Complete Derandomization of Identity Testing and Reconstruction of Read-Once Formulas*

Daniel Minahan¹ and Ilya Volkovich²

¹ Departments of Mathematics and EECS, CSE Division, University of Michigan, Ann Arbor, MI, USA
dminahan@umich.edu

² Department of EECS, CSE Division, University of Michigan, Ann Arbor, MI, USA
ilyavol@umich.edu

Abstract

In this paper we study the identity testing problem of arithmetic read-once formulas (ROF) and some related models. A read-once formula is a formula (a circuit whose underlying graph is a tree) in which the operations are \{+, ×\} and such that every input variable labels at most one leaf. We obtain the first polynomial-time deterministic identity testing algorithm that operates in the black-box setting for read-once formulas, as well as some other related models. As an application, we obtain the first polynomial-time deterministic reconstruction algorithm for such formulas. Our results are obtained by improving and extending the analysis of the algorithm of [52].

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity

Keywords and phrases Derandomization, Read-Once Formulas, Identity Testing, Arithmetic Circuits, Reconstruction

Digital Object Identifier 10.4230/LIPIcs.CCC.2017.32

1 Introduction

In this paper we study the problem of Polynomial Identity Testing (PIT): given an arithmetic circuit \(C\) over a field \(F\), with input variables \(x_1, x_2, \ldots, x_n\), determine whether \(C\) computes the identically zero polynomial. Given its connections to a wide range of problems, PIT is considered a central problem in algebraic complexity theory and algorithms design. Particular instances include: perfect matchings in graphs [40, 43, 20], primality testing [2], \(\text{IP} = \text{PSPACE}\) [41, 48] the PCP theorem [10, 9] and many more. PIT is one of a few natural problems which have a simple efficient randomized algorithm [18, 47, 55] but lack a deterministic one. Indeed, it has been a long standing open question to come up with an efficient deterministic algorithm for this problem.

In this paper we consider the PIT problem in the black-box setting. In this setting, one is not given the full description of the circuit \(C\) but only allowed black-box (oracle) access to \(C\). The problem of derandomizing identity testing in this setting reduces to that of finding for every \(s\) an explicit set of points \(\mathcal{H} \subseteq F^n\) of size \(\text{poly}(s)\) such that any non-zero circuit of size \(s\) does not vanish on \(\mathcal{H}\). We refer to such sets as hitting sets. Indeed, the randomized algorithm of [18, 47, 55] provides an exponential-size hitting set. Furthermore, applying

* Research partially supported by NSF.
standard probabilistic arguments one can show existence of “small” hitting sets. Yet, coming up with an explicit hitting set is believed to be very difficult task as it would immediately imply explicit exponential lowers bounds [30, 1].

Yet, for several restricted classes of arithmetic circuits, efficient deterministic black-box PIT algorithms were found. For example, efficient black-box PIT algorithms were shown for depth-2 arithmetic circuits [13, 36, 39] and depth-3 arithmetic circuits with bounded top fan-in (also known as $\Sigma\Pi\Sigma(k)$ circuits) [19, 35, 34, 11, 45, 33, 46, 3]. There has also been a lot of progress on PIT for restricted classes of depth-4 circuits [44, 11, 12, 3, 32, 26, 7, 42, 52, 37, 38]. Another body of research has been focused on PIT algorithms for bounded-read models. That is, classes of circuits where each variable appears some bounded number of times [22, 4, 24, 23, 21, 7, 52, 28, 6, 27, 20], with the simplest case being the read-once formulas.

A read-once formula (ROF for short) is an arithmetic formula (i.e. a tree) in which the operations are $\{+, \times\}$ and such that every input variable labels at most one leaf. These formulas can be thought of as the smallest formulas that depend on all their variables and the simplest non-trivial subclass of multilinear formulas. Although ROFs form a very restricted model of computation they have received a lot of attention in both the Boolean [31, 8, 17] and the algebraic [29, 16, 14, 15, 51, 52, 54] worlds.

