Read-once Polynomial Identity Testing

A. Shpilka* I. Volkovich*

Abstract

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. In this paper we study the problems of identity testing and reconstruction of read-once formulas. the following are some of the results that we obtain.

1. Given k ROFs in n variables, over a field $\mathbb{F}$, we give a deterministic (non black-box) algorithm that checks whether they sum to zero or not. The running time of the algorithm is $n^{\mathcal{O}(k^2)}$.
2. We give an $n^{\mathcal{O}(d+k^2)}$ time deterministic algorithm for checking whether a black box holding the sum of k depth d ROFs in n variables computes the zero polynomial. In other words, we provide a hitting set of size $n^{\mathcal{O}(d+k^2)}$ for the sum of k depth d ROFs. If $|\mathbb{F}|$ is too small then we make queries from a polynomial size extension field. This implies a deterministic algorithm that runs in time $n^{\mathcal{O}(d)}$ for the reconstruction of depth d ROFs.
3. We give a hitting set of size $\exp(\tilde{\mathcal{O}}(\sqrt{n} + k^2))$ for the sum of k ROFs (without depth restrictions). In particular this implies a sub-exponential time deterministic algorithm for black-box identity testing and reconstructing of ROFs. As before, if $|\mathbb{F}|$ is too small then we make queries from a polynomial size extension field.

To the best of our knowledge our results give the first sub-exponential time black-box identity testing algorithm for the sum of (a constant number of) ROFs and the first polynomial time identity testing algorithm for the sum of (a constant number of) ROFs, in the non black-box setting.

Another question that we study is the following generalization of the polynomial identity testing problem. Given an arithmetic circuit computing a polynomial $P(\bar{x})$, decide whether there is a ROF computing $P(\bar{x})$. If there is such a formula then output it. Otherwise output "No". We call this question the *read-once testing problem* (ROT for short). Previous (randomized) algorithms for reconstruction of ROFs imply that there exists a randomized algorithm for the read-once testing problem. In this work we show that most previous algorithms for polynomial identity testing can be strengthen to yield algorithms for the read-once testing problem. In 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 (both in the black-box and non black-box settings, where the running time depends on the model), non-commutative formulas and sum of k ROFs. The running time of the ROT algorithm is essentially the same running time as the corresponding PIT algorithm for the class.

The main tool in most of our results is a new connection between polynomial identity testing and reconstruction of read-once formulas. Namely, we show that in any model that is closed under partial derivatives (that is, 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.

*Faculty of Computer Science, Technion, Haifa 32000, Israel. Email: {shpilka,ilyav}@cs.technion.ac.il.
Research supported by the Israel Science Foundation (grant number 439/06).

Contents

1 Introduction **3**
- 1.1 Comparison to Previous Works . . . 6
- 1.2 Organization . . . 6

2 Preliminaries **7**
- 2.1 Partial Derivatives . . . 8
- 2.2 Some useful Facts about Polynomials . . . 10
- 2.3 Polynomials and Circuit Classes . . . 10

3 Read-Once Formulas and Read-Once Polynomials **11**
- 3.1 The Gates-Graph of ROFs and ROPs . . . 12
- 3.2 Factors of ROPs . . . 14
- 3.3 Partial Derivatives of ROPs . . . 15
- 3.4 Multiplicative and $\bar{0}-$Justified ROPs . . . 16

4 ROF Graph-Related Algorithms **17**
- 4.1 Reconstruction of a ROF . . . 18
- 4.2 Factoring a ROF . . . 18
- 4.3 Counting the Number of Monomials in a ROF . . . 19
- 4.4 Computing a Partial Derivative of a ROF . . . 21

5 Generalization of PIT Algorithms to Read-Once Testing **21**
- 5.1 Generic Scheme . . . 21
- 5.2 From PIT to Justifying Assignments . . . 22
  - 5.2.1 From a Hitting Set to a Justifying Set . . . 24
  - 5.2.2 $\bar{0}-$Justification . . . 26
- 5.3 Read-Once Verification . . . 26
  - 5.3.1 Alternative Verification Tests . . . 29
- 5.4 Proof of Theorem 7 . . . 30

6 ROT for Specific Models **30**
- 6.1 Sparse Polynomials . . . 30
- 6.2 Depth-3 Circuits . . . 31
- 6.3 Non-Commutative Formulas . . . 32

7 PIT Algorithm for Alternating Read-Once Formulas **35**
- 7.1 Some Properties of AROFs . . . 35
- 7.2 Black-Box PIT Algorithm for Bounded-Depth AROFs . . . 36
- 7.3 PIT for General Read-Once Formulas . . . 38

8 PIT for Sum of Read-Once Formulas **40**
- 8.1 The Algorithm . . . 41
- 8.2 Analysis of the Algorithm . . . 41
- 8.3 Hardness of Representation Theorem for Sum of $\bar{0}-$Justified ROFs . . . 42
- 8.4 Proofs of Theorems 4, 5, 6 and 8 . . . 44

1 Introduction

Let $\mathbb{F}$ be a field and C an arithmetic circuit in the variables $\bar{x} = (x_1, \ldots, x_n)$ over $\mathbb{F}$. The output $C(\bar{x})$ is a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$. Can we determine whether $C(\bar{x})$ is the zero[1] polynomial? This is the well known *polynomial identity testing* problem (PIT for short). The celebrated Schwartz-Zippel result [Zip79, Sch80] implies a polynomial time randomized algorithm for the PIT problem, but the main question is whether we can do this deterministically. There are two scenarios in which PIT is considered. In the black-box setting we can only access the unknown polynomial by querying its values on inputs of our choice. Thus, a PIT algorithm basically gives a set of points such that if the polynomial is non-zero then it must be non-zero on one of the points in the set (such a set is known as a *hitting set*). The only information that we have about the polynomial is that there exists a circuit of a certain form that computes it. In the non black-box setting we also have the circuit computing the polynomial at our disposal, so the algorithm has much more power in this setting. We shall consider both the black-box setting and the non black-box setting, and will make clear when we consider each of the settings.

The problem of giving deterministic sub-exponential PIT algorithms is believed to be very difficult. In particular it was shown that efficient PIT algorithms imply lower bounds for arithmetic circuits [KI04, Agr05]. In view of the difficulty of providing sub-exponential deterministic PIT algorithms for general arithmetic circuits, research focused on restricted models such as depth-2 and depth-3 circuits and non-commutative arithmetic formulas. It is a major open problem to give sub-exponential time deterministic algorithms for PIT of bounded depth circuits and of multilinear formulas.

The focus of this work is on models related to arithmetic read-once formulas (ROF for short). An *arithmetic read-once formula* is a formula (a circuit in which the fan-out of every gate is at most 1) in which the operations are $\{+, \times\}$ and such that every input variable labels at most one leaf. Although read-once formulas form a very restricted model of computation they received a lot of attention both in the boolean world [KLN$^+$93, AHK93, BHH95b] and in the algebraic world [HH91, BHH95a, BB98, BC98]. However, no deterministic sub-exponential time black-box PIT algorithm for arithmetic ROF was known prior to this work. This sad state of affairs implies that if we want to give efficient algorithms for bounded depth circuits or for multilinear formulas then we should first try to find algorithms for read-once formulas. In view of this we study several models related to ROFs and give new PIT algorithms for them. In particular, we give the first deterministic sub-exponential time black-box algorithm for identity testing and reconstruction of read-once formulas.

Theorem 1. *There is a deterministic* $\exp(\tilde{\mathcal{O}}(\sqrt{n}))$ *time black-box algorithm for identity testing of read-once formulas. In particular, this implies a sub-exponential time deterministic algorithm for reconstruction of read-once formulas (that is, there is an algorithm that outputs a ROF computing the same polynomial). In case that* $|\mathbb{F}|$ *is too small we make queries to the black-box from a polynomial size extension field.*

In particular, we construct a hitting set of size $\exp(\tilde{\mathcal{O}}(\sqrt{n}))$ for read-once formulas, that also enables the reconstruction of the formula. This result relies on the following theorem that studies read-once formulas of relatively small depth.

Theorem 2. *There is an* $n^{\mathcal{O}(d)}$ *time deterministic algorithm for checking whether a black-box holding an* n*-variate read-once formula of depth* d *(in the unbounded fan-in model) computes the zero polynomial. In case that* $|\mathbb{F}|$ *is too small we make queries to the black-box from a polynomial*

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

size extension field. As a consequence we also get a reconstruction algorithm of roughly the same running time for depth d read-once formulas.

If fact, we prove the above theorem for the *alternating* model which is a generalization of the (standard) unbounded fan-in model. The following result gives an efficient PIT algorithm (in the non black-box setting) when the input is the sum of a small number of read-once formulas

Theorem 3. *Given k read-once formulas in n variables, over a field $\mathbb{F}$, there is a deterministic algorithm that checks whether they sum to zero or not. The running time of the algorithm is $n^{\mathcal{O}(k^2)}$.*

By combining ideas from the proofs of the previous theorems we are able to prove the following theorems that strengthen the results of Theorems 1 and 2 [2]. Basically, the theorems show how to perform black-box PIT when the input is a sum of a small number of read-once formulas.

Theorem 4. *There is an $\exp(\tilde{\mathcal{O}}(\sqrt{n}+k^2))$ time black-box deterministic algorithm for identity testing for the sum of k read-once formulas. In case that $|\mathbb{F}|$ is too small we make queries to the black-box from a polynomial size extension field.*

Theorem 5. *There is an $n^{\mathcal{O}(d+k^2)}$ time deterministic algorithm for checking whether a black box holding a sum of k read-once formulas of depth d (in the unbounded fan-in model), in n variables, computes the zero polynomial. In case that $|\mathbb{F}|$ is too small we make queries to the black-box from a polynomial size extension field.*

In fact we can generalize the previous theorems to the case where we have a sum of ROFs that is read-k. That is, every variable appears in at most k of the formulas (see Definition 8.5).

Theorem 6. *Let $F=\sum_{m=1}^{\ell} F_m$ be a read-k sum of ROFs. Then there is an $\exp(\tilde{\mathcal{O}}(\sqrt{n}+k^2))$ time deterministic black-box PIT algorithm for F. If in addition we are guaranteed that the F_m's are depth d ROFs then there is an $n^{\mathcal{O}(d+k^2)}$ time deterministic black-box PIT algorithm for F. In the non black-box setting there is an $n^{\mathcal{O}(k^2)}$ deterministic PIT algorithm for F.*

In addition to giving PIT algorithms for models related to ROFs, we are interested in a generalization of the problem, that we call the *read-once testing problem* (ROT for short).[3]

Problem 1. *Given a circuit C (maybe as a black-box) computing some polynomial $P(\bar{x})$, decide whether P can be computed by an (arithmetic) read-once formula, and if the answer is positive then compute the read-once formula for it.*

This problem is a generalization of the PIT problem, as the zero polynomial is computable by a read-once formula. Moreover, given a read-once formula it is easy to check whether it computes the zero polynomial. Similarly to the PIT problem, there is an efficient randomized algorithm for the read-once testing problem. Indeed by the results of [HH91] there is a randomized algorithm for reconstructing read-once formulas, that are given as black-box, and then using the Schwartz-Zippel randomized identity testing algorithm [Zip79, Sch80] we can check whether the read-once formula that we computed, computes the same polynomial as the one in the black-box. The main question is whether we can find an efficient deterministic algorithm for the read-once testing problem. We show that if we have an efficient PIT algorithm for a circuit class $\mathcal{M}$, that satisfy some closure properties, then we can also solve the read-once testing problem for the class efficiently. Formally we prove the following theorem.

[2] The reason for stating the theorems and their strengthening is that in order to prove the stronger theorems we first have to prove Theorems 1 and 2.

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

Theorem 7. *Let $\mathcal{M}$ be a class of arithmetic circuits, that can compute any univariate linear function (i.e. for every $f = ax_i + b$ there is a circuit in $\mathcal{M}$ that computes it). Denote with $\mathcal{M}_V$ 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 $\mathcal{L}(\mathcal{M})$[4] and C_2, C_3 and C_4 are variable disjoint. Assume that there is a deterministic PIT algorithm that runs in time $T(n, s)$, when given access (explicit or via a black-box) to a circuit in n-variables, of size s, that belongs to $\mathcal{M}_V$. Then there is a deterministic algorithm that runs in time $\text{poly}(n, s, T(s, n))$ that when given access (explicit or via a black-box) to an n-variate circuit of size s from $\mathcal{M}$, solves the ROT problem.

Note that for most circuit classes for which we have efficient PIT algorithms, we also have efficient PIT algorithms for their "closure", $\mathcal{M}_V$. As a corollary we get that all previous PIT algorithms can be generalized to yield algorithms for ROT. In particular we obtain the following results. We first give a strengthening of Theorems 3, 4 and 5.

Theorem 8. *Let $\{F_m\}_{m\in[k]}$ be k read-once formulas in n variables, over a field $\mathbb{F}$. Let F be their sum, i.e. $F = F_1 + \ldots + F_k$. Then we have the following algorithms.*

1. *There is a deterministic algorithm that given the ROFs $\{F_m\}_{m\in[k]}$ solves the ROT problem and runs in time $n^{\mathcal{O}(k^2)}$.*
2. *There is a deterministic algorithm that given a black-box access to F solves the ROT problem in time $\exp(\tilde{\mathcal{O}}(\sqrt{n} + k^2))$.*
3. *If in addition all the F_m's are of depth d, then there is a deterministic algorithm that given a black-box access to F solves the ROT problem in time $n^{\mathcal{O}(d+k^2)}$.*

As before the results remain the same when F is read-k sum of ROFs and not just the sum of k ROFs.

The following result strengthens many reconstruction algorithms that were given for the model of sparse polynomials ([BOT88, GKS90, KS01] just to name a few). We are not sure however whether our result is new, but we give it anyway (see last sentence of paragraph 4 on page 707 of [BHH95a] that refers to [Lho91]).

Theorem 9. *Given a black-box access to a polynomial P with m monomials (i.e. a polynomial computed by a depth-2 arithmetic circuit with m multiplication gates) there is a polynomial (in n, m) time algorithm that solves the read-once testing problem for P. If $|\mathbb{F}|$ is too small then we make queries from a polynomial size extension field.*

Our next result gives read-once formula testing for depth-3 circuits with bounded top fan-in. This result generalizes the results of [DS06, KS07b, KS07a, AM07] who gave deterministic identity testing algorithms for this model.

Theorem 10. *Given a black box holding a depth-3 circuit C with k multiplication gates (i.e. a $\Sigma\Pi\Sigma(k)$ circuit) there is an $\exp(\log^{\mathcal{O}(k^2)} n)$ time algorithm that solves the read-once testing problem. If C is a multilinear circuit then the running time of our algorithm is $\text{poly}(n)$ (for a constant k). If C is explicitly given to us then there is a polynomial time algorithm that solves the read-once testing problem.*

[4] $\mathcal{L}(\mathcal{M})$ is the class of all circuits of the form $\alpha \cdot C + \beta$ for $\alpha, \beta \in \mathbb{F}$ and $C \in \mathcal{M}$. See Definition 2.18.

The last result of this kind concerns non-commutative formulas and generalizes the polynomial identity testing result of [RS05] for this model.

Theorem 11. *Given a non-commutative formula (in the non black-box setting) there is a polynomial time algorithm (in the size of the formula) for the read-once testing problem.*

1.1 Comparison to Previous Works

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

In [HH91] learning algorithms for read-once arithmetic formulas that use *membership and equivalence* queries were given. A membership query to a ROF $f(\bar{x})$ is simply a query that asks for the value of $f(\bar{x})$ on a specific input. An equivalence query on the other hand, gives the oracle a certain hypothesis, $h(\bar{x})$, and the oracle answers "equal" if $f \equiv h$ or returns an input $\bar{\alpha}$ such that $f(\bar{\alpha}) \neq h(\bar{\alpha})$. Note that when it comes to deterministic PIT algorithms, equivalence queries are irrelevant (as we can always give as an hypothesis the zero polynomial). Using this language, our results for the black-box setting use only membership queries (both for the PIT and read-once testing problems).

The results of [HH91] were later extended by [BHH95a] in two different ways. First, they allowed the formulas to contain division gates (in addition to $\{+, \times\}$), and secondly they allowed the algorithm to use randomness instead of using equivalence queries. They also considered a model in which *justifying assignments* (a notion that we later define) are given in advance to the algorithm. Again, this approach is irrelevant when it comes to deterministic PIT algorithms. However, we do use some of the techniques of [HH91, BHH95a] in our read-once testing algorithms. That is, we use their ideas in order to construct a candidate ROF that *may* compute the same polynomial as the one computed by the given circuit (recall Problem 1). The main difficulty is then to verify deterministically that this ROF indeed computes the same polynomial as the given circuit. Indeed, the main difference between the reconstruction problem and the read-once testing problem is that in the reconstruction setting we are guaranteed that there exists a ROF for the given circuit, so if we use the results of [HH91, BHH95a] we are guaranteed that we have the correct ROF, whereas in the read-once testing setting we first have to check whether the polynomial can be computed by a ROF and only then we can try and find the formula. Thus, when we use the reconstruction algorithm we get a ROF and then we still have to verify that it does compute the same polynomial as the given circuit (or black-box). This may seem like a small point, however this is exactly the reason that we cannot use the results of [KS07b] to get a polynomial time black-box algorithm in Theorem 10, and instead settle for the results of [DS06, KS07a] due to the additional structure they guarantee.

1.2 Organization

The paper is organized as follows. In Sections 2 and 3 we give the basic definitions and notations. In Section 4 we give several basic algorithms for ROFs (some of them already known but used in this paper as well). Then in Section 5 we show how to convert PIT algorithms to ROT algorithms and prove Theorem 7. In Section 6 we show how to use Theorem 7 to prove Theorems 9, 10 and 11. New PIT algorithms for bounded depth and black-box ROFs is given in Section 7. Finally, in Section 8 we give PIT algorithms for sum of ROFs.

2 Preliminaries

For a positive integer n we denote $[n] = \{1, \ldots, n\}$. For a graph $G = (V, E)$ we denote with G^c the complement graph. That is $G^c = (V, E')$ such that for $i \neq j \in V$ we have that $(i, j) \in E$ if and only if $(i, j) \notin E'$. The following definitions are for a polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and an assignment $\bar{a} \in \mathbb{F}^n$:

Definition 2.1. *We say that* P depends on x_i *if there exist* $\bar{a} \in \mathbb{F}^n, b \in \mathbb{F}$ *such that:*

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

We denote $\mathrm{var}(P) \stackrel{\Delta}{=} \{x_i \mid P \text{ depends on } x_i\}$. *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 2.2. *Given a subset* $I \subseteq [n]$ *and an assignment* $\bar{a} \in \mathbb{F}^n$ *we define* $P^I_{\bar{a}}(\bar{x})$ *to be the polynomial resulting from substituting* a_i *to the variable* x_i *for every* $i \in I$. *In particular* $P^I_{\bar{a}}(\bar{x})$ *is defined on* $[n] \setminus I$.

Example 2.3. *Let* $P(x_1, x_2, x_3) = x_1^2 + 2x_2x_3 + 4x_1x_3^2$, $I = \{1, 3\}$ *and* $\bar{a} = (0, 4, 1)$. *Then* $\mathrm{var}(P) = \{1, 2, 3\}$, $P^I_{\bar{a}}(\bar{x}) = P(0, x_2, 1) = 2x_2$ *and* $\mathrm{var}(P^I_{\bar{a}}(\bar{x})) = \{2\}$.

Example 2.4. *Let* $P(x_1, x_2, x_3) = 2x_2x_3 + 1$, $I = \{2\}$ *and* $J = \{2, 3\}$. *For* $\bar{a} = (0, 0, 0)$ *we get that* $P^I_{\bar{a}}(\bar{x}) = P(x_1, 0, x_3) = 1$ *and* $P^J_{\bar{a}}(\bar{x}) = P(x_1, 0, 0) = 1$. *Note that* $\mathrm{var}(P) = \{2, 3\}$ *but* $\mathrm{var}(P^I_{\bar{a}}(\bar{x})) = \emptyset$.

