Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 25 (2008)

New results on Noncommutative and Commutative Polynomial Identity Testing

V. Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan
Institute of Mathematical Sciences
C.I.T Campus,Chennai 600 113, India
{arvind,partham,srikanth}@imsc.res.in

Abstract

Using ideas from automata theory we design a new efficient (deterministic) identity test for the *noncommutative* polynomial identity testing problem (first introduced and studied in [RS05, BW05]). More precisely, given as input a noncommutative circuit $C(x_1, \cdots, x_n)$ computing a polynomial in $\mathbb{F}\{x_1, \cdots, x_n\}$ of degree d with at most t monomials, where the variables x_i are noncommuting, we give a deterministic polynomial identity test that checks if $C \equiv 0$ and runs in time polynomial in $d, n, |C|$, and t.

The same methods works in a black-box setting: Given a noncommuting black-box polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ of degree d with t monomials we can, in fact, reconstruct the entire polynomial f in time polynomial in n, d and t. Indeed, we apply this idea to the reconstruction of black-box noncommuting algebraic branching programs (the ABPs considered by Nisan in [N91] and Raz-Shpilka in [RS05]). Assuming that the black-box model allows us to query the ABP for the output at any given gate then we can reconstruct an (equivalent) ABP in deterministic polynomial time.

Finally, we turn to commutative identity testing and explore the complexity of the problem when the coefficients of the input polynomial come from an arbitrary finite commutative ring with unity whose elements are uniformly encoded as strings and the ring operations are given by an oracle. We show that several algorithmic results for polynomial identity testing over fields also hold when the coefficients come from such finite rings.

1 Introduction

Polynomial identity testing (denoted PIT) over fields is a well studied algorithmic problem: given an arithmetic circuit C computing a polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ over a field $\mathbb{F}$, the problem is to determine whether the polynomial computed by C is identically zero. The problem is also studied when the input polynomial f is given only via black-box access. I.e. we can evaluate it at any point in $\mathbb{F}^n$ or in $\mathbb{F}'^n$ for a field extension $\mathbb{F}'$ of $\mathbb{F}$. When f is given by a circuit the problem is in randomized polynomial time. Even in the black-box setting, when $|\mathbb{F}|$ is suitably larger than $\deg(f)$, the problem is in randomized polynomial time. A major challenge it to obtain deterministic polynomial time algorithms even for restricted versions of the problem. The results of Impagliazzo and Kabanets [KI03] 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 Σ gate as output [DS05, KS07].

As shown by Nisan [N91] noncommutative algebraic computation is somewhat easier to prove lower bounds. Using a rank argument Nisan has shown exponential size lower bounds for noncommutative formulas (and noncommutative algebraic branching programs) that compute the noncommutative permanent or

ISSN 1433-8092

determinant polynomials in the ring $\mathbb{F}\{x_1, \cdots, x_n\}$ where x_i are noncommuting variables. Thus, it seems plausible that identity testing in the noncommutative setting ought to be easier too. Indeed, Raz and Shpilka in [RS05] have shown that that for noncommutative formulas (and algebraic branching programs) there is a deterministic polynomial time algorithm for polynomial identity testing. However, for noncommutative circuits the situation is somewhat different. Bogdanov and Wee in [BW05] show using Amitsur-Levitzki's theorem that identity testing for *polynomial degree* noncommutative circuits is in randomized polynomial time. Basically, the Amitsur-Levitzki theorem allows them to randomly assign elements from a matrix algebra $M_k(\mathbb{F})$ for the noncommuting variables x_i, where $2k$ exceeds the degree of the circuit.

The main contribution of this paper is the use of ideas from *automata theory* to design new efficient (deterministic) polynomial identity tests for *noncommutative* polynomials. More precisely, given a noncommutative circuit $C(x_1, \cdots, x_n)$ computing a polynomial of degree d with t monomials in $\mathbb{F}\{x_1, \cdots, x_n\}$, where the variables x_i are noncommuting, we give a deterministic polynomial identity test that checks if $C \equiv 0$ and runs in time polynomial in $d, |C|, n$, and t. The main idea in our algorithm is to think of the noncommuting monomials over the x_i as words and to design finite automata that allow us to distinguish between different words. Then, using the connection between automata, monoids and matrix rings we are able to deterministically choose a relatively small number of matrix assignments for the noncommuting variables to decide if $C \equiv 0$. Thus, we are able to avoid using the Amitsur-Levitzki theorem. Indeed, using our automata theory method we can easily an alternative proof of (a weaker) version of Amitsur-Levitzki which is good enough for algorithmic purposes as in [BW05] for example.

Our method actually works in a black-box setting. In fact, given a noncommuting black-box polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ of degree d with t monomials, which we can evaluate by assigning matrices to x_i, we can reconstruct the entire polynomial f in time polynomial in n, d and t.

Furthermore, we also apply this idea to *black-box* noncommuting algebraic branching programs. We extend the result of Raz and Shpilka [RS05] by giving an efficient deterministic reconstruction algorithm for black-box noncommuting algebraic branching programs (wherein we are allowed to only query the ABP for input variables set to matrices of polynomial dimension). Our black-box model assumes that we can query for the output of *any gate* of the ABP, not just the output gate.

We now motivate and explain the other results in the paper. Recently, in [AM07] we studied PIT (the usual commuting variables setting) and its connection to the polynomial ideal membership problem. Although ideal membership is EXPSPACE-complete, there is an interesting similarity between the two problems which is the motivation for the present paper. Suppose $I \subset \mathbb{F}[x_1, \cdots, x_n]$ is an ideal generated by polynomials $g_1, \cdots, g_r \in \mathbb{F}[x_1, \cdots, x_k]$ and $f \in \mathbb{F}[x_1, \cdots, x_n]$. We observe that $f \in I$ if and only if f is identically zero in the ring $\mathbb{F}[x_1, \cdots, x_k]/I[x_{k+1}, \cdots, x_n]$. Thus, ideal membership is easily reducible to polynomial identity testing when the coefficient ring is $\mathbb{F}[x_1, \cdots, x_k]/I$. Consequently, identity testing for the coefficient ring $\mathbb{F}[x_1, \cdots, x_k]/I$ is EXPSPACE-hard even when the polynomial f is given explicitly as a linear combination of monomials.

This raises the question about the complexity of PIT for a polynomial ring $R[x_1, \cdots, x_n]$ where R is a commutative ring with unity. How does the complexity depend on the structure of the ring R? We give a precise answer to this question in this paper. We show that the algebraic structure of R is not important. It suffices that the elements of R have polynomial-size encoding, and w.r.t. this encoding the ring operations can be efficiently performed. This is in contrast to the ring $\mathbb{F}[x_1, \cdots, x_k]/I$: we have double exponential number of elements of polynomial degree in $\mathbb{F}[x_1, \cdots, x_k]$ and the ring operations in $\mathbb{F}[x_1, \cdots, x_k]/I$ are essentially ideal membership questions and hence computationally hard.

More precisely, we study polynomial identity testing for *finite* commutative rings R, where we assume that the elements of R are uniformly encoded as strings in $\{0, 1\}^m$ with two special strings encoding 0 and

1, and the ring operations are carried out by queries to the *ring oracle*.

1.1 Organization

The paper is organized as follows. In Section 2, we study the identity testing problem for the noncommutative circuits computing sparse and small degree polynomial. We also show an interpolation algorithm for such polynomials in the black-box setting. In Section 3, we show an interpolation algorithm for algebraic branching programs (ABP). In Section 4, we discuss the consequence of the derandomization of identity testing in the noncommutative model. Section 5 onwards, we discuss commutative identity testing over finite rings. In Section 5, we prove an analogue of Schwartz-Zippel Lemma for finite commutative rings with unity. The results for commutative identity testing (both in the black box model and circuit model) are given in the section 6.

2 Noncommutative Polynomial Identity Testing

Recall that an *arithmetic circuit* C over a field $\mathbb{F}$ is defined as follows: C takes as inputs, a set of indeterminates (either commuting or noncommuting) and elements from $\mathbb{F}$ as scalars. If f, g are the inputs of an addition gate, then the output will be $f + g$. Similarly for a multiplication gate the output will be fg. For noncommuting variables the circuit respect the order of multiplication. An arithmetic circuit is a formula if the fan-out of every gate is at most one.

Noncommutative identity testing was studied by Raz and Shpilka in [RS05] and Bogdanov and Wee in [BW05]. In the Bogdanov-Wee paper, they considered a polynomial f of small degree over $\mathbb{F}\{x_1, \cdots, x_n\}$, for a field $\mathbb{F}$, given by an arithmetic circuit. They were able to give a randomized polynomial time algorithm for the identity testing of f. The key feature of their algorithm was a reduction from noncommutative identity testing to commutative identity testing which is based on a classic theorem of Amitsur and Levitzki [AL50] about minimal identities for algebras.

Raz and Shpilka [RS05] give a deterministic polynomial-time algorithm for noncommutative formula identity testing by first converting a homogeneous formula into a noncommutative algebraic branching program (ABP), as done in [N91].

In this section we study the noncommutative polynomial identity testing problem. Using simple ideas from automata theory, we design a new deterministic identity test that runs in polynomial time if the input circuit is sparse and of small degree. Our algorithm works with only black-box access to the noncommuting polynomial, and we can even efficiently reconstruct the polynomial.

We will first describe the algorithm to test if a sparse polynomial of polynomial degree over noncommuting variables is identically zero. Then we give an algorithm that reconstructs this sparse polynomial. Though the latter result subsumes the former, for clarity of exposition, we describe both. Furthermore, we note that we can assume that the polynomial is given as an arithmetic circuit over a field $\mathbb{F}$.

In the case of commuting variables, [BT88] gives an interpolation algorithm that computes the given sparse polynomial, and thus can be used for identity testing. It is not clear how to generalize this algorithm to the noncommutative setting. Our identity testing algorithm evaluates the given polynomial at specific, well-chosen points in a matrix algebra (of polynomial dimension over the base field), such that any non-zero sparse polynomial is guaranteed to evaluate to a non-zero matrix at one of these points. The reconstruction algorithm uses the above identity testing algorithm as a subroutine in a prefix-based search to find all the monomials and their coefficients.