While ROFs have a trivial polynomial (and, in fact, linear-time) white-box PIT algorithm, the first sub-exponential time $n^{O(\sqrt{n})}$ black-box PIT algorithm for ROFs was given in [49]. Later on, in [52] the result was improved to $n^{O(\log n)}$ via another algorithm. A different analysis for the latter algorithm, resulting in roughly the same run time, was given in [7]. Yet, despite the rich body of work devoted to the problem, prior to our work no polynomial-time black-box PIT algorithm was known even for ROFs. In this paper we give the first black-box PIT algorithm for ROFs, and some related classes of formulas, thus achieving a complete derandomization of the PIT problem for these classes. For more information on PIT we refer the reader to the survey [53].

It is important to point out that while PIT asks whether the resulting polynomial is identically zero as a formal sum of monomials, some non-identically zero polynomials might evaluate to the zero function. For example, $x^5 - x$ will always evaluate to zero over the field of five elements. For this reason we will allow our algorithm to evaluate the polynomial on elements from a polynomially large extension field of $\mathbb{F}$. In [52] it was shown one cannot achieve polynomial-time black-box PIT algorithms if $|\mathbb{F}| = o(n/\log n)$.

1.1 Our Results

In this section we describe and discuss our results. In fact, our results hold for the slightly richer class of preprocessed read-once formulas. A preprocessed ROF (PROF for short) is a ROF in which we are allowed to replace each variable $x_i$ with a univariate polynomial $T_i(x_i)$. A polynomial $P(\bar{x})$ is a Preprocessed Read-Once Polynomial (PROP for short) if it can be computed by a preprocessed read-once formula. This PROPs also generalize the “sum-of-univariates” model. (see Section 3.2 for a formal definition). We begin with our main result: polynomial-time black-box PIT algorithm for PROFs.

Theorem 1. Let $n, d \in \mathbb{N}$. There exist a deterministic algorithm that given black-box (oracle) access to a preprocessed read-once formula $\Phi$ on $n$ variables and individual degrees (of the preprocessing) at most $d$, checks whether $\Phi \equiv 0$. The running time of the algorithm is polynomial in $n$ and $d$.

\footnote{Conference version first appeared in [50].}
In [52] it was shown how to extend a PIT algorithm for a single PROF into a PIT algorithm for a sum of PROFs. By plugging in our main result we obtain a black-box PIT algorithm for sums of PROFs.

Theorem 2. Let \( k, n, d \in \mathbb{N} \). There exist a deterministic algorithm that given black-box (oracle) access to \( \Phi = \Phi_1 + \ldots + \Phi_k \), where the \( \Phi_i \)-s are preprocessed read-once formulas in \( n \) variables, with individual degrees at most \( d \), checks whether \( \Phi \equiv 0 \). The running time of the algorithm is \( (nd)^{O(k)} \).

Observe that for a fixed \( k \in \mathbb{N} \) the algorithm runs in polynomial time with respect to \( n \) and \( d \). Furthermore, observe that if \( \mathcal{H} \) is hitting set for a sum of two PROPs then \( \mathcal{H} \) is an interpolating set for a single PROF. That is, the values of a single PROF \( P \) on \( \mathcal{H} \) contain enough information to uniquely identify \( P \). Indeed, a consequence of Theorem 2 is an interpolating set of polynomial size for PROPs. However, in general, \( \mathcal{H} \) does not provide us with an efficient algorithm to reconstruct a corresponding PROF.

In [16], a randomized polynomial-time reconstruction algorithm for ROFs was given. In [51], the algorithm was extended to PROFs. Moreover, it was shown how to convert a black-box PIT algorithm into a reconstruction algorithm paying a polynomial overhead. Indeed, by plugging in the result of [52], the first deterministic sub-exponential (and, in fact, quasi-polynomial) time reconstruction algorithm for PROFs was given. By plugging in our main result, we achieve a complete derandomization of the reconstruction algorithm by obtaining a deterministic polynomial-time reconstruction algorithm for PROFs.

Theorem 3. There exist a deterministic algorithm that given black-box (oracle) access to a preprocessed read-one formula \( \Phi \), on \( n \) variables and individual degrees at most \( d \), reconstructs \( \Phi \). Namely, the algorithm outputs a PROF \( \hat{\Phi} \) that computes the same polynomial. The running time of the algorithm is polynomial in \( n \) and \( d \).

