Deterministic Polynomial Identity Testing in Non Commutative Models

Ran Raz*
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel
ranraz@wisdom.weizmann.ac.il

Amir Shpilka†
DEAS, Harvard University
LCS, M.I.T.
Cambridge, MA
amirs@deas.harvard.edu

Abstract

We give a deterministic polynomial time algorithm for polynomial identity testing in the following two cases:

1. **Non Commutative Arithmetic Formulas:** The algorithm gets as an input an arithmetic formula in the non-commuting variables $x_1, ..., x_n$ and determines whether or not the output of the formula is identically 0 (as a formal expression).
2. **Pure Arithmetic Circuits:** The algorithm gets as an input a pure arithmetic circuit (as defined by Nisan and Wigderson [3]) in the variables $x_1, ..., x_n$ and determines whether or not the output of the circuit is identically 0 (as a formal expression).

We also give a deterministic polynomial time identity testing algorithm for non commutative algebraic branching programs as defined by Nisan [2]. One application is a deterministic polynomial time identity testing for multilinear arithmetic circuits of depth 3.

Finally, we observe an exponential lower bound for the size of pure arithmetic circuits for the permanent and for the determinant. (Only lower bounds for the *depth* of pure circuits were previously known [3]).

*Part of this work was done when the author was a member at the Institute for Advanced Study, Princeton. Research supported by The Israel Science Foundation.

†Supported by National Security Agency (NSA) and Advanced Research and Development Activity (ARDA) under Research Office (ARO) contract no. DAAD19-01-1-0506.

1 Introduction

Let $\mathbb{F}$ be a field and let $\mathcal{C}$ be an arithmetic circuit in the input variables $x_1, ..., x_n$, over the field $\mathbb{F}$. The output $\mathcal{C}(x_1, ..., x_n)$ is hence a polynomial in the ring $\mathbb{F}[x_1, ..., x_n]$. Can one determine whether this polynomial is identically 0 ? [1]

Schwartz and Zippel showed that if $f \in \mathbb{F}[x_1, ..., x_n]$ is a polynomial of degree d, different than the 0 polynomial, and $c_1, ..., c_n \in \mathbb{F}$ are chosen uniformly at random among D possibilities, then $f(c_1, ..., c_n) = 0$ with probability at most d/D [6, 4]. This gives a very efficient probabilistic procedure to test whether the output of an arithmetic circuit is identically 0. An outstanding open problem is to find an equivalent *deterministic* procedure that runs in polynomial time (in the size of the circuit).

In this paper, we are interested in the non-commutative case, where the input variables for the circuit $\mathcal{C}$ are non-commuting. The output $\mathcal{C}(x_1, ..., x_n)$ is hence a formal expression in the ring $\mathbb{F}\{x_1, ..., x_n\}$ of polynomials over non-commuting variables $x_1, ..., x_n$. Note that any arithmetic circuit can be evaluated over commuting variables, to get a polynomial in $\mathbb{F}[x_1, ..., x_n]$, as well as over non-commuting variables [2], to get a polynomial in $\mathbb{F}\{x_1, ..., x_n\}$. In some cases, the output of the circuit is identically 0 when evaluated over commuting variables and non-zero when evaluated over non-commuting variables. At the other hand, if the output is identically 0 over non-commuting variables it is surely identically 0 over commuting variables as well. Thus, the fact that the output is identically 0 over non-commuting variables may serve as a *proof* that it is identically 0 over commuting variables.

An arithmetic circuit is called a formula if the fan-out of every gate in the circuit is at most one. Our main result is a deterministic polynomial time algorithm for identity testing for arithmetic formulas in the non-commutative case. The algorithm gets as an input an arithmetic formula and determines whether or not the output of the formula is identically 0, when evaluated over non-commutative variables.

We also give a deterministic polynomial time algorithm for identity testing for, so called, pure arithmetic circuits (first defined by Nisan and Wigderson [3]). Roughly speaking, a pure arithmetic circuit is a multilinear arithmetic circuit in sets of variables $X_1, ..., X_d$, such that if T_1 is the set of sets of variables that the output of a gate v_1 depends on and T_2 is the set of sets of variables that the output of a gate v_2 depends on then either $T_1 \subset T_2$ or $T_2 \subset T_1$ or $T_1 \cap T_2 = \emptyset$ (for every two gates v_1, v_2 in the circuit).

One application of our results is a deterministic polynomial time algorithm for identity testing for multilinear arithmetic circuits of depth 3. The algorithm gets as an input a multilinear arithmetic circuit of depth 3 and determines whether or not the output of the circuit is identically 0 (in the standard commutative case).

