Electronic Colloquium on Computational Complexity, Report No. 26 (2009)

Arithmetic Circuit Size, Identity Testing, and Finite Automata

V. Arvind and Pushkar S. Joglekar

Institute of Mathematical Sciences
C.I.T Campus,Chennai 600 113, India
{arvind,pushkar}@imsc.res.in

Abstract. Let $\mathbb{F}\langle x_1, x_2, \cdots, x_n\rangle$ be the noncommutative polynomial ring over a field $\mathbb{F}$, where the x_i's are free noncommuting formal variables. Given a finite automaton $\mathcal{A}$ with the x_i's as alphabet, we can define polynomials $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$ obtained by natural operations that we call *intersecting* and *quotienting* the polynomial f by $\mathcal{A}$. Related to intersection, we also define the *Hadamard product* $f \circ g$ of two polynomials f and g.

In this paper we study the circuit and algebraic branching program (ABP) complexities of the polynomials $f(\text{mod } \mathcal{A})$, $f(\text{div } \mathcal{A})$, and $f \circ g$ in terms of the corresponding complexities of f and g and size of the automaton $\mathcal{A}$. We show upper and lower bound results. Our results have consequences in new polynomial identity testing algorithms (and algorithms for its corresponding search version of finding a nonzero monomial). E.g. we show the following:

(a) A deterministic NC^2 identity test for noncommutative ABPs over rationals. In fact, we tightly classify the problem as complete for the logspace counting class $\text{C}_=\text{L}$.
(b) Randomized NC^2 algorithms for finding a nonzero monomial in both noncommutative and commutative ABPs.
(c) Over monomial algebras $\mathbb{F}\langle x_1, \cdots, x_n\rangle/I$ we derive an exponential size lower bound for ABPs computing the Permanent. We also obtain deterministic polynomial identity testing for ABPs over such algebras.

We also study analogous questions in the *commutative* case and obtain some results. E.g. we show over any commutative monomial algebra $\mathbb{Q}[x_1, \cdots, x_n]/I$ such that the ideal I is generated by $o(n/\lg n)$ monomials, the Permanent requires exponential size monotone circuits.

1 Introduction

The basic goal of this paper is to study polynomial identity testing for polynomials given by circuits or algebraic branching programs (ABPs), as well as its connection to proving circuit/ABP size lower bounds. We consider these problems mainly in the noncommutative setting, but we also have observations in the commutative setting.

The main tool we use to study circuits and ABPs is finite automata and its properties. To this end, we define the notion of *intersection* of a circuit or an ABP by a finite automaton and the notion of the *quotient* of a circuit/ABP by a finite automaton.

More precisely, suppose $X = \{x_1, x_2, \cdots, x_n\}$ is a set of n noncommuting variables. The free monoid X^* consists of all words over these variables. For a field $\mathbb{F}$ let $\mathbb{F}\langle x_1, x_2, \cdots, x_n\rangle$ denote the free noncommutative polynomial ring over $\mathbb{F}$ generated by the variables in X. Thus, the polynomials in this algebra are $\mathbb{F}$-linear combinations of words over X.

For a given polynomial $f \in \mathbb{F}\langle X\rangle$, let $\text{mon}(f) = \{m \in X^* \mid m \text{ is a nonzero monomial in } f\}$.

Definition 1. *Let $f \in \mathbb{F}\langle X\rangle$ be a polynomial and $\mathcal{A}$ be a finite automaton (deterministic or nondeterministic) accepting a subset of X^*. The* quotient *of the polynomial $f = \sum_\alpha c_\alpha m_\alpha$ by the automaton $\mathcal{A}$ is defined as the polynomial*

$$f(\textit{mod } \mathcal{A}) = \sum_{m_\alpha \in \textit{mon}(f) \setminus L(\mathcal{A})} c_\alpha m_\alpha,$$

ISSN 1433-8092

where $L(\mathcal{A})$ is the language accepted by the automaton. Similarly, the intersection *of the polynomial f by the automaton $\mathcal{A}$ is the polynomial*

$$f(\textit{div } \mathcal{A}) = \sum_{m_\alpha \in \textit{mon}(f) \cap L(\mathcal{A})} c_\alpha m_\alpha.$$

Note: The automaton $\mathcal{A}$ splits the polynomial f into two parts as $f = f(\textit{mod } \mathcal{A}) + f(\textit{div } \mathcal{A})$.

Thus, given an arithmetic circuit C (or an ABP P) computing a polynomial in $\mathbb{F}\langle X\rangle$ and a finite automaton $\mathcal{A}$ (a DFA or an NFA) we can talk of the polynomials $C(\text{mod } \mathcal{A})$, $C(\text{div } \mathcal{A})$, $P(\text{mod } \mathcal{A})$ and $P(\text{div } \mathcal{A})$. We are interested in the *expressive power* of intersection and quotienting of circuits and ABPs by finite automaton. In some of these cases, we are able to show bounds on the circuit (respectively, ABP) size of these intersections and quotients in terms of the sizes of C (or P) and $\mathcal{A}$. In contrast, quotienting with NFAs is much more expressive. For example, the permanent can be expressed as $P(\text{mod } \mathcal{A})$ for an ABP P and an NFA $\mathcal{A}$ that are polynomial size bounded. These results are presented in Section 2 and Section 5. Additionally, we define the *Hadamard product* of two polynomials f and g in $\mathbb{F}\langle X\rangle$, which is an "algebraic" version of the intersection and we examine the expressive power of the Hadamard product.[1] It turns out that the *noncommutative* branching program complexity of the Hadamard product $f \circ g$ is essentially the product of the branching program sizes for f and g. We are able to use this property to give a *new* deterministic polynomial time identity test for noncommutative algebraic branching programs over $\mathbb{Q}$ (shown to be deterministic polynomial time by Raz and Shpilka [RS05]). Our identity test can also be parallelized to give a deterministic NC^2 algorithm (in fact, we show it is complete for the logspace counting class $\mathrm{C_=L}$). These results are explained in Section 2.

We define the **Monomial Search Problem** which is the natural search version of polynomial identity testing:

Given a polynomial $f \in \mathbb{F}\langle X\rangle$ (or, in the commutative case $f \in \mathbb{F}[X]$) of total degree d by an arithmetic circuit C or an ABP, the problem is to *find* a nonzero monomial of the polynomial f.

Applying our results on intersection of noncommutative ABPs over $\mathbb{F}$ with automata (specifically DFAs), we give a randomized RNC^2 algorithm for finding a nonzero monomial and its coefficient. With some more work we can show the same result even for the case of commutative ABPs.

Our result for monomial search for commutative ABPs can be seen as an algebraic generalization of the search version of the maximum matching problem for which the Mulmuley-Vazirani-Vazirani algorithm provided an RNC^2 algorithm in a celebrated paper [MVV87] in which they introduced the isolation lemma to do a parallel search. Indeed, the result on matchings can be derived in our setting.

In general, for arithmetic circuits too, we obtain a randomized NC reduction of the monomial search problem to identity testing.

Arithmetic Circuit Size and Ideal Membership

Lower bounds for noncommutative computation were first studied in the pioneering paper of Nisan [N91]. He studies noncommutative arithmetic circuits, formulas and algebraic branching programs. Using a rank argument Nisan shows that the noncommutative permanent or determinant polynomials in the ring $\mathbb{F}\langle x_{11}, \cdots, x_{nn}\rangle$ require exponential size noncommutative formulas (and noncommutative algebraic branching programs).

[1] We call this the Hadamard product, because it is motivated by the Hadamard product of matrices associated with algebraic branching programs [N91].

Nisan's results are over the *free* noncommutative ring $\mathbb{F}\langle X\rangle$. Chien and Sinclair, in [CS04], explore the same question over other noncommutative algebras. They refine Nisan's rank argument to show exponential size lower bounds for formulas computing the permanent or determinant over specific noncommutative algebras, like the algebra of 2×2 matrices over $\mathbb{F}$, the quaternion algebra, and a host of other examples.

In this paper, we apply our results on intersection of circuits and ABPs with DFAs to easily derive similar results for *monomial algebras*.