1.2 Organization

The paper is organized as follows. In Section 2 we give the basic definitions and notations. In Section 3 we formally introduce ROFs and its generalizations along with some structural properties, and in Section 3.3 we discuss the PIT algorithm of [52]. In Section 3.4 we prove some additional properties of the algorithm, which is the main technical contribution of the paper. Next, in Section 4 we give our main result, thus proving Theorem 1. We discuss the applications of our main result in Section 5 proving Theorems 2 and 3. We conclude the paper with some open questions in Section 6.

2 Preliminaries

For a positive integer \( n \), let \([n]\) denote the set \( \{1, \ldots, n\} \). We now give some definitions that apply to polynomials \( P, Q \in \mathbb{F}[x_1, \ldots, x_n] \). For a polynomial \( P \), a variable \( x_i \), and \( \alpha \in \mathbb{F} \), let \( P|_{x_i=\alpha} \) denote the polynomial that results upon setting \( x_i = \alpha \). We say that \( P \) depends on \( x_i \) if there exist \( \bar{a}, \bar{b} \in \mathbb{F}^n \) that differ in only the \( i \)-coordinate such that \( P(\bar{a}) \neq P(\bar{b}) \). We denote var\( (P) \) \( \triangleq \{ i : P \) depends on \( x_i \} \). Intuitively, \( P \) depends on \( x_i \) if \( x_i \) appears when \( P \) is written as a sum of monomials.

We say that \( P \) is a homogeneous polynomial if all monomials in \( P \) has the same total degree. For \( i \in \mathbb{N} \) we define \( H_i[P] \) as the homogeneous part of degree \( i \) of \( P \). That is, all the monomials of total degree \( i \) that appear in \( P \). If \( P \) does not have monomials of degree \( i \) then
$H_i[P] \equiv 0$. We say that $P$ and $Q$ are similar and denote $P \sim Q$ if there exist $\alpha \in \mathbb{F} \setminus \{0\}$ such that $\alpha \cdot P = Q$.

In order to actually calculate the complexity of our algorithm we need to define a formal model of computation for polynomials.

**Definition 4 (Arithmetic formula).** An arithmetic formula is a binary tree where each leaf is labeled with a variable $x_i \in \{x_1, \ldots, x_n\}$ and each internal node, called a gate, is labeled with an operation $+$ or $\times$. Additionally, each leaf and node are labeled with some $(\alpha, \beta) \in \mathbb{F}^2$.

The tree is evaluated by recursively calculating the values of the left subtree $P_1$ and the right subtree $P_2$ and then combining them by $\alpha (P_1 \text{ op } P_2) + \beta$ where op is the operation at the top gate.

The efficiency of a formula over a set of formulas $\mathcal{C}$ is measured by the number of gates in $\mathcal{C}$. Thus when we say polynomial time, we mean polynomial in the number of gates.

Often times, we will implicitly associate a class of formulas $\mathcal{C}$ with the class of polynomials computed by these formulas.

We consider formulas in the black-box (or oracle) setting. That is, the algorithm cannot directly look at the formula and is only allowed to query the polynomial computed by the formula on $\mathbb{F}^n$. Hereafter, we assume that an evaluation query can be carried out in $O(1)$ time. In case $\mathbb{F}$ is small, we allow to query the formula on a polynomially-large extension field of $\mathbb{F}$.

### 2.1 Generators and Hitting Sets

Our black-box PIT algorithms use the notion of generators. In this section, we formally define this notion, describe a few of its useful properties and give the connection to hitting sets. Intuitively, a generator $G$ for a polynomial class $\mathcal{C}$, is a function that stretches $t$ independent variables into $n \gg t$ dependent variables that can be plugged into any polynomial $P \in \mathcal{C}$ without causing it to vanish. Recall that a hitting set $\mathcal{H} \subseteq \mathbb{F}^n$ for a class of polynomials $\mathcal{C}$ is a set such that for any nonzero polynomial $P \in \mathcal{C}$, there exists $\bar{a} \in \mathcal{H}$, such that $P(\bar{a}) \neq 0$.

**Definition 5 (Hitting Set).** Let $\mathcal{C}$ be a class of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. A set $\mathcal{H}$ is called a hitting set for $\mathcal{C}$ provided that $\forall P \in \mathcal{C}$ with $P \neq 0$ we have that $P|_{\mathcal{H}} \neq 0$.

This leads us to a basic algorithm for PIT.