Note that identity testing is equivalent to *equivalence testing*, e.g., given two polynomials f, g determine whether or not $f = g$. In many research areas of theoretical computer science, the ability to do efficient equivalence testing is a very desirable property of a computational model. One example for a computational model with efficient equivalence testing is OBDDs (restricted order bounded decision diagrams), and possibly this is one of the reasons for the popularity of OBDDs

[1] Note that there are two different ways to interpret this question. First, one can ask whether the output is identically 0 as a function. In this paper, we are interested in the second interpretation: Is the output of the circuit identically 0 as a formal expression ? (that is, the output is the 0 member of the ring $\mathbb{F}[x_1, ..., x_n]$). Note however that if the degree of a polynomial is smaller than the size of the field then the two notions of being identically 0 are equivalent.

[2] We assume here that the inputs for every product gate in the circuit are ordered.

in many research areas. Here we give deterministic equivalence testing for computational models stronger than OBDDs. In particular, we give deterministic equivalence testing for, so called, *non-commutative arithmetic branching programs* (as defined by Nisan [2]). Note that non-commutative arithmetic branching programs are stronger than OBDDs.

1.1 Methods

Our methods are non-black-box methods. That is, we use the structure of the circuit, as opposed to using it as a black box that evaluate the output polynomial. We use tools and methods that were previously used to prove lower bounds for non-commutative models of arithmetic circuits [2, 3].

Nisan showed that if $f \in \mathbb{F}\{x_1, ..., x_n\}$ is computed by a small arithmetic formula then the space spanned by all partial derivatives of f is of small dimension [2]. Here, we show how to recursively check whether or not all these partial derivatives are identically 0. In the case of pure circuits, it is not true that the space spanned by all partial derivatives is of small dimension. Nevertheless, it is still true that the space spanned by certain subsets of all partial derivatives is of small dimension. As before, we show how to recursively check whether or not all the partial derivatives in these subsets are identically 0.

Lower bounds for the size of non-commutative arithmetic formulas and for the depth of pure arithmetic circuits are known [2, 3]. A recent result of Impagliazzo and Kabanets shows that there are tight connections between deterministic polynomial identity testing and proving lower bounds for circuit size [1]. In particular, Impagliazzo and Kabanets prove that strong lower bounds for the size of arithmetic circuits imply quasi-polynomial time derandomization of Schwartz-Zippel test. Note however that this result doesn't hold for non-commutative models and hence it doesn't give anything for the models discussed in this paper. Here, we use methods developed to prove lower bounds for certain classes, to directly give a deterministic polynomial time identity testing for these classes.

Previous to this paper, there was no lower bound for the size of pure arithmetic circuits. To complete the picture, we prove such a lower bound. We show an exponential lower bound for the size of pure arithmetic circuits for the permanent and for the determinant. The lower bound follows easily by the method of partial derivatives, but was somehow overlooked before.

1.2 Organization

In section 2 we give a polynomial identity testing algorithm to non commutative arithmetic formulas. In section 3 we give an algorithm for polynomial identity testing in pure circuits. In section 4 we show that these algorithms also give identity testing for multilinear depth-3 $\Sigma\Pi\Sigma$ circuits. In section 5 we state our lower bound for pure circuits. The proof of the lower bound is in appendix A.

2 Non Commutative Arithmetic Formulas

2.1 Preliminaries

In this section we give a polynomial time deterministic algorithm to decide whether a polynomial given by a non commutative formula is identically zero or not. Let $X = \{x_1, ..., x_n\}$ be our set of variables. An arithmetic formula is a binary tree whose edges are directed toward the root. The leaves of the tree are labeled with input variables or field elements. Any inner vertex is labeled with

one of the arithmetic operations $\{+, \times\}$. Every edge is labeled with a constant from the field in which we are working. A node v that is labeled with the arithmetic operation $+$, and whose children are v_1 and v_2, computes the polynomial $\alpha_1 P(v_1) + \alpha_2 P(v_2)$, where $P(v_i)$ is the polynomial computed at v_i, and α_i is the constant labeling the edge between v and v_i (and similarly when v is labeled with $\times$). The output of the formula is the polynomial computed at the root. The size of the formula is the number of nodes of the tree. We consider a variant of this model - non commutative arithmetic formulas. Let $\mathbb{F}\{x_1, ..., x_n\}$ be the polynomial ring over the field $\mathbb{F}$ in the non commuting variables $x_1, \ldots, x_n$. That is, in $\mathbb{F}\{x_1, ..., x_n\}$ the formal expressions $x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}$ and $x_{j_1} \cdot x_{j_2} \cdot \ldots \cdot x_{j_l}$ are equal if and only if $k = l$ and $\forall l \ i_l = j_l$ (whereas in the commutative ring of polynomials we have that any monomial remains the same even if we permute its variables, e.g. $x_1 \cdot x_2 = x_2 \cdot x_1$). A non commutative arithmetic formula is an arithmetic formula where multiplications are done in the ring $\mathbb{F}\{x_1, ..., x_n\}$. As two polynomials in this ring do not necessarily commute, we have to distinguish in every multiplication gate between the left son and the right son.