We now describe the identity testing algorithm informally. Our idea is to view each monomial as a short binary string. A sparse polynomial, hence, is given by a polynomial number of such strings (and the coefficients of the corresponding monomials). The algorithm proceeds in two steps; in the first step, we construct a small set of finite automata such that, given any small collection of short binary strings, at least one automaton from the set accepts exactly one string from this collection; in the second step, for each of the automata constructed, we construct a tuple of points over a matrix algebra over $\mathbb{F}$ such that the evaluation of any monomial at the tuple 'mimics' the run of the corresponding string on the automaton. Now, given any non-zero polynomial f of small degree with few terms, we are guaranteed to have constructed an automaton A 'isolating' a string from the collection of strings corresponding to monomials in f. We then show that evaluating f over the tuple corresponding to A gives us a non-zero output: hence, we can conclude f is non-zero. We now describe both algorithms formally.

2.1 Preliminaries

We first recall some standard automata theory notation (see, for example, [HU78]). Fix a finite automaton $A = (Q, \delta, q_0, q_f)$ which takes as input strings in $\{0,1\}^*$. Q is the set of states of A, $\delta : Q \times \{0,1\} \to Q$ is the transition function, and q_0 and q_f are the initial and final states respectively (throughout, we only consider automata with unique accepting states). For each letter $b \in \{0,1\}$, let $\delta_b : Q \to Q$ be the function 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 and is well-studied in automata theory: for example, see [Str94, page 55]. We now define the 0-1 matrix $M_b \in \mathbb{F}^{|Q|\times|Q|}$ as follows:

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

The matrix M_b is simply the adjacency matrix of the graph of the function δ_b. As the entries of M_b are only zeros and ones, we can consider M_b to be a matrix over any field $\mathbb{F}$.

Furthermore, for any $w = w_1 w_2 \cdots w_k \in \{0,1\}^*$ we define the matrix M_w 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 $|Q| \times |Q|$. For a string w, 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. It is easy to check that:

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

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.

2.2 The output of a circuit on an automaton

Now, we consider the ring $\mathbb{F}\{x_1, \cdots, x_n\}$ of polynomials with noncommuting variables $x_1, \cdots, x_n$ over a field $\mathbb{F}$. Let C be a noncommutative arithmetic circuit computing a polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$. Let d be an upper bound on the degree of f. We can consider monomials over the noncommuting variables $x_1, \cdots, x_n$ as strings over an alphabet of size n. For our construction in Section 2.3, 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 basically a unary encoding with delimiters. Clearly, each monomial over the x_i's of degree at most d maps uniquely to a binary string of length at most $d(n+2)$.

Let $A = (Q, \delta, q_0, q_f)$ be a finite automaton over the alphabet $\{0, 1\}$. With respect to automaton A we have matrices $M_{v_i} \in \mathbb{F}^{|Q|\times|Q|}$ as defined in Section 2.1, where each v_i is the binary string that encodes x_i. We are interested in the output matrix obtained when the inputs x_i to the circuit C are replaced by the matrices M_{v_i}. This output matrix is defined in the obvious way: the inputs are $|Q| \times |Q|$ matrices and we do matrix addition and matrix multiplication at each addition (resp. multiplication) of the circuit C. We define the *output of C on the automaton A* to be this output matrix M_{out}. Clearly, given circuit C and automaton A, the matrix M_{out} can be computed in time $\text{poly}(|C|, |A|, n)$.

We observe the following property: 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. This is important for us, since we are only interested in f. In particular, the output is always 0 when $f \equiv 0$.

More specifically, consider what happens when C computes a polynomial with a single term, say $f(x_1, \cdots, x_n) = cx_{j_1} \cdots x_{j_k}$, with a non-zero coefficient $c \in \mathbb{F}$. In this case, the output matrix M_{out} is clearly the matrix $cM_{v_{j_1}} \cdots M_{v_{j_k}} = cM_w$, where $w = v_{j_1} \cdots v_{j_k}$ is the binary string representing the monomial $x_{j_1} \cdots x_{j_k}$. Thus, by Equation 1 above, we see that the entry $M_{out}(q_0, q_f)$ is 0 when A rejects w, and c when A accepts w. In general, suppose C computes a polynomial $f = \sum_{i=1}^{t} c_i m_i$ with t nonzero terms, 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 $w_i = v_{i_1} \cdots v_{i_{d_i}}$ denote the binary string representing monomial m_i. Finally, let $S_A^f = \{i \in \{1, \cdots, t\} \mid A \text{ accepts } w_i\}$.

Theorem 2.1 *Given any arithmetic circuit C computing polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ and any finite automaton $A = (Q, \delta, q_0, q_f)$, then the output M_{out} of C on A is such that $M_{out}(q_0, q_f) = \sum_{i \in S_A^f} c_i$.*

Proof. The proof is an easy consequence of the definitions and the properties of the matrices M_w stated in Section 2.1. Note that $M_{out} = f(M_{v_1}, \cdots, M_{v_n})$. But $f(M_{v_1}, \cdots, M_{v_n}) = \sum_{i=1}^{s} c_i M_{w_i}$, where $w_i = v_{i_1} \cdots v_{i_{d_i}}$ is the binary string representing monomial m_i. By Equation 1, we know that $M_{w_i}(q_0, q_f)$ is 1 if w_i is accepted by A, and 0 otherwise. Adding up, we obtain the result. ■

We now explain the role of the automaton A in testing if the polynomial f computed by C is identically zero or not. Our basic idea is to try and design an automaton A that accepts exactly one word from among all the words that correspond to the non-zero terms in f. This would ensure that $M_{out}(q_0, q_f)$ is the non-zero coefficient of the monomial filtered out. More precisely, we will use the above theorem primarily in the following form, which we state as a corollary.

Corollary 2.2 *Given any arithmetic circuit C computing polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ and any finite automaton $A = (Q, \delta, q_0, q_f)$, 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 $\text{poly}(|C|, |A|, n)$.

Proof. Both points (1) and (2) are immediate consequences of the above theorem. The complexity of computing M_{out} easily follows from its definition. ■

Another interesting corollary to the above theorem is the following.

Corollary 2.3 *Given any arithmetic circuit C over $\mathbb{F}\{x_1, \cdots, x_n\}$, and any monomial m of degree d_m, we can compute the coefficient of m in C in time* $\text{poly}(|C|, d_m, n)$.

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 $|A| = O(nd_m)$. ■

Remark 2.4 *Observe that Corollary 2.3 is highly unlikely to hold in the commutative setting $\mathbb{F}[x_1, \cdots, x_n]$. For, in the commutative case, computing the coefficient of the monomial $x_1 \cdots x_n$ in even an arbitrary product of linear forms $\Pi_i \ell_i$ is at least as hard as the permanent problem over $\mathbb{F}$, which is $\#$P-complete when $\mathbb{F} = \mathbb{Q}$.*

Remark 2.5 *Corollary 2.2 can also be used to give an independent proof of a weaker form of the result of Amitsur and Levitzki that is stated in Lemma A.4. In particular, it is easy to see that the algebra $M_d(\mathbb{F})$ of $d \times d$ matrices over the field $\mathbb{F}$ does not satisfy any nontrivial identity of degree $< d$. To prove this, we will consider noncommuting monomials as strings directly over the n letter alphabet $\{x_1, \cdots, x_n\}$. Suppose $f = \sum_{i=1}^{t} c_i m_i \in \mathbb{F}\{x_1, \cdots, x_n\}$ is a nonzero polynomial of degree $< d$. Clearly, we can construct an automaton B over the alphabet $\{x_1, \cdots, x_n\}$ that accepts exactly one string, namely one nonzero monomial, say m_{i_0}, of f and rejects all the other strings over $\{x_1, \cdots, x_n\}$. Also, B can be constructed with at most d states. Now, consider the output M_{out} of any circuit computing f on B. By Corollary 2.2 the output matrix is non-zero, and this proves the result.*

2.3 Construction of finite automata

We begin with a useful definition.

Definition 2.6 *Let W be a finite set of binary strings and $\mathcal{A}$ be a finite family of finite automata over the binary alphabet $\{0, 1\}$.*

- *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 for every subset $W = \{w_1, \cdots, w_s\}$ of s many binary strings, each of length at most m, there is a $A \in \mathcal{A}$ such that A is isolating for W.*

Fix parameters $m, s \in \mathbb{N}$. Our first aim is to construct an (m, s) isolating family of automata $\mathcal{A}$, where both $|\mathcal{A}|$ and the size of each automaton in $\mathcal{A}$ is polynomially bounded in size. Then, combined with Corollary 2.2 we will be able to obtain deterministic identity testing and interpolation algorithms in the sequel.

Recall that we only deal with finite automata that have unique accepting states. In what follows, for a string $w \in \{0, 1\}^*$, we denote by n_w the positive integer represented by the binary numeral $1w$. 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 p states and 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 $\text{poly}(m, s)$. In the following lemma we show that $\mathcal{A}$ is an (m, s)-isolating automata family.

Lemma 2.7 *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 $(m+2)\binom{s}{2} = N - 1$. This concludes the proof. ∎

We note that the above (m, s)-isolating family $\mathcal{A}$ can clearly be constructed in time $\text{poly}(m, s)$.

2.4 The identity testing algorithm

We now describe the identity testing algorithm. Let C be the input circuit computing a polynomial f over $\mathbb{F}\{x_1, \cdots, x_n\}$. Let t be an upper bound on the number of monomials in f, and d be an upper bound on the degree of f. As in Section 2.2, we represent monomials over $x_1, \cdots, x_n$ as binary strings. Every monomial in f is represented by a string of length at most $d(n+2)$.

Our algorithm proceeds as follows: Using the construction of Section 2.3, we compute a family $\mathcal{A}$ of automata such that $\mathcal{A}$ is isolating for any set W with at most t strings of length at most $d(n+2)$ each. For each $A \in \mathcal{A}$, the algorithm computes the output M_{out} of C on A. If $M_{out} \neq 0$ for any A, then the algorithm concludes that the polynomial computed by the input circuit is not identically zero; otherwise, the algorithm declares that the polynomial is identically zero.

The correctness of the above algorithm is almost immediate from Corollary 2.2. If the polynomial is identically zero, it is easy to see that the algorithm outputs the correct answer. If the polynomial is nonzero, then by the construction of $\mathcal{A}$, we know that there exists $A \in \mathcal{A}$ such that A accepts precisely one of the strings corresponding to the monomials in f. Then, by Corollary 2.2, the output of C on A is nonzero. Hence, the algorithm correctly deduces that the polynomial computed is not identically zero.