We can conclude that by substituting a value to a variable of P we, obviously, eliminate the dependence of P on this variable, however we may also eliminate the dependence of P on other variables and thus lose more information than intended. To conclude we give the following trivial observation.

Observation 2.5. *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* $\mathrm{var}(P^I_{\bar{a}}) \subseteq \mathrm{var}(P^J_{\bar{a}}) \subseteq \mathrm{var}(P) \setminus J$.

For the purposes of reconstruction and identity testing 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 [HH91] and [BHH95a], but we repeat the definitions here to ease the reading of the paper (we also slightly change some of the definitions).

Definition 2.6 (Justifying assignment)**.** *Given an assignment* $\bar{a} \in \mathbb{F}^n$ *we say that* $\bar{a}$ *is a* justifying assignment of P *if for every subset* $I \subseteq \mathrm{var}(P)$ *we have that*

$$\mathrm{var}(P^I_{\bar{a}}) = \mathrm{var}(P) \setminus I \tag{1}$$

Though this definition is very intuitive, ensuring that a given assignment $\bar{a}$ satisfies condition 1 for every $I \subseteq \mathrm{var}(P)$ seems to be a computationally hard problem. The following proposition provides an alternative and more useful definition:

Proposition 2.7. $\bar{a} \in \mathbb{F}^n$ *is a justifying assignment of* P *if and only if condition 1 holds for every subset* I *of size* $|\mathrm{var}(P)| - 1$.

Proof. Let $J \subseteq \text{var}(P)$. If $J = \text{var}(P)$ then, obviously, $\text{var}(P_{\bar{a}}^J) = \emptyset = \text{var}(P) \setminus J$. Otherwise, let $x \in \text{var}(P) \setminus J$. Consider the set $I_x = \text{var}(P) \setminus \{x\}$. From the definition: $|I_x| = |\text{var}(P)| - 1$ and $J \subseteq I_x$, and thus condition 1 holds for I_x. Combining with Observation 2.5 we obtain $\forall x \in \text{var}(P) \setminus J$:

$$\{x\} = \text{var}(P) \setminus I_x = \text{var}(P_{\bar{a}}^{I_x}) \subseteq \text{var}(P_{\bar{a}}^J) \subseteq \text{var}(P) \setminus J$$

and consequently $\text{var}(P_{\bar{a}}^J) = \text{var}(P) \setminus J$. The second direction of the claim is trivial. □

The justification notion can be both expanded and weakened:

Definition 2.8. *Given an assignment $\bar{a} \in \mathbb{F}^n$ we say that $\bar{a}$ is a* weakly-justifying assignment of P *if condition 1 holds for $|I| = 1$ (we ignore sets I of size larger that 1).*

Note that any justified assignment is also weakly-justified, but not vice versa. Later on we will encounter several cases in which justification and weak-justification occur simultaneously. We can, as well, define justification as a property of polynomials:

Definition 2.9. *We say that a polynomial* P *is* $\bar{a}-$justified *if $\bar{a}$ is justifying assignment of P. We define the term $\bar{a}$-weakly-justified in a similar manner.*

We shall make use of the following notations in some of our proofs and algorithms.

Definition 2.10. *For $n \geq 1$ let $\mathcal{P}_n(\bar{x}) = \prod_{i=1}^{n} x_i$. When $n = 0$ we define $\mathcal{P}_0(\bar{x}) = 1$.*

For a set $0 \in W \subseteq \mathbb{F}$ and $k \leq n$ let $A_k^n(W)$ be the set of all vectors in W^n with Hamming weight at most k, that is the set of all vectors in W^n that have **at most** k non-zero coordinates. We formalize it in the following definition.

Definition 2.11. *For subsets $W \subseteq \mathbb{F}$, $I \subseteq [n]$ we define:*

1. *Characteristic sets w.r.t I: $j \in [n]$, $W_j^I = \begin{cases} W & j \in I \\ \{0\} & j \notin I \end{cases}$*
2. *Cartesian product of characteristic sets. $\prod_I(W) = W_1^I \times W_2^I \times \cdots \times W_n^I$.*
   Meaning that the set I gives the indices of the non-zero coordinates of the vectors in $\prod_I(W)$
3. $\mathcal{A}_k^n(W) \bigcup_{\substack{I \subseteq [n] \\ |I| \leq k}} \prod_I(W)$
   As required, the set of all vectors with at most k non-zero coordinates.

An immediate conclusion from the Definition 2.11 is that $|\mathcal{A}_k^n(W)| = \sum_{i=0}^{k} \binom{n}{i} \cdot (|W| - 1)^i$.

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.). Here we use the natural extension of the concept for polynomials over arbitrary fields.

Definition 2.12. *Let P be a n variate polynomial over a field $\mathbb{F}$ and let d_i be the degree of x_i in P. For every variable x_i, P can be uniquely written as $P = \sum_{k=0}^{d_i} x_i^k P_k(\bar{x})$, where $P_k(\bar{x})$ are polynomials in the rest of the variables. We define: $\frac{\partial P}{\partial x_i} \triangleq \sum_{k=1}^{d_i} k x_i^{k-1} P_k(\bar{x})$.*

Notice that this definition coincides with the "analytical" one when $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$. The following lemma is easy to verify.

Lemma 2.13. *For a multilinear polynomial P the following basic properties hold:*

- $\frac{\partial P}{\partial x_i} = P|_{x_i=1} - P|_{x_i=0}$.
- *P depends on x_i if and only if $\frac{\partial P}{\partial x_i} \not\equiv 0$.*
- *$\frac{\partial P}{\partial x_i}$ does* not *depend on x_i (in particular $\frac{\partial^2 P}{\partial x_i^2} \equiv 0$).*
- $\frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}(\frac{\partial P}{\partial x_j}) = \frac{\partial^2 P}{\partial x_j \partial x_i}$.
- $\forall i \neq j \ \frac{\partial P}{\partial x_i}|_{x_j=a} = \frac{\partial}{\partial x_i}(P|_{x_j=a})$.
- *$\bar{a}$ in$\mathbb{F}^n$ is a justifying assignment of P if and only if $\forall i \in \mathrm{var}(P)$ we have $\frac{\partial P}{\partial x_i}(\bar{a}) \neq 0$.*

Since the above properties trivially hold, we will use them implicitly. The following lemma contains the equivalent derivation rules when we restrict ourselves to the multilinear polynomials domain. That is, in all the cases we must ensure that the results of the specified arithmetic operations are indeed, multilinear polynomials. As this is not the general case, it imposes various (implicit) variable-disjointness restrictions. For example - a quotient of two arbitrary multilinear polynomial is not necessarily a multilinear polynomial. Moreover, in some cases it might not be a polynomial at all.

Lemma 2.14. *Let P, G, Q be multilinear polynomials. Then the following derivation rules hold (with the appropriate implicit restrictions)*

1. ***Sum Rule**. Let $Q = P + G$ then trivially*

$$\frac{\partial Q}{\partial x_i} = \frac{\partial P}{\partial x_i} + \frac{\partial G}{\partial x_i}$$

2. ***Product Rule**. Let $Q = P \cdot G$. In this case either $\frac{\partial P}{\partial x_i} \equiv 0$ or $\frac{\partial G}{\partial x_i} \equiv 0$ holds. Hence,*

$$\frac{\partial Q}{\partial x_i} = \frac{\partial P}{\partial x_i} \cdot G + P \cdot \frac{\partial G}{\partial x_i}$$

3. ***Quotient Rule**. Let $Q = \frac{P}{G}$. The equality*

$$G^2 \cdot \frac{\partial Q}{\partial x_i} = G \cdot \frac{\partial P}{\partial x_i} - P \cdot \frac{\partial G}{\partial x_i}$$

can be obtained by applying the Product rule on $P = Q \cdot G$.

4. ***Chain Rule***. *Let $Q(y, \bar{z})$ be a polynomial such that $P(\bar{x}, \bar{z}) \equiv Q(G(\bar{x}), \bar{z})$. Then*

$$\frac{\partial P}{\partial x_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial G}{\partial x_i}$$

Notice that since Q is a multilinear polynomial, $\frac{\partial Q}{\partial y}$ does not depend on y.

From now, unless otherwise is specified, we denote by $\frac{\partial P}{\partial x_i}$ the *discrete* partial derivative of P with regard to x_i that is: $P|_{x_i=1} - P|_{x_i=0}$.

Definition 2.15. *For a non-empty set $V = \{x_{i_1}, x_{i_2}, x_{i_3}, \ldots, x_{i_{|V|}}\}$ we define the partial derivative with respect to V in the following way:*

$$\partial_V P \triangleq \frac{\partial^{|V|} P}{\partial x_{i_1} \partial x_{i_2} \partial x_{i_3} \cdots \partial x_{i_{|V|}}}.$$

Similarly, for a subset $I \subseteq [n]$, $I = \{i_1, \ldots, i_{|I|}\}$, we denote

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

2.2 Some useful Facts about Polynomials

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

Lemma 2.16. *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial. Suppose that for $i \in [n]$ the degree of x_i is bounded by d_i, and let $S_i \subseteq \mathbb{F}$ be such that $|S_i| > d_i$. Let $S = S_1 \times S_2 \times \cdots \times S_n$. Then $P \equiv 0$ iff $P|_S \equiv 0$.*

A proof can be found in [Alo99].

Lemma 2.17 (Gauss)**.** *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n, y]$ be a non-zero polynomial and $g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be such that $P_{|y=g(\bar{x})} \equiv 0$. Then $y - g(\bar{x})$ is an irreducible factor of P in the ring $\mathbb{F}[x_1, x_2, \ldots, x_n, y]$.*

2.3 Polynomials and Circuit Classes

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

Definition 2.18. *We shall use the following notations.*

1. *We say that the circuit class $\mathcal{M}$* contains *the polynomial P if P can be computed by some circuit C from $\mathcal{M}$. We denote it by $P \in \mathcal{M}$.*
2. *We say that the circuit class $\mathcal{M}_1$* contains *the circuit class $\mathcal{M}_2$ if it contains all its polynomials (e.g. $P \in \mathcal{M}_1 \implies P \in \mathcal{M}_2$). We denote it by $\mathcal{M}_1 \subseteq \mathcal{M}_2$.*
3. *Two circuit classes $\mathcal{M}_1, \mathcal{M}_2$ are called* equivalent *iff they contain the same polynomials (e.g. $\mathcal{M}_1 \subseteq \mathcal{M}_2$ and $\mathcal{M}_2 \subseteq \mathcal{M}_1$). We denote them by $\mathcal{M}_1 \cong \mathcal{M}_2$.*
4. *Similarly we define* union, intersection *and* subtraction *of two circuit classes $\mathcal{M}_1 \cup \mathcal{M}_2$, $\mathcal{M}_1 \cap \mathcal{M}_2$, $\mathcal{M}_1 \setminus \mathcal{M}_2$.*

5. *For a circuit class $\mathcal{M}$ we define the (discrete)* Partial Derivatives *of $\mathcal{M}$ as:* $\partial\mathcal{M} \stackrel{\Delta}{=} \{\frac{\partial P}{\partial x_i} | P \in \mathcal{M}, i \in [n]\}$.

6. *For a circuit class $\mathcal{M}$ we define the* linear closure *of $\mathcal{M}$ as:* $\mathcal{L}(\mathcal{M}) \stackrel{\Delta}{=} \{\alpha \cdot P + \beta | P \in \mathcal{M}, \alpha, \beta \in \mathbb{F}\}$.

Trivially, two equivalent circuit classes admit the same black-box PIT algorithms. Yet, they may not admit the same explicit (non black-box) PIT algorithms due to structural difference between them.

3 Read-Once Formulas and Read-Once Polynomials

In this section we give some basic definitions related to read-once formulas. Most of the definitions are from [HH91], or are small variants of their definitions. We start by formally defining the notions of a read-once formula, a skeleton of a read-once formula and a read-once polynomial.

Definition 3.1. *A* read-once arithmetic formula *(ROF for short) over a field $\mathbb{F}$ in the variables $\bar{x} = (x_1, \ldots, x_n)$ is a binary tree whose leafs are labelled with the input variables and whose internal nodes are labelled with the arithmetic operations $\{+, \times\}$ and with a pair of field elements* [5] *$(\alpha, \beta) \in \mathbb{F}^2$. 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 op and with (α, β), and its children compute the polynomials f_{v_1} and f_{v_2} then the polynomial computed at v is*

$$f_v = \alpha \cdot (f_{v_1} \ op \ f_{v_2}) + \beta.$$

The skeleton *of a ROF f is the tree obtained from f after removing all the labels (α, β) from the nodes (but keeping the operations and the input variables).*

We say that a ROF (instance) f is non-degenerate *if it depends on all the variables appearing in it.*

As our ROF model does not allow constants as inputs (as we let constants label the internal nodes of the formula), we represent the constant polynomials with a special gate CONST *that has only one label, a. We denote by $\mathcal{C}_a$ the non-degenerate ROF computing the constant polynomial $P \equiv a$.*

Definition 3.2. *A polynomial $P(\bar{x})$ is a* read-once polynomial *(ROP for short) if it can be computed by a read-once formula. For a ROP* P *we define* $\mathcal{ROF}(P) \stackrel{\Delta}{=} \{f | f$ *is a ROF computing* $P\}$.

A special class of ROFs that will play an important role in our proofs is the class of multiplicative ROFs.

Definition 3.3. *A ROF is called* multiplicative ROF *if it has no addition gates. A polynomial computed by a multiplicative ROF is called a* multiplicative ROP.

Note, because we allow gates to apply linear functions on the results of their operations, the output of a multiplicative ROF can be more than a mere monomial.

Example 3.4. *The polynomial* $(5x_1 \cdot x_2 + 1) \cdot ((-x_3 + 2) \cdot (2x_4 - 1) + 5)$ *has a multiplicative ROF.*

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

When considering read-once formulas of small depth we allow the tree to have unbounded fan-in (and not just fan-in 2 as in the definition above). Moreover, we shall allow small depth ROF to use generalized multiplication gates. A generalized multiplication gate on the inputs $(x_1, \ldots, x_k)$ is allowed to compute *any* multiplicative ROP in its input variables.

Definition 3.5. *An* alternating read-once formula *(AROF) over a field $\mathbb{F}$ in the variables $\bar{x} = (x_1, \ldots, x_n)$ is a tree, of unbounded fan-in, whose leafs are labelled with the input variables and whose internal nodes are labelled with either + or MUL. Each input variable can label at most one leaf. Every leaf and every + gate are labelled with two field elements $(\alpha, \beta) \in \mathbb{F}^2$. In addition, any children of a MUL (+) gate is either a leaf or a + (MUL) gate (this is the reason for the name alternating ROF). The computation is performed in the following way. A leaf labelled with the variable x_i and with (α, β) computes the polynomial $\alpha x_i + \beta$. If a node v, of fan-in k, is labelled with + and (α, β) and its children compute the polynomials $f_{v_1}, \ldots, f_{v_k}$ then the polynomial computed at v is*

$$f_v = \alpha \cdot \left(\sum_{i=1}^{k} f_{v_1} \right) + \beta.$$

If v is labelled with MUL then it computes a multiplicative ROP in its input variables. That is, if v computes the multiplicative ROP g, and its children compute the polynomials $f_{v_1}, \ldots, f_{v_k}$, then the output of v will be the polynomial

$$f_v = g(f_{v_1}, \ldots, f_{v_k}).$$

The skeleton *of an AROF f is defined, as before, as the tree of f without the labels (α, β) on the nodes (but with the operations and the input variables).*

The depth *of an AROF is defined as the depth of its tree. In other words, the length of the longest path from a leaf to the root.*

Definition 3.6. *For a ROP P we define $\mathcal{AROF}(P) \stackrel{\Delta}{=} \{f | f$ is an AROF computing $P\}$.*

Example 3.7. *The polynomial computed in Example 3.4 has an alternating ROF of depth 1 that contains a single MUL gate.*

3.1 The Gates-Graph of ROFs and ROPs

In this section we define the important notion of *gates-graph* of a ROF and a ROP. A similar notion was defined in [HH91], but here we define it in a slightly more general form (so it is defined for alternating ROFs as well).

Given a graph of computation of a ROF it is natural to define the first common gate of a subset of the gates. A similar notion was presented in [BHH95a].

Definition 3.8. *Let $V \subseteq \mathrm{var}(f)$, $|V| \geq 2$ be a subset of the input variables of a ROF f. We define the* first common gate *of V, $\mathrm{fcg}(V)$, to be the first gate in the graph of computation of f 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 looking at the formula as a tree. We now define for every read-once formula several new graphs that are related to it. The notion of a *t-graph* of a ROF (where t is a type of a gate) was introduced and studied in [HH91] for various types of gates (such as threshold gates, modular gate, division gates etc.). We shall focus on $+, \times$ and MUL gates as these are the only gates appearing in our ROF's. We now give the definition of a t-graph that is slightly different from the one in [HH91].

Definition 3.9 (t-graph)**.** *Let f be a ROF in the input variables* $\mathrm{var}(f)$*. Let $t \in \{+, \times, MUL\}$ be a possible label of an internal node. The t-graph of the formula f is an undirected graph whose vertex set is* $\mathrm{var}(f)$ *and its edges are defined as follows: there is an edge (i, j) if and only if the arithmetic operation that labels the gate* $\mathrm{fcg}(x_i, x_j)$ *is t. We denote this graph with G_f^t.*

The following simple lemma (whose basic form can be found in [HH91]) gives some basic properties of t-graphs. We state it here (mainly) for alternating ROF's. We omit the proof as it is implicit in the proof of Lemma 3 in [HH91].

Lemma 3.10. *The graphs satisfy the following relations.*

1. *Let P be a ROP. Let A be an AROF computing P and let f be a ROF computing P. Then $G_f^\times = G_A^{MUL}$. In particular, the graphs $G_P^\times \triangleq G_f^\times$ and $G_P^+ \triangleq G_f^+$ are well defined. As a result, we shall focus on the graphs of the ROPs from now on instead of the graphs of the ROFs computing them.*
2. *For every ROP P we have $G_P^\times = \left(G_P^+\right)^c$. That is, the addition and the multiplication graphs of each ROF f for P are simply complements to each other.*
3. *For every (alternating) ROF f exactly one of (G_f^{MUL}) $G_f^\times$ and G_f^+ is disconnected. More precisely, the type of the top gate of f is t if and only if G_f^t is connected. Alternatively, the type of the top gate of f is t if and only if $\left(G_f^t\right)^c$ is disconnected.*

We define the *gates-graph* of a ROF f as $G_f \triangleq G_f^\times$. Similarly, we define G_P the gates-graph of a ROP P. The proof of the following claim is not difficult.

Lemma 3.11. *Let P be a ROP. Then there is an edge $(i, j) \in G_P$ 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 $\prod_{i \in I} x_i$.*

Next, we bring an example of a family of multilinear polynomials that are not ROPs.

Example 3.12. *The polynomial $P(x_1, x_2, \ldots, x_n) = x_1x_2 + x_2x_3 + \cdots + x_{n-1}x_n$, $n \geq 4$ is not a ROP. Indeed, assume the contrary. Then by Lemma 3.11 we get that both the $+$-graph and the $\times$-graph are connected for this polynomial. This contradicts Lemma 3.10.*

The following lemma (a variation of Lemma 3 in [HH91]) demonstrates the information that lies inside the gates-graph by showing that from G_P we can reconstruct a skeleton of an alternating ROF f computing P. That is, we can construct a tree and label the internal nodes and the input variables, such that there exists a labelling of a form (α, β) for the nodes and multiplicative ROPs for the MUL gates for which the resulting ROF computes P.