Recall that an ideal I (more precisely, a 2-sided ideal) of the noncommutative polynomial ring $\mathbb{F}\langle X\rangle$ is a subring that is closed under both left and right multiplication by the ring elements. We would like to study arithmetic circuit lower bounds and identity testing in the quotient algebra $\mathbb{F}\langle X\rangle/I$ for different classes of ideals, where the ideal I is given by a generating set of polynomials. For instance, the circuit size $C_I(f)$ in the algebra $\mathbb{F}\langle X\rangle/I$ is

$$\min_{g\in I} C(f+g).$$

Polynomial identity testing in the algebra $\mathbb{F}\langle X\rangle/I$ is basically testing membership of a polynomial in the ideal I. The problem is intractable in general, but for special ideals it can be easier.

If I is a *finitely generated monomial ideal* of $\mathbb{F}\langle X\rangle$, we can design a polynomial-size DFA $\mathcal{A}$ that accepts precisely the monomials in I. Applying this idea, we show that the Permanent (and Determinant) in the quotient algebra $\mathbb{F}\langle X\rangle/I$ still requires exponential size ABPs. Furthermore, the Raz-Shpilka deterministic identity tests for ABPs carry over easily to $\mathbb{F}\langle X\rangle/I$.

For commutative monomial algebras too our approach is through finite automata. The results we obtain are weaker. Our main result here is an exponential size monotone circuit lower bound for the Permanent over any monomial algebra $\mathbb{Q}[x_1, x_2, \cdots, x_n]/I$, where I is generated by $o(n/\lg n)$ monomials.

2 Intersecting and Quotienting Circuits by DFAs

In this section we focus on the circuit and ABP size complexities of $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$, in terms of the circuit (resp. ABP) complexity of f and the size of the automaton $\mathcal{A}$, in the case when $\mathcal{A}$ is a deterministic finite automaton. The bounds we obtain are *constructive*: we will efficiently compute a circuit (resp. ABP) for $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$ from the given circuit (resp. ABP) for f and $\mathcal{A}$.

We will apply these results to obtain a randomized NC^2 algorithm for the monomial search problem for ABPs (both noncommutative and commutative).

We first recall the following complexity measures for a polynomial $f \in \mathbb{F}\langle X\rangle$ from Nisan [N91].

Definition 2. [N91] *For $f \in \mathbb{F}\langle X\rangle$, we denote its formula complexity by $F(f)$, its circuit complexity by $C(f)$, and the algebraic branching program complexity by $B(f)$. For $\mathbb{F} = \mathbb{R}$ and a polynomial $f \in \mathbb{F}\langle X\rangle$ with positive coefficients, its* monotone *circuit complexity is denoted by $C^+(f)$,*

We have the following theorem relating the complexity of $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$ to the complexity of f.

Theorem 1. *Let $f \in \mathbb{F}\langle X\rangle$ and $\mathcal{A}$ be a DFA with s states accepting some subset of X^*. Then, for $g \in \{f(\text{mod } \mathcal{A}), f(\text{div } \mathcal{A})\}$ we have*

1. $C(g) \leq C(f) \cdot (ns)^{O(1)}$.
2. $C^+(g) \leq C^+(f) \cdot (ns)^{O(1)}$.

3. $B(g) \leq B(f) \cdot (ns)^{O(1)}$.

Furthermore, the circuit (ABP) of the size given above for polynomial g can be computed in deterministic logspace (hence in NC^2*) on input a circuit (resp. ABP) for f and the DFA $\mathcal{A}$.*

Proof. We first describe a circuit construction that proves parts 1 and 2 of the theorem.

Let $\mathcal{A} = (Q, X, \delta, q_0, F)$ be the quintuple describing the given DFA with s states. We can extend the transition function δ to words (i.e. monomials) in X^* as usual: $\delta(a, m) = b$ for states $a, b \in Q$ and a monomial m if the DFA goes from state a to b on the monomial m. In particular, we note that $\delta(a, \epsilon) = a$ for each state a. As in automata theory, this is a useful convention because when we write a polynomial $f \in \mathbb{F}\langle X \rangle$ as $\sum c_\alpha m_\alpha$, we can allow for ϵ as the monomial corresponding to the constant term in f.

Let C be the given circuit computing polynomial f. For each gate g of C, let f_g denote the polynomial computed by C at the gate g. In the new circuit C' we will have s^2 gates $\langle g, a, b \rangle, a, b \in Q$ corresponding to each gate g of C. Recall that $\mathrm{mon}(f)$ is the set of monomials of f. Let $M_{ab} = \{m \in X^* \mid \delta(a, m) = b\}$ for states $a, b \in Q$. At the gate $\langle g, a, b \rangle$ the circuit C' will compute the polynomial

$$f_g^{a,b} = \sum_{m_\alpha \in M_{ab} \cap mon(f_g)} c_\alpha m_\alpha,$$

where $f_g = \sum c_\alpha m_\alpha$.

The input-output connections between the gates of C' are now easy to define. If g is a $+$ gate with input gates h and k so that $f_g = f_h + f_k$, we have

$$f_g^{a,b} = f_h^{a,b} + f_k^{a,b},$$

implying that $\langle h, a, b \rangle$ and $\langle k, a, b \rangle$ are the inputs to the $+$ gate $\langle g, a, b \rangle$. If g is a $\times$ gate, with inputs h and k so that $f_g = f_h \cdot f_k$, we have

$$f_g^{a,b} = \sum_{c \in Q} f_h^{a,c} \cdot f_k^{c,b}.$$

This simple formula can be easily computed by a small subcircuit with $O(s)$ many $+$ gates and $\times$ gates.

Finally, let out denote the output gate of circuit C, so that $f_{out} = f$. It follows from the definitions that

$$f(\mathrm{mod}\ \mathcal{A}) = \sum_{a \notin F} f_{out}^{q_0,a}$$
$$f(\mathrm{div}\ \mathcal{A}) = \sum_{a \in F} f_{out}^{q_0,a}$$

Hence, by introducing a small formula for this computation with suitably designated output gate, we can easily get the circuit C' to compute $f(\mathrm{mod}\ \mathcal{A})$ or $f(\mathrm{div}\ \mathcal{A})$.

The correctness of our construction is immediate. Furthermore, $\mathrm{size}(C')$ satisfies the claimed bound. Note that C' will remain a monotone circuit if the given circuit C is monotone. This completes the proof of the first two parts.

Now we prove part 3 of the statement. Let P be an ABP computing polynomial f and $\mathcal{A}$ be a given DFA. The idea for the construction of ABPs that compute $f(\mathrm{mod}\ \mathcal{A})$ and $f(\mathrm{div}\ \mathcal{A})$ is quite similar to

the construction described for part 1. Consider for instance the ABP P' for $f(\text{mod } \mathcal{A})$. Consider the directed acyclic layered graph underlying the ABP P. In the new ABP P' we will have exactly the same number of layers as for P. However, for each node b in the i^{th} layer of ABP P we will have nodes $\langle b, q\rangle$ for each state q of the DFA $\mathcal{A}$. Now, let f_b denote the polynomial that is computed at the node b by the ABP P. The property that the construction of P' can easily ensure is

$$f_b = \sum_q f_{\langle b,q\rangle},$$

where $f_{\langle b,q\rangle}$ is the polynomial computed at node $\langle b, q\rangle$ by the ABP P'. More precisely, let M_q be the set of all nonzero monomials m of f_b such that on input m the DFA $\mathcal{A}$ goes from start state to state q. Then the polynomial $f_{\langle b,q\rangle}$ will be actually the sum of all the terms of the polynomial f_b corresponding to the monomials in M_q. This construction can now be easily used to obtain ABPs for each of $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$, and the size of the ABP will satisfy the claimed bound. We omit the easy details of the construction of the ABPs.

