A Polynomial Time Deterministic Algorithm for Identity Testing Read-Once Polynomials

Daniel Minahan [*] Ilya Volkovich [†]

August 15, 2016

Abstract

The polynomial identity testing problem, or PIT, asks how we can decide if a polynomial is equivalent to zero. A read-once polynomial (ROP) is a polynomial over some field that can be expressed with an arithmetic formula where each variable appears at most once. We construct a deterministic polynomial time algorithm for identity testing a black-box ROP. In particular, we show that for a read-once polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$, we can decide if $P \equiv 0$ by plugging in $n^{\mathcal{O}(1)}$ points to P. From this algorithm we can decide in polynomial time whether the sum of k ROPs is identically zero and we can re-construct a black-box *ROP* in polynomial time.

1 Introduction

Let $\mathbb{F}$ be a field. In general we would like to be able to efficiently decide if some $P \in \mathbb{F}[x_1, \ldots, x_n]$ is identically zero as a formal sum of monomials. Observe that some polynomials might not be the sum of monomials that are all identically zero but might always evaluate to zero. For example, $x^4 - x$ will always evaluate to zero on a field of size 5. For this reason we assume that we can access elements from a polynomially large extension field of $\mathbb{F}$.

Extensive research has already been done to answer this question in different contexts. It has already been shown that PIT can be solved in polynomial time up to an arbitrary degree of error given access to a random input generator [Zip79]. In particular, this demonstrates that the general PIT problem is in NP. This observation leads us to search for a deterministic algorithm for PIT that also achieves polynomial complexity. Although no such algorithm is known currently, work has already been done to find polynomial or quasi-polynomial algorithms for certain types of polynomials. In particular, we already have an algorithm for PIT on Read-Once Polynomials, or ROPs, that runs in time $n^{\mathcal{O}(\log n)}$ where n is the number of variables in the polynomial [SV15]. It is this algorithm that we use to develop our new algorithm.

In this paper, we construct an algorithm for PIT on ROPs that runs in time $n^{\mathcal{O}(1)}$. We first demonstrate that our algorithm works on homogenous ROPs,

[*]Departments of Mathematics and EECS, CSE Division, University of Michigan, Ann Arbor, MI. Email: `dminahan@umich.edu`. Research partially supported by NSF.

[†]Department of EECS, CSE Division, University of Michigan, Ann Arbor, MI. Email: `ilyavol@umich.edu`.

i.e. ROPs where every term has the same total degree. We use this fact to show that our algorithm works for general ROPs. We also discuss the application of this new algorithm in improving the complexity of other algorithms for related problems.

2 Preliminaries

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

Definition 1 (Arithmetic formula)**.** *A arithmetic formula is a binary tree where each leaf has a variable $x_i \in \{x_1, \dots, x_n\}$ and each internal node, called a gate, has 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 $\alpha(P_1 * P_2) + \beta$ where $*$ is the operation at the top gate.*

The efficiency of an algorithm over a set of formulas $\mathcal{C}$ is therefore based on the number of gates in $\mathcal{C}$. Note that this definition allows for some variables to be multiplied by 0 or canceled out. For this reason, we define $\text{var}(P)$.

Definition 2. *Pick $P \in \mathbb{F}[x_1, \dots, x_n]$. Define $\text{var}(P) \subseteq \{x_1, \dots, x_n\}$ the set of variables that P depends on, i.e. $y \in \text{var}(P)$ provided that $\exists \alpha, \beta \in \mathbb{F}$ such that $P|_{y=\alpha} \not\equiv P|_{y=\beta}$.*

Observe that this definition requires that $\mathbb{F}$ have enough elements. Additionally, we assume that our arithmetic formula is in a black box. This means that we cannot directly look at the circuit and are only allowed to send inputs and receive outputs. Therefore, any PIT algorithm will involve choosing a set of points and evaluating a polynomial on those points. Hereafter we assume that sending an input to a formula C and receiving an output takes $\mathcal{O}(1)$ time.