Note that every polynomial can be computed by a non commutative formula. And that given a non commutative formula we can evaluate it over a commutative domain.

In [2] Nisan proved exponential lower bounds on the size of non commutative formulas that compute the determinant or the permanent. An important ingredient of Nisan's proof was a reduction from non commutative formulas to algebraic branching programs.

Definition 1 (Nisan) *An algebraic branching program (ABP) is a directed acyclic graph with one vertex of in-degree zero, which is called* source*, and one vertex of out-degree zero, which is called* sink*. The vertices of the graph are partitioned into levels numbered* $0, ..., d$*. Edges may only go from level* i *to level* $i + 1$ *for* $i = 0, ..., d - 1$*. The source is the only vertex at level* 0 *and the sink is the only vertex at level* d*. Each edge is labeled with a homogeneous linear form in the input variables. The size of the ABP is the number of vertices.*

The polynomial that is computed by an ABP is the sum over all directed paths from the source to the sink of the product of the linear functions that label the edges of the path. Notice that algebraic branching program is a *commutative* model of computation, but we will be interested in non commutative ABP's which are ABP's in which multiplications are done in the ring $\mathbb{F}\{x_1, ..., x_n\}$. In particular if the edges of a sink-source path are $e_1, ..., e_d$, where e_i goes between level $i - 1$ and level i, and $\ell_i(X)$ is the linear function labeling e_i, then the order in which we take the product is

$$\ell_1(X) \cdot \ell_2(X) \cdot \ldots \cdot \ell_d(X)$$

It is clear that an ABP with $d + 1$ levels computes a homogeneous polynomial of degree d. For two vertices v_1, v_2, of an ABP A, such that v_1 is in level i_1 and v_2 is in level i_2 and $i_1 < i_2$, we denote with $A(v_1, v_2)$ the polynomial computed when considering a new ABP in which v_1 is the source and v_2 is the sink (that is, we only consider paths leading from v_1 to v_2).

Lemma 2 (Nisan) *For a polynomial* f *of degree* d *that is computed by a non commutative formula of size* s*, we can compute* $d + 1$ *non commutative ABP's such that the* i*'th ABP computes the homogeneous part of* f *of degree* i *for* $i = 0, ..., d$*. Moreover, the size of each of these ABP's is at most* $d(s + 1)$*. The total time that it takes to output these ABP's is polynomial in* s*.*

Proof: The reduction is done in four stages. In the first stage we recursively construct a "non homogeneous" ABP for the right son of the root of the formula and a "non homogeneous" ABP for the left son of the root of the formula, where in a non homogeneous ABP we mean that the ABP is not necessarily leveled, that edges may be labeled with constants, and that it may output a non

homogeneous polynomial. Then, according to the arithmetic operation in the root, we either wire the ABP's in parallel (a plus gate) or wire them sequentially (a product gate). After doing so we get a non homogeneous ABP. In the second stage, we replace each vertex v (except the source) of the non homogeneous ABP with $d+1$ new vertices, $(v,0),(v,1),...,(v,d)$, such that vertex (v,i) computes the homogeneous part of degree i of v. Since the ABP computes a polynomial of degree d we don't need to keep monomials of higher degree. Notice that this process results in an almost leveled ABP. At level i we have all the vertices of the form (v,i). It is easy to see that edges that are not labeled with constants connect vertices in two consecutive levels, and that edges labeled with constants connect two vertices in the same level. Next, in the third stage, we get rid of the edges that are labeled with constants. Given an edge labeled with a constant α, which is directed from v to u, we erase the edge, and instead we connect every (directed) neighbor of u to v, and label the new edge with α times the label of its original copy at u. If u was a sink (no edges going out) then we do not erase the edge. It is easy to see that every sink in the ABP computes the same output as before. However, this is not an ABP as there might be more than one sink. Actually there can be a sink in every level (except level 0). In the final step we take $d+1$ copies of the ABP. In the i'th copy $(i=0,...,d)$ we only keep the vertices that are used in the computation of the sink that is in level i. □

Note that the commutative version of lemma 2 (in which we consider commutative formulas and commutative ABP's) is also true. As we are interested in the non commutative models we stated and proved only the non commutative case.

2.2 Identity Testing of Non Commutative Formulas

Our algorithm for deciding whether a non commutative formula computes the zero polynomial will first transform the given non commutative formula to $d+1$ ABP's, such that the i'th ABP computes the homogeneous part of degree i of the output of the formula (which might be 0). Then it will verify that each of those ABP's compute the zero polynomial. As we already showed the transformation from a non commutative formula to the ABP's, all that we need to do is to show how to verify that a given ABP is identically zero. The first step in the verification is to reduce the number of levels of the ABP by 1, without increasing the size, and get a new ABP such that the new ABP is identically zero iff the old ABP is identically zero. Clearly if we can perform this transformation in time polynomial in n and the size of the ABP then we are done.

