Electronic Colloquium on Computational Complexity, Report No. 198 (2024)

Randomized Black-Box PIT for Small Depth +-Regular Non-commutative Circuits

G V Sumukha Bharadwaj*and S Raja†

IIT Tirupati, India

Abstract

In this paper, we address the black-box polynomial identity testing (PIT) problem for non-commutative polynomials computed by +-regular circuits, a class of homogeneous circuits introduced by Arvind, Joglekar, Mukhopadhyay, and Raja (STOC 2017, Theory of Computing 2019). These circuits can compute polynomials with a number of monomials that are doubly exponential in the circuit size. They gave an efficient randomized PIT algorithm for +-regular circuits of depth 3 and posed the problem of developing an efficient black-box PIT for higher depths as an open problem.

We present a randomized black-box polynomial-time algorithm for +-regular circuits of any constant depth. Specifically, our algorithm runs in $s^{O(d^2)}$ time, where s and d represent the size and the depth of the +-regular circuit, respectively. Our approach combines several key techniques in a novel way. We employ a nondeterministic substitution automaton that transforms the polynomial into a structured form and utilizes polynomial sparsification along with commutative transformations to maintain non-zeroness. Additionally, we introduce matrix composition, coefficient modification via the automaton, and multi-entry outputs—methods that have not previously been applied in the context of black-box PIT. Together, these techniques enable us to effectively handle exponential degrees and doubly exponential sparsity in non-commutative settings, enabling polynomial identity testing for higher-depth circuits. Our work resolves an open problem from [AJMR19].

In particular, we show that if f is a non-zero non-commutative polynomial in n variables over the field $\mathbb{F}$, computed by a depth-d +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and the size of the field $\mathbb{F}$ depending on the degree of f. Our result can be interpreted as an Amitsur-Levitzki-type result [AL50] for polynomials computed by small-depth +-regular circuits.

*cs21d003@iittp.ac.in

†raja@iittp.ac.in

ISSN 1433-8092

Contents

1 Introduction

The non-commutative polynomial ring, denoted by $\mathbb{F}\langle X\rangle$, over a field $\mathbb{F}$ in non-commuting variables X, consists of non-commuting polynomials in X. These are just $\mathbb{F}$-linear combinations of words (we call them monomials) over the alphabet $X = \{x_1, \ldots, x_n\}$. Hyafil [Hya77] and Nisan [Nis], studied the complexity of *non-commutative* arithmetic computations, in particular the complexity of computing the determinant polynomial with non-commutative computations.

Non-commutative arithmetic circuit families compute non-commutative polynomial families in a non-commutative polynomial ring $\mathbb{F}\langle X\rangle$, where multiplication is non-commutative (i.e., for distinct $x, y \in X$, $xy \neq yx$). We now recall the formal definition of non-commutative arithmetic circuits.

Definition 1.0.1 (Non-commutative arithmetic circuits)**.** *A* non-commutative arithmetic circuit *C over a field $\mathbb{F}$ is a directed acyclic graph such that each in-degree* 0 *node of the graph is labelled with an element from $X \cup \mathbb{F}$, where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of noncommuting variables. Each internal node has fan-in two and is labeled by either $(+)$ or $(\times)$ – meaning a $+$ or $\times$ gate, respectively. Furthermore, each $\times$ gate has a designated left child and a designated right child. Each gate of the circuit inductively computes a polynomial in $\mathbb{F}\langle X\rangle$: the polynomials computed at the input nodes are the labels; the polynomial computed at a $+$ gate (respectively $\times$ gate) is the sum (respectively product in left-to-right order) of the polynomials computed at its children. The circuit C computes the polynomial at the designated output node. An arithmetic circuit is a* formula *if the fan-out of every gate is at most one.*

Designing an efficient deterministic algorithm for non-commutative polynomial identity testing is a major open problem. Let $f \in \mathbb{F}\langle X\rangle$ be a polynomial represented by a non-commutative arithmetic circuit C. In this work, we assume that the polynomial f is given by a black-box access to C, meaning we can evaluate the polynomial f on matrices with entries from $\mathbb{F}$ or an extension field. Note that the degree of an n-variate polynomial computed by the circuit C of size s can be as large as 2^s and the sparsity, i.e., the number of non-zero monomials, can be as large as n^{2^s}. For example, the non-commutative polynomial $(x+y)^{2^s}$ has degree 2^s, doubly exponential sparsity 2^{2^s}, and has a circuit of size $O(s)$.

The classical result of Amitsur-Levitzki [AL50] shows that a non-zero non-commutative polynomial f of degree $2d-1$ does not vanish on the matrix algebra $\mathcal{M}_d(\mathbb{F})$. Bogdanov and Wee [BW05] have given an efficient randomized PIT algorithm for non-commutative circuits computing polynomials of degree $d = poly(s, n)$. Their algorithm is based on the result of Amitsur-Levitzki [AL50], which states the existence of matrix substitutions $M = (M_1, M_2, \ldots, M_n)$ such that the matrix $f(M_1, M_2, \ldots, M_n)$ is not the zero matrix, where the dimension of the matrices in M depends linearly on the degree d of the polynomial f.

Since the degree of the polynomial computed by circuit C can be exponentially large in the size of the circuit, their approach will not work directly. Finding an efficient randomized PIT algorithm for general non-commutative circuits is a well-known open problem. It was highlighted at the workshop on algebraic complexity theory (WACT 2016) as one of the key problems to work on.

Recently, [AJMR19] gave an efficient randomized algorithm for the PIT problem when the circuits are allowed to compute polynomials of exponential degree, but the sparsity could be exponential in the size of the circuit. To handle doubly-exponential sparsity, they studied a class of homogeneous non-commutative circuits, that they call $+$-regular circuits, and gave an efficient deterministic white-box PIT algorithm. These circuits can compute non-commutative polynomials with the number of monomials doubly exponential in the circuit size. For the black-box setting,

they obtain an efficient randomized PIT algorithm only for depth-3 +-regular circuits. In particular, they show that if a non-zero non-commutative polynomial $f \in \mathbb{F}\langle X\rangle$ is computed by a depth-3 +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_s(\mathbb{F})$ for a sufficiently large field $\mathbb{F}$. Finding an efficient randomized PIT algorithm for higher depth +-regular circuits is listed as an interesting open problem. We resolve this problem for constant depth +-regular circuits. In particular, we show that if $f \in \mathbb{F}\langle X\rangle$ is a non-zero non-commutative polynomial computed by a depth-d +-regular circuit of size s, then f cannot be a polynomial identity for the matrix algebra $\mathbb{M}_N(\mathbb{F})$, with $N = s^{O(d^2)}$ and the size of the field $\mathbb{F}$ depends on the degree of polynomial f. This resolves an open problem given in [AJMR19]. We note that we get a *black-box* randomized polynomial time black-box PIT algorithm for constant depth +-regular circuits.

Our results

We consider the black-box PIT problem for *+-regular circuits*. These are a natural subclass of homogeneous non-commutative circuits and these circuits can compute polynomials of exponential degree and a double-exponential number of monomials. Recall that a polynomial f is homogeneous if all of its monomials have the same degree. The syntactic degree is inductively defined as follows: For a leaf node labeled by a variable, the syntactic degree is 1, and 0 if it is labeled by a constant. For a + gate, its syntactic degree is the maximum of the syntactic degree of its children. For a $\times$ gate, its syntactic degree is the sum of the syntactic degree of its children. A circuit is called homogeneous if all gates in the circuit compute homogeneous polynomials. Now we recall the definition and some observations of +-regular circuits from [AJMR19].

Definition 1.0.2 (+-regular circuits [AJMR19])**.** *A non-commutative circuit C, computing a polynomial in $\mathbb{F}\langle X\rangle$, where $X = \{x_1, x_2, \ldots, x_n\}$, is* +-regular *if it satisfies the following properties:*

1. *The circuit is homogeneous. The + gates are of unbounded fanin and $\times$ gates are of fanin 2.*
2. *The + gates in the circuit are partitioned into layers (termed +-layers) such that if g_1 and g_2 are + gates in the same +-layer then there is no directed path in the circuit between g_1 and g_2.*
3. *All gates in a +-layer have the same syntactic degree.*
4. *The output gate is a + gate.*
5. *Every input-to-output path in the circuit goes through a gate in each +-layer.*
6. *Additionally, we allow scalar edge labels in the circuit. For example, suppose g is a + gate in C whose inputs are gates $g_1, g_2, \ldots, g_t$ such that $\beta_i \in \mathbb{F}$ labels edge $(g_i, g), i \in [t]$. If polynomial P_i is computed at gate $g_i, i \in [t]$, then g computes the polynomial $\sum_{i=1}^{t} \beta_i P_i$.*

The +-depth, denoted by d^+, refers to the number of + layers in C. The + layers in circuit C are numbered from the bottom upwards. For $i \in [d]$, let $\mathcal{L}_i^+$ represent the i-th layer of addition (+) gates, and let $\mathcal{L}_i^\times$ represent the i-th layer of multiplication ($\times$) gates that are inputs to the addition gates in $\mathcal{L}_i^+$. It's important to note that all gates in $\mathcal{L}_i^\times$ and $\mathcal{L}_i^+$ have the same syntactic degree.

The sub-circuit in C between any two consecutive addition layers $\mathcal{L}_i^+$ and $\mathcal{L}_{i+1}^+$ consists of multiplication gates and is denoted by Π^*. The inputs of this sub-circuit come from layer $\mathcal{L}_i^+$. Let $\mathcal{L}_{i+1}^\times$ consist of all output gates of this sub-circuit, where $1 \le i \le d-1$. Note that all the gates of

$\mathcal{L}^{\times}_{i+1}$ are product gates. It is important to note that this sub-circuit depth, which is the number of gates in any input-to-output gate path in the sub-circuit, can be arbitrary and is only bounded by the size of the circuit C. The bottom-most $\times$-layer $\mathcal{L}^{\times}_1$ can be assumed without loss generality to be the input variables and gates in $\mathcal{L}^{+}_1$ compute homogeneous linear forms.

Remark 1.0.1. *If the top layer is Π^* (i.e., the output gate is a $\times$ gate), we can add an extra $+$ gate at the top with the $+$ gate having a single input (i.e., fan-in 1). This ensures that the top layer is a Σ layer. If the bottom layer is Π^*, then for each input variable, we can add a sum gate having a single input. This will increase the circuit size by at most $n+1$, where n is the number of input variables. This allows us to assume that in $+$-regular circuits, both the top and bottom layers are Σ layers. This is done to simplify the analysis.*

The size of the $+$-regular circuit is the number of gates in the circuit. As noted earlier, the non-commutative polynomial $(x+y)^{2^s}$ can be computed by a depth-3 $+$-regular circuit, denoted by $\Sigma\Pi^*\Sigma$, of size $O(s)$ using repeated squaring. This circuit consists of two addition layers, namely $\mathcal{L}^{+}_1, \mathcal{L}^{+}_2$ and two multiplication layers, namely $\mathcal{L}^{\times}_1, \mathcal{L}^{\times}_2$. The multiplication layer $\mathcal{L}^{\times}_1$ consists of only the two input gates labeled by x and y respectively. The addition layer $\mathcal{L}^{+}_1$ consists of only one addition gate computing the homogeneous linear form $(x+y)$. The multiplication layer $\mathcal{L}^{\times}_2$ consists of only one gate computing the polynomial $(x+y)^{2^s}$. The addition layer $\mathcal{L}^{+}_2$ consists of only the ouptut gate computing the polynomial $(x+y)^{2^s}$. In this example, the top-most addition gate (i.e., the output gate) essentially has one input. The main result of the paper is the following theorem.