Definition 3 (Hitting Set)**.** *Let $\mathcal{C}$ be a circuit class of polynomials in $\mathbb{F}[x_1, \dots, x_n]$. A set S is called a hitting set for $\mathcal{C}$ provided that $\forall P \in \mathcal{C}$ with $P \not\equiv 0$ we have $P|_S \not\equiv 0$.*

This leads us to a basic algorithm for PIT.

Algorithm 1. *Let $\mathcal{C}$ be a circuit class of polynomial formulas in $\mathbb{F}[x_1, \dots, x_n]$. Let S be a hitting set for $\mathcal{C}$. Then we have an algorithm for PIT some $P \in \mathbb{P}$ that runs in time $\mathcal{O}(|S|)$.*

Proof. Observe that for some $P \in \mathcal{C}$, we have $P|_S \equiv 0 \iff P \equiv 0$ by the definition of hitting set. Thus we can test P on every element of S. If some $x \in S$ has $P(x) \neq 0$ then $P \not\equiv 0$. Otherwise we must have $P \equiv 0$. Since sending an input to a formula takes time $\mathcal{C}$, testing $|S|$ points will take $\mathcal{O}(|S|)$ □

One of the reasons that we want to achieve polynomial efficiency in our algorithm is that PIT is relatively straightforward if we allow any complexity. We could test a polynomial $P \in \mathbb{F}[x_1, \dots, x_n]$ on every $x \in \mathbb{F}^n$ given unlimited time. However, we can achieve exponential complexity in the general PIT problem using a slight generalization of the fundamental theorem of algebra.

Lemma 1 ([Alo99]). *Let $\mathbb{P} \subseteq \mathbb{F}[x_1, \ldots, x_n]$. Suppose the individual degree of any variable in $\mathbb{P}$ is bounded by some $d \in \mathbb{N}$. Pick $S \subseteq \mathbb{N}$ with $|S| > d$. Note that this means $|\mathbb{F}| > d$, so assume we can choose elements from an appropriately large extension field. Then we have that S^n is a hitting set for $\mathbb{P}$.*

Explicitly finding a hitting set for a circuit class $\mathcal{C}$ is quite difficult, so we develop our algorithm using a slightly different tool.

Definition 4 (Generator). *Let $\mathbb{P} \subseteq \mathbb{F}[x_1, \ldots, x_n]$. For some $t \in \mathbb{N}$, a polynomial map $G : \mathbb{F}^t \to \mathbb{F}^n$ is called a generator for $\mathbb{P}$ provided that $\forall P \in \mathbb{P}$ with $P \not\equiv 0$, we have $P(G) \not\equiv 0$.*

Intuitively, a generator G for $\mathbb{P}$ is a polynomial mapping that has a hitting set for $\mathbb{P}$ in its image. Finding a generator for a set of polynomials makes finding a hitting set for that set of polynomials easier by using **Lemma 1**, since a polynomial composed with a polynomial map is still a polynomial. A generator reduces the number of variables that $P \in \mathbb{F}[x_1, \ldots, x_n]$ depends on, but it may increase the degree of P. Thus, since **Lemma 2** produces a hitting set based on both the number of variables and the degree of the polynomial, we want to find a generator that reduces the number of variables that P depends on without drastically increasing its degree. Our algorithm will make use of a generator to produce a polynomial size hitting set.

2.1 Read-Once Polynomials

In order to simplify what is quite a complicated problem, we have to reduce the size of the set of polynomials that we study.

Definition 5 (Read-Once Formula). *A read-once formula is an arithmetic formula where each variable appears at most once. A read-once polynomial (ROP) is a polynomial defined by such a circuit.*

Note that the number of gates in a ROC 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 $n^{\mathcal{O}(1)}$. This definition also leads to a few nice properties of ROP.

Lemma 2 (Structural Lemma for ROPs). *Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a ROP. Then $\exists P_1, P_2$ (and non-constant if $n > 1$) variable disjoint ROPs and some $c \in \mathbb{F}$ such that either $P = P_1 + P_2$ or $P = P_1 \cdot P_2 + c$.*

Proof. This follows from the definition of a ROF. □

Note also that if P_1 and P_2 are variable disjoint ROPs, then $P_1 + P_2$ and $P_1 \cdot P_2$ are both ROPs.

