\Delta_{1j} Q \neq c \cdot \frac{\partial^2 Q}{\partial x_1 \partial x_j}$$

for any $c \in \mathbb{F}$. Thus, by Corollary 2.17, Q is (again) **not** (x_1, x_j)-separated mod $\mathbb{F}$.

$(\supseteq)$ Suppose Q is **not** (x_1, x_j)-separated mod $\mathbb{F}$. Lemma 3.16 implies that there exists $k \in \text{var}(P) \setminus \{1, j\}$ such that

$$Q|_{x_{[n]\setminus\{1,j,k\}}=\bar{0}_{[n]\setminus\{1,j,k\}}}$$

is **not** (x_i, x_j)-separated mod $\mathbb{F}$. Now, if

$$\frac{\partial^2 Q}{\partial z_1 \partial z_j} \equiv 0$$

then $j \in L$; otherwise, by Lemma 3.14,

$$\Delta_{1j}\left(Q|_{x_{[n]\setminus\{1,j,k\}}=\bar{0}_{[n]\setminus\{1,j,k\}}}\right) = g(x_k) \cdot \frac{\partial^2(Q(x_1, x_j, x_k, \bar{0}))}{\partial x_1 \partial x_j},$$

where $g(x_k)$ is a univariate polynomial. By Corollary 2.17 $g(x_k)$ must be non-constant (since we assumed that Q is not (x_1, x_j)-separated mod $\mathbb{F}$) and hence

$$\deg\left(\Delta_{1j}(Q(x_1, x_j, x_k, \bar{0}))\right) \neq \deg\left(\frac{\partial^2 Q(x_k, \bar{0})}{\partial x_i \partial x_j}\right),$$

implying (again) that $j \in L$.

The rest of the algorithm follows the same rationale as before. We first find the value of the constant c and then recursively construct the left and right subformulas. We note that since P is $\bar{0}$-justified, we can perform the division in the last step. □

5.1.5 Conclusion

We can now prove Lemma 5.1.

Proof of Lemma 5.1. The proof is by induction on $|\text{var}_0(P)| = |\text{var}(P)|$. By Lemma 5.3 we can learn the set $\text{var}_0(P)$ and handle the case $|\text{var}_0(P)| \leq 1$. Now, assume $|\text{var}_0(P)| \geq 2$. This implies that any PROF computing P must have at least one gate. Lemma 5.4 allows us to construct the gate-graph from which we can learn the labeling of the top gate (Lemma 3.4, Property 3). From this point on Lemmas 5.6 and 5.7 ensure the correctness of construction. We note that in each step we query P only on inputs from the set $\mathcal{A}_3^n(W)$. The claim regarding the running time follows from previous lemmas. □

The next lemma shows that Algorithm 3 gives useful information also when applied to PROFs that are not $\bar{0}$-justified.

Lemma 5.8. *Let P a PROP, with individual degrees bounded by d, given via black-box access. Algorithm 3, when given P as input, constructs a (non-degenerate) PROF Φ satisfying $\Phi \equiv P|_{x_{[n]\setminus \text{var}_0(P)}=\bar{0}_{[n]\setminus \text{var}_0(P)}}$.*

Proof. Denote

$$P' \triangleq P|_{x_{[n]\setminus \text{var}_0(P)}=\bar{0}_{[n]\setminus \text{var}_0(P)}}.$$

Observe that $\text{var}(P') = \text{var}_0(P) = \text{var}_0(P')$ hence P' is a $\bar{0}$-justified PROP. To complete the proof note that the algorithm runs in the same way when given either P or P' as input. □

5.2 Reconstruction of a general PROF

So far we gave a reconstruction algorithm for $\bar{0}$-justified PROPs. By combining our algorithm with Lemma 2.5 we obtain an algorithm for reconstructing arbitrary PROPs, assuming that we know a justifying assignment for the black-box polynomial.

Theorem 5.9. *Let $P \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a PROP, with individual degrees bounded by d, given via 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 Φ such that $\Phi \equiv P$ by querying P on inputs from the set $\bar{a} + \mathcal{A}_3^n(W)$. The running time of the algorithm is* $\operatorname{poly}(n,d)$.

Proof. Invoke the algorithm described in Lemma 5.1 on $P'(\bar{x}) \triangleq P(\bar{x}+\bar{a})$. By Lemma 2.5 P' is a $\bar{0}$-justified PROP. Therefore, the algorithm will return a PROF Φ such that $\Phi(\bar{x}) \equiv P'(\bar{x}) \triangleq P(\bar{x}+\bar{a})$. The algorithm returns the formula $\Phi(\bar{x}-\bar{a})$. □

Consequently, to extend our algorithm to handle a general PROP P, we first need to find a justifying assignment $\bar{a}$ for P and then apply Theorem 5.9. We give a randomized algorithm, a deterministic algorithm and a non-adaptive deterministic algorithm for computing a justifying assignment, thus obtaining a reconstruction algorithm for general PROPs.

5.2.1 Randomized PROF reconstruction

The simplest way to get a justifying assignment is to pick one at random. This idea was used in the previous works on ROF reconstruction [18, 10]. We give it here for completeness. We first state the Schwartz-Zippel lemma:

Lemma 5.10 ([46, 38])**.** *Let $P \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a non-zero polynomial with individual degrees bounded by d. Then,*

$$\Pr_{\bar{a}\in\mathbb{F}^n}[P(\bar{a})=0] \le \frac{nd}{|\mathbb{F}|}.$$

Since a justifying assignment is simply a common non-zero assignment of several polynomials, we get the following corollary:

Corollary 5.11. *Let $n,d \ge 1$. Assume that $|\mathbb{F}| > 4n^2d$. Let $P \in \mathbb{F}[x_1, x_2, \dots, x_n]$ be a PROP with individual degrees bounded by d. Then,*

$$\Pr_{\bar{a}\in\mathbb{F}^n}[\bar{a} \textit{ is a justifying assignment of } P] \ge \frac{3}{4}.$$

Proof. Let $(Q(\bar{z}), T(\bar{x}))$ be a standard decomposition of P and let $\bar{\alpha}$ be a corresponding witness for P. By Lemma 3.22:

$$\Pr_{\bar{a}\in\mathbb{F}^n}[\bar{a} \text{ is not a justifying assignment of } P] \le \Pr_{\bar{a}\in\mathbb{F}^n}\left[\exists i \text{ s.t. } \frac{\partial P}{\partial_{\alpha_i} x_i} \not\equiv 0 \text{ but } \frac{\partial P}{\partial_{\alpha_i} x_i}(\bar{a}) = 0\right]$$
$$\le n \cdot \frac{nd}{|\mathbb{F}|} \le \frac{1}{4}.$$
□

In fact, the statement can be shown to hold for arbitrary polynomials. We can now prove Theorem 1.3.