Theorem 1. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a $+$-regular circuit of depth d and size s. Then $f \not\equiv 0$ if and only if f is not identically zero on the matrix algebra $\mathbb{M}_N(\mathbb{F})$, with $N = s^{O(d^2)}$ and $\mathbb{F}$ is sufficiently large.*

For degree D non-zero non-commutative polynomial f, the classical Amitsur-Levitzki [AL50] theorem guarantees that f does not vanish on the matrix algebra $\mathcal{M}_{\frac{D}{2}+1}(\mathbb{F})$. If $D = 2^{\Omega(s)}$, this gives us an exponential time randomized PIT algorithm [BW05], where s is the size of the circuit computing f. If the sparsity of the polynomial, i.e., the number of non-zero monomials, is doubly exponential, then the main result of [AJMR19] gives only an exponential time randomized PIT algorithm as their matrix dimension depends on the logarithm of the sparsity.

This above theorem demonstrates that if the polynomial f is computed by a $+$-regular circuit of size s and depth $o(\sqrt{s}/\log s)$, we can determine if f is identically zero or not using a $2^{o(s)}$ time randomized PIT algorithm, which is exponentially faster than the existing methods. In particular, if depth is $O(1)$ then our algorithm runs in polynomial time. It is important to note that the number of product gates (within each Π^* layers) in any input-to-output path can be arbitrary and is only bounded by the circuit size s.

We note that [AJMR19] presented a white-box deterministic polynomial-time PIT for arbitrary depth $+$-regular circuits. For the small-degree case, [RS05] provided a white-box deterministic polynomial-time PIT for non-commutative ABPs, while [FS13, AGKS15] have shown a quasi-polynomial-time black-box PIT algorithm for non-commutative ABPs.

1.1 Outline of the Proofs: High-level Idea

In the rest of the paper, we will use "n.c." as an abbreviation for "non-commutative". We first explain the main ideas behind the randomized PIT algorithm for depth-5 $+$-regular circuits, as this is the major bottleneck. Consider an n.c. polynomial f over $X = \{x_1, \ldots, x_n\}$ computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. The polynomial f, computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s, can be written as follows:

$$f = \sum_{i \in [s]} \prod_{j \in [D_2]} Q_{ij}. \tag{1}$$

Here, the degree of each Q_{ij}, $i \in [s], j \in [D_2]$, is denoted by D_1 and can be computed by a $\Sigma\Pi^*\Sigma$ circuit of size at most s. As the size of the circuit is s, the output gate's fan-in is bounded by s. The syntactic degree D of the circuit can be expressed as $D = D_1 \times D_2$. In general, both D_1 and D_2 can be exponential in s. Note that each Q_{ij} is a polynomial computed at layer $\mathcal{L}_2^+$. One natural idea is that since each Q_{ij} can be computed by a $\Sigma\Pi^*\Sigma$ circuit, we can try to use the known result of depth-3 +-regular circuits [AJMR19] and convert the given polynomial f into a commutative polynomial, and then perform the randomized PIT using the Polynomial Identity Lemma for commutative polynomials (also known as the DeMillo-Lipton-Schwartz-Zippel Lemma [DL78, Zip79, Sch80]).

Recall that in [AJMR19], the given polynomial f is computed by a depth-3 +-regular circuit of size s. That is, f is a sum of products of homogeneous linear forms. Formally, $f = \sum_{i=1}^{s} P_i$, where for all $i \in [s], P_i = L_{i,1} \cdots L_{i,D}$ and D could be exponential in s. They show that there exists an index set $I \subseteq [D]$ of size at most $s-1$ such that by considering only those linear forms positions indexed by I as n.c. and the remaining as commutative, the non-zeroness of f is preserved. This fact is crucially used in their black-box identity-testing algorithm for depth-3 +-regular circuits. In our depth-5 setting, that is, when $f = \sum_{i \in [s]} \prod_{j \in [D_2]} Q_{ij}$, it is only natural to wonder if there exists such a small index. Note that the number of Q_{ij} polynomials in each product, denoted by D_2, can be exponential, in general. It is generally impossible to have a small index set of polynomial size. This is because if the index set is only polynomial in size, then as a result, no variables in some of the Q_{ij} are considered non-commutative. Thus, these Q_{ij} are considered commutative, possibly resulting in the commutative polynomial becoming 0. We cannot consider a n.c. polynomial as commutative and still maintain non-zeroness, in general.

To resolve this problem, we convert the n.c. polynomial into a commutative polynomial in several steps, utilizing the fact that a +-regular circuit computes it.

1.1.1 Step 1: Transforming the Polynomial for Improved Structure

In this step, we show that the polynomial can be converted into a more structured polynomial at the cost of introducing some spurious monomials. We show that in each Q_{ij} there is a small index set such that by considering only those homogeneous linear forms appearing in that Q_{ij} as n.c. and the remaining as commutative, the non-zeroness of f is preserved. The index sets are encoded using n.c. variables. This results in an exponential-sized index set. This is because the number of terms in the product, denoted by D_2, can be exponential in general and each term in the product has a small index set.

This fact is formalized in Lemma 4.1.1. However, this fact alone will not be sufficient to design an identity-testing algorithm for depth-5 +-regular circuits. This is because, to choose a small index set from each Q_{ij}, one would have to know the *boundary* between Q_{ij} and $Q_{i,j+1}$ (i.e., the position at which Q_{ij} ends and $Q_{i,j+1}$ begins), for any $i \in [s], j \in [D_2 - 1]$. To know where Q_{ij} ends and $Q_{i,j+1}$ begins exactly, the algorithm will have to keep track of the degree (length) of Q_{ij}. This is not feasible as the degree of Q_{ij} could be exponential in s. This is one of the main challenges in designing a black-box identity-testing algorithm for depth-5 +-regular circuits and is one of the main differences between depth-3 and depth-5 +-regular circuits (small depth +-regular circuits,

in general).

To address this issue, we first convert each Q_{ij} into a more *structured n.c. polynomial*, which we refer to as a *k-ordered power-sum polynomial* (see Definition 4.0.1), where k is the number of variables. This new n.c. polynomial has the property that we can consider it as a commutative polynomial preserving non-zeroness (Claim 4.0.1). Note that when the variables in any n.c. linear form is considered commutative, then non-zeroness is preserved. However, this may not be true for n.c. polynomials, in general. For example, the polynomial $(xy - yx)$ is 0 if x and y are commuting variables. This property (Claim 4.0.1) alone will not let us consider the whole polynomial as commutative. This is because we are dealing with the sum of the product of Q_{ij} polynomials and the number of terms in the product in general could be exponential. Thus, we can not use a fresh set of variables for each *k-ordered power-sum polynomial* as this will have exponentially many *new* variables, in general. Because of this, we use the *same k variables* to convert each Q_{ij} in the product to a *k-ordered power-sum polynomial*, instead of using a fresh set of k variables for each Q_{ij}. If we simply consider the resulting product as commutative, then variables that belong to k-ordered power-sum polynomials of different Q_{i,j_1} and Q_{i,j_2} in the product, could mix (i.e., exponents of the same variable appearing in different k-ordered power-sum polynomials gets added). As a result, we cannot guarantee that the resulting commutative polynomial will preserve non-zeroness.

To address this issue, we will not be considering the modified polynomial as commutative at this stage. Instead, we will transform each Q_{ij} polynomial into a more structured n.c. polynomial, which we denote by $\hat{Q}_{ij}$, by introducing a *new* set of n.c. variables. We show that this transformation preserves non-zeroness.

The substitution automaton we use for the first step generates some spurious monomials along with a structured polynomial (Claim 4.1.4). The spurious monomials are produced because we cannot definitively identify the boundaries of each Q_{ij} polynomial. We can consider spurious monomials as noise generated by our approach.

Let F_1 be the sum of all spurious monomials produced and $\hat{f}_1$ be the structured polynomial resulting from this transformation. We will prove that $\hat{f}_1 + F_1 \not\equiv 0$ (see Lemma 4.1.1 and Claim 4.1.5). The key to this proof lies in demonstrating that the spurious monomials possess a distinctive property, allowing us to differentiate them from the structured part.

After completing Step 1, we can transform the polynomial computed by the depth-5 circuit into a combination of a structured part and a spurious part. One of the outcomes of this first step is that we can efficiently identify the boundaries of the $\hat{Q}_{ij}$ polynomials in the structured part, particularly using a small automaton, even though this process introduces some spurious monomials. A similar concept of boundary also applies to the monomials in the spurious part, and we show that these boundaries can also be identified using a small automaton. Another outcome is that each $\hat{Q}_{ij}$ polynomial can be treated as commutative without resulting in a zero polynomial. These two outcomes are crucial for the remaining steps of our method.

1.1.2 Step 2: Product Sparsification

In the second step, we demonstrate that within the polynomial $\hat{f}_1$ if we treat a small number of the $\hat{Q}_{ij}$ polynomials as n.c. while considering the rest as commutative, non-zeroness is preserved. A key aspect of this step is the ability to treat each $\hat{Q}_{ij}$ polynomial as commutative. Although treating n.c. polynomials as commutative while preserving non-zeroness is generally not feasible, in Step 1, we transformed f into a sum of a structured part $\hat{f}_1$ and a spurious part F_1. The n.c. polynomials $\hat{Q}_{ij}$ within $\hat{f}_1$ can be treated as n.c. without resulting in a zero polynomial (as established in Claim 4.0.1). We then show that the n.c. polynomial obtained after this transformation remains non-zero (see Lemma 4.2.1), a result we refer to as *product sparsification*. We call this sparsification because

we reduce the number of n.c. $\hat{Q}_{ij}$ polynomials in each product, which can be exponential, to a small number of them while preserving non-zeroness. It is important to note, however, that the sum of the exponents of the n.c. variables (a.k.a *n.c. degree*) in this new polynomial can still be generally exponential.

It's important to highlight that this product sparsification step affects both the structured part $\hat{f}_1$ and the spurious part F_1 of the polynomial obtained after Step 1. Let $\hat{f}_2$ and F_2 represent the polynomials derived from $\hat{f}_1$ and F_1 respectively as a result of this product sparsification step. We first consider the product sparsification of the n.c. polynomial $\hat{f}_1$. We will return to $\hat{f}_2 + F_2$ later.

1.1.3 Step 3: Commutative Transformation

In the third and final step, we show that the sum of products of a small number of structured $\hat{Q}_{ij}$ n.c. polynomials can be transformed into a commutative polynomial while preserving non-zeroness (see Lemma 4.3.1). As mentioned earlier, the n.c. degree of the polynomial obtained after Step 2 can be exponential in general. However, we show that this exponential degree n.c. polynomial can be converted into a commutative polynomial using only a small number of *new* commutative variables. It is noteworthy that there is no known method to convert a general exponential degree n.c. polynomial into a commutative polynomial with just a small number of commutative variables while preserving non-zeroness.

Such transformations are known only when the number of non-zero monomials is bounded single-exponentially (a.k.a sparsity of the polynomial) or when the polynomial is computed by a $\Sigma\Pi^*\Sigma$ circuit [AJMR19]. The key to our transformation lies in the structured nature of the $\hat{Q}_{ij}$ polynomials, each of which requires only a small number of *new commutative variables*. Since the number of n.c $\hat{Q}_{ij}$ polynomials is small, we conclude that we have to introduce only a small number of *new commutative variables* for this transformation while ensuring non-zeroness is maintained.

It's important to highlight that this commutative transformation step also affects both the structured part $\hat{f}_2$ and the spurious part F_2 of the polynomial derived after Step 2. Let $\hat{f}_3^{(c)}$ and $F_3^{(c)}$ represent the polynomials derived from $\hat{f}_2$ and F_2 respectively as a result of this commutative transformation step. We are ready to consider the sum of the structured part and spurious part now.

1.1.4 Efficient Coefficient Modifications

Both Steps (2) and (3) will be applied to the structured polynomial $\hat{f}_1$ and the spurious polynomial F_1 obtained after Step 1. Let $\hat{f}_3^{(c)}$ and $F_3^{(c)}$ denote the respective commutative polynomials obtained after applying these two steps to $\hat{f}_1$ and F_1. Since the same commutative variables are used, it is possible for monomials in $\hat{f}^{(c)}$ and $F^{(c)}$ to cancel each other.