Reducing the depth of Algebraic Branching Programs

The idea is the following: Given an ABP A, compute the polynomials of degree 2 that are computed between the source and the vertices on level 2. For every monomial of the form x_ix_j (i may equal j) we get a new ABP by considering only the terms x_ix_j from each of these polynomials. Since we are working in the non commutative polynomial ring, A computes the zero polynomial iff all the new ABP's compute the zero polynomial. We then show how to merge the new ABP's to a single ABP, $\hat{A}$, which has only d levels, such that $\hat{A}$ is identically zero iff A is identically zero. From the proof it will be clear that $\hat{A}$ is constructed by removing from A the vertices of level 1 and the edges adjacent to them, and then connecting the source directly to level 2. The technical part is to label the edges, going out of the source, with the "correct" linear functions.

So assume that $f(X)$ is computed by an ABP A, with levels $0,...,d$. Denote with m the number of vertices in level 2 of A. Let Y be a new set of auxiliary variables such that $Y=\{y_1,...,y_m\}$. We

now construct a new ABP $\hat{A}$ with the following properties

- $\hat{A}$ has d levels.
- The size of $\hat{A}$ is smaller than the size of A.
- The graph spanned by levels $1, ..., d-1$ of $\hat{A}$ is the same as the graph spanned by levels $2, ..., d$ of A. The linear functions labeling two corresponding edges are the same.
- Only the Y variables appear in the linear functions going from the source to level 1 of $\hat{A}$.
- $\hat{A}$ computes the zero polynomial iff A computes the zero polynomial.
- We make at most $O(mn^2)$ operations when transforming A to $\hat{A}$.

Denote with s the source of A and with t the sink of A. Let $v_1, ..., v_m$ be the vertices in level 2 of A. Let $A(v_i, t)$ be the polynomial that is computed by the ABP with v_i as source and t as sink. Similarly define $A(s, v_i)$. Clearly

$$A(s,t) = \sum_{i=1}^{m} A(s,v_i) \cdot A(v_i,t)$$

Let

$$A(s,v_i) = \sum_{k=1}^{n} \alpha_{i,k} {x_k}^2 + \sum_{1 \le j < k \le n} \beta_{i,j,k} x_j x_k + \sum_{1 \le j < k \le n} \gamma_{i,j,k} x_k x_j$$

We have that

$$A(s,t) = \sum_{i=1}^{m} A(s,v_i) \cdot A(v_i,t) =$$

$$\sum_{i=1}^{m} \left(\sum_{k=1}^{n} \alpha_{i,k} {x_k}^2 + \sum_{1 \le j < k \le n} \beta_{i,j,k} x_j x_k + \sum_{1 \le j < k \le n} \gamma_{i,j,k} x_k x_j \right) \cdot A(v_i,t) =$$

$$\sum_{k=1}^{n} {x_k}^2 \sum_{i=1}^{m} \alpha_{i,k} \cdot A(v_i,t) + \sum_{1 \le j < k \le n} x_j x_k \sum_{i=1}^{m} \beta_{i,j,k} \cdot A(v_i,t) + \sum_{1 \le j < k \le n} x_k x_j \sum_{i=1}^{m} \gamma_{i,j,k} \cdot A(v_i,t)$$

The following claim is obvious

Claim 3 *$A(s,t)$ computes the zero polynomial ($A \equiv 0$) iff*

$$\forall 1 \le k \le n \ \ \sum_{i=1}^{m} \alpha_{i,k} \cdot A(v_i,t) = 0$$

and

$$\forall 1 \le j < k \le n \ \ \sum_{i=1}^{m} \beta_{i,j,k} \cdot A(v_i,t) = \sum_{i=1}^{m} \gamma_{i,j,k} \cdot A(v_i,t) = 0$$

Proof: The if part is trivial. The other direction follows immediately from the fact that each $A(v_i, t)$ is a homogeneous polynomial of degree $d-2$, and that the x_i's do not commute. □

Define the following vectors

$$1 \leq k \leq n \quad \alpha_k = (\alpha_{1,k}, ..., \alpha_{m,k})$$
$$1 \leq j < k \leq n \quad \beta_{j,k} = (\beta_{1,j,k}, ..., \beta_{m,j,k})$$
$$1 \leq j < k \leq n \quad \gamma_{j,k} = (\gamma_{1,j,k}, ..., \gamma_{m,j,k})$$
$$\vec{a} = (A(v_1, t), A(v_2, t), \ldots, A(v_m, t))$$

We can restate our claim in the following manner: $A(s,t) \equiv 0$ iff

$$\forall 1 \leq k \leq n \ \ \langle \alpha_k, \vec{a} \rangle = 0$$

and

$$\forall 1 \leq j < k \leq n \ \ \langle \beta_{j,k}, \vec{a} \rangle = \langle \gamma_{j,k}, \vec{a} \rangle = 0$$