Lemma 3.13. *Given the graph G_P of a read-once polynomial P we can construct the skeleton of an alternating read-once formula computing P.*

Proof. The proof is by induction on the number of variables in P which we denote by n (we shall assume w.l.o.g. that P depends on all n variables). The claim is trivial for $n = 1$. For $n > 1$ we have two cases. Either G_P is connected or not. We shall describe only the case that the graph is connected as the proof is similar for the disconnected case. As G_P is connected we get by Lemma 3.10 that the top gate of any alternating ROF for P is MUL. Consider the complement

graph G_P^c. By the same lemma we see that it is disconnected, in particular it can be broken up to 2 or more connected components. Consider now an alternating ROF f for P. By Lemma 3.10 we get that $G_f = G_P$. In particular the top gate of f is MUL and there is a level of $+$ gates below it. Assume that the fan-in of the top gate in f is k. Then there exist a multiplicative ROF, M, such that $P = M(p_1, p_2, \ldots, p_k)$, where each p_i is a ROP computed by one of the $+$ gates below the top gate of f. By Lemma 3.10 we get that there is a one to one correspondence between the connected components of G_f^+ to the f_i's. Namely the variables appearing in p_i form a connected component in G_f^+. By the induction hypothesis we can reconstruct the skeleton for all the p_i's from G_{p_i} (which can be deduced from G_P). □

The following lemma is also an easy corollary of Lemma 3.10.

Lemma 3.14 (ROP Representation Lemma)**.** *Every ROP $P(x)$ such that $|\mathrm{var}(P)| \geq 2$ can be presented in* exactly *one of the following forms:*

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

When P_1, P_2 are non-constant variable disjoint ROPs and c is a constant.

Proof. Let f be a ROF computing P. Let v be the top gate of f and f_1, f_2 be the inputs of v. Clearly, f_1,f_2 are variable-disjoint ROFs. Now, If v is an addition gate then $f = a \cdot (f_1 + f_2) + b = (a \cdot f_1) + (a \cdot f_2 + b)$. We set $P_1 \stackrel{\Delta}{=} a \cdot f_1$, $P_2 \stackrel{\Delta}{=} a \cdot f_2 + b$ and we are done. Otherwise, v is multiplication gate. Therefore $f = a \cdot (f_1 \cdot f_2) + b$ and we proceed similarly. Notice, that P_1 and P_2 are not necessarily unique, however P can not be represented in both forms 1 and 2 according to Lemma 3.10. □

The following is another simple claim regarding representations of ROFs.

Lemma 3.15. *Let $P(\bar{x})$ be a ROP and v a node in a ROF f computing P. We denote by $p_v(\bar{x})$ the polynomial that is computed by v. Then there exists a polynomial $Q(y, \bar{x})$ such that $Q(p_v(\bar{x}), \bar{x}) \equiv P(\bar{x})$ and, in addition, p_v and Q are variable-disjoint ROPs.*

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

3.2 Factors of ROPs

A nice property of ROPs is that every factor of a ROP is a ROP itself. We can use this fact to develop an efficient factoring algorithm for ROPs.

Lemma 3.16. *Every factor of a ROP is a ROP.*

Proof. Let $P(\bar{x}) = h_1(\bar{x}) \cdot h_2(\bar{x}) \cdot \cdots \cdot h_t(\bar{x})$ be a reducible ROP and its irreducible factors, respectively ($t \geq 2$). According to Lemma 3.14 P can be in exactly one of the following two forms:

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. Notice that the h_k-s are variable disjoint. Consequently there exist x_i, x_j and distinct factors $h_{k_1} \neq h_{k_2}$ such that P_1 and h_{k_1} depend on x_i and, P_2 and h_{k_2} depend on x_j. Otherwise all the variables of P will be in the same factor thus implying $t = 1$. Assume w.l.o.g. that $i = k_1 = 1,\ j = k_2 = 2$. Now, on one hand $\frac{\partial^2 P}{\partial x_1 \partial x_2} = \frac{\partial^2}{\partial x_1 \partial x_2}(P_1 + P_2) = 0$ since P_1 and P_2 are variable-disjoint. On the other hand $\frac{\partial^2 P}{\partial x_1 \partial x_2} = \frac{\partial h_1}{\partial x_1} \cdot \frac{\partial h_2}{\partial x_2} \cdot h_3 \cdot \cdots \cdot h_t$. As $\frac{\partial h_1}{\partial x_1}, \frac{\partial h_2}{\partial x_2}$ and other factors are non-zero, we reach 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 P_1 and h_1 depend on x_1, P_2 and h_2 depend on x_2 and hence $\frac{P}{h_1}$ does **not** depend on x_1. We get that (Lemma 2.14)

$$0 = h_1^2 \cdot \frac{\partial}{\partial x_1}\left(\frac{P}{h_1}\right) = h_1 \cdot \frac{\partial P}{\partial x_1} - P \cdot \frac{\partial h_1}{\partial x_1} = P_2 \cdot \left(h_1 \cdot \frac{\partial P_1}{\partial x_1} - \frac{\partial h_1}{\partial x_1} \cdot P_1\right) - c\frac{\partial h_1}{\partial x_1}.$$

   By taking a partial derivative with respect to x_2 we get that

$$\frac{\partial P_2}{\partial x_2} \cdot \left(h_1 \cdot \frac{\partial P_1}{\partial x_1} - \frac{\partial h_1}{\partial x_1} \cdot P_1\right) = 0.$$

   As we assumed that P_2 depends on x_2 we get that $\left(h_1 \cdot \frac{\partial P_1}{\partial x_1} - \frac{\partial h_1}{\partial x_1} \cdot P_1\right) = 0$. 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.

□

3.3 Partial Derivatives of ROPs

The following is a more algebraic definition of the gates-graph of a ROP P. The proof is immediate from Lemma 3.11.

Lemma 3.17. *Let $P(x_1, \ldots, x_n)$ be a ROP. Then (i, j) is an edge in the gates-graph G_P if and only if $\frac{\partial^2 P}{\partial x_i \partial x_i} \not\equiv 0$. 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.*

Example 3.18. *For every $n \geq 3$, the polynomial $P(x_1, x_2, \ldots, x_n) = \sum_{i \neq j} x_i x_j$ is not a ROP.*

The following is another important property of ROPs.

Lemma 3.19. *A partial derivative of a ROP is a ROP*

Proof. Let P be a ROP and $i \in [n]$. We prove the claim by induction on $k = |\mathrm{var}(P)|$. For $k = 0, 1$ the claim is trivial. For $k \geq 2$ we get by Lemma 3.14 that P can be in a one of the two forms:

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.

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

□

We now give a very useful property of the derivatives of ROPs.

Lemma 3.20 (2-nd Derivative Lemma)**.** *Let $P(x_1, x_2, \ldots, x_n)$ be a ROP such that $\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0$ and $\frac{\partial^2 P}{\partial x_i \partial x_j}|_{x_k=\alpha} \equiv 0$ for some $i \neq j \neq k$ and $\alpha \in \mathbb{F}$. Then either $\frac{\partial P}{\partial x_i}|_{x_k=\alpha} \equiv 0$ or $\frac{\partial P}{\partial x_j}|_{x_k=\alpha} \equiv 0$ holds.*

Proof. First, notice that $x_j, x_i \in \text{var}(P)$. Let $f \in \mathcal{ROF}(P)$ and let $v = \text{fcg}\{x_j, x_i\}$. Let $G(\bar{x})$ be the polynomial computed by v in f. From Lemma 3.15 there exists a ROP $Q(y, \bar{x})$ such that $Q(G(\bar{x}), \bar{x}) \equiv P(\bar{x})$. Clearly, $x_j, x_i \in \text{var}(G) \setminus \text{var}(Q)$. As $\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0$, it follows that v is a multiplication gate. By the definition of fcg and Lemma 3.14 we obtain that $G(\bar{x})$ can be presented as $P_1 \cdot P_2 + c$ where $x_i \in \text{var}(P_1), x_i \in \text{var}(P_2)$ and $c \in \mathbb{F}$. By the chain rule (Lemma 2.14):

$$\begin{aligned} \frac{\partial P}{\partial x_i} &= \frac{\partial Q}{\partial y} \cdot \frac{\partial G}{\partial x_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2 \\ \frac{\partial P}{\partial x_j} &= \frac{\partial Q}{\partial y} \cdot \frac{\partial G}{\partial x_j} = \frac{\partial Q}{\partial y} \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j}. \end{aligned}$$

Consequently, $\frac{\partial^2 P}{\partial x_i \partial x_j} = \frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j}$. The following is an easy consequence of the above.

$$\begin{gathered} \frac{\partial P}{\partial x_i}|_{x_k=\alpha} \cdot \frac{\partial P}{\partial x_j}|_{x_k=\alpha} = \\ (\frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot P_2)|_{x_k=\alpha} \cdot (\frac{\partial Q}{\partial y} \cdot P_1 \cdot \frac{\partial P_2}{\partial x_j})|_{x_k=\alpha} = \\ (\frac{\partial Q}{\partial y} \cdot \frac{\partial P_1}{\partial x_i} \cdot \frac{\partial P_2}{\partial x_j})|_{x_k=\alpha} \cdot (\frac{\partial Q}{\partial y} \cdot P_1 \cdot P_2)|_{x_k=\alpha} = \\ \frac{\partial^2 P}{\partial x_i \partial x_j}|_{x_k=\alpha} \cdot (\frac{\partial Q}{\partial y} \cdot P_1 \cdot P_2)|_{x_k=\alpha} \equiv 0 \end{gathered}$$

In particular, either $\frac{\partial P}{\partial x_i}|_{x_k=\alpha} \equiv 0$ or $\frac{\partial P}{\partial x_j}|_{x_k=\alpha} \equiv 0$. □

Example 3.21. *The polynomial $P(x_1, x_2, x_3) = x_1 x_2 x_3 + x_1 + x_2$ is not a ROP. To see this apply Lemma 3.20 on P with the parameters $i = 1, j = 2, k = 3, \alpha = 0$.*

3.4 Multiplicative and $\bar{0}$−Justified ROPs

Recall that multiplicative ROFs are ROFs with no addition gates. In the terms of the gates-graph this means that the graph is a clique. We would like to explore the algebraic properties of the polynomials computed by those ROFs. Note, that the definition is very abstract - meaning that given a ROP P it is unclear how to find a multiplicative ROF f for it, or even determine whether one exists. Nonetheless, we can use the fact that all gates-graphs of the ROFs in $\mathcal{ROF}(P)$ are the same, and thus provide a an algebraic definition resembling Definition 3.17. The proof is an immediate consequence of Lemmas 3.11 and 3.17.

Claim 3.22. *A ROP P is a multiplicative ROP if and only if for any two variables $x_i \neq x_j \in \text{var}(P)$ we have that $\frac{\partial^2 P}{\partial x_i \partial x_j} \neq 0$.*

From now on, whenever we discuss multiplicative ROP we shall use the property described in the claim as an alternative definition. The following lemma gives a very useful property of multiplicative ROP that we shall rely on later.

Lemma 3.23. *Let $P(x_1, x_2, \ldots, x_n)$ be a multiplicative ROP with $|\text{var}(P)| \geq 2$. Then for every variable $x_i \in \text{var}(P)$ there exists another variable $x_j \in \text{var}(P)$ such that $\frac{\partial P}{\partial x_j} = (x_i - \alpha)h(\bar{x})$ for some $\alpha \in \mathbb{F}$ and h, when $\text{var}(h) = \text{var}(P) \setminus \{x_i, x_j\}$. In particular, $\frac{\partial P}{\partial x_j}|_{x_i=\alpha} \equiv 0$.*

Proof. Let $f \in \mathcal{ROF}(P)$ be a multiplicative ROF. As $|\text{var}(f)| = |\text{var}(P)| \geq 2$, f has at least one gate. Assume, w.l.o.g. that $i = 1$. Let v be the unique entering gate of x_1. We denote by $p_v(\bar{x})$ the ROP that is computed by v. Let $\text{var}(p_v) = [k]$ for some $2 \leq k \leq n$. By Lemma 3.15 there exists some polynomial $Q(y, x_{k+1}, \ldots, x_n)$ such that $Q(p_v(x_1, x_2, \ldots, x_k), x_{k+1}, \ldots, x_n) \equiv P(x_1, x_2, \ldots, x_n)$. Since v is a multiplication gate (recall that f is a multiplicative ROF) and is the entering gate of x_1 we get, in a similar manner to Lemma 3.14, that p_v can be written as $p_v(\bar{x}) = (x_1 + \beta)H(\bar{x}) + c$ for some polynomial $H(\bar{x})$ such that $x_1 \notin \text{var}(H)$. By the chain rule: (Lemma 2.14): $\frac{\partial P}{\partial x_2} = \frac{\partial Q}{\partial y} \cdot \frac{\partial p_v}{\partial x_2} = \frac{\partial Q}{\partial y} \cdot (x_1 + \beta) \cdot \frac{\partial H}{\partial x_2}$. As a result we get that $\frac{\partial P}{\partial x_2}|_{x_1=-\beta} \equiv 0$. □

The following lemma is an extension of Lemma 3.19 to $\bar{0}-$weakly-justified ROPs.

Lemma 3.24. *A partial derivative of a $\bar{0}-$weakly-justified ROP is a $\bar{0}-$weakly-justified ROP.*

Proof. Let P to be a $\bar{0}-$weakly-justified ROP. From Lemma 3.19 it only remains to show that the partial derivative of P is $\bar{0}-$weakly-justified. Assume for a contradiction that for some $i \in [n]$, we have that $\frac{\partial P}{\partial x_i}$ is not $\bar{0}-$weakly-justified. That is, there exist some $j, k \in [n]$ such that $\frac{\partial P}{\partial x_i}$ depends on x_j however $\frac{\partial P}{\partial x_i}|_{x_k=0}$ does not. In other words, we have that: $\frac{\partial^2 P}{\partial x_i \partial x_j} \not\equiv 0$ and $\frac{\partial^2 P}{\partial x_i \partial x_j}|_{x_k=0} \equiv 0$. Note that $\{x_i, x_j\} \subseteq \text{var}(P)$. Lemma 3.20 implies that either $\frac{\partial P}{\partial x_i}|_{x_k=0} \equiv 0$ or $\frac{\partial P}{\partial x_j}|_{x_k=0} \equiv 0$ holds. On the other hand, $\{x_i, x_j\} \subseteq \text{var}(P|_{x_k=0})$ since P is a $\bar{0}-$weakly-justified ROP and hence $\frac{\partial P}{\partial x_i}|_{x_k=0} \not\equiv 0$ and $\frac{\partial P}{\partial x_j}|_{x_k=0} \not\equiv 0$, in contradiction. □

It is also possible to extend this proof for $\bar{0}-$justified ROPs. In a similar (and simpler) way, we observe the following.

Lemma 3.25. *Every factor of a $\bar{0}-$weakly-justified ROP is a $\bar{0}-$weakly-justified ROP.*

4 ROF Graph-Related Algorithms

In this section we give several basic algorithms for ROFs. Some of the algorithms are small variants of already known algorithms. To slightly simplify the algorithms we assume w.l.o.g that all the ROFs are non-degenerate. We can make this assumption due to the fact that there is a trivial $\mathcal{O}(n)$ algorithm that can convert any ROF f to a non-degenerate ROF f' such that $f \equiv f'$. In fact, this algorithm can also be used as a (non-black box) PIT algorithm for ROFs. We conclude this discussion with two trivial observations. The first is that a non-degenerate ROF f computes the zero polynomial if and only if f is the "zero ROF" i.e $f = \mathcal{C}_0$ (the graph of computation of f is no other than $\mathcal{C}_0$). Secondly, in a non-degenerate ROF, in all the labels (α, β) of the gates, we have that $\alpha \neq 0$. We now give some notations that will be used in our algorithms.

Notation 4.1. *Let G be the graph of computation of a ROF. We think of G as a planar graph (and in particular the notion of "left child" and "right child" of a node are well defined). Let v be a gate in the formula.*

- *p_v - denotes the polynomial computed by v.*
- *v.var - denotes $\text{var}(p_v)$. Note, that it can be computed recursively.*

- $v.\alpha$ - *denotes the value of* α *labelling* v.
- $v.\beta$ - *denotes the value of* β *labelling* 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 labelling* v. *(**IN** - Input, **CONST** - Constant Function (non-degenerate form),* $\times$ *- Multiplication,* $+$ *- Addition, **MUL** - Generalized "Multiplicative ROP" gate.)*

4.1 Reconstruction of a ROF

We first give a very high level description of an algorithm of [HH91, BHH95a] that shows how to reconstruct a ROF given a justifying assignment (an adaptation of Algorithm 3.2.2 in [HH91]). We shall later use this algorithm as a subroutine.

Algorithm 1 Reconstructing ROF $(f, \bar{a})$

Input: ROF f given as a black-box access, justifying assignment $\bar{a}$ of f
Output: ROF $\hat{f}$ such that $\hat{f} \equiv f$.

Step 1: Construct the gates-graph of f (using the justifying assignment or by Lemma 3.17).
Step 2: Construct the skeleton of f (using the justifying assignment).
Step 3: Construct the ROF by recursively constructing its sub-formulas.

The following lemma is an adaptation of Lemma 3 of [BHH95a].

Lemma 4.2 (Lemma 3 of [BHH95a])**.** *Given oracle access to a black-box holding a ROF* f *and a justifying assignment* $\bar{a}$ *of* f *there is a polynomial time algorithm* A *that can reconstruct* f*, that is construct a (non-degenerate) ROF* $\hat{f}$ *such that* $\hat{f} \equiv f$*. A high level description of* A *is given in Algorithm 1.*

4.2 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 3.16. In the proof of the lemma we showed that a ROF f 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 f 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 2 returns a list $S = \{h_i\}$ of the irreducible factors of p_v (the polynomial computed by the node v) and a constant α such that $p_v = \alpha \cdot \prod_i h_i$.

The following lemma gives the analysis of the algorithm. We omit the proof as it is similar to previous proofs.

Lemma 4.3. *Given a node* v *in a graph of computations of a non-constant ROF* f*, Algorithm 2 returns a pair* (S, α) *where* S *is a list of (ROFs computing) the irreducible factors of the polynomial* p_v *and* α *is a constant satisfying* $p_v = \alpha \cdot \prod_{h \in S} h$*. The running time of the algorithm is* $\mathcal{O}(n)$*.*

Algorithm 2 FactorROF(v)

Input: The root v of a non-constant ROF.

Output: (S, α) where S is a list of the irreducible factors of p_v,
α a constant in the field.

1: **if** v.Type = IN **then** {Univariate polynomial}
2: **return** $(p_v\ ,\ 1)$
3: **if** v.Type = $+$ **then**
4: **return** $(p_v\ ,\ 1)$
5: **if** $v.\beta \neq 0$ **then**
6: **return** $(p_v\ ,\ 1)$
{Now v.Type = $\times$ and $v.\beta = 0$, thus $p_v = v.\alpha \cdot p_{v.\mathsf{Right}} \times p_{v.\mathsf{Left}}$}
7: $(S_L\ ,\ \alpha_L) \leftarrow$ FactorROF(v.Left)
8: $(S_R\ ,\ \alpha_R) \leftarrow$ FactorROF(v.Right)
9: **return** (**union**$(S_L\ ,\ S_R)\ ,\ \alpha_{Left} \cdot v.\alpha \cdot \alpha_{Right}$)

4.3 Counting the Number of Monomials in a ROF

The number of monomials in a polynomial P is the number of monomials with non-zero coefficients. We now give an efficient algorithm for counting the number of monomials of a given ROF. The main idea is that once a non-constant monomial appears, it can not be cancelled later. Consequently, we only have to sum the number of non-constant monomials and keep track of the behavior of the constant term.