If $\hat{f}_3^{(c)} + F_3^{(c)} \neq 0$, we have successfully transformed the exponential degree n.c. polynomial into a commutative polynomial using only a small number of commutative variables while preserving non-zeroness. We can then evaluate the resulting commutative polynomial $\hat{f}_3^{(c)} + F_3^{(c)}$ by randomly chosen matrices for non-zeroness.

On the other hand, if $\hat{f}_3^{(c)} + F_3^{(c)} = 0$, we know that the transformations in Steps (2) and (3) carried out only on the structured part $\hat{f}_1$ obtained from Step (1) ensure that $\hat{f}_3^{(c)} \neq 0$, which implies that $F_3^{(c)} \neq 0$. If a group of n.c. monomials in $\hat{f}_1$ transform into a commutative monomial m with coefficient α_m in $\hat{f}_3^{(c)}$ after Steps (2) and (3) then another group of n.c. monomials in F transformed into the *same* commutative monomial m with the coefficient $-\alpha_m$ in $F_3^{(c)}$.

To address this cancellation issue, we show that we can carefully modify the coefficients of at least one such group of n.c. monomials in the polynomial $\hat{f}_1 + F_1$ obtained after Step 1 before executing Steps (2) and (3). The key to this coefficient modification lies in the fact that the spurious monomials introduced in Step (1) have a distinctive property, enabling us to differentiate them from the structured part using a small automaton. We show that such a coefficient modification preserves non-zeroness (see Lemma 4.4.1). We then establish that if we apply Steps (2) and (3) on this newly modified polynomial, we get a commutative polynomial while preserving non-zeroness.

Though our transformation of a non-commutative polynomial into a commutative one involves modifying the monomial coefficients using substitution matrices obtained from an automaton, it does not turn a zero polynomial into a non-zero one. Since we apply the same matrix substitution for each occurrence of a given variable, monomials that cancel before the transformation will continue to cancel afterward, as they are affected in the same way. This property is crucially used in applying our method inductively for higher depths.

1.1.5 Matrix Compositions

Each of these steps (Steps (1)-(3) and coefficient modification step) is performed using a small substitution automaton, which defines a substitution matrix for each n.c. variable. Throughout the process, we obtain four different sets of matrices. It's important to note that the matrices used in each step evaluate a n.c. polynomial obtained from the previous step.

As our model is black-box, it is not possible to evaluate the polynomial in this way. We need a single matrix substitution for each n.c. variable. We show that the substitution matrices used across these four steps can be combined into a single matrix for each n.c. variable (see Lemma 3.0.1).

This lemma on matrix composition enables us to establish the existence of small substitution matrices for testing non-zeroness in a sequence of steps, which is a key novelty and contribution of this work. This enables us to develop an efficient randomized polynomial identity testing (PIT) algorithm for depth-5 +-regular circuits (see Theorem 2). We further extend this idea to higher depths to develop an efficient randomized PIT algorithm (see Theorem 5).

2 Preliminaries

2.1 Substitution Automaton

The paper uses the standard definition of substitution automaton from automata theory using the terminology of [AJR16, AJR18, AJMR19].

Definition 2.1.1 (Substitution NFA)**.** *A finite* nondeterministic substitution automaton *is a finite nondeterministic automaton $\mathcal{A}$ along with a substitution map*

$$\psi : Q \times X \to \mathcal{P}(Q \times Y \cup \mathbb{F}),$$

where Q is the set of states of $\mathcal{A}$, and Y is a set of variables and $\mathcal{P}(\mathcal{S})$ is the power set of $\mathcal{S}$. If $(j, u) \in \psi(i, x)$ it means that when the automaton $\mathcal{A}$ in state i reads variable x it can replace x by $u \in Y \cup \mathbb{F}$ and can make a transition to state $j \in Q$. In our construction, Y consists of both commuting and noncommuting variables.

Now, for each $x \in X$ we can define the transition matrix M'_x as follows:

$$M'_x(i, j) = u, 1 \leq i, j \leq |Q|, \text{ where } (j, u) \in \psi(i, x). \tag{2}$$

The substitution map ψ can be naturally extended to handle strings as follows: $\hat{\psi} : Q \times X^* \to \mathcal{P}(Q \times (Y \cup \mathbb{F})^*)$. For a state $j \in Q$ and string $m' \in (Y \cup \mathbb{F})^*$, if $(j, m') \in \hat{\psi}(i, x)$, then it means that the automaton starting at state i, on input string $m \in X^*$, can nondeterministically move to state j by transforming the input string m to m' on some computation path.

Now, we explain how a substitution automaton $\mathcal{A}$ processes a given polynomial. First, we will describe the output of a substitution automaton $\mathcal{A}$ on a monomial $w \in X^D$. Let s and t be the designated initial and final states of $\mathcal{A}$, respectively. For each variable $x \in X$, we have a transition matrix M'_x of size $|Q| \times |Q|$. If $w = x_{j_1} x_{j_2} \dots x_{j_d}$, then the output of the substitution automaton $\mathcal{A}$ on the monomial w is given by the (s,t)-th entry of the matrix $M'_w = M'_{x_{j_1}} M'_{x_{j_2}} \dots M'_{x_{j_D}}$. Note that this entry can be a polynomial since there could be multiple paths from s to t labeled by the same monomial w.

2.2 Kronecker Product and the Mixed Product Property

We will crucially use the Kronecker product of matrices and the mixed product property of the Kronecker product in the proof of Lemma 3.0.1. We define them below.

Definition 2.2.1. *(Kronecker Product of Matrices) Let A be a $m \times n$ matrix and let B be a $p \times q$ matrix. Then their Kronecker product $A \otimes B$ is defined as follows.*

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix}$$

where a_{ij} are the entries of matrix A.

Lemma 2.2.1. *(Mixed Product Property) Let A, B, C, D be matrices of dimensions such that the matrix products AB and CD can be defined. Then, the mixed product property states that:*

$$(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D).$$

2.3 The Polynomial Identity Lemma

We state the well-known DeMillo-Lipton-Schwartz-Zippel lemma (a.k.a The Polynomial Identity Lemma) for commutative polynomials.

Lemma 2.3.1 (DeMillo-Lipton-Schwartz-Zippel Lemma [DL78, Zip79, Sch80])**.** *Let $f(x_1, x_2, \dots, x_n)$ be a non-zero degree D polynomial over a field $\mathbb{F}$, and let $S \subseteq \mathbb{F}$ be a finite subset. If we choose a uniformly at random point $a = (a_1, a_2, \dots, a_n) \in S^n$ then*

$$Pr[f(a_1, a_2, \dots, a_n) = 0] \leq D/|S|.$$

3 Composing Substitution Matrices

In our proof outline, we discussed the process of transforming the polynomial computed by a depth-5 +-regular circuit into a commutative polynomial. Our method involves a series of transformations of the polynomial computed by a depth-5 +-regular circuit (or any constant depth +-regular circuit, in general). The final result is a commutative polynomial that maintains non-zeroness.

Each transformation uses substitution matrices obtained from the corresponding substitution automaton. It is important to note that a sequence of matrix substitutions cannot be used in our

black-box model, as it requires a single matrix substitution for each input variable. We show that this sequence of substitutions can be combined into the substitution of a single matrix.

We formally present this in the following lemma, and the proof can be found in Appendix E. The proof relies on the Kronecker product, and in particular, on the mixed product property of the Kronecker product and matrix multiplication (see §2.2).

Lemma 3.0.1 (Composing K Substitution Matrices)**.** *Let $f \in \mathbb{F}\langle X\rangle$ be a non-zero n.c. polynomial of degree D over $X = \{x_1, \ldots, x_n\}$. Let* K *be a natural number. For each $2 \le i \le \mathrm{K}+1, n_i \in \mathbb{N}$, define sets of non-commutative variables $Z_i = \{z_{i1}, \ldots, z_{in_i}\}$, where $n_i \in \mathbb{N}$.*

Assume we have matrices $A_i = (A_{i1}, A_{i2}, \ldots, A_{in_i})$ of dimension d_i for each $2 \le i \le \mathrm{K}+1$. For each $j \in [n_i]$, the matrix A_{ij} belongs to $\mathbb{F}^{d_i \times d_i}\langle Z_{i+1}\rangle$ and is expressed as:

$$A_{ij} = \sum_{k=1}^{n_{i+1}} A_{ij}^{(k)} z_{i+1,k},$$

where $A_{ij}^{(k)} \in \mathbb{F}^{d_i \times d_i}$. Set $n_1 = n$ and define $f_0 = f$. For $i \ge 1$, let f_i be the $[1, d_i]$-th entry of $f_{i-1}(A_{i1}, A_{i2}, \ldots, A_{in_i})$ for all $i \ge 1$.

There exists a matrix substitution $C = (C_1, C_2, \ldots, C_n)$ where each C_i is of dimension $\left(\prod_{i\in[\mathrm{K}]} d_i\right)$ such that the polynomial f_{K} is equivalent to the $\left[1, \prod_{i\in[\mathrm{K}]} d_i\right]$-th entry of the matrix $f(C_1, C_2, \ldots, C_n)$.

In our application of the above lemma, the final substitution matrices $A_{\mathrm{K}+1}$ only contain commutative variables as their entries. Consequently, the resulting polynomial f_{K} is a commutative polynomial in these variables.

4 Black-Box PIT for $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ Circuits

In this section, we show that if $f \in \mathbb{F}\langle X\rangle$ is a n.c. polynomial of degree D computed by a depth-5 $+$-regular circuit of size s, then there exists an efficient randomized algorithm for identity testing the polynomial f. The main result of this section is the following theorem.

Theorem 2. *Let f be a n.c. polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then $f \not\equiv 0$ if and only if f is not identically zero on the matrix algebra $\mathbb{M}_{s^6}(\mathbb{F})$.*

The polynomial f can be written as follows: $f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{ij}$. Here, the degree of each Q_{ij} is D_1, where $i \in [s], j \in [D_2]$, and it can be computed by a $\Sigma\Pi^*\Sigma$ circuit of size at most s. We establish the Theorem 2 in three steps. First, we transform each polynomial Q_{ij} into a more structured n.c. polynomial, as defined below (see Definition 4.0.1). This structured polynomial has the property that we can simply consider it as a commutative polynomial preserving non-zeroness (Claim 4.0.1). As noted above, this is not true for n.c. polynomials, in general.

Definition 4.0.1. *Let $s \in \mathbb{N} \cup \{0\}$. We call a n.c. polynomial g over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ as an s-ordered power-sum polynomial if it is of the form*

$$g = \sum_{i_1 \ge 0, \ldots, i_s \ge 0} \alpha_{\bar{i}} . \xi_1^{i_1} \xi_2^{i_1} \ldots \xi_s^{i_s}, \text{ where } \alpha_{\bar{i}} \in \mathbb{F}.$$

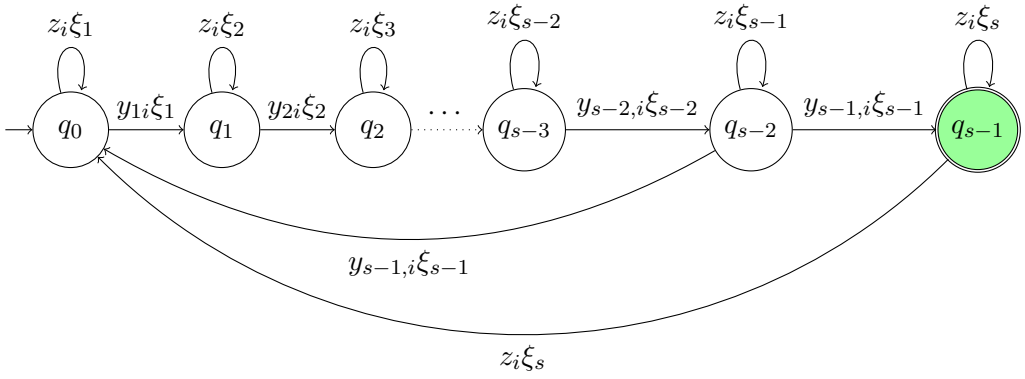

Figure 1: The transition diagram for the variable $x_i : 1 \leq i \leq n$

Remark 4.0.1. *In the above definition, it is important to note that $i_j \geq 0$ for each $j \in [s]$. However, in this paper, we will focus on a special case of s-ordered power-sum polynomials, where $i_j > 0$, for each $j \in [s-1]$ and $i_s \geq 0$.*

We have the following observation about the *s-ordered power-sum polynomial* and a proof of this can be found in Appendix G.1.

