The Computational Complexity Column

by

V. Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

Starting with this issue, I am taking over the editing of this column from Jacobo Torán. First of all, I thank Jacobo for a wonderful job of running this column. He has covered a wide range of topics in it, in articles that reflect the current trends and developments in the field. My aim is to continue in the direction set by Jacobo and the previous editors.
My first column is on some results in noncommutative arithmetic circuit complexity based on automata theory.

Noncommutative Arithmetic Circuits meet Finite Automata

V. Arvind *

Abstract

Ideas and tools from automata theory have often played a significant role in computational complexity. A prominent and well-known example is the fine structure of NC^1 which was discovered using automata theory and the classification of finite monoids [11].

*Institute of Mathematical Sciences, Chennai, India arvind@imsc.res.in

In this article we discuss some recent applications of basic automata theory to noncommutative arithmetic circuit complexity. The results described here are from [6, 5, 2, 7].

1 Introduction

Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of *n noncommuting* variables. The free monoid X^* consists of all words, i.e. *monomials*, over these variables. For a field $\mathbb{F}$, let $\mathbb{F}\langle X\rangle$ denote the free noncommutative polynomial ring over $\mathbb{F}$ generated by the variables in X. Polynomials in the ring $\mathbb{F}\langle X\rangle$ are $\mathbb{F}$-linear combinations of words (or monomials) in X^*. For a given polynomial $f \in \mathbb{F}\langle X\rangle$, the set $\mathcal{M}(f) = \{m \in X^* \mid$ coefficient of m in f is nonzero$\}$ of its nonzero monomials is a finite subset of X^*.

An *arithmetic circuit* C over a field $\mathbb{F}$ and variables $x_1, x_2, \cdots, x_n$ is a directed acyclic graph (DAG) with each node of indegree zero labeled by a variable or a scalar constant from $\mathbb{F}$: the indegree 0 nodes are the input nodes of the circuit. Each internal node of the DAG is of indegree two and is labeled by either a $+$ or a $\times$ (indicating that it is a plus gate or multiply gate, respectively). A node of C is designated as the *output* gate. Each internal gate of the arithmetic circuit computes a polynomial (by adding or multiplying its input polynomials), where the polynomial computed at an input node is defined to be its label. The *polynomial computed* by the circuit is the polynomial computed at its output gate. An arithmetic circuit is a formula if the fan-out of every gate is at most one.

An arithmetic circuit computes a polynomial in the commutative ring $\mathbb{F}[X]$ when $\times$ is a commutative operation and the x_i are commuting variables. In *noncommutative* circuits, the x_i are free noncommuting variables, and each $\times$ gate in the circuit has a left and right child. The circuit then computes a polynomial in the noncommutative ring $\mathbb{F}\langle X\rangle$.

Proving *superpolynomial size lower bounds* for commutative arithmetic circuits (or commutative arithmetic formulas) that compute "explicit" polynomials such as the Permanent polynomial is an outstanding open question in computational complexity. This problem has been well-studied for over four decades (see e.g. [39, 37]). The best known superpolynomial lower bounds are for depth-3 arithmetic circuits over finite fields [22, 24]. More recent lower bound results for multilinear formulas can be found in [34].

In his seminal paper [33] Nisan first systematically studied lower bounds for *noncommutative* computation. The focus of his study was noncommutative arithmetic circuits, noncommutative arithmetic formulas and noncommutative algebraic branching programs.

Definition 1.1. [33, 35] *An Algebraic Branching Program (ABP) is a directed acyclic graph with one vertex of in-degree zero, called the* source*, and a vertex of out-degree zero, called the* sink*. The vertices of the graph are partitioned into levels numbered* $0, 1, \cdots, d$*. Edges may only go from level* i *to level* $i+1$ *for* $i \in \{0, \cdots, d-1\}$*. 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.*

Nisan's main result in [33], based on a rank argument, is that the noncommutative permanent and the noncommutative determinant polynomials, in the ring $\mathbb{F}\langle x_{11}, x_{12}, \ldots, x_{nn}\rangle$, require exponential size noncommutative algebraic branching programs. Proving superpolynomial size lower bounds for noncommutative arithmetic circuits remains an open problem. The recent paper of Hrubes, Wigderson, and Yehudayoff [26] offers interesting insights into this problem.

Polynomial Identity Testing and Interpolation

Two fundamental algorithmic problems in the area of arithmetic circuits are polynomial identity testing and polynomial interpolation (or polynomial reconstruction). We focus on these problems for polynomials (given as ABPs or circuits or by black-box access) in the noncommutative polynomial ring $\mathbb{F}\langle X\rangle$.

Given a polynomial f in $\mathbb{F}\langle X\rangle$, either as an arithmetic circuit or by black-box access, the *Polynomial Identity Testing* problem is to determine whether f is identically zero. By *black-box* access to a polynomial f we mean that to each variable x_i we can assign a $k \times k$ matrix M_i over the field $\mathbb{F}$ (or an extension of $\mathbb{F}$) and query the black-box for the $k \times k$ matrix $f(M_1, M_2, \ldots, M_n)$. We assume that we can do this evaluation for any choice of matrices and for any dimension k (of course k will be a factor in the running time).

We discuss why this notion of black-box access is appropriate. Now, we can evaluate a polynomial $f \in \mathbb{F}\langle X\rangle$ over different domains to get information about the polynomial (or the domain). For instance, consider the "commutator" polynomial $xy - yx \in \mathbb{F}\langle x, y\rangle$. If we evaluate this over a commutative domain (like the field $\mathbb{F}$) it is always zero. On the other hand, it is nonzero in the domain of all 2×2 matrices.

A natural question is, which noncommutative polynomials vanish on all $k \times k$ matrices for a given k? The Amitsur-Levitzki theorem [3], a celebrated result in algebra answers this question.

Theorem 1.2 (Amitsur-Levitzki)**.** [3, 19] *No nonzero polynomial in $\mathbb{F}\langle X\rangle$ of degree less than $2k$ vanishes on all $k\times k$ matrices. Furthermore, there is (essentially) a unique degree $2k$ polynomial that vanishes on all $k \times k$ matrices.*

The Amitsur-Levitzki theorem justifies our black-box access model which allows evaluation of f on $k \times k$ matrices for any k.
Bogdanov and Wee [12] give a randomized algorithm whose running time is polynomial in n and $\deg(f)$ for testing if $f \in \mathbb{F}\langle X\rangle$ is identically zero. Their algorithm is a direct application of the Amitsur-Levitzki theorem. For, by the Amitsur-Levitzki theorem it follows that a nonzero degree d polynomial f cannot vanish on $k \times k$ matrices over $\mathbb{F}$ for $k > d/2$. The Schwartz-Zippel lemma can now be applied to argue that f will not vanish on a random assignment of $k \times k$ matrices (over $\mathbb{F}$ or a suitable extension field) to the variables x_i.
Raz and Shpilka [35] give a deterministic polynomial-time algorithm when the noncommutative polynomial is input as an algebraic branching program. Since noncommutative formulas can be efficiently transformed into algebraic branching programs (e.g. see [33]), this also yields a deterministic polynomial-time identity testing algorithm for noncommutative formulas.

Given black-box access to a polynomial $f \in \mathbb{F}\langle X\rangle$, the *Polynomial Interpolation* problem is to efficiently reconstruct the polynomial f. If f is a sparse polynomial the reconstruction algorithm can output f explicitly. Otherwise, if f has a small formula, ABP or arithmetic circuit, the algorithm can be required to output such a representation for f. Interpolation is essentially like exact learning with membership queries (as in Angluin's model of exact learning [4]).