In order to determine whether a certain value is zero or non-zero we use an auxiliary function $f_{nz}(x)$ that is defined as

$$f_{nz}(x) = \begin{cases} 0 & x = 0 \\ 1 & \text{otherwise} \end{cases} .$$

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

Lemma 4.4. *Given a node v in the graph of computation of a ROF f, Algorithm 3 returns a pair (M, C) when M is the (exact) number of monomials of p_v and C is the constant term of p_v. The running time of the algorithm is $\mathcal{O}(n)$.*

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 will follow from the analysis) that for every v the expression $M - f_{nz}(C)$ stands for the number of non-constant monomials of p_v (where (M, C) is returned after the execution of the algorithm on the node v). 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 labelling v (i.e. v.Type).

- v.Type = CONST: p_v computes the constant function β. Clearly, $C = \beta$ and M can be either 0 or 1 depending on the value of β.
- v.Type = IN: p_v computes a univariate polynomial $v.\alpha \cdot x_i + v.\beta$ for some x_i. The analysis is similar to the previous case.

When v is not a leaf there is recursive call of the function for to the left and right children. Denote with p_L and p_R the polynomials computed by the left child and right child, respectively. We again analyze the different options for v.Type.

Algorithm 3 CountMonomials(v)

Input: The root v of a ROF.
Output: (M, C) where M is the number of monomials of p_v,
C is the constant term of p_v.

1: **if** v.Type = CONST **then** {Constant Function}
2: **return** $(f_{nz}(v.\beta)\ ,\ v.\beta)$
3: **if** v.Type = IN **then** {Univariate polynomial}
4: **return** $(1 + f_{nz}(v.\beta)\ ,\ v.\beta)$
5: $(M_L,\ C_L) \leftarrow$ CountMonomials(v.Left)
6: $(M_R\ ,\ C_R) \leftarrow$ CountMonomials(v.Right)
7: **if** v.Type = $\times$ **then**
8: $C' \leftarrow C_L \cdot C_R$
9: $C \leftarrow v.\alpha \cdot C' + v.\beta$
10: $M \leftarrow M_L \cdot M_R - f_{nz}(C') + f_{nz}(C)$
11: **return** $(M\ ,\ C)$
{Now v.Type = +}
12: $C \leftarrow v.\alpha \cdot (C_L + C_R) + v.\beta$
13: $M \leftarrow M_L + M_R - f_{nz}(C_L) - f_{nz}(C_R) + f_{nz}(C)$
14: **return** $(M\ ,\ C)$

- v.Type = $\times$: p_v computes a product of two functions. The output of p_v is $v.\alpha \cdot p_L \times p_R + v.\beta$ then:
  - The total number of monomials of $p_L \times p_R$ is $M_L \cdot M_R$
  - $C'$is the constant term of $p_L \times p_R$, hence the number of non-constant monomials of $p_L \times p_R$ is $M_L \cdot M_R - f_{nz}(C')$.
  - The polynomial $v.\alpha \cdot p_L \times p_R + v.\beta$ has the same number of non-constant monomials as the polynomial $p_L \times p_R$.
  - C is constant term of $v.\alpha \cdot p_L \times p_R + v.\beta$, hence the total number of monomial of $v.\alpha \cdot p_L \times p_R + v.\beta$ is $M_L \cdot M_R - f_{nz}(C') + f_{nz}(C)$.
- v.Type = +: p_v computes a sum of two functions. The output of v is $v.\alpha \times (p_L + p_R) + v.\beta$ then (similarly):
  - The number of non-constant monomials of $p_L + p_R$ is $M_L - f_{nz}(C_L) + M_R - f_{nz}(C_R)$.
  - C is the constant term of p_v.
  - The total number of monomials of $v.\alpha \cdot (p_L + p_R) + v.\beta$ is $M_L + M_R - f_{nz}(C_L) - f_{nz}(C_R) + f_{nz}(C)$.

It is clear from the above analysis that the algorithm returns a correct answer. The claim regarding the running time follows easily from the fact that the algorithm is a simple graph traversal that requires $\mathcal{O}(n)$ time where at each step we have to multiply two n bits numbers (note that a ROF computes a multilinear polynomial thus $0 \leq M \leq 2^n$). Thus the total running time is $\mathcal{O}(n)$ (we assume that we work in the unit cost model). □

4.4 Computing a Partial Derivative of a ROF

We present a recursive algorithm that computes a partial derivative of a given ROF. Our algorithm is based on Lemma 3.19. In particular, Lemma 3.19 in conjunction with Lemma 2.14 guarantee that the algorithm outputs a ROF.

Algorithm 4 DerriveROF(v,x_i)

Input: The root v of a ROF, a variable x_i.
Output: ROF g such that $g \equiv \frac{\partial f}{\partial x_i}$.

1: **if** $x_i \notin v$.var **then** {The polynomial does not depend on x_i}
2: **return** $\mathcal{C}_0$ {The "zero ROF"}
3: **if** v.Type = IN **then** {Input gate of x_i, i.e. the polynomial is $v.\alpha \cdot x_i + v.\beta$}
4: **return** $\mathcal{C}_{v.\alpha}$
{In both following cases exactly one of the ROF is the "zero ROF" ($\mathcal{C}_0$),
thus each of the returned sums is, in fact, a single ROF.}
5: Left_Derivative $\leftarrow$ DerriveROF(v.Left, i)
6: Right_Derivative $\leftarrow$ DerriveROF(v.Right, i)
7: **if** v.Type = $+$ **then**
8: **return** Left_Derivative + Right_Derivative
{Here, v.Type = $\times$ }
9: **return** Left_Derivative $\times p_{v.\mathsf{Right}} + p_{v.\mathsf{Left}} \times$ Right_Derivative

The following lemma summarizes the properties of the algorithm.

Lemma 4.5. *Given a ROF f and a variable x_i Algorithm 4 outputs a (non-degenerate) ROF g that computes $\frac{\partial f}{\partial x_i}$. The running time of the algorithm is $\mathcal{O}(n)$.*

5 Generalization of PIT Algorithms to Read-Once Testing

In this section we study the relation between the *polynomial identity testing* problem (PIT) and the *read-once testing* problem and prove Theorem 7. We present a generic scheme that can be used to strengthen efficient PIT algorithms to yield efficient read-once testing algorithms. Then we use the scheme to obtain ROT algorithms for models for which PIT algorithms are known.

5.1 Generic Scheme

The idea behind the scheme is: "Doveriai, no Proveriai"[6]. First, we give a *read-once reconstruction* algorithm (based on Algorithms 6 and 1). That is, an algorithm that given a circuit C computing a ROP, outputs a ROF f such that $f \equiv C$. Then, to preform the read-once testing, we assume that the given circuit C computes a ROP and run the read-once reconstruction algorithm based on this assumption. If the algorithm encounters an error or is unable to run correctly, then we conclude that our assumption was wrong (i.e. C *does not* compute a ROP) and thus we stop and report a failure. Things are more complicated in the case of success, that is, when the algorithm does output a ROF. The problem is that we do not have a guarantee regarding the correctness of our assumption (that the circuit computes a ROP) and hence the correctness of its output. Moreover, for any circuit C that computes a ROP there exist many circuits (computing different

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

polynomials) aliasing C. Meaning, that an execution of our read-once reconstruction algorithm on each such circuit $C' \not\equiv C$ will succeed and yield a ROF f such that $f \equiv C \neq C'$. Consequently, to complete the read-once testing we need to *verify* the correctness of the output. For this purpose we need a *verification* procedure (Algorithm 9). Algorithm 5 gives the generic scheme for ROT.[7] The algorithm works both in the black-box and in the non black-box settings, depending on the PIT algorithm for $\mathcal{M}$ at hand.

Algorithm 5 Generic ROT Scheme

Input: A (black-box holding a) circuit C that belongs to $\mathcal{M}$.
Output: ROF f such that $f \equiv C$, if C computes a ROP,
"failure" otherwise.

1: Acquire a justifying assignment $\bar{a}$ using Algorithm 6 (given in Subsection 5.2).
2: Given the justifying assignment $\bar{a}$ reconstruct f from C using Algorithm 1.
3: Verify that $f \equiv C$ using Algorithm 9 (given in Subsection 5.3).

Algorithms 6 and 9 are given in later sections. In Section 5.4 we give the proof of Theorem 7, that basically analyzes the running time of Algorithm 5 given the running times of Algorithms 1, 6 and 9. Then in Section 6 we go one by one over circuit classes that have PIT algorithms and give the corresponding ROT algorithm (as mentioned earlier, in some cases we shall give more efficient ROT algorithms that do not use the verification step of Algorithm 5).

5.2 From PIT to Justifying Assignments

In Subsection 4.1 we showed (by citing the results of [HH91, BHH95b]) that a reconstruction of a ROF is possible when we have a justifying assignment. We now show how to get a justifying assignment from a PIT algorithm. We shall consider a circuit class $\mathcal{M}$ (e.g. depth-3 circuit, sum of ROFs, etc.) for which there exists another circuit class $\mathcal{M}'$ such that $\partial\mathcal{M} \subseteq \mathcal{M}'$ and $\mathcal{M}'$ has an efficient PIT algorithm.[8]. Algorithm 6 returns a justifying assignment for P if it manages to find one. Otherwise it returns "ERROR". The algorithm will invoke (as a routine) the supplied PIT algorithm for $\mathcal{M}'$.

Before giving the algorithm we explain the intuition behind it. What we are after is a vector $(a_1, \ldots, a_n) \in \mathbb{F}^n$ such that if P depends on x_i then the polynomial $P|_{x_j=a_j}$ (for any $j \neq i$) also depends on x_i. In other words we want that if $\frac{\partial P}{\partial x_i} \not\equiv 0$ then also $\frac{\partial P}{\partial x_i}|_{x_j=a_j} \not\equiv 0$. Therefore, we consider the polynomials $\{g_i = \frac{\partial P}{\partial x_i}\}_{i\in[n]}$, and look for a vector $\bar{a}$, such that for any $j \neq i$, $g_i|_{x_j=a_j} \not\equiv 0$. As each g_i is a linear polynomial in x_j there is at most one "bad value" that we can assign to a_j. Namely, for most values of a_j, we have that $g_i|_{x_j=a_j} \not\equiv 0$. Hence, if we check enough values we should find a good a_j for every g_i. To verify that $g_i|_{x_j=a_j} \not\equiv 0$ we use the PIT algorithm for $\mathcal{M}'$. In fact, what we just described only gives a weakly justifying assignment. To find a justifying assignment we have to verify that after we assign a_j to x_j for all $j \neq i$ then g_i is still not zero. We manage to do it by first finding a_1, then we assign a_1 to x_1 to get a new set of polynomials g_i's etc. Actually, we shall acquire a *Common Justifying Assignment* for a set of polynomials $\{P_m\}_{m\in[k]}$. A common justifying assignment is an assignment $\bar{a}$ that is simultaneously a justifying assignment for each P_m. Acquiring a common justifying assignment is a simple generalization of the above ideas. In fact, it can be regarded as a simultaneous acquisition of a (single) justifying

[7] In fact, in most cases we shall give "tailored" algorithms that are somewhat more efficient than the generic algorithm.

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

assignment. Algorithm 6 simply computes the partial derivatives $\{g_i^m \stackrel{\Delta}{=} \frac{\partial P_m}{\partial x_i}\}_{i\in[n],m\in[k]}$ and then invokes Algorithm 7 that finds the required assignment. Note that Algorithm 6 runs Algorithm 7 as a subroutine and through the entire execution g_i^m's and $\bar{a}$ are shared by Algorithm 6 and all recursive calls of Algorithm 7. Lemma 5.1 discusses the properties of Algorithm 6.

Algorithm 6 Acquire Common Justifying Assignment $(P_1, P_2, \ldots, P_k)$

Input: (black-box holding) Circuits $C_1, C_2, \ldots, C_k$ from $\mathcal{M}$ computing $P_1, P_2, \ldots, P_k$,
Subset $V \subseteq \mathbb{F}$ of size $|V| = kn$,
Access to a PIT algorithm for $\mathcal{M}'$ such that $\partial\mathcal{M} \subseteq \mathcal{M}'$.

Output: Common Justifying assignment $\bar{a}$ for $P_1, P_2, \ldots, P_k$ on a success,
"ERROR" on a failure.

$\bar{a} \leftarrow \bar{0}$
for all $m \in [k]$ and $i \in [n]$ **do**
  $g_i^m(\bar{x}) \leftarrow \frac{\partial P_m}{\partial x_i}$
$result \leftarrow$ Fill_Up_to(n) {Invoke Algorithm 7}
if $result =$ true **then**
  return $\bar{a}$ {$\bar{a}$ was computed during the execution of Fill_Up_to(n)}
else
  return "ERROR"

Algorithm 7 Fill_Up_to(j)

if $j = 0$ **then**
  return true
$result \leftarrow$ Fill_Up_to$(j-1)$
if $result =$ false **then**
  return false
for all $c \in V$ **do**
  $flag \leftarrow$ true
  for all $m \in [k]$ and $i \neq j \in [n]$ **do**
    if $g_i^m \not\equiv 0$ and $g_i^m|_{x_j=c} \equiv 0$ **then**
      $flag \leftarrow$ false
  {This is done by running the PIT algorithm for $\mathcal{M}'$}
  if $flag =$ true **then** {That is, c is a good assignment to x_j}
    $a_j \leftarrow c$
    for all $m \in [k]$ and $i \neq j \in [n]$ **do**
      $g_i^m(\bar{x}) \leftarrow g_i^m|_{x_j=c}$
    return true
{Then, if no match was found}
return false

Lemma 5.1. *Assume that $|\mathbb{F}| > kn$. For a set of ROPs $\{P_m\}_{m\in[k]}$ and $|V| \geq kn$ Algorithm 6 returns a common justifying assignment $\bar{a}$ for $\{P_m\}_{m\in[k]}$ in time $\mathcal{O}(n^3k^2 \cdot t)$, where t is the running time of the PIT algorithm for $\mathcal{M}'$.*

Proof. Note that the algorithm computes the partial derivatives and then run Algorithm 7, which is supposed to find an assignment $\bar{a}$ such that for every $m \in [k]$ and $i \in [n]$ satisfy that $g_i^m(\bar{a}) \neq 0$.

We show that Algorithm 7 succeeds and returns the expected result. Due to the recursive nature of the algorithm we use induction to show that at each phase $0 \leq j \leq n$ algorithm $\mathsf{Fill\_Up\_to}(j)$ finds values for the first j coordinates of $\bar{a}$ such that $g_i^m \not\equiv 0$ for every $i \in \mathrm{var}(P_m)$ (note that the polynomials $\{g_i^m\}$ change in every recursive step of the algorithm).

When $j = 0$ the correctness follows from the definitions of g_i^m's and $\mathrm{var}(P_m)$'s. We now deal with the case that $j > 0$. From the induction hypothesis we obtain that $\mathsf{Fill\_Up\_to}(j-1)$ succeeds, and after the completion of this phase we get non-zero polynomials $\{g_i^m\}_{i\in[n],m\in[k]}$ (these polynomials were defined in the $(j-1)$-th phase). Consequently, in order to succeed in j-th phase the algorithm must find $c \in V$ that is good for every $g_i^m \not\equiv 0$. That is, a c such that for every m and $i \neq j$ we have that if $g_i^m \not\equiv 0$ then $g_i^m|_{x_j=c} \not\equiv 0$. Note, that g_i^m's are multilinear polynomials hence from Lemma 2.17 each g_i^m has *at most* one root of the form of $x_j = c$. Therefore, there are at most $k(n-1)$ "bad" values of c (e.g. values for which there exist m and $i \neq j$ with $g_i^m \not\equiv 0$ and $g_i^m|_{x_j=c} \equiv 0$). Consequently, V contains at least one good value of c. We now notice that after $\mathsf{Fill\_Up\_to}(n)$ is (successfully) completed we have, for every i and m, that $g_i^m = \frac{\partial P_m}{\partial x_i}(\bar{a}) \neq 0$. This follows from the definition of the g_i^m's and the fact that at each phase of $\mathsf{Fill\_Up\_to}(j)$ we substitute a_j to x_j in every g_i^m. This ensures that $\bar{a}$ is indeed a common justifying assignment.

Next we analyze the running time. Algorithm 6 first computes $\{g_i^m\}_{i\in[n],m\in[k]}$. This can be done in $\mathcal{O}(nk)$ time. The execution of $\mathsf{Fill\_Up\_to}(j)$, requires for each $c \in V$ to perform $2k(n-1)$ PIT checks, [9] given that we have the output of $\mathsf{Fill\_Up\_to}(j-1)$. Thus for every j, $\mathsf{Fill\_Up\_to}(j)$ runs at most $2k^2n^2$ PIT checks. Therefore, we do at most $2k^2n^3$ PIT checks during the execution of $\mathsf{Fill\_Up\_to}(n)$. Hence the total running time of the algorithm is $\mathcal{O}(k^2n^3 \cdot t)$, where t is the cost of every PIT check for a circuit in $\mathcal{M}'$. □

5.2.1 From a Hitting Set to a Justifying Set

Note that in Algorithm'6, even if the PIT for $\mathcal{M}'$ is in the black-box setting then the algorithm still runs in an adaptive manner. In this subsection we give a "Black-Box" version of Algorithm 6. More precisely, given a hitting set A for $\mathcal{M}'$ we construct a *$k-$justifying set* for $\mathcal{M}$. That is, a set of vectors that contains a common justifying assignment for any set of k ROPs computed by $\mathcal{M}$. The construction is performed in two steps: First, we interpolate the vectors in A by multivariate polynomials (i.e. take a low degree extension of A). Afterwards, we expand the set A by evaluating those polynomials on more points.

Given a hitting set $A \subseteq \mathbb{F}^n$ we chose a subset of $V \subseteq \mathbb{F}$ of size n and set $q \stackrel{\Delta}{=} \lceil \log_n |A| \rceil$. It follows that

$$|A| \leq n^q < n|A|.$$

Then, we arbitrarily order the vectors in A. Let $A = \{\bar{a}^1, \bar{a}^2, \ldots, \bar{a}^{|A|}\}$ where each $\bar{a}^j = (a_1^j, a_2^j, \ldots, a_n^j)$. Let $\varphi : V^q \to \{1, 2, \ldots, |A|\} \subseteq \mathbb{N}$ be a surjection.

Definition 5.2. *For every $i \in [n]$ we define $L_i(\bar{y}) : \mathbb{F}^q \to \mathbb{F}^n$ to be the interpolation polynomial of the i-th coordinates of the vectors in A. That is, $L_i(\bar{y})$ is a q-variate polynomial of degree at most $n-1$ in every variable such that for every $\bar{b} \in V^q$ we have that*

$$L_i(\bar{b}) = a_i^{\varphi(\bar{b})}.$$

Let $L(\bar{y}) : \mathbb{F}^q \to \mathbb{F}^n$ be defined as

$$L(\bar{y}) \stackrel{\Delta}{=} (L_1(\bar{y}), L_2(\bar{y}), \ldots, L_n(\bar{y})).$$

[9] Actually we can save a little and only run $k(n-1)$ PIT checks but it does not really matter

Given a hitting set A and a positive integer k Algorithm 8 returns a k-justifying set, $\mathcal{J}_A^k$, for ROPs computed by circuits from $\mathcal{M}$. As before, to obtain a justifying set ($\mathcal{J}_A = \mathcal{J}_A^1$) for a single ROP simply run the algorithm for $k = 1$.

Algorithm 8 Construct $k-$Justifying Set (A, k)

Input: Integer $k \geq 1$,
Hitting set A for $\mathcal{M}'$ such that $\partial\mathcal{M} \subseteq \mathcal{M}'$,
Subsets $V \subseteq W \subseteq \mathbb{F}$ of sizes $|V| = n, |W| = kn^3$.
Output: The $k-$justifying set $\mathcal{J}_A^k$ for $\mathcal{M}$.