**Lemma 6.** Let $\mathcal{C}$ be a class of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ and let $\mathcal{H}$ be a hitting set for $\mathcal{C}$. Then there exists a deterministic PIT algorithm for $\mathcal{C}$ that runs in time $O(|\mathcal{H}|)$.

The following generalization of the fundamental theorem of algebra provides hitting sets of exponential size for every polynomial. A proof can be found in [5].

**Lemma 7.** Let $P \neq 0 \in \mathbb{F}[x_1, \ldots, x_n]$ and suppose the individual degree of any variable in $P$ is bounded by some $d \in \mathbb{N}$. Pick $S \subseteq \mathbb{F}$ with $|S| > d$. Then $P|_{S^n} \neq 0$.

**Remark.** The precondition of the lemma implies that $|\mathbb{F}| > d$. In case that $\mathbb{F}$ is small, this assumption is met by choosing elements from an appropriately large extension field of $\mathbb{F}$.

A related notion is the notion of generators. Many hitting sets are constructed by means of generators.

**Definition 8 (Generator).** Let $\mathcal{C}$ be a class of polynomials over $\mathbb{F}$. A polynomial map $G : \mathbb{F}^t \rightarrow \mathbb{F}^n$ is a generator for $\mathcal{C}$ provided that $\forall P \neq 0 \in \mathcal{C}$ we have $P(G) \neq 0$. 
Intuitively, a generator $\mathcal{G}$ for $\mathcal{C}$ is a polynomial mapping that has a hitting set for $\mathcal{C}$ in its image. More specifically, Lemma 7 allows us to convert a generator into a hitting set by observing that a polynomial composed with a polynomial map results in another polynomial. Such composition typically reduces the number of variables that the polynomial depends on, but it may increase the total degree. Thus, since the size of the hitting set produced by Lemma 7 depends on both parameters, we want to find a generator that reduces the number of variables without drastically increasing its degree.

### 3 Read-Once Formulas

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

#### 3.1 Read-Once Formulas and Read-Once Polynomials

Most of the definitions that we give in this section are from [29] and [52] or some small variants. We start by formally defining the notions of a read-once formula and a read-once polynomial.

▶ **Definition 9** (Read-Once Formula). A read-once formula (ROF) is an arithmetic formula where each variable appears at most once. A polynomial $P(\bar{x})$ is a read-once polynomial (ROP for short) if it can be computed by a read-once formula.

Clearly, ROPs form a subclass of multilinear polynomials. In addition, note that the number of gates in a ROF is at most twice the number of variables. This means that our complexity scales with $n$, so we need only be concerned about how the runtime of our algorithm scales with respect to the number of variables. Thus, our ideal efficiency for an algorithm is $n^{O(1)}$. The next lemma also follows easily from the definition.

▶ **Lemma 10** (ROP Structural Lemma). Every ROP $P(\bar{x})$ that depends on at least two variables can be presented in exactly one of the following forms:

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

where $P_1$ and $P_2$ are non-constant, variable-disjoint ROPs and $c \in \mathbb{F}$ is a constant.

#### 3.2 Preprocessed Read-Once Polynomials

In this section we extend the model of ROFs by allowing a preprocessing step of the input variables. While the basic model is read-once in its variables, the extended model can be considered as read-once in univariate polynomials. In addition, this model generalizes the “sum-of-univariates” model, which, in particular, contains the “sum-of-squares” model.

▶ **Definition 11.** A preprocessing is a transformation $T(\bar{x}) : \mathbb{F}^n \to \mathbb{F}^n$ of the form $T(\bar{x}) \triangleq (T_1(x_1), T_2(x_2), \ldots, T_n(x_n))$ such that each $T_i$ is a non-constant univariate polynomial.