2 Enter Automata

For a polynomial $f \in \mathbb{F}\langle X\rangle$ its monomial set $\mathcal{M}(f)$ is a set of words in X^*. A new idea introduced in [6] is to design finite automata that allow us to distinguish between different words in $\mathcal{M}(f)$ and, using the connection between automata, monoids and matrix rings to evaluate the polynomial f at suitably chosen matrices. This will allow us to design new polynomial identity testing and interpolation algorithms.
We recall some automata theory (see, for example, [25]). Fix a finite automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ which takes inputs in Σ^*, Σ is the alphabet, Q is the set of states, $\delta : Q \times \Sigma \to Q$ is the transition function, and q_0 and q_f are the initial and final states respectively (we only consider automata with unique accepting states). For each letter $b \in \Sigma$, let $\delta_b : Q \to Q$ be defined by: $\delta_b(q) = \delta(q, b)$. These

functions generate a submonoid of the monoid of all functions from Q to Q. This is the transition monoid of the automaton A [38, page 55]. For each $b \in \Sigma$ define the following matrix $M_b \in \mathbb{F}^{|Q|\times|Q|}$:

$$M_b(q,q') = \begin{cases} 1 & \text{if } \delta_b(q) = q', \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, M_b is the adjacency matrix of the graph of δ_b. As M_b is a 0-1 matrix, we can consider it as a matrix over any field $\mathbb{F}$.

Now, for a string $w = w_1 w_2 \cdots w_k \in \Sigma^*$ we define the matrix M_w in $\mathbb{F}^{|Q|\times|Q|}$ to be the matrix product $M_{w_1} M_{w_2} \cdots M_{w_k}$. If w is the empty string, define M_w to be the identity matrix of dimension $|Q| \times |Q|$. Let δ_w denote the natural extension of the transition function to w; if w is the empty string, δ_w is simply the identity function. We have

$$M_w(q,q') = \begin{cases} 1 & \text{if } \delta_w(q) = q', \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Thus, M_w is also a matrix of zeros and ones for any string w. Also, $M_w(q_0, q_f) = 1$ if and only if w is accepted by the automaton A.

Running Automata on Noncommutative Polynomials

Let $f \in \mathbb{F}\langle X\rangle$ be a polynomial of degree bounded by d, where f is given by black-box access. We can consider monomials in variables $X = \{x_1, \cdots, x_n\}$ as strings over the alphabet $\{x_1, x_2, \cdots, x_n\}$.

Let $A = (Q, X, \delta, q_0, q_f)$ be a finite automaton over the alphabet X. We have matrices $M_{x_i} \in \mathbb{F}^{|Q|\times|Q|}$ as defined above. We are interested in the output matrix obtained when each input x_i to the polynomial f is replaced by the matrix M_{x_i}. The output matrix of f on automaton A, denoted M_{out}, is the matrix $f(M_{x_1}, \ldots, M_{x_n})$.

If f is given by a circuit C (instead of black-box access), the circuit inputs are M_{x_i} and M_{out} is the circuit output. Clearly, given the polynomial f and automaton A, the matrix M_{out} can be computed in time polynomial in n, size(A) and size(C) (if f given by a circuit C).

The matrix output M_{out} of C on A is determined completely by the polynomial f computed by C; the structure of the circuit C is otherwise irrelevant. In particular, the output is always 0 when $f \equiv 0$.

Suppose $f(x_1, \cdots, x_n) = c x_{j_1} \cdots x_{j_k}$, with a non-zero coefficient $c \in \mathbb{F}$. Clearly, $M_{out} = c M_{x_{j_1}} \cdots M_{x_{j_k}} = c M_w$, where $w = x_{j_1} \cdots x_{j_k}$. Thus, by Equation 1 above, the matrix entry $M_{out}(q_0, q_f)$ is 0 when A rejects w, and c when A accepts w. Now, suppose $f = \sum_{i=1}^{t} c_i m_i$ with t nonzero monomials, where $c_i \in \mathbb{F} \setminus \{0\}$ and

$m_i = \prod_{j=1}^{d_i} x_{i_j}$, where $d_i \leq d$. Let w_i denotes the string representing monomial m_i. Finally, let $S_A^f = \{i \in \{1, \cdots, t\} \mid A \text{ accepts } w_i\}$. Then, by linearity, we have

Theorem 2.1. [6] *For any polynomial $f \in \mathbb{F}\langle X\rangle$ and any finite automaton $A = (Q, X, \delta, q_0, q_f)$, the output M_{out} matrix of f on A is such that $M_{out}(q_0, q_f) = \sum_{i \in S_A^f} c_i$.*

Now, suppose A is an automaton that accepts exactly one monomial among all the nonzero monomials of f. Then $M_{out}(q_0, q_f)$ is the nonzero coefficient of the unique monomial accepted by A. More precisely, we have the following corollary.

Corollary 2.2. [6] *Given any arithmetic circuit C computing polynomial $f \in \mathbb{F}\langle X\rangle$ and any finite automaton $A = (Q, X, \delta, q_0, q_f)$, then the output M_{out} of C on A satisfies:*

(1) If A rejects every string corresponding to a monomial in f, then $M_{out}(q_0, q_f) = 0$.

(2) If A accepts exactly one string corresponding to a monomial in f, then $M_{out}(q_0, q_f)$ is the nonzero coefficient of that monomial in f.

Moreover, M_{out} can be computed in time poly$(|C|, |A|, n)$.

Another important corollary is the following.

Corollary 2.3. [6] *Given any noncommutative arithmetic circuit C computing a polynomial in $\mathbb{F}\langle X\rangle$, and any monomial m of degree d_m, we can compute the coefficient of m in C in time poly$(|C|, d_m, n)$.*

Proof. Apply Corollary 2.2 with A being any standard automaton that accepts the string corresponding to monomial m and rejects every other string. Clearly, A can be chosen so that A has a unique accepting state and $|A| = O(nd_m)$. ∎

Corollary 2.3 is unlikely to hold in the commutative ring $\mathbb{F}[x_1, \cdots, x_n]$. For, in the commutative case, the coefficient of the monomial $x_1 \cdots x_n$ in the polynomial $\prod_{i=1}^{n}(\Sigma_{j=1}^{n} A_{ij}x_j)$ is the permanent of A. Hence, computing the coefficient of $x_1 \cdots x_n$ in this polynomial is #P-complete when $\mathbb{F} = \mathbb{Q}$.

Remark 2.4. *Corollary 2.2 also yields an automata-theoretic proof of a weaker form of the Amitsur-Levitzki theorem (Theorem 1.2). It implies that all $d \times d$ matrices over the field $\mathbb{F}$ do not satisfy any nontrivial identity of degree $< d$: Suppose $f = \sum_{i=1}^{t} c_i m_i \in \mathbb{F}\langle X\rangle$ is a nonzero polynomial of degree $< d$. Clearly, we can construct an automaton B with at most d states over the alphabet X that accepts exactly one string, namely one nonzero monomial, say m_1, of f and rejects all the other strings over X. By Corollary 2.2, the output matrix M_{out} of the polynomial f on B is non-zero which proves the claim.*

3 A noncommutative polynomial identity test

Building on the observations in the previous section, we now describe a new polynomial identity test for noncommutative circuits based on finite automata and the Mulmuley-Mulmuley-Vazirani Isolation Lemma [32].