1: **for all** $i \in [n]$ **do**
2: **compute** L_i using V as in Definition 5.2.
3: $\mathcal{J}_A^k \leftarrow L(W^q)$ { The image of W^q under L }
4: **return** $\mathcal{J}_A^k$

Note, that indeed for every k we have that $A \subseteq \mathcal{J}_A^k$, and that L_i's can be computed in time polynomial in $|A|$ using simple interpolation. We now prove the correctness of the algorithm. We start by a trivial observation which follows immediately from the definition of L.

Observation 5.3. *For each $\bar{a} \in A$ there exists $\bar{b} \in V^q$ such that $L(\bar{b}) = \bar{a}$.*

The crucial property of L, as induced from A, is given in the following proposition.

Proposition 5.4. *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree at most d computed by a circuit in $\mathcal{M}'$. Then $P(L(\bar{y}))$ is a non-zero polynomial in $(y_1, y_2, \ldots, y_q)$, and the degree of each y_j is at most $d \cdot (n-1)$.*

Proof. From the definition of A there exists $\bar{a} \in A$ such that $P(\bar{a}) \neq 0$. From the observation we obtain that there exists $\bar{b} \in V^q$ such that $L(\bar{b}) = \bar{a}$. In particular, $P(L(\bar{b})) = P(\bar{a}) \neq 0$ and hence $P(L(\bar{y}) \not\equiv 0$. By definition, the degree of every L_i in each y_j is at most $n-1$. Consequently, the degree of $P(L(\bar{y}))$ in every y_j is at most $d(n-1)$. □

Next lemma shows the correctness of Algorithm 8. Note that a ROP is a multilinear polynomial and hence its degree is bounded by n.

Lemma 5.5. *Let $P_1, P_2, \ldots, P_k$ be ROPs of degree d computed by circuits from $\mathcal{M}$. Then $\mathcal{J}_A^k$ contains a common justifying assignment for $P_1, P_2, \ldots, P_k$. Moreover, $|\mathcal{J}_A^k| \leq n^3 k^q \cdot |A|^3$.*

Proof. Let $\{g_i^m(\bar{y}) \stackrel{\Delta}{=} \frac{\partial P_m}{\partial x_i}(L(\bar{y}))\}_{i\in[n],m\in[k]}$. As $\partial\mathcal{M} \subseteq \mathcal{M}'$ we get by Proposition 5.4 that if $\frac{\partial P_m}{\partial x_i} \not\equiv 0$ then $g_i^m \not\equiv 0$. Consider the polynomial $g(\bar{y}) \stackrel{\Delta}{=} \prod_{i,m|g_i^m\not\equiv 0} g_i^m(\bar{y})$. It follows that g is a non-zero q-variate polynomial of degree at most $kn \cdot d(n-1) < kn^3$ in each variable. Lemma 2.16 implies that $g|_{W^q} \not\equiv 0$. Equivalently, there exists $\bar{\alpha} \in W^q$ such that for each pair $i \in [n], m \in [k]$ if $g_i^m \not\equiv 0$ then $g_i^m(\bar{\alpha}) \neq 0$. Now, let $i \in [n], m \in [k]$ be such that $\frac{\partial P_m}{\partial x_i} \not\equiv 0$. Proposition 5.4 implies that $g_i^m(\bar{y}) = \frac{\partial P_m}{\partial x_i}(L(\bar{y})) \not\equiv 0$. Consequently, we get that $\frac{\partial P_m}{\partial x_i}(L(\bar{\alpha})) = g_i^m(\bar{\alpha}) \neq 0$ and hence $L(\bar{\alpha})$ is a justifying assignment for every P_m.

The claim regarding the size of $|\mathcal{J}_A^k|$ follows immediately from the bound $|A| \leq n^q < n|A|$. Calculating we get $|\mathcal{J}_A^k| = (kn^3)^q \leq n^3 k^q \cdot |A|^3$. □

5.2.2 $\bar{0}-$Justification

By the definition, if $\bar{a}$ is a justifying assignment of a function (or a set of functions) then the function is $\bar{a}-$justified. In many cases, though, it would be much convenient to deal with $\bar{0}-$justified functions rather then any other $\bar{a}$. For this purpose we can simply convert *any* $\bar{a}-$justified function $f(x_1, x_2, \ldots, x_n)$ into $\bar{0}-$justified by shifting the variables in the following way: $f_{\bar{a}}(x_1, x_2, \ldots, x_n) \stackrel{\Delta}{=} f(x_1 + a_1, x_2 + a_2, \ldots, x_n + a_3)$. In addition, note that $f_{\bar{b}} \equiv 0 \iff f \equiv 0$, for any $\bar{b} \in \mathbb{F}^n$. Algorithmically, in order to commonly convert a set of functions $f_1, f_2, \ldots, f_k$ into $\bar{0}-$justified we can use Algorithm 6 and return $(f_1, f_2, \ldots, f_k)_{\bar{a}}$ (shifting all the functions) for the acquired $\bar{a}$ - upon success and "ERROR" otherwise. Note that the asymptotic time complexity remains the same.

In the black-box version we use Algorithm 8 to obtain a $k-$justifying set $\mathcal{J}_A^k$, and afterwards produce $|\mathcal{J}_A^k|$ shifted instances of $f_1, f_2, \ldots, f_k$ by returning $\{(f_1, f_2, \ldots, f_k)_{\bar{a}}\}_{\bar{a} \in \mathcal{J}_A}$. By the definition of $\mathcal{J}_A^k$ at least one of the produced instances must be $\bar{0}-$justified .

5.3 Read-Once Verification

Read-Once Verification is testing whether a given circuit C, from a certain circuit class $\mathcal{M}$, and a given ROF f compute the same polynomial. Note though, that while the verification might have the nature of a polynomial identity testing, it is somewhat a harder problem since it requires determining the equivalence of polynomials computed by circuits from two different circuit classes. We shall work under the assumption that $\mathcal{M}$ has a PIT algorithm and the ROF f is given to us explicitly (e.g. as a graph of computations). The circuit C, on the other hand, may be given to us as a black-box, depending on the PIT algorithm for $\mathcal{M}$. Clearly, if $C - f \in \mathcal{M}$ then the verification procedure is trivial (as $\mathcal{M}$ has a PIT algorithm), the general case however is more complicated. We shall present a verification procedure that enables us to take care of the case where $C - f \notin \mathcal{M}$. The idea behind the algorithm is to recursively ensure that every gate v of f computes the same polynomial as a "corresponding" restriction of C. To give a rough sketch, we first find a justifying assignment for f, $\bar{a}$. Then we consider v, the root of f. It has two sub formulas. W.l.o.g. assume that $f = \alpha \cdot (f_{v_1}\ op\ f_{v_2}) + \beta$, where f_{v_i} is the ROF 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 equals to C after we substitute the corresponding values from $\bar{a}$ to the variables in S_2. Similarly we define C_2, and the ROFs[10] f_1 and f_2. We now recursively verify that $C_i \equiv f_i$. The only thing left now is to verify that indeed $C \equiv \alpha \cdot (C_1\ op\ C_2) + \beta$. This basically reduces the verification problem to the problem of verifying that $C \equiv C_1\ op\ C_2$ where C_1 and C_2 compute variable disjoint ROPs and op is either $+$ or $\times$. Note, that this is a PIT problem for a circuit class that is closely related to $\mathcal{M}$, although slightly different (e.g. if $\mathcal{M}$ is the class of $\Sigma\Pi\Sigma(k)$ circuits, defined in Section 6.2, then we need a PIT algorithm for $\Sigma\Pi\Sigma(\mathcal{O}(k^2))$ circuits). Therefore, we shall assume that a slightly larger circuit class has an efficient PIT algorithm (e.g. a class containing $C - C_1\ op\ C_2$ 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. Note, that for most circuit classes that have efficient PIT algorithms, there also exists a PIT algorithm for a corresponding verifying class.

Definition 5.6. *We call a circuit class $\mathcal{M}_V$ a* Verifying Class *of $\mathcal{M}$ if*

1. $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{M}_V) \subseteq \mathcal{M}_V$.
2. *If $P \in \mathcal{L}(\mathcal{M})$ then $P + x_i \in \mathcal{M}_V$, for every $i \in [n]$.*

[10] In fact, in the algorithm it will be more convenient to have $f_i = f_{v_i}$ and so we will slightly change the definition of C_i.

3. *If $P_1, P_2, P_3 \in \mathcal{L}(\mathcal{M})$ then $P_1 + P_2 + P_3 \in \mathcal{M}_V$, for **variable disjoint** P_2, P_3.*

4. *If $P_1, P_2, P_3 \in \mathcal{L}(\mathcal{M})$ then $P_1 + P_2 \times P_3 \in \mathcal{M}_V$ for **variable disjoint** P_2, P_3.*

5. *$\mathcal{M}_V$ has an efficient PIT algorithm.*

Algorithm 9 Verify (C, f)

Input: Circuit C from a circuit class $\mathcal{M}$,
The root v of the a ROF f,
Justifying assignment $\bar{a}$ for f,
Access to a PIT algorithm for $\mathcal{M}_V$.
Output: "true" if $C \equiv f$ and "false" otherwise.

1: **if** v.Type = IN **OR** v.Type = CONST **then** {p_v is a univariate or constant polynomial}
2: **if** $C - p_v \not\equiv 0$ **then** {Using the PIT algorithm for $\mathcal{M}_V$}
3: **return** false
{Multiplication Gate}
4: **if** v.Type = $\times$ **then**
5: $C_L \leftarrow (C_{\bar{a}}^{(v.\mathsf{Right}).\mathrm{var}} - v.\beta)/(v.\alpha \cdot p_{v.\mathsf{Right}}(\bar{a}))$
6: $C_R \leftarrow (C_{\bar{a}}^{(v.\mathsf{Left}).\mathrm{var}} - v.\beta)/(v.\alpha \cdot p_{v.\mathsf{Left}}(\bar{a}))$
7: **if** Verify(C_L, v.Left) = false **OR** Verify(C_R, v.Right) = false **then**
8: **return** false
9: **if** $C - v.\alpha \cdot (C_L \times C_R) + v.\beta \not\equiv 0$ **then** {Using the PIT algorithm for $\mathcal{M}_V$}
10: **return** false
{Addition Gate}
11: **if** v.Type = $+$ **then**
12: $C_L \leftarrow (C_{\bar{a}}^{(v.\mathsf{Right}).\mathrm{var}} - v.\beta)/v.\alpha - p_{v.\mathsf{Right}}(\bar{a})$
13: $C_R \leftarrow (C_{\bar{a}}^{(v.\mathsf{Left}).\mathrm{var}} - v.\beta)/v.\alpha - p_{v.\mathsf{Left}}(\bar{a})$
14: **if** Verify(C_L, v.Left) = false **OR** Verify(C_R, v.Right) = false **then**
15: **return** false
16: **if** $C - v.\alpha \cdot (C_L + C_R) + v.\beta \not\equiv 0$ **then** {Using the PIT algorithm for $\mathcal{M}_V$}
17: **return** false
{Everything is OK}
18: **return** true

Theorem 5.7. *Let C and f be a circuit from a circuit class $\mathcal{M}$, and a ROF correspondingly. In addition, let $\mathcal{M}_V$ be a verifying class of $\mathcal{M}$ and $\bar{a}$ a justifying assignment of f. Then given $C, \bar{a}$ and f Algorithm 9 runs in time $\mathcal{O}(n \cdot t)$, when t is the cost of a single PIT algorithm for $\mathcal{M}_V$, and outputs "true" if and only if $C \equiv f$.*

Proof. We start by analyzing the running time. As described above, the algorithm preforms a traversal over the tree of f. The PIT algorithm of $\mathcal{M}_V$ is invoked once for every gate (Input, Multiplication, Addition) hence the total running time is $\mathcal{O}(n \cdot t)$ when t is the cost of a single PIT algorithm for $\mathcal{M}_V$. We now prove the correctness of the algorithm.

There are two things to prove. First, we show that all the PIT calls that we make are indeed for circuits from $\mathcal{M}_V$. Then, we show that given a PIT algorithm for $\mathcal{M}_V$ the algorithm returns the correct answer.

From the first glance, we might expect "hazards" executing the lines which require an invocation of a PIT algorithm. That is, lines 2, 9 and 16. At each stage C and C_L, C_R (if defined) compute polynomials in $\mathcal{L}(\mathcal{M})$. Moreover, whenever the algorithm reaches line 9 it must be the case that the polynomials computed by C_L and C_R are *variable disjoint*. To see that, notice that the algorithm could reach line 9 only upon a successful outcome in line 7. That is, prior to line 9 we have that $C_L \equiv p_v.\mathsf{Left}$ and $C_R \equiv p_v.\mathsf{Right}$ and hence

$$(\mathrm{var}(C_L) = \mathrm{var}(p_v.\mathsf{Left})) \cap (\mathrm{var}(C_R) = \mathrm{var}(p_v.\mathsf{Right})) = \varnothing,$$

as $p_v.\mathsf{Left}$ and $p_v.\mathsf{Right}$ are variable disjoint by definition. The same holds for line 16. Having the above, we show that all the required identity tests can be, indeed, carrier out. Recall that (we assume w.l.o.g. that) f in a non-degenerate ROF and hence $v.\alpha \neq 0$ for every gate v.

- Line 2:
  $C \in \mathcal{L}(\mathcal{M})$ and $p_v = a \cdot x_i + b$ for some $i \in [n]$ and $a, b \in \mathbb{F}$. Consequently,
  $-a^{-1} \cdot C \in \mathcal{L}(\mathcal{M}) \implies -a^{-1} \cdot C + x_i \in \mathcal{M}_V \implies C - p_v = C - a \cdot x_i - b \in \mathcal{M}_V$. [11]

- Line 9:
  $C, C_L, C_R \in \mathcal{L}(\mathcal{M})$ and C_L, C_R are variable-disjoint. Consequently,
  $-(v.\alpha)^{-1} \cdot C \in \mathcal{L}(\mathcal{M}) \implies -(v.\alpha)^{-1} \cdot C + C_L \times C_R \in \mathcal{M}_V \implies C - v.\alpha \cdot (C_L \times C_R) + v.\beta \in \mathcal{M}_V$

- Line 16:
  $C, C_L, C_R \in \mathcal{L}(\mathcal{M})$ and C_L, C_R are variable-disjoint. Consequently,
  $-(v.\alpha)^{-1} \cdot C \in \mathcal{L}(\mathcal{M}) \implies -(v.\alpha)^{-1} \cdot C + C_L + C_R \in \mathcal{M}_V \implies C - v.\alpha \cdot (C_L + C_R) + v.\beta \in \mathcal{M}_V$

Hence, we can conclude that all the above identity tests can be carried out using the PIT algorithm of $\mathcal{M}_V$.

The correctness of the algorithm's output can be proven by a simple induction on v. That is, the algorithm outputs "true" iff $C \equiv p_v$. The base case is trivial. The correctness of the step follows from Lemmas 5.9 and 5.10 (and the correctness of the PIT algorithm of $\mathcal{M}_V$). This completes the proof. □

Notice, that as f is given to us explicitly we can acquire a justifying assignment $\bar{a}$ for f in $\mathcal{O}(n^4)$ time (see Lemma 5.1). The next corollary is immediate.

Corollary 5.8. *In time $\mathcal{O}(n \cdot t + n^4)$ we can verify whether $C \equiv f$, where t is the cost of a single PIT for $\mathcal{M}_V$.*

Lemma 5.9. *Let $C(\bar{x}, \bar{y}, \bar{z})$ and $f(\bar{x}, \bar{y}) = \alpha \cdot f_\ell(\bar{x}) \times f_r(\bar{y}) + \beta$ be two polynomials and let $(\bar{x}_0, \bar{y}_0)$ be a justifying assignment of f. Then $C(\bar{x}, \bar{y}, \bar{z}) \equiv f(\bar{x}, \bar{y})$ if and only if the following conditions hold:*

1. $f_\ell(\bar{x}) = \frac{C(\bar{x}, \bar{y}_0, \bar{z}) - \beta}{\alpha \cdot f_r(\bar{y}_0)}$
2. $f_r(\bar{y}) = \frac{C(\bar{x}_0, \bar{y}, \bar{z}) - \beta}{\alpha \cdot f_\ell(\bar{x}_0)}$
3. $C(\bar{x}, \bar{y}, \bar{z}) = \alpha \cdot \frac{C(\bar{x}, \bar{y}_0, \bar{z}) - \beta}{\alpha \cdot f_r(\bar{y}_0)} \times \frac{C(\bar{x}_0, \bar{y}, \bar{z}) - \beta}{\alpha \cdot f_\ell(\bar{x}_0)} + \beta$

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

[11] For the case of the constant polynomial notice that $C + b \in \mathcal{L}(\mathcal{M}) \subseteq \mathcal{M}_V$.

The additive version:

Lemma 5.10. *Let $C(\bar{x},\bar{y},\bar{z})$ and $f(\bar{x},\bar{y}) = \alpha\cdot(f_\ell(\bar{x})+f_r(\bar{y}))+\beta$ be two polynomials and let $(\bar{x}_0,\bar{y}_0)$ be a justifying assignment of f. Then $C(\bar{x},\bar{y},\bar{z}) \equiv f(\bar{x},\bar{y})$ if and only if the following conditions hold:*

1. $f_\ell(\bar{x}) = \frac{C(\bar{x},\bar{y}_0,\bar{z})-\beta}{\alpha} - f_r(\bar{y}_0)$
2. $f_r(\bar{y}) = \frac{C(\bar{x}_0,\bar{y},\bar{z})-\beta}{\alpha} - f_\ell(\bar{x}_0)$
3. $C(\bar{x},\bar{y},\bar{z}) = \alpha\cdot(\frac{C(\bar{x},\bar{y}_0,\bar{z})-\beta}{\alpha} - f_r(\bar{y}_0) + \frac{C(\bar{x}_0,\bar{y},\bar{z})-\beta}{\alpha} - f_\ell(\bar{x}_0)) + \beta$

Proof. Both proofs are preformed by simple substitutions. □

5.3.1 Alternative Verification Tests

The might be some cases in which it would be possible to replace or even eliminate one of the tests in lines 9 and 16 applying the following "separated" versions of Lemmas 5.9 and 5.10.