A careful inspection of the constructions shows that for a given circuit C and DFA $\mathcal{A}$ we can construct the circuits that compute $C(\text{mod } \mathcal{A})$ and $C(\text{div } \mathcal{A})$ in deterministic logspace (and hence in NC^2). Likewise, the construction of the ABPs for $P(\text{mod } \mathcal{A})$ and $P(\text{div } \mathcal{A})$ for a given ABP P can also be computed in deterministic logspace. ■

The following is an immediate corollary of Theorem 1.

Corollary 1. *1. Given a noncommutative ABP P computing a polynomial $f \in \mathbb{F}\langle X\rangle$ and a deterministic finite automaton $\mathcal{A}$ we can test in deterministic polynomial time whether $f(\text{mod } \mathcal{A})$ is identically zero.*

2. Given a noncommutative circuit C computing $f \in \mathbb{F}\langle X\rangle$ and a DFA $\mathcal{A}$ then we can test whether $f(\text{mod } \mathcal{A})$ is identically zero in randomized polynomial time (if C is a monotone circuit then we can test if $f(\text{mod } \mathcal{A})$ is zero in deterministic polynomial time).

We now prove a similar result for commutative ABPs. However, we need to be careful to consider the right kind of DFAs that capture commutativity so that the constructions of Theorem 1 are meaningful and go through.

Definition 3 (Commutative Automata). *Let $w \in X^d$ be any string of length d over the alphabet $X = \{x_1, \cdots, x_n\}$. Let $C_w \subset X^d$ denote the set of all words w' obtained by shuffling the letters of w.*

A DFA (or NFA) $\mathcal{A}$ over the alphabet $X = \{x_1, \cdots, x_n\}$ is said to be commutative *if for every word $w \in X^*$, w is accepted by $\mathcal{A}$ if and only if every word in C_w is accepted by $\mathcal{A}$.*

The following theorem is the analogue of Theorem 1 for intersecting and quotienting commutative circuits and commutative ABPs by commutative DFAs. We omit the proofs as the constructions are identical to those in the proof of Theorem 1. It can be easily seen from Definition 1 and the proof of Theorem 1 that in the commutative case the constructions are meaningful and work correctly when the DFAs considered are commutative.

Theorem 2. *Let $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ and $\mathcal{A}$ be a commutative DFA with s states over alphabet $X = \{x_1, \cdots, x_n\}$. Then, for $g \in \{f(\text{mod } \mathcal{A}), f(\text{div } \mathcal{A})\}$ we have*

1. $C(g) \leq C(f) \cdot (ns)^{O(1)}$.
2. $C^+(g) \leq C^+(f) \cdot (ns)^{O(1)}$.
3. $B(g) \leq B(f) \cdot (ns)^{O(1)}$.

Furthermore, the commutative circuit (ABP) for polynomial g meeting the above size bounds are computable in deterministic logspace (hence in NC^2) on input a circuit (resp. ABP) for f and DFA $\mathcal{A}$.

2.1 Monomial search problem

Next we consider the monomial search problem for ABPs in both commutative and noncommutative setting. The goal is to compute a nonzero monomial of the polynomial computed by the given ABP. We apply Theorem 1 to prove these results.

Theorem 3. *Given a noncommutative ABP P computing a polynomial f in $\mathbb{F}\langle X\rangle$ there is a randomized* NC^2 *algorithm that computes a nonzero monomial of f. More precisely, the algorithm is a randomized* $\mathrm{FL}^{\mathrm{GapL}}$ *algorithm.*

Proof. We can assume wlog that the given ABP P computes a homogeneous degree d polynomial. The proof is by a careful application of the isolating lemma of [MVV87]. Define the universe $U = \{x_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq d\}$, where the element x_{ij} stands for the occurrence of x_i in the j^{th} position in a monomial. With this encoding every degree d monomial m over X can be encoded as a subset S_m of size d in U, where $S_m = \{x_{ij} \mid x_i \text{ occurs in } j^{th} \text{ position in } m\}$. Following the isolation lemma, we pick a random weight assignment $w : U \longrightarrow [4dn]$. The weight of a monomial m is defined as $w(m) = w(S_m) = \sum_{x_{ij} \in S_m} w(x_{ij})$, and with probability $1/2$ there is a unique minimum weight monomial.

Construction of weight-checking DFA: For any weight value a such that $1 \leq a \leq 4nd^2$, we can easily construct a DFA M_w^a that accepts a monomial $m \in X^*$ iff $m \in X^d$ and $w(m) = a$. This DFA will have $O(4nd^3)$ many states. Furthermore, we can compute this DFA in deterministic logspace, given the weight function w.

Next, by Theorem 1 we can compute an ABP P_w^a that computes the polynomial $P(\text{div } M_w^a)$ for each of $1 \leq a \leq 4nd^3$. With probability $1/2$ we know that one of $P(\text{div } M_w^a)$ accepts precisely one monomial of the original polynomial f (with the same coefficient).

In order to find each variable occurring in that unique monomial accepted by, say, P_w^a we will design another DFA $\mathcal{A}_{ij}$ which will accept a monomial $m \in X^d$ if and only if x_i occurs in the j^{th} position. Again by Theorem 1 we can compute an ABP $B_{i,j,a,w}$ that accepts precisely $P_w^a(\text{div } \mathcal{A}_{ij})$. Now, the ABP $B_{i,j,a,w}$ either computes the zero polynomial (if x_i does not occur in the j^{th} position of the unique monomial of P_w^a) or it computes that unique monomial of P_w^a. In order to test which is the case, notice that we can *deterministically* assign the values $x_i = 1$ for each variable x_i. Crucially, since P_w^a has a *unique* monomial it will be nonzero even for this deterministic and commutative evaluation. Since the evaluation of an ABP is for commutative values (scalar values), we can carry it out in NC^2 in fact, in $\mathrm{FL}^{\mathrm{GapL}}$ for any fixed finite field or over $\mathbb{Q}$, (see e.g. [T91], [V91], [MV97]).

Let m be the monomial that is finally constructed. We can construct a DFA $\mathcal{A}_m$ that accepts only m and no other strings. By Theorem 1 we can compute an ABP P' for the polynomial $P(\text{div } \mathcal{A}_m)$ and again check if P' is zero or nonzero by substituting all $x_i = 1$ and evaluating. This will make the algorithm actually a zero-error NC^2 algorithm.

The success probability can be boosted by parallel repetition. This completes the proof sketch. ∎

Next we describe a randomized NC^2 algorithm for the Monomial search problem for commutative ABPs. This is the best we can currently hope for, since deterministic polynomial-time identity testing for commutative ABPs is a major open problem. Our monomial search algorithm will use the following generalized isolation lemma of Klivans and Spielman [KS01].

Lemma 1. [KS01, Lemma 4] *Let L be a collection of linear forms over variables $z_1, \ldots, z_n$ with integer coefficients in $\{0, 1, \ldots, K\}$. If each z_i is picked independently and uniformly at random from*

$\{0, 1, \ldots, 2Kn\}$ *then with probability at least* $\frac{1}{2}$ *there is a unique linear form in* L *which attains minimum value at* $(z_1, z_2, \ldots, z_n)$.

Theorem 4. *The monomial search problem for commutative algebraic branching programs is in randomized* NC^2 *(more precisely, it is in randomized* $\mathrm{FL}^{\mathrm{GapL}}$*).*

Proof. We will only provide a proof outline, since it is similar to that of Theorem 3 and builds on Theorem 2.

Let P be a commutative algebraic branching program computing a polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$. We assume, without loss of generality, that f is homogeneous of degree d. First, pick a random weight function $w : \{x_1, \cdots, x_n\} \longrightarrow [2dn]$. Next for each number a such that $0 \leq a \leq 2d^2n$ we construct a DFA A_w^a which will accept a monomial $m \in X^*$ iff $m \in X^d$ and $w(m) = a$, where $w(m) = \sum_i w(x_i) \cdot \alpha_i$, and x_i occurs exactly α_i times in m. Crucially, notice that the DFA A_w^a is a *commutative* DFA. Hence, applying Theorem 2, for each number a we can obtain an ABP P_w^a in deterministic logspace.

By Lemma 1 with probability at least $1/2$ one of the ABPs P_w^a accepts a unique monomial $m = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$ of the given polynomial f. Suppose that value of a is u. Let $c \neq 0$ denote the coefficient of the unique monomial m in f computed by the ABP P_w^u.