Where $\langle \, , \rangle$ is the inner product form. Consider

$$V = \mathrm{span}\{\alpha_{\mathrm{k}}, \beta_{\mathrm{j,k}}, \gamma_{\mathrm{j,k}}\}_{\mathrm{j,k}}$$

Then $A(s,t) = 0$ iff $\vec{a} \perp V$. Let $r = \dim(V)$ and $u_1, ..., u_r$ be a basis for V. Clearly $r \leq m$. Let

$$u_j = (u_{j,1}, ..., u_{j,m})$$

For $1 \leq i \leq m$ define

$$\ell_i(y_1, ..., y_r) = \sum_{j=1}^{r} u_{j,i} y_j$$

Let

$$\hat{A} = \sum_{i=1}^{m} \ell_i \cdot A(v_i, t) = \sum_{i=1}^{m} (\sum_{j=1}^{r} u_{j,i} y_j) \cdot A(v_i, t) = \sum_{j=1}^{r} y_j \sum_{i=1}^{m} u_{j,i} A(v_i, t) = \sum_{j=1}^{r} y_j \langle u_j, \vec{a} \rangle$$

We have that

$$\hat{A} \equiv 0 \ \ \Leftrightarrow \ \ \forall 1 \leq j \leq r \ \langle u_j, \vec{a} \rangle = 0 \ \ \Leftrightarrow \ \ \vec{a} \perp V \ \ \Leftrightarrow \ \ A \equiv 0$$

Notice that $\hat{A}$ can be computed by an ABP with d levels in the following manner. Remove all the vertices in level 1 from the ABP A. For every $i = 1, ..., m$ we connect v_i (the i'th vertex in level 2) to the sink with an edge labeled with ℓ_i. We now give a bound on the number of operations needed to compute $\hat{A}$. With $O(mn^2)$ operations we can compute the vectors $\{\alpha_k, \beta_{j,k}, \gamma_{j,k}\}_{j,k}$. Similarly $O(mn^2)$ operations are needed to compute the ℓ_j's. Thus with $O(mn^2)$ operations we transformed A to $\hat{A}$. Clearly $\hat{A}$ has all the properties that we stated above.

Theorem 4 *Let A be an ABP of size S then we can verify whether $A \equiv 0$ in time $O(Sn^2)$.*

Proof: Let $S = m_1 + ... + m_{d-1}$ where m_i is the size of level i. By the above procedure we can reduce A to an ABP with 2 levels in time

$$\sum_{i=1}^{d-1} O(m_i n^2) = O(Sn^2)$$

We can verify in linear time whether an ABP with 2 levels is identically zero or not. □

Our main theorem of this section now follows.

Theorem 5 *Given a non commutative arithmetic formula of size s we can verify in time polynomial in s whether the formula is identically zero.*

Proof: The theorem follows immediately from lemma 2 and theorem 4. □

3 Pure Circuits

In [3] the authors defined a class of circuits called *pure* circuits, and proved lower bounds on the depth of pure circuits computing the product of d $n \times n$ matrices. Other lower bounds on the depth of pure circuits follow from the work of Nisan [2].

In this section we give a deterministic polynomial time algorithm that checks whether a pure circuit computes the zero polynomial or not.

3.1 Preliminaries

Let $[d] = \{1, ..., d\}$. Let $X = \dot{\cup}_{i=1}^{d} X_i$ be a partition of the set of variables to d disjoint sets. Let $X_i = \{x_{i,1}, ..., x_{i,n_i}\}$. A polynomial $f(X)$ is multilinear if each of its monomials contains exactly one variable from each of the d sets. For a set $S \subset [d]$ let $P(S)$ be the set of all multilinear polynomials with respect to $\dot{\cup}_{i \in S} X_i$.

Definition 6 *[3] A multilinear arithmetic circuit is an arithmetic circuit in which every gate computes a multilinear polynomial (that is, if f_v is the polynomial computed at gate v then $f_v \in P(S)$ for some S).*

Thus, for every gate in a multilinear circuit corresponds a set $S \subset [d]$. We now define pure circuits.

Definition 7 *[3] A multilinear circuit is pure if for every two sets S and T, associated with nodes of the circuit, we have that $S \subset T$ or $T \subset S$ or $S \cap T = \emptyset$.*

A way to view pure circuits is the following. A *partition tree* of $[d]$ is a binary tree with d leaves where each leaf is labeled with one of the variables in a $1-1$ correspondence (each variable appears in exactly one of the leaves). Every internal node of the tree is labeled with the variables labeling the leaves in its subtree.