Lemma 5.11. *Let $C(\bar{x},\bar{y},\bar{z})$ and $f(\bar{x},\bar{y}) = \alpha\cdot f_\ell(\bar{x})\times f_r(\bar{y})+\beta$ be two polynomials and let $(\bar{x}_0,\bar{y}_0)$ be a justifying assignment of f. Then $C(\bar{x},\bar{y},\bar{z}) \equiv f(\bar{x},\bar{y})$ if and only if the following conditions hold:*

1. $f_\ell(\bar{x}) = \frac{C(\bar{x},\bar{y}_0,\bar{z})-\beta}{\alpha\cdot f_r(\bar{y}_0)}$
2. $f_r(\bar{y}) = \frac{C(\bar{x}_0,\bar{y},\bar{z})-\beta}{\alpha\cdot f_\ell(\bar{x}_0)}$
3. *$C(\bar{x},\bar{y},\bar{z})-\beta$ can be represented as product $H(\bar{x})\cdot G(\bar{y})$ for some two function G,H*

Additive version:

Lemma 5.12. *Let $C(\bar{x},\bar{y},\bar{z})$ and $f(\bar{x},\bar{y}) = \alpha\cdot(f_\ell(\bar{x})+f_r(\bar{y}))+\beta$ be two polynomials and let $(\bar{x}_0,\bar{y}_0)$ be a justifying assignment of f. Then $C(\bar{x},\bar{y},\bar{z}) \equiv f(\bar{x},\bar{y})$ if and only if the following conditions hold:*

1. $f_\ell(\bar{x}) = \frac{C(\bar{x},\bar{y}_0,\bar{z})-\beta}{\alpha} - f_r(\bar{y}_0)$
2. $f_r(\bar{y}) = \frac{C(\bar{x}_0,\bar{y},\bar{z})-\beta}{\alpha} - f_\ell(\bar{x}_0)$
3. *$C(\bar{x},\bar{y},\bar{z})$ can be represented as a sum $H(\bar{x})+G(\bar{y})$ for some two function G,H*

Proof. The first direction is trivial for both. For the second direction we give a proof for the multiplicative version. We make the following observations:

- $H(\bar{x}_0)\cdot G(\bar{y}) = C(\bar{x}_0,\bar{y},\bar{z})-\beta = \alpha\cdot f_\ell(\bar{x}_0)f_r(\bar{y})$
- $H(\bar{x})\cdot G(\bar{y}_0) = C(\bar{x},\bar{y}_0,\bar{z})-\beta = \alpha\cdot f_\ell(\bar{x})f_r(\bar{y}_0)$
- $H(\bar{x}_0)\cdot G(\bar{y}_0) = C(\bar{x}_0,\bar{y}_0,\bar{z})-\beta = \alpha\cdot f_\ell(\bar{x}_0)f_r(\bar{y}_0)$

Notice that $f_\ell(\bar{x}_0), f_r(\bar{y}_0) \neq 0$ implies $H(\bar{x}_0), G(\bar{y}_0) \neq 0$. A simple algebraic manipulations complete the proof. The proof of Lemma 5.12 is similar. □

5.4 Proof of Theorem 7

Before giving the proof we show why any "reasonable" model that poses an interest from the ROT point of view can, indeed, compute any univariate linear function. To see that notice that all the univariate ROPs are, in fact, all the univariate linear functions. Now, let C be a circuit from a circuit class $\mathcal{M}$ that computes a non-constant ROP. By considering all the corresponding restrictions we will obtain circuits in $\mathcal{M}$ that compute univariate linear functions.

Proof of Theorem 7. We will show that given $\mathcal{M}$ and $\mathcal{M}_V$ satisfying the required conditions we can successfully preform each phase of the "Generic ROT Scheme" described in Algorithm 5.

1. **Acquiring a Justifying Assignment**
   As $\partial\mathcal{M} \subseteq \mathcal{M}_V$ we can acquire a justifying assignment $\bar{a}$ by invoking Algorithm 6.
   The running time is $\mathcal{O}(n^3 \cdot T(n,s))$.

2. **Reconstructing ROF f from C**
   Given the justifying assignment $\bar{a}$ we can reconstruct f from C using Algorithm 1.
   The running time is $\text{poly}(n,s)$.

3. **Verifying that $f \equiv C$**
   Notice that $\mathcal{M}_V$ satisfies the conditions of Definition 5.6 and hence can serve a verifying class of $\mathcal{M}$. This allows to invoke the verification procedure described in Algorithm 9.
   The running time is $\mathcal{O}(n \cdot T(n,s) + n^4)$.

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

□

In the next section we use Theorem 7 and the Generic Scheme suggested in Algorithm 5 in order to get ROT algorithms for several restricted models for which PIT is known. Given our result it is necessary that a PIT algorithm is known for the class that we wish to get ROT algorithm for. However, we note that basically any "reasonable" PIT algorithm yields a ROT algorithm. For example, an immediate conclusion of Theorem 7 is that an efficient PIT algorithm for multilinear circuits (for both black-box and none black-box settings) implies an efficient read-once testing algorithm for multilinear circuits.

6 ROT for Specific Models

In this section we prove Theorems 9, 10 and 11 by applying the ROT scheme of Section 5 (Algorithm 5). We start with the case of sparse polynomials (Theorem 9).

6.1 Sparse Polynomials

A *$m-$sparse polynomial* is polynomial with at most m (non-zero) monomials. Equivalently, it is a polynomial computed by a depth-2 circuit of size m. Sparse polynomials were deeply studied (for example [BOT88, KS01, LV03]) and, in fact, several polynomial time algorithms for both reconstruction and PIT were given, over sufficiently large fields (or extension fields). We now show how to do ROT for sparse polynomials. Note that in [BHH95a] it is mentioned that a polynomial time ROT algorithm is known (see last sentence of paragraph 4 on page 707 of [BHH95a] that refers to [Lho91]), however, we give another proof here for completeness (thus proving Theorem 9). We denote by $\Sigma\Pi(m,d)$ the class of $m-$sparse polynomials of degree d.

In order to apply Algorithm 5 we shall need a PIT algorithm for $\Sigma\Pi(m,d)$. Several results are known and for simplicity we shall use the following result of [KS01].

Lemma 6.1 (Theorem 10 of [KS01])**.** *In time polynomial in m, n, d and $\log|\mathbb{F}|$, we can output a test set of vectors such that every non-zero $n-$variate polynomial in $\Sigma\Pi(m,d)$ over the field $\mathbb{F}$ must be non-zero at one of the vectors in the set. If $\mathbb{F}=\mathbb{R}$ then each element of each vector in the set has bit-length at most $\mathcal{O}(\log(nd))$. If $\mathbb{F}$ is a finite field with less than $(nd)^6$ elements, then the elements of the vectors lie in the smallest extension of $\mathbb{F}$ with at least $(nd)^6$ elements; otherwise, the vectors contain just elements of $\mathbb{F}$.*

Note that as $\partial\Sigma\Pi(m,d) \subseteq \Sigma\Pi(m,d)$ we can use the above lemma in conjunction with Algorithm 5 and a verification class $\mathcal{M}_V = \Sigma\Pi(m^2+3m+2, d^2)$ (see Definition 5.6) to prove Theorem 9. We now give an alternative proof of Theorem 9 that follows the same lines as the scheme of Algorithm 5, however uses a different verification procedure.

Proof of Theorem 9. Denote with P the sparse polynomial inside the black-box. We follow the scheme outlined in Algorithm 5. We first run Algorithms 6 and 1 to get a ROF. Of course, if any of the algorithms fail then we output "not ROF". Denote with f the resulting ROF. We now wish to verify that $f \equiv P$. We could use Algorithm 9, but instead we use a different scheme. We first run Algorithm 3 to count the number of monomials in the (read-once) polynomial computed by f (denoted by M). Now, if $M > m$ then, obviously, $f \not\equiv P$. Otherwise, we let $P' \stackrel{\Delta}{=} f - P$. Note that P' has at most $2m$ monomials and hence is contained in $\Sigma\Pi(2m, d')$ (when $d' = \max(n,d)$). Consequently, we can apply Lemma 6.1 to find 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 9 equipped with a PIT algorithm for $\Sigma\Pi(m^2+3m+2, d^2)$). □

6.2 Depth-3 Circuits

Depth-3 circuits (also known as $\Sigma\Pi\Sigma$ circuits) computes polynomials of the following form

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

where the L_{ij}'s are linear functions. The degree of the circuit is $\max_{i\in[k]} d_i$. A depth-3 circuit of degree d with k multiplications gate is also called a $\Sigma\Pi\Sigma(k,d)$ circuit. Efficient PIT algorithms for depth-3 circuits with constant fan-in ($k = \mathcal{O}(1)$) were given in [DS06, KS07b, KS07a, AM07]. We can use those PIT algorithms in conjunction with Algorithm 5 to construct a ROT algorithm for depth-3 circuits. We now give the proof of Theorem 10.

Proof of Theorem 10. We consider three different cases.

Explicit $\Sigma\Pi\Sigma(k,d)$ read-once testing: Assuming that a $\Sigma\Pi\Sigma(k,d)$ circuit is given to us explicitly, a polynomial time PIT (for $k = \mathcal{O}(1)$) was given by [KS07b].

Lemma 6.2. *[Theorem 1.1 of [KS07b]] There exists a deterministic algorithm that on input $\Sigma\Pi\Sigma(k,d)$ circuit C over a field $\mathbb{F}$, determines if the polynomial computed by the circuit is identically zero in time $\mathrm{poly}(n, d^k)$, where k is the fan-in of the topmost addition gate and n is the number of inputs. In particular if k is bounded, then we get a deterministic polynomial time algorithm for identity testing of depth 3 circuits.*

Clearly, $\mathcal{L}(\Sigma\Pi\Sigma(k,d)) \subseteq \Sigma\Pi\Sigma(k+1,d)$. Thus,the class $\mathcal{M}_V = \Sigma\Pi\Sigma(k^2+4k+3,d^2)$ satisfies the conditions of Theorem 7. Consequently, by using Lemma 6.2 as a PIT algorithm for $\mathcal{M}_V$ we get a $\text{poly}(n,d^{k^2})$ time ROT algorithm for $(\Sigma\Pi\Sigma(k,d))$ circuits. Note that the algorithm runs in polynomial time for $k=\mathcal{O}(1)$.

Black-box $\Sigma\Pi\Sigma(k,d)$ read-once testing: Unfortunately, the state of affairs is worse for black-box setting. No polynomial time black-box PIT algorithm for $\Sigma\Pi\Sigma(k,d)$ circuits is known. The best, and in fact, the only such algorithm is quasi-polynomial for $k=\mathcal{O}(1)$.

Lemma 6.3 (Theorem 1 of [KS07a])**.** *Let C be a $\Sigma\Pi\Sigma(k,d)$ circuit over a field $\mathbb{F}$, in n indeterminates, for some constant k. Then there is a deterministic black-box algorithm that on input k, d, n and black-box access to C determines whether C computes the zero polynomial. The running time of the algorithm is $\text{poly}(n,d)\cdot\exp(\log^{k-1} d)$. If $|\mathbb{F}|$ is $\mathcal{O}(d^3\cdot n)$ then the algorithm is allowed to make queries to C from an algebraic extension field of $\mathbb{F}$ of polynomial size.*

As in the previous case we get a $\text{poly}(n,d)\cdot\exp(\log^{\mathcal{O}(k^2)} d)$ time ROT algorithm for $(\Sigma\Pi\Sigma(k,d))$ circuits. Note that the running time is quasi-polynomial for $k=\mathcal{O}(1)$.

Multilinear Black-box $\Sigma\Pi\Sigma(k)$ read-once testing: A multilinear $\Sigma\Pi\Sigma(k)$ circuit is a circuit in which every multiplication gate ($\prod$) computes a multilinear polynomial. For multilinear $\Sigma\Pi\Sigma(k)$ circuits a polynomial time black-box PIT algorithm is known (again, when $k=\mathcal{O}(1)$).

Lemma 6.4 (Theorem 1 of [KS07a])**.** *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit in n indeterminates over a field $\mathbb{F}$ where k is a constant. Then there is a deterministic polynomial time (in the number of variables) black-box algorithm that on input k,n and black-box access to C determines whether C computes the zero polynomial. If $|\mathbb{F}|\leq\mathcal{O}(n^3\cdot k^2)$ then the algorithm is allowed to make queries to C from an algebraic extension field of $\mathbb{F}$ of polynomial size.*

As before, $\mathcal{L}(\Sigma\Pi\Sigma(k)) \subseteq \Sigma\Pi\Sigma(k+1)$ (in its multilinear version) and it is not hard to see that the class $\mathcal{M}_V = \Sigma\Pi\Sigma(k^2+4k+3)$ (of multilinear circuits) satisfies the conditions of Theorem 7. Therefore, we get a ROT algorithm for multilinear $\Sigma\Pi\Sigma(k)$ circuits that runs in polynomial time for $k=\mathcal{O}(1)$.

□

6.3 Non-Commutative Formulas

In the regular (commutative) setting an arithmetic circuit C over the field $\mathbb{F}$ computes a polynomial over the ring $\mathbb{F}[x_1,x_2,\ldots,x_n]$. In particular, both addition and multiplication are commutative operations. For example the formal expressions $x_1x_2+x_3$ and $x_2x_1+x_3$ are considered as the same polynomial. In the non-commutative setting, a circuit C computes a formal expression over $\mathbb{F}\{x_1,x_2,\ldots,x_n\}$, the ring of polynomials over non-commutating variables. In this case only the addition operation remains commutative and thus the above formal expressions are considered as *different* polynomials. Note though, that if the output of C is the zero polynomial over $\mathbb{F}\{x_1,x_2,\ldots,x_n\}$ it is also the zero polynomial over $\mathbb{F}[x_1,x_2,\ldots,x_n]$, but not the other way around.

The non-commutative model was introduced and studied in [Nis91, BW05, RS05]. Known results include reduction to algebraic branching programs, deterministic PIT in the non black-box setting and randomized PIT in the black-box setting. In this subsection we study the properties of ROPs computed by this circuit class and provide a proof for Theorem 11.

Definition 6.5. Non-commutative read-once polynomial *is a polynomial computed by a ROF with non-commutating variables.*

We begin with a simple observation:

Observation 6.6. *If the output of a circuit C over $\mathbb{F}\{x_1, x_2, \ldots, x_n\}$ is a non-commutative ROP then the output of a circuit C over $\mathbb{F}[x_1, x_2, \ldots, x_n]$ is a (commutative) ROP (that is, even if we let the variables commute the circuit will still compute a ROP).*

Due to the distinct nature of the non-commutative polynomials, one should be careful when applying the standard definitions and/or relying on the basic properties known for the commutative polynomials when dealing with their non-commutative "cousins". For example consider the non-commutative polynomial $P(x_1, x_2, x_3) = x_1 \cdot x_2 \cdot x_3 - x_3 \cdot x_1 \cdot x_2$. Clearly, P depends on x_3 (as it appears in P when listed as a sum of monomials). However,

$$\frac{\partial P}{\partial x_3} = x_1 \cdot x_2 \cdot 1 - 1 \cdot x_1 \cdot x_2 - (x_1 \cdot x_2 \cdot 0 - 0 \cdot x_1 \cdot x_2) \equiv 0$$

Moreover, even the basic definition of dependence (see section 2) does not sustain here, since for every $\alpha_1, \alpha_2 \in \mathbb{F}$ $P(\alpha_1, \alpha_2, x_3) = \alpha_1 \cdot \alpha_2 \cdot x_3 - x_3 \cdot \alpha_1 \cdot \alpha_2 \equiv 0$. In addition, notice that Lemma 2.17 does not hold either as for every $\beta \in \mathbb{F}$ $P(x_1, x_2, \beta) \equiv 0$. On the other hand, Lemmas 3.14 and 3.19 still hold in their non-commutative version. Nevertheless, we show that we can still use the generic schema to obtain a ROT algorithm for the non-commutative formulas.

Proof of Theorem 11. Let P be a non-commutative polynomial. As mentioned above, we would like to apply Algorithm 5 for the ROT task. In order to do so, we show that we can perform each of its steps. I.e. acquire a justifying assignment, then constructing a non-commutative ROF F out of it, and finally verify that $F \equiv P$. We start by showing that Algorithm 6 also works in the non-commutative case. For this we first observe that the correctness of the Algorithm, given an appropriate PIT algorithm, relies on the following properties:

- A partial derivative of a ROP is a ROP.
- A ROP P depends on x_i iff $\frac{\partial P}{\partial x_i} \not\equiv 0$.
- Give a non-zero ROP P for every variable x_i there exists at most one $\alpha \in \mathbb{F}$ such that $P|_{x_i=\alpha} \equiv 0$.

Actually, these properties can be restated to hold for every commutative polynomial. The reason for choosing this statement is that it also holds for non-commutative ROPs. Note that unlike the example above of the polynomial $x_1 \cdot x_2 \cdot x_3 - x_3 \cdot x_1 \cdot x_2$, the properties above are stated for non-commutative ROPs and not for a general non-commutative polynomial. To show that the properties hold for non-commutative ROPs we first note that, as mentioned before, the first property follows by slightly changing the proof of Lemma 3.19. The correctness of other two properties follows from the next lemmas.

Lemma 6.7. *Let $P \in \mathbb{F}\{x_1, x_2, \ldots, x_n\}$ be a non-commutative ROP. Then P depends on x_i iff $\frac{\partial P}{\partial x_i} \not\equiv 0$.*

Proof. The first direction is trivial. For the second direction we apply an induction on $k = |\mathrm{var}(P)|$

Base Case: For $k = 1$ the claim is trivial.

Step: For $k \geq 2$ let $i \in [n]$. By the non-commutative version of Lemma 3.14 we get that P can be in a one of the two forms:

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$. In this case $\frac{\partial P}{\partial x_i} = \frac{\partial P_1}{\partial x_i} + \frac{\partial P_2}{\partial x_i}$ and consequently $\frac{\partial P}{\partial x_i} \not\equiv 0$ implies w.l.o.g that $\frac{\partial P_1}{\partial x_i} \not\equiv 0$. By definition $|\text{var}(P_1)| < |\text{var}(P)|$ and thus from the induction hypothesis P_1 depends on x_i. Finally, since P_1 and P_2 are variable disjoint we obtain that P depends on x_i as well.

2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$. In this case we have that $\frac{\partial P}{\partial x_i} \not\equiv 0$ implies $P_1|_{x_i=1} \cdot P_2|_{x_i=1} \neq P_1|_{x_i=0} \cdot P_2|_{x_i=0}$. Assume w.l.o.g that $P_1|_{x_i=1} \neq P_1|_{x_i=0}$ which is equivalent to $\frac{\partial P_1}{\partial x_i} \not\equiv 0$. The result follows similarly to the previous case.

□

Lemma 6.8. *Let $P \in \mathbb{F}\{x_1, x_2, \ldots, x_n\}$ be a non-commutative ROP such that for some variable x_i and $\alpha \neq \beta \in \mathbb{F}$ we have that $P|_{x_i=\alpha} = P|_{x_i=\beta} \equiv 0$ then $P \equiv 0$.*

Proof. Note that $\frac{\partial P}{\partial x_i} \cdot (\beta - \alpha) = P|_{x_i=\beta} - P|_{x_i=\alpha}$. Hence, $\frac{\partial P}{\partial x_i} \equiv 0$ and from the previous lemma P does not depend on x_i. Consequently, $P = P|_{x_i=\alpha} \equiv 0$. □

The final ingredient in the acquisition of a justifying assignment is a PIT procedure for the partial derivative of the circuit class. It is easy to see that the (discrete) partial derivative of a non-commutative formula is a non-commutative formula as well. Thus, the following lemma of [RS05] gives the required PIT algorithm.

Lemma 6.9 (Theorem 5 of [RS05])**.** *Given a non-commutative arithmetic formula of size s we can verify in time polynomial in s whether the formula is identically zero.*

As a result of the above, we can use Algorithm 6 to get a justifying assignment in the non-commutative case as well. The next step is to give an analog of Algorithm 1 to the case of non-commutative formulas. The following modifications are required in order to adapt Algorithm 1 to non-commutative polynomials. The Gates Graph of a non-commutative ROF should be directed, indicating the order of multiplication (that is we put an edge (i, j) if x_i and x_j appear in the same monomial and x_i is to the left of x_j). The non-commutative analog of Algorithm 1 is given as follows.

- Use the justifying assignment to construct a *directed* Gates Graph
- Use the justifying assignment construct the *directed* skeleton
- Recursively learn sub-formulas

We do not give a proof here that all the steps can be implemented efficiently as the proof is basically the same as the proof in the commutative case, and only requires noting that the order of multiplicands in every monomial of the constructed ROF is indeed the same as the one in the underlying ROP. We leave the details to the reader. We now show how to implement the verification step of Algorithm 5. Note that as the reconstruction steps returns a non-commutative ROF, that in theory should compute the same polynomial as the one at hand, we can simply consider $F - P$ and use the PIT guaranteed by Lemma 6.9 to check whether $F \equiv P$. This completes the proof of Theorem 11. □

7 PIT Algorithm for Alternating Read-Once Formulas

In this section we prove Theorems 1 and 2. It will be more convenient for us to use the notion of alternating ROF instead of the usual notion of ROF (see Definition 3.5). We start by generalizing some of the notions and proofs to AROFs.

7.1 Some Properties of AROFs

We first define the depth of a ROP.