Our analysis is actually analysis on a specific type of ROP. To that end we define a homogeneous polynomial.

Definition 6 (Homogeneous Polynomial). *$H \in \mathbb{F}[x_1, \ldots, x_n]$ is homogeneous provided that every monomial in H has the same total degree.*

For $i \in \mathbb{N}$ we also define $H_i(P)$ to be the monomials in P that have total degree i. We let $H_{max}(P)$ be the homogeneous part of P of degree equal to the degree of P.

2.2 Existing Algorithm

In this paper we improve the complexity analysis of a previous algorithm. This algorithm constructs a generator for a ROP in n variables. It also assumes that we have access to some extension field of $\mathbb{F}$ with $|\mathbb{F}| > n$. This is a necessary assumption, since it can be shown that polynomial complexity cannot be achieved over constant fields so we have to assume that this is extension field is available.

Definition 7 ([SV15])**.** *Pick $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{F}$. Pick some $t \in \mathbb{N}$. For each $i \in [n]$, let $p_i(y)$ be the Lagrange polynomial that has $p_i(y) = 1$ for $y = a_i$ and $p_i(y) = 0$ for $y = a_j$ with $j \neq i$. Note that this polynomial is just the product of $n-1$ linear factors $y - a_j$. Then for each $i \in [n]$, let $G_t^i(z_1, \ldots, z_t, y_1, \ldots, y_t) = \sum_{k=1}^{t} p_i(y_k) z_k$. Finally let $G_{n,t}(z_1, \ldots, y_t) = (G_t^1(z_1, \ldots, y_t), \ldots, G_t^n(z_1, \ldots, y_t))$.*

The intuition behind this algorithm is that everytime we double the number of variables in a polynomial P, we have to add one additional variable in the generator $G_{n,t}$. The analysis presented here does not actually make use of this intuition explicitly, but does utilize the fact that each variable is uniquely identified by some $a_i \in A$. It has already been shown that $G_{n,\log n}$ is a generator for the set of ROPs in n variable. Observe also that if $G_{n,t}$ is a generator for the set of ROPs for some $t \in \mathbb{N}$, then using **Lemma 1** we can construct a hitting set for $P(G_{n,t})$ of size $n^{\mathcal{O}(t)}$. We see the maximal degree of $P(G_{n,t})$ is $n(n-1) = n^{\mathcal{O}(1)}$. This means we need a hitting set of size $(n^{\mathcal{O}(1)})^{\mathcal{O}(t)} = n^{\mathcal{O}(t)}$, which is therefore the runtime of our algorithm. In this paper we demonstrate that $G_{n,1}$ is a generator for the set of ROPs, so in fact we have an algorithm that runs in time $n^{\mathcal{O}(1)}$.

3 Main Result

For any $P \in \mathbb{F}[x_1, \ldots, x_n]$, define $H_i(P)$ as the homogeneous part of degree $i \in \mathbb{N}$ for P. We want to show that $G_{n,1}$ is a generator for the set of ROPs. We show first that for any non-constant ROP $P \in \mathbb{F}[x_1, \ldots, x_n]$, of degree $d \in \mathbb{N}$ we have $H_d(P)$ a ROP. We then show that $G_{n,1}$ is a generator for the set of homogeneous ROPs. We can then combine these two results to show that $G_{n,1}$ is a generator for the set of ROPs.

The first thing we need to do is describe some properties of homogenous ROPs. In particular, we show that the ROP structural lemma holds for HROPs.