Theorem 3.1. [5] *Let $f \in \mathbb{F}\{x_1, x_2, \cdots, x_n\}$ be a polynomial given by an arithmetic circuit C of size m. Let d be an upper bound on the degree of f. Then there is a randomized algorithm which runs in time poly(n, m, d) and can test whether $f \equiv 0$.*

Proof. Let $[d] = \{1, 2, \cdots, d\}$ and $[n] = \{1, 2, \cdots, n\}$. Consider the set of tuples $U = [d] \times [n]$. Let $v = x_{i_1} x_{i_2} \cdots x_{i_t}$ be a nonzero monomial of f. Then the monomial can be identified with the following subset S_v of U :

$$S_v = \{(1, i_1), (2, i_2), \cdots, (t, i_t)\}$$

Let $\mathcal{F}$ denotes the family of subsets of U corresponding to the nonzero monomials of f i.e,

$$\mathcal{F} = \{S_v \mid v \text{ is a nonzero monomial in } f\}$$

By the Isolation Lemma we know that if we assign random weights from $[2dn]$ to the elements of U, with probability at least $1/2$, there is a unique minimum weight set in $\mathcal{F}$. Our aim will be to construct a family of small size automata which are indexed by weights $w \in [2nd^2]$ and $t \in [d]$, such that the automaton $A_{w,t}$ will precisely accept all the strings (corresponding to the monomials) v of length t, such that the weight of S_v is w. Then from the Isolation Lemma we will argue that the automaton corresponding to the minimum weight will precisely accept only one string (monomial). Now for $w \in [2nd^2]$, and $t \in [d]$, we describe the construction of the automaton $A_{w,t} = (Q, \Sigma, \delta, q_0, F)$ as follows: $Q = [d] \times [2nd^2] \cup \{(0, 0)\}$, $\Sigma = \{x_1, x_2, \cdots, x_n\}$, $q_0 = \{(0, 0)\}$ and $F = \{(t, w)\}$. We define the transition function $\delta : Q \times \Sigma \to Q$,

$$\delta((i, V), x_j) = (i + 1, V + W),$$

where W is the random weight assign to $(i + 1, j)$. Our automata family $\mathcal{A}$ is simply,

$$\mathcal{A} = \{A_{w,t} \mid w \in [2nd^2], t \in [d]\}.$$

For each of the automata $A_{w,t} \in \mathcal{A}$, we mimic the run of the automaton $A_{w,t}$ on the circuit C as described in Section 2. If the output matrix corresponding to any of the automaton is nonzero, our algorithm declares $f \neq 0$, otherwise declares $f \equiv 0$.

The correctness of the algorithm follows easily from the Isolation Lemma. By the Isolation Lemma we know, on random assignment, a unique set S in $\mathcal{F}$ gets the minimum weight $w_{\min}$ with probability at least $1/2$. Let S corresponds to the monomial $x_{i_1}x_{i_2}\cdots x_{i_\ell}$. Then the automaton $A_{w_{\min},\ell}$ accepts the string (monomial) $x_{i_1}x_{i_2}\cdots x_{i_\ell}$. Furthermore, as no other set in $\mathcal{F}$ get the same minimum weight, $A_{w_{min},\ell}$ rejects all the other monomials. So the (q_0, q_f) entry of the output matrix M_{out}, that we get in running $A_{w_{\min},\ell}$ on C is nonzero. Hence with probability at least $1/2$, our algorithm correctly decide that f is nonzero. The success probability can be boosted to any constant by standard independent repetition of the same algorithm. Finally, it is trivial to see that the algorithm always decides correctly if $f \equiv 0$. ■

4 Sparse Noncommutative Polynomials

Definition 4.1. *Let $W \subset X^*$ be a finite subset of monomials and $\mathcal{A}$ be a finite collection of deterministic finite automata over alphabet X. We say that $\mathcal{A}$ is* isolating *for W if there exists a string $w \in W$ and an automaton $A \in \mathcal{A}$ such that A accepts w and rejects all $w' \in W \setminus \{w\}$.*
We say that $\mathcal{A}$ is an (m, s)-isolating family *if $\mathcal{A}$ is isolating for every subset $W = \{w_1, \cdots, w_s\}$ of s many strings in X^*, each of length at most m.*

Fix parameters $m, s \in \mathbb{N}$. We construct an (m, s) isolating family of automata $\mathcal{A}$, where $|\mathcal{A}|$ and the size of each automaton in $\mathcal{A}$ is polynomially bounded in m and s. Then, applying Corollary 2.2, we get deterministic identity testing and interpolation algorithms in the sequel.
It is convenient to encode the variables x_i in the alphabet $\{0, 1\}$. We do this by encoding the variable x_i by the string $v_i = 01^i0$, which is unary encoding with delimiters. For a string $w = x_{i_1} \ldots x_{i_k} \in X^*$, let n_w denote the positive integer represented by the binary numeral $1v_{i_1} \ldots v_{i_k}$. For each prime p and each integer $i \in \{0, \cdots, p-1\}$, we can easily construct an automaton $A_{p,i}$ that accepts exactly those w such that $n_w \equiv i \pmod{p}$. Moreover, $A_{p,i}$ can be constructed so as to have polynomially many states and with exactly one final state.
Our collection of automata $\mathcal{A}$ is just the set of $A_{p,i}$ where p runs over the first few polynomially many primes, and $i \in \{0, \cdots, p-1\}$. Formally, let N denote $(m+2)\binom{s}{2} + 1$; $\mathcal{A}$ is the collection of $A_{p,i}$, where p runs over the first N primes and $i \in \{0, \cdots, p-1\}$. Notice that, by the prime number theorem, all the primes chosen above are bounded in value by N^2, which is clearly polynomial in m and s. Hence, $|\mathcal{A}| = \text{poly}(m, s)$, and each $A \in \mathcal{A}$ is bounded in size by $\text{poly}(m, s)$.

Lemma 4.2. *The family of finite automata $\mathcal{A}$ defined as above is an (m, s)-isolating automata family.*

Proof. Consider any set of s binary strings W of length at most m each. By the construction of $\mathcal{A}$, $A_{p,i} \in \mathcal{A}$ isolates W if and only if p does not divide $n_{w_j} - n_{w_k}$ for some j and all $k \neq j$, and $n_{w_j} \equiv i \pmod{p}$. Clearly, if p satisfies the first of these conditions, i can easily be chosen so that the second condition is satisfied. We will show that there is some prime among the first N primes that does not divide $P = \prod_{j \neq k}(n_{w_j} - n_{w_k})$. This easily follows from the fact that the number of distinct prime divisors of P is at most $\log |P|$, which is clearly bounded by $(m+2)\binom{s}{2} = N - 1$. ∎

Clearly, this (m, s)-isolating family $\mathcal{A}$ can be constructed in time poly(m, s).
We first describe the deterministic identity test for sparse polynomials. Let the polynomial $f \in \mathbb{F}\langle X \rangle$ be given by black-box access. Let t be an upper bound on the number of monomials in f, and d be an upper bound on the degree of f. Using the above construction, we first compute a family $\mathcal{A}$ of automata such that $\mathcal{A}$ is isolating for any set W with at most t strings. Now, for each $A \in \mathcal{A}$, the algorithm computes the output M_{out} of the polynomial f on automaton A. If the output matrix is such that $M_{out}(q_0, q_f) \neq 0$ for some automaton A, the algorithm concludes that f is not identically zero; otherwise, the algorithm concludes that f is identically zero.
The correctness follows from Corollary 2.2. The matrices M_{x_i} for each A (all of which are of size poly(d, n, t)) can be constructed in polynomial time. Hence, the entire algorithm runs in time poly(d, n, t). We have proved the following theorem.