Notice that preprocessings do not affect the PIT problem in the white-box setting as for every $n$-variate polynomial $P(\bar{y})$ it holds that $P(\bar{y}) \equiv 0$ if and only if $P(T(\bar{x})) \equiv 0$. We now give a formal definition and list some immediate properties.
**Definition 12.** A **preprocessed arithmetic read-once formula** (PROF for short) over a field $F$ in the variables $\bar{x} = (x_1, \ldots, x_n)$ is a binary tree whose leaves are labelled with non-constant univariate polynomials $T_1(x_1), T_2(x_2), \ldots, T_{n}(x_{n})$ (all together forming a preprocessing) and whose internal nodes are labelled with the arithmetic operations $\{+, \times\}$ and with a pair of field elements $(\alpha, \beta) \in F^2$. Each $T_i$ can label at most one leaf. The computation is performed in the following way. A leaf labelled with the polynomial $T_i(x_i)$ and with $(\alpha, \beta)$ computes the polynomial $\alpha \cdot T_i(x_i) + \beta$. If a node $v$ is labelled with the operation $\op$ and with $(\alpha, \beta)$, and its children compute the polynomials $\Phi_{v_1}$ and $\Phi_{v_2}$ then the polynomial computed at $v$ is $\Phi_v = \alpha \cdot (\Phi_{v_1} \op \Phi_{v_2}) + \beta$.

A polynomial $P(\bar{x})$ is a **Preprocessed Read-Once Polynomial** (PROP for short) if it can be computed by a preprocessed read-once formula. A **Decomposition** of a polynomial $P$ is a pair $Q(\bar{x}), T(\bar{x})$ such that $P(\bar{x}) = Q(T(\bar{x}))$ when $Q$ is a ROP and $T$ is a preprocessing. An immediate consequence from the definition is that each PROP admits a decomposition. The following lemma is the PROPs analog of Lemma 10.

**Lemma 13 (PROP Structural Lemma).** Every PROP $P(\bar{x})$ with \(|\var(P)| \geq 2\) can be presented in one of the following forms:
1. $P(\bar{x}) = P_1(\bar{x}) + P_2(\bar{x})$,
2. $P(\bar{x}) = P_1(\bar{x}) \cdot P_2(\bar{x}) + c$,

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

### 3.3 The Algorithm of [52]

In this paper we improve the complexity analysis of the PIT algorithm of [52]. We begin by describing their algorithm. The heart of the algorithm is a construction of polynomial map $G_{n, t}$ which is shown to be a generator for PROPs for a certain range of parameters.

As in [52], we fix a set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq F$ of $n$ distinct elements. It is also assumed that in case that $F$ is small, we have access to some extension field of $F$ with more than $n$ elements. As was shown in [52], this assumption is necessary in order to achieve a polynomial-time algorithm.

**Definition 14 (The generator of [52]).** Let $t \in \mathbb{N}$. For each $i \in [n]$ let $L_i(y)$ denote the $i$-th Lagrange Interpolation polynomial. Formally: $L_i(y) = \prod_{j=1, j \neq i}^n \frac{(y - \alpha_j)}{(\alpha_i - \alpha_j)}$. That is, $L_i(y)$ is a degree $n - 1$ polynomial satisfying: $L_i(\alpha_j) = 1$ when $j = i$ and $L_i(\alpha_j) = 0$ when $j \neq i$. For each $i \in [n]$, let $G^t_{i}(y_1, \ldots, y_l, z_1, \ldots, z_l) \triangleq \sum_{k=1}^l L_i(y_k) \cdot z_k$. Finally, let $G_{n, t}(y_1, \ldots, y_l, z_1, \ldots, z_l) \triangleq (G^t_1(y_1, \ldots, z_l), \ldots, G^t_l(y_1, \ldots, z_l))$.

$G_{n, t}$ can be seen as a sum of $t$ variable-disjoint copies of $G_{n, 1}$. This can be seen as the algebraic analogue of $t$-wise independent bits. The main part of the analysis of the algorithm is to establish that for every $n \in \mathbb{N}$ the map $G_{n, \log n}$ is a generator for PROPs on $n$ variables.

**Lemma 15 ([52]).** Let $P \in F[x_1, \ldots, x_n]$ be a non-constant PROP. Then $P(G_{n, \log n})$ is non-constant.

The intuition behind the proof is that a PROP can be written as either a sum or a product of two variable-disjoint polynomials (Lemma 13). Hence, (at least) one of these polynomials contains at most half of the variables. The map $G_{n, t}$ allows to “move” to a smaller polynomial by “shaving” a copy of $G_{n, 1}$. Finally, applying Lemma 7 one could show that if $G_{n, t}$ is a generator for a class of polynomials, then it can be converted into a relatively small hitting set for that class.
Lemma 16. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial of degree $d$ such that $P(G_{n,1}) \neq 0$ for some $t, d \in \mathbb{N}$. Then $P$ has a hitting set of size $(nd)^{O(1)}$.