Lemma 3 (Structural Lemma for HROPs)**.** *Pick $P \in \mathbb{F}[x_1, \ldots, x_n]$ a non-constant HROP of degree $d \in \mathbb{N}$ with $|\mathrm{var}(P)| > 1$. Then $\exists P_1, P_2 \in \mathbb{F}[x_1, \ldots, x_n]$ non-constant variable disjoint HROPs such that one of the following is true:*

- $P = P_1 + P_2$
- $P = P_1 \cdot P_2$

Proof. In the multiplicative case, since P is a ROP, it follows immediately from the ROP Structural Lemma that we can choose $P_1, P_2 \in \mathbb{F}[x_1, \ldots, x_n]$ ROPs such that we have our result up to some added constant. Assume by way of contradiction that WOLOG P_1 is non-homogeneous. Then observe that the highest total degree term of P_1 multiplied by the highest total degree term of

P_2 will have a different total degree than the lowest total degree terms multiplied together. In the additive case, we can again choose P_1 and P_2 ROPs by the ROP Structural Lemma. Since P_1 and P_2 are variable disjoint, no terms can cancel out. Therefore if WOLOG P_1 is non-homogeneous, terms with differing total degree will remain and P will not be homogeneous. Thus P_1 and P_2 must be homogeneous. Observe that this means we have no non-zero constant term added in the multiplicative case. $\square$

Lemma 4. *Pick a non-constant ROP $P \in \mathbb{F}[x_1, \dots, x_n]$. Then $H_{max}(P)$ is a non-constant ROP.*

Proof. For the base case when $n = 1$, we just have a linear term $\alpha x + \beta$ so we have αx is a ROP that forms the highest total degree terms. Inductively, suppose our result is true for every ROP with $|\text{var}(P)| \leq n$. Pick P a ROP with $|\text{var}(P)| = n + 1$. We have two cases.

1. $P = P_1 + P_2$ where both are non-constant. Observe that the maximum degree terms in P will appear in either P_1 or P_2, so $\deg(P) = \max\{\deg(P_1), \deg(P_2)\}$. Since P_1 and P_2 are variable disjoint and non-constant, each depends on strictly fewer than $n + 1$ variables. Thus by the inductive hypothesis $H_{max}(P_1)$ and $H_{max}(P_2)$ are ROPs. Therefore $H_{max}(P)$ is either a ROP or the sum of variable disjoint ROPs in which case it is also an ROP.

2. $P = P_1 \cdot P_2 + c$. Observe that the maximum total degree terms in P are simply the product of all the maximal total degree terms in P_1 and P_2, so $H_{max}(P) = H_{max}(P_1) \cdot H_{max}(P_2)$. Note that $|\text{var}(P_1)|, |\text{var}(P_2)| \leq n$ so by the inductive hypothesis $H_{max}(P_1)$ and $H_{max}(P_2)$ are ROPs. Since $H_{max}(P)$ is a product of variable disjoint ROPs, it is also an ROP.

Thus we have $H_{max}(P)$ is an ROP. $\square$

We first show that $P \in \mathbb{F}[x_1, \dots, x_n]$ a non-constant homogeneous ROP implies $P(G_{n,1})$ non-constant. We begin by demonstrating this result for $\deg(P) = 1$.

Lemma 5. *If $P \in \mathbb{F}[x_1, \dots, x_n]$ is a non-constant ROP of degree one then $P(G_{n,1})$ is non-constant.*

Proof. This proof is available in [SV15] but we reproduce it here as well. Pick some $P \in \mathbb{F}[x_1, \dots, x_n]$ of degree 1. Observe that P is an HROP and is non-constant. Pick $x_i \in \text{var}(P)$. Then observe that $P(G_{n,1})|_{y_1=a_i}$ sets every monomial in P to 0 except the monomial that contains x_i, so $P(G_{n,1})|_{y_1=a_i} = \alpha z_1$ for some $\alpha \in \mathbb{F} \setminus \{0\}$. This is non-constant so $P(G_{n,1})$ is non-constant. $\square$

We now prove the correctness of our algorithm for HROPs.