As for the running time of the algorithm, it is easy to see that the family of automata $\mathcal{A}$ can be constructed in time $\text{poly}(d, n, t)$. Also, the matrices M_{v_i} for each A (all of which are of size $\text{poly}(d, n, t)$) can be constructed in polynomial time. Hence, the entire algorithm runs in time $\text{poly}(|C|, d, n, t)$. We have proved the following theorem:

Theorem 2.8 *Given any arithmetic circuit C with the promise that C computes a polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ of degree d with at most t monomials, we can check, in time* $\text{poly}(|C|, d, n, t)$*, if f is identically zero. In particular, if f is sparse and of polynomial degree, then we have a deterministic polynomial time algorithm.*

In the case of arbitrary noncommutative arithmetic circuits, [BW05] gives a randomized exponential time algorithm for the identity testing problem. Their algorithm is based on the Amitsur-Levitzki theorem, which forces the identity test to randomly assign exponential size matrices for the noncommuting variables since the circuit could compute an exponential degree polynomial. However, notice that Theorem 2.8 gives a deterministic exponential-time algorithm under the additional restriction that the input circuit computes a polynomial with at most *exponentially* many monomials. In general, a polynomial of exponential degree can have a double exponential number of terms.

2.5 Interpolation of noncommutative polynomials

We now describe an algorithm that efficiently computes the noncommutative polynomial given by the input circuit. Let C, f, t and d be as in Section 2.4. Let W denote the set of all strings corresponding to monomials with non-zero coefficients in f. For all binary strings w, let A_w denote any standard automaton that accepts w and rejects all other strings. For any automaton A and string w, we let $[A]_w$ denote the automaton that accepts those strings that are accepted by A and in addition, contain w as a prefix. For a set of finite automata $\mathcal{A}$, let $[\mathcal{A}]_w$ denote the set $\{[A]_w \mid A \in \mathcal{A}\}$.

We now describe a subroutine `Test` that takes as input an arithmetic circuit C and a set of finite automata $\mathcal{A}$ and returns a field element $\alpha \in \mathbb{F}$. The subroutine `Test` will have the following properties:

(P1) If $\mathcal{A}$ is isolating for W, the set of strings corresponding to monomials in f, then $\alpha \neq 0$.

(P2) In the special case when $|\mathcal{A}| = 1$, and the above holds, then α is in fact the coefficient of the isolated monomial.

(P3) If no $A \in \mathcal{A}$ accepts any string in W, then $\alpha = 0$.

We now give the easy description of `Test(`C`,`$\mathcal{A}$`)`:

For each $A \in \mathcal{A}$, the subroutine `Test` computes the output matrix M^A_{out} of C on A. If there is an $A \in \mathcal{A}$ such that $M^A_{out}(q^A_0, q^A_f) \neq 0$, then for the first such automaton $A \in \mathcal{A}$, `Test` returns the scalar $\alpha = M^A_{out}(q^A_0, q^A_f)$. Here, notice that q^A_0, q^A_f denote the initial and final states of the automaton A. If there is no such automaton $A \in \mathcal{A}$ is found, then the subroutine returns the scalar 0.

It follows directly from Corollary 2.2 that `Test` has Properties (P1)-(P3). Furthermore, clearly `Test` runs in time poly$(|C|, ||\mathcal{A}||)$, where $||\mathcal{A}||$ denotes the sum of the sizes of the automata in $\mathcal{A}$.

Let $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ denote the noncommuting polynomial computed by the input circuit C. We now describe a recursive prefix-search based algorithm `Interpolate` that takes as input the circuit C and a binary string u, and computes all those monomials of f (along with their coefficients) which contain u as a prefix when encoded as strings using our encoding $x_i \mapsto v_i = 01^i0$. Clearly, in order to obtain all monomials of f with their coefficients, it suffices to run this algorithm with $u = \epsilon$, the empty string.

In what follows, let $\mathcal{A}_0$ denote the (m, s)-isolating automata family $\{A_{p,i}\}$ as constructed in Section 2.3 with parameters $m = d(n + 2)$ and $s = t$. As explained in Section 2.3, we can compute $\mathcal{A}_0$ in time poly(d, n, t).

Suppose f is the polynomial computed by the circuit C. We now describe the algorithm `Interpolate(`C`,`u`)` formally (Algorithm 1).

The correctness of this algorithm is clear from the correctness of the `Test` subroutine and Lemma 2.7. To bound the running time, note that the algorithm never calls `Interpolate` on a string u unless u is the prefix of some string corresponding to a monomial. Hence, the algorithm invokes `Interpolate` for at most $O(td(n + 2))$ many prefixes u. Since $||[\mathcal{A}_0]_{u0}||$ and $|A_u|$ are both bounded by poly(d, n, t) for all prefixes u, it follows that the running time of the algorithm is poly$(|C|, d, n, t)$. We summarize this discussion in the following theorem.

Theorem 2.9 *Given any arithmetic circuit C computing a polynomial $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ of degree at most d and with at most t monomials, we can compute all the monomials of f, and their coefficients, in time*

Algorithm 1 The Interpolation algorithm

```
1: procedure Interpolate($C$,$u$)
2:     $\alpha, \alpha', \alpha'' \leftarrow 0$.
3:     $\alpha \leftarrow$ Test$(C, \{A_u\})$          ▷ $A_u$ is the standard automaton that accepts only $u$
4:     if $\alpha = 0$ then
5:         Break.          ▷ $u$ can not corresponds to a monomial of $f$
6:     else
7:         Output $(u, \alpha)$.          ▷ $u$ is the binary encoding of a monomial of $f$ with coefficient $\alpha$
8:     end if
       Now the algorithm find all monomials (along with their coefficient)
       containing $u0$ or $u1$ as prefix in the binary encoding.
9:     if $|u| = d(n+2)$ then
10:        Stop.
11:    else
12:        $\alpha' \leftarrow$Test$(C, [\mathcal{A}_0]_{u0})$, $\alpha'' \leftarrow$Test$(C, [\mathcal{A}_0]_{u1})$.
13:    end if
14:    if $\alpha' \neq 0$ then
15:        Interpolate$(C, u0)$.          ▷ There is some monomial in $C$ extending $u0$
16:    end if
17:    if $\alpha'' \neq 0$ then
18:        Interpolate$(C, u1)$.          ▷ There is some monomial in $C$ extending $u1$
19:    end if
20: end procedure
```

poly$(|C|, d, n, t)$. *In particular, if C computes a sparse polynomial f of polynomial degree, then f can be reconstructed in polynomial time.*

3 Interpolation of Algebraic Branching Programs over noncommuting variables

In this section, we study the interpolation problem for black-box Algebraic Branching Programs (ABP) computing a polynomial in the noncommutative ring $\mathbb{F}\{x_1, \cdots, x_n\}$. We are given as input an ABP (defined below) P in the black-box setting, and our task is to output an ABP P' that computes the same polynomial as P. To make the task feasible in the black-box setting, we assume that we are allowed to evaluate P at any of its intermediate gates.

We first observe that all the results in Section 2 hold under the assumption that the input polynomial f is allowed only *black-box access*. In the noncommutative setting, we shall assume that the black-box access allows the polynomial to be evaluated for input values from an arbitrary matrix algebra over the base field $\mathbb{F}$. It is implicit here that the cost of evaluation is polynomial in the dimension of the matrices. Note that this is a reasonable noncommutative black-box model, because if we can evaluate f only over $\mathbb{F}$ or any commutative extension of $\mathbb{F}$, then we cannot distinguish the non-commutative polynomial represented by f from the corresponding commutative polynomial. We state the black-box version of our results below.

Theorem 3.1 (Similar to Theorem 2.1) *Given black-box access to any polynomial $f = \sum_{i=1}^{t} c_i m_i \in \mathbb{F}\{x_1, \cdots, x_n\}$ and any finite automaton $A = (Q, \delta, q_0, q_f)$, then the output M_{out} of f on A is such that $M_{out}(q_0, q_f) = \sum_{i \in S_A^f} c_i$, where $S_A^f = \{i \mid 1 \leq i \leq t$ and A accepts the string w_i corresponding to $m_i\}$*

Here the output of polynomial f on A is defined analogously to the output of a circuit on A in Section 2.2.

Corollary 3.2 (Similar to Corollary 2.3) *Given black-box access to a polynomial f in $\mathbb{F}\{x_1, \cdots, x_n\}$, and any monomial m of degree d_m, we can compute the coefficient of m in f in time* poly(d_m, n).

Finally we have,

Theorem 3.3 (Similar to Theorem 2.9) *Given black-box access to a polynomial f in $\mathbb{F}\{x_1, \cdots, x_n\}$ of degree at most d and with at most t monomials, we can compute all the monomials of f, and their coefficients, in time* poly(d, n, t). *In particular, if f is a sparse polynomial of polynomial degree, then it can be reconstructed in polynomial time.*

Our interpolation algorithm for noncommutative ABPs is motivated by Raz and Shpilka's [RS05] algorithm for identity testing of ABPs over noncommuting variables. Our algorithm interpolates the given ABP layer by layer using ideas developed in Section 2 (principally Corollary 3.2).

Definition 3.4 *[N91, RS05] 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.*

Notice that an ABP with no edge between two vertices u and v on levels i and $i+1$ is equivalent to an ABP with an edge from u to v labeled with the zero linear form. Thus, without loss of generality, we assume that in the given ABP there is an edge between every pair of vertices on adjacent levels.

As mentioned before, we will assume black-box access to the input ABP P where we can evaluate the polynomial computed by P at any of its gates over arbitrary matrix rings over $\mathbb{F}$. In order to specify the gate at which we want the output, we index the gates of P with a layer number and a gate number (in the layer).

Based on [RS05], we now define a ***Raz-Shpilka basis*** for the level i of the ABP. Let the number of nodes at the i-th level be G_i and let $\{p_1, p_2, \cdots, p_{G_i}\}$ be the polynomials computed at the nodes. We will identify this set of polynomials with the $G_i \times n^i$ matrix M_i where the columns of M_i are indexed by n^i different monomials of degree i, and the rows are indexed by the polynomials p_j. The entries of the matrix M_i are the corresponding polynomial coefficients. A Raz Shpilka basis is a set of at most G_i linearly independent column vectors of M_i that generates the entire column space. Notice that every vector in the basis is identified by a monomial.

In the algorithm we need to compute a Raz-Shpilka basis at every level of the ABP. Notice that at the level 0 it is trivial to compute such a basis. Inductively assume we can compute such a basis at the level i. Denote the basis by $B_i = \{v_1, v_2, \cdots, v_{k_i}\}$ where $v_j \in \mathbb{F}^{G_i}$, and $k_i \leq G_i$. Assume that the elements of this basis corresponds to the monomials $\{m_1, m_2, \cdots, m_{k_i}\}$. We compute a Raz Shpilka basis at the level $i+1$ by computing the column vectors corresponding to the set of monomials $\{m_j x_s\}_{j \in [k_i], s \in [n]}$ in M_{i+1} and then extracting the linear independent vectors out of them. Computing these column vectors requires the computation of the coefficients of these monomials, which can be done in polynomial time using the Corollary 3.2. Notice that we also know the monomials that the elements of this basis correspond to.