Our goal now is to find the α_i's. To find α_i, we evaluate the ABP P_w^a by setting $x_j = 1$ for all $j \neq i$. Clearly, the ABP P_w^u will evaluate on this input to $cx_i^{\alpha_i}$. Evaluating each of the ABPs P_w^a on the input $(1, \cdots, 1, x_i, 1, \cdots, 1)$ can be done in NC^2. Indeed, it can be done in $\mathrm{FL}^{\mathrm{GapL}}$, since we only need determinant computation over the field $\mathbb{F}$. This completes the proof sketch. ∎

Theorem 4 is a generalization of the Mulmuley et al. result on constructing maximum matchings in RNC^2 [MVV87] using the connection between matchings and the rank of the Tutte matrix M. We can apply Theorem 4 to find maximum matchings in RNC^2 (more precisely, randomized $\mathrm{FL}^{\mathrm{GapL}}$).

An easy prefix search gives a deterministic polynomial time algorithm for the monomial search problem for noncommutative ABPs.

Theorem 5. *There is a deterministic polynomial time algorithm for the monomial search problem for noncommutative algebraic branching programs.*

Proof. Let P be the input noncommutative ABP that computes a polynomial $f \in \mathbb{F}\langle X \rangle$. W.l.o.g. assume that $f = \sum_\alpha a_\alpha m_\alpha$ is homogeneous polynomial of degree d, where m_α are the nonzero monomials of f. We solve the monomial search problem by a prefix search that is guided by the Raz-Shpilka deterministic identity test [RS05]. For any string $w \in X^k, k \leq d$ we can define a polynomial-sized DFA D_w that accepts only words of the form wy for some word $y \in X^*$. Applying Theorem 1 we can construct an ABP P_w that computes $f(\text{div } D_w)$ for any given w. Notice that

$$f(\text{div } D_w) = \sum_{m_\alpha = wy} a_\alpha m_\alpha.$$

The prefix search algorithm is now easy to describe. Starting with $w = \epsilon$, we successively compute ABPs $P_\epsilon, P_{w_1}, \cdots, P_{w_d}$, where $|w_k| = k$ and w_k is a prefix of w_{k+1} for each k.

Each P_{w_k} is an ABP that computes $f(\text{div } D_{w_k})$ as described above. Notice that P_ϵ computes f. Suppose $f(\text{div } D_{w_k})$ is nonzero. Then the prefix search sets $w_{k+1} = w_k x_i$ for the first indeterminate x_i such that $P_{w_{k+1}}$ computes a nonzero polynomial (to check this we use the Raz-Shpilka identity test on $P_{w_{k+1}}$ [RS05]). Since $f(\text{div } D_{w_k}) \neq 0$ for some indeterminate x_i the polynomial $f(\text{div } D_{w_{k+1}})$ is

nonzero. Hence the prefix search will successfully continue. The output of the monomial search will be w_d. ■

Finally, we note that the same technique of isolating using a DFA and DFA intersection with a circuit can be used to give a randomized NC reduction from monomial search for noncommutative (commutative) circuits to noncommutative (resp. commutative) polynomial identity testing.

Theorem 6. *Monomial search for noncommutative (commutative) circuits is randomized* NC *reducible to noncommutative (resp. commutative) polynomial identity testing.*

3 The Hadamard Product

Motivated by the well-known Hadamard product of matrices (see e.g. [Bh97]), in this section we consider the Hadamard product of polynomials.

Definition 4. *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*

$$f \circ g = \sum_{\alpha} a_\alpha b_\alpha m_\alpha,$$

where $f = \sum_\alpha a_\alpha m_\alpha$ and $g = \sum_\alpha b_\alpha m_\alpha$.

Clearly, $\text{mon}(f \circ g) = \text{mon}(f) \cap \text{mon}(g)$. Thus, the Hadamard product can be seen as an algebraic version of the intersection of formal languages.

Our definition of the Hadamard product of polynomials is actually motivated by the well-known Hadamard product $A \circ B$ of two $m \times n$ matrices A and B. We recall the following well-known bound for the rank of the Hadamard product.

Proposition 1. *Let A and B be two $m \times n$ matrices over any field $\mathbb{F}$. Then*

$$\text{rank}(A \circ B) \leq \text{rank}(A)\text{rank}(B).$$

It is known from Nisan's paper [N91] 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 $B(f) = \sum_k \text{rank}(M_k(f))$.

Indeed as a consequence of Nisan's result and the above proposition we can easily obtain the following nice bound on the ABP complexity of $f \circ g$.

Lemma 2. *For $f, g \in \mathbb{F}\langle X\rangle$ we have $B(f \circ g) \leq B(f)B(g)$.*

Proof.

$$B(f \circ g) = \sum_k \text{rank}(M_k(f \circ g)) \leq \sum_k \text{rank}(M_k(f))\text{rank}(M_k(g)) \quad (1)$$

$$\leq (\sum_k \text{rank}(M_k(f))(\sum_k \text{rank}(M_k(g))) \quad (2)$$

■

We will now prove a more interesting algorithmic version of this upper bound.

Theorem 7. *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 polynomial time algorithm that will output an ABP R for $f \circ g$ such that the size of R is a constant factor of product of the sizes of P and Q. (Indeed, the construction of R can be done in deterministic logspace.)*

Proof. Without loss of generality we can assume that both P and Q are homogeneous ABP's of degree d. If not we can easily construct an ABP to compute $f_i \circ g_i$ separately for each i, where f_i, g_i denotes degree i homogeneous parts of the polynomial f and g, respectively. Since hadamard product is distributive over addition we can compute $f \circ g$ by adding all $f_i \circ g_i$. By allowing parallel edges between nodes of ABPs P, Q we can assume that the labels associated with each edge in an ABP is either 0 or $a \cdot x_i$ for some $a \in \mathbb{F}$, $i \in [n]$. Let s_1 and s_2 bound the number of nodes in any particular layer of P and Q respectively. We denote the j^{th} node in layer i by $\langle i, j\rangle$ for ABP's P and Q. Next we give the construction of the ABP R which will compute 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 add it with label 0.

In the new ABP R we will add an edge from $\langle i, a, b\rangle$ to $\langle i+1, c, e\rangle$ with label $\alpha \cdot \beta x_t$ in R if and only if there is an edge from node $\langle i, a\rangle$ to $\langle i+1, c\rangle$ with label $\alpha \cdot x_t$ in P and an edge from $\langle i, b\rangle$ to $\langle i+1, e\rangle$ with label $\beta \cdot x_t$ in ABP Q. Let $\langle 0, a, b\rangle$ and $\langle d, c, e\rangle$ be the source and the sink nodes of ABP R respectively, 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. 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 see that $h_{\langle i,a,b\rangle} = f_{\langle i,a\rangle} \circ g_{\langle i,b\rangle}$ by using an induction argument on the number of layers in the ABPs, we skip the details of the inductive proof. It follows from this 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. ■

This theorem has an interesting consequence for noncommutative identity testing of ABP's (and formulas) over $\mathbb{Q}$. This gives an alternative algorithm to the one due to Raz and Shpilka's deterministic polynomial time identity test [RS05]. In fact, we actually strengthen the result to give an NC^2 upper bound.

Theorem 8. *The problem of polynomial identity testing for noncommutative algebraic branching programs over $\mathbb{Q}$ is in NC^2. (In fact, it is complete for the logspace counting class $\mathrm{C}_=\mathrm{L}$ under logspace reductions).*

Proof. Let P be the given ABP computing $f \in \mathbb{Q}\langle X\rangle$. We apply the construction of Theorem 7 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 7, 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 checking if a specific entry of an iterated integer matrix product is zero is in the logspace counting class $\mathrm{C}_=\mathrm{L}$ (e.g. see [ABO99]). However, $\mathrm{C}_=\mathrm{L}$ is contained in NC^2, in fact in TC^1. On the other hand, the problem of checking if an integer matrix A is singular is complete for $\mathrm{C}_=\mathrm{L}$

The standard GapL algorithm for computing $\det(A)$ [T91] can be converted to an ABP P_A which will compute $\det(A)$. Hence P_A computes the identically zero polynomial iff A is singular. Putting it all together, it follows that identity testing of noncommutative ABPs over rationals is complete for the class $\mathrm{C_{=}L}$. ■

Analogous to Theorem 7 we show that $f \circ g$ has small circuits if f has a small circuit and g has a small ABP.