To every pure circuit we can associate a partition tree of $[d]$ such that every gate in the circuit computes a polynomial in $P(S)$ only for sets S that label some node v of the tree. Moreover, a multiplication gate associated with v, multiplies two polynomials, one belonging to $P(T_1)$ and the other belonging to $P(T_2)$, where T_1 and T_2 are the sets labeling the children of v in the tree.

3.2 Identity Testing of Pure Circuits

We now give our identity testing algorithm for pure circuits. The idea is similar to the non commutative case. Given a pure circuit we can easily construct the corresponding partition tree. Then in a similar way to the non commutative case we erase two leaves of the tree that share a common father, and define a new pure circuit that corresponds to the new tree, such that the new circuit is identically zero iff the old circuit is. We continue doing so until we are left with a partition tree with only one leaf and then we easily check the circuit.

Reducing the number of leaves

Let $\mathcal{C}$ be a pure circuit and $T(\mathcal{C})$ be its associated partition tree. Assume w.l.o.g. that the leaves labeled with $1,2$ are the children of the same father. Denote $S=\{1,2\}$. Clearly the father of $1,2$ is labeled with S. Let $M_1,...,M_t$ be all the multiplication gates in $\mathcal{C}$ that compute polynomials from $P(S)$. Clearly we can write

$$\forall 1\leq i\leq t \quad M_i=\ell_{i,1}(X_1)\cdot\ell_{i,2}(X_2)$$

for two linear functions $\ell_{i,1},\ell_{i,2}$. Let

$$\forall 1\leq j\leq 2 \ , \ \forall 1\leq i\leq t \quad \ell_{i,j}(X_j)=\sum_{k=1}^{n_j}\alpha_{i,j,k}x_{j,k}$$

Since $\mathcal{C}$ is a multilinear circuit we get that there exist polynomials $g_1,...,g_t\in P(\{3,...,d\})$ such that the output of $\mathcal{C}$ is

$$\mathcal{C}=\sum_{i=1}^{t}M_i\cdot g_i=\sum_{i=1}^{t}\ell_{i,1}(X_1)\cdot\ell_{i,2}(X_2)\cdot g_i=\sum_{i=1}^{t}\left(\sum_{k=1}^{n_1}\alpha_{i,1,k}x_{1,k}\right)\cdot\left(\sum_{l=1}^{n_2}\alpha_{i,2,l}x_{2,l}\right)\cdot g_i=$$

$$\sum_{k=1}^{n_1}\sum_{l=1}^{n_2}x_{1,k}x_{2,l}\sum_{i=1}^{t}\alpha_{i,1,k}\cdot\alpha_{i,2,l}\cdot g_i$$

As $\mathcal{C}$ is multilinear we get, as before, that $\mathcal{C}\equiv 0$ iff

$$\forall 1\leq k\leq n_1 \ , \ \forall 1\leq l\leq n_2 \quad \sum_{i=1}^{t}\alpha_{i,1,k}\cdot\alpha_{i,2,l}\cdot g_i=0$$

Let

$$v_{k,l}=(\alpha_{1,1,k}\cdot\alpha_{1,2,l},...,\alpha_{t-1,1,k}\cdot\alpha_{t-1,2,l},\alpha_{t,1,k}\cdot\alpha_{t,2,l})$$

$$G=(g_1,...,g_t)$$

We have

$$\mathcal{C}\equiv 0 \ \Leftrightarrow \ \forall 1\leq k\leq n_1 \ , \ \forall 1\leq l\leq n_2 \ \langle v_{k,l},G\rangle=0$$

Let $V=\text{span}\{v_{k,l}\}_{k,l}$. Let $u_1,...,u_r$ be a basis for V (obviously $r\leq t$). Denote $u_i=(u_{i,1},...u_{i,t})$. We have

$$\mathcal{C}\equiv 0 \ \Leftrightarrow \ \forall 1\leq i\leq r \ \langle u_i,G\rangle=0$$

Let X_{d+1} be a new set of variables, $X_{d+1}=\{x_{d+1,1},...,x_{d+1,r}\}$. Define

$$\forall 1\leq j\leq t \quad \ell_j(X_{d+1})=\sum_{i=1}^{r}u_{i,j}x_{d+1,i}$$

We get

$$\sum_{j=1}^{t}\ell_j\cdot g_j=\sum_{j=1}^{t}\left(\sum_{i=1}^{r}u_{i,j}x_{d+1,i}\right)\cdot g_j=\sum_{i=1}^{r}x_{d+1,i}\sum_{j=1}^{t}u_{i,j}\cdot g_j=\sum_{i=1}^{r}x_{d+1,i}\langle u_i,G\rangle$$

Thus

$$\mathcal{C}\equiv 0\Leftrightarrow\sum_{j=1}^{t}\ell_j\cdot g_j=0$$

We now construct a new pure circuit $\hat{\mathcal{C}}$ in the following way: replace the multiplication gates $M_1, ..., M_t$ with the linear functions $\ell_1(X_{d+1}), ..., \ell_t(X_{d+1})$ (that is, remove from $\mathcal{C}$ all gates computing linear functions in X_1 or X_2, and replace each of the M_i's with a small circuit computing the relevant linear form). We get that $\hat{\mathcal{C}}$ is a pure circuit satisfying

$$\hat{\mathcal{C}} = \sum_{j=1}^{t} \ell_j \cdot g_j$$

The partition tree corresponding to $\hat{\mathcal{C}}$ is obtained from the tree of $\mathcal{C}$ by removing the leaves $1, 2$ and labeling their father with $d+1$, and then "fixing" the labels of the other internal nodes.