We now describe the interpolation algorithm formally. As mentioned before, we will construct the output ABP P' layer by layer such that every gate of P' computes the same polynomial as the corresponding gate in P. Clearly, this task is trivial at level 0.

Assume that we have completed the construction up to level $i < d$. We now construct level $i+1$. This only involves computation of the linear forms between level i and level $i+1$. Hence, there are $k_i \leq G_i$ vectors in the Raz-Shpilka basis at the ith level. Let the monomials corresponding to these vectors be $B = \{m_1, \cdots, m_{k_i}\}$. Fix any gate u at level $i+1$ in P, and let p_u be the polynomial compute at this gate in P. Clearly,

$$p_u = \sum_{j=1}^{G_i} p_j \ell_j$$

where p_j is the polynomial computed at the jth gate at level i, and ℓ_j is the linear form labeling the edge between the jth gate at level i and u.

We have,

$$\begin{aligned}
p_u &= \sum_{j=1}^{G_i} p_j \ell_j \\
&= \sum_{j=1}^{G_i} \left(\sum_{m:|m|=i} c_m^{(j)} m \right) \left(\sum_{s=1}^{n} a_s^{(j)} x_s \right) \\
&= \sum_{m:|m|=i,s} m x_s \left(\sum_{j=1}^{G_i} c_m^{(j)} a_s^{(j)} \right) \\
&= \sum_{m:|m|=i,s} m x_s \langle c_m, a_s \rangle
\end{aligned}$$

where c_m and a_s denote the vectors of field elements $(c_m^{(j)})_j$ and $(a_s^{(j)})_j$ respectively. Note that a_s denotes a vector of unknowns that we need to compute. Each monomial mx_s in the above equation gives us a linear constraint on a_s. However, this system of constraints is exponential in size. To obtain a feasible solution for $\{a_s\}_{s\in[n]}$, we observe that it is sufficient to satisfy the constraints corresponding only to monomials mx_s where $m \in B$. All other constraints are simply linear combinations of these and are thus automatically satisfied by any solution to these.

Now, for $m \in B$ and $s \in \{1, \cdots, n\}$, we compute the coefficients of mx_s in p_u and those of m in each of the p_i's using the algorithm of Corollary 3.2. Hence, we have all the linear constraints we need to solve for $\{a_s\}_{s\in[n]}$. Firstly, note that such a solution exists, since the linear forms in the black box ABP P give us such a solution. Moreover, any solution to this system of linear equations generates the same polynomial p_u at gate u. Hence, we can use any solution to this system of linear equations as our linear forms. We perform this computation for all gates u at the $i+1$st level. The final step in the iteration is to compute the Raz-Shpilka basis for the level $i+1$.

We can use induction on the level numbers to argue correctness of the algorithm. From the input black-box ABP P, for each level k, let $P_{jk}, 1 \leq j \leq G_k$ denote the algebraic branching programs computed by P with output gate as gate j in level k. Assume, as induction hypothesis, that the algorithm has computed linear forms for all levels upto level i and, furthermore, that the algorithm has a correct Raz-Shpilka basis for all levels upto level i. This gives us a reconstructed ABP P' upto level i with the property, for $1 \leq k \leq i$, each ABP $P'_{jk}, 1 \leq j \leq G_k$ computes the same polynomials as the corresponding $P_{jk}, 1 \leq j \leq G_k$, where P'_{jk} is obtained from P' by designating gate j at level k as output gate. Under this induction hypothesis, it is clear that our interpolation algorithm will compute a correct set of linear forms between levels i and $i+1$. Consequently, the algorithm will correctly reconstruct an ABP P' upto level $i+1$ along with a corresponding Raz-Shpilka basis for that level.

We can now summarize the result in the following theorem.

Theorem 3.5 *Let P be an ABP of size s and depth d over $\mathbb{F}\{x_1, x_2, \cdots, x_n\}$ given by black-box access that allows evaluation of any gate of P for inputs x_i chosen from a matrix algebra $M_k(\mathbb{F})$ for a polynomially bounded value of k. Then in deterministic time* $\text{poly}(d, s, n)$*, we can compute an ABP P' such that P' evaluates to the same polynomial as P.*

4 Noncommutative identity testing and circuit lower bounds

In Section 2 we gave a new deterministic identity test for noncommuting polynomials which runs in polynomial time for sparse polynomials of polynomially bounded degree.

However, the real problem of interest is identity testing for polynomials given by small degree noncommutative circuits for which Bogdanov and Wee [BW05] give an efficient randomized test. When the noncommutative circuit is a formula, Raz and Shpilka [RS05] have shown that the problem is in deterministic polynomial time. Their method uses ideas from Nisan's lower bound technique for noncommutative formulae [N91].

How hard would it be to show that noncommutative PIT is in deterministic polynomial time for *circuits* of polynomial degree? In the commutative case, Impagliazzo and Kabanets [KI03] have shown that derandomizing PIT implies circuit lower bounds. It implies that either NEXP $\not\subseteq$ P/poly or the integer Permanent does not have polynomial-size arithmetic circuits.

We observe that this result also holds in the noncommutative setting. I.e., if noncommutative PIT has a deterministic polynomial-time algorithm then either NEXP $\not\subseteq$ P/poly or the *noncommutative* Permanent function does not have polynomial-size noncommutative circuits.

As noted, in some cases noncommutative circuit lower bounds are easier to prove than for commutative circuits. Nisan [N91] has shown exponential-size lower bounds for noncommutative formula size and further results are known for pure noncommutative circuits [N91, RS05]. However, proving superpolynomial size lower bounds for general noncommutative circuits computing the Permanent has remained an open problem.

The noncommutative Permanent function $Perm(x_1, \cdots, x_n) \in R\{x_1, \cdots, x_n\}$ is defined as

$$Perm(x_1, \cdots, x_n) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)},$$

where the coefficient ring R is any commutative ring with unity. Specifically, for the next theorem we choose $R = \mathbb{Q}$.

Theorem 4.1 *If* PIT *for noncommutative circuits of polynomial degree $C(x_1, \cdots, x_n) \in \mathbb{Q}\{x_1, \cdots, x_n\}$ is in deterministic polynomial-time then either* NEXP $\not\subseteq$ P/poly *or the* noncommutative *Permanent function does not have polynomial-size noncommutative circuits.*

Proof. Suppose NEXP $\subseteq$ P/poly. Then, by the main result of [IKW02] we have NEXP $=$ MA. Furthermore, by Toda's theorem MA $\subseteq$ P$^{Perm_{\mathbb{Z}}}$, where the oracle computes the integer permanent. Now, assuming PIT for noncommutative circuits of polynomial degree is in deterministic polynomial-time we will show that the (noncommutative) Permanent function does not have polynomial-size noncommutative circuits. Suppose to the contrary that it does have polynomial-size noncommutative circuits. Clearly, we can use it to compute the integer permanent as well. Furthermore, as in [KI03] we notice that the noncommutative $n \times n$ Permanent is also uniquely characterized by the identities $p_1(x) \equiv x$ and $p_i(X) = \sum_{j=1}^{i} x_{1j} p_{i-1}(X_j)$ for $1 < i \leq n$, where X is a matrix of i^2 noncommuting variables and X_j is its j-th minor w.r.t. the first row. I.e. if arbitrary polynomials $p_i, 1 \leq i \leq n$ satisfies these n identities over *noncommuting* variables $x_{ij}, 1 \leq i, j \leq n$ if and only if p_i computes the $i \times i$ permanent of noncommuting variables. The rest of the proof is exactly as in Impagliazzo-Kabanets [KI03]. We can easily describe an NP machine to simulate a P$^{Perm_{\mathbb{Z}}}$ computation. The NP machine guesses a polynomial-size noncommutative circuit for $Perm$ on $m \times m$ matrices, where m is a polynomial bound on the matrix size of the queries made. Then the NP verifies that the circuit computes the permanent by checking the m *noncommutative* identities it must satisfy. This can be done in deterministic polynomial time by assumption. Finally, the NP machines uses the circuit to answer all the integer permanent queries. Putting it together, we get $\mathrm{NEXP} = \mathrm{NP}$ which contradicts the nondeterministic time hierarchy theorem. ■

5 Schwartz-Zippel lemma over finite rings

In this section we give a generalization of Schwartz-Zippel Lemma to finite commutative rings and apply it for identity testing of black-box polynomials in $R[x_1, \cdots, x_n]$, where R is a finite commutative ring with unity whose elements are uniformly encoded by strings from $\{0,1\}^m$ with a special string e denote unity, and the ring operations are performed by a ring oracle.

We recall some facts about finite commutative rings [Mc74, AM69]. A commutative ring R with unity is a *local ring* if R has a *unique* maximal ideal M. An element $r \in R$ is *nilpotent* if $r^n = 0$ for some positive integer n. An element $r \in R$ is a *unit* if it is invertible. I.e. $rr' = 1$ for some element $r' \in R$. Any element of a finite local ring is either a nilpotent or a unit. An ideal I is a *prime ideal* of R if $ab \in I$ implies either $a \in I$ or $b \in I$. For finite commutative rings, prime ideals and maximal ideals coincide. These facts considerably simplify the study of finite commutative rings (in contrast to infinite rings).

The *radical* of a finite ring R denoted by $\mathrm{Rad}(R)$ is defined as the set of all nilpotent elements, i.e

$$\mathrm{Rad}(R) = \{r \in R \mid \exists n > 0 \text{ s.t } r^n = 0\}$$

The radical $\mathrm{Rad}(R)$ is an ideal of R, and it is the unique maximum ideal if R is a local ring. Let m denote the least positive integer such that for every nilpotent $r \in R$, $r^m = 0$, i.e $(\mathrm{Rad}(R))^m = 0$. Let R be any finite commutative ring with unity and $\{P_1, P_2, \cdots, P_\ell\}$ by the set of all maximal ideals of R. Let R_i denote the quotient ring R/P_i^m for $1 \le i \le \ell$. Then, it is easy to see that each R_i is a local ring and P_i/P_i^m is the unique maximal ideal in R_i. We recall the following structure theorem for finite commutative rings.

Theorem 5.1 ([Mc74], Theorem VI.2, page 95) *Let R be a finite commutative ring. Then R decomposes (up to order of summands) uniquely as a direct sum of local rings. More precisely*

$$R \cong R_1 \oplus R_2 \oplus \cdots \oplus R_\ell,$$

via the map $\phi(r) = (r + P_1^m, r + P_2^m, \cdots, r + P_\ell^m)$, where $R_i = R/P_i^m$ and $P_i, 1 \le i \le \ell$ are all the maximal ideals of R.