Theorem 9. *Let $f, h \in \mathbb{F}\langle x_1, x_2, \cdots, x_n\rangle$ be given by a degree p circuit C and an ABP P respectively, where $p \leq poly(n)$. Then we can compute in polynomial time a circuit C' that computes $f \circ h$ where the size of C' is polynomially bounded by sizes of C and P.*

Proof. Without loss of generality we can assume that both f and h are homogeneous polynomials of degree d. If not we can compute $f_i \circ h_i$ separately for each $1 \leq i \leq p$, where f_i, h_i denotes degree i homogeneous parts of f and h respectively. Since the Hadamard product is distributive over addition we can compute $f \circ h$ by adding all $f_i \circ h_i$'s. Let f_g denotes the polynomial computed at gate g of C. Let w denotes number of nodes in any layer of P. Let $\langle i, a\rangle$ denote the a^{th} node in the i^{th} layer of P for $0 \leq i \leq d, 1 \leq a \leq w$. Let $h_{(i,a),(j,b)}$ denote the polynomial computed by ABP P'. The ABP P' is same as P but with source node $\langle i, a\rangle$ and sink node $\langle j, b\rangle$. We construct circuit C' that computes polynomial $f \circ h$. In C' we have gates $\langle g, l, (i, a), (i + l, b)\rangle$ for $0 \leq l \leq d, 0 \leq i \leq d$, $1 \leq a, b \leq w$ associated with each gate g of C, such that at each gate $\langle g, l, (i, a), (i + l, b)\rangle$ the circuit C' will compute

$$r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)}$$

where $f_{\langle g,l\rangle}$ denotes the degree l homogeneous component of the polynomial f_g. Next we describe the input-output connections for C'. If g is a $+$ gate of C with input gates g_1, g_2 so that $f_g = f_{g_1} + f_{g_2}$, we have $r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = r^{\langle g_1,l\rangle}_{(i,a),(i+l,b)} + r^{\langle g_2,l\rangle}_{(i,a),(i+l,b)}$, for $0 \leq l \leq d$, $0 \leq i \leq d, 1 \leq a, b \leq w$. In other words, $\langle g, l, (i, a), (i+l, b)\rangle$ is $+$ gate in C' with input gates $\langle g_1, l, (i, a), (i+l, b)\rangle$ and $\langle g_2, l, (i, a), (i+l, b)\rangle$. If g is a $\times$ gate in C we will have

$$r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = \sum_{j=0}^{l} \sum_{t=1}^{w} r^{\langle g_1,j\rangle}_{(i,a),(i+j,t)} \cdot r^{\langle g_2,l-j\rangle}_{(i+j,t),(i+l,b)}$$

This formula can be easily computed by a small subcircuit. Let $\langle g, d, (0, 1), (d, 1)\rangle$ be the output gate of C', where g is the output gate of C and $(0, 1), (d, 1)$ denotes the source and the sink of the ABP P respectively. This completes the description of the circuit C'. Next we inductively prove that at gate $\langle g, l, (i, a), (i + l, b)\rangle$, C' indeed computes the polynomial $f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)}$. At $+$ gate g of C the claim is easy to see. Let g is $\times$ gate of C with inputs g_1, g_2 such that $f_g = f_{g_1} \cdot f_{g_2}$ and assume that the claim holds true for the gates g_1 and g_2. This implies $f_{\langle g,l\rangle} = \sum_{i=0}^{l} f_{\langle g_1,i\rangle} \cdot f_{\langle g_2,l-i\rangle}$. So we have,

$$f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)} = [\sum_{j=0}^{l} f_{\langle g_1,j\rangle} \cdot f_{\langle g_2,l-j\rangle}] \circ h_{(i,a),(i+l,b)} \tag{3}$$

$$= \sum_{j=0}^{l} (f_{\langle g_1,j\rangle} \cdot f_{\langle g_2,l-j\rangle} \circ h_{(i,a),(i+l,b)}) \tag{4}$$

$$= \sum_{j=0}^{l} \sum_{t=1}^{w} f_{\langle g_1,j\rangle} \cdot f_{\langle g_2,l-j\rangle} \circ h_{(i,a),(i+j,t)} \cdot h_{(i+j,t),(i+l,b)} \tag{5}$$

$$= \sum_{j=0}^{l} \sum_{t=1}^{w} (f_{\langle g_1,j\rangle} \circ h_{(i,a),(i+j,t)}) \cdot (f_{\langle g_2,l-j\rangle} \circ h_{(i+j,t),(i+l,b)}) \tag{6}$$

The first two equalities easily follows from distributivity of Hadamard product and the fact that, for any $i \leq j \leq i+l$, $h_{(i,a),(i+l,b)} = \sum_{t=1}^{w} h_{(i,a),(i+j,t)} \cdot h_{(i+j,t),(i+l,b)}$ and the last equality follows since $f_{\langle g_1,j\rangle}, h_{(i,a),(i+j,t)}$ are both degree j homogeneous polynomials. By induction hypothesis we have $r^{\langle g_1,j\rangle}_{(i,a),(i+j,t)} = f_{\langle g_1,j\rangle} \circ h_{(i,a),(i+j,t)}$ and $r^{g_2,l-j}_{(i+j,t),(i+l,b)} = f_{\langle g_2,l-j\rangle} \circ h_{(i+j,t),(i+l,b)}$. This proves the desired claim

$$r^{\langle g,l\rangle}_{(i,a),(i+l,b)} = f_{\langle g,l\rangle} \circ h_{(i,a),(i+l,b)}.$$

Clearly, at the output gate $\langle g, d, (0,1), (d,1)\rangle$ the circuit C' will compute the polynomial $f \circ h$. The size of C' is bounded by a polynomial in the sizes of C and P. ∎

On the other hand, suppose f and g individually have small circuit complexity. Two questions arise: does $f \circ g$ have small circuit complexity? Can we compute such a circuit for $f \circ g$ from the circuits for f and g?

We first consider these questions for monotone circuits. It is useful to understand the connection between monotone noncommutative circuits and context-free grammars. We recall the following definition.

Definition 5. *We call a context-free grammar in Chomsky normal form $G = (V, T, P, S)$ an* acyclic CFG *if for any nonterminal $A \in V$ there does not exist any derivation of the form $A \Rightarrow^* uAw$.*

The size $size(G)$ of an acyclic CFG $G = (V, T, P, S)$ is defined as $|V| + |T| + \text{size}(P)$, where V, T, and P are the sets of variables, terminals, and production rules. We note the following easy proposition that relates acyclic CFGs to monotone noncommutative circuits over X.

Proposition 2. *For a monotone circuit C of size s computing a polynomial $f \in \mathbb{Q}\langle X\rangle$ let mon(f) denote the set of nonzero monomials of f. Then there is an acyclic CFG G for mon(f) with* size$(G) = O(s)$*. Conversely, if G is an acyclic CFG of size s computing some finite set $L \subset X^*$ of monomials over X, there exists a monotone circuit of size $O(s)$ that computes a polynomial $\sum_{m_\alpha \in L} a_\alpha m_\alpha \in \mathbb{Q}\langle X\rangle$, where the positive integer a_α is the number of derivation trees for m_α in the grammar G.*

Proof. First we prove the forward direction by constructing an acyclic CFG $G = (V, T, P, S)$ for mon(f). Let $V = \{A_g |$ g is a gate of circuit $C\}$ be the set of nonterminals of G. We include a production in P for each gate of the circuit C. If g is an input gate with input $x_i, 1 \leq i \leq n$ include the production $A_g \to x_i$ in P. If the input is a *nonzero* field element then add the production $A_g \to \epsilon$.[2] Let f_g denote the polynomial computed at gate g of C. If g is a $\times$ gate with $f_g = f_h \times f_k$ then include the production $A_g \to A_h A_k$ and if it is $+$ gate with $f_g = f_h + f_k$ include the productions $A_g \to A_h \mid A_k$. Let the start symbol $S = A_g$, where g is the output gate of C. It is easy to see from the above construction that G is acyclic moreover $size(G) = O(s)$ and it generates the finite language mon(f). The converse direction is similar. ∎

Theorem 10. *There are monotone circuits C and C' computing polynomials f and g in $\mathbb{Q}\langle X\rangle$ respectively, such that the polynomial $f \circ g$ requires monotone circuits of size exponential in $|X|$,* size(C)*, and* size(C')*.*

[2] If the circuit takes as input 0, we can first propagate it through the circuit and eliminate it.