Claim 4.0.1. *Suppose $g \in \mathbb{F}\langle\xi\rangle$ is an s-ordered power-sum polynomial. Let $g^{(c)}$ be the polynomial obtained by treating variables in ξ as commutative. Then, $g \not\equiv 0$ if and only if $g^{(c)} \not\equiv 0$.*

Remark 4.0.2. *We will later allow coefficients $\alpha_{\bar{i}}$ to be commutative polynomials. The proof of this generalized statement follows the same reasoning as the proof of Claim 4.0.1.*

Let us define the following concept, which will be relevant in subsequent sections:

Definition 4.0.2 (ξ-Pattern)**.** *We refer to the string $\xi_1^{\ell_1}\xi_2^{\ell_2}\cdots\xi_s^{\ell_s}$ as a* ξ-pattern*, regardless of the specific exponents of the ξ variables.*

We now explain the three steps of our method to establish Theorem 2.

4.1 Step 1: Transforming the Polynomial for Improved Structure

The initial step of our method involves transforming the polynomial to introduce more structure. During this process, we obtain a structured polynomial but also introduce some additional spurious monomials. This is one of the main differences between this work and [AJMR19]. We show that these spurious monomials have a distinguishing property that can be used to differentiate them from the structured part. We discuss the process of transforming each polynomial Q_{ij} into an s-ordered power-sum polynomial. To do this, we introduce a *new* set of commutative and n.c. variables as follows.
Let $Z = \{z_1, \ldots, z_n\}$ and let $Y = \{y_{ij} \mid i \in [n] \text{ and } j \in [s-1]\}$. The variables in Y and Z are commutative. Let $\xi = \{\xi_1, \xi_2, \cdots, \xi_s\}$ be the set of n.c. variables.

4.1.1 Idea of the Substitution Automaton

As each Q_{ij} polynomial can be computed by a $\Sigma\Pi^*\Sigma$ circuit of size bounded by s, it is natural to try to use the PIT results that exist for $\Sigma\Pi^*\Sigma$ circuits. In [AJMR19], it was shown that there

is a small substitution automaton that transforms the polynomial into a commutative polynomial while preserving non-zeroness. We can try to transform each Q_{ij} polynomial into a commutative polynomial using the result of [AJMR19]. This approach presents two issues.

1. As explained in the proof outline, one would have to identify the *boundary* between Q_{ij} and $Q_{i,j+1}$ (i.e., the position at which Q_{ij} ends and $Q_{i,j+1}$ begins), for any $i \in [s], j \in [D_2 - 1]$.

2. We need to ensure that the resulting commutative polynomial is non-zero because the same commutative variables appearing in different transformed Q_{ij} polynomials get mixed (i.e., exponents of the same variable appearing at different transformed Q_{ij} polynomials are added).

As the degree of Q_{ij} could be exponential in s, it is not feasible to detect the *boundary* by counting. So we *guess* the boundary using a substitution automaton. The two states q_{s-2} and q_{s-1} of the substitution automaton given in Figure 1 guess the boundary between Q_{ij} and $Q_{i,j+1}$. We need two states for guessing the boundary. This is because in [AJMR19], it was proved that for polynomials computed by the $\Sigma\Pi^*\Sigma$ circuits, there is a small index set I such that if we consider all linear forms appearing at positions indexed by I as n.c. and rest as commutative then non-zeroness is preserved. Depending on whether the position of the last linear form is part of I or not, the automaton is either in state q_{s-2} or q_{s-1}, respectively.

We consider the output of the substitution automaton given in Figure 1 on the n.c. polynomial f. The $s \times s$ substitution matrix $\mathbf{M_{x_i}}$ for each variable x_i is defined from Figure 1, as follows:

$$\mathbf{M_{x_i}} = \begin{pmatrix} z_i\xi_1 & y_{1i}\xi_1 & 0 & \dots & 0 & 0 \\ 0 & z_i\xi_2 & y_{2i}\xi_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ y_{s-1,i}\xi_{s-1} & 0 & 0 & \dots & z_i\xi_{s-1} & y_{s-1,i}\xi_{s-1} \\ z_i\xi_s & 0 & 0 & \dots & 0 & z_i\xi_s \end{pmatrix}$$

It is helpful to consider $\mathbf{M_{x_i}}$ as the sum of matrices as follows. This view is useful when we compose matrices (see Lemma 3.0.1) from all three steps to obtain a single matrix later on.

$$\mathbf{M_{x_i}} = \begin{pmatrix} z_i & y_{1i} & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \cdot \xi_1 + \ldots + \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \\ z_i & 0 & 0 & \dots & 0 & z_i \end{pmatrix} \cdot \xi_s$$

Output of the Automaton

Let $\mathbf{M} = f(\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \dots, \mathbf{M_{x_n}})$. Then we consider the *output of the automaton* as:

$$f' = \mathbf{M}[q_0, q_{s-1}] \tag{3}$$

which is a polynomial in the variables $\xi \sqcup Y \sqcup Z$. This differs from previous works ([AJMR19, BW05]) where only a single matrix entry was used as the output.

Suppose a monomial m is computed by a +-regular circuit C. The monomial m has non-zero coefficient in $\prod_{j \in [D_2]} Q_{ij}$ for some $i \in [s]$. This monomial can be written as $m = m_1 \cdot m_2 \cdots m_{D_2}$, where each sub-monomial $m_j \in X^{D_1}$ has non-zero coefficient in Q_{ij}.

Next, we consider the output of the substitution automaton $\mathcal{A}$ on m. The automaton knows how to replace any variable x_j at any state q. For simplicity, Figure 1 illustrates only the information

for variable x_i. Suppose $m = x_{i_1}.x_{i_2}.x_{i_3}\dots x_{i_D}$, the output of the substitution automaton $\mathcal{A}$ on the monomial m is given by

$$\mathbf{M_m}[q_0, q_{s-1}]$$

where $\mathbf{M_m} = \mathbf{M_{x_{i_1}}} \cdot \mathbf{M_{x_{i_2}}} \cdots \mathbf{M_{x_{i_D}}}$. Each variable $x_i, i \in [n]$, is substituted by a degree two monomial over $\xi \sqcup Y \sqcup Z$ (one n.c. variable and one commutative variable). Consequently, the automaton transforms the monomial m into a degree $2D$ polynomial over $\xi \sqcup Y \sqcup Z$. Importantly, the *new* n.c. degree (i.e., sum of exponents of ξ variables) equals to the original degree D.

Computation by the substitution automaton

The automaton has exponentially many paths (with states allowed to repeat) from q_0 to q_{s-1}, all labeled by the same monomial m. Each computation path transforms the monomial m, originally over the variables X, into a new monomial over $\xi \sqcup Y \sqcup Z$.

For any path ρ from q_0 to q_{s-1}, we denote the transformed monomial as m_ρ. The polynomial computed by $\mathbf{M_m}[q_0, q_{s-1}]$ is given by

$$\sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} m_\rho,$$

which is a polynomial in $\mathbb{F}[Y \sqcup Z]\langle\xi\rangle$. Recall that n.c. polynomials are $\mathbb{F}$-linear combinations of words/strings (called monomials). For a n.c. monomial m, we can identify the variable at position c in m, where $1 \le c \le |m|$.

Recall that m can be written as $m = m_1 \cdot m_2 \cdots m_{D_2}$. Each computation path ρ substitutes each n.c. variable in m according to the automaton's transition rules, resulting in a monomial m_ρ over new variables $\xi \sqcup Y \sqcup Z$. We group all commutative variables appearing in m_ρ and denote it by c_m, which is a commutative monomial over $Y \sqcup Z$. The resulting monomial m_ρ has the following form and the proof can be found in Appendix F

Proposition 4.1.1. *Let ρ be a path from q_0 to q_{s-1} labeled by the monomial m. The transformed monomial m_ρ can be expressed in the form: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$, where $N \ge 1$, c_m is a monomial over $Y \sqcup Z$, and each m'_ℓ is given by $m'_\ell = \xi_1^{\ell_1}.\xi_2^{\ell_2} \cdots \xi_s^{\ell_s}$, where $\ell_k > 0, k \in [s-1]$ and $\ell_s \ge 0$.*

For $i \ne j$, the exponents of the ξ variables in the sub-monomials m'_i and m'_j can vary. In particular, it is generally possible that $(i_1, i_2, \dots, i_s) \ne (j_1, j_2, \dots, j_s)$.

Types of sub-monomials: Two cases

It is important to note that the number of new sub-monomials m'_i, denoted as N, may not be equal to D_2. This is because N depends on how many times the path ρ returns to the initial state q_0. Also, the sum of exponents of ξ variables in each sub-monomial m'_i in m_ρ can vary. This leads us to consider two possible cases for each computation path ρ: (recall $m = m_1 \cdot m_2 \cdots m_{D_2}$).

- **Case 1:** For each $j < D_2$, the boundary between m_j and m_{j+1} in m is respected by the path ρ. In this computation path ρ, the state of $\mathcal{A}$ is at q_0 precisely when it begins processing each sub-monomial $m_j \in X^{D_1}$ for $j \in [D_2]$. This means that when $\mathcal{A}$ reads the last variable of the sub-monomial m_{j-1} (for $j > 1$), it transitions back to state q_0. As a result, $\mathcal{A}$ is in state q_0 exactly when it reads the first variable of the sub-monomial m_j. This holds true for all sub-monomials m_j where $j \in [N]$. By Proposition 4.1.1, the transformed monomial can be expressed as: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$ where c_m is a monomial over $Y \sqcup Z$ and each m'_ℓ

is of form $m'_\ell = \xi_1^{\ell_1} \cdots \xi_s^{\ell_s}$. In this case, we observe that $N = D_2$ since there are exactly D_2 sub-monomials in m. *or*

- **Case 2:** For some $j < D_2$, the boundary between m_j and m_{j+1} in m is not respected by the path ρ. In this case, there exists a sub-monomial m_j, where $j \in [D_2]$, such that either (1) the computation path ρ visits the state q_0 while processing the variable located at position c, where $1 < c \leq D_1$. This means ρ returns to q_0 in the middle of processing m_j. *or* (2) the path ρ is in a state $q_j, j \neq 0$ (i.e., other than the initial state q_0) while processing the variable that appears at the first position of the sub-monomial m_j. By Proposition 4.1.1, the transformed monomial can be expressed as: $m_\rho = c_m \cdot m'_1 \cdot m'_2 \cdots m'_N$ where c_m is a monomial over $Y \sqcup Z$ and each m'_ℓ is of form $m'_\ell = \xi_1^{\ell_1} \cdots \xi_s^{\ell_s}$. In this case, we cannot definitively say whether N is equal to D_2 or not.

Remark 4.1.1. *Any path ρ from q_0 to q_{s-1} labeled by a monomial $m \in X^D$ will satisfy either Case 1 or Case 2, but not both.*

In Case 1, we can make the following important observation about the obtained monomial m_ρ and the proof can be found in Appendix F. Recall that D_1 is the degree Q_{ij} polynomial.

Claim 4.1.1. *Let ρ be a path from q_0 to q_{s-1} labeled by the monomial m that satisfies Case 1. In this case, for each sub-monomial m'_ℓ, where $\ell \in [D_2]$, of the monomial m_ρ, the sum of the exponents of its n.c. variables is D_1. That is, $\sum_{j\in[s]} \ell_j = D_1$.*

For all paths ρ that satisfy Case 2, this is not true. We note this down as the following claim and the proof can be found in Appendix F.

