On the Relation between Polynomial Identity Testing and Finding Variable Disjoint Factors

Amir Shpilka and Ilya Volkovich

Faculty of Computer Science, Technion, Haifa 32000, Israel
{shpilka,ilyav}@cs.technion.ac.il.

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

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

An immediate corollary is that polynomial identity testing and polynomial factorization are equivalent (up to a polynomial overhead) for multilinear polynomials. In addition, we observe that derandomizing the polynomial decomposition problem is equivalent, in the sense of Kabanets and Impagliazzo [1], to proving arithmetic circuit lower bounds for NEXP.

Our approach uses ideas from [2], that showed that the polynomial identity testing problem for a circuit class $\mathcal{C}$ is essentially equivalent to the problem of deciding whether a circuit from $\mathcal{C}$ computes a polynomial that has a read-once arithmetic formula.

1 Introduction

In this paper we study the relation between two fundamental algebraic problems, polynomial identity testing and polynomial factorization. We show that the tasks of giving deterministic algorithms for polynomial identity testing and for a variant of the factorization problem (that we refer to as the polynomial decomposition problem) are essentially equivalent. We first give some background on both problems and then discuss our results in detail.

Polynomial Decomposition. Let $X = (x_1, \ldots, x_n)$ be the set of variables. For a set $I \subseteq [n]$ denote with X_I the set of variables whose indices belong to I. A polynomial f, depending on X, is said to be *decomposable* if it can be written

S. Abramsky et al. (Eds.): ICALP 2010, Part I, LNCS 6198, pp. 408–419, 2010.

as $f(X) = g(X_S) \cdot h(X_{[n]\setminus S})$ for some $\emptyset \subsetneq S \subsetneq [n]$. The *indecomposable factors* of a polynomial $f(X)$ are polynomials $f_1(X_{I_1}), \ldots, f_k(X_{I_k})$ such that the I_j-s are disjoint sets of indices, $f(X) = f_1(X_{I_1}) \cdot f_2(X_{I_2}) \cdots f_k(X_{I_k})$ and the f_i's are indecomposable. It is not difficult to see that every polynomial has a unique factorization to indecomposable factors (up to multiplication by field elements). The problem of polynomial decomposition is defined in the following way: Given an arithmetic circuit from an arithmetic circuit class $\mathcal{C}$ computing a polynomial f, we have to output circuits for each of the indecomposable factors of f. If we only have a black-box access to f then we have to output a black-box (i.e. an algorithm that may use the original black-box) for each of the indecomposable factors of f. Clearly, finding the indecomposable factors of a polynomial f is an easier task than finding all the irreducible factors of f. It is not hard to see though, that for the natural class of multilinear polynomials the two problems are the same. We also consider the decision version of the polynomial decomposition problem: Given an arithmetic circuit computing a multivariate polynomial decide whether the polynomial is decomposable or not. Note that in the decision version the algorithm just has to answer 'yes' or 'no' and is not required to find the decomposition.

Many randomized algorithms are known for factoring multivariate polynomials in the black-box and non black-box models (see the surveys in [3,4,5]). These algorithms also solve the decomposition problem. However, it is a long standing open question whether there is an efficient *deterministic* algorithm for factoring multivariate polynomials (see [3, 6]). Moreover, there is no known deterministic algorithm even for the decision version of the problem (that is defined analogously). Furthermore, even for the simpler case of factoring multilinear polynomials (which is a subproblem of polynomial decomposition) no deterministic algorithms are known.

Polynomial Identity Testing. Let $\mathcal{C}$ be a class of arithmetic circuits defined over some field $\mathbb{F}$. The polynomial identity testing problem (PIT for short) for $\mathcal{C}$ is the question of deciding whether a given circuit form $\mathcal{C}$ computes the identically zero polynomial. This question can be considered both in the black-box model, in which we can only access the polynomial computed by the circuit using queries, or in the non black-box model where the circuit is given to us. The importance of this fundamental problem stems from its many applications. For example, the deterministic primality testing algorithm of [7] and the fast parallel algorithm for perfect matching of [8] are based on solving PIT problems.

PIT has a well known randomized algorithm [9,10,11]. However, we are interested in the problem of obtaining efficient *deterministic* algorithms for it. This question received a lot of attention recently [12, 13, 14, 1, 15, 16, 17, 18, 19, 20, 21, 2, 22, 23, 24, 25, 26, 27, 28] but its deterministic complexity is still far from being well understood. In [29, 1, 15, 22] results connecting PIT to lower bounds for arithmetic circuits where proved, shedding light on the difficulty of the problem. It is interesting to note that the PIT problem becomes very difficult already for depth-4 circuits. Indeed, [23] proved that a polynomial time black-box PIT algorithm for depth-4 circuits implies an exponential lower bound for general arithmetic circuits (and hence using the ideas of [1] a quasi-polynomial time deterministic PIT algorithm for general circuits).