Proof. Let $X = \{x_1, \cdots, x_n\}$. Define the finite language $L_1 = \{zww^r \mid z, w \in X^*, |z| = |w| = n\}$ and the corresponding polynomial $f = \sum_{m_\alpha \in L_1} m_\alpha$. Similarly let $L_2 = \{ww^r z \mid z, w \in X^*, |z| = |w| = n\}$, and the corresponding polynomial $g = \sum_{m_\alpha \in L_2} m_\alpha$. It is easy to see that there are poly(n) size *unambiguous* acyclic CFGs for L_1 and L_2. Hence, by Proposition 2 there are monotone circuits C_1 and C_2 of size poly(n) such that C_1 computes polynomial f and C_2 computes polynomial g.

We first show that the finite language $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n \lg n)}$. Assume to the contrary that there is an acyclic CFG $G = (V, T, P, S)$ for $L_1 \cap L_2$ of size $2^{o(n/\lg n)}$. Notice that,

$$L_1 \cap L_2 = \{t \mid t = ww^r w, w \in X^*, |w| = n\}.$$

Consider any derivation tree T' for a word $ww^r w = w_1 w_2 \dots w_n w_n w_{n-1} \dots w_2 w_1 w_1 \dots w_n$. Starting from the root of the binary tree T', we traverse down the tree always picking up the child with larger yield. It is easy to see that there must be a nonterminal $A \in V$ in the derivation tree such that $A \Rightarrow^* u$, $u \in X^*$ and $n \leq |u| < 2n$. Crucially, note that any word that A generates must have same length since every word generated by the grammar G is in $L_1 \cap L_2$ and hence of length $3n$. Let $ww^r w = s_1 u s_2$ where $|s_1| = k$. If both $|s_1|, |s_2| \leq n$ then w^r is a substring of u. As $|u| < 2n$, it is easy to see that the string $s_1 s_2$ will contain at least one occurrence of each $w_i, 1 \leq i \leq n$. If either $|s_1|$ or $|s_2|$ is greater than n then clearly w is a subword of $s_1 s_2$ so it contains w_i for $i = 1$ to n. Hence the pair of subwords s_1 and s_2 completely determines the string $ww^r w$. Therefore, the nonterminal A can derive *at most* one word in X^* for each value of $1 \leq k \leq 2n$. Since there are n^n distinct words in $L_1 \cap L_2$, it follows that there must be at least $\frac{n^n}{2n}$ *distinct* nonterminals in V. This contradicts the size assumption of G.

Since $L_1 \cap L_2$ cannot be generated by any acyclic CFG of size $2^{o(n/\lg n)}$, it follows from Lemma 2 that the polynomial $f \circ g$ can not be computed by any monotone circuit of $2^{o(n/\lg n)}$ size. ■

Remark 1. Theorem 10 shows that the Hadamard product of monotone circuits is more expressive than monotone circuits. It raises the question whether the permanent polynomial can be expressed as the Hadamard product of polynomial-size (or even subexponential size) monotone circuits. Here we note that the permanent *can* be easily expressed as the Hadamard product of $O(n^3)$ many monotone circuits (in fact, monotone ABPs).

Theorem 11. *Suppose there is a deterministic subexponential-time algorithm that takes as input circuits computing polynomials f and g (in $\mathbb{Q}\langle x_1, \cdots, x_n\rangle$) and outputs a circuit for $f \circ g$. Then either* NEXP *is not in* P/poly *or the Permanent does not have polynomial size noncommutative circuits.*

Proof. Let C_1 is a circuit computing polynomial $h \in \mathbb{Q}\langle x_1, \dots, x_n\rangle$. By assumption, we can compute a circuit C_2 for the polynomial $h \circ h$ in subexponential time. Therefore h is identically zero iff $h \circ h$ is identically zero iff C_2 evaluates to 0 on all 1's input. We can easily check if C_2 evaluates to 0 on all 1's input by substitution and evaluation. This gives a deterministic subexponential time algorithm for testing if h is identically zero. By the noncommutative analogue of [KI03], shown in [AMS08], this implies either NEXP is not in P/poly or the Permanent does not have polynomial size noncommutative circuits. ■

We now consider the following problem: given $f, g \in \mathbb{F}\langle X\rangle$ by circuits we want to test if $f \circ g$ is identically zero. We show this problem is coNP-complete. We first need the following Proposition from [AMS08].

Proposition 3. [AMS08] *Given a non-commutative circuit C computing a polynomial $f \in \mathbb{F}\langle X\rangle$ and a monomial $m \in X^*$, in deterministic polynomial time we can compute the coefficient of m in the polynomial f.*

Theorem 12. *Given two* monotone polynomial-degree *circuits C and C' computing polynomial $f, g \in \mathbb{Q}\langle X\rangle$ it is* coNP*-complete to check if $f \circ g$ is identically zero.*

Proof. First we show that the complement of the problem is in NP. We guess a monomial $m_\alpha \in X^*$, $X = \{x_1, \ldots, x_n\}$ and check if coefficient of m_α is nonzero in both C and C'. Note that we can compute the coefficient of m_α in C and C' by Proposition 3 in deterministic polynomial time. This shows that the complement of the problem is in NP. Denote by CFGINT the problem of testing emptiness of the intersection of two acyclic CFGs that generate poly(n) length strings. By Lemma 2 CFGINT is polynomial time many-one reducible to testing if $f \circ g$ is identically zero. The problem of testing if the intersection of two CFGs (with recursion) is empty is known to be undecidable via a reduction from post correspondence problem [HMU, Chapter 9,Page 422]. We can give an analogous reduction from the *bounded* post correspondence problem to CFGINT. The coNP-hardness of CFGINT follows from NP-hardness of bounded post correspondence problem [GJ79]. This completes the proof sketch. ■

4 Monomial Algebras and Automata

Definition 6. *A two-sided ideal $I = \langle m_1, m_2, \cdots, m_r\rangle$ of the noncommutative ring $\mathbb{F}\langle X\rangle$ generated by a finite set of monomials $m_1, \cdots, m_r$ is a* finitely generated monomial ideal *of $\mathbb{F}\langle X\rangle$. The quotient algebra $\mathbb{F}\langle X\rangle/I$ is a* finitely generated monomial algebra*.*

Nisan's work [N91] on lower bounds for noncommutative formulas (and ABPs), followed by Raz and Shpilka's deterministic polynomial time identity test for noncommutative ABPs motivates the question whether such results can be proved over algebras other than $\mathbb{F}\langle X\rangle$. The ultimate goal in this direction would be to obtain lower bounds and deterministic identity tests in the commutative setting. Chien and Sinclair [CS04] extend Nisan's lower bound to matrix algebras (and various other algebras). They show that Nisan's lower bounds for ABPs computing the permanent or determinant holds for the algebra of 2×2 matrices.

In this section we examine this question for monomial algebras. For a polynomial f given by a circuit (or ABP) and a monomial ideal I we are interested in the circuit (resp. ABP) complexity of the polynomial $f(\text{mod } I)$. The corresponding identity testing problem is the *Ideal Membership* problem whether $f \in I$?

First we consider the problem in noncommutative setting.