Proof of Theorem 1.3. The algorithm picks an assignment $\bar{a} \in \mathbb{F}^n$ at random and then runs the algorithm described in Theorem 5.9. The claim follows from Corollary 5.11 and Theorem 5.9. Note that randomization is required only for picking the justifying assignment $\bar{a}$. □

5.2.2 Deterministic PROF reconstruction

We use a result from [43] to obtain a deterministic reconstruction algorithm.

Lemma 5.12 (Theorem 1 of [43][10]). *Let $n,d \geq 1$. Assume $|\mathbb{F}| > n^2 d$. Then there exists a black-box PIT algorithm for PROPs over $\mathbb{F}[x_1, x_2, \dots, x_n]$ with individual degrees bounded by d, that runs in time $(nd)^{\mathcal{O}(\log n)}$.*

We are ready to prove Theorem 1.1.

Proof of Theorem 1.1. Given a PROP P invoke the algorithm in Lemma 4.1 to obtain a justifying assignment $\bar{a}$ for P. Recall that Lemma 4.1 requires a PIT algorithm for a class that contains all partial derivatives of PROPs. By Lemma 3.22, PROPs are closed under taking partial derivatives. Hence, the PIT algorithm given in Lemma 5.12 satisfies the requirements of Lemma 4.1. The running time of the algorithm $(nd)^{\mathcal{O}(\log n)}$.

Now, apply Theorem 5.9. The reconstruction will take $\mathrm{poly}(n,d)$ time. Consequently, the whole execution takes $(nd)^{\mathcal{O}(\log n)}$ time. □

5.2.3 Non-adaptive PROF reconstruction

The reconstruction algorithm described by Algorithm 3 in non-adaptive as it queries the PROP on the points in the set $\mathcal{A}_3^n(W)$ regardless of the results of previous queries. Consequently, the randomized version of the algorithm is non-adaptive as well. On the other hand, the algorithm given in Lemma 4.1 is in fact adaptive: its subsequent queries to P depend on the result of the previous queries (for more details see [43, Section 4.1]). This implies that the deterministic algorithm given in Theorem 1.1 is adaptive.

To obtain a non-adaptive algorithm we proceed as follows: instead of finding a (single) justifying assignment and using it for the reconstruction, we will attempt to reconstruct P using every assignment in a *justifying set* for PROPs (see Lemma 4.2).[11] Let S be the set of all resulting PROFs. Following Lemma 5.8, the PROF with the largest number of variables is the one we are looking for.

Proof of Theorem 1.2. The algorithm follows the steps specified in Algorithm 8.

Running time: By Lemma 4.2 $\mathcal{J}_d$ can be computed in time $(nd)^{\mathcal{O}(\log n)}$. Other steps can be carried out in polynomial time. Hence, the total running time is $(nd)^{\mathcal{O}(\log n)}$.

[10]The size of the field does not appear in the statement of Theorem 1 of [43], but this is what the proof gives.

[11]Recall that a justifying set contains a justifying assignment for every PROP.

Algorithm 8 Non-Adaptive PROF Reconstruction

Input: A black-box access to a PROP P with individual degrees bounded by d.
Output: PROF Φ such that $\Phi \equiv P$.

```
1: $S \leftarrow \emptyset$
2: Compute the set $\mathfrak{J}_d$ {Using Lemma 4.2}
3: **for all** $\bar{a} \in \mathfrak{J}_d$ **do**
4:     $\Phi_{\bar{a}} \leftarrow$ Reconstruct $P$ {Using Theorem 5.9}
5:     $S \leftarrow S \cup \Phi_{\bar{a}}$
6: Find $\Phi' \in S$ with the maximal var$(\Phi')$, denote it with $\Phi_M$
7: **return** $\Phi_M$
```

Correctness of the algorithm: First of all, notice that since Algorithm 3 is non-adaptive so is Algorithm 8. By Lemma 4.2, $\mathfrak{J}_d$ contains a justifying assignment $\bar{a}^*$ of P, hence S must contain $\Phi_{\bar{a}^*}$—the result of reconstructing P using $\bar{a}^*$ as a justifying assignment input. By Theorem 5.9 $\Phi_{\bar{a}^*} \equiv P$. We now argue that $\Phi_M \equiv \Phi_{\bar{a}^*} \equiv P$. By Lemma 5.8 $\text{var}(\Phi_M) \subseteq \text{var}(P)$. On the other hand, from the definition of Φ_M, it holds that $|\text{var}(\Phi_M)| \geq |\text{var}(\Phi_{\bar{a}^*})| = |\text{var}(P)|$. This implies that $\text{var}(\Phi_M) = \text{var}(P)$. From Lemma 5.8 we conclude that the assignment used to construct Φ_M is a justifying assignment as well. Thus, $\Phi_M \equiv P$. □

6 Read-once testing

In this section we study the relation between the PIT problem and the ROT problem and prove Theorem 1.5. We first present a generic scheme that can be used to strengthen efficient deterministic PIT algorithms to yield efficient deterministic ROT algorithms. Then we use known schemes to obtain ROT algorithms for models for which PIT algorithms are known.[12]

6.1 Generic scheme

As was mentioned in Section 1.3, the main idea behind the scheme is "Doveryai, no Proveryai" (trust, but verify): Given a circuit we run a PROP reconstructing algorithm on the circuit (based on Theorem 5.9). If the algorithm encounters an error or is unable to run correctly, then we conclude that the circuit 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 construct a PROF. The problem is that we do not have any guarantee regarding the correctness of our assumption (that the circuit computes a PROP) and hence no guarantee that the PROF constructed indeed computes the same polynomial as the circuit. Moreover, for any circuit C that computes a PROP, there exist many circuits (computing different polynomials) "aliasing" C. That is, an execution of our reconstruction algorithm on each such circuit $C' \not\equiv C$ will succeed and yield a PROF Φ such that $\Phi \equiv C \neq C'$.[13] Consequently, for ROT we need to *verify* the correctness of the output. The verification procedure is the technical core of our ROT algorithm, and is

[12]In fact, in several cases we give "tailored" algorithms that are somewhat more efficient than the generic algorithm.
[13]C' just needs to agree with C on the set of queries made by the reconstruction algorithm.

given in Algorithm 10. Algorithm 9 gives the generic scheme for ROT. The algorithm works both in the black-box and in the white-box models, depending on the PIT algorithm at hand.