Theorem 4.3. [6] *Given as input a black-box noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d with at most t monomials, we can check, in time poly(d, n, t), if f is identically zero. In particular, if f is polynomially sparse and of polynomial degree, then we have a deterministic polynomial-time identity testing algorithm.*

We can easily generalize the above identity test to a deterministic interpolation algorithm that, given black-box access to a sparse noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$ makes matrix-valued queries to f and reconstructs the entire polynomial. The idea is to find all the nonzero monomials by a prefix search using the (m, s)-isolating automata family. Let $\mathcal{A}$ denote the (m, s)-isolating automata family $\{A_{p,i}\}$ as constructed before. For each $A \in \mathcal{A}$ let $[A]_w$ denote the automaton that accepts those strings that are accepted by A and in addition, contain w as a prefix. Let $[\mathcal{A}]_w$ denote the collection of all such $[A]_w$ for $A \in \mathcal{A}$. Clearly, w is a prefix of a nonzero monomial in f if and only if for some automaton in $[A]_w \in [\mathcal{A}]_w$ the output of $[A]_w$ on f is nonzero. Using this property we can easily design a

polynomial-time prefix search to find all the t monomials of f along with their coefficients in time polynomial in n, d and t. We have sketched the proof of the following:

Theorem 4.4. [6] *Given as input a black-box noncommutative polynomial $f \in \mathbb{F}\langle X\rangle$ of degree at most d and with at most t monomials, we can compute all the monomials of f, and their coefficients, in deterministic time poly(d, n, t). In particular, if f is a polynomially sparse polynomial of polynomial degree, then f can be reconstructed in deterministic polynomial time.*

5 Interpolation of Algebraic Branching Programs

Let $f \in \mathbb{F}\langle X\rangle$ be a noncommutative polynomial given by black-box access such that f has an ABP of size s. The *interpolation problem* we consider in this section is to compute an ABP for the polynomial f in time polynomial in n and s. Now, if we have an algorithm for this interpolation problem we can compute an ABP P for f in time polynomial in n and s. We can then invoke the Raz-Shpilka deterministic identity test to check if $P \equiv 0$ in polynomial time. Hence, polynomial identity testing for black-box noncommutative ABPs is deterministic polynomial-time reducible to the interpolation problem.

This interpolation problem is closely related to exact learning of DFAs in Angluin's model (of membership and equivalence queries) [4], and its generalization by Beimel at al [10] to exact learning of multiplicity automata in polynomial time. Since noncommutative ABPs are easily seen to be a restricted form of multiplicity automata, it follows from [10]'s results that ABPs can be exactly learnt in polynomial time with membership and equivalence queries. More precisely, let $f = \sum_{w\in A} f_w w$ be noncommutative polynomial of degree d in $\mathbb{F}\{x_1, x_2, \ldots, x_n\}$, where $f_w \in \mathbb{F}$. In the model of [10], a membership query is a monomial w and the teacher returns its coefficient f_w in the polynomial f. An equivalence query made by the learning algorithm is a hypothesis ABP P computing a polynomial $h = \sum_{w\in A} h_w w$, and if $h \neq f$ the teacher gives a counterexample monomial m such that $h_m \neq f_m$. The algorithm of [10] will output a multiplicity automaton (which actually gives an ABP).

It turns out that we can combine the algorithm of [10] with results explained in Section 4 to give a randomized polynomial-time algorithm for interpolating black-box ABPs. This is because we can simulate a membership query with black-box access to the ABP in deterministic polynomial time, and we can simulate an equivalence query with black-box access to the ABP in randomized polynomial time.

Theorem 5.1. [6] *The interpolation problem for black-box ABPs has a randomized polynomial-time algorithm.*

Proof. It suffices to show how membership and equivalence queries can be simulated efficiently with black-box access to the polynomial f which has an ABP of size s. Then the entire exact learning algorithm of [10] can be simulated in polynomial time with black-box access to f. Given a monomial m as a membership query, by Corollary 2.3 we can compute its coefficient f_m in the polynomial f in deterministic polynomial time. Hence, membership queries are easy to simulate.
Now consider an equivalence query where the hypothesis polynomial h is given by an ABP P. We need to test if P computes the polynomial f and, if not, find a monomial m such that $h_m \neq f_m$. Testing if $h \neq f$ is reducible to polynomial identity testing for black-box ABPs which can be done in randomized polynomial time by substituting randomly picked $s \times s$ matrices over $\mathbb{F}$ for the variables x_i using the algorithm in [12].
With black-box access to f, we can compute a monomial m such that $h_m \neq f_m$ in randomized polynomial time. We apply ideas from the noncommutative polynomial identity test of Theorem 3.1. Suppose f is of degree d (which is bounded above by the ABP size s). For $1 \leq i \leq n$ and $1 \leq j \leq d$ we pick independent random weights $w_{ij} \in [2dn]$, where w_{ij} is the weight assigned to variable x_i when it occurs in position j of a monomial. The weight of a monomial is defined as the sum of weights of the variables occurring in it. By the isolation lemma [32], if $h \neq f$ then with probability at least $1/2$ there is a *unique* minimum weight monomial m such that $h_m \neq f_m$. We can design a DFA $A_{w,i,j}$ of size polynomial in n and d such that $A_{w,i,j}$ accepts a monomial m iff m has weight w and x_i occurs in the j^{th} position. Suppose $w_0 \in [2d^2n]$ is the weight of the unique minimum weight monomial. By Theorem 2.1 we can evaluate $f - h$, using the black-box for f, on each automaton $A_{w,i,j}$ for $w \in [2d^2n]$, $1 \leq i \leq n$, and $1 \leq j \leq d$. Conditioned on the event that there is a unique monomial m of weight w_0, for the choice $w = w_0$, the evaluation of $f - h$ on $A_{w_0,i,j}$ will output $f_m - h_m$ if x_i is the j^{th} variable in m and 0 otherwise. Hence, we can compute m in randomized polynomial time, implying that equivalence queries can be simulated in randomized polynomial time. ■

It is an open question if interpolation (or even polynomial identity testing) of black-box ABPs can be done in deterministic polynomial time. Suppose we assume a *stronger* black-box access in which the polynomial f is given by an unknown ABP P, and the interpolating algorithm is allowed to evaluate P at any specific gate for a matrix-valued input. In this case there is a deterministic polynomial time interpolation algorithm [6].

6 Hadamard Product of Polynomials

We know that the intersection of two regular languages is regular. Is there an analogue of the "intersection" of two noncommutative polynomials $f, g \in \mathbb{F}\langle X\rangle$? We define the Hadamard product of polynomials which turns out to be a useful operation.