We now analyze the time that it takes us to reduce the number of leaves. Let $s = s(\mathcal{C})$ be the size of $\mathcal{C}$. We assume for simplicity that $s \geq \max\{n_1, ..., n_d\}$ (otherwise we let s equal that maximum). For every set $T \subset [d]$ let t_T be the number of multiplication gates in $\mathcal{C}$ that output a polynomial in $P(T)$. We see that the time that it takes to compute $\ell_{i,j}$ for $j = 1, 2$ and $i = 1, ..., t_{\{1,2\}}$ (in the procedure above we used t instead $t_{\{1,2\}}$) is bounded by s, as we can easily compute the output of each plus gate. Computing the v_i's requires $O(n_1 \cdot n_2)$ steps. Then computing the u_i's and the ℓ_i's takes at most $O(n_1 \cdot n_2 \cdot t_{\{1,2\}}) = O(s^3)$ steps.

Theorem 8 *Let $\mathcal{C}$ be a pure circuit of size s, then we can verify whether $\mathcal{C} \equiv 0$ in time polynomial in s.*

Proof: Given a pure circuit $\mathcal{C}$ of size s we perform the process above. At the i'th step we introduce a new set of variables, X_{d+i}, whose size is at most s. Thus, after the i'th step we have a pure circuit defined on $d - i$ sets of variables, each of size smaller than s. Therefor, each step requires at most $O(s^3)$ operations. We perform $d - 1$ steps and then verify that the circuit outputs the zero polynomial (as it now computes a linear function). Thus, in time $O(s^3 d)$ we can verify whether $\mathcal{C}$ is identically zero or not. □

4 Depth-3 Multilinear Circuits

As an example we show that our identity testing algorithm, for non commutative formulas, applies to depth-3 multilinear ΣΠΣ circuits, as they are a special case of non commutative formulas.

A depth-3 arithmetic circuit with a plus gate at the root is called a ΣΠΣ circuit. We can assume w.l.o.g. that a ΣΠΣ circuit, $\mathcal{C}$, has the following structure: There are 3 levels of computations in $\mathcal{C}$. The gates at the bottom level compute linear functions in the input variables. Then there is a level of multiplication gates, each computing a product of linear functions. The top gate sums the outputs of the multiplication gates. It is obvious that we can write

$$\mathcal{C} = \sum_{i=1}^{s} \prod_{j=1}^{d_i} L_{i,j}(X)$$

where each $L_{i,j}$ is a linear function in the input variables $X = \{x_1, \ldots, x_n\}$.

Definition 9 *A ΣΠΣ circuit is called multilinear with respect to $X = \dot{\cup}_{i=1}^{d} X_i$ if every linear function, computed at the bottom, is defined on one of the sets X_i, and each multiplication gate multiplies d*

linear functions $L_1, \ldots, L_d$ where every L_i is over a distinct set of variables X_i. Thus, a depth-3 multilinear circuit $\mathcal{C}$ has the following structure:

$$\mathcal{C} = \sum_{i=1}^{s} \prod_{j=1}^{d} L_{i,j}(X_j)$$

where $L_{i,j}$ is a linear function.

Depth 3 multilinear circuits were studied in [2, 3] and exponential lower bounds are known for the determinant and the permanent.

It is easy to verify that a depth-3 multilinear circuit can be viewed as both a non commutative formula and a pure circuit. We thus get the following theorem as a corollary of our previous results.

Theorem 10 *Let $X = \dot{\cup}_{i=1}^{d} X_i$, $|X| = n$. Given a depth 3 multilinear circuit on X, with s multiplication gates, we can determine in time* poly(s) *whether the circuit is identically zero or not.*

5 Lower Bounds on the Size of Pure Circuits

Let $X = \dot{\cup}_{i=1}^{n} X_i$, where $X_i = \{x_{i,1}, ..., x_{i,n}\}$. The permanent and determinant of X are defined in the following way

$$PERM(X) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)} \quad , \quad DET(X) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \cdot \prod_{i=1}^{n} x_{i,\sigma(i)}$$

In [1] Impagliazzo and Kabanets show that derandomizing identity testing for arithmetic circuits implies one of the following: $NEXP \not\subset P/\text{poly}$ or the permanent does not have polynomial size arithmetic circuits. This result shows the tight connection between circuit lower bounds and derandomization of polynomial identity testing. Exponential lower bounds for the permanent and the determinant are known in the models of non commutative formulas [2] and depth 3 multilinear circuits [3]. For pure circuits only lower bounds on the depth are known (again [2, 3]). By a slight modification of the arguments of [1] we get that our derandomization algorithm for pure circuits implies that $NEXP \not\subset P/\text{poly}$ or the permanent does not have polynomial size *pure* circuits.