Consequently, the result of Lemma 15 translates into a hitting set of size $(nd)^{O(\log n)}$ for PROPs. We note that the generator of [52] has been used as an ingredient in some subsequent PIT algorithms (e.g. [21, 7, 6, 25]).

3.4 Our Technical Contribution

In this section we explore additional properties of the generator of [52]. The main observation is a structural property of the generator when applied to a polynomial that depends only on a “small” subset of variables. This is the main technical contribution of the paper.

Let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be the set of elements that is used to define the generator.

Definition 17. For $I \subseteq [n]$, define $\Phi_I(y) \overset{\Delta}{=} \prod_{i \in I}(y - \alpha_i)$. For notational convenience, $\Phi_{\emptyset}(y) \overset{\Delta}{=} 1$.

In order to provide some intuition for the definition, we observe that for any $A$, let $\delta$ be an upper bound on the individual degrees of all variables $x_i$ in $P$. Then there exists a polynomial $P'(y)$ of degree at most $\delta \cdot |\var(P)| - d$ such that

$$P(G_{n,1}(y, z)) = z^d \cdot \Phi_{\{n\}}^d(y) \cdot P'(y) \cdot \Phi_{\var(P) \setminus \{n\}}^d(y).$$

In particular, there exist a polynomial $P''(y)$ of degree at most $d \cdot (|\var(P)| - 1)$ such that

$$P(G_{n,1}(y, z)) = z^d \cdot P''(y) \cdot \Phi_{\var(P) \setminus \{n\}}^d(y).$$

Proof. Let $V \subseteq [n]$ and let $m(\bar{x}) = \alpha \prod_{i \in V} x_i^{e_i}$ be a monomial s.t. $\sum_{i \in V} e_i = d$ and $\forall i \in V : 0 \leq e_i \leq \delta$.

$$m(G_{n,1}(y, z)) = \alpha z^d \cdot \prod_{i \in V} L_i(y) = \beta z^d \cdot \prod_{i \in V} \Phi_{\{n\} \setminus \{i\}}^d(y) = \beta z^d \cdot \Phi_{\{n\} \setminus \{i\}}^d(y) \cdot \prod_{i \in V} (y - \alpha_i)^{e_i} \cdot \Phi_{\var(P) \setminus \{n\}}^d(y) = \beta z^d \cdot \Phi_{\{n\} \setminus \{i\}}^d(y) \cdot \prod_{i \in V} (y - \alpha_i)^{e_i} \cdot \Phi_{\var(P) \setminus \{n\}}^d(y).$$

Take $m'(y) = \beta \prod_{i \in V} (y - \alpha_i)^{d - e_i}$ and observe that degree of $m'(y)$ is $\delta \cdot |V| - d$. By definition, the polynomial $P$ consists of a sum of such monomial where $V = \var(P)$. Therefore, the first claim follows by a linearity argument. The second claim follows by observing that $d$ is an upper bound on the individual degrees of all variables $x_i$ in $P$, so we can set $\delta = d$.

4 Main Result

In this section we prove our main result Theorem 1. We begin by showing that $P(G_{n,1})$ hits sums of univariate polynomials. This proof is available in [52] but we reproduce it here for completeness.
Lemma 19. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-constant polynomial of the form $P = \sum_{i=1}^{n} T_i(x_i)$. Then $P(G_{n,1}(y,z))$ is non-constant.

Proof. Pick $x_i$ such that $T_i(x_i)$ is non-constant. Observe that: $P(G_{n,1})|_{y=\alpha_i} = T_i(z) + \sum_{j \neq i} T_j(0)$. This is a non-constant polynomial and so $P(G_{n,1})$ is non-constant as well.

We now move to the proof of our main result. We want to show that $G_{n,1}$ is a generator for the set of PROPs. The idea is to proceed by induction using Lemma 13. Recall that for any $P \in \mathbb{F}[x_1, \ldots, x_n]$ and $i \in \mathbb{N}$, $H_i[P]$ denotes the homogeneous part of degree $i$ of $P$. Consequently, we can write $P = \sum_{i=0}^{d} H_i[P]$.