In this work we (essentially) show equivalence between the PIT and polynomial decomposition problems. Namely, we prove that for any (reasonable) circuit class $\mathcal{C}$, it holds that $\mathcal{C}$ has a polynomial time deterministic PIT algorithm if and only if it has a polynomial time deterministic decomposition algorithm. The result holds both in the black-box and the non black-box models. That is, if the PIT for $\mathcal{C}$ is in the black-box model then deterministic black-box decomposition is possible and vice versa, and similarly for the non black-box case. Before giving the formal statement of our results we give some definitions.

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

We shall say that a polynomial $f(x_1, \ldots, x_n)$ has *individual degrees* bounded by d, if no variable has degree higher than d in f. An arithmetic circuit C has *individual degrees* bounded by d if the polynomial that C computes has individual degrees bounded by d. Finally, we shall say that C is an (n, s, d)-*circuit* if it is an n-variate arithmetic circuit of size s with individual degrees bounded by d. Sometimes we shall think of an arithmetic circuit and of the polynomial that it computes as the same objects.

In this paper we will usually refer to a model of arithmetic circuits $\mathcal{C}$. It should be thought of as either the general model of arithmetic circuits or as some restricted model such as bounded depth circuits, etc.

Our Results. We now formally state our results. We give them in a very general form as we later apply them to very restricted classes such as depth-3 circuits, read-once formulas etc.

Theorem 1 (Main). *Let $\mathcal{C}$ be a class of arithmetic circuits, defined over a field $\mathbb{F}$. Consider circuits of the form $C = C_1 + C_2 \times C_3$, where the C_i-s are (n, s, d)-circuits from $\mathcal{C}$ and, C_2 and C_3 are defined over disjoint sets of variables.*[1] *Assume that there is a deterministic algorithm that when given access (explicit or via a black-box) to such a circuit C runs in time $T(s, d)$ and decides whether $C \equiv 0$. Then, there is a deterministic algorithm that when given*

[1] This requirement seems a bit strange but we need it in order to state our results in the most general terms.

access (explicit or via a black-box) to an (n, s, d)*-circuit* $C' \in \mathcal{C}$*,*[2] *runs in time* $\mathcal{O}(n^3 \cdot d \cdot T(s, d))$ *and outputs the indecomposable factors,* $H = \{h_1, \ldots, h_k\}$*, of the polynomial computed by* C'*. Moreover, each* h_i *is in* $\mathcal{C}$ *and* $size(h_i) \leq s$*.*

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

Observation 1. *Let* $\mathcal{C}$ *be a class of arithmetic circuits. Assume that there is an algorithm that when given access (explicit or via a black-box) to an* (n, s, d)*-circuit* $C \in \mathcal{C}$ *runs in time* $T(s, d)$ *and outputs "true" iff the polynomial computed by* C *is decomposable. Then, there is a deterministic algorithm that runs in time* $\mathcal{O}(T(s + 2, d))$ *and solves the PIT problem for size* s *circuits from* $\mathcal{C}$*.*

As mentioned above, the irreducible factors of multilinear polynomials are simply their indecomposable factors. Hence we obtain the following corollary. We give here a slightly informal statement. The full statement is given in Section 3.1.

Corollary 1 (informal). *Let* $\mathcal{C}$ *be an arithmetic circuit class computing multilinear polynomials. Then, up to a polynomial overhead, the deterministic polynomial identity testing problem and the deterministic factorization problem for circuits from* $\mathcal{C}$ *are equivalent, in both the black-box the and non black-box models.*

We also obtain some extensions to the results above. The first result shows how to get a non-adaptive decomposition from a PIT algorithm (Theorem 1 gives an adaptive algorithm). To prove it we need a stronger PIT algorithm than the one used in the proof of Theorem 1. The second extension gives an algorithm deciding whether for a given polynomial f there are two variables x_i, x_j such that $f(X) = f_1(X_{[n]\setminus\{i\}}) \cdot f_2(X_{[n]\setminus\{j\}})$. This can be thought of as a generalization of Theorem 1. Finally, we obtain a connection between the decomposition problem and lower bounds in the sense of Kabanets and Impagliazzo. Due to space limitations the proofs of those results are omitted from this version.