Algorithm 9 Generic ROT Scheme

Input: (Black-box access to a) Circuit C that belongs to $\mathcal{C}$.

Output: A PROF Φ such that $\Phi \equiv C$ if C computes a PROP and "reject" otherwise.

```
1: Compute a justifying assignment $\bar{a}$ using Lemma 4.1.
2: Reconstruct a PROF $\Phi$ from $C$ using Theorem 5.9.
3: Verify that $\Phi \equiv C$ using Algorithm 10 (given in Subsection 6.2).
```

In Section 6.3 we give the proof of Theorem 1.5, that basically analyzes the running time of Algorithm 9, given Lemma 4.1 and Theorems 5.9 and 6.2. Then, in Section 7, we go over some circuit classes that have efficient PIT algorithms and give the corresponding ROT algorithms.

6.2 Read-once verification

Read-Once Verification is testing whether a given circuit C, from a certain circuit class $\mathcal{C}$, and a given PROF Φ compute the same polynomial. This is somewhat a harder problem than PIT since it requires determining the equivalence of polynomials computed by circuits from two different circuit classes. We shall work under the assumptions that $\mathcal{C}$ has a PIT algorithm and that the PROF Φ 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 assumed 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. The idea behind the algorithm is to recursively ensure that every gate v of Φ computes the same polynomial as a corresponding restriction of C. To give a rough sketch, we first find a justifying assignment $\bar{a}$ for Φ. Then we consider v, the root of Φ. It has two subformulas. Denote $\Phi = \alpha \cdot (\Phi_{v_1} \text{ op } \Phi_{v_2}) + \beta$, where Φ_{v_i} is the PROF computed at v_i and op is either $+$ or $\times$. Assume that the variables of v_i are S_i (S_1 and S_2 are disjoint). Consider the circuit C_1 that results from C after substituting the corresponding values from $\bar{a}$ to the variables in S_2. Similarly we define C_2, and the PROFs Φ_1 and Φ_2.[14] We now recursively verify that $C_i \equiv \Phi_i$. The only thing left to do 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 op is either $+$ or $\times$. 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 7.2, 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 deterministic 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 2.18.

Definition 6.1. 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.

[14] 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.

Note that for most circuit classes that have efficient deterministic PIT algorithms, there exists an efficient deterministic PIT algorithm for a corresponding verifying class.

Algorithm 10 solved the verification problem following the outline described above. It uses notation from Definition 4.3.

Algorithm 10 Verify (C, Φ)

```
Input:  Circuit $C$ from a circuit class $\mathcal{C}$, a PROF $\Phi$,
        Access to a PIT algorithm for $\mathcal{C}_V$,
        and a justifying assignment $\bar{a}$ for $\Phi$ and $C$.
Output: "accept" if $C \equiv \Phi$ and "reject" otherwise.
 1: if $v$.Type = IN then {$\Phi$ is a univariate polynomial}
 2:     Check that $C \equiv \Phi$ {As two univariate polynomials of degree at most $d$}
    {Internal Gate}
 3: $L \leftarrow \mathrm{var}(\Phi_{v.\mathrm{Left}})$
 4: $R \leftarrow \mathrm{var}(\Phi_{v.\mathrm{Right}})$
    {Multiplication Gate}
 5: if $v$.Type = $\times$ then
 6:     $C_L \leftarrow (C|_{x_R=\bar{a}_R} - v.\beta) \,/\, (v.\alpha \cdot \Phi_{v.\mathrm{Right}}(\bar{a}))$
 7:     $C_R \leftarrow (C|_{x_L=\bar{a}_L} - v.\beta) \,/\, (v.\alpha \cdot \Phi_{v.\mathrm{Left}}(\bar{a}))$
 8:     Verify($C_L, \Phi_{v.\mathrm{Left}}$) and Verify($C_R, \Phi_{v.\mathrm{Right}}$) {Recursively}
 9:     Check that $C \equiv v.\alpha \cdot (C_L \times C_R) + v.\beta$ {Using the PIT algorithm for $\mathcal{C}_V$}
    {Addition Gate}
10: if $v$.Type = + then
11:     $C_L \leftarrow (C|_{x_R=\bar{a}_R} - v.\beta) \,/\, v.\alpha - \Phi_{v.\mathrm{Right}}(\bar{a})$
12:     $C_R \leftarrow (C|_{x_L=\bar{a}_L} - v.\beta) \,/\, v.\alpha - \Phi_{v.\mathrm{Left}}(\bar{a})$
13:     Verify($C_L, \Phi_{v.\mathrm{Left}}$) and Verify($C_R, \Phi_{v.\mathrm{Right}}$) {Recursively}
14:     Check that $C \equiv v.\alpha \cdot (C_L + C_R) + v.\beta$ {Using the PIT algorithm for $\mathcal{C}_V$}
    {Everything is OK}
15: If any of the steps failed than reject, otherwise accept.
```

Theorem 6.2. *Let C be an n-variate circuit from a circuit class $\mathcal{C}$, that computes a polynomial with individual degrees at most d, and Φ a PROF, such that* $\mathrm{var}(C) = \mathrm{var}(\Phi)$. *Let $\mathcal{C}_V$ be a verifying class of $\mathcal{C}$ and $\bar{a}$ a justifying assignment for Φ and C. Then, given C, Φ and $\bar{a}$ as input, Algorithm 10 runs in time $\mathcal{O}(nd + n \cdot t)$, where t is the cost of the given PIT algorithm for $\mathcal{C}_V$, and accepts if and only if $C \equiv \Phi$.*

Proof. We start by analyzing the running time.

Running time: As described above, the algorithm preforms a traversal over the tree of Φ. The PIT algorithm of $\mathcal{C}_V$ is invoked once for every internal gate. For each input gate we interpolate a univariate polynomial using $d+1$ queries. 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," namely, that in lines 9 and 14 we indeed have the correct input to the relevant PIT algorithm. We make the following observation: At each step of the algorithm it holds that

1. $C, C_L, C_R \in \mathcal{L}(\mathcal{C})$ (when C_L and C_R are defined). This follows (recursively) from the definitions of C_L and C_R (both are obtained by applying a linear function to a restriction of C).