Lemma 6. *If $P \in \mathbb{F}[x_1, \dots, x_n]$ a non-constant HROP with $\deg(P) = d$, then $P(G_{n,1})$ non-constant.*

Proof. We want to show $\forall P \in \mathbb{F}[x_1, \dots, x_n]$ with P a HROP, we have $P(G_{n,1})$ non-constant $\iff$ P is non-constant. Note that we drop the indices in y and z when writing out $G_{n,1}$. $P(G_{n,1})$ non-constant immediately implies P

non-constant, so it remains to show the other direction. We induct on $n = |\mathrm{var}(P)|$. The base case where $n = 1$ follows from **Lemma 5**. Suppose our result holds $\forall i \leq n \in \mathbb{N}$. Pick a non-constant HROP $P \in \mathbb{F}[x_1, \ldots, x_{n+1}]$ of degree $d \in \mathbb{N}$ with $d > 1$ since the case where $d = 1$ follows from **Lemma 5**. If we have $|\mathrm{var}(P)| \leq n$ we are done by the inductive hypothesis, so assume we have $|\mathrm{var}(P)| = n + 1$. By the HROP Structural Lemma we have two cases.

1. $P = P_1 \cdot P_2$. Note that $|\mathrm{var}(P_1)|, |\mathrm{var}(P_2)| \leq n$, so by the inductive hypothesis $P_1(G_{n,1})$ and $P_2(G_{n,1})$ are non-constant, so their product is non-constant.

2. $P = P_1 + P_2$. Note that we have $d = \deg(P_1) = \deg(P_2)$. For notational convenience assume that $\mathrm{var}(P_1) \subseteq \{x_1, \ldots, x_b\}$ and $\mathrm{var}(P_2) \subseteq \{x_{b+1}, \ldots, x_{n+1}\}$. Let $U_1(y) = \prod_{i=b+1}^{n+1}(y - a_i)$, $U_2(y) = \prod_{j=1}^{b}(y - a_j)$ and $\Phi(y) = (y - a_1) \cdot \ldots \cdot (y - a_{n+1})$. Then to each P_i we have
$$P_i(G_{n,1}(z, y)) = z^d \Phi(y)^{d-1} \cdot U_i(y) P_i'(y)$$
for some polynomial $P_i' \in \mathbb{F}[y]$. As an example, if $P_1 = (x_1 + x_2 \cdot) \ldots \cdot x_b$ then we have
$$P_1 = z^{b-1} \Phi(y)^{b-2} (x - a_{b+1}) \cdot \ldots \cdot (x - a_{n+1})((y - a_1) + (y - a_2)).$$
Since $d > 1$ we have that $\Phi(y)^{d-1}$ is non-constant. Observe that P_i' is non-zero by the inductive hypothesis since $\mathrm{var}(P_i) \leq n$ for $i = 1$ and $i = 2$. We see that $\deg_y(P_1(G_{n,1})) = nd$, so $\deg(P_1'(y)U_1(y)) = \deg(P_1'(y)) + \deg(U_1(y)) = nd - (n + 1)(d - 1) = n - d + 1$. Note also that $\deg(U_1) + \deg(U_2) = n + 1$. Thus
$$P = \phi(y)^{d-1} z^d (P_1'(y)U_1(y) + P_2'(y)U_2(y))$$
Note that $\phi(y)^{d-1} z^d$ is non-constant, so it is sufficient to show that $P_1'(y)U_1(y) \neq -P_2'U_2(y)$. Suppose by way of contradiction we have $P_1'(y) \cdot U_1(y) = -P_2'(y) \cdot U_2(y)$. Note that $U_1(y)$ and $U_2(y)$ have no common factors, so we must have $U_2(y)|P_1'(y)$. This means that $\deg(P_1'(y)) \geq \deg(U_2(y))$, so $\deg(P_1'(y)) + \deg(U_1(y)) \geq n+1 > n-d+1$ which is a contradiction. Thus the rightmost term of P is not identically 0 and since Φ^{d-1} is non-constant we have P non-constant.

Thus we have our result. ☐

We now combine these results to show a more general result for ROPs.