Motivation. The motivation for this work is twofold. First, the most obvious motivation is that we think that the problem of connecting the complexity of PIT and polynomial factorization is very natural. Another motivation is to better understand the PIT problem for multilinear formulas.[3] Although lower bounds are known for multilinear formulas [30, 31, 32, 33], we do not have an efficient PIT algorithm even for depth-3 multilinear formulas. Consider the following approach towards PIT of multilinear formulas. Start by making the formula a read-once formula. I.e. a formula in which every variable labels at most one leaf. This can be done by replacing, for each i and j, the j-th occurrence of x_i with a new variable $x_{i,j}$. Now, using PIT algorithm for read-once formulas [2, 26], check whether this formula is zero or not. If it is zero then the original formulas was also zero and we are done. Otherwise start replacing back each $x_{i,j}$ with x_i. After each replacement we would like to verify

[2] $C' \in \mathcal{C}$ denotes that the circuit C' is from $\mathcal{C}$.

[3] A multilinear formula is a formula in which every gate computes a multilinear polynomial, see [30].

that the resulting formula is still not zero. Notice that when replacing $x_{i,j}$ with x_i we get zero if and only if the linear function $x_i - x_{i,j}$ is a factor of the formula at hand. Thus, we somehow have to find a way of verifying whether a linear function is a factor of a multilinear formula. Notice that as we start with a read-once formula for which PIT is known [2, 26], we can assume that we know many inputs on which the formula does not vanish. One may hope that before replacing $x_{i,j}$ with x_i we somehow managed to obtain inputs that will enable us to verify whether $x_i - x_{i,j}$ is a factor of the formula or not. This of course is not formal and only gives a sketch of an idea, but it shows the importance of understanding how to factor multilinear formulas given a PIT algorithm. As different factors of multilinear formulas are variable disjoint this motivates the study of polynomial decomposition.

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

The more interesting direction is obtaining a decomposition algorithm given a PIT algorithm. Note that if $f(X) = f_1(X_{I_1}) \cdot \ldots \cdot f_k(X_{I_k})$ is the decomposition of f and if we know the sets $I_1, \ldots, I_k$ then using the PIT algorithm we can easily obtain circuits for the different f_i's. Indeed, if $\bar{a} \in \mathbb{F}^n$ is such that $f(\bar{a}) \neq 0$ then, for some constant α_j, $f_j(X_{I_j}) = \alpha_j \cdot f|_{\bar{a}_{[n]\setminus I_j}}(X)$, where $\bar{a}_{[n]\setminus I_j}$ is the assignment that assigns values to all the variables except those whose index belongs to I_j.[4] Now, given a PIT algorithm we can use it to obtain such $\bar{a}$ in a manner similar to finding a satisfying assignment to a CNF formula given a SAT oracle. Consequently, finding the partition $I_1, \ldots, I_k$ of $[n]$ is equivalent to finding the indecomposable factors (assuming that we have a PIT algorithm).

We present two approaches for finding the partition. The first is by induction: Using the PIT algorithm we obtain an assignment $\bar{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$ that has the property that for every $j \in [n]$ it holds that f depends on x_j if and only if $f|_{\bar{a}_{[n]\setminus\{j\}}}$ depends on x_j. Following [34, 35, 2, 26] we call $\bar{a}$ a *justifying* assignment of f. Given a justifying assignment $\bar{a}$, we find, by induction, the indecomposable factors of $f|_{x_n=a_n}$. Then, using simple algebra we recover the indecomposable factors of f from those of $f|_{x_n=a_n}$. This is the idea behind the proof of Theorem 1.

In the second approach, we observe that the variables x_i and x_j belong to the same set I in the partition iff $\Delta_{ij} f \stackrel{\Delta}{=} f \cdot f|_{x_i=y, x_j=w} - f|_{x_i=y} \cdot f|_{x_j=w} \not\equiv 0$, when y and w are two new variables. Using this observation we obtain the partition by constructing a graph G on the set of vertices $[n]$ in which i and j are neighbors if and only if $\Delta_{ij} f \not\equiv 0$. The sets of the partition are exactly the connected components of G. Due to space limitations we present only the first approach.

[4] In fact, we also need a constant α, that is easily computable, to get that $f(X) = \alpha \cdot f_1(X_{I_1}) \cdot \ldots \cdot f_k(X_{I_k})$.

Related Works. The only work that we are aware of that studies the relation between PIT and polynomial factorization is [36]. There, Kaltofen and Koiran gave polynomial time deterministic identity testing algorithm for a class of super-sparse univariate polynomials, and from there obtained a deterministic factorization algorithm for a related class of bivariate polynomials. This model is, of course, very different from the models studied here.