2. $\mathrm{var}(C) = \mathrm{var}(\Phi)$. By induction: For the base case we are guaranteed that $\mathrm{var}(C) = \mathrm{var}(\Phi)$. For the induction step, consider the definition of the sets L and R. As $\bar{a}$ is a justifying assignment of C we have that
$$\mathrm{var}(C_L) = \mathrm{var}(C) \setminus R = \mathrm{var}(\Phi) \setminus \mathrm{var}(\Phi_{v.\mathrm{Right}}) = \mathrm{var}(\Phi_{v.\mathrm{Left}})$$
and similarly $\mathrm{var}(C_R) = \mathrm{var}(\Phi_{v.\mathrm{Right}})$. It is only left to notice that this is in line with the recursive invocation of the algorithm.

3. C_L and C_R are variable-disjoint. Indeed,
$$\mathrm{var}(C_L) \cap \mathrm{var}(C_R) = \mathrm{var}(\Phi_{v.\mathrm{Left}}) \cap \mathrm{var}(\Phi_{v.\mathrm{Right}}) = \emptyset\,.$$

From Observations 1 and 3 it follows that

$$C - v.\alpha \cdot (C_L \times C_R) - v.\beta \in \mathcal{C}_V \qquad \text{and} \qquad C - v.\alpha \cdot (C_L + C_R) - v.\beta \in \mathcal{C}_V\,.$$

Hence, we can execute Lines 9 and 14 using the PIT algorithm of $\mathcal{C}_V$.

Correctness: The correctness of the algorithm can be proved by a simple induction. We need to show that the algorithm accepts iff $C \equiv \Phi$. The base case (i. e., Φ is a univariate polynomial of degree at most d) is trivial. The inductive step follows from Lemmas 6.3 and 6.4 given below (and the correctness of the PIT algorithm of $\mathcal{C}_V$). This completes the proof. □

We first handle the case of a multiplication gate.

Lemma 6.3. *Let $C(\bar{x},\bar{y})$ and $\Phi(\bar{x},\bar{y}) = \alpha \cdot \Phi_\ell(\bar{x}) \times \Phi_r(\bar{y}) + \beta$ be two polynomials and let $(\bar{x}_0,\bar{y}_0)$ be a justifying assignment of Φ. Then $C(\bar{x},\bar{y}) \equiv \Phi(\bar{x},\bar{y})$ if and only if the following conditions hold:*

1. $\Phi_\ell(\bar{x}) = \dfrac{C(\bar{x},\bar{y}_0) - \beta}{\alpha \cdot \Phi_r(\bar{y}_0)}$,

2. $\Phi_r(\bar{y}) = \dfrac{C(\bar{x}_0,\bar{y}) - \beta}{\alpha \cdot \Phi_\ell(\bar{x}_0)}$,

3. $C(\bar{x},\bar{y}) = \alpha \cdot \dfrac{C(\bar{x},\bar{y}_0) - \beta}{\alpha \cdot \Phi_r(\bar{y}_0)} \times \dfrac{C(\bar{x}_0,\bar{y}) - \beta}{\alpha \cdot \Phi_\ell(\bar{x}_0)} + \beta$.

Notice that the conditions are well defined since $(\bar{x}_0,\bar{y}_0)$ is a justifying assignment of Φ and hence $\Phi_\ell(\bar{x}_0), \Phi_r(\bar{y}_0) \neq 0$.

We next handle the case of an addition gate.

Lemma 6.4. *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 Φ. Then $C(\bar{x},\bar{y}) \equiv \Phi(\bar{x},\bar{y})$ if and only if the following conditions hold:*

1. $\Phi_\ell(\bar{x}) = \dfrac{C(\bar{x},\bar{y}_0)-\beta}{\alpha} - \Phi_r(\bar{y}_0)$, *or*
2. $\Phi_r(\bar{y}) = \dfrac{C(\bar{x}_0,\bar{y})-\beta}{\alpha} - \Phi_\ell(\bar{x}_0)$, *or*
3. $C(\bar{x},\bar{y}) = \alpha \cdot \left(\dfrac{C(\bar{x},\bar{y}_0)-\beta}{\alpha} - \Phi_r(\bar{y}_0) + \dfrac{C(\bar{x}_0,\bar{y})-\beta}{\alpha} - \Phi_\ell(\bar{x}_0)\right) + \beta.$

Proof. Both proofs follow from straightforward manipulations. □

6.3 Proof of Theorem 1.5

Proof of Theorem 1.5. We show that given $\mathcal{C}$ and $\mathcal{C}_V$, satisfying the assumptions, we can successfully preform each step of the "generic ROT scheme" described in Algorithm 9.

1. **Computing a Justifying Assignment**
   As $\partial\mathcal{C} \subseteq \mathcal{C}_V$ we can compute a justifying assignment $\bar{a}$ using Lemma 4.1.
   The running time is $\mathcal{O}(n^3 d \cdot T(n,d,s))$.

2. **Reconstructing a PROF Φ from C**
   Given a justifying assignment $\bar{a}$ we can reconstruct Φ from C using Theorem 5.9.
   The running time is $\mathrm{poly}(n,d,s)$.

3. **Verifying that $\Phi \equiv C$**
   Notice that $\mathcal{C}_V$ satisfies the conditions of Definition 6.1 and hence is a verifying class of $\mathcal{C}$. We can thus invoke the verification procedure described in Algorithm 10.
   The running time is $\mathcal{O}(nd + n \cdot T(n,d,s))$.

The total running time is $\mathrm{poly}(n,s,d,T(n,d,s))$. □

In the next section we use Theorem 1.5 and the generic ROT scheme of Algorithm 9 in order to get ROT algorithms for several restricted models for which PIT is known. Theorem 1.5 requires a PIT algorithm for a class slightly larger than the class for which we design the ROT algorithm. However, we note that basically any "reasonable" PIT algorithm yields a PIT algorithm for that larger class, and hence implies a ROT algorithm. For example, an immediate consequence of Theorem 1.5 is that an efficient deterministic PIT algorithm for multilinear circuits (either in the black-box or the white-box model) implies an efficient deterministic ROT algorithm for multilinear circuits.

7 ROT for specific models

In this section we prove Theorems 1.6, 1.7, 1.8 and 1.9 by applying the ROT scheme of Section 6 (Algorithm 9). We start with the case of sparse polynomials (Theorem 1.7).