Theorem 1. *$\forall P \in \mathbb{F}[x_1, \ldots, x_n]$ with P an ROP, we have $P(G_{n,1}) \not\equiv 0 \iff P \not\equiv 0$.*

Proof. Pick $P \in \mathbb{F}[x_1, \ldots, x_n]$. If $P(G_{n,1}) \not\equiv 0$ then $P \not\equiv 0$. To show the other direction, first we see that P a non-zero constant implies that $P(G_{n,1})$ is also a non-zero constant. Otherwise suppose that P is non-constant and that $\deg(P) = d$ for some $d \in \mathbb{N}$. Write $P = \sum_{i=0}^{d} H_d(P)$. Now note $\forall i \leq d$ we have $\deg_z(P_i(G_{n,1})) = i$. This means that the only way to have $P(G_{n,1}) \equiv 0$ is if $P_i(G_{n,1}) \equiv 0\ \forall i \leq d$. Recall from **Lemma 4** that $H_{max}(P)$ is an HROP. Then from **Lemma 6** we have $P_d(G_{n,1})$ non-constant, so $P(G_{n,1})$ non-constant and thus $P(G_{n,1}) \not\equiv 0$. ☐

Now that we have a generator for ROP we can use existing results to explicitly answer PIT for ROPs in polynomial time.

Algorithm 2. *We have an algorithm for ROP Identity Testing that runs in time $n^{\mathcal{O}(1)}$.*

Proof. Pick a ROP $P \in \mathbb{F}[x_1, \ldots, x_n]$. Recall from **Theorem 1** that if $P \not\equiv 0$ then $P(G_{n,1}) \not\equiv 0$. Now note that $P(G_{n,1})$ is a polynomial in 2 variables with maximal degree $n(n-1) = \mathcal{O}(n^2)$. Thus from **Lemma 1** we have a hitting size of size $\mathcal{O}(n^2)^2$ for $P(G_{n,1})$ so we have a PIT algorithm that runs in time $n^{\mathcal{O}(1)}$. □

4 Applications

There are many existing algorithms that make use of a PIT algorithm to solve other problems. In particular, these algorithms query a PIT algorithm a polynomial number of times, so the complexity of the algorithm is limited only by the complexity of the PIT algorithm. Our new result allows us to show that other problems besides PIT can be solved more rapidly.

One natural extension of this problem is to ask whether a sum of k ROPs is identically 0 or not. There are existing algorithms to decide whether the sum of k ROPs is identically 0 that make use of a PIT algorithm.

Lemma 7 (Follows from [SV15]). *Let $\mathcal{G}$ be a generator for ROPs in n variables. Then $\mathcal{G} + G_{n,3k}$ is a generator for a sum of k ROPs.*

This result leads us to an algorithm for PIT on a sum of k ROPs.

Algorithm 3. *From **Lemma 7** and **Theorem 1** we have that $G_{n,3k+1}$ is a generator for a sum of k ROPs. From **Lemma 1** we have an algorithm for PIT of the sum of k ROPs that runs in time $n^{\mathcal{O}(k)}$.*

Observe for a fixed $k \in \mathbb{N}$ that this algorithm runs in polynomial time with respect to n.

Another natural extension of the PIT problem is to ask what other information we can determine about a polynomial besides whether it is identically zero. In particular we can ask if given a formula F in a black box we can exactly produce the polynomial that F calculates. There already exist efficient algorithms to accomplish this task that are bounded in complexity by the run time of a PIT algorithm. Since we now have a more efficient PIT, we can efficiently solve reconstruction.

Lemma 8 (Follows from [SV15]). *Suppose $|\mathbb{F}| > n^2$. Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be a ROP. Suppose the class of ROPs admits a deterministic PIT algorithm A. Then there exists a deterministic algorithm B that has black-box access to A and computes a justifying assignment $\bar{a}$ for P in time $O(n^4 t)$ where t is the running time of A.*

Lemma 9 (Follows from [SV14]). *Suppose we have a ROP formula P in a black box. Then we can reconstruct P in time $n^{\mathcal{O}(1)}$.*

References

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

[SV14] A. Shpilka and I. Volkovich. On reconstruction and testing of read-once formulas. *Theory of Computing*, 18(10):465–514, 2014.

[SV15] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 3(24):477 – 532, 2015.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials, 1979.