Theorem 13. *Let $I = \langle m_1, \cdots, m_r\rangle$ be a monomial ideal in $\mathbb{F}\langle X\rangle$. Let P (resp. C) be a noncommutative ABP (resp. a polynomial degree monotone circuit) computing a polynomial $f \in \mathbb{F}\langle X\rangle$. Then there is a deterministic polynomial-time algorithm to test if the polynomial $f(\text{mod } \mathcal{I})$ is identically zero.*

Proof. Consider monomials m_i as strings over alphabet $\{x_1, \ldots, x_n\}$. Let $d = \max_i\{\text{length}(m_i)\}$. Using the Aho-Corasick pattern matching automaton [AC75] we construct a DFA A with $O(dr)$ states which on input a string $s \in X^*$ accepts s if s contains m_i as a substring for some i. Now using Theorem 1 we obtain an ABP P' (resp. a monotone circuit C') of size $poly(n, d, r)$ which computes

the polynomial $g = f(\text{mod } \mathcal{A})$. Clearly, $f \in \mathcal{I}$ iff $g \equiv 0$. Now we can invoke Corollary 1 to complete the proof. ■

As an easy consequence of Theorem 1, we have an immediate lower bound observation for the Permanent.

Corollary 2. *Let $I = \langle m_1, \cdots, m_r \rangle$ be a monomial ideal and C be a noncommutative ABP (or a polynomial degree noncommutative monotone circuit) over indeterminates $\{x_{ij} \mid 1 \leq i, j \leq n\}$. If $C = Perm_n(\text{mod } I)$ then either* size(C) *or the number of generating monomials r for I is $2^{\Omega(n)}$.*

4.1 Commutative monomial algebras

In this subsection we examine the same problem in the commutative case. Let $I = \langle m_1, \cdots, m_k \rangle$ be a monomial ideal contained in $\mathbb{F}[x_1, \cdots, x_n]$. As before $f(\text{mod } I)$ is a meaningful polynomial in $\mathbb{F}[x_1, \cdots, x_n]$ for $f \in \mathbb{F}[x_1, \cdots, x_n]$.

First we introduce some useful notations. For $m, m' \in X^d$ we say $m \sim m'$ if m' can be obtained by shuffling m. Clearly, $\sim$ is an equivalence relation and defines a partition on set of all d length strings. For $\alpha = (\alpha_1, \ldots, \alpha_n)$, $0 \leq \alpha_i \leq d$ and $\sum_i \alpha_i = d$, the partition M_α contains all strings $m \in X^d$ such that m can be obtained by shuffling $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$. For a degree d homogeneous polynomial f in noncommuting variables $x_1, \ldots, x_n$, let $f_\alpha = \sum_{m \in \text{mon}(f) \cap M_\alpha} a_m m$ where $\alpha = (\alpha_1, \ldots, \alpha_n)$, $0 \leq \alpha_i \leq d, \sum_i \alpha_i = d$ and a_m is a coefficient of m in f.

Lemma 3. *Let C is a circuit (resp. ABP R) computing homogeneous polynomial $f \in \mathbb{Q}[X]$ of degree d and A is commutative NFA of size s computing language $L(\mathcal{A}) \subseteq X^d$. There is a deterministic polynomial (in $size(C), s$) time algorithm to construct a circuit C' (resp. ABP R') which computes polynomial $g \in \mathbb{Q}[X]$ such that mon$(f(\text{div } \mathcal{A})) =$ mon(g). Moreover, if C is a monotone then C' is also monotone.*

Proof. We prove the claim for circuits (the ABP case can be similarly done). Since $L = L(\mathcal{A}) \subseteq X^d$ is a finite language, we can assume that the underlying transition graph of NFA A is a layered directed acyclic graph. We can modify $\mathcal{A}$ to an ABP P which will have a node corresponding to each state in $\mathcal{A}$ with edges going between layers. If $\delta(a, x_i) = b$ is a transition in the NFA then in the ABP P we will include a directed edge from a to b with label x_i. Clearly, P computes a polynomial $h = \sum_{m \in L} a_m m$, where each a_m is the number of accepting paths of the NFA $\mathcal{A}$ on input m. Notice that, for any $m, m' \in L$ such that $m \sim m'$ we have $a_m = a_{m'}$, because the commutative NFA $\mathcal{A}$ has the *same* number of accepting paths for m and m'. So for any $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\alpha_i \in [d]$ such that $\sum_i \alpha_i = d$, we have $h_\alpha = a_\alpha \sum_{m \in \text{mon}(h) \cap M_\alpha} m$, where $a_\alpha \geq 1$ is number of accepting paths of A on input any $m \in M_\alpha \cap \text{mon}(h)$. Suppose $f' \in \mathbb{Q}\langle X \rangle$ is a degree d homogeneous polynomial computed by the circuit C, assuming the multiplication gates of C are *noncommutative*, and the inputs to each gate are ordered left to right. It is easy to see that the polynomial $f(\text{div } \mathcal{A})$ in commuting variables $x_1, \ldots, x_n$ will have nonzero monomial $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$ iff $\sum_{m \in S_0} a_m \neq 0$, where $S_0 = \text{mon}(f') \cap L \cap M_\alpha$ and a_m is the coefficient of m in f'. By Theorem 9 we can compute in deterministic polynomial time a circuit C' of size polynomial in size(C) and $size(P)$ such that C' computes $g' = f' \circ h \in \mathbb{Q}\langle X \rangle$. We have,

$$g' = f' \circ h = \sum_\alpha f'_\alpha \circ h_\alpha = \sum_\alpha [\sum_{m \in S_1} a_m m] \circ [a_\alpha \sum_{m \in S_2} m] \tag{7}$$

Where $S_1 = \text{mon}(f') \cap M_\alpha$ and $S_2 = \text{mon}(h) \cap \text{M}_\alpha$. Let $g \in \mathbb{Q}[X]$ be the polynomial obtained on evaluating C' in commutative algebra $\mathbb{Q}[X]$. Clearly, g will have a nonzero monomial $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$

iff $a_\alpha \sum_{m \in S_1 \cap S_2} a_m \neq 0$ iff $a_\alpha \sum_{m \in S_0} a_m \neq 0$ since a_α is a positive integer this shows that $(f(\text{div } \mathcal{A}))$ has nonzero monomial $x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$ iff g has nonzero monomial $x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$. Thus $\text{mon}(f(\text{div } \mathcal{A})) = \text{mon}(g)$. It follows easily from the construction in Theorem 9 that the circuit C' will be monotone, if C is monotone. ■

Theorem 14. *Let $I = \langle m_1, \dots, m_k \rangle$ be a commutative monomial ideal in $\mathbb{Q}[x_{11}, \dots, x_{nn}]$, generated by $k = o(\frac{n}{\lg n})$ many monomials. Suppose C is a monotone circuit computing a polynomial f in $\mathbb{Q}[x_{11}, \dots, x_{nn}]$ such that the permanent $Perm_n = f(\text{mod } I)$ then $C^+(f) = 2^{\Omega(n)}$.*

Proof. Let X denote the set of variables $\{x_{11}, \dots, x_{nn}\}$. For each monomial m_i in the generating set for I write $m_i = \prod_{j=1}^{n} x_j^{e_{ij}}$, where e_{ij} are nonnegative integers.

Consider the language $L \subset X^n$ containing all strings m such that for each $i, 1 \leq i \leq k$ there is some $j \in [n]$ such that the number of occurrences of x_j in m is strictly less that e_{ij}. Notice that L is precisely $X^* \setminus I$. Clearly, the language L is *commutative*: if $m \in L$ then so is every reordering of the word m. It is easy to see that there is a *commutative* NFA $\mathcal{A}$ with $n^{O(k)} = 2^{o(n)}$ states such that $L = L(\mathcal{A})$ (the NFA is designed using counters for each i and guessed j). So we have $Perm_n = f(\text{div } \mathcal{A})$.