Claim 4.1.2. *Let ρ be apath from q_0 to q_{s-1} labeled by the monomial m that satisfies Case 2. In this case, there exists a sub-monomial m'_ℓ, where $\ell \in [D_2]$, in the obtained monomial m_ρ such that the sum of the exponents of its n.c. variables is* not *equal to D_1. That is, $\sum_{j\in[s]} \ell_j \neq D_1$.*

We crucially utilize Claims 4.1.1 and 4.1.2 later to ensure the non-zeroness of the transformed commutative polynomial.

The *structured* part and the *spurious* part

For a monomial $m = m_1 \cdot m_2 \cdots m_{D_2}$, we define the polynomial $\hat{f}_m$ as the sum of all monomials that are obtained from computation paths ρ labeled by m from Case 1 above. Similarly, the polynomial F_m is defined as the sum of all monomials obtained from computation paths ρ labeled by m from Case 2 above. We consider the output of the substitution automaton $\mathcal{A}$ on the given n.c. polynomial $f \in \mathbb{F}\langle X \rangle$.

The output of the automaton is the sum of all monomials produced by computation paths ρ starting from q_0 and leading to q_{s-1}, with these paths labeled by monomials generated by a depth-5 +-regular circuit.

Let $Mon(f)$ be the set of all monomials computed/generated by the given depth-5 +-regular circuit computing f. That is, suppose m is computed by $\prod_{j\in[D_2]} Q_{i,j}$ for some $i \in [s]$, with coefficient $\alpha_{m,i}$ then $\alpha_{m,i} \cdot m \in Mon(f)$. Let

$$\hat{f} = \sum_{\alpha_{m,i}\cdot m \in Mon(f)} \hat{f}_{\alpha_{m,i}\cdot m} \quad \text{and } F = \sum_{\alpha_{m,i}\cdot m \in Mon(f)} F_{\alpha_{m,i}\cdot m}.$$

We refer to F as the sum of spurious monomials obtained from the automaton, which can be viewed as noise resulting from our method.

We assume that the linear forms in the $Q_{i,j}$ polynomials are numbered from 1 to D_1. For $I \subseteq [D_1]$ with size at most $s-1$, define $Q_{i,j,I}$ as the polynomial obtained from $Q_{i,j}$ by treating linear forms indexed by I as noncommuting and the rest of the linear forms as commuting.

We have the following proposition regarding the polynomial $\hat{f}$. The proof can be found in Appendix F.

Claim 4.1.3. *The polynomial $\hat{f}$ can be expressed as*

$$\hat{f} = \sum_{i\in[s]} \prod_{j\in[D_2]} \sum_{I\subseteq[D_1],|I|=s-1} Q_{i,j,I} \times \xi_I$$

where $\xi_I = \xi_1^{\ell_1}.\xi_2^{\ell_2-\ell_1}\cdots\xi_s^{D-\ell_{s-1}}$ for $I = \{\ell_1, \ell_2, \cdots, \ell_{s-1}\}$ such that $\ell_1 < \ell_2 < \cdots < \ell_{s-1}$.

Let

$$\hat{Q}_{ij} = \sum_{I\subseteq[D_1],|I|=s-1} Q_{i,j,I} \times \xi_I.$$

Then we can express $\hat{f}$ as:

$$\hat{f} = \sum_{i\in[s]} \left(\prod_{j\in[D_2]} \hat{Q}_{ij} \right).$$

The output of the substitution automaton $\mathcal{A}$ on the polynomial f is given by:

$$f' = \hat{f} + F.$$

This is stated in the following claim. The proof can be found in Appendix F.

Claim 4.1.4. *Let f be a homogeneous n.c. polynomial computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then, the output $f' \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ of the substitution automaton $\mathcal{A}$ on the polynomial f can expressed as $f' = \hat{f} + F$.*

4.1.2 Non-zeroness of $\hat{f}$

We establish that f' is non-zero by first proving that $\hat{f}$ is not zero. This is shown in Lemma 4.1.1, which builds on the result of PIT for $\Sigma\Pi^*\Sigma$ circuits (see Section 6.2 in [AJMR19]). We briefly discuss this result.

Let $Z = \{z_1, \ldots, z_n\}$ be the set of *new* commuting variables. Let $g \in \mathbb{F}\langle X\rangle$ be a polynomial of degree D computed by a $\Sigma\Pi^*\Sigma$ circuit of size s. Then g can be expressed as $g = \sum_{i\in[s]} \prod_{j\in[D]} L_{ij}$, where L_{ij} are homogeneous linear forms. Let $P_i = \prod_{j\in[D]} L_{ij}$, $i \in [s]$. We have $g = \sum_{i\in[s]} P_i$. For $I \subseteq [D]$ with size at most $s-1$, define $P_{i,I}$ as the polynomial obtained from P_i by treating linear forms indexed by I as noncommuting and the rest of the linear forms as commuting. We replace x_i variable appearing in $[D] \setminus I$ by a *new* commuting variable z_i.

The number of n.c. variables appearing in $P_{i,I} \in \mathbb{F}[Z]\langle X\rangle$ is bounded by $|I| < s$. We refer to this as the n.c. degree of the polynomial $P_{i,I}$. Since this degree is small, $P_{i,I}$ can be converted into a commutative polynomial while preserving its non-zeroness. Let $P_{i,I}^{(c)}$ denote the commutative polynomial obtained from $P_{i,I}$ and define $g_I = \sum_{i\in[s]} P_{i,I}^{(c)}$. To keep all guesses of the set I distinct, additional commutative variables $\xi = \{\xi_1, \xi_2, \cdots, \xi_{k+1}\}$ are introduced in [AJMR19]. The

transformed commutative polynomial obtained in [AJMR19] is given by:

$$g^\star = \sum_{I\subseteq[D_1],|I|=k} g_I \times \xi'_I \tag{4}$$

where $\xi'_I = \xi_1^{\ell_1-1}.\xi_2^{\ell_2-\ell_1-1}\cdots\xi_{k+1}^{D-\ell_k}$ with $I = \{\ell_1, \ell_2, \cdots, \ell_k\}$. The degree of the monomial ξ'_I is $D - |I|$.

By Lemma 6.2 in [AJMR19], there exists a set of indices $I \subseteq [D]$, $|I| < s$, such that $g_I \neq 0$ implying $g^\star \neq 0$. Replacing ξ'_I with $\xi_I = \xi_1^{\ell_1}.\xi_2^{\ell_2-\ell_1}\cdots\xi_{k+1}^{D_1-\ell_k}$ in $g^\star$ retains the non-zeroness of $g^\star$ while the degree of ξ_I becomes D.

Remark 4.1.2. *1. Without loss of generality, we assume that the automaton nondeterministically guesses exactly $(s-1)$ indices, i.e., $|I| = s-1$ and the rest as commutative. If $|I| < s-1$, adding more indices still preserves non-zeroness.*

2. If the degree of the polynomial g is smaller than $(s-1)$, we will handle this small-degree case separately (See §4.1.3). For now, we assume $D_1 \geq s-1$ in Lemma 4.1.1.

We have the following lemma that shows $\hat{f} \neq 0$. The proof can be found in Appendix A.

Lemma 4.1.1. *Let $f = \sum_{i\in[s]}\prod_{j\in[D_2]} Q_{ij}$ be a n.c. polynomial over $X = \{x_1, \cdots, x_n\}$, computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Let D_1 denote the degree of the polynomial Q_{ij}, $i \in [s], j \in [D_2]$. Let $\hat{f} \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ be defined as $\hat{f} = \sum_{i\in[s]}\prod_{j\in[D_2]}\hat{Q}_{ij}$, where $\hat{Q}_{ij}$ are as defined earlier. Then, if $f \neq 0$ then $\hat{f} \neq 0$.*

This above lemma 4.1.1 can be generalized. We will utilize this generalization to transform polynomials computed by larger depth +-regular circuits. We note this as a remark.

Remark 4.1.3. *The proof of Lemma 4.1.1 is based on the observation that if each polynomial Q_{ij} (where $i \in [s]$ and $j \in [D_2]$, computable by a depth-3 circuit) can be converted into a n.c. polynomial $\hat{Q}_{ij}$ while preserving non-zeroness, then this lemma asserts that this transformation can be applied to each Q_{ij} to create the n.c. polynomial $\hat{f}$, maintaining the non-zeroness of the entire polynomial. This can be extended to higher depths as follows: suppose f is computed by the sum of the products of Q_{ij}'s, where each polynomial Q_{ij} is computed by a depth d +-regular circuit and each Q_{ij} can be converted into a n.c. polynomial $\hat{Q}_{ij}$ while preserving non-zeroness, then each Q_{ij} can be converted to $\hat{Q}_{ij}$ to create the n.c. polynomial $\hat{f}$, maintaining the non-zeroness of the entire polynomial. The proof follows a similar structure to that of Lemma 4.1.1*

The resulting n.c. polynomial $\hat{f}$ still has an exponential degree in ξ variables, but each $\hat{Q}_{ij}$ is structured as s-ordered power-sum polynomials. Importantly, $\hat{f}$ does not contain any monomials from the spurious polynomial $F = \sum_{m\in Mon(f)} F_m$.

4.1.3 Small degree case

It is important to note that we require the degree D_1 of each polynomial Q_{ij} to be at least $s-1$. If $D_1 < s-1$, we can not use Lemma 4.1.1. The advantage of this lemma lies in the fact that in the polynomial $\hat{f}$, each polynomial $\hat{Q}_{ij}$ is an s-ordered power-sum polynomial. By Claim 4.0.1, each $\hat{Q}_{ij}$ can be treated as a commutative polynomial without leading to a zero polynomial. This property is essential for the product sparsification lemma (see Lemma 4.2.1).

When $D_1 < s-1$, we can use a small substitution automaton of size c, bounded by s, to substitute fresh n.c. double-indexed variables at each position within each Q_{ij}. Let $Z = \{z_{\ell k} \mid \ell \in [c] \text{ and } k \in [n]\}$ be the *new* set of n.c. variables. The automaton replaces the variable x_k in the ℓ-th position of a monomial of Q_{ij} with $z_{\ell k}$. The resulting polynomial is denoted by $\hat{Q}_{ij}$, which remains n.c. over Z. Unlike the high-degree case, in the small-degree case, the automaton can identify the *boundary* between Q_{ij} and $Q_{i,j+1}$, ensuring that no spurious monomials are produced, that is, $F = 0$.

The substitution automaton that accomplishes these substitutions and the corresponding substitution matrix for each variable x_j can be found in Appendix 6. The following proposition is true because there is a bijection between monomials of $\hat{Q}_{ij}$ and Q_{ij}.

Proposition 4.1.2. *For $i \in [s], j \in [D_2]$, each Q_{ij} in the polynomial $f \in \mathbb{F}\langle X\rangle$ is transformed into $\hat{Q}_{ij}$ such that $\hat{Q}_{ij} \equiv 0$ if and only if $Q_{ij} \equiv 0$.*

It is easy to observe the following because the first index of each variable $z_{\ell k}$ indicates the position of the variable x_k within each Q_{ij}.

Observation 4.1.1. *Suppose $\hat{Q}_{ij} \neq 0$. If we treat the variables $z_{\ell k}, \ell \in [c], k \in [n]$, appearing in $\hat{Q}_{ij}$ as commuting, the resulting commutative polynomial $\hat{Q}_{ij}^{(c)}$ remains non-zero.*

This guarantees that for the small degree case, we can transform the polynomial f similarly to Lemma 4.1.1, ensuring that each Q_{ij} is transformed into $\hat{Q}_{ij}$ which can be regarded as a commutative polynomial without making it zero. While $\hat{Q}_{ij}$ remains a n.c. polynomial over Z, we acknowledge that our model is black-box and we do not know the value of D_1. However, for the purpose of analyzing the existence of matrices of small dimensions for identity testing, we can assume D_1 is known.