Definition 7.1. *For a ROP $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ we define:* $\text{depth}(P)$ *to be the depth of the AROF computing it.*

This definition is well defined as the depth of an AROF is determined by its skeleton, which is in turn determined by the gates-graph of P (Lemma 3.13). The depth of a ROP P will play an important role in our PIT algorithms. We now give the analog of Lemmas 3.14, 3.19 and 3.16 for the case of AROF.

Lemma 7.2. *Every ROP $P(x)$ with $|\text{var}(P)| \geq 2$ of depth d can be presented in* exactly *one of the following forms:*

1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x}) + \ldots + P_k(\bar{x})$.
2. $P(\bar{x}) = f\left(P_1(\bar{x}), P_2(\bar{x}), \ldots, P_k(\bar{x})\right)$.

Where, for every $j \in [k]$, the polynomials $P_j(\bar{x})$ are non-constant variable-disjoint ROPs of depth at most $d-1$, and f is a multiplicative ROP.

Proof. The proof is similar to the proof of Lemma 3.14. □

Lemma 7.3. *A partial derivative of a ROP $P(\bar{x})$ of depth d is a ROP of depth at most d.*

Proof. Let P be a ROP of depth at most d and $x_i \in \text{var}(P)$. We prove the lemma by induction on $m = |\text{var}(P)|$. For $m = 0, 1$ the claim is trivial. For $m \geq 2$ we get by Lemma 7.2 that P can be in a one of the two forms:

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

This concludes the proof of the lemma. □

Finally, we state a lemma analogous to Lemma 3.16.

Lemma 7.4. *A ROP P of depth at most d is reducible if and only if it is of the form $P(\bar{x}) = f\left(P_1(\bar{x}), P_2(\bar{x}), \ldots, P_k(\bar{x})\right)$ where the P_j's are non-constant variable-disjoint ROPs of depth at most $d-1$, and $f(y_1, y_2, \ldots y_k)$ is a reducible polynomial.*

Using this lemma it is very easy to convert Algorithm 2 to an algorithm that finds the irreducible factors of a given AROF.

7.2 Black-Box PIT Algorithm for Bounded-Depth AROFs

In this subsection we prove Theorem 2. The main idea is to convert a ROP P, that has many variables, each with degree 1, to a polynomial P' with a "small" number of variables and a "reasonable" degree , such that $P' \equiv 0$ if and only if $P \equiv 0$. For this task, we define the following two families of maps.

Definition 7.5. *Let $n \geq m \geq 1$ and $d \geq 1$ be integers. Define the trace functions*

$$\tau_i(y_1, \ldots, y_d) \stackrel{\Delta}{=} \sum_{j=1}^{d} y_j^i.$$

Let $S_d(y_1, \ldots, y_d) : \mathbb{F}^d \to \mathbb{F}^n$ be defined as

$$S_d(y_1, \ldots, y_d) = (\tau_1(y_1, \ldots, y_d), \ldots, \tau_n(y_1, \ldots, y_d)).$$

Denote with $e_i \in \{0,1\}^n$ the vector that has a single non zero entry at the i-th coordinate. Let $T_{d,m}(z, y_1, y_2, \ldots, y_d) : \mathbb{F}^{d+1} \to \mathbb{F}^n$ be defined as

$$T_{d,m} = (\tau_1(y_1, \ldots, y_d), \ldots, \tau_n(y_1, \ldots, y_d)) + z \cdot e_m.$$

That is, the only difference between S_d and $T_{d,m}$ is the m-th coordinate in which $(S_d)_m = \tau_m = \sum_{j=1}^{d} y_j^m$, and $(T_{d,m})_m = \tau_m + z = z + \sum_{j=1}^{d} y_j^m$. In particular, for every m we have that $(T_{d,m})|_{z=0} = S_d$.

The transformation takes advantage of the fact that at each variable is read at most once, so there is always a variable with the smallest index that can not be cancelled. Note, that the bounded depth AROF model generalizes the standard bounded depth model.

Definition 7.6. *For a non-constant polynomial $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ we define:* $\text{minarg}(P) = \min\{i \mid x_i \in \text{var}(P)\}$.

We now give the main idea behind our PIT algorithm. We say that AROF is an *additive* AROF if its root is an addition gate.

Lemma 7.7. *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-constant ROP computed by an additive AROF $\varphi : \mathbb{F}^n \to \mathbb{F}$ of depth $< 2d$ and let $m = \text{minarg}(P)$. Then the following holds:*

1. *$P(T_{d,m})$ depends on z.*
2. *$P(S_d)$ is a non-constant polynomial in $\bar{y}$ (in particular $P(S_d) \not\equiv 0$).*

Proof. We show that the two properties hold by a joint induction on d. When $d = 1$ we have that an additive AROF of at most depth 1 must have the form $P = a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$. It is clear that the two properties hold. We now consider the case $d > 1$. We can represent P as

$$P = \sum_{i=1}^{k} g_i(\bar{x}) = \sum_{i=1}^{k} f_i\left(P_{i,1}(\bar{x}), P_{i,2}(\bar{x}), \ldots, P_{i,k_i}(\bar{x})\right)$$

where $\{f_i\}_{i \in [k]}$ are the multiplicative ROPs that label the children of the root of φ (which are MUL gates), and each $P_{i,j}$ is a ROP of depth $< 2d - 2 = 2(d-1)$, computed by an additive AROF that

enters the MUL gate labelled with f_i. Let $m = \text{minarg}(P)$. Assume w.l.o.g. that $m \in \text{var}(P_{1,1})$. We first show that $P(T_{d,m})$ depends on z. Consider $P_{1,j}$ for $j > 1$. As $m \notin \text{var}(P_{1,j})$ we get from the induction hypothesis that

$$P_{1,j}(T_{d-1,m}) = P_{1,j}(S_{d-1}) \text{ is a non-constant polynomial}$$

Similarly, we get by the induction hypothesis that $P_{1,1}(T_{d-1,m})$ is a non-zero polynomial depending on z. The following lemma guarantees that $g_1(T_{d-1,m})$ is a non zero polynomial depending on z.

Lemma 7.8. *Let $Q(x_1, \ldots, x_k) : \mathbb{F}^k \to \mathbb{F}$ be a non-constant multiplicative ROP and let $h_1(z, \bar{y}), h_2(\bar{y}), \ldots, h_k(\bar{y})$ be non-constant polynomials such that $z \in \text{var}(h_1)$. Then $Q(h_1, \ldots, h_k) \not\equiv 0$ and depends on z.*

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

As a corollary we get that

$$g_1(T_{d,m})|_{y_d=0} = g_1(T_{d-1,m}) = f_1\left(P_{1,1}(T_{d-1,m}), P_{1,2}(T_{d-1,m}), \ldots, P_{1,k_1}(T_{d-1,m})\right) \not\equiv 0$$

and $g_1(T_{d,m})|_{y_d=0}$ depends on z. As for $i > 1$ we have that $g_i(T_{d-1,m})$ does not depend on z, it follows that

$$P(T_{d,m})|_{y_d=0} = P(T_{d-1,m}) = \sum_{i=1}^{k} g_i(T_{d-1,m}) \not\equiv 0$$

and $P(T_{d,m})|_{y_d=0}$ depends on z. In particular $P(T_{d,m}) \not\equiv 0$ and depends on z, as required. We now want to show that $P(S_d) = P(T_{d,m})|_{z=0} \not\equiv 0$. Recall that $m = \text{minarg}(P)$. Therefore we can write

$$g_1(\bar{x}) = x_m \cdot A_1(x_{m+1}, \ldots, x_n) + B_1(x_{m+1}, \ldots, x_n),$$

where A_1 is a non-zero polynomial. Moreover, $A_1(S_{d-1}) = A_1(T_{d-1,m}) \not\equiv 0$ (as $g_1(T_{d-1,m}) = g_1(T_{d,m})|_{y_d=0}$ depends on z). Summing all together we can write

$$P(\bar{x}) = x_m \cdot A_1(x_{m+1}, \ldots, x_n) + B(x_{m+1}, \ldots, x_m),$$

as the other g_i's do not depend on x_m (because they come from a ROF). It follows that

$$P(T_{d,m}) = (z + \sum_{j=1}^{d} y_j^m) \cdot A_1(T_{d,m}) + B(T_{d,m}) = (z + \sum_{j=1}^{d} y_j^m) \cdot A_1(S_d) + B(S_d).$$

We now make two observations. The first is that $A_1(S_d)$ has a monomial not depending on y_d. This follows from the fact mentioned above that $A_1(S_{d-1}) \not\equiv 0$. The second observation is that in every monomial involving y_d in either $A_1(S_d)$ or $B(S_d)$, the degree of y_d is at least $m + 1$. This follows from the fact that these two polynomials only depend on $x_{m+1}, \ldots, x_n$. In particular we have that

$$P(T_{d,m}) = (z + \sum_{j=1}^{d} y_j^m) \cdot A_1(S_{d-1}) + B(S_{d-1}) + y_d^{m+1} \cdot C(z, y_1, \ldots, y_d),$$

for some polynomial C. Note that $A_1(S_{d-1})$ and $B(S_{d-1})$ do not depend on y_d. We now get that

$$P(S_d) = P(T_{d,m})|_{z=0} = y_d^m \cdot A_1(S_{d-1}) + y_d^{m+1} \cdot C(0, y_1, \ldots, y_d) + \sum_{j=1}^{d-1} y_j^m \cdot A_1(S_{d-1}) + B(S_{d-1})$$

In particular, any non-zero monomial of $y_d^m \cdot A_1(S_{d-1})$ cannot be cancelled (as the degree of y_d in such monomials is exactly m, and in any other monomial the degree of y_d is either 0 or at least $m+1$). As a conclusion we get that $P(S_d)$ is a non-constant polynomial in $\bar{y}$ and particularly, $P(S_d) \not\equiv 0$, and the proof is completed. □

The following corollary is immediate.

Corollary 7.9. *Let P be a non-zero ROP computed by an AROF (not necessarily additive) of depth $\leq 2d$ then $P(S_{d+1}) \not\equiv 0$* [12].

We now construct the hitting set suggested by Lemma 7.7.

Theorem 7.10. *Let $W \subseteq \mathbb{F}$ be a set of size n^2. Let $A = S_{d+1}\left(W^{d+1}\right) \subseteq \mathbb{F}^n$ (that is, we take the image of W^{d+1} under S_{d+1}). Then an n-variate ROP P of depth $\leq 2d$ is zero if and only if $P|_A \equiv 0$.*

Note, that $|A| \leq (n^2)^{d+1} = n^{\mathcal{O}(d)}$.

Proof. If $P \equiv 0$ then the claim is trivial. Assume that $P \not\equiv 0$. By Corollary 7.9 we get that $P(S_{d+1}) \not\equiv 0$. From the definition of S_{d+1}, and the fact that P is a multilinear polynomial, it follows that $P(S_{d+1}(\bar{y}))$ is a $(d+1)$-variate polynomial, of degree at most $1+2+3+\ldots+n < n^2$ in each variable. Lemma 2.16 implies that $P \not\equiv 0$ if and only if $P|_A \not\equiv 0$. □

The proof of Theorem 2 now follows easily.

Proof of Theorem 2. Clearly, Theorem 7.10 implies a black-box PIT algorithm that runs in time $n^{\mathcal{O}(d)}$. To complete the proof of Theorem 2 we describe an interpolation procedure for bounded depth read-once formulas - AROF(d). Let $P \in \text{AROF}(d)$. Lemma 7.3 implies that $\partial\text{AROF}(d) \subseteq \text{AROF}(d)$ consequently, we can acquire a justifying assignment $\bar{a}$ for P by invoking Algorithm 6 and using the hitting set from Theorem 7.10. Once we have a justifying assignment, we can construct a ROF f that computes P applying Algorithm 1. Note, that the entire reconstruction procedure requires $n^{\mathcal{O}(d)}$ running time. □

7.3 PIT for General Read-Once Formulas

In this section we prove Theorem 1. The idea of the proof is the following. Given a black-box holding a ROP P we have two options. Either the depth of P, denoted with d, is smaller than $\mathcal{O}(\sqrt{n})$, and in this case we can use Theorem 7.10. Or, we are in a case where any AROF for P has a "high" depth. We show that in such a case there exists an input variable x_i such that $\frac{\partial P}{\partial x_i}$ is not the zero polynomial and, more importantly, depends on at most $n - d/2$ variables. The idea behind the proof is to show a that if there is a variable x_i that is "very far" from the root of the AROF for P, then there are "many" variables x_j such that $x_i \cdot x_j$ does not appear in any monomial. Consequently we can eliminate the dependency on those variables by taking a partial derivative w.r.t. x_i. By repeating this several times we get as a result that there exists a set $I \subseteq [n]$ of size

[12] Consider $\hat{P} \triangleq P + x_{n+1}$. We leave the rest of details to the reader.

$\mathcal{O}(\sqrt{n})$ such that $\partial_I P$ is non zero and it is either of depth smaller than $\sqrt{n}$ or it depends on a small number of variables. In either case we can perform PIT efficiently (that is, in sub-exponential time).

Lemma 7.11. *Let $P \not\equiv 0 \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a ROP. For every $k \geq 0$ there exist a subset $I_k \subseteq [n]$ such that the following properties hold.*

1. *The size of I_k is at most k.*
2. *The partial derivative $\partial_{I_k} P$ is non-zero.*
3. *The partial derivative w.r.t I eliminates many variables. That is, $|\text{var}(\partial_{I_k} P)| \leq |\text{var}(P)| - \frac{k}{2} \cdot \text{depth}(\partial_{I_k} P)$.*

Proof. We shall show the proof only for the case $k = 1$, as after taking a partial derivative the depth cannot increase (see Lemma 7.3), and so we can construct the set I by adding one element at a time. We assume w.l.o.g. that $\text{depth}(P) > 0$ as otherwise there is nothing to prove.

Consider an alternating ROF f for P. As $\text{depth}(f) \geq 1$, it must be the case that f contains at least one gate and that $|\text{var}(P)| \geq 2$. Let x_i be the variable labelling a leaf that is farthest from the roof of f. Let π be the path from x_i to the root. By definition π is an alternating path (that it, it has alternating MUL and + gates) of length $d \stackrel{\Delta}{=} \text{depth}(f) = \text{depth}(P)$. In particular π contains at least $\lfloor \frac{d}{2} \rfloor \geq (d-1)/2$ addition gates. Consider such an addition gate v. Consider the child of v that is not on π. There must be a leaf leading to this child. Assume that x_j is labelling this leaf. As $\text{fcg}(x_i, x_j) = v$ it must be the case that $x_j \notin \text{var}(\frac{\partial P}{\partial x_i})$. In addition, $x_i \notin \text{var}(\frac{\partial P}{\partial x_i})$ as well. As this is the case for x_i and for every addition gate on π we have that

$$|\text{var}(\frac{\partial P}{\partial x_i})| \leq |\text{var}(P)| - \frac{d-1}{2} - 1 < |\text{var}(P)| - \frac{d}{2}$$

as required. □

The following is an immediate corollary.

Corollary 7.12. *For every non-zero ROP $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and $k \geq 0$ there exists a subset $I_k \subseteq [n]$ such that:*

1. *$|I_k| \leq k$.*
2. *$\partial_{I_k} P \not\equiv 0$.*
3. *$\text{depth}(\partial_{I_k} P) \leq |\text{var}(P)| \cdot \frac{2}{k+2}$*

Proof. Let I_k be the set guaranteed by Lemma 7.11. It remains to verify the third claim. As for every ROP P, we have that $\text{depth}(P) \leq |\text{var}(P)|$ we get that

$$\text{depth}(\partial_{I_k} P) \leq |\text{var}(\partial_{I_k} P)| \leq |\text{var}(P)| - \frac{k}{2} \cdot \text{depth}(\partial_{I_k} P).$$

By changing sides we get that

$$\text{depth}(\partial_{I_k} P) \leq \frac{2}{k+2} \cdot |\text{var}(P)|.$$

□

Algorithm 10 Black-Box PIT for ROP

Input: (A black-box access to) ROP P.
Output: "true" if $P \equiv 0$ and "false" otherwise.

1: **for all** $I \subseteq [n]$ such that $|I| \leq \sqrt{n}$ **do**
2: **if** $\partial_I P \not\equiv 0$ as a ROP of depth $\leq 2\sqrt{n}$ **then** {Using the hitting set from Theorem 7.10 }
3: **return** false
{Everything is OK}
4: **return** true

We now give the black-box algorithm for PIT of general read-once formulas.

Theorem 7.13. *Algorithm 10 is a deterministic PIT algorithm for ROF that runs in time* $\exp\left(\tilde{\mathcal{O}}(\sqrt{n})\right)$.

Proof. The correctness of the algorithm is clear from Corollary 7.12, by taking $k = \sqrt{n}$. We now analyze the running time of the algorithm. Note, that in order to compute the value of $\frac{\partial P}{\partial x_1}$ at a given point $\bar{a}$ we have to query the polynomial at two points:

$$\frac{\partial P}{\partial x_1}(\bar{a}) = P(1, \bar{a}) - P(0, \bar{a}).$$

It follows that in order to compute the value of $\partial_I(P)$, at a given input, we need to make $2^{|I|}$ queries to the black-box holding P. According to Theorem 7.10 we have to make $n^{\mathcal{O}(\sqrt{n})}$ queries to $\partial_I(P)$ in order to check whether it computes the zero polynomial. Therefore, we have to make a total of $2^{|I|} \cdot n^{\mathcal{O}(\sqrt{n})}$ queries to P, in order to check whether $\partial_I P \equiv 0$. As we have to do it for every $I \subseteq [n]$ of size at most $\sqrt{n}$ we get that the running time is

$$\sum_{k=0}^{\sqrt{n}} \binom{n}{k} \cdot 2^{|I|} \cdot n^{\mathcal{O}(\sqrt{n})} = n^{\mathcal{O}(\sqrt{n})} = \exp\left(\tilde{\mathcal{O}}(\sqrt{n})\right).$$

□

Proof of Theorem 1. As before, Theorem 7.13 gives a PIT algorithm of the appropriate running time. We now describe a reconstruction procedure for read-once formulas. Let P be a ROP. Lemma 3.19 implies that $\partial\text{ROF} \subseteq \text{ROF}$ consequently, we can acquire a justifying assignment $\bar{a}$ for P by invoking Algorithm 6 and using the PIT Algorithm 10. Once we have a justifying assignment, we can construct a ROF f that computes P (Algorithm 1). Note, that the entire interpolation procedure requires $n^{\mathcal{O}(\sqrt{n})} = \exp\left(\tilde{\mathcal{O}}(\sqrt{n})\right)$ running time. □

8 PIT for Sum of Read-Once Formulas

In this section we prove Theorem 3. That is, we are given the computation graphs of k ROFs $\{F_m\}_{m\in[k]}$ and we have to find whether they sum to zero. The proof of Theorem 8 appears in Subsection 8.4 and is based on the results that we prove here. Denote $F = \sum_{m=1}^{k} F_m$, then we have to check whether $F \equiv 0$. Our algorithm for the problem has two steps. In the first step we find a common justifying assignment to $F_1, \ldots, F_k$ using Algorithm 6. Assume w.l.o.g. that all the input formulas are $\bar{0}-$justified (we can shift each input variable to get this see Subsection 5.2.2). The

second step is to verify that F vanishes on all the 0/1 inputs of weight at most k^2. Theorem 8.2 then guarantees that $F \equiv 0$. To establish the theorem we need Theorem 8.3 that shows that we cannot represent $\mathcal{P}_n$ as a sum of less than $\sqrt{n}$ many $\bar{0}-$justified ROPs.

We call this approach a *hardness of representation* approach as the proof is based on the fact that a monomial cannot be represented (computed) by a sum of a "small" number of $\bar{0}-$justified ROPs.