Definition 6.1. *Let $f, g \in \mathbb{F}\langle X\rangle$ where $X = \{x_1, x_2, \cdots, x_n\}$. The* Hadamard product *of f and g, denoted $f \circ g$, is the polynomial $f \circ g = \sum_m a_m b_m m$, where $f = \sum_m a_m m$ and $g = \sum_m b_m m$, where the sums index over monomials m.*

Analogous to regular languages being closed under intersection, the *noncommutative* branching program complexity of the Hadamard product $f \circ g$ is upper bounded by the product of the branching program sizes for f and g. There is an NC^2 algorithm (in fact a logspace algorithm) for computing an ABP for $f \circ g$.
We apply this to polynomial identity testing. Raz and Shpilka [35] have shown that polynomial identity testing of noncommutative ABPs is in deterministic polynomial time. A simple divide and conquer yields a deterministic NC^3 algorithm. What then is the precise complexity of the problem? For noncommutative ABPs over *rationals*, using the hadamard product we can put it in NC^2. In fact, the problem is logspace equivalent to the problem of testing if a rational square matrix is singular [2].
Let $f, g \in \mathbb{F}\langle X\rangle$ where $X = \{x_1, x_2, \cdots, x_n\}$. Clearly, $\mathcal{M}(f \circ g) = \mathcal{M}(f) \cap \mathcal{M}(g)$. Our definition is also motivated by the Hadamard product $A \circ B$ of two $m \times n$ matrices A and B. We recall the following bound for the rank of the Hadamard product.

Proposition 6.2. *Let A and B be $m \times n$ matrices over a field $\mathbb{F}$. Then* $\mathrm{rank}(A \circ B) \leq \mathrm{rank}(A)\,\mathrm{rank}(B)$.

It is known [33] that the ABP complexity $B(f)$ of a polynomial $f \in \mathbb{F}\langle X\rangle$ is closely connected with the ranks of the communication matrices $M_k(f)$, where $M_k(f)$ has its rows indexed by degree k monomials and columns by degree $d-k$ monomials and the $(m, m')^{th}$ entry of $M_k(f)$ is the coefficient of mm' in f. Nisan showed that $B(f) = \sum_k \mathrm{rank}(M_k(f))$. Indeed, Nisan's result and the above proposition easily imply the following bound on the ABP complexity of $f \circ g$.

Lemma 6.3. *For $f, g \in \mathbb{F}\langle X\rangle$ we have $B(f \circ g) \leq B(f)B(g)$.*

Proof. By Nisan's result $B(f \circ g) = \sum_k \mathrm{rank}(M_k(f \circ g))$. The above proposition

implies

$$\begin{aligned}\sum_k \operatorname{rank}(M_k(f \circ g)) &\leq \sum_k \operatorname{rank}(M_k(f)) \operatorname{rank}(M_k(g)) \\ &\leq (\sum_k \operatorname{rank}(M_k(f)))(\sum_k \operatorname{rank}(M_k(g))).\end{aligned}$$

■

We now show an algorithmic version of this upper bound.

Theorem 6.4. [2] *Let P and Q be two given ABP's computing polynomials f and g in $\mathbb{F}\langle x_1, x_2, \ldots, x_n\rangle$, respectively. Then there is a deterministic logspace (hence NC^2) algorithm that will output an ABP R for the polynomial $f \circ g$ such that the size of R is a constant multiple of the product of the sizes of P and Q.*

Proof. Let f_i and g_i denote the i^{th} homogeneous parts of f and g respectively. Then $f = \sum_{i=0}^{d} f_i$ and $g = \sum_{i=0}^{d} g_i$. Since the Hadamard product is distributive over addition and $f_i \circ g_j = 0$ for $i \neq j$ we have $f \circ g = \sum_{i=0}^{d} f_i \circ g_i$. Thus, we can assume that both P and Q are homogeneous ABP's of degree d. Otherwise, we can easily construct an ABP to compute $f_i \circ g_i$ separately for each i and put them together. Note that we can easily compute ABPs for f_i and g_i in logspace given as input the ABPs for f and g.

By allowing parallel edges between nodes of P and Q we can assume that the labels associated with each edge in an ABP is either 0 or αx_i for some variable x_i and scalar $\alpha \in \mathbb{F}$. Let s_1 and s_2 bound the number of nodes in each layer of P and Q respectively. Denote the j^{th} node in layer i by $\langle i, j\rangle$ for ABPs P and Q. Now we describe the construction of the ABP R for computing the polynomial $f \circ g$. Each layer i, $1 \leq i \leq d$ of R will have $s_1 \cdot s_2$ nodes, with node labeled $\langle i, a, b\rangle$ corresponding to the node $\langle i, a\rangle$ of P and the node $\langle i, b\rangle$ of Q. We can assume there is an edge from every node in layer i to every node in layer $i + 1$ for both ABPs. For, if there is no such edge we can always include it with label 0.

In the new ABP R we put an edge from $\langle i, a, b\rangle$ to $\langle i + 1, c, e\rangle$ with label $\alpha\beta x_t$ if and only if there is an edge from node $\langle i, a\rangle$ to $\langle i + 1, c\rangle$ with label αx_t in P and an edge from $\langle i, b\rangle$ to $\langle i + 1, e\rangle$ with label βx_t in ABP Q. Let $\langle 0, a, b\rangle$ and $\langle d, c, e\rangle$ denote the source and the sink nodes of ABP R, where $\langle 0, a\rangle, \langle 0, b\rangle$ are the source nodes of P and Q, and $\langle d, c\rangle, \langle d, e\rangle$ are the sink nodes of P and Q respectively. It is easy to see that ABP R can be computed in deterministic logspace. Let $h_{\langle i,a,b\rangle}$ denote the polynomial computed at node $\langle i, a, b\rangle$ of ABP R. Similarly, let $f_{\langle i,a\rangle}$ and $g_{\langle i,b\rangle}$ denote the polynomials computed at node $\langle i, a\rangle$ of P and node $\langle i, b\rangle$ of Q. We can easily check that $h_{\langle i,a,b\rangle} = f_{\langle i,a\rangle} \circ g_{\langle i,b\rangle}$ by an induction argument on the number

of layers in the ABPs. It follows from this inductive argument that the ABP R computes the polynomial $f \circ g$ at its sink node. The bound on the size of R also follows easily. ■

Applying the above theorem we can improve the NC^3 upper bound for identity testing of noncommutative ABPs over rationals.

Theorem 6.5. [2] *The problem of polynomial identity testing for noncommutative algebraic branching programs over* $\mathbb{Q}$ *is in* NC^2.

Proof. Let P be the given ABP computing $f \in \mathbb{Q}\langle X\rangle$. We apply the construction of Theorem 6.4 to compute a polynomial sized ABP R for the Hadamard product $f \circ f$ (i.e. of f with itself). Notice that $f \circ f$ is nonzero iff f is nonzero. Now, we crucially use the fact that $f \circ f$ is a polynomial whose nonzero coefficients are all *positive*. Hence, $f \circ f$ is nonzero iff it evaluates to nonzero on the all 1's input. The problem thus boils down to checking if R evaluates to nonzero on the all 1's input.

By Theorem 6.4, the ABP R for polynomial $f \circ f$ is computable in deterministic logspace, given as input an ABP for f. Furthermore, evaluating the ABP R on the all 1's input can be easily converted to iterated integer matrix multiplication (one matrix for each layer of the ABP), and checking if R evaluates to nonzero can be done by checking if a specific entry of the product matrix is nonzero. It is well known that computing the iterated integer matrix product is in NC^2 which completes the proof.

With a more careful argument we can show that the problem is, in fact, logspace *equivalent* to testing if a rational matrix is singular. ■

Can we give a similar tight complexity characterization for identity testing of noncommutative ABPs over finite fields?