Thus, we can successfully transform the given polynomial in both scenarios – whether $D_1 \geq s-1$ or $D_1 < s-1$ – ensuring that the resulting $\hat{Q}_{ij}$ can be considered as a commutative polynomial without making it a zero polynomial.

However, it is important to note that this transformation alone will not provide a black-box PIT, as we cannot guarantee the non-zeroness of the sum of products of these $\hat{Q}_{ij}$ polynomials if we simply treat all $\hat{Q}_{ij}$ as commutative. At this stage, the variables in Z are still considered n.c. in the polynomial $\hat{f}$.

4.1.4 Non-zeroness of f'

By Lemma 4.1.1, we established that $\hat{f} \not\equiv 0$. Next, we show that the polynomial $f' = \hat{f} + F \not\equiv 0$. In $\hat{f}$, for every monomial $m = m_1 m_2 \cdots m_{D_2}$, and for all $\ell \in [D_2]$ each m_ℓ takes the form $\xi_1^{\ell_1}.\xi_2^{\ell_2} \cdots \xi_s^{\ell_s}$ where $\sum_{k\in[s]} \ell_k = D_1$ (see Claim 4.1.1). However, this property does not hold for monomials appearing in F (see Claim 4.1.2). Specifically, for any monomial $m' = m'_1 m'_2 \cdots m'_N$ in F, there exists a sub-monomial $m'_a = \xi_1^{a_1}.\xi_2^{a_2} \cdots \xi_s^{a_s}$ such that $\sum_{h\in[s]} a_h \neq D_1$. This distinction ensures that the monomials of $\hat{f}$ do not cancel with those of F. Thus, we conclude that $f' = \hat{f} + F \not\equiv 0$. It's important to note that if $f \equiv 0$ then clearly $f' \equiv 0$ as well (converse statement). We note these observations in the following claim.

Claim 4.1.5. *Let f be a homogeneous n.c. polynomial computed by a depth-5 +-regular circuit of size s. Then, $f \not\equiv 0$ if and only if $f' = \hat{f} + F \not\equiv 0$.*

Next, we can simplify the polynomial f' using the Polynomial Identity Lemma for commutative polynomials. We replace the commuting variables $Y \sqcup Z$ with scalar substitutions from $\mathbb{F}$ or an

extension field, yielding a non-zero polynomial. Let us denote this resulting non-zero polynomial as $\tilde{f}$. After this substitution, the only remaining variables in $\tilde{f}$ will be n.c. variables ξ.

Let us denote new polynomials obtained after replacing the commuting variables by scalars in $\hat{f}$ and F by $\hat{f}_1$ and F_1 respectively. That is, $\tilde{f} = \hat{f}_1 + F_1$.

Remark 4.1.4. *It is important to note that each monomial of $\tilde{f}$ is a product of ξ-patterns (see Definition 4.0.2), and the boundaries of each ξ-pattern can be easily identified by an automaton which is crucially used by the remaining steps.*

4.2 Step 2: Product Sparsification

In the second step of our transformation, we focus on the sparsification of the n.c. polynomial $\tilde{f} \in \mathbb{F}\langle\xi\rangle$ with respect to the s-ordered power-sum polynomials $\hat{Q}_{ij}$. This transformation affects both the good part $\hat{f}_1$ and the spurious part F_1 of the polynomial $\tilde{f}$.

We begin by analyzing the transformation of $\hat{f}_1$, which is defined as:

$$\hat{f}_1 = \sum_{i\in[s]} \left(\prod_{j\in[D_2]} \hat{Q}_{ij} \right),$$

where each $\hat{Q}_{ij}$ is an s-ordered power-sum polynomial in the n.c. variables $\xi = \{\xi_1, \ldots, \xi_s\}$.

The key observation is that we can preserve the non-zeroness of $\hat{f}_1$ by retaining at most $s-1$ of the s-ordered power-sum polynomials $\hat{Q}_{ij}$ in each product $\prod_{j\in[D_2]} \hat{Q}_{ij}$ as n.c. while treating the remaining ones as commutative. This is stated in the following lemma, which we refer to as the *product sparsification lemma*. This lemma generalizes Lemma 6.2 from [AJMR19]. Unlike [AJMR19], we are dealing with the product of n.c. polynomials where the degree of individual factors can be greater than 1 if we consider them as commutative they may become 0. The proof of this lemma is crucially used the Claim 4.0.1, and the proof can be found in Appendix B. We will revisit the combination of $\hat{f}_1$ and F_1 in §4.4 to complete our analysis.

Lemma 4.2.1 (Product Sparsification Lemma)**.** *Let*

$$\hat{f}_1 = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij},$$

where each $\hat{Q}_{ij}$ is an s-ordered power-sum polynomial over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$. Then, there exists a subset $I \subseteq [D_2]$ with size at most $s-1$ such that if we treat the polynomials $\hat{Q}_{ij}$ for $j \in I$, as non-commutative and the others($j \notin I$) as commutative, then the polynomial $\hat{f}_1$ remains non-zero.

Remark 4.2.1. *The proof of Lemma 4.2.1 relies only on the fact that each $\hat{Q}_{ij}$ where $i \in [s]$ and $j \in [D_2]$, is an s-ordered power-sum polynomial. The size of the index set I depends only on the number of summands s. Importantly, the proof does not depend on the fact that each $\hat{Q}_{ij}$ is obtained from a polynomial computed by a $\Sigma\Pi^*\Sigma$ circuit. This allows us to generalize the lemma to sums of products of* arbitrary *k-ordered power-sum polynomials where $k \in \mathbb{N}$, with a proof that is analogous to that of the original lemma. We will utilize this generalization to transform polynomials computed by higher-depth $+$-regular circuits.*

Since we do not know the index set I, the substitution guesses the index set I. The substitution automaton that accomplishes this product sparsification can be found in Appendix B.1. Since the index set $I \subseteq [D_2]$ is unknown, the automaton non-deterministically selects which $\hat{Q}_{ij}$ polynomials

will be treated as non-commutative. Given the structured nature of the polynomial $\hat{f}$, we can identify the *boundary* of each $\hat{Q}_{ij}$, ensuring that no additional spurious monomials are generated.

In the high-degree case ($D_1 \geq s-1$), either ξ_s or ξ_{s-1} followed by ξ_1 indicates the end of each $\hat{Q}_{ij}$, which can be easily recognized by the automaton. In the low-degree case ($D_1 < s-1$), the smaller degree allows us to identify the ends of each $\hat{Q}_{ij}$ with a small automaton of size at most $s-2$.

The substitution automaton selects at most $(s-1)$ of the $\hat{Q}_{ij}$ polynomials to be treated as n.c. while treating the remaining ones as commutative. The ξ variables in the chosen commutative polynomials $\hat{Q}_{ij}$ are substituted with fresh commutative variables $\zeta = \{\zeta_1, \ldots, \zeta_s\}$. In the selected commutative polynomials $\hat{Q}_{ij}$, each n.c. variable $\xi_k, k \in [s]$ is replaced by the corresponding commuting variable ζ_k. Additionally, to distinguish between different guesses made by the substitution automaton, we use fresh commutative block variables $\chi = \{\chi_1, \ldots, \chi_s\}$.

Let $J = \{j_1, j_2, \ldots, j_{s-1}\} \subseteq [D_2]$ with $j_1 < j_2 < \ldots < j_{s-1}$. We define $\chi_J = \chi_1^{j_1-1} \cdot \chi_2^{j_2-j_1-1} \ldots \chi_s^{D_2-j_{s-1}}$. If the automaton guesses the $\hat{Q}_{ij}$ polynomials corresponding to the positions in the index set J as n.c., the output g_J of the substitution automaton for this specific guess J will be

$$g_J = \sum_{i \in [s]} \left(\prod_{j \in \overline{J}} \hat{Q}_{ij} \right) \left(\prod_{j \in J} \hat{Q}_{ij} \right) \times \chi_J.$$

Note that $\left(\prod_{j \in \overline{J}} \hat{Q}_{ij} \right)$ is a commutative polynomial over $\zeta = \{\zeta_1, \ldots, \zeta_s\}$. We have the following lemma.

Lemma 4.2.2. *Let $\hat{f}_1 \in \mathbb{F}\langle \xi \rangle$ be the structured part of the polynomial obtained after Step 1. Let $\hat{f}_1'$ be the output of the substitution automaton given in Figure B.1 (can be found in Appendix B) on the structured polynomial $\hat{f}_1$ and it can be expressed as*

$$\hat{f}_1' = \sum_{J \subseteq [D_2], |J| = s-1} g_J \times \chi_J.$$

Moreover, $\hat{f}_1 \neq 0$ if and only if $\hat{f}_1' = 0$.

It's evident that for distinct guesses J and J' where $J \neq J'$, the monomials of g_J and g_J' will not mix, since the sub-monomials χ_J and $\chi_{J'}$ are distinct. By Lemma 4.2.1, there exists index set $J \subseteq [D_2]$ with size at most $s-1$, such that $g_J \neq 0$ implying $\hat{f}_1' \neq 0$.

Next, we can simplify $\hat{f}_1'$ by using the Polynomial Identity Lemma for commutative polynomials to eliminate the commuting variables $\zeta \cup \chi$ by substituting scalars. As a result, the remaining variables in the polynomial will be solely the n.c. variables ξ.

Let us denote the new polynomial obtained after replacing the commuting variables by scalars in $\hat{f}'$ by $\hat{f}_2$.

This product sparsification step affects both the good part $\hat{f}_1$ and the spurious part F_1 of the polynomial $\tilde{f}$ obtained after Step 1. We will denote the new polynomial derived from the spurious part F_1 by F_2. In Step 2, we apply product sparsification to both $\hat{f}_1$ and F_1, which yields the n.c. polynomials $\hat{f}_2$ and F_2 respectively (with all commuting variables replaced by scalars).

4.3 Step 3: Commutative Transformation of $\hat{f}_2$

In this final step, we describe how to transform $\hat{f}_2$ into a commutative polynomial while preserving its non-zeroness. Note that $\hat{f}_2$ is a polynomial over $\mathbb{F}[\zeta \cup \chi]\langle \xi \rangle$. If we treat $\hat{f}_2$ as commutative by

considering the n.c. variables ξ as commutative, the exponents of the variable ξ_i (for $i \in [s]$) from different n.c. $\hat{Q}_{ij}$ polynomials will be summed (or mixed). This mixing makes it impossible to guarantee that the resulting polynomial remains non-zero.

However, we can carefully convert $\hat{f}_2$ into a commutative polynomial while preserving its non-zeroness.

This is stated in the following lemma and the proof can be found in Appendix C. The substitution automaton for this step can be found in Appendix C.1.

Lemma 4.3.1. *Let $g = \sum_{i\in[s]} \beta_i \big(\prod_{j\in[s]} \hat{Q}_{ij}\big)$, where $\beta_i \in \mathbb{F}$ and each $\hat{Q}_{ij}$ is an s-ordered power-sum polynomial over $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ of degree D. This can be expressed as: $g = \sum_m \alpha_m m$, where $m = \big(\prod_{j\in[s]} \xi_1^{i_{j1}} \xi_2^{i_{j2}} \ldots \xi_s^{i_{js}}\big)$. The n.c. polynomial g can be transformed into a commutative polynomial $g^{(c)}$ while preserving its non-zeroness.*

Remark 4.3.1. *Similar to Lemma 4.2.1, Lemma 4.3.1 is more general because its proof relies only on the fact that each $\hat{Q}_{ij}$ polynomial is an s-ordered power-sum polynomial, and the number of products is bounded by some polynomial function in s. This allows us to generalize the lemma to sum of products of* arbitrary *k-ordered power-sum polynomials where $k \in \mathbb{N}$, with a proof that is analogous to that of the original lemma. We will utilize this generalization to transform polynomials computed by higher-depth $+$-regular circuits.*

By applying Lemma 4.3.1, we can transform the polynomial $\hat{f}_2$, the structured part obtained after Step 2, into a commutative polynomial while preserving its non-zeroness. Let $\hat{f}_3^{(c)}$ denote the resulting commutative polynomial derived from $\hat{f}_2$. Consequently, we establish that $\hat{f}_3^{(c)} \not\equiv 0$ as a result of this lemma.