By a more direct approach we are able to prove lower bounds for pure circuits computing the permanent or the determinant. Our proof of the lower bounds uses one of the two techniques known today for proving exponential lower bounds in models of arithmetic circuits - the partial derivative method (the other technique is the approximation method). The partial derivatives technique was implicitly used in [2] and was formally defined in [3] (see also [5]).

We now state our lower bound. The proof is postponed to appendix A.

Theorem 11 *Any pure circuit computing the permanent or the determinant of an $n \times n$ matrix X, has size $2^{\Omega(n)}$.*

References

[1] Russell Impagliazzo and Valentine Kabanets. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, 2003. to appear.

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

[3] Noam Nisan and Avi Wigderson. Lower bound on arithmetic circuits via partial derivatives. *Computational Complexity*, 6:217–234, 1996.

[4] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[5] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic formulae over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

[6] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation (EUROSAM '79, Internat. Sympos., Marseille, 1979)*, pages 216–226. Springer, Berlin, 1979.

A Proof of theorem 11

We use the notations of section 3. Let $X = \dot{\cup}_{i=1}^{n} X_i$, $X_i = \{x_{i,1}, ..., x_{i,n}\}$, be a partition of an $n \times n$ matrix of indetermined according to its rows. For a set $S \subset [n]$ we define $M(S)$ to be the set of all multilinear *monomials* with respect to $\dot{\cup}_{i \in S} X_i$ (this is similar to the definition of $P(S)$), that is

$$M(S) = \{\, m \mid m = \prod_{i \in S} x_{i,j_i} \text{ for some } \{j_i\}_{i \in S}\}$$

Definition 12 *Let f be a multilinear polynomial in $P([n])$. For a monomial $m = \prod_{i \in S} x_{i,j_i} \in M(S)$ let*

$$\frac{\partial f}{\partial m} = \frac{\partial^{|S|} f}{\prod_{i \in S} \partial x_{i,j_i}}$$

be the partial derivative of f with respect to m. Define

$$\partial_S(f) = \{\, \frac{\partial f}{\partial m} \mid m \in M(S) \,\}$$

Our complexity measure is

$$\text{rank}_S(f) = \dim\left(\text{span}\left(\partial_S(f)\right)\right)$$

Example 13 *Let X be a 4×4 matrix, $f(X) = PERM(X)$, $m = x_{1,1} \cdot x_{3,2} \in P(\{1,3\})$. We have that*

$$\frac{\partial f}{\partial m} = x_{2,3}x_{4,4} + x_{2,4}x_{4,3}$$

The next lemma demonstrates the usefulness of our complexity measure.

Lemma 14 *Let $\mathcal{C}$ be a pure circuit computing a multilinear polynomial $f \in P([d])$. Let v be a node in the partition tree of $\mathcal{C}$, and let $S \subset [d]$ be the set labeling v. Then the number of multiplication gates in $\mathcal{C}$, that compute polynomials in $P(S)$, is at least* $\text{rank}_S(f)$.

Proof: Denote with $M_1, ..., M_t$ the different multiplication gates in $\mathcal{C}$ that compute polynomials in $P(S)$. As $\mathcal{C}$ is a pure circuit, there exist $g_1, \ldots, g_t \in P([d] \setminus S)$ such that

$$f = \sum_{i=1}^{t} M_i \cdot g_i$$

As any $m \in M(S)$ and any g_i, $1 \leq i \leq t$, are defined on a different set of variables, it is obvious that

$$\frac{\partial f}{\partial m} = \sum_{i=1}^{t} \frac{\partial M_i}{\partial m} \cdot g_i \quad \in \text{span}\{g_1, \ldots, g_t\}$$

Hence

$$\text{rank}_S(f) \leq t$$

□

As a corollary we immediately get the lower bounds.

Proof of theorem 11: Consider the partition tree of the circuit. Obviously there is a node in the tree that is labeled with a set S, such that $\frac{n}{3} \leq |S| \leq \frac{2n}{3}$. Consider the set of partial derivatives of the permanent (or determinant) with respect to $M(S)$. It is not hard to see that

$$\text{rank}_S(DET) \ , \ \text{rank}_S(PERM) \geq \binom{n}{|S|} \geq \binom{n}{\frac{n}{3}}$$

Applying lemma 14 our result follows.