It is easy to see that ϕ is a homomorphism with trivial kernel. The isomorphism ϕ naturally extends to the polynomial ring $R[x_1, x_2, \cdots, x_n]$, and gives the isomorphism $\hat{\phi} : R[x_1, x_2, \cdots, x_n] \rightarrow \oplus_{i=1}^{\ell} R_i[x_1, x_2, \cdots, x_n]$.

5.1 The Schwartz-Zippel lemma

We observe the following easy fact about zeros of univariate polynomials over finite commutative rings with unity.

Proposition 5.2 *Let R be a finite commutative ring with unity 1 containing an integral domain D such that $1 \in D$. If $f \in R[x]$ is a nonzero polynomial of degree d then $f(a) = 0$ for at most d distinct values of $a \in D$.*

Proof. Notice that D is a finite integral domain as it is contained in the finite ring R. Thus, D must in fact be a finite field. Now, suppose $a_1, a_2, \cdots, a_{d+1} \in D$ are distinct points such that $f(a_i) = 0, 1 \leq i \leq d+1$. Then we can write $f(x) = (x - a_1)q(x)$ for $q(x) \in R[x]$. Dividing $q(x)$ by $x - a_2$ yields $q(x) = (x - a_2)q'(x) + q(a_2)$, for some $q'(x) \in R[x]$. Thus, $f(x) = (x - a_1)(x - a_2)q'(x) + (x - a_1)q(a_2)$. Putting $x = a_2$ in this equation gives $(a_2 - a_1)q(a_2) = 0$. But $(a_2 - a_1)$ is nonzero in the field D and hence is invertible and $(a_2 - a_1)^{-1}(a_2 - a_1) = 1$. Multiplying by $(a_2 - a_1)^{-1}$ we get $q(a_2) = 0$. Consequently, $f(x) = (x - a_1)(x - a_2)q'(x)$ in $R[x]$. Applying this argument successively for the other a_i finally yields $f(x) = g(x)\prod_{i=1}^{d+1}(x - a_i)$ for some nonzero polynomial $g(x) \in R[x]$. Since $\prod_{i=1}^{d+1}(x - a_i)$ is a monic polynomial, this forces $\deg(f) \geq d + 1$ which is a contradiction. ■

Using Proposition 5.2 we describe an easy generalization of the Schwarz-Zippel lemma to finite commutative rings with unity containing integral domains.

Lemma 5.3 *Let R be a finite commutative ring with 1 such that R contains an integral domain D with $1 \in D$. Let $g \in R[x_1, x_2, \cdots, x_n]$ be any polynomial of degree at most d. If $g \not\equiv 0$, then for any subset A of D we have*

$$\text{Prob}_{a_1 \in A, \cdots, a_n \in A}[g(a_1, a_2, \cdots, a_n) = 0] \leq \frac{d}{|A|}.$$

Proof. We need to show that the number of n-tuples $(a_1, \cdots, a_n) \in A^n$ such that $g(a_1, a_2, \cdots, a_n) = 0$ is at most $d|A|^{n-1}$. The proof is by induction on n. The base case $n = 1$ involves a univariate polynomial $g(x_1)$ in $R[x_1]$ and follows directly from Proposition 5.2. As induction hypothesis suppose the lemma holds for multivariate polynomials in $n - 1$ indeterminates. Write $g(x_1, x_2, \cdots, x_n)$ as $g(x_1, x_2, \cdots, x_n) = \sum_{i=0}^{k} x_n^i g_i(x_1, x_2, \cdots, x_{n-1})$, where $k \leq d$ is the largest exponent of x_n in g with nonzero coefficient g_k, and each $g_i \in R[x_1, x_2, \cdots, x_{n-1}]$. Since $g_k \neq 0$ and $\deg(g_k) \leq d - k$, by the induction hypothesis there are at most $(d-k)|A|^{n-2}$ tuples $(a_1, \cdots, a_{n-1}) \in A^{n-1}$ such that $g_k(a_1, \cdots, a_{n_1}) = 0$. Let

$$E_1 = \{(a_1, \cdots, a_n) \mid g_k(a_1, \cdots, a_{n_1}) = 0\}.$$

Then $|E_1| \leq (d-k)|A|^{n-1}$. Now consider the univariate polynomial $\hat{g}(x_n) = \sum_{i=0}^{k} x_n^i g_i(a_1, a_2, \cdots, a_{n-1})$ in $R[x_n]$ for $(a_1, \cdots, a_{n-1}) \in A^{n-1}$. If $g_k(a_1, a_2, \cdots, a_{n-1})$ is nonzero then $\hat{g}(x_n)$ is a nonzero polynomial. Let

$$E_2 = \{(a_1, \cdots, a_n) \mid \hat{g}(x_n) \neq 0 \text{ and } \hat{g}(a_n)) = 0\}.$$

It follows from Proposition 5.2 that $|E_2| \leq k|A|^{n-1}$.

Since $\{(a_1, \cdots, a_n) \mid g(a_1, \cdots, a_n) = 0\} \subseteq E_1 \cup E_2$, we obtain the required bound

$$|\{(a_1, \cdots, a_n) \mid g(a_1, \cdots, a_n) = 0\}| \leq |E_1| + |E_2| \leq (d-k)|A|^{n-1} + k|A|^{n-1} = d|A|^{n-1}.$$

This completes the proof. ■

In general Lemma 5.3 is not applicable, because the given finite ring R may not contain a large finite field $\mathbb{F}$ containing 1. We explain how to get around this problem for finite commutative local rings. Because of the structure theorem, it suffices to consider local rings.

Let R be a finite local ring with unity given by a ring oracle. Suppose the characteristic of R is p^α for a prime p. If the elements of R are encoded in $\{0,1\}^m$ then 2^m upper bounds the size of R. Let $M > 2^m$, to be fixed later in the analysis. Let $U = \{ce \mid 0 \leq c \leq M\}$, where e denotes the unity of R. We will argue that, for a suitable M, if we sample ce uniformly from U then $(c \bmod p)\ e$ is almost uniformly distributed in $\mathbb{Z}_p e$. Pick x uniformly at random from $\mathbb{Z}_M$ and output xe. Let $a \in \mathbb{Z}_p$ and $P = \text{Prob}[x \equiv a \pmod p]$. The x for which $x \equiv a \pmod p$ are $a, a+p, \cdots, a+p\lfloor \frac{M-a}{p} \rfloor$. Let $M' = \lfloor \frac{M-a}{p} \rfloor$. Then $P = M'+1/M \leq \frac{1}{p}(1+\frac{2^m}{M})$. Clearly, $P \geq \frac{1}{p}(1-\frac{2^m}{M})$. For a given $\epsilon > 0$, choose $M = 2^{m+1}/\epsilon$. Then $\frac{1-\epsilon/2}{p} \leq P \leq \frac{1+\epsilon/2}{p}$. So $(x \bmod p)e$ is $\frac{\epsilon}{2}$-uniformly distributed in $\mathbb{Z}_p e$.

Lemma 5.4 *Let R be a finite local commutative ring with unity 1 and of characteristic p^α for a prime p. The elements of R are encoded using binary strings of length m. Let $g \in R[x_1, x_2, \cdots, x_n]$ be a polynomial of degree at most d and $\epsilon > 0$ be a given constant. If $g \not\equiv 0$, then*

$$\text{Prob}_{a_1 \in U, \cdots, a_n \in U}[g(a_1, a_2, \cdots, a_n) = 0] \leq \frac{d}{p}(1 + \frac{\epsilon}{2}),$$

where $U = \{ce \mid 0 \leq c \leq M\}$ and $M > 2^{m+1}/\epsilon$.

Proof. Consider the following tower of ideals inside R :

$$R \supseteq pR \supseteq p^2R \supseteq \cdots \supseteq p^\alpha R = \{0\}.$$

Let k be the integer such that $g \in p^k R[x_1, \cdots, x_n] \setminus p^{k+1}R[x_1, \cdots, x_n]$. Write $g = p^k \hat{g}$. Consider the ring, $\hat{I} = \{r \in R \mid p^k r = 0\}$. Clearly, $\hat{I}$ is an ideal of R. Let $S = R/(\hat{I} + pR)$ which is a finite commutative ring with unity $1 + (\hat{I} + pR)$.

We claim that $\hat{g}$ is a nonzero polynomial in $S[x_1, \cdots, x_n]$. Otherwise, let $\hat{g} \in (\hat{I} + pR)[x_1, \cdots, x_n]$. Write $\hat{g} = g_1 + g_2$, where $g_1 \in \hat{I}[x_1, \cdots, x_n]$ and $g_2 \in pR[x_1, \cdots, x_n]$. Then $p^k \hat{g} = p^k g_2$ as $p^k g_1 = 0$. But $g_2 \in pR[x_1, \cdots, x_n]$, which contradicts the fact that k is the largest integer such that $g \in p^k R[x_1, \cdots, x_n]$. Thus $\hat{g}$ is a nonzero polynomial in $S[x_1, \cdots, x_n]$. Now we argue that S contains the finite field $\mathbb{F}_p$, and then using the Lemma 5.3, the proof of the lemma will follow easily. To see a copy of $\mathbb{F}_p$ inside S, it is enough to observe that $\{i + (\hat{I} + pR) \mid 0 \leq i \leq p-1\}$ as a field is isomorphic to $\mathbb{F}_p$ and contains the unity of S. Clearly the failure probability for identity testing of g in $R[x_1, \cdots, x_n]$ is upper bounded by the failure probability for the identity testing of $\hat{g}$ in $S[x_1, \cdots, x_n]$. Consider the natural homomorphism $\phi : U \to \mathbb{F}_p$, given by $\phi(ce) = c \bmod p$. Thus if we sample uniformly from U, using ϕ, we can $\frac{\epsilon}{2}$-uniformly sample from $\mathbb{F}_p$. Notice that for any $b \in \mathbb{F}_p$, $\frac{1-\epsilon/2}{p} \leq \text{Prob}_{x \in \mathbb{Z}_M}[x \equiv b \bmod p] \leq \frac{1+\epsilon/2}{p}$. Now using the Lemma 5.3, we conclude the following :

$$\text{Prob}_{a_1 \in U, a_2 \in U \cdots a_n \in U}[g(a_1, \cdots, a_n) = 0] \leq \text{Prob}_{b_1 \in \mathbb{F}_p \cdots b_n \in \mathbb{F}_p}[\hat{g}(b_1, \cdots, b_n) = 0] \leq \frac{d}{p}(1 + \frac{\epsilon}{2}),$$

where $b_i = a_i \pmod p$. The additional factor of $(1 + \frac{\epsilon}{2})$ comes from the fact that we are only sampling $\frac{\epsilon}{2}$-uniformly from $\mathbb{F}_p$. This can be easily verified from the proof of Lemma 5.3. Hence we have proved the lemma. ■

6 Randomized Polynomial Identity Testing over finite rings

In this section we study the identity testing problem over finite commutative ring oracle with unity. For the input polynomial, we consider both black-box representation and circuit representation. First we consider the black-box case. Our identity testing algorithm is a direct consequence of Lemma 5.4.