We note that identity testing noncommutative ABPs over any field is hard for NL by a reduction from directed graph reachability. Let (G, s, t) be a reachability instance. Without loss of generality, we assume that G is a layered directed acyclic graph. The graph G defines an ABP with source s and sink t as follows: label each edge e in G with a *distinct* variable x_e and for each absent edge put the label 0. The polynomial computed by the ABP is nonzero if and only if there is a directed s-t path in G.

Theorem 6.6. [2] *The problem of polynomial identity testing for noncommutative algebraic branching programs over any field is hard for* NL.

7 The Noncommutative Determinant

We consider polynomials over an arbitrary field $\mathbb{F}$ (for the algorithmic results $\mathbb{F}$ is either rational numbers or a finite field). The main result of this section is that if there is a polynomial-time algorithm to compute the $2n \times 2n$ Cayley determinant over inputs from $M_S(\mathbb{F})$ for $S = c \cdot n^2$ (for a suitable constant c) then there is a polynomial-time algorithm to compute the $n \times n$ permanent over $\mathbb{F}$.

Throughout this section let X denote $\left\{x_{ij} \,\middle|\, 1 \leq i, j \leq 2n\right\}$, and Y denote $\left\{y_{ij} \,\middle|\, 1 \leq i, j \leq n\right\}$. Our aim is to show that if there is a polynomial-time algorithm for computing $\mathrm{Cdet}_{2n}(X)$ where x_{ij} takes values in $M_S(\mathbb{F})$ then there is a polynomial-time algorithm that computes $\mathrm{Cperm}_n(Y)$ where y_{ij} takes values in $\mathbb{F}$.

The $2n \times 2n$ determinant has $2n!$ many signed monomials of degree $2n$ of the form $x_{1,\sigma(1)}x_{2,\sigma(2)} \cdots x_{2n,\sigma(2n)}$ for $\sigma \in S_{2n}$. We will identify $n!$ of these monomials, all of which have the same sign. More precisely, we will design a small ABP with which we will be able to pick out these $n!$ monomials of the same sign.

We now define these $n!$ many permutations from S_{2n} which have the same sign and the corresponding monomials of Cdet_{2n} that can be picked out by a small ABP.

Definition 7.1. *Let $n \in \mathbb{N}$. For each permutation $\pi \in S_n$, we define a permutation $\rho(\pi)$ in S_{2n}, called the* interleaving *of π, as follows:*

$$\rho(\pi)(i) = \begin{cases} \pi(\frac{i+1}{2}), & \text{if } i \text{ is odd,} \\ n + \pi(\frac{i}{2}), & \text{if } i \text{ is even.} \end{cases}$$

That is, the elements $\rho(\pi)(1), \rho(\pi)(2), \cdots, \rho(\pi)(2n)$ are simply $\pi(1), (n + \pi(1)), \pi(2), (n + \pi(2)), \cdots, \pi(n), (n + \pi(n))$.

Lemma 7.2. *The sign of the permutation $\rho(\pi)$ is independent of π. More precisely, for every $\pi \in S_n$, we have $\mathrm{sgn}(\rho(\pi)) = \mathrm{sgn}(\rho(1_n))$, where 1_n denotes the identity permutation in S_n.*

Proof. For each $\pi \in S_n$ we can define the permutation $\pi_2 \in S_{2n}$ as $\pi_2(i) = \pi(i)$ for $1 \leq i \leq n$ and $\pi_2(n + j) = n + \pi(j)$ for $1 \leq j \leq n$. It is easy to verify that $\mathrm{sgn}(\pi_2) = \mathrm{sgn}(\pi)^2 = 1$ for every $\pi \in S_n$. To see this we write π_2 as a product of disjoint cycles and notice that every cycle occurs an even number of times. Furthermore, we can check that $\rho(\pi) = \rho(1_n)\pi_2$, where we evaluate products of permutations from left to right. Hence it follows that $\mathrm{sgn}(\rho(\pi)) = \mathrm{sgn}(\rho(1_n))\mathrm{sgn}(\pi_2) = \mathrm{sgn}(\rho(1_n))$. ∎

We will denote by ρ_0 the permutation $\rho(1_n)$, where 1_n denotes the identity permutation in S_n.

For $\sigma \in S_{2n}$, we will denote by m_σ the monomial $x_{1,\sigma(1)}x_{2,\sigma(2)}\cdots x_{2n,\sigma(2n)} \in X^*$. There is an ABP that filters out all monomials not of the form $m_{\rho(\pi)}$ from among the m_σ.

Lemma 7.3. *There is an ABP P of size $O(n^2)$ and width n that computes a homogeneous polynomial $F \in \mathbb{F}\langle X\rangle$ of degree $2n$ such that for any $\sigma, \tau \in S_{2n}$, $F(m_\sigma) = 1$ if $\sigma = \rho(\pi)$ for some $\pi \in S_n$, and 0 otherwise. Moreover, the ABP P can be constructed in time poly(n).*

Proof. The ABP is essentially a finite automaton over the alphabet X with the following properties: for input monomials of the form m_σ it accepts only those monomials that are of the form $m_{\rho(\pi)}$. Further, for input monomials of the form $m_{\sigma,\tau}$ it accepts only those monomials of the form $m_{1_{2n},\tau}$. We give the formal description of this ABP P below.
The ABP P contains $2n + 1$ layers, labelled $\{0, 1, \ldots, 2n\}$. For each even $i \in \{0, 1, \ldots, 2n\}$, there is exactly one node q_i at level i; for each odd $i \in \{0, 1, \ldots, 2n\}$, there are n nodes $p_{i,1}, p_{i,2}, \ldots, p_{i,n}$ at level i. We now describe the edges of P: for each even $i \in \{0, 1, \ldots, 2n-2\}$ and $j \in [n]$, there is an edge from q_i to $p_{i+1,j}$ labelled $x_{i+1,j}$; for each odd $i \in \{0, 1, \ldots, 2n\}$ and $j \in [n]$, there is an edge from $p_{i,j}$ to q_{i+1} labelled $x_{i+1,n+j}$. ∎

Before we proceed we need one more analogy to a well-known closure property in formal language theory, namely, that the intersection of a context-free language with a regular language is context-free. We show something similar for noncommutative polynomials, where the analogue of context-free languages is noncommutative circuits.

Theorem 7.4. [2] *Let $Z = \{z_1, z_2, \ldots, z_m\}$ be a set of noncommuting variables. Given a noncommutative circuit of size S' for a polynomial $f \in \mathbb{F}\langle Z\rangle$ and an ABP of size S for a homogeneous polynomial $g \in \mathbb{F}\langle Z\rangle$, we can efficiently compute a noncommutative circuit of size poly(S', S) for $f \circ g$.*

Proof. We present the proof given in [7]. Let P be the ABP of size S, with nodes named $\{1, 2, \ldots, S\}$, computing the homogeneous polynomial $g \in \mathbb{F}\langle Z\rangle$, where the source node of P is 1 and the sink node is S. We describe a polynomial-time algorithm for constructing n $S \times S$ matrices $A_1, A_2, \ldots, A_n$ over $\mathbb{F}$ such that $f \circ g = f(A_1z_1, A_2z_2, \ldots, A_nz_n)(1, S)$.
Define the matrices $A_1, A_2, \ldots, A_n \in M_S(\mathbb{F})$ as follows: $A_i(k, l)$ is the coefficient of the variable z_i in the linear form labelling the edge that goes from vertex k to vertex l; if there is no such edge, the entry $A_i(k, l) = 0$. For any monomial