Theorem 20. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-constant PROP. Then $P(G_{n,1})$ is non-constant.

Proof. Let $d$ denote the total degree of $P$. We induct on $m = |\text{var}(P)|$. The base case where $m = 1$ follows from Lemma 19. Now, suppose that $m \geq 2$. By the PROP Structural Lemma (Lemma 13) we have two cases.

1. $P = P_1 \cdot P_2 + c$. Note that $|\text{var}(P_1)|, |\text{var}(P_2)| \leq m - 1$, so by the inductive hypothesis $P_1(G_{n,1})$ and $P_2(G_{n,1})$ are non-constant polynomials and hence their product is non-constant as well. Adding a constant does not affect this.

2. $P = P_1 + P_2$. For $j = 1, 2$: we can write $P_j = \sum_{i=0}^{d} P_{i,j}$ where $P_{i,j} = H_i[P_j]$. By Lemma 18, for each $0 \leq i \leq d$ and $j = 1, 2$ there exists a polynomial $P'_{i,j}(y)$ of degree at most $i \cdot (|\text{var}(P_{i,j})| - 1)$ such that

$$P_j(G_{n,1}(y,z)) = \sum_{i=0}^{d} P_{i,j}(G_{n,1}(y,z)) = \sum_{i=0}^{d} z^i \cdot P'_{i,j}(y) \cdot \Phi_{\text{var}(P_{i,j})}(y)$$

and hence

$$P(G_{n,1}(y,z)) = \sum_{i=0}^{d} z^i \cdot \left( P'_{1,1}(y) \cdot \Phi_{\text{var}(P_{1,1})}(y) + P'_{1,2}(y) \cdot \Phi_{\text{var}(P_{1,2})}(y) \right). \quad (1)$$

As before, by the inductive hypothesis $P_1(G_{n,1})$ and $P_2(G_{n,1})$ are non-constant polynomials. Therefore, there exist $1 \leq k \leq d$ such that

$$z^k \cdot P_{k,1}(y) \cdot \Phi_{\text{var}(P_{k,1})}(y) \neq 0$$

and in particular $P'_{k,1}(y) \neq 0$. Let us denote $V_j = \text{var}(P_{k,j})$ and $W = [n] \setminus (V_1 \cup V_2)$. Consider the expression that corresponds to the $z^k$ term in Equation 1:

$$P'_{k,1}(y) \cdot \Phi_{\text{var}(P_{k,1})}(y) + P'_{k,2}(y) \cdot \Phi_{\text{var}(P_{k,2})}(y) \quad (2)$$

As $P_{k,1}$ and $P_{k,2}$ are variable-disjoint, Equation 2 can be rewritten as:

$$P'_{k,1}(y) \cdot \Phi_{V_1 \cup W}(y) + P'_{k,2}(y) \cdot \Phi_{V_1 \cup W}(y) = \Phi_{V_1}(y) \cdot (P'_{k,1}(y) \cdot \Phi_{V_1}(y) + P'_{k,2}(y) \cdot \Phi_{V_1}(y))$$

The last equality follows from the properties of $\Phi$ (see Definition 17).

We claim that the obtained expression is non-constant. To this end, it sufficient to show that $P'_{k,1}(y) \cdot \Phi_{V_1}(y) + P'_{k,2}(y) \cdot \Phi_{V_1}(y) \neq 0$. Assume the contrary. We obtain that
\[ P'_{k,1}(y) \cdot \Phi^k_{V_2}(y) = -P'_{k,2}(y) \cdot \Phi^k_{V_1}(y). \] As \( V_1 \) and \( V_2 \) are disjoint sets, \( \Phi^k_{V_1}(y) \) and \( \Phi^k_{V_2}(y) \) have no common roots. Therefore, it must be the case that \( \Phi^k_{V_1} \) divides \( P'_{k,1} \). As \( P'_{k,1} \neq 0 \), we get that

\[ \deg \left( P'_{k,1} \right) \geq \deg \left( \Phi^k_{V_1} \right) = k |V_1| \]

while by Lemma 18, \( P'_{k,1}(y) \) is a polynomial of degree at most \( k \cdot (|V_1| - 1) \). Consequently, the coefficient of \( z^k \) in \( P(G_{n,1}(y,z)) \) is non-constant and the claim follows.