Theorem 6.1 *Let R (which decomposes into local rings as $\oplus_{i=1}^{\ell} R_i$) be a finite commutative ring with unity given as a oracle. Let the input polynomial $f \in R[x_1, \cdots, x_n]$ of degree at most d be given via black-box access. Suppose R_i's is of characteristic $p_i^{\alpha_i}$. Let $\epsilon > 0$ be a given constant. If $p_i \geq kd$ for all i, for some integer $k \geq 2$, we have a randomized polynomial time identity test with success probability $1 - \frac{1}{k}(1 + \frac{\epsilon}{2})$.*

Proof. Consider the natural isomorphism $\hat{\phi} : R[x_1, x_2, \cdots, x_n] \to \oplus_{i=1}^{\ell} R_i[x_1, x_2, \cdots, x_n]$. Let $\hat{\phi}(f) = (f_1, f_2, \cdots, f_\ell)$. If $f \not\equiv 0$ then $f_i \not\equiv 0$ for some $i \in [\ell]$, where $f_i \in R_i[x_1, x_2, \cdots, x_n]$. Fix such an i. Our algorithm is a direct application of Lemma 5.4. Define $U = \{ce \mid 0 \leq c \leq M\}$, assign values for the x_i's independently and uniformly at random from U, and evaluate f using the black-box access. The algorithm declares $f \not\equiv 0$ if and only if the computed value is nonzero. By Lemma 5.4, our algorithm outputs the correct answer with probability $1 - \frac{d}{p_i}(1 + \frac{\epsilon}{2}) \geq 1 - \frac{1}{k}(1 + \frac{\epsilon}{2})$. [1] ■

The drawback of Theorem 6.1 is that we get a randomized polynomial-time algorithm only when $p_i \geq kd$.

However, when the polynomial f is given by an arithmetic circuit we will get a randomized identity test that works for all finite commutative rings given by oracle. This is the main result in this section. A key idea is to apply the transformation from [AB03] to convert the given multivariate polynomial to a univariate polynomial. The following lemma has an identical proof as [AB03, Lemma 4.5].

Lemma 6.2 *Let R be an arbitrary commutative ring and $f \in R[x_1, x_2, \cdots, x_n]$ be any polynomial of maximum degree d. Consider the polynomial $g(x)$ obtained from $f(x_1, x_2, \cdots, x_n)$ by replacing x_i by $x^{(d+1)^{i-1}}$ i.e $g(x) = f(x, x^{(d+1)}, \cdots, x^{(d+1)^{n-1}})$. Then $f \equiv 0$ over $R[x_1, \cdots, x_n]$ if and only if $g \equiv 0$ over $R[x]$.*

By Lemma 6.2, it suffices to describe the identity test for a univariate polynomial in $R[x]$ given by an arithmetic circuit. Notice that if $\deg(f) = d$ then we can bound $\deg(g)$ by $d(d+1)^{n-1}$ which we denote by D. Our algorithm is simple and essentially the same as the Agrawal-Biswas identity test over the finite ring $\mathbb{Z}_n$ [AB03].

We will randomly pick a monic polynomial $q(x) \in U[x]$ of degree $\lceil \log O(D) \rceil$. Then we carry out a division of $f(x)$ by the polynomial $q(x)$ and compute the remainder $r(x) \in R[x]$. Our algorithm declares f to be identically zero if and only if $r(x) = 0$. Notice that we will use the structure of the circuit to carry out the division. At each gate we carry out the division. More precisely, if the inputs of a $+$ gate are the remainders $r_1(x)$ and $r_2(x)$, then the output of this $+$ gate is $r_1 + r_2$. Similarly if r_1 and r_2 are the inputs of a $*$ gate, then we divide $r_1(x)r_2(x)$ by $q(x)$ and obtain the remainder as its output. Crucially, since $q(x)$ is a monic polynomial, the division algorithm will make sense and produce unique remainder even if $R[x]$ is not a U.F.D (which is the case in general).

We now describe the pseudocode of the identity testing algorithm (Algorithm 2). Our algorithm takes as input an arithmetic circuit C computing a polynomial $f \in R[x_1, x_2, \cdots, x_n]$ of degree at most d and an $\epsilon > 0$.

We will now prove the correctness of the above randomized identity test in Lemmas 6.3, 6.4, and 6.5.

Lemma 6.3 *Let R be a local commutative ring with unity and of characteristic p^α for some prime p and integer $\alpha > 0$. Let g be a nonzero polynomial in $R[x]$ such that $g \in p^k R[x] \setminus p^{k+1} R[x]$ for $k < \alpha$. Let*

[1] Notice that we have to compute ce using the ring oracle for addition in R. Starting with e, we need to add it c times. The running time for this computation can be made polynomial in $\log c$ by writing c in binary and applying the standard doubling algorithm.

Algorithm 2 The Identity Testing algorithm

```
1: procedure IdentityTesting($C$,$\epsilon$)
2:     for $i = 1, n$ do
3:         $x_i \leftarrow x^{(d+1)^{i-1}}$                         ▷ Univariate transformation
4:     end for
5:     $g(x) \leftarrow C(x, x^{(d+1)}, \cdots, x^{(d+1)^{n-1}})$.
6:     $D \leftarrow d(d+1)^{n-1}$.             ▷ The formal degree of $g(x)$ is at most $D$
7:     Choose a monic polynomial $q(x) \in U[x]$ of degree $\lceil \log \frac{12D}{1-\epsilon} \rceil$ uniformly at random.
8:     Divide $g(x)$ by $q(x)$ and compute the remainder $r(x)$.   ▷ The division algorithm uses the structure
   of the circuit.
9:     if $r(x) = 0$ then
10:        $C$ computes a zero polynomial.
11:    else
12:        $C$ computes a nonzero polynomial.
13:    end if
14: end procedure
```

$\hat{I} = \{r \in R \mid p^k r = 0\}$, $g = p^k \hat{g}$ where $\hat{g} \notin pR$ and q is a monic polynomial in $R[x]$. If q divides g in R, then q divides $\hat{g}$ in $R/(\hat{I} + pR)$.

Proof. As $q(x)$ divides $g(x)$ in $R[x]$, we have $g(x) = q(x)q_1(x)$ for some polynomial $q_1(x) \in R[x]$. Suppose $\hat{g}(x) = q(x)\bar{q}(x) + r(x)$ in $R[x]$ where the degree of $r(x)$ is less than the degree of $q(x)$. Also note that the division makes sense even over the ring as $q(x)$ is monic. We want to show that $r(x) \in (\hat{I} + pR)[x]$. We have the following relation in $R[x]$:

$$g = qq_1 = p^k \hat{g} = p^k q\bar{q} + p^k r.$$

So, $p^k r = q(q_1 - p^k \bar{q})$. If $(q_1 - p^k \bar{q}) \not\equiv 0$ in $R[x]$, then the degree of the polynomial $q(q_1 - p^k \bar{q})$ is strictly more than the degree of $p^k r$ as q is monic and degree of q is more than the degree of r. Thus $(qq_1 - p^k q\bar{q}) \equiv 0$ in $R[x]$ forcing $p^k r = 0$ in $R[x]$. So by the choice of $\hat{I}$, we have $r(x) \in \hat{I}[x]$. Thus $r(x) \in (\hat{I} + pR)[x]$. Notice that in the Lemma 5.4, we have already proved that $\hat{g}(x) \not\equiv 0$ in $S[x]$, where $S = R/(\hat{I} + pR)$. Also q is nonzero in $S[x]$ as it is a monic polynomial. Hence we have proved that $q(x)$ divides $\hat{g}(x)$ over $S[x]$. ∎

The following lemma is basically chinese remaindering tailored to our setting.

Lemma 6.4 *Let R be a local ring with characteristic p^α. Let $g(x) \in p^k R[x] \setminus p^{k+1} R[x]$ for some $k \geq 0$. Let $g(x) = p^k \hat{g}(x)$ and $\hat{I} = \{r \in R \mid p^k r = 0\}$. Suppose $q_1(x), q_2(x)$ are two monic polynomials over $R[x]$ such that each of them divides g in $R[x]$. Moreover, suppose there exist polynomials $a(x), b(x) \in R[x]$ such that $aq_1 + bq_2 = 1$ in $R/(\hat{I} + pR)$. Then $q_1 q_2$ divides $\hat{g}$ in $R/(\hat{I} + pR)$.*

Proof. By the Lemma 6.3, we know that q_1 and q_2 divide $\hat{g}$ in $R/(\hat{I} + pR)$. Let $\hat{g} = q_1 \bar{q}_1$ and $\hat{g} = q_2 \bar{q}_2$ in $R/(\hat{I} + pR)$. Let $\bar{q}_1 = q_2 q_3 + r$ in $R/(\hat{I} + pR)$. So, $\hat{g} = q_1 q_2 q_3 + q_1 r$. Substituting $q_2 \bar{q}_2$ for $\hat{g}$, we get $q_2(\bar{q}_2 - q_1 q_3) = q_1 r$. Multiplying both side by a and substituting $aq_1(x) = 1 - bq_2$, we get $q_2[a(\bar{q}_2 - q_1 q_3) + br] = r$. If $r \not\equiv 0$ in $R/(\hat{I} + pR)$, we arrive at a contradiction since q_2 is monic and thus the degree of $q_2[a(\bar{q}_2 - q_1 q_3) + br]$ is more than the degree of r. ∎

Let $f(x)$ be a nonzero polynomial in $R[x]$ of degree at most D. The next lemma states that, if we pick a random monic polynomial $q(x) \in U[x]$ (U is similarly defined as before)of degree $d \approx \log O(D)$, with high probability, $q(x)$ will not divide $f(x)$.