$m = z_{i_1} z_{i_2} \cdots z_{i_d} \in Z^d$, let A_m denote the matrix $A_{i_1} A_{i_2} \cdots A_{i_d}$. We see that

$$\begin{aligned} f(A_1 z_1, A_2 z_2, \ldots, A_n z_n) &= \sum_{i_1, i_2, \ldots, i_d \in [n]} f(z_{i_1} z_{i_2} \cdots z_{i_d})(A_{i_1} z_{i_1})(A_{i_2} z_{i_2}) \cdots (A_{i_d} z_{i_d}) \\ &= \sum_{i_1, i_2, \ldots, i_d \in [n]} f(z_{i_1} z_{i_2} \cdots z_{i_d})(A_{i_1} A_{i_2} \cdots A_{i_d})(z_{i_1} z_{i_2} \cdots z_{i_d}) \\ &= \sum_{m \in Z^d} f(m) A_m m. \end{aligned}$$

Note that the coefficient $g(m)$ of a monomial $m = z_{i_1} z_{i_2} \cdots z_{i_d}$ in g is just $A_m(1, S) = \sum_{k_1, k_2, \ldots, k_{d-1} \in [S]} \prod_{j=1}^{d} A_{i_j}(k_{j-1}, k_j)$, where $k_0 = 1$ and $k_d = S$. Putting the above observations together, we see that $f(A_1 z_1, A_2 z_2, \ldots, A_n z_n)(1, S) = \sum_{m \in Z^d} f(m) A_m(1, S) m = \sum_{m \in Z^d} f(m) g(m) m = f \circ g$. Since the entries of the matrices $A_1, A_2, \ldots, A_n$ can be read off from the labels of P, it can be seen that $A_1, A_2, \ldots, A_n$ can be computed in polynomial time given the ABP P.
Now, given a noncommutative circuit of size S' for $f \in \mathbb{F}\langle Z \rangle$ and an ABP of size S for $g \in \mathbb{F}\langle Z \rangle$, by applying the above construction we can efficiently compute a noncommutative circuit of size $\text{poly}(S', S)$ for $f \circ g$. ■

We are now ready to prove that if there is a small noncommutative arithmetic circuit that computes the Cayley determinant polynomial, then there is a small noncommutative arithmetic circuit that computes the Cayley permanent polynomial.

Theorem 7.5. [7] *For any $n \in \mathbb{N}$, if there is a circuit C of size s computing $\text{Cdet}_{2n}(X)$, then there is a circuit C' of size polynomial in s and n that computes $\text{Cperm}_n(Y)$.*

Proof. Assuming the existence of the circuit C as stated above, by Theorem 7.4, there is a noncommutative arithmetic circuit C'' of size $\text{poly}(s, n)$ that computes the polynomial $F'' = \text{Cdet}_{2n} \circ F$, where F is the polynomial referred to in Lemma 7.3. For any monomial m, if $m \neq m_\sigma$ for any $\sigma \in S_{2n}$, then $\text{Cdet}_{2n}(m) = 0$ and hence, in this case, $F''(m) = 0$; moreover, for $m = m_\sigma$, we have $F(m) = 1$ if $\sigma = \rho(\pi)$ for some $\pi \in S_n$, and 0 otherwise. Hence, we see that

$$F''(X) = \sum_{\pi \in S_n} \text{sgn}(\rho(\pi)) m_{\rho(\pi)} = \text{sgn}(\rho_0) \left(\sum_{\pi \in S_n} m_{\rho(\pi)} \right)$$

where the last equality follows from Lemma 7.2.
Let C' be the circuit obtained from C'' by substituting x_{ij} with $y_{\frac{1+i}{2}, j}$ if i is odd and $j \in [n]$, and by 1 if i is even or $j \notin [n]$, and by multiplying the output of the

resulting circuit by $\mathrm{sgn}(\rho_0)$. Let F' denote the polynomial computed by C'. Then, we have

$$F'(X) = \sum_{\pi \in S_n} m'_{\rho(\pi)}$$

where $m'_{\rho(\pi)}$ denotes the monomial obtained from $m_{\rho(\pi)}$ after the substitution. It can be checked that for any $\pi \in S_n$, the monomial $m'_{\rho(\pi)} = y_{1,\pi(1)} y_{2,\pi(2)} \cdots y_{n,\pi(n)}$. Hence, the polynomial F' computed by C' in indeed $\mathrm{Cperm}_n(Y)$. It is easily seen that the size of C' is $\mathrm{poly}(s, n)$. ■

8 Bibiographic Notes

Polynomial identity testing (PIT) is a well-studied algorithmic problem, both when the input polynomial f is given as a circuit and when it is given via black-box access that allows the evaluation of the polynomial f at any point in $\mathbb{F}'^n$ for a field extension $\mathbb{F}'$ of $\mathbb{F}$. The problem is in randomized polynomial time [17, 40, 36, 1], and most of these algorithms work even in the black-box setting, as long as $|\mathbb{F}|$ is suitably larger than $\deg(f)$. It is a major challenge to obtain deterministic polynomial time algorithms even for restricted versions of the problem. The results of Kabanets and Impagliazzo [27] show that the problem is as hard as proving superpolynomial circuit lower bounds. Indeed, the problem remains open even for depth-3 arithmetic circuits with an unbounded + gate as output [18, 30].

As shown by Nisan [33], it can be easier to prove lower bounds for noncommutative algebraic computation. Nisan has shown exponential size lower bounds for noncommutative formulas (and noncommutative algebraic branching programs) that compute the noncommutative permanent or determinant polynomials. Chien and Sinclair, in [15], explore the problem for other noncommutative algebras. They refine Nisan's rank argument to show exponential size lower bounds for formulas computing the permanent or determinant over the algebra of 2×2 matrices over $\mathbb{F}$ and several other examples.

Raz and Shpilka [35] have shown that for noncommutative formulas (and algebraic branching programs) there is a deterministic polynomial-time algorithm for polynomial identity testing. For noncommutative circuits [12] show using the Amitsur-Levitzki theorem that identity testing for *polynomial degree* noncommutative circuits is in randomized polynomial time. It is open whether there is a similar algorithm for unrestricted degree circuits. Sparse polynomial identity testing and polynomial interpolation problems in the setting of commuting variables (i.e. for polynomials in the *commutative* ring $\mathbb{F}[x_1, x_2, \ldots, x_n]$) have been studied over several years [21, 11, 23, 13, 29].

An important motivation for studying the noncommutative determinant is an approach to designing randomized approximation algorithms for the $0-1$ permanent by designing good unbiased estimators based on the determinant. This approach has a long history starting with [20, 28]. Of specific interest are the works of Barvinok [9], Chien, Rasmussen, and Sinclair [16], and Moore and Russell [31]. Barvinok [9] defines a variant of the noncommutative determinant called the *symmetrized determinant* and shows that given inputs from a *constant dimensional* matrix algebra, the symmetrized determinant over these inputs can be evaluated in polynomial time. He uses these to define a series of algorithms that he conjectures might yield progressively better randomized approximation algorithms for the (commutative) permanent. Chien, Rasmussen, and Sinclair [16] show that efficient algorithms to compute the determinant over Clifford algebras of polynomial dimension would yield efficient approximation algorithms for the permanent. Moore and Russell [31] provide evidence that Barvinok's approach might not work, but their results also imply that computing the symmetrized or standard noncommutative determinant over polynomial dimensional matrix algebras would give a good estimator for the permanent. The recent results in [7, 14] show that the noncommutative determinant is, for most noncommutative algebras, as hard as computing the permanent.