As mentioned above, justifying assignments were first defined and used in [34,35] for the purpose of giving randomized polynomial time learning algorithms for read-once arithmetic formulas. In [2, 26] justifying assignments were used in conjunction with new PIT algorithms in order to obtain deterministic quasi-polynomial time interpolation algorithms for read-once formulas. We rely on the ideas from [26] for obtaining justifying assignments from PIT algorithms.

Another line of works that is related to our results is that of Kabanets and Impagliazzo [1] and of [22]. There it was shown that the question of derandomizing PIT is closely related to the problem of proving lower bounds for arithmetic circuits. These results use the fact that factors of small arithmetic circuits can also be computed by small arithmetic circuits. This gives another connection between PIT and polynomial factorization, although a less direct one.

The results of [1] relate PIT to arithmetic lower bounds for NEXP. However, these lower bounds are not strong enough and do not imply that derandomization of PIT gives derandomization of other algebraic problems. Similarly, the results of [15] show that polynomial time *black-box* PIT algorithms give rise to exponential lower bounds for arithmetic circuits which in turn, using ideas a-la [1], may give *quasi-polynomial* time derandomization of polynomial factorization.[5] However, this still does not guarantee polynomial time derandomization as is achieved in this work.

Organization. In Section 2 we give the required definition and discuss partial derivatives and justifying assignments. In Section 3 we prove our main result and derive some corollaries.

2 Preliminaries

For an integer n denote $[n] = \{1, \ldots, n\}$. In this paper all the circuits and polynomials are defined over some field $\mathbb{F}$. In most of our algorithms we will need to assume that $\mathbb{F}$ is larger than some function depending on n (we will mostly have the requirement $|\mathbb{F}| > nd$, where n is the number of variables and d is an upper bound on the individual degrees of the given circuit/polynomial). We note that this is not a real issue as in most works on factoring or on PIT it is assumed that we can access a polynomially large extension field of $\mathbb{F}$. From now on we assume that $\mathbb{F}$ is sufficiently large.

For a polynomial $f(x_1, \ldots, x_n)$, a variable x_i and a field element α we denote with $f|_{x_i=\alpha}$ the polynomial resulting from substituting α to x_i.

[5] We note that, currently, it is not clear how to derandomize the factorization problem using lower bounds for arithmetic circuits.

Similarly, given a subset $I \subseteq [n]$ and an assignment $\bar{a} \in \mathbb{F}^n$ we define $f|_{x_I=\bar{a}_I}$ to be the polynomial resulting from substituting a_i to x_i for every $i \in I$. We say that f *depends* on x_i if there exist $\bar{a} \in \mathbb{F}^n$ and $b \in \mathbb{F}$ such that: $f(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n) \neq f(a_1, a_2, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n)$. We denote $\mathrm{var}(f) \triangleq \{i \in [n] \mid f \text{ depends on } x_i\}$. It is not difficult to see that f depends on x_i if and only if x_i appears when f is written as a sum of monomials. By substituting a value to a variable of f we obviously eliminate the dependence of f on this variable. However, this can also eliminate the dependence of f on other variables, so we may lose more 'information' than intended. To handle this problem we define a 'lossless' type of an assignment. Similar definitions were given in [34, 35, 2, 26]. For completeness we repeat the definitions here.

Definition 1 (Justifying assignment). *Given an assignment $\bar{a} \in \mathbb{F}^n$ we say that $\bar{a}$ is a* justifying assignment *of f if for every subset $I \subseteq \mathrm{var}(f)$ we have that* $\mathrm{var}(f|_{x_I=\bar{a}_I}) = \mathrm{var}(f) \setminus I$.

Proposition 1 ([2]). *An assignment $\bar{a} \in \mathbb{F}^n$ is a justifying assignment of f if and only if* $\mathrm{var}(f|_{x_I=\bar{a}_I}) = \mathrm{var}(f) \setminus I$ *for every subset I of size $|I| = |\mathrm{var}(f)| - 1$.*

We now show how to get a justifying assignment from a polynomial identity testing algorithm. This was first done in [2] (and generalized in [26]). Before stating the result we shall need the following definition.

Definition 2 (Partial Derivative). *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial. The* partial derivative *of f w.r.t. x_i and direction $\alpha \in \mathbb{F}$ is defined as $\frac{\partial f}{\partial_\alpha x_i} \triangleq f|_{x_i=\alpha} - f|_{x_i=0}$. For an arithmetic circuit C we define $\frac{\partial C}{\partial_\alpha x_i} \triangleq C|_{x_i=\alpha} - C|_{x_i=0}$.*

Our algorithm will consider a circuit class $\mathcal{C}$ but will require a PIT algorithm for a slightly larger class. Namely, for every circuit $C \in \mathcal{C}$ we will need a PIT algorithm for circuits of the form $\frac{\partial C}{\partial_\alpha x_i}$. To ease the reading we shall refer to all circuits of the form $\frac{\partial C}{\partial_\alpha x_i}$ as ∂C.