7.1 Sparse polynomials

An *m-sparse polynomial* is a polynomial with at most m (non-zero) monomials. Equivalently, it is a polynomial computed by a depth-2 $\Sigma\Pi$ circuit of size m. Sparse polynomials received a lot of attention (see, e. g., [7, 29, 31]) and several polynomial time algorithms for both reconstruction and PIT were given, over sufficiently large fields. We now show how to solve the ROT problem for sparse polynomials. Note that in [10] 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 [10] that refers to [30]). We present another proof here that gives a stronger result (thus proving Theorem 1.7). We denote by $\Sigma\Pi(m,d)$ the class of m-sparse polynomials, in n variables, of degree d.[15] In order to apply Algorithm 9 on $\Sigma\Pi(m,d)$ we shall need a PIT algorithm for $\Sigma\Pi(\text{poly}(m),\text{poly}(d))$. Several such results are known, see [29] and references within.

Lemma 7.1 ([29])**.** *Let $n,m,d \geq 1$. Assume $|\mathbb{F}| > (nd)^6$. There is a deterministic black-box PIT algorithm for $\Sigma\Pi(m,d)$ that runs in time* $\text{poly}(m,d)$.

Note that $\Sigma\Pi(m^2+4m+3,2d)$ is a verifying class for $\Sigma\Pi(m,d)$ (recall Definition 6.1). Hence, we can use the above lemma in conjunction with Algorithm 9 to prove Theorem 1.7. We now give an alternative proof of the theorem that follows the same lines as the scheme of Algorithm 9 albeit using a different verification procedure.

Proof of Theorem 1.7. Let $P \in \Sigma\Pi(m,d)$ be a sparse polynomial given via black-box access. We follow the scheme outlined in Algorithm 9 noting that $\partial\Sigma\Pi(m,d) \subseteq \Sigma\Pi(m,d)$. We first run the algorithms suggested by Lemma 4.1 and Theorem 5.9 to obtain a candidate PROF Φ. Of course, should any of the algorithms fail, we reject ("not PROF"). We now wish to verify that $\Phi \equiv P$. Originally, we used Algorithm 10, but here we use a different scheme instead. We first compute the degree of Φ (denoted by D) by a simple traversal, and then run Algorithm 2 to count the number of monomials in the PROP computed by Φ (denoted by M). Now, if $M > m$ or $D > d$ then, obviously, $\Phi \not\equiv P$. 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 7.1 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 10 when given access to a PIT algorithm for the class $\Sigma\Pi(m^2+4m+3,2d)$). $\square$

7.2 Depth-3 circuits

A depth-3 $\Sigma\Pi\Sigma(k)$ circuit C 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;

$$L_{ij}(\bar{x}) = \sum_{t=1}^{n} a_{ij}^{t} x_t + a_{ij}^{0}$$

[15] As the number of variables is always n we do not mention it.

with $a^t_{ij} \in \mathbb{F}$. We denote by $\Sigma\Pi\Sigma(k,d)$ a $\Sigma\Pi\Sigma(k)$ in which each F_i has degree at most d (i. e., $d_i \le d$). Those circuits were a subject of a long line of research [12, 27, 26, 41, 5, 25, 35, 36, 37] yielding several efficient deterministic PIT algorithms. The state of the art black-box PIT algorithm for this circuit class was given in [36].

Lemma 7.2 ([36])**.** *Let $n,k,d \ge 1$. Assume $|\mathbb{F}| > dnk^2$. There is a deterministic black-box PIT algorithm for n-variate polynomials computed by $\Sigma\Pi\Sigma(k,d)$ circuits that runs in time* $\mathrm{poly}(n)\cdot d^{\mathcal{O}(k)}$.

We now give the proof of Theorem 1.8.

Proof of Theorem 1.8. 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 9 in conjunction with Lemma 7.2. Theorem 1.5 guarantees the correctness. The running time is $\mathrm{poly}(n)\cdot d^{\mathcal{O}(k^2)}$. □

7.3 Multilinear depth-4 circuits

A depth-4 $\Sigma\Pi\Sigma\Pi(k)$ circuit C has four layers of alternating Σ and Π gates (the top Σ 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 polynomials computed by a $\Sigma\Pi$ circuit (i. e., sparse polynomials). In other words, the P_{ij}s are being computed by the last two $\Sigma\Pi$ layers of the circuit and are the inputs to the Π gates at the second level. A *multilinear* $\Sigma\Pi\Sigma\Pi(k)$ circuit is a $\Sigma\Pi\Sigma\Pi(k)$ circuit in which each multiplication gate F_i computes a multilinear polynomial. PIT algorithms were given for restricted classes of depth-4 circuits in [34, 41, 5, 3, 2, 33, 23]. The state-of-the-art PIT algorithm for multilinear depth-4 circuits was given in [33].

Lemma 7.3 ([33])**.** *Let $n,k,s \ge 1$. Assume $|\mathbb{F}| > n^2$. There is a deterministic black-box PIT algorithm for n-variate polynomials, computed by multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits, of size s, that runs in time $(ns)^{\mathcal{O}(k^3)}$.*

We can now prove Theorem 1.9.

Proof of Theorem 1.9. First, observe that the class of multilinear $\Sigma\Pi\Sigma\Pi(k^2+4k+3)$ circuits of size $4s+4$ is a verifying class for multilinear $\Sigma\Pi\Sigma\Pi(k)$ circuits of size s. Hence, we can invoke Algorithm 9 in conjunction with Lemma 7.3. Theorem 1.5 guarantees correctness. The running time is $(ns)^{\mathcal{O}(k^6)}$. □

7.4 Multilinear read-k formulas

Read-k formulas are formulas in which each variable appears at most k times. The study of this class of formulas was initiated in [40, 41] where sums of read-once formulas were studied. Efficient deterministic PIT algorithms were given for multilinear [3] and bounded depth [2] read-k formulas. Here we only consider multilinear read-k formulas, the bounded depth case is similar.

Lemma 7.4 ([3]). *Let $n,k \geq 1$ and assume $|\mathbb{F}| > n^2$. Let F be a multilinear read-k formula over $\mathbb{F}[x_1,x_2,\ldots,x_n]$. Then:*

- *There is a deterministic algorithm that given F explicitly decides if $F \equiv 0$ in time $n^{k^{\mathcal{O}(k)}}$.*
- *There is a deterministic algorithm that given black-box access to F decides if $F \equiv 0$ in time $n^{k^{\mathcal{O}(k)}+\mathcal{O}(k\log n)}$.*

We now show a generalization of those PIT algorithms to ROT algorithms, thus proving Theorem 1.6.

Proof of Theorem 1.6. Let F be a multilinear read-k formula. We first use Lemma 4.1 to compute a justifying assignment for F. It is easy to see that this circuit class is closed under partial derivatives, thus we can use the PIT algorithms from Lemma 7.4.