Next, given $\tilde{f} = \hat{f}_1 + F_1$, where $\tilde{f}$ was obtained after Step 1, we can likewise transform $\tilde{f}$ into a commutative polynomial. Let $F_3^{(c)}$ represent the commutative polynomial obtained from F after applying steps (2) and (3). If $\hat{f}_3^{(c)} + F_3^{(c)} \not\equiv 0$, we have successfully converted a n.c. polynomial f, computed by a depth-5 $+$-regular circuit, into a commutative polynomial that preserves non-zeroness. We can now check the non-zeroness of this commutative polynomial using the Polynomial Identity Lemma for commutative polynomials.

Assume $\hat{f}_3^{(c)} + F_3^{(c)} = 0$. We will now detail how to modify the coefficients of certain monomials in $\tilde{f}$, which was obtained in Step 1, before executing Steps (2) and (3). We establish that this coefficient modification maintains non-zeroness and remains non-zero even after the application of Steps (2) and (3).

4.4 Coefficient Modification by Modulo Counting Automaton

Assuming $\hat{f}_3^{(c)} + F_3^{(c)} = 0$, we know that $\hat{f}_3^{(c)} \not\equiv 0$ which implies that $F_3^{(c)} \not\equiv 0$ and $\hat{f}_3^{(c)} = -F_3^{(c)}$. To resolve this, we will carefully modify some of the monomial coefficients in the n.c. polynomial $\tilde{f} = \hat{f}_1 + F_1$ (see Lemma 4.4.1) before proceeding with product sparsification (Lemma 4.2.1) and commutative transformation (Lemma 4.3.1). We will show that these modifications ensure the resulting polynomial remains non-zero after Steps (2) and (3). Recall that $\tilde{f} = \hat{f}_1 + F_1$ is derived after replacing the commutative variables $Y \sqcup Z$ in the polynomial $f' \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ with scalar substitutions (as part of Step 1).

Let $\tilde{m}$ be a monomial in the commutative polynomial $\hat{f}_3^{(c)} \in \mathbb{F}[W]$ with a non-zero coefficient $\alpha_{\tilde{m}} \in \mathbb{F}$. This monomial also appears in $F_3^{(c)} \in F[Z]$ with coefficient $-\alpha_{\tilde{m}} \in \mathbb{F}$.

Now, consider a non-zero n.c. monomial m in $\hat{f}_1$. We use $m \in \hat{f}_1$ to denote this. The monomial m can be expressed as $m = m_1.m_2 \ldots m_{D_2}$ where $m_i = \xi_1^{i_1}.\xi_2^{i_2} \ldots \xi_s^{i_s}$ for $i \in [D_2]$.

By Claim 4.1.1, for all $i \in [D_2]$, we have $\sum_j i_j = D_1$. If we apply product sparsification and commutative transformation (Steps (2) and (3)) to this monomial m, the resulting polynomial G_m over commutative variables W may include the commutative monomial $\tilde{m}$ with a non-zero coefficient, denoted as $\exists m \rightsquigarrow \tilde{m}$.

We define the set $A^{\tilde{m}}_{\hat{f}_1} = \{m \in \hat{f}_1 \mid \exists m \rightsquigarrow \tilde{m}\}$ and the set $B^{\tilde{m}}_{F_1} = \{m' \in F_1 \mid \exists m' \rightsquigarrow \tilde{m}\}$.

It's important to note that $A^{\tilde{m}}_{\hat{f}_1} \cap B^{\tilde{m}}_{F_1} = \phi$. Let $m' \in B^{\tilde{m}}_{F_1}$, expressed as $m' = m'_1.m'_2 \dots m'_N$ with each $m'_i = \xi_1^{i_1}.\xi_2^{i_2} \dots \xi_s^{i_s}$, where N may differ from D_2. By Claim 4.1.2, there exists an $i \in [N]$ such that the exponents of the sub-monomial m'_i satisfies $\sum_j i_j \neq D_1$.

To ensure non-zeroness during the commutative transformation, we will modify the coefficients of the n.c. monomials in $A^{\tilde{m}}_{\hat{f}_1}$ and $B^{\tilde{m}}_{F_1}$ differently. Since only the monomials in $A^{\tilde{m}}_{\hat{f}_1} \sqcup B^{\tilde{m}}_{F_1}$ are transformed into the commutative monomial $\tilde{m}$, after this coefficient modification the monomial $\tilde{m}$ will have a non-zero coefficient after product sparsification and commutative transformation (Steps (2) and (3)).

It is important to note that, during this process, each n.c. variable $\xi_i, i \in [s]$ is replaced by the same n.c. variable ξ_i ensuring no additional monomials are created; the only change involves the adjustment of the monomial coefficients.

This is stated in the following lemma and the proof can be found in Appendix D.

Lemma 4.4.1. *Let $\tilde{f} = \hat{f}_1 + F_1 \in \mathbb{F}\langle\xi\rangle$ be the non-zero non-commutative polynomial obtained in Step 1. Let $\hat{f}_3^{(c)}$ be the commutative polynomial obtained from $\hat{f}_1$ after Steps (2) and (3) and let $F_3^{(c)}$ be the commutative polynomial obtained from F_1 after Steps (2) and (3). If $\hat{f}_3^{(c)} + F_3^{(c)} = 0$, then it is possible to modify the coefficients of $\tilde{f}$ obtained in Step (1) before executing Steps (2) and (3) such that the resulting non-commutative polynomial $\tilde{f'} \in \mathbb{F}\langle\xi\rangle$ can be transformed into a commutative polynomial through Steps (2) and (3) while ensuring that the resulting commutative polynomial retains its non-zeroness.*

4.5 Black-box Randomized PIT for $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ Circuits

Each of these three steps, along with the coefficient modification step, results in its own set of matrices for evaluation. In particular, the matrices obtained in each step evaluate a n.c. polynomial derived from the previous step.

Given that our model operates as a black box, we cannot evaluate the polynomial in this manner. Instead, we require a single matrix substitution for each n.c. variable. To address this, we apply Lemma 3.0.1 to combine the substitution matrices from all four steps into a single matrix for each n.c. variable.

This approach allows us to establish an efficient randomized polynomial identity testing (PIT) algorithm for depth-5 +-regular circuits, as demonstrated in the following theorem.

Theorem 3. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \dots, x_n\}$, computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then $f \not\equiv 0$ if and only if it does not evaluate to zero on the matrix algebra $\mathbb{M}_{s^6}(\mathbb{F})$.*

Proof. There are two directions in the proof. The backward direction is straightforward, as evaluating a zero polynomial f will always result in a zero matrix. Now, we will proceed to demonstrate the forward direction.

The dimensions of the substitution matrices obtained for Steps (1), (2), and (3) are s, $(4s-2)$, and s^2 respectively (see Figures 1, B.1 and C.1). Let $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ denote these substitution

matrices. The output of Step 1 is the polynomial $\mathbf{A}[1, s]$. Evaluating this output on $\mathbf{B}$, we obtain the output of Step 2, which is given by the polynomial $\mathbf{B}[1, 4s-3] + \mathbf{B}[1, 4s-2]$. Finally, evaluating the output of Step 2 on $\mathbf{C}$, gives us the output of Step 3, $\mathbf{C}[1, s^2]$.

We define the dimensions as follows: let $d_1 = s$, $d_2 = 4s-3$, $d_2' = 4s-2$, and $d_3 = s^2$.

According to Lemma 3.0.1, the substitution matrices $\mathbf{A} = (\mathbf{A_1}, \ldots, \mathbf{A_n})$, $\mathbf{B} = (\mathbf{B_1}, \ldots, \mathbf{B_s})$ and $\mathbf{C} = (\mathbf{C_1}, \ldots, \mathbf{C_s})$ can be combined into a single matrix substitution $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ of dimension $O(s^4)$. This results in an overall matrix substitution $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ of dimension $O(s^4)$. We evaluate the polynomial as $\mathbf{O} = f(\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$.

The output of the substitution automaton is defined as the sum of two entries of the matrix O.

$$f^{(c)} = \mathbf{O}\left[\mathbf{1}, (\mathbf{d_1} \cdot \mathbf{d_2} \cdot \mathbf{d_3})\right] + \mathbf{O}\left[\mathbf{1}, \left(\mathbf{d_1} \cdot \mathbf{d_2'} \cdot \mathbf{d_3}\right)\right]$$

It is important to note that, by the matrix composition lemma (see Lemma 3.0.1), the polynomial $f^{(c)}$ is equal to the polynomial obtained as the output of Step 3, which is $\mathbf{C}[1, s^2]$. This summation arises because the automaton for Step (2) has two accepting states, leading to two entries in the final composed matrix.

If $f^{(c)} \in \mathbb{F}[W]$ is identically zero, we can apply the coefficient modification step (See §4.4) to obtain a non-zero commutative polynomial. The maximum dimension of the substitution matrix in this modification step is $O(s^2)$ (see Figures 5 and 4), which will result in a final matrix of dimension $O(s^6)$ due to the matrix composition lemma (Lemma 3.0.1).

Since one of the automatons used in the coefficient modifications step (see Figure 5) has two accepting states, the output will be the *sum of four entries* of the resulting matrix. Consequently, we have shown that $f^{(c)} \in \mathbb{F}[W]$ is non-zero if and only if $f \neq 0$. The non-zeroness of $f^{(c)}$ implies that at least one of the four entries of the matrix is non-zero. The summation of entries was defined for analysis purposes.

Finally, the non-zeroness of the polynomial $f^{(c)}$ can be tested using the DeMillo-Lipton-Schwartz-Zippel lemma. Thus, we conclude that the polynomial f is non-zero if and only if it is not identically zero on the matrix algebra $\mathbb{M}_{s^6}(\mathbb{F})$.

This completes the proof of the theorem. □

Remark 4.5.1. *We observe that the commutative polynomial $f^{(c)} \in \mathbb{F}[W]$ is an s^2-ordered power-sum polynomial over W, in the sense that we can arrange the variables in each monomial of $f^{(c)}$ in increasing order according to the first index of the W variables, allowing for some exponents to be zero as specified in Definition 4.0.1.*

This is summarized in the following theorem.

Theorem 4. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a $\Sigma\Pi^*\Sigma\Pi^*\Sigma$ circuit of size s. Then, f can be transformed into an s^2-ordered power-sum polynomial while preserving its non-zeroness.*

4.5.1 An Automaton for Theorem 4

We can envision a substitution automaton $\mathcal{A}$ for Theorem 4 as follows. By applying the matrix composition Lemma 3.0.1, we can combine the substitution matrices obtained from Steps (1) through (3), along with the modifications to coefficients, into a single substitution matrix $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ of dimension $O(s^6)$. We then evaluate the polynomial as $\mathbf{O} = f(\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$.

The output of the substitution automaton is defined as the sum of several entries of the matrix $\mathbf{O}$ (refer to the polynomial $f^{(c)}$ defined in the proof of Theorem 3).

It is crucial to note that each entry of the matrices in $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ is a monomial over $Y \sqcup Z \sqcup \zeta \sqcup \chi \sqcup W$, where $Y \sqcup Z$ are commutative variables from Step (1), and $\zeta \sqcup \chi$ are commutative variables from Step (2), while W contains commutative variables from Step (3).

As noted in Steps (1) and (2), we can replace the commutative variables in $Y \sqcup Z \sqcup \zeta \sqcup \chi$ with scalars without losing the non-zeroness of the output polynomial (by the Polynomial Identity Lemma).

After these replacements, each entry of the matrices in $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ transforms into scalar multiples of variables over W. We denote the resulting matrices as $\mathbf{M'} = (\mathbf{M'_{x_1}}, \mathbf{M'_{x_2}}, \ldots, \mathbf{M'_{x_n}})$.