Theorem 2 ([2, 26]). *Let $\mathbb{F}$ be a field of size $|\mathbb{F}| \geq nd$. Let f be a polynomial that is computed by an (n, s, d)-circuit $C \in \mathcal{C}$. Then, there is an algorithm that returns a justifying assignment $\bar{a}$ for f in time $\mathcal{O}(n^3 d \cdot T(s, d))$, where $T(s, d)$ is the running time of the PIT algorithm for circuits of the form ∂C where $C \in \mathcal{C}$ is an (n, s, d)-circuit.*

2.1 Indecomposable Polynomials

Definition 3. *We say that a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ is* indecomposable *if it is non-constant and cannot be represented as the product of two (or more) non-constant variable disjoint polynomials. Otherwise, we say that f is decomposable.*

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

Lemma 1 (Unique decomposition). *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-constant polynomial. Then f has a unique (up to multiplication by field elements) factorization to indecomposable factors.*

Observation 2. *Let f be a multilinear polynomial. Then f is indecomposable if and only if f is irreducible. In particular, if $f(\bar{x}) = f_1(\bar{x}) \cdot f_2(\bar{x}) \cdot \ldots \cdot f_k(\bar{x})$ is the decomposition of f, then the f_i-s are f's irreducible factors.*

3 Decomposition

In this section we give the proof of Theorem 1. Algorithm 1 shows how to find the indecomposable factors for a polynomial computed by $\mathcal{C}$ using the PIT algorithm. In fact, the algorithm returns a partition $\mathcal{I} = \{I_1, \ldots, I_k\}$ of $[n]$ such that the decomposition of f is $f = h_1(X_{I_1}) \cdot \ldots \cdot h_k(X_{I_k})$, for some polynomials $h_1, \ldots, h_k$. We call $\mathcal{I}$ the *variable-partition* of f. The idea behind the algorithm is to first find a justifying assignment $\bar{a}$ to f using Theorem 2. Then, to find the partition of $f|_{x_n=a_n}$. Finally, by using the PIT algorithm, to decide which sets in the partition of $f|_{x_n=a_n}$ belong to the partition of f and which sets must be unified.

Algorithm 1. Finding variable partition

Input: An (n, s, d)-circuit C from a circuit class $\mathcal{C}$, a justifying assignment $\bar{a}$ for C, and access to a PIT algorithm as in the statement of Theorem 1.
Output: A variable-partition $\mathcal{I}$

1: Set $\mathcal{I} = \emptyset$, $J = [n]$ ($\mathcal{I}$ will be the partition that we seek).
2: Set $x_n = a_n$ and recursively compute the variable-partition of $C' = C|_{x_n=a_n}$. Let $\mathcal{I}'$ be the resulting partition (note that when $n = 1$ then we just return $\mathcal{I} = \{\{1\}\}$).
3: For every set $I \in \mathcal{I}'$ check whether $C(\bar{a}) \cdot C \equiv C|_{x_I=\bar{a}_I} \cdot C|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}}$. If this is the case then add I to $\mathcal{I}$ and set $J \leftarrow J \setminus I$. Otherwise, move to the next I.
4: Finally, add the remaining elements to $\mathcal{I}$. Namely, $\mathcal{I} \leftarrow \mathcal{I} \cup \{J\}$.

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

Lemma 2. *Let C be an (n, s, d)-circuit from $\mathcal{C}$ such that $\mathrm{var}(C) = [n]$. Assume there exists a PIT algorithm as in the statement of Theorem 1. Let $\bar{a}$ be a justifying assignment of C. Then given C and $\bar{a}$ Algorithm 1 outputs a variable-partition $\mathcal{I}$ for the polynomial computed by C. The running time of the algorithm is $\mathcal{O}(n^2 \cdot T(s, d))$, where $T(s, d)$ is as in the statement of Theorem 1.*