Next, we construct a candidate ROF Φ by applying Theorem 5.9. To complete the procedure, we need to verify that $\Phi = F$ or, in other words, that $F - \Phi \equiv 0$. This is a case of a PIT of a multilinear read-$(k+1)$ formula, hence we can use Lemma 7.4 again to obtain the result. □

8 Acknowledgments

The authors would like to thank Lisa Hellerstein [20] for sharing the containment of [30] with us. We thank Laci Babai and the anonymous referees for useful comments that greatly improved the presentation of our results.

A Proof of Lemma 3.10

In this section we give the proof of Lemma 3.10. To ease the reading we repeat the statement of the lemma.

Lemma 3.10. *A partial derivative of a ROP is a ROP. Moreover, a partial derivative of a $\bar{0}$-justified ROP is also a $\bar{0}$-justified ROP.*

The following lemma will be useful.

Lemma A.1. *Let $P(x_1,x_2,\ldots,x_n)$ be a ROP such that $x_i,x_j \in \mathrm{var}(P)$. Set $I = \mathrm{var}(P) \setminus \{x_i,x_j\}$. Assume that*

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0 \qquad \textit{and} \qquad \left.\frac{\partial^2 P}{\partial x_i \partial x_j}\right|_{x_I=\bar{0}} \equiv 0\,.$$

Then,

$$\left.\frac{\partial P}{\partial x_i}\right|_{x_I=\bar{0}} \equiv 0 \qquad \textit{or} \qquad \left.\frac{\partial P}{\partial x_j}\right|_{x_I=\bar{0}} \equiv 0\,.$$

Proof. Let Φ be a ROF computing P and $v = \mathrm{fcg}\{x_j,x_i\}$. As

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0\,,$$

it follows that v is a multiplication gate. Let $P_v(\bar{x})$ be the polynomial computed by v in Φ. It is not hard to see that there exist polynomials $A(\bar{x})$ and $B(\bar{x})$, that do not share variables with $P_v(\bar{x})$, such that $P(\bar{x}) \equiv A(\bar{x}) \cdot P_v(\bar{x}) + B(\bar{x})$. Clearly, $x_i, x_j \in \mathrm{var}(P_v) \setminus (\mathrm{var}(A) \cup \mathrm{var}(B))$. By the definition of fcg and Lemma 3.9, $P_v(\bar{x})$ has the form $P_v = P_1 \cdot P_2 + c$, where $x_i \in \mathrm{var}(P_1), x_j \in \mathrm{var}(P_2)$ and $c \in \mathbb{F}$. We thus have that

$$\frac{\partial P}{\partial x_i} = A \cdot \frac{\partial P_v}{\partial x_i} = A \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2 \quad \text{and} \quad \frac{\partial P}{\partial x_j} = A \cdot \frac{\partial P_v}{\partial x_j} = A \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j}\,.$$

In addition, we have that

$$\frac{\partial^2 P}{\partial x_i \partial x_j} = A \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j}\,.$$

This implies that

$$\begin{aligned}
\frac{\partial P}{\partial x_i}\Big|_{x_I=\bar{0}} \cdot \frac{\partial P}{\partial x_j}\Big|_{x_I=\bar{0}} &= \left(A \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2\right)\Big|_{x_I=\bar{0}} \cdot \left(A \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j}\right)\Big|_{x_I=\bar{0}} \\
&= \left(A \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j}\right)\Big|_{x_I=\bar{0}} \cdot (A \cdot P_1 \cdot P_2)\Big|_{x_I=\bar{0}} \\
&= \frac{\partial^2 P}{\partial x_i \partial x_j}\Big|_{x_I=\bar{0}} \cdot (A \cdot P_1 \cdot P_2)\Big|_{x_I=\bar{0}} \equiv 0\,.
\end{aligned}$$

In particular, either

$$\frac{\partial P}{\partial x_i}\Big|_{x_I=\bar{0}} \equiv 0 \qquad \text{or} \qquad \frac{\partial P}{\partial x_j}\Big|_{x_I=\bar{0}} \equiv 0\,. \qquad \square$$

Proof of Lemma 3.10. Let P be a ROP and $i \in [n]$. We prove the claim by induction on $k = |\mathrm{var}(P)|$. For $k = 0, 1$ the claim is trivial. For $k \geq 2$ we get by Lemma 3.9 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, $|\mathrm{var}(P_1)| < |\mathrm{var}(P)|$ and so by the induction hypothesis we get that

$$\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i}$$

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.

For the "moreover" part, assume for a contradiction that for some $i \in [n]$, we have that $\frac{\partial P}{\partial x_i}$ is not $\bar{0}$-justified. That is, there exists some $j \in [n]$ such that $\frac{\partial P}{\partial x_i}$ depends on x_j, however, for $I = \mathrm{var}(P) \setminus \{i, j\}$, the polynomial

$$\frac{\partial P}{\partial x_i}\Big|_{x_I=\bar{0}}$$

does not depend on x_j. In other words, we have that

$$\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0 \qquad \text{and} \qquad \frac{\partial^2 P}{\partial x_i \partial x_j}\Big|_{x_I=\bar{0}} \equiv 0\,.$$

Lemma A.1 implies that either

$$\frac{\partial P}{\partial x_i}\Big|_{x_I=\bar{0}} \equiv 0 \qquad \text{or} \qquad \frac{\partial P}{\partial x_j}\Big|_{x_I=\bar{0}} \equiv 0$$

holds. On the other hand, $\{x_i, x_j\} \subseteq \mathrm{var}(P\Big|_{x_I=\bar{0}})$, since P is a $\bar{0}$-justified ROP, and hence

$$\frac{\partial P}{\partial x_i}\Big|_{x_I=\bar{0}} \not\equiv 0 \qquad \text{and} \qquad \frac{\partial P}{\partial x_j}\Big|_{x_I=\bar{0}} \not\equiv 0\,,$$

in contradiction. □

References

[1] MANINDRA AGRAWAL: Proving lower bounds via pseudo-random generators. In *25th Internat. Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05), 2005*, pp. 92–105, 2005. [doi:10.1007/11590156_6] 469

[2] MANINDRA AGRAWAL, CHANDAN SAHA, RAMPRASAD SAPTHARISHI, AND NITIN SAXENA: Jacobian hits circuits: Hitting-sets, lower bounds for depth-D occur-k formulas & depth-3 transcendence degree-k circuits. In *Proc. 44th STOC*, pp. 599–614. ACM Press, 1980. [doi:10.1145/2213977.2214033, arXiv:1111.0582] 506