Lemma 6.5 *Let R be a commutative ring with unity. Suppose $f(x) \in R[x]$ is a nonzero polynomial of degree at most D and $\epsilon > 0$ be a given constant. Choose a random monic polynomial $q(x)$ of degree $d = \lceil \log \frac{12D}{1-\epsilon} \rceil$ in $U[x]$. Then with probability at least $\frac{1-\epsilon}{4d}$, $q(x)$ will not divide $f(x)$ over $R[x]$.*[2]

Proof. Let $R \cong \bigoplus_i R_i$ is the local ring decomposition of R. As f is nonzero in $R[x]$, there exists j such that $f_j = \hat{\phi}_j(f)$ is nonzero in $R_j[x]$. Clearly, we can lower bound the required probability by the probability that $q_j = \hat{\phi}_j(q)$ does not divide f_j in $R_j[x]$. Let the characteristic of R_j is p^α. If q_j divides f_j in $R_j[x]$, then it also divides over $R_j/(\hat{I}_j + pR_j)$. It is shown in the proof of the Lemma 5.4, $\mathbb{F}_p \subset R_j/(\hat{I}_j + pR_j)$. Now the number of irreducible polynomials in $\mathbb{F}_p$ of degree d is at least $\frac{p^d - 2p^{d/2}}{d}$. Let $t = \frac{p^d - 2p^{d/2}}{d}$. Let $\hat{q}(x) = \sum_{i=0}^{d-1} b_i x^i + x^d \in \mathbb{F}_p[x]$ be a monic polynomial. Now if a monic polynomial $P(x)$ of degree d is randomly chosen from $U[x]$ then, Prob$[P(x) \equiv \hat{q}(x) \bmod p] = \frac{\prod_{i=0}^{d-1} \lfloor (M-b_i)/p \rfloor + 1}{M^d} \geq \frac{1}{p^d}(1 - \frac{2^m}{M})^d$. Again, choosing $M > d2^{m+1}/\epsilon$, we get Prob$[P(x) \equiv \hat{q}(x) \bmod p] \geq (1 - \epsilon/2)/p^d$.

So, the probability that q_j is an irreducible polynomial in $\mathbb{F}_p[x]$ is at least $t(1-\epsilon)/p^d > (1-\epsilon)/2d$. Let $f_j \in p^k R_j[x] \setminus p^{k+1} R_j[x]$. So we can write $f_j = p^k f'$, where $f' \in R_j[x] \setminus pR_j[x]$. By the Lemma 6.3, q_j divides f' in $R/(\hat{I}_j + pR)$. Also, by the Lemma 6.4, the number of different monic polynomials that are irreducible in $\mathbb{F}_p$ and divides f' in $R_j/(\hat{I}_j + pR_j)$ is at most D/d. In the sample space for q, any monic polynomial of degree d in $R_j/(\hat{I}_j + pR_j)$ occurs at most $(\frac{M}{p} + 1)^d$ times. So the probability that a random monic irreducible polynomial q will divide f is at most $\frac{(D/d)(\frac{M}{p}+1)^d}{M^d} \leq \frac{D}{dp^d}(1 + \frac{1}{d})^d < \frac{3D}{d2^d}$. So a random monic polynomial $q \in U[x]$ (which is irreducible in $\mathbb{F}_p$ with reasonable probability) will not divide $f(x)$ with probability at least $\frac{1-\epsilon}{2d} - \frac{3D}{dp^d} > \frac{1-\epsilon}{4d}$ for $d \geq \lceil \log \frac{12D}{1-\epsilon} \rceil$. ∎

The correctness of Algorithm 2 and its success probability follow directly from Lemma 6.3, Lemma 6.4 and Lemma 6.5.

In particular, by Lemma 6.5, the success probability of our algorithm is at least $\frac{1-\epsilon}{4t}$, where $t = \lceil \log \frac{12D}{1-\epsilon} \rceil$. As $\frac{1-\epsilon}{4t}$ is an inverse polynomial quantity in input size and the randomized algorithm has one-sided error, we can boost the success probability by repeating the test polynomially many times. We summarize the result in the following theorem.

Theorem 6.6 *Let R be a finite commutative ring with unity given as an oracle and $f \in R[x]$ be a polynomial, given as an arithmetic circuit. Then in randomized time polynomial in the circuit size and* $\log\ |R|$ *we can test whether $f \equiv 0$ in $R[x]$.*

Randomized polynomial time identity testing for multivariate polynomials $f \in R[x_1, \cdots, x_n]$ given by arithmetic circuits follows from Theorem 6.6 and Lemma 6.2.

Theorem 6.7 *Let R be a commutative ring with unity given as an oracle. Let f be a polynomial in $R[x_1, x_2, \cdots, x_n]$ of formal degree at most d, is given by an arithmetic circuit over R. Then in randomized time polynomial in circuit size and* $\log |R|$ *we can test whether $f \equiv 0$ in $R[x_1, x_2, \cdots, x_n]$.*

[2]An alternative proof of this lemma based on [AB03, Lemma 4.7] is given in the appendix.

Remark 6.8 *The randomized polynomial-time identity test of Bogdanov and Wee [BW05] for noncommutative circuits of polynomially bounded degree in* $\mathbb{F}\{x_1, \cdots, x_n\}$ *for a field* $\mathbb{F}$*, can be extended to such circuits over any commutative ring* R *with unity, where* R *is given by a ring oracle. This follows from the fact that the Amitsur-Levitzki theorem is easily seen to hold even in the ring* $R\{x_1, \cdots, x_n\}$*. The easy details are given in the appendix.*

Remark 6.9 *Finally, we note that the results in Section 2 carry over without changes to noncommuting polynomials in* $R\{x_1, \cdots, x_n\}$*, where* R *is a commutative ring with unity given by a ring oracle.*

Acknowledgements. For their insightful comments and suggestions we are grateful to Amir Shpilka and the referees of the CCC'08 conference.

References

[AB03] M. AGRAWAL AND S. BISWAS. Primality and identity testing via Chinese remaindering. *J. ACM.,* 50(4):429-443, 2003.

[AL50] S.A AMITSUR AND J. LEVITZKI. Minimal Identities for algebras. *In Proceedings of the American Mathematical Society.,* volume 1, pages 449-463, 1950.

[AM69] M.F. ATIYAH AND I.G. MACDONALD. Introduction to commutative algebra. *Addison-Wesley Publishing Company,* 1969.

[AM07] V. ARVIND AND P. MUKHOPADHYAY The Ideal Membership problem and Polynomial Identity Testing. *ECCC report TR07-095,* 2007.

[BT88] 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.

[BW05] 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.

[DS05] 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.

[GZ05] A. GIAMBRUNO AND M. ZAICEV. Polynomial Identities and Asymptotic Methods. *American Mathematical Society.,* Vol. 122, 2005.

[HU78] J.E. HOPCROFT AND J.D. ULLMAN Introduction to Automata Theory, Languages and Computation, *Addison-Wesley,* 1979.

[IKW02] R. IMPAGLIAZZO, V. KABANETS AND A. WIGDERSON. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences 65(4).,* pages 672-694, 2002.

[KI03] 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.

[KS05] NEERAJ KAYAL, NITIN SAXENA, On the Ring Isomorphism and Automorphism Problems. *IEEE Conference on Computational Complexity,* 2-12, 2005.

[KS07] N. KAYAL AND N. SAXENA. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity.*, 16(2):115-138, 2007.

[Le92] H.W.LENSTRA JR. Algorithms in algebraic number theory. *Bulletin of the AMS.*, 26(2), 211-244, 1992.

[Mc74] B R. MACDONALD. Finite Rings with Identity. *Marcel Dekker, INC. New York*, 1974.

[MR95] R. MOTWANI AND P. RAGHAVAN. Randomized Algorithms. *Cambridge University Press.,*, 1995.

[N91] N. NISAN. Lower bounds for non-commutative computation. *In Proc. of the 23rd annual ACM Sym. on Theory of computing.,* pages 410-418, 1991.

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

[Sch80] JACOB T. SCHWARTZ. Fast Probabilistic algorithm for verification of polynomial identities. *J. ACM.,* 27(4), pages 701-717, 1980.

[Str94] HOWARD STRAUBING. Finite automata, formal logic, and circuit complexity. *Progress in Theoretical Computer Science.* Birkhuser Boston Inc., Boston, MA, 1994.

[Zip79] R. ZIPPEL. Probabilistic algorithms for sparse polynomials. *In Proc. of the Int. Sym. on Symbolic and Algebraic Computation.,* pages 216-226, 1979.

A Noncommutative identity testing over commutative coefficient rings

Here we extend the noncommutative identity testing of Bogdanov and Wee [BW05] to over $R\{x_1, \cdots, x_n\}$ where R is an arbitrary commutative ring with unity. Our algorithm is a combination of Amitsur-Levitzki's theorem and the Theorem 6.7. We first briefly discuss the Amitsur-Levitzki's result tailored to our application and the result of [BW05]. Let $M_k(\mathbb{F})$ be the $k \times k$ matrix algebra over $\mathbb{F}$. The following algebraic lemma was the key result used in [BW05].

Lemma A.1 *[AL50, GZ05] $M_k(\mathbb{F})$ does not satisfy any non-trivial polynomial identity of degree $< 2k$.*

Based on Lemma A.1, a noncommutative version of the Schwartz-Zippel lemma over $\mathbb{F}\{x_1, \cdots, x_n\}$ is described in [BW05]. We first give an intuitive description of the identity testing algorithm in [BW05]. Assume $f \in \mathbb{F}\{x_1, \cdots, x_n\}$ is of degree d and is given by an arithmetic circuit. Fix k such that $k > \lceil d/2 \rceil$. Consider a field extension $\mathbb{F}'$ of $\mathbb{F}$ such that $|\mathbb{F}'| >> d$. The idea is to evaluate the circuit on random $k \times k$ matrices from $M_k(\mathbb{F}')$. We think each entry of the matrix as an indeterminate and view the k^2 indeterminates as commuting variables. So at the output of the circuit, we get a $k \times k$ matrix such that each of its entries are polynomials in commuting variables. Lemma A.1 guarantees that $f \equiv 0$ in $\mathbb{F}\{x_1, \cdots, x_n\}$ if and only if each of the k^2 polynomials computed as the entries of the matrix at the output gate, are identically zero. Then we get a lower bound of the success probability via commutative Schwartz-Zippel lemma.

We give a randomized polynomial time identity testing algorithm over $R\{x_1, \cdots, x_n\}$ where R is any finite commutative ring with unity and is given by a ring oracle. Our algorithm is based on the observation that Lemma A.1 is valid over $M_k(R)$. For the sake of completeness, we briefly discuss the proof of the Lemma A.1 tailored to R. The following fact is the key in proving the Lemma A.1.