Proof. The proof of correctness is by induction on n. For the base case ($n = 1$) we recall that a univariate polynomial is an indecomposable polynomial. Now assume that $n > 1$. Let $C = h_1(X_{I_1}) \cdot \ldots \cdot h_{k-1}(X_{I_{k-1}}) \cdot h_k(X_{I_k})$ be the decomposition of C where $\mathcal{I} = \{I_1, \ldots, I_k\}$ is its variable-partition. Assume w.l.o.g. that $n \in I_k$. Consider $C' = C|_{x_n=a_n}$. It holds that $C' = C|_{x_n=a_n} = h_1 \cdot \ldots \cdot h_{k-1} \cdot h_k|_{x_n=a_n} = h_1 \cdot \ldots \cdot h_{k-1} \cdot g_1 \cdot g_2 \cdot \ldots \cdot g_\ell$ where the g_i-s are the indecomposable factors of $h_k|_{x_n=a_n}$. Denote with $\mathcal{I}_k = \{I_{k,1},\ldots,I_{k,\ell}\}$ the variable-partition of $h_k|_{x_n=a_n}$. As $\bar{a}$ is a justifying assignment of C we obtain that $\mathrm{var}(C') = [n-1]$. From the uniqueness of the decomposition (Lemma 1) and by the induction hypothesis we get that, when running on C', the algorithm returns $\mathcal{I}' = \{I_1,\ldots,I_{k-1},I_{k,1},\ldots,I_{k,\ell}\}$. The next lemma shows that the algorithm indeed returns the variable-partition $\mathcal{I}$.

Lemma 3. *Let $f(\bar{x}) \in \mathbb{F}[x_1,\ldots,x_n]$ be a polynomial and let $\bar{a} \in \mathbb{F}^n$ be a justifying assignment of f. Then $I \subseteq [n]$ satisfies that $f(\bar{a})\cdot f \equiv f|_{x_I=\bar{a}_I}\cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}}$, if and only if I is a disjoint union of sets from the variable-partition of f.*

Proof. Assume that equality holds. Then, as $\bar{a}$ is a justifying assignment of f we have that $f|_{x_I=\bar{a}_I}, f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \not\equiv 0$ and hence $f(\bar{a}) \neq 0$. Consequently, if we define $h(X_I) \triangleq f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}}$ and $g(X_{[n]\setminus I}) \triangleq \frac{f|_{x_I=\bar{a}_I}}{f(\bar{a})}$ then we obtain that $f(\bar{x}) = h(X_I)\cdot g(X_{[n]\setminus I})$. The result follows by uniqueness of decomposition.

To prove the other direction notice that we can write $f(\bar{x}) \equiv h(X_I)\cdot g(X_{[n]\setminus I})$ for two polynomials h and g. We now have that, $f|_{x_I=\bar{a}_I} \equiv h(\bar{a})\cdot g(X_{[n]\setminus I})$ and similarly $f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}} \equiv h(X_I)\cdot g(\bar{a})$. Hence, $f(\bar{a})\cdot f \equiv h(\bar{a})\cdot g(\bar{a})\cdot h(X_I)\cdot g(X_{[n]\setminus I}) \equiv f|_{x_I=\bar{a}_I}\cdot f|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}}$. This concludes the proof of Lemma 3. □

By the lemma, each I_j $(j<k)$ will be added to $\mathcal{I}$ whereas no $I_{k,j}$ will be added to it. Eventually we will have that $J = I_k$ as required. To finish the proof of Lemma 2 we now analyze the running time of the algorithm. The following recursion is satisfied, where $t(n,s,d)$ is the running time of the algorithm on input an (n,s,d)-circuit $C \in \mathcal{C}$: $t(n,s,d) = t(n-1,s,d) + \mathcal{O}(|\mathcal{I}'|\cdot T(s,d)) = t(n-1,s,d) + \mathcal{O}(n\cdot T(s,d))$, which implies that $t(n,s,d) = \mathcal{O}(n^2\cdot T(s,d))$. □

The proof of Theorem 1 easily follows.

Proof (of Theorem 1). We first note that the assumed PIT algorithm also works for circuits in ∂C, when C is an (n,s,d)-circuit from $\mathcal{C}$. Therefore, by Theorem 2 we have an algorithm that finds a justifying assignment $\bar{a}$, as well as computes $\mathrm{var}(C)$.[6] This requires $\mathcal{O}(n^3\cdot d\cdot T(s,d))$ time. Once $\mathrm{var}(C)$ is known we can assume w.l.o.g that $\mathrm{var}(C) = [n]$. Lemma 2 guarantees that Algorithm 1 returns a variable-partition $\mathcal{I}$ in time $\mathcal{O}(n^2\cdot T(s,d))$. At this point we can define, for every $I \in \mathcal{I}$ the polynomial $h_I \triangleq C|_{x_{[n]\setminus I}=\bar{a}_{[n]\setminus I}}$. It is now clear that for $\alpha = C(\bar{a})^{1-|\mathcal{I}|}$ we have that $C = \alpha \prod_{I\in\mathcal{I}} h_I$ is the decomposition of C. Moreover, note that from the definition, each h_i belongs to $\mathcal{C}$ and has size at most s. The total running time can be bounded from above by $\mathcal{O}(n^3\cdot d\cdot T(s,d))$. □

To complete the equivalence between polynomial decomposition and PIT we provide a short proof of Observation 1.