[3] MATTHEW ANDERSON, DIETER VAN MELKEBEEK, AND ILYA VOLKOVICH: Derandomizing polynomial identity testing for multilinear constant-read formulae. In *Proc. 26th IEEE Conf. on Computational Complexity (CCC'11)*, pp. 273–282, 2011. See also at ECCC. [doi:10.1109/CCC.2011.18] 472, 506, 507

[4] DANA ANGLUIN, LISA HELLERSTEIN, AND MAREK KARPINSKI: Learning read-once formulas with queries. *J. ACM*, 40(1):185–210, 1993. Preliminary version in COLT'89. [doi:10.1145/138027.138061] 470

[5] V. ARVIND AND PARTHA MUKHOPADHYAY: The monomial ideal membership problem and polynomial identity testing. *Inform. and Comput.*, 208(4):351–363, 2010. See also at ECCC. [doi:10.1016/j.ic.2009.06.003] 506

[6] AMOS BEIMEL, FRANCESCO BERGADANO, NADER H. BSHOUTY, EYAL KUSHILEVITZ, AND STEFANO VARRICCHIO: Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000. Preliminary version in FOCS'96. [doi:10.1145/337244.337257] 469

[7] MICHAEL BEN-OR AND PRASOON TIWARI: A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In *Proc. 20th STOC*, pp. 301–309. ACM Press, 1988. [doi:10.1145/62212.62241] 472, 505

[8] DAOUD BSHOUTY AND NADER H. BSHOUTY: On interpolating arithmetic read-once formulas with exponentiation. *J. Comput. System Sci.*, 56(1):112–124, 1998. Preliminary version in COLT'94. [doi:10.1006/jcss.1997.1550] 470, 473

[9] NADER H. BSHOUTY AND RICHARD CLEVE: Interpolating arithmetic read-once formulas in parallel. *SIAM J. Comput.*, 27(2):401–413, 1998. [doi:10.1137/S009753979528812X] 470

[10] NADER H. BSHOUTY, THOMAS R. HANCOCK, AND LISA HELLERSTEIN: Learning arithmetic read-once formulas. *SIAM J. Comput.*, 24(4):706–735, 1995. Preliminary version in STOC'92. [doi:10.1137/S009753979223664X] 470, 471, 473, 474, 475, 479, 498, 505

[11] NADER H. BSHOUTY, THOMAS R. HANCOCK, AND LISA HELLERSTEIN: Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. *J. Comput. System Sci.*, 50(3):521–542, 1995. Preliminary version in COLT'92. [doi:10.1006/jcss.1995.1042] 470

[12] ZEEV DVIR AND AMIR SHPILKA: Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. Comput.*, 36(5):1404–1434, 2007. Preliminary version in STOC'05. [doi:10.1137/05063605X] 472, 506

[13] MICHAEL A. FORBES, RAMPRASAD SAPTHARISHI, AND AMIR SHPILKA: Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proc. 46th STOC*, pp. 867–875. ACM Press, 2014. See also at ECCC. [doi:10.1145/2591796.2591816] 473

[14] MICHAEL A. FORBES AND AMIR SHPILKA: Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *Proc. 54th FOCS*, pp. 243–252. IEEE Comp. Soc. Press, 2013. See also at ECCC. [doi:10.1109/FOCS.2013.34] 473

[15] JOACHIM VON ZUR GATHEN AND ERICH KALTOFEN: Factoring sparse multivariate polynomials. *J. Comput. System Sci.*, 31(2):265–287, 1985. Preliminary version in FOCS'83. [doi:10.1016/0022-0000(85)90044-3] 471

[16] DIMA YU. GRIGORIEV, MAREK KARPINSKI, AND MICHAEL F. SINGER: Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, 19(6):1059–1063, 1990. [doi:10.1137/0219073] 472

[17] ANKIT GUPTA, NEERAJ KAYAL, AND SATYANARAYANA V. LOKAM: Reconstruction of depth-4 multilinear circuits with top fan-in 2. In *Proc. 44th STOC*, pp. 625–642. ACM Press, 1980. See also at ECCC. [doi:10.1145/2213977.2214035] 469

[18] THOMAS R. HANCOCK AND LISA HELLERSTEIN: Learning read-once formulas over fields and extended bases. In *Proc. 4th Ann. Workshop on Computational Learning Theory (COLT'91)*, pp. 326–336, 1991. [ACM:114867] 470, 473, 474, 475, 479, 480, 498

[19] JOOS HEINTZ AND CLAUS-PETER SCHNORR: Testing polynomials which are easy to compute (extended abstract). In *Proc. 12th STOC*, pp. 262–272. ACM Press, 1980. [doi:10.1145/800141.804674] 469

[20] LISA HELLERSTEIN: Private communication, 2009. 472, 507

[21] VALENTINE KABANETS AND RUSSELL IMPAGLIAZZO: Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complexity*, 13(1-2):1–46, 2004. Preliminary version in STOC'03. See also at ECCC. [doi:10.1007/s00037-004-0182-6] 469

[22] MAURICIO KARCHMER, NATI LINIAL, ILAN NEWMAN, MIKE SAKS, AND AVI WIGDERSON: Combinatorial characterization of read-once formulae. *Discrete Mathematics*, 114(1-3):275–282, 1993. [doi:10.1016/0012-365x(93)90372-z] 470

[23] ZOHAR SHAY KARNIN, PARTHA MUKHOPADHYAY, AMIR SHPILKA, AND ILYA VOLKOVICH: Deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in. *SIAM J. Comput.*, 42(6):2114–2131, 2013. Preliminary version in STOC'10. [doi:10.1137/110824516] 506

[24] ZOHAR SHAY KARNIN AND AMIR SHPILKA: Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Proc. 24th IEEE Conf. on Computational Complexity (CCC'09)*, pp. 274–285, 2009. Full version on author's website. [doi:10.1109/ccc.2009.18] 469

[25] ZOHAR SHAY KARNIN AND AMIR SHPILKA: Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. *Combinatorica*, 31(3):333–364, 2011. Preliminary version in CCC'08. [doi:10.1007/s00493-011-2537-3] 472, 506

[26] NEERAJ KAYAL AND SHUBHANGI SARAF: Blackbox polynomial identity testing for depth 3 circuits. In *Proc. 50th FOCS*, pp. 198–207. IEEE Comp. Soc. Press, 2009. See also at ECCC. [doi:10.1109/FOCS.2009.67] 472, 506