Theorem 1 follows by combining Theorem 20 with Lemma 16.

5 Applications

In this section we show two application for our main result, proving Theorems 2 and 3. The first application is testing whether several PROPs sum up to the zero polynomial. To this end, we require the following result which shows that a generator for the class of PROPs can be extended to yield a generator for the class of sums of PROPs.

\begin{itemize}
  \item \textbf{Lemma 21} ([52]). Let \( G_n \) be a generator for PROPs on \( n \) variables. Then for any \( k \in \mathbb{N} \), \( G_n + G_{n,3k} \) is a generator for sums of \( k \) PROPs on \( n \) variables.
  \item \textbf{Remark.} As both \( G_n \) and \( G_{n,3k} \) represent polynomial maps with the same output length, the sum \( G + G_{n,3k} \) should be interpreted as component-wise sum, where we implicitly assume the variables of \( G_n \) and \( G_{n,3k} \) have been relabelled so as to be disjoint.
\end{itemize}

The next corollary follows by combining Lemma 21 with Theorem 20 and the properties of \( G_{n,t} \) (see Definition 14).

\begin{itemize}
  \item \textbf{Corollary 22.} For any \( k,n \in \mathbb{N} \), the map \( G_{n,3k+1} \) is a generator for sums of \( k \) PROPs on \( n \) variables.
\end{itemize}

Theorem 2 follows by applying Lemma 16. Observe that if \( H \) is hitting set for a sum of two PROPs then \( H \) is an interpolating set for a single PROP. That is, the values of a single PROP \( P \) on \( H \) contain enough information to uniquely identify \( P \). Indeed, a consequence of Theorem 2 is an interpolating set of polynomial size for PROPs. However in general, \( H \) does not provide us with an efficient algorithm to reconstruct a corresponding PROF. In [51] it was shown how to use an interpolating set to devise a reconstruction algorithm with a polynomial overhead.

\begin{itemize}
  \item \textbf{Lemma 23} ([51]). Let \( n,d \in \mathbb{N} \). There exists a deterministic algorithm that given a hitting set \( H_{n,d} \) for PROPs on \( n \) variable and degree at most \( d \), and black-box (oracle) access to a PROP \( P \) as above, outputs a PROF \( \Phi \) that computes \( P \), in time polynomial in \( n,d \) and \(|H_{n,d}|\).
\end{itemize}

Combining the Lemma with Theorem 1 results in Theorem 3.

6 Conclusions & Open Questions

In this paper we present the first polynomial-time black-box identity testing and reconstruction algorithms for read-once formulas, which form a subclass of multilinear formulas. In [7], quasi-polynomial-time and polynomial-time PIT algorithms were given for multilinear read-\( k \) formulas in the black-box and the white-box settings, respectively for constant values of \( k \). At a high-level, both algorithms go by alternating the following two steps:
Step 1: Reduce PIT of a read-\((k + 1)\) formula to PIT of sum of two read-\(k\) formulas.

Step 2: Reduce PIT of sum of two read-\(k\) formulas to PIT of a (single) read-\(k\) formula.

While Step 2 introduces an overhead of (roughly) \(n^{kO(k)}\) in both settings, the gap in the final complexity results from the overhead introduced by Step 1. Indeed, in the whitebox setting, the overhead is \(\text{poly}(n, k)\) while in black-box setting the overhead is \(n^{O(\log n)}\). Moreover, for \(k = 0\) the the analysis of Step 2 can be seen as a different analysis of the black-box PIT algorithm for ROFs of [52], resulting in roughly the same run time. We hope that the ideas presented in this paper could be extended further to improve the analysis of the black-box PIT algorithms of [7], and, perhaps lead to new PIT algorithms.

Some open questions: can one obtain a polynomial-time black-box PIT algorithm for multilinear read-\(k\) formula with a constant \(k\)? What about \(k = 2\)? I.e. multilinear read-twice formulas. Even more specifically, can one show a black-box reduction from a PIT instance of a multilinear read-twice formula to polynomially-many PIT instances of sums of constantly-many read-once formulas, introducing only a polynomial overhead?

Acknowledgments. The authors would like to thank the anonymous referees for useful comments.

References