Fact A.2 [GZ05, page 7] *Let A be an $\mathbb{F}$-algebra spanned by a set B over $\mathbb{F}$. If the algebra A satisfies an identity of degree k in $\mathbb{F}\{x_1, \cdots, x_n\}$, then it satisfies a multilinear identity of degree $\leq k$.*

We observe that the result of the Fact A.2 holds, even if A be an algebra over R. Proof is analogous to the proof of the Fact A.2. Following [GZ05, page 7], we call a polynomial f *multilinear* if every variable occurs with degree exactly one in every monomial of f.

Lemma A.3 *Let A be an R-algebra such that A satisfies an identity of degree k. Then it satisfies a multilinear identity of degree k.*

Proof. The lemma follows from an identical argument to that in the proof of Theorem 1.3.7 in [GZ05]. ∎

Using Lemma A.3, it follows that Lemma A.1 extends to $M_k(R)$. The proof is analogous to the proof of Theorem 1.7.2 in [GZ05]. Let f be an identity for $M_k(R)$ of degree $< 2k$. By the Lemma A.3, we can assume that f is multilinear. Also, multiplying f by the new variables from the right, we can assume that the degree of f is $2k - 1$. Let,

$$f(x_1, x_2, \cdots, x_{2k-1}) = \sum_{\sigma \in S_{2k-1}} \alpha_\sigma x_{\sigma(1)} \cdots x_{\sigma(2k-1)}$$

with $\alpha_1 \neq 0$, where 1 denotes the identity permutation. Let e_{ij} be the $k \times k$ matrix with unity (of R) at the (i, j)-th entry and zero in all other entries. It is easy to see that $f(e_{11}, e_{12}, e_{22}, e_{23}, \cdots, e_{k-1,k}, e_{kk}) = \alpha_1 e_{1k} \neq 0$, since $x_1 \cdots x_{2k-1}$ is the only monomial that does not vanish during the evaluation. So f is not an identity for $M_k(R)$. The fact that R is a ring with unity is crucially used.

Lemma A.4 *Let R be a finite commutative ring with unity. Then $M_k(R)$ does not satisfy any polynomial identity of degree $< 2k$.*

Now we a randomized polynomial time identity testing algorithm over $R\{x_1, \cdots, x_n\}$.

Theorem A.5 *Let $f \in R\{x_1, \cdots, x_n\}$ be a polynomial of degree d, given by a noncommutative arithmetic circuit C. R is given as a ring oracle and its elements are encoded using binary strings of length m. Then there is a randomized polynomial time algorithm (*poly*(n,d,m)) to test if $f \equiv 0$ over $R\{x_1, \cdots, x_n\}$.*

Proof. Let $x_1, x_2, \cdots, x_n$ are the indeterminates in C. Choose $k = \lceil d/2 \rceil + 1$. Replace each x_i by a $k \times k$ matrix over the set of indeterminates $\{y_{j\ell}^{(i)}\}_{1 \le j,\ell \le k}$. Once we replace x_i by matrices , the inputs and the outputs of the gates will be matrices. Replace each addition (multiplication) gate by a block of circuits computing the sum (product) of two $k \times k$ matrices (without loss of generality, assume that the fan-in of all gates is two). This can be easily achieved using $O(k^2)$ gates. Let $\hat{C}$ be the arithmetic circuit obtained from C by these modifications. Clearly, $\hat{C}$ computes a function from $\mathbb{F}^{nk^2} \to \mathbb{F}^{k^2}$ and the size of $\hat{C}$ is only polynomial in the size of C. Denote by $\bar{Y}$ the variable list $(y_{11}^{(1)}, \cdots, y_{kk}^{(1)}, \cdots, y_{11}^{(n)}, \cdots, y_{kk}^{(n)})$. Then,

$$\hat{C}(\bar{Y}) = (P_1(\bar{Y}), \cdots, P_{k^2}(\bar{Y})).$$

Also, by the Lemma A.4, $M_k(R)$ does not satisfy any identity of degree $< 2k$ over $R\{x_1, \cdots, x_n\}$. So f satisfies $M_k(R)$ if and only if $f \equiv 0$ in $R\{x_1, \cdots, x_n\}$, which equivalently implies that $P_i \equiv 0$ over $R[\bar{Y}]$ for all i. Notice that the degree of P_i is $\le d$. Now we appeal to the Theorem 6.7 in order to test whether $P_i \equiv 0$ in time poly(n, d, m). ■

Bogdanov and Wee in [BW05] evaluate the noncommutative circuit over a field extension $\mathbb{F}'$ of $\mathbb{F}$ in case $\mathbb{F}$ is a small field compared to the degree. In our proof of Theorem A.5, when coefficients come from the ring R, we avoid working in a ring extension and instead apply Theorem 6.7.

B Alternative proof of Lemma 6.5

Let R be a finite commutative ring with unity (denoted e) and its elements uniformly encoded in $\{0,1\}^m$.

Recall we need to show the following: if we divide a nonzero polynomial $g(x) \in R[x]$ of degree D by a random monic polynomial $q(x) \in U[x]$ of degree $\log O(D)$ then with high probability we get a nonzero remainder. Recall from Section 6 that $U = \{ke \mid 0 \le k \le M-1\}$, where $M > 2^{m+1}/\epsilon$.

Indeed, Agrawal and Biswas essentially show in [AB03, Lemma 4.7] that the above result holds for the special case when the ring R is the ring $\mathbb{Z}_t$ of integers modulo t, where t is any positive integer given in binary. In Section 6 we gave a self-contained proof of Lemma 6.5. In the sequel we give a different proof which applies the [AB03] result for $\mathbb{Z}_t$ and brings out an interesting property of the division algorithm.

Let n denote the characteristic of the ring R. Then sampling from $U[x]$ amounts to almost uniform sampling from the copy of $\mathbb{Z}_n[x]$, namely $\mathbb{Z}_n e[x]$, contained in $R[x]$ as a subring. Since $(R, +)$ is a finite abelian group, by the fundamental theorem for abelian groups, we can write $(R, +)$ as a direct sum $R = \bigoplus_{i=1}^k \mathbb{Z}_{n_i} e_i$, where $e_1, \cdots, e_k$ forms an independent generating set for $(R, +)$, and n_i is the additive order of e_i for each i. Notice that the lcm of $n_1, \cdots, n_k$ is the ring's characteristic n. This decomposition extends naturally to the additive group $(R[x], +)$ to give

$$R[x] = \bigoplus_{i=1}^{k} \mathbb{Z}_{n_i}[x] e_i. \tag{2}$$

Thus, every polynomial $g(x) \in R[x]$ can be uniquely written as $g(x) = \sum_{i=1} g_i(x)e_i$, where g_i is a polynomial with integer coefficients in the range $0, \cdots, n_i - 1$ for each i. Clearly, dividing $g(x)$ by $q(x)$ amounts to dividing each term in $\sum_{i=1} g_i(x)e_i$. The following claim tells us how to analyze this term by term division. More precisely, we analyze the quotient and remainder when we divide $g_i(x)e_i \in R[x]$ by $q(x) \in \mathbb{Z}_n[x]$ ($\cong Z_n e[x] \subseteq R[x]$).

Claim B.1 *Let $g_i(x) = q(x)q'(x) + r'(x)$ be the quotient and remainder when we divide $g_i(x)$ by $q(x)$ in the ring $\mathbb{Z}_{n_i}[x]$. Let $g_i(x)e_i = q(x)q''(x) + r''(x)$ be the quotient and remainder when we divide $g_i(x)e_i$ by $q(x)$ in the ring $R[x]$. Then $q'(x)e_i = q''(x)$ and $r'(x)e_i = r''(x)$.*

This claim is somewhat surprising because Equation 2 only gives us a *group* decomposition of $R[x]$ and not a *ring* decomposition. Thus, it is not clear why division in the ring $\mathbb{Z}_{n_i}[x]$ can be related to division in $R[x]$. Indeed, the crucial reason why we can relate the two divisions is because the divisor polynomial $q(x)$ lives in the copy of $\mathbb{Z}_n[x]$ inside $R[x]$.

To see the claim, we will carry out the division of $g_i(x)$ by $q(x)$ over $R[x]$. Since both g_i and $q(x)$ have integer coefficients, this amounts to carrying out division in $\mathbb{Z}_n[x]$ which yields, say, $g_i(x) = q(x)q_1(x) + r_1(x)$. We can also write $q_1(x) = a(x) + n_i b(x)$ and $r_1(x) = c(x) + n_i d(x)$. Then, over $\mathbb{Z}_{n_i}$, notice that we must have $g_i(x) = q(x)a(x) + c(x)$. Therefore, $a(x) = q'(x)$ and $c(x) = r'(x)$. Now, multiplying both sides by e_i we will get $q_1(x)e_i = a(x)e_i + n_i e_i b(x) = a(x)e_i = q'(x)e_i$. Similarly, we get $r_1(x)e_i = c(x)e_i = r'(x)e_i$. Furthermore, again multiplying both sides by e_i, we also get $g_i(x)e_i = q(x)q_1(x)e_i + r_1(x)e_i$. Hence, $q''(x) = q_1(x)e_i = q'(x)e_i$ and $r''(x) = r_1(x)e_i = r'(x)e_i$. This proves the claim.

A consequence of the claim is the following nice property of the division algorithm: in order to divide $g(x)$ by $q(x)$ over the ring R, for each i we can carry out the division of $g_i(x)$ by $q(x)$ over the ring $\mathbb{Z}_{n_i}$ and obtain the quotients and remainders:

$$g_i(x) = q(x)q'_i(x) + r'_i(x).$$

Then we can put together the term by term divisions to obtain

$$g(x) = q(x)(\sum_{i=1}^{k} q'_i(x)e_i) + (\sum_{i=1}^{k} r'_i(x)e_i). \tag{3}$$

More precisely, when we divide $g(x)$ by $q(x)$ in $R[x]$, the quotient is $\sum_{i=1}^{k} q'_i(x)e_i$ and the remainder is $\sum_{i=1}^{k} r'_i(x)e_i$. Now, since $g \in R[x]$ is nonzero, there is an index j such that $g_j[x] \in \mathbb{Z}_{n_j}[x]$ is nonzero. Furthermore, since n_j is a factor of n, the polynomial $q(x)$ modulo n_j is still an almost uniformly distributed random monic polynomial. It follows from the Agrawal-Biswas result [AB03, Lemma 4.7] applied to division of $g_j(x)$ by $q(x)$ over $\mathbb{Z}_{n_j}$ that $r'_j(x)$ will be nonzero with high probability. Consequently, by Equation 3 the remainder $\sum_{i=1}^{k} r'_i(x)e_i$ on dividing $g(x)$ by $q(x)$ in the ring $R[x]$ is also nonzero with the same probability.