Proof (of Observation 1). Let C be an arithmetic circuit. Consider $C' \triangleq C + y\cdot z$ where y,z are new variables. Clearly, C' is decomposable iff $C \not\equiv 0$ (we also notice that C' is multilinear iff C is). □

[6] It is not difficult to compute $\mathrm{var}(C)$ given Theorem 2 and in fact it is implicit in the proof of the theorem.

3.1 Some Corollaries

An immediate consequence of Theorem 1 is that there are efficient algorithms for polynomial decomposition in circuit classes for which efficient PIT algorithms are known. The proof of the following corollary is immediate given the state of the art PIT algorithms.

Corollary 2. *Let $f(\bar{x})$ be a polynomial. We obtain the following algorithms.*

1. *If f has degree d and m monomials then there is a polynomial time (in m, n, d) black-box algorithm for computing the indecomposable factors of f (this is the circuit class of sparse polynomials, see e.g. [12]).*
2. *If f is computed by a depth-3 circuit with top fan-in k (i.e. a $\Sigma\Pi\Sigma(k)$ circuit, see [17]) and degree d then there is an $(nd)^{\mathcal{O}(k^2)}$ non black-box algorithm for computing the indecomposable factors of f (see [18]). In the black-box model there is an $n^{\mathcal{O}(k^6 \log d)}$ time algorithm over finite fields and an $(nd)^{\mathcal{O}(k^4)}$ time algorithm over $\mathbb{Q}$, for the task (see [28]).*
3. *If f is computed by sum of k Preprocessed Read-Once arithmetic formulas of individual degrees at most d (see [26]), then there is an $(nd)^{\mathcal{O}(k^2)}$ non black-box algorithm for computing the indecomposable factors of f and an $(nd)^{\mathcal{O}(k^2+\log n)}$ black-box algorithm for the problem.*

We now prove Corollary 1. We give a more formal statement, again, in full generality, so that it can be applied to restricted models of arithmetic circuits as well.

Corollary 1 restated: *Let $\mathcal{C}$ be an arithmetic circuit class computing multilinear polynomials. Assume that there is a deterministic PIT algorithm that runs in time $T(s)$ when given as input a circuit of the form $C = C_1 + C_2 \times C_3$, where all the C_i-s $\in \mathcal{C}$ are n-variate circuits of size s and C_2 and C_3 are variable disjoint. Then, there is a deterministic algorithm that when given access (explicit or via a black-box) to an n-variate circuit $C' \in \mathcal{C}$, of size s, runs in time* $\mathrm{poly}(n, T(s))$ *and outputs the* irreducible *factors, $h_1, \ldots, h_k$, of the polynomial computed by C'. Moreover, each h_i can be computed by a size s circuit from $\mathcal{C}$.*

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

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

Proof. The claim is immediate from Theorem 1 and Observations 1 and 2. □

4 Concluding Remarks

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

We note that in restricted models it may be the case that a polynomial and one of its factors will have a different complexity. For example, consider the polynomial $f(x_1, \ldots, x_k) = \prod_{i=1}^{k}(x_i^k - 1) + \prod_{i=1}^{k}(x_i - 1) = \prod_{i=1}^{k}(x_i - 1) \cdot \left(\prod_{i=1}^{k}(x_i^{k-1} + \ldots + 1) + 1\right)$. Then f has $2^{k+1} - 1$ monomials, but one of its irreducible factors has k^k monomials. Thus, for $k = \log n$ we can compute f as a sparse polynomial, but some of its factors will not be sparse (the fact that f has only $\log n$ variables is not really important as we can multiply f by $x_{\log n+1} \cdot \ldots \cdot x_n$ and still have the same problem). Thus, it is also interesting to understand whether it is even possible to have some analog of our result for the factorization problem in restricted models. This question touches of course the interesting open problem of whether the depth of a factor can increase significantly with respect to the depth of the original polynomial.