References

[1] M. Agrawal and S. Biswas. Primality and identity testing via Chinese remaindering. *J. ACM.*, 50(4):429-443, 2003.

[2] V. Arvind, P. S. Joglekar, S. Srinivasan. Arithmetic Circuits and the Hadamard Product of Polynomials. *Proceedings of the 29th FSTTCS Conference,* LIPIcs 4 Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik 2009, ISBN 978-3-939897-13-2.

[3] S.A Amitsur and J. Levitzki. Minimal Identities for algebras. *In Proceedings of the American Mathematical Society.,* volume 1, pages 449-463, 1950.

[4] Dana Angluin. Queries and Concept Learning. *Machine Learning* 2(4): 319-342 (1987).

[5] V. Arvind, P. Mukhopadhyay Derandomizing the Isolation Lemma and Lower Bounds for Circuit Size.*Proceedings of APPROX-RANDOM 2008,* 276-289, LNCS, Springer.

[6] V. Arvind, P. Mukhopadhyay, S. Srinivasan. New results on Noncommutative Polynomial Identity Testing*In Proc. of Annual IEEE Conference on Computational Complexity,*268-279,2008. *Computational Complexity,* Volume 19, Number 4, December 2010, pp. 521-558.

[7] V. Arvind and S. Srinivasan. On the hardness of the noncommutative determinant. *Symp. Theory of Computing 2010,* pp. 677-686.

[8] D.A.M. Barrington and D. Therién. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, Volume 35, 1988 Volume 35, Number 1, January 1988, pp. 941-952.

[9] A. Barvinok. New Permanent Estimators via Non-Commutative Determinants. preprint `http://www.math.lsa.umich.edu/~barvinok/papers.html`

[10] A. Beimel, F. Bergadano, N.H. Bshouty, E. Kushilevitz, S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM* 47(3): 506-530 (2000).

[11] M. Ben-Or and P. Tiwari. A Deterministic Algorithm For Sparse Multivariate Polynomial Interpolation. *In Proc. of the 20th annual ACM Sym. on Theory of computing.*, pages 301-309, 1988.

[12] A. Bogdanov and H. Wee More on Noncommutative Polynomial Identity Testing . *In Proc. of the 20th Annual Conference on Computational Complexity*, pp. 92-99, 2005.

[13] M. Clausen, A. W. M. Dress, J. Grabmeier, and M. Karpinski. On Zero-Testing and Interpolation of k-Sparse Multivariate Polynomials Over Finite Fields. *Theoretical Computer Science* **84**(2), 151–164.

[14] S. Chien, P. Harsha, A. Sinclair, S. Srinivasan. Almost Settling the Hardness of Noncommutative Determinant. In *Symposium Theory of Computing*, 2011, to appear.

[15] S. Chien, A. Sinclair. Algebras with polynomial identities and computing the determinant *In Proc. Annual IEEE Sym. on Foundations of Computer Science*,352-361, 2004.

[16] S. Chien, L. E. Rasmussen, A. Sinclair. Clifford algebras and approximating the permanent. *J. Comput. Syst. Sci. 67(2)*: 263-290 (2003).

[17] R. A. DeMillo and R. J. Lipton. A Probabilistic Remark on Algebraic Program Testing. *Information Processing Letters* **7**(4), 193–195.

[18] Z. Dvir and A. Shpilka. Locally Decodable Codes with 2 queries and Polynomial Identity Testing for depth 3 circuits. *In Proc. of the 37th annual ACM Sym. on Theory of computing.*, 2005.

[19] A. Giambruno and M. Zaicev. Polynomial Identities and Asymptotic Methods. *American Mathematical Society.*, Vol. 122, 2005.

[20] C. Godsil, I. Gutman. On the matching polynomial of a graph, *Algebraic Methods in Graph Theory, 1981*, pp. 241–249.

[21] D. Grigoriev and M. Karpinski The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanents Is in NC (Extended Abstract). In *Proceedings of the 20th Annual Symposium on the Foundations of Computer Science*, 166–172.

[22] D. Grigoriev and M. Karpinski. An Exponential Lower Bound for Depth 3 Arithmetic Circuits. *In Proceedings of the 30th annual ACM Sym. on Theory of computing*, 577–582, 1998.

[23] D. Grigoriev, M. Karpinski, and M. F. Singer Fast Parallel Algorithms for Sparse Multivariate Polynomial Interpolation over Finite Fields. *SIAM Journal of Computing* **19**(6), 1059–1063.

[24] D. Grigoriev and A. Razborov. Exponential Lower Bounds for Depth 3 Arithmetic Circuits in Algebras of Functions over Finite Fields. *Appl. Algebra Eng. Commun. Comput,* 10(6): 465–487, 2000.

[25] J.E. Hopcroft and J.D. Ullman Introduction to Automata Theory, Languages and Computation, *Addison-Wesley*, 1979.

[26] P. Hrubes, A. Wigderson and A. Yehudayoff. Non-commutative circuits and the sum-of-squares problem. *Symposium on Theory of Computing,* 2010: 667-676.

[27] V. Kabanets and R. Impagliazzo. Derandomization of polynomial identity tests means proving circuit lower bounds. *In Proc. of the thirty-fifth annual ACM Sym. on Theory of computing.,* pages 355-364, 2003.

[28] N. Karmarkar, R. M. Karp, R. J. Lipton, L. Lovàsz, Michael Luby. A Monte-Carlo Algorithm for Estimating the Permanent. *SIAM J. Comput.* 22(2): 284-293 (1993).

[29] A. Klivans and D. Spielman. Randomness Efficient Identity Testing. *Proceedings of the 33rd Symposium on Theory of Computing (STOC),* 216–223, 2001.

[30] N. Kayal and N. Saxena. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity.*, 16(2):115-138, 2007.

[31] C. Moore, A. Russell. Approximating the Permanent via Nonabelian Determinants, *CoRR abs/0906.1702: (2009).* `http://arxiv.org/abs/0906.1702`

[32] K. Mulmuley, U. V. Vazirani, V. V. Vazirani. Matching Is as Easy as Matrix Inversion *STOC 1987*: 345-354.

[33] N. Nisan. Lower bounds for non-commutative computation. *In Proc. of the 23rd annual ACM Sym. on Theory of computing.,* pages 410-418, 1991.

[34] R. Raz. Separation of Multilinear Circuit and Formula Size. *Theory Of Computing,* Vol. 2, article 6 (2006).

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

[36] J. T. Schwartz. Fast Probabilistic algorithm for verification of polynomial identities. *J. ACM.,* 27(4), pages 701-717, 1980.

[37] A. Shpilka and A. Yehudayoff. *Arithmetic Circuits: A survey of recent results and open questions,* Foundations and TrendsÂő in Theoretical Computer Science: Vol. 5: No 3-4, pp 207-388. `http://dx.doi.org/10.1561/0400000039`.

[38] H. Straubing. Finite automata, formal logic, and circuit complexity. *Progress in Theoretical Computer Science.* BirkhÃd'user Boston Inc., Boston, MA, 1994.

[39] A. Wigderson. Arithmetic circuit complexity: a survey. Available online at `http://www.math.ias.edu/ avi/BOOKS/arithmetic_complexity.pdf`.

[40] R. Zippel. Probabilistic algorithms for sparse polynomials. *In Proc. of the Int. Sym. on Symbolic and Algebraic Computation.*, pages 216-226, 1979.