If possible assume that f can be computed by a monotone circuit C of size $2^{o(n)}$. By Lemma 3 there is a monotone circuit of size $2^{o(n)}$ computing a polynomial g such that $\text{mon}(g) = \text{mon}(f(\text{div } \mathcal{A})) = \text{mon}(Perm_n)$. We observe that the $2^{\Omega(n)}$ size lower bound proof for commutative circuits computing the permanent (specifically, the Jerrum-Snir work [JS82]) also imply the same lower bound for the polynomial g, because the coefficients do not play a role and $\text{mon}(g) = \text{mon}(Perm_n)$. This completes the proof. ■

Theorem 15. *There is a monomial ideal $I = \langle m_1, \cdots, m_t \rangle$ of $\mathbb{F}[X]$, where $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$,$t = O(n^3)$ and a polynomial-sized commutative formula $F(x_{11}, \cdots, x_{nn})$ such that*

$$Perm_n = F(\text{mod } I).$$

Proof. Let $F = \prod_{i=1}^{n} (x_{i1} + x_{i2} + \cdots + x_{in})$ and I be the monomial ideal generated by the set of monomials $\{x_{ik}x_{jk} \mid 1 \leq i, j, k \leq n\}$. Clearly, $Perm_n = F(\text{mod } I)$. ■

5 Intersecting Formulas and ABPs with NFAs

We now consider the *expressive power* of the intersecting and quotienting of formulas and ABPs by NFAs. It turns out that quotienting formulas with NFAs is powerful enough to express hard polynomials like the permanent.

Theorem 16. *There is are a polynomial $f \in \mathbb{F}\langle X \rangle$ given by an $O(n^2)$ size formula and an NFA $\mathcal{A}$ of size $O(n^3)$, for $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ such that*

$$Perm_n = f(\text{mod } \mathcal{A}).$$

Proof. Consider the polynomial $f = \prod_i \sum_j x_{ij}$. Let L be a set of all words $m \in X^n$ such that $m = ux_{ik}vx_{jk}w$ for some strings $u, v, w \in X^*$ and $1 \leq i, j, k \leq n$. It is easy to see that $L = L(\mathcal{A})$ for an NFA $\mathcal{A}$ with $O(n^3)$ states and $Perm_n = f(\text{mod } \mathcal{A})$. ■

In contrast to the power of quotienting by NFAs we can easily show that intersection by small NFAs cannot express the Permanent. Following Theorem is an easy application of the result on Hadamard product of ABPs (Theorem 7).

Theorem 17. *Let $f \in \mathbb{Q}\langle X\rangle$ and A is an NFA with $2^{o(n)}$ many states accepting a language $L \subseteq X^*$, $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ such that $Perm_n = f(\text{div } \mathcal{A})$, then $B(f) = 2^{\Omega(n)}$.*

Thus, hard polynomials can be expressed by quotienting a polynomial of small formula size with a small NFA, whereas we have exponential lower bound for the ABP complexity of any polynomial whose intersection with a subexponential-size NFA is the Permanent. We can ask the "dual" question about the complexity of identity testing for $f(\text{mod } \mathcal{A})$ and $f(\text{div } \mathcal{A})$ for NFAs $\mathcal{A}$.

Given a formula F computing a polynomial in $\mathbb{F}\langle X\rangle$ and an NFA $\mathcal{A}$ test if $f(\text{mod } \mathcal{A})$ is identically zero. This problem turns out to be intractable unlike for DFAs (see Theorem 1).

Theorem 18. *Given a formula F computing a polynomial $f \in \mathbb{Q}\langle Z\rangle$ and an NFA A accepting language $L(A) \subseteq Z^*$ then the problem of testing whether the polynomial $f(\text{mod } \mathcal{A})$ is identically zero is* coNP*-complete.*

Proof. We give a reduction from 3CNF-SAT to the complement of the problem. Let $S = C_1 \wedge C_2 \wedge \ldots \wedge C_t$ be a 3CNF formula where $C_i = c_{i1} \vee c_{i2} \vee c_{i3}$ for $1 \leq i \leq t$, and C_{ij}'s are from $\{w_1, \ldots, w_n\} \cup \{\neg w_1, \ldots, \neg w_n\}$. Let $f = \prod_{i=1}^{t} \sum_{j=1}^{3} z_{ij}$ where $z_{ij} = x_i$ if $c_{ij} = w_i$ and $z_{ij} = y_i$ if $c_{ij} = \neg w_i$ for $1 \leq i \leq n, 1 \leq j \leq 3$. Clearly, there is an $O(t)$ size formula F over indeterminates $Z = \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ for the polynomial f.

Let $L \subseteq Z^*$ be the set of all words of the form $m = ux_ivy_iw$ or $m = uy_ivx_iw$ for some $1 \leq i \leq n$. Clearly, there is an $O(n)$ size NFA A such that $L = L(\mathcal{A})$. Notice that the 3CNF formula S is satisfiable if and only if the polynomial $f(\text{mod } \mathcal{A})$ is *not* identically zero. Hence the given problem is coNP-hard. The problem is in coNP, since we can guess a monomial m, check in polynomial time that it has nonzero coefficient in f (by Proposition 3), and check in polynomial time that m is not in $L(\mathcal{A})$ using the NFA. ■

Finally, we consider the problem of testing if $f(\text{div } \mathcal{A})$ is identically zero for f given by an ABP and an NFA $\mathcal{A}$. We show that for the field of rationals, this problem is in deterministic polynomial time. It is an easy application of the result on Hadamard product of ABPs (Theorem 7).

Theorem 19. *Given an ABP P of size s computing polynomial $f \in \mathbb{Q}\langle X\rangle$ and an NFA A of size t then we can test whether the polynomial $f(\text{div } \mathcal{A})$ is identically zero, in deterministic polynomial (in s, t) time.*

References

[ABO99] E. ALLENDER, R. BEALS, AND M. OGIHARA, The complexity of matrix rank and feasible systems of linear equations, *Computational Complexity* , 8(2):99-126, 1999.

[AC75] A. V. AHO, M. J. CORASICK, Efficient String Matching: An Aid to Bibliographic Search. *Commun. ACM,* 18(6): 333-340, 1975

[AMS08] 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.

[Bh97] R. BHATIA, Matrix Analysis, Springer-Verlag Publishing Company, 1997.

[BW05] A. BOGDANOV, H. WEE More on Noncommutative Polynomial Identity Testing *In Proc. of 20th Annual Conference on Computational Complexity,* 92-99, 2005.

[CS04] 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.

[GJ79] M. R. GAREY, D. S. JOHNSON Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.*p. 228. ISBN 0-7167-1045-5,* 1979.

[HMU] J. E. HOPCROFT, R. MOTAWANI, J. D. ULLMAN, Introduction to Automata Theory Languages and Computation,*Second Edition,* Pearson Education Publishing Company.

[JS82] M. JERRUM, M. SNIR, Some Exact Complexity Results for Straight-Line Computations over Semirings.*J. ACM,* 29(3): 874-897, 1982.

[KI03] V. KABANETS, R. IMPAGLIAZZO, Derandomization of polynomial identity test means proving circuit lower bounds, *In Proc. of 35th ACM Sym. on Theory of Computing,*355-364,2003.

[KS01] A. KLIVANS, D. A. SPIELMAN, Randomness efficient identity testing of multivariate polynomials. *STOC 2001*, 216-223.

[MV97] M. MAHAJAN, V. VINAY, A Combinatorial Algorithm for the Determinant, *SODA 1997*, 730-738.

[MVV87] K. MULMULEY, U. V. VAZIRANI, V. V. VAZIRANI, Matching Is as Easy as Matrix Inversion *STOC 1987*, 345-354.

[N91] N. NISAN Lower bounds for noncommutative computation *In Proc. of 23rd ACM Sym. on Theory of Computing,* 410-418, 1991.

[RS05] R. RAZ, A. SHPILKA Deterministic polynomial identity testing in non commutative models *Computational Complexity,*14(1):1-19, 2005.

[T91] S. TODA, Counting Problems Computationally Equivalant to the Determinant, manuscript.

[V91] V. VINAY, Counting Auxiliary Pushdown Automata and Semi-unbounded Arithmetic Circuits, *Proc. 6th Structures in Complexity Theory Conference,*270-284, 1991.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/