References

1. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1-2), 1–46 (2004)
2. Shpilka, A., Volkovich, I.: Read-once polynomial identity testing. In: Proceedings of the 40th Annual STOC, pp. 507–516 (2008)
3. Gathen, J.v.z., Gerhard, J.: Modern computer algebra. Cambridge University Press, Cambridge (1999)
4. Kaltofen, E.: Polynomial factorization: a success story. In: ISSAC, pp. 3–4 (2003)
5. Gathen, J.v.z.: Who was who in polynomial factorization. In: ISSAC, vol. 2 (2006)
6. Kayal, N.: Derandomizing some number-theoretic and algebraic algorithms. PhD thesis, Indian Institute of Technology, Kanpur, India (2007)
7. Agrawal, M., Kayal, N., Saxena, N.: Primes is in P. Annals of Mathematics 160(2), 781–793 (2004)
8. Mulmuley, K., Vazirani, U., Vazirani, V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
9. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. JACM 27(4), 701–717 (1980)
10. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Symbolic and algebraic computation, pp. 216–226 (1979)
11. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Inf. Process. Lett. 7(4), 193–195 (1978)
12. Klivans, A., Spielman, D.: Randomness efficient identity testing of multivariate polynomials. In: Proceedings of the 33rd Annual STOC, pp. 216–223 (2001)
13. Agrawal, M., Biswas, S.: Primality and identity testing via chinese remaindering. JACM 50(4), 429–443 (2003)
14. Lipton, R.J., Vishnoi, N.K.: Deterministic identity testing for multivariate polynomials. In: Proceedings of the 14th annual SODA, pp. 756–760 (2003)

15. Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
16. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non commutative models. Computational Complexity 14(1), 1–19 (2005)
17. Dvir, Z., Shpilka, A.: Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. SIAM J. on Computing 36(5), 1404–1434 (2006)
18. Kayal, N., Saxena, N.: Polynomial identity testing for depth 3 circuits. Computational Complexity 16(2), 115–138 (2007)
19. Arvind, V., Mukhopadhyay, P.: The monomial ideal membership problem and polynomial identity testing. In: Proceedings of the 18th ISAAC, pp. 800–811 (2007)
20. Karnin, Z.S., Shpilka, A.: Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. In: Proceedings of the 23rd Annual CCC, pp. 280–291 (2008)
21. Saxena, N., Seshadhri, C.: An almost optimal rank bound for depth-3 identities. In: Proceedings of the 24th annual CCC (2009)
22. Dvir, Z., Shpilka, A., Yehudayoff, A.: Hardness-randomness tradeoffs for bounded depth arithmetic circuits. SIAM J. on Computing 39(4), 1279–1293 (2009)
23. Agrawal, M., Vinay, V.: Arithmetic circuits: A chasm at depth four. In: Proceedings of the 49th Annual FOCS, pp. 67–75 (2008)
24. Arvind, V., Mukhopadhyay, P.: Derandomizing the isolation lemma and lower bounds for circuit size. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 276–289. Springer, Heidelberg (2008)
25. Kayal, N., Saraf, S.: Blackbox polynomial identity testing for depth 3 circuits. Electronic Colloquium on Computational Complexity (ECCC), (32) (2009)
26. Shpilka, A., Volkovich, I.: Improved polynomial identity testing for read-once formulas. In: APPROX-RANDOM, pp. 700–713 (2009)
27. Karnin, Z.S., Mukhopadhyay, P., Shpilka, A., Volkovich, I.: Deterministic identity testing of depth 4 multilinear circuits with bounded top fan-in. In: Proceedings of the 42th Annual STOC (2010)
28. Saxena, N., Seshadhri, C.: From sylvester-gallai configurations to rank bounds: Improved black-box identity test for depth-3 circuits. Electronic Colloquium on Computational Complexity (ECCC) (013) (2010)
29. Heintz, J., Schnorr, C.P.: Testing polynomials which are easy to compute (extended abstract). In: Proceedings of the 12th annual STOC, pp. 262–272 (1980)
30. Raz, R.: Multi-linear formulas for permanent and determinant are of super-polynomial size. In: Proceedings of the 36th Annual STOC, pp. 633–641 (2004)
31. Raz, R.: Multilinear $NC_1 \neq$ Multilinear NC_2. In: Proceedings of the 45th Annual FOCS, pp. 344–351 (2004)
32. Raz, R., Shpilka, A., Yehudayoff, A.: A lower bound for the size of syntactically multilinear arithmetic circuits. SIAM J. on Computing 38(4), 1624–1647 (2008)
33. Raz, R., Yehudayoff, A.: Lower bounds and separations for constant depth multilinear circuits. In: IEEE Conference on Computational Complexity, pp. 128–139 (2008)
34. Hancock, T.R., Hellerstein., L.: Learning read-once formulas over fields and extended bases. In: Proceedings of the 4th Annual COLT, pp. 326–336 (1991)
35. Bshouty, N.H., Hancock, T.R., Hellerstein, L.: Learning arithmetic read-once formulas. SIAM J. on Computing 24(4), 706–735 (1995)
36. Kaltofen, E., Koiran, P.: On the complexity of factoring bivariate supersparse (lacunary) polynomials. In: ISSAC, pp. 208–215 (2005)