[27] NEERAJ KAYAL AND NITIN SAXENA: Polynomial identity testing for depth 3 circuits. *Comput. Complexity*, 16(2):115–138, 2007. Conference version at CCC'06. [doi:10.1007/s00037-007-0226-9] 472, 506

[28] ADAM R. KLIVANS AND AMIR SHPILKA: Learning restricted models of arithmetic circuits. *Theory of Computing*, 2(1):185–206, 2006. Preliminary version in COLT'03. [doi:10.4086/toc.2006.v002a010] 469

[29] ADAM R. KLIVANS AND DANIEL A. SPIELMAN: Randomness efficient identity testing of multivariate polynomials. In *Proc. 33rd STOC*, pp. 216–223. ACM Press, 2001. [doi:10.1145/380752.380801] 469, 470, 472, 505

[30] BARBARA LHOTZKY: *On the computational complexity of some algebraic counting problems*. Ph.D. thesis, University of Bonn, Department of Computer Science, Bonn, Germany, 1991. 472, 505, 507

[31] RICHARD J. LIPTON AND NISHEETH K. VISHNOI: Deterministic identity testing for multivariate polynomials. In *Proc. 14th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'03)*, pp. 756–760. ACM Press, 2003. [ACM:644233] 505

[32] RAN RAZ AND AMIR SHPILKA: Deterministic polynomial identity testing in non-commutative models. *Comput. Complexity*, 14(1):1–19, 2005. Preliminary version in CCC'04. [doi:10.1007/s00037-005-0188-8] 473

[33] SHUBHANGI SARAF AND ILYA VOLKOVICH: Black-box identity testing of depth-4 multilinear circuits. In *Proc. 43rd STOC*, pp. 421–430. ACM Press, 2011. See also at ECCC. [doi:10.1145/1993636.1993693] 472, 506

[34] NITIN SAXENA: Diagonal circuit identity testing and lower bounds. In *Proc. 35th Internat. Colloq. on Automata, Languages and Programming (ICALP'08)*, pp. 60–71, 2008. See also at ECCC. [doi:10.1007/978-3-540-70575-8_6] 506

[35] NITIN SAXENA AND C. SESHADHRI: An almost optimal rank bound for depth-3 identities. *SIAM J. Comput.*, 40(1):200–224, 2011. Preliminary version in CCC'09. See also at ECCC. [doi:10.1137/090770679] 506

[36] NITIN SAXENA AND C. SESHADHRI: Blackbox identity testing for bounded top-fanin depth-3 circuits: The field doesn't matter. *SIAM J. Comput.*, 41(5):1285–1298, 2012. Preliminary version in STOC'11. See also at ECCC. [doi:10.1137/10848232] 472, 506

[37] NITIN SAXENA AND C. SESHADHRI: From Sylvester-Gallai configurations to rank bounds: Improved blackbox identity test for depth-3 circuits. *J. ACM*, 60(5):33, 2013. Preliminary version in FOCS'10. See also at ECCC. [doi:10.1145/2528403] 506

[38] JACOB T. SCHWARTZ: Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. Preliminary version in ISSAC'79. [doi:10.1145/322217.322225] 471, 498

[39] AMIR SHPILKA: Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM J. Comput.*, 38(6):2130–2161, 2009. Preliminary version in STOC'07. [doi:10.1137/070694879] 469

[40] AMIR SHPILKA AND ILYA VOLKOVICH: Read-once polynomial identity testing. In *Proc. 40th STOC*, pp. 507–516. ACM Press, 2008. [doi:10.1145/1374376.1374448] 465, 473, 474, 506

[41] AMIR SHPILKA AND ILYA VOLKOVICH: Improved polynomial identity testing for read-once formulas. In *Proc. 12th Internat. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'09)*, pp. 700–713, 2009. Full version at ECCC. [doi:10.1007/978-3-642-03685-9_52] 473, 474, 506, 513

[42] AMIR SHPILKA AND ILYA VOLKOVICH: On the relation between polynomial identity testing and finding variable disjoint factors. In *Proc. 37th Internat. Colloq. on Automata, Languages and Programming (ICALP'10)*, pp. 408–419, 2010. See also at ECCC. [doi:10.1007/978-3-642-14165-2_35] 478

[43] AMIR SHPILKA AND ILYA VOLKOVICH: Read-once polynomial identity testing. *Electron. Colloq. on Comput. Complexity (ECCC)*, 17:11, 2010. Full version of [41]. [ACM:1616551] 474, 481, 488, 492, 499

[44] AMIR SHPILKA AND AMIR YEHUDAYOFF: Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. [doi:10.1561/0400000039] 469, 470

[45] ILYA VOLKOVICH: Characterizing arithmetic read-once formulae. *Manuscript*, 2014. [arXiv:1408.1995] 471, 475

[46] RICHARD ZIPPEL: Probabilistic algorithms for sparse polynomials. In *Internat. Symp. Symbolic and Algebraic Computation (ISSAC'79)*, pp. 216–226, 1979. [doi:10.1007/3-540-09519-5_73] 471, 498

AUTHORS

Amir Shpilka
Associate Professor
Tel-Aviv University, Tel-Aviv, Israel
shpilka@post.tau.ac.il
http://www.cs.tau.ac.il/~shpilka

Ilya Volkovich
Postdoctoral Visiting Scholar
Department of Electrical Engineering and Computer Science
University of Michigan
ilyavol@umich.edu
http://www.umich.edu/~ilyavol

ABOUT THE AUTHORS

AMIR SHPILKA obtained his Ph. D. in Computer Science and Mathematics from the Hebrew University in Jerusalem in 2001 under the supervision of Avi Wigderson. From 2005 to 2014 he was a faculty member at the CS department at the Technion. In October 2014 he joined the CS department at Tel-Aviv University. He is married to Carmit and has two children. His research interests lie in complexity theory, especially in arithmetic circuit complexity. When not working or enjoying his family he likes to read and play chess.

ILYA VOLKOVICH graduated from the Computer Science Department of the Technion in 2012, where he had the pleasure of doing his Ph. D. under the supervision of Amir Shpilka. The topic of his thesis was Polynomial Identity Testing and related algebraic problems. Subsequently he was a postdoc at the Center for Computational Intractability at Princeton University and held a visiting position at the School of Mathematics at the Institute for Advanced Study. Currently he is a postdoctoral visitor at the University of Michigan. His research interests lie in theoretical computer science and discrete mathematics, in particular in computational complexity theory, derandomization, algebraic complexity and applications of algebraic tools to algorithms and other areas of CS. His recent interests also include algorithmic and general game theory.