8.1 The Algorithm

We present an identity testing algorithm for the sum of read-once formulas. For the algorithm we assume that $|\mathbb{F}| \geq kn$ (recall that we are allowed to make queries from an extension field). We shall use $\mathcal{A}^n_{k^2}(\{0,1\})$ as defined in Definition 2.11 with the set $W = \{0,1\}$. Recall that there is a PIT algorithm for a single ROF (in a non black-box setting) that requires in $\mathcal{O}(n)$ time (see the discussion in Section 4).

Algorithm 11 PIT algorithm for sum of read-once formulas

Input: ROFs $F_1, \ldots, F_k$ when $F = F_1 + \cdots + F_k$.
Output: "true" if $F \equiv 0$ and "false" otherwise.

1: acquire common justifying assignment $\bar{a}$ for the ROFs $F_1, F_2, \ldots, F_k$ {using Algorithm 6}
2: **if** $F_{\bar{a}}|_{\mathcal{A}^n_{k^2}(\{0,1\})} \equiv 0$ **then**
3: **return** true
4: **else**
5: **return** false

We now analyze the running time of the algorithm. The proof of correctness is given in the next section.

Lemma 8.1. *The running time of Algorithm 11 is* $n^{\mathcal{O}(k^2)}$.

Proof. Acquiring a common justifying assignment $\bar{a}$ formulas requires time $\mathcal{O}(n^4k^2)$ (see Lemma 5.1 with $t = \mathcal{O}(n)$). Verifying that $F_{\bar{a}}|_{\mathcal{A}^n_{k^2}(\{0,1\})} \equiv 0$ requires at most $\left(\sum_{i=0}^{k^2} \binom{n}{i}\right) \cdot k$ time. We thus get that the running time is $\mathcal{O}(k \cdot \binom{n}{k^2}) = n^{\mathcal{O}(k^2)}$. □

8.2 Analysis of the Algorithm

In this subsection we show that if $\sum_{m=1}^{k} F_m$, a sum of $\bar{0}-$justified ROPs, vanishes on a certain set of points then the sum is zero. This guarantees the correctness of the algorithm as by the definition of $\bar{a}$ it must be the case that $F_{\bar{a}} = \sum_{m=1}^{k} (F_m)_{\bar{a}}$ is a sum of $\bar{0}-$justified ROPs (see Section 5.2.2).

Theorem 8.2. *Let* $\{F_m(\bar{x})\}_{m\in[k]}$ *be* $\bar{0}-$*justified ROPs over* $\mathbb{F}[x_1, x_2, \ldots, x_n]$ *and let* $F(\bar{x}) = \sum_{m=1}^{k} F_m(\bar{x})$. *Assume that* $k \leq \sqrt{n}$. *Then* $F \equiv 0$ *if and only if* $F|_{\mathcal{A}^n_{k^2}(\{0,1\})} \equiv 0$.

Before giving the proof of the theorem we show that together with Lemma 8.1 it implies Theorem 3.

Proof of Theorem 3. By Theorem 8.2 we get that Algorithm 11 correctly decides whether the given k ROFs sum to zero. The claim regarding the running time was proved in Lemma 8.1. □

In particular, for a small k we obtain an efficient PIT algorithm. For example: for $k = \mathcal{O}(1)$ (constant fan-in) we obtain a polynomial time algorithm, for $k = \text{poly}\log(n)$ a quasi-polynomial time algorithm and for $k = o(\frac{n}{\log n})^{1/2}$ - sub-exponential time algorithm. We now give the proof of Theorem 8.2.

Proof of Theorem 8.2. We first direction is trivial. For the second direction we apply induction on n. Our base case is when $n \leq k^2$. In this case F is a multilinear polynomial in n variables that vanishes on the boolean cube $\{0,1\}^n$ and therefore (Lemma 2.16) $F \equiv 0$. We now assume that $n > k^2 \geq 4$. Let $\ell \in [n]$. Consider the restriction of the F_m's and F to the subspace $x_\ell = 0$. Namely, let $F' \triangleq F|_{x_\ell=0}$ and $\{F'_m \triangleq F_m|_{x_\ell=0}\}_{m\in[k]}$. It is clear from the definition that the F'_m's are also $\bar{0}-$justified ROFs. As $\mathcal{A}^{n-1}_{k^2}(\{0,1\}) = \mathcal{A}^n_{k^2}(\{0,1\}) \cap \{\bar{x} \in \mathbb{F}^n |\ x_\ell = 0\}$, it follows that $F'|_{\mathcal{A}^{n-1}_{k^2}(\{0,1\})} \equiv 0$. Therefore, $\{F'_m(\bar{x})\}_{m\in[k]}$ satisfy the conditions of the theorem and so by the induction hypothesis we get that $F' = 0$. In other words, $F|_{x_\ell=0} \equiv 0$ and therefore x_ℓ is a factor of F (see Lemma 2.17). As this holds for every ℓ we get that $\mathcal{P}_n$ divides F. Since F is a multilinear polynomial this implies that $F = a \cdot \mathcal{P}_n$ for some $a \in \mathbb{F}$. It follows that $a \cdot \mathcal{P}_n$ is a sum of $k < \sqrt{n}$ $\bar{0}-$justified ROPs. Theorem 8.3 (that we prove in the next subsection) shows that in such a case we must have that $a = 0$. In particular, we get that $F = a \cdot \mathcal{P}_n \equiv 0$. This completes the proof of the theorem. □

8.3 Hardness of Representation Theorem for Sum of $\bar{0}-$Justified ROFs

In this section we complete the last piece needed for the proof of Theorem 8.2. We give a *hardness of representation* theorem, that shows that there is no way to write the polynomial $\mathcal{P}_n$ as a sum of $k \leq \sqrt{n}$ $\bar{0}-$justified ROPs. In fact, we show a stronger statement.

Theorem 8.3 (HOR)**.** *The polynomial $\mathcal{P}_n(\bar{x})$ cannot be represented as sum of k $\bar{0}-$weakly-justified ROPs for $k \leq \sqrt{n}$.*

Proof. Let $\{F_m(\bar{x})\}_{m\in[k]}$ be k $\bar{0}-$weakly-justified ROPs over $\mathbb{F}[x_1, x_2, \ldots, x_n]$. We prove the claim by induction on k. For $k = 0, 1$ the claim is trivial. We now assume that $k \geq 2$ and that $n \geq k^2$. We shall assume for a contradiction that $\sum_{m=1}^{k} F_m = \mathcal{P}_n$. The idea of the proof is to find a set of (indices of) input variables $I \subseteq [n]$ such that after we take a partial derivative with respect to all of them (that is we consider the ROPs $\{\partial_I F_m\}_{m\in[k]}$) we get that for every $m \in [k]$, $\partial_I F_m = (x_n - \alpha_m) \cdot h_m$, for some $\alpha_m \neq 0$ and a ROP h_m. Now we substitute $x_n = \alpha_k$ and get that we can represent the polynomial $(\partial_I \mathcal{P}_n)|_{x_n=\alpha_k}$ as a sum of at most $k-1$ $\bar{0}-$weakly-justified ROPs (the fact that h_m is a $\bar{0}-$weakly-justified ROP is not hard to show). Then we use the induction hypothesis to reach a contradiction. We now proceed with the proof. There are two cases to consider.

- There exist $i \neq j \in [n]$ and $m \in [k]$ such that $\frac{\partial^2 F_m}{\partial x_i \partial x_j} \equiv 0$ (namely, F_m does not contain $x_i \cdot x_j$ in any of its monomials). Assume w.l.o.g. that $i = n-1, j = n, m = k$. By considering the partial derivatives with respect to $\{x_n, x_{n-1}\}$ we get that

$$\sum_{m=1}^{k-1} \frac{\partial^2 F_m}{\partial x_n \partial x_{n-1}} = \mathcal{P}_{n-2}.$$

It may be the case that more than one F_m vanishes when we take a partial derivative w.r.t. $\{x_n, x_{n-1}\}$, however they cannot all vanish simultaneously (as $\mathcal{P}_n$ contains $x_n \cdot x_{n-1}$). By Lemma 3.24 we have that the polynomials $\{\frac{\partial^2 F_m}{\partial x_n \partial x_{n-1}}\}$ are $\bar{0}-$weakly-justified ROPs. Hence, we obtain a representation of P_{n-2} as a sum of $0 < \hat{k} \leq k-1$ $\bar{0}-$weakly-justified ROPs such that $0 < \hat{k}^2 \leq (k-1)^2 = k^2 - 2k + 1 \leq n - 2 - (2k-3) \leq n-2$ which contradicts the induction hypothesis.

- For every $i \neq j \in [n]$ and $m \in [k]$ we have that $\frac{\partial^2 F_m}{\partial x_i \partial x_j} \not\equiv 0$. Thus, the polynomials $\{F_m\}_{m\in[k]}$ are $\bar{0}-$weakly-justified and multiplicative. In addition, for every $m \in [k]$ we have that $\text{var}(F_m) = [n]$. In particular $|\text{var}(F_m)| \geq 4$. Lemma 3.23 implies that $\forall m \in [k]$ there exist $j_m \in [n]$, $\alpha_m \in \mathbb{F}$ and a ROP $h_m(\bar{x})$ such that

$$\frac{\partial F_m}{\partial x_{j_m}} = (x_n - \alpha_m) h_m(\bar{x}).$$

Let $I = \{j_m | m \in [k]\}$. Clearly, $1 \leq |I| \leq k$ and $n \notin I$. Consider the following ROPs

$$F'_m \triangleq \partial_I F_m.$$

Then the ROPs F'_m's have the following properties.

1. By Lemma 3.24 we get that every F'_m is a $\bar{0}-$weakly-justified ROP.
2. $F'_m = (x_n - \alpha_m) h'_m(\bar{x})$ for some $h'_m(\bar{x})$. Indeed, as $j_m \in I$ we have that

$$F'_m = \partial_I F_m = \partial_{I\setminus\{j_m\}}(\frac{\partial F_m}{\partial x_{j_m}}) = \partial_{I\setminus\{j_m\}}\left((x_n - \alpha_m) h_m(\bar{x})\right) = (x_n - \alpha_m) \cdot \partial_{I\setminus\{j_m\}} h_m(\bar{x}).$$

3. For every $m \in [k]$ we have that $\alpha_m \neq 0$ and $h'_m(\bar{x})$ is a $\bar{0}-$weakly-justified ROP (this follows by Lemma 3.25 and the two properties we proved above).
4. For every $\alpha \neq \alpha_m \in \mathbb{F}$ we have that $F'_m|_{x_n=\alpha}$ is a non-zero $\bar{0}-$weakly-justified ROP. This can be easily concluded from property 3 since $F'_m|_{x_n=\alpha} = (\alpha - \alpha_m) h'_m(\bar{x})$.

Let

$$F''_m \triangleq \partial_I F_m|_{x_n=\alpha_k} = F'_m|_{x_n=\alpha_k} = (\alpha_k - \alpha_m) h'_m(\bar{x}).$$

As for every m in $[k]$ we have that h'_m is a $\bar{0}-$weakly-justified ROP then so does F''_m. In addition, we note that $F''_k \equiv 0$. W.l.o.g. let us assume that $I = \{\hat{n}+1, \hat{n}+2, \ldots, n-2, n-1\}$ for some $\hat{n}$. We get that

$$\sum_{m=1}^{k-1} F''_m = \partial_I \mathcal{P}_n|_{x_n=\alpha_k} = \alpha_k \cdot \mathcal{P}_{\hat{n}}.$$

That is, we have a representation of $\mathcal{P}_{\hat{n}}$ as a sum of $\bar{0}-$weakly-justified ROPs. We note that as $k \geq |I| = (n-1) - \hat{n}$, it follows that

$$\hat{k}^2 \leq (k-1)^2 = k^2 - 2k + 1 \leq k^2 - k - (n - 1 - \hat{n}) + 1 \leq n - k - n + 2 + \hat{n} \leq \hat{n}.$$

Therefore, we have found a representation of $\alpha_k \mathcal{P}_{\hat{n}}$ as a sum of $\hat{k} \leq \sqrt{\hat{n}}$ $\bar{0}-$weakly-justified ROPs. By our induction hypothesis we get that $\alpha_k = 0$. This is in contradiction to property 3 above. Hence, $\mathcal{P}_n$ cannot be represented as a sum of at most $\sqrt{n}$ $\bar{0}-$weakly-justified ROPs and the theorem is proved. □

8.4 Proofs of Theorems 4, 5, 6 and 8

We first give a generic algorithm and then show how to apply it in several different scenarios. We start by presenting the black-box version of Algorithm 11.

Theorem 8.4. *Let A be a hitting set for the ROPs $F_1, \ldots, F_k$* [13]*. Let $\mathcal{J}_A^k$ be as defined in Section 5.2.1. Then Algorithm 12 determines whether $\sum_{m=1}^{k} F_m \equiv 0$, and its running time is $|\mathcal{J}_A^k| \cdot \binom{n}{k^2} = \text{poly}(n, |A|) \cdot n^{O(k^2)}$.*

Algorithm 12 Black-Box PIT algorithm for sum of read-once formulas

Input: Black-box holding F, a sum of k ROFs $F_1, \ldots, F_k$ (i.e. $F = F_1 + \cdots + F_k$).
Output: "true" if $F \equiv 0$ and "false" otherwise.

```
for all γ̄ ∈ J_A^k {defined in Section 5.2.1} do
    if F_γ̄|_{A^n_{k^2}({0,1})} ≢ 0 then
        return false
{Everything is OK}
return true
```

Proof. The claim regarding the running time is clear from the definition of A in Section 5.2.1. We now show the correctness of the algorithm. Clearly if $F \equiv 0$ then the algorithm is correct. Otherwise, by the discussion in Section 5.2.2 there exists $\bar{\gamma} \in \mathcal{J}_A^k$ such that $F_{\bar{\gamma}} = \sum_{m=1}^{k} (F_m)_{\bar{\gamma}}$ is a sum of $\bar{0}-$justified ROPs. By Theorem 8.2 it must be the case that $F_{\bar{\gamma}}|_{\mathcal{A}^n_{k^2}(\{0,1\})} \not\equiv 0$, and in particular the algorithm finds a non zero input for F. □

We now show how to implement Algorithm 12 is several scenarios. We first implement the algorithm for the case that the F_m's are general ROFs.

Proof of Theorem 4. Lemma 3.19 implies that $\partial\text{ROF} \subseteq \text{ROF}$. Hence, we can invoke Algorithm 12 with A taken from Algorithm 10 to obtain a black-box PIT algorithm for the sum of k ROPs that runs in time $(n^3k)^{\sqrt{n}} \cdot n^{\mathcal{O}(k^2)} = n^{\mathcal{O}(k^2+\sqrt{n})}$. □

The next case is when all the F_m's are bounded depth ROFs.

Proof of Theorem 5. As $\partial\text{AROF}(d) \subseteq \text{AROF}(d)$ (Lemma 7.3) we can invoke Algorithm 12 with A taken from Theorem 7.10 to obtain a black-box PIT algorithm for the sum of k ROPs of depth d with running time $(n^3k)^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(k^2)} = n^{\mathcal{O}(k^2+d)}$ [14]. □

The final result in this vein is a black-box PIT algorithm for the case where the black box holds a sum of ROFs that is a read-k formula (that is, every input variable appears in at most k of the formula).

Definition 8.5. *For $F = \sum_{m=1}^{\ell} F_m$, where each F_m is a ROF, we say that F is* read-k sum *if for each $i \in [n]$ there are at most k functions F_m that depend on x_i. In other words, each variable is read at most k times in F.*

[13] That is, there is some $\bar{a} \in A$ such that for every m in $[k]$ we have that $F_m(\bar{a}) \neq 0$.

[14] Notice that $d \leq n$ implies $k^{\mathcal{O}(d)} = n^{\mathcal{O}(k^2+d)}$.

We can easily extend a PIT algorithm for sum of k ROFs to a PIT algorithm for read-k sum with the following observation:

Observation 8.6. *Let F be a read-k sum then $\frac{\partial F}{\partial x_i}$ is a sum of (at most) k ROFs, for every $i \in [n]$.*

Proof of Theorem 6. The proof immediately follows from the observation and from Theorems 3, 4 and 5. □

So far we have several versions of PIT algorithms for sum of k ROFs model. We now show a generalization of those PIT algorithms to read-once testing algorithms.

Proof of Theorem 8. As previously, let $F = F_1 + F_2 + \cdots + F_k$. For each of the claims in the theorem use Algorithm 6 to obtain a justifying assignment for F. Next, construct a candidate ROF $\hat{F}$ by applying Algorithm 1. Given $\hat{F}$ we simply need to verify that $\hat{F} = F_1 + F_2 + \cdots + F_k$ or, in other words, that $F_1 + F_2 + \cdots + F_k - \hat{F} = 0$. This is exactly a PIT of sum of $k+1$ ROFs [15]. Note, that the running times of the read-once testing algorithms remain asymptotically similar to the running times of the corresponding PIT algorithms.

As in the proof of Theorem 6 we can use Observation 8.6 to generalize Theorem 8 to the case where F is a read-k sum of ROFs. □

References

[Agr05] M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the 25th FSTTCS*, volume 3821 of *Lecture Notes in Computer Science*, pages 92–105, 2005.

[AHK93] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. *J. ACM*, 40(1):185–210, 1993.

[Alo99] N. Alon. Combinatorial nullstellensatz. *Combinatorics, Probability and Computing*, 8:7–29, 1999.

[AM07] V. Arvind and P. Mukhopadhyay. The ideal membership problem and polynomial identity testing. *ECCC Report TR07-095*, 2007.

[BB98] D. Bshouty and N. H. Bshouty. On interpolating arithmetic read-once formulas with exponentiation. *J. of Computer and System Sciences*, 56(1):112–124, 1998.

[BC98] N. H. Bshouty and R. Cleve. Interpolating arithmetic read-once formulas in parallel. *SIAM J. on Computing*, 27(2):401–413, 1998.

[BHH95a] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning arithmetic read-once formulas. *SIAM J. on Computing*, 24(4):706–735, 1995.

[BHH95b] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. *Journal of Computer and System Science*, 50:521–542, 1995.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation. In *Proceedings of the 20th Annual STOC*, pages 301–309, 1988.

[15] Since in the bounded-depth version $\hat{F}$ might be an AROF of a higher depth, will we use $\mathcal{J}^{k+1}_{A'}$ for the verification when $A' \triangleq A \cup \{\bar{a}\}$, A is the hitting set and $\bar{a}$ is a justifying assignment for $\hat{F}$.

[BW05] A. Bogdanov and H. Wee. More on noncommutative polynomial identity testing. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 92–99, 2005.

[DS06] Z. Dvir and A. Shpilka. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. on Computing*, 36(5):1404–1434, 2006.

[GKS90] D. Grigoriev, M. Karpinski, and M. F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. on Computing*, 19(6):1059–1063, 1990.

[HH91] T. R. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. In *Proceedings of the 4th Annual COLT*, pages 326–336, 1991.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.

[KLN+93] M. Karchmer, N. Linial, I. Newman, M. E. Saks, and A. Wigderson. Combinatorial characterization of read-once formulae. *Discrete Mathematics*, 114(1-3):275–282, 1993.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual STOC*, pages 216–223, 2001.

[KS07a] Z. Karnin and A. Shpilka. Black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. *ECCC Report TR07-042*, 2007.

[KS07b] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

[Lho91] B. Lhotzky. *On the computational complexity of some algebraic counting problems*. Ph.D. thesis, University of Bonn, Department of computer Science, Bonn, Germany, 1991.

[LV03] R. J. Lipton and N. K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the 14th annual SODA*, pages 756–760, 2003.

[Nis91] N. Nisan. Lower bounds for non-commutative computation. In Baruch Awerbuch, editor, *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 410–418, New Orleans, LS, May 1991. ACM Press.

[RS05] R. Raz and A. Shpilka. Deterministic polynomial identity testing in non commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *JACM*, 27(4):701–717, 1980.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation*, pages 216–226. 1979.