We can construct a substitution automaton $\mathcal{A}$ such that the substitution matrix for the variable x_i is given by the matrix $\mathbf{M'_{x_i}}$, where the entries are scalar multiples of variables in W. These entries correspond to transitions that substitute a n.c. variable with a scalar multiple of a variable in W. This automaton $\mathcal{A}$ effectively transforms f into an s^2-ordered power-sum polynomial $f^{(c)}$ while preserving its non-zeroness.

We can view the resulting s^2-ordered power-sum polynomial $f^{(c)}$ as a n.c. polynomial over W. This idea is crucial for developing black-box randomized polynomial identity testing (PIT) for circuits of larger depths using induction.

It is important to note that the monomials of $f^{(c)}$ do not correspond to a single entry of the output matrix $\mathbf{O} = f(\mathbf{M'_{x_1}}, \mathbf{M'_{x_2}}, \ldots, \mathbf{M'_{x_n}})$. Instead, they represent the sum of several entries, as indicated in the polynomial $f^{(c)}$ defined in the proof of Theorem 3. Effectively, the column numbers of these entries form the set of accepting states for the new automaton $\mathcal{A}$, with row 1 serving as the starting state of this automaton.

5 Black-Box Randomized PIT for Small Depth +-Regular Circuits

In this section, we present an efficient randomized black-box polynomial identity testing (PIT) algorithm for polynomials computed by small-depth +-regular circuits. The main result of this section is the following theorem.

Theorem 5. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$, computed by a +-regular circuit of size s and depth d. We denote the number of addition (i.e., $\sum$) layers in the circuit by d^+. Then, $f \not\equiv 0$ if and only if f is not identically zero on $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and $|\mathbb{F}|$ is sufficiently large.*

5.1 Transforming f into an ordered power-sum polynomial

To demonstrate Theorem 5, we first establish that the polynomial f can be transformed into a more structured n.c. polynomial that facilitates the testing for non-zeroness. Specifically, we show that f can be converted into a c-ordered power-sum polynomial while preserving its non-zeroness, where c is a function of the depth d and size s. We have the following theorem.

Theorem 6. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$ computed by a +-regular circuit of size s and depth d. We denote the number of addition (i.e., $\sum$) layers in the circuit by d^+.*

Then, f can be transformed into a $s^{(d^+-1)}$-ordered power-sum polynomial while preserving its non-zeroness.

Proof. The proof is by induction on the number of $\sum$ layers in the circuit. In the rest of the proof, +-regular circuits are referred to as circuits. The number of $\sum$ layers in the circuit is denoted as $d^+ = (d+1)/2$, and we refer to d^+ as +-depth of the circuit.

- **Base Cases:** The theorem holds for the base cases: $d^+ = 1$ (linear forms), and $d^+ = 2$ (as shown in [AJMR19]) and $d^+ = 3$ as established in Theorem 4.
- **Inductive Step:**
  - **Induction Hypothesis:** For the induction hypothesis, we assume that the theorem holds for any polynomial g computed by a circuit of size s and +-depth $(d^+ - 1)$. Specifically, this means that g can be converted to an $s^{(d^+-1)}$*-ordered power-sum polynomial* while preserving non-zeroness.

  Now, suppose f is computed by a +-depth d^+ circuit of size s. We can express this polynomial as:

$$f = \sum_{i \in [s]} \prod_{j \in [D_2]} Q_{ij}.$$

  Here, each Q_{ij} (for $i \in [s], j \in [D_2]$) can be computed by a +-depth $(d^+ - 1)$ circuit of size at most s. We denote the degree of Q_{ij} polynomials as D_1. Therefore, f is a sum of products of polynomials computed by +-depth $(d^+ - 1)$ circuits of size at most s. By induction hypothesis, for all $i \in [s], j \in [D_2]$, each Q_{ij} can be converted into an $s^{(d^+-2)}$*-ordered power-sum polynomial* $\hat{Q}_{ij}$ while preserving non-zeroness.

  Next, we need to establish that these individual conversions preserve the non-zeroness of the transformed polynomial. As noted in Remark 4.1.3, the proof of Lemma 4.1.1 can be generalized to affirm this case. Specifically, the n.c. polynomial $\hat{f}_1$ defined as

$$\hat{f}_1 = \sum_{i \in [s]} \prod_{j \in [D_2]} \hat{Q}_{ij},$$

  where each $\hat{Q}_{ij}$ polynomial is a $s^{(d^+-2)}$*-ordered power-sum polynomial*, maintains non-zeroness.

  Following this, the polynomial $\hat{f}_1$ can be product sparsified in a manner analogous to the depth-5 case. As highlighted in Remark 4.2.1, the proof of Lemma 4.2.1 can be extended to product sparsify the polynomial $\hat{f}_1$. Due to the structured nature of $\hat{Q}_{ij}$ polynomials, boundaries can be identified by an automaton. Thus, we can construct an automaton $\mathcal{A}_2$ similar to the one in Figure B.1. We consider the output of $\mathcal{A}_2$ on $\hat{f}_1$ and we can replace the commutative variables from this resulting polynomial with scalars, yielding a new n.c. polynomial $\hat{f}_2$ while maintaining non-zeroness.

  Finally, the polynomial $\hat{f}_2$ can be converted into a commutative polynomial while maintaining non-zeroness, similar to the depth-5 case. As noted in Remark 4.3.1, the proof of Lemma 4.3.1 can be generalized to convert $\hat{f}_2$ into a commutative polynomial $\hat{f}_3$ that preserves non-zeroness. Similar to the depth-5 case, boundaries of $\hat{Q}_{ij}$ polynomials can be identified by an automaton due to their structured nature. Thus, we can construct an automaton $\mathcal{A}_3$ similar to one in Figure C.1. We consider the output of $\mathcal{A}_3$ on $\hat{f}_2$. The resulting commutative polynomial is denoted $\hat{f}_3$. The commutative polynomial $\hat{f}_3$ is an $s^{(d^+-1)}$*-ordered power-sum polynomial*.

Thus, we have shown that a polynomial f computed by a circuit of size s and +-depth d^+ can be transformed into a $s^{(d^+-1)}$-ordered power-sum polynomial while preserving its non-zeroness.

□

5.1.1 An Automaton for Theorem 6

Let $f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{ij}$ be a polynomial computed by a depth-d circuit of size s. We denote the number of addition (i.e., $\sum$) layers in the circuit by d^+.

Suppose we have substitution matrices $\mathbf{M} = (\mathbf{M_{x_1}}, \mathbf{M_{x_2}}, \ldots, \mathbf{M_{x_n}})$ that transforms each Q_{ij} polynomials into an $s^{(d^+-2)}$-ordered power-sum polynomial while preserving its non-zeroness.. Let $\mathcal{A}$ be the corresponding substitution automaton with these substitution matrices $\mathbf{M}$.

We can then construct an automaton to perform Step (1) of our method, transforming f into

$$f_1 = \hat{f}_1 + F_1,$$

as follows. In §4.5.1, we described how to construct a substitution automaton that converts a polynomial f computed by a depth-5 circuit into an s^2-ordered power-sum polynomial $f^{(c)}$ while preserving its non-zeroness. We can treat $f^{(c)}$ as a n.c. polynomial.

Given that f is a sum of products of Q_{ij} polynomials and that the boundaries of these Q_{ij} polynomials are unknown, we can modify the substitution automaton $\mathcal{A}$ to guess the boundaries of each Q_{ij}. This involves adding transitions to the starting state of automaton $\mathcal{A}$ (similar to the automaton depicted in Figure 1). We denote the new substitution automaton by $\mathcal{A}'$

Due to the uncertainty in identifying the boundaries, the polynomial f, computed by a depth-d circuit, transforms into the n.c. polynomial

$$f_1 = \hat{f}_1 + F_1,$$

where F_1 represents the spurious part resulting from incorrect guesses made by the automaton $\mathcal{A}'$. Here,

$$\hat{f}_1 = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij},$$

where each $\hat{Q}_{ij}$ being an $s^{(d^+-2)}$-ordered power-sum polynomial. The polynomial f_1 preserves non-zeroness.

The advantage of $\hat{f}_1$ is that the boundaries of $\hat{Q}_{ij}$ can be effectively identified by the automaton. Subsequently, similar to the depth-5 case, Steps (2), (3), and coefficient modifications can be applied to the polynomial f_1. Using Lemma 3.0.1, the resulting substitution matrices at each step can be combined to form a single matrix for each input variable of the polynomial f.

When we evaluate the polynomial f using these substitution matrices, the output is an $s^{(d^+-1)}$-ordered power-sum polynomial while preserving its non-zeroness. As in the depth-5 case, the output is generally spread across several entries of the output matrix due to the automata used having multiple accepting states.

Size of the substitution automaton $\mathcal{A}'$

Let $\hat{f}_1$ be the structured part of the polynomial obtained after Step (1), applied to the polynomial f, which is computed by a depth-d circuit of size s. Recall that $\hat{f}_1$ is a sum of products of $s^{(d^+-2)}$-

ordered power-sum polynomials.

- **Automaton size for Step 2:** The size of the substitution automaton that product-sparsifies the polynomial $\hat{f}_1$ is $(4s-2)$. It is very similar to the substitution automaton shown in Figure B.1. We only need to add more transitions to this figure because $\hat{f}_1$ is a sum of products of $s^{(d^+-2)}$-ordered power-sum polynomials, where $s^{(d^+-2)}$ could be greater than s. Thus, there are more variables than in the depth-5 case. Let the resulting polynomial be $\hat{f}_2$ with no commutative variables.

- **Automaton size for Step 3:** The size of the substitution automaton that transforms $\hat{f}_2$ into a commutative polynomial, specifically into an $s^{(d^+-1)}$-ordered power-sum polynomial, is $s^{(d^+-1)}$. This automaton is quite similar to that used in the depth-5 case (see Figure 3).

- **Automaton size for coefficients modification step:** The size of the substitution automaton that filters out one spurious monomial from the polynomial F_1 (obtained by Step 1) is at most $14s$. This is due to the prime number p required for this step being bounded by $\leq 4.4s$ (see Lemma D.0.1). Additionally, we need to 3 states to process the ξ-patterns (see Definition 4.0.2). This automaton construction is similar to the automaton illustrated in Figure 4 in the depth-5 case.

  The size of the substitution automaton that modifies the coefficients of the polynomial $\hat{f}_1 + F_1$ (also obtained by Step 1) is bounded by $4.4s^{(d^+-1)} + 3$. This is analogous to Figure 5 in the depth-5 case. Each ξ-pattern (see Definition 4.0.2) of $\hat{f}_1$ contains $s^{(d^+-2)}$ variables, and the prime number p required for this step is bounded by $4.4 \log D$, where D is the degree of the f and bounded by 2^s. Therefore, with one starting state and two additional accepting states, the automaton size becomes $4.4s^{(d^+-1)} + 3$.

  Consequently, the maximum size of the automaton for this step is $s^{(d^+-1)} + 3$.

- **Automaton size for Step 1:** Let

$$M_n = \sum_{\substack{k \text{ odd} \\ k \leq n}} k - 3$$

  for $n \geq 5$. Note that M_n represents the sum of all odd numbers up to n, excluding 3. Assuming $d \geq 5$, the automaton for this step has a size of $O\left(18^{(d^+-3)} \cdot s^{M_{d-2}}\right)$. This matches the automaton size of Step 1 in the depth-5 case (where $d^+ = 3$), which is s.

Therefore, the final automaton size is the product of the automaton sizes in each step. This results in the following expression for the final automaton size:

$$\begin{aligned}
\text{Size} &= O\left(18^{(d^+-3)} \cdot s^{M_{d-2}}\right) \times \left(4.4s^{(d^+-1)} + 3\right) \times (4s-2) \times \left(s^{(d^+-1)}\right) \\
&= O\left(18^{(d^+-3)} \cdot s^{M_{d-2}}\right) \times \left(17.6s^d - 8.8s^{(2d^+-2)} + 12s^{d^+} - 6s^{(d^+-1)}\right) \text{ as } d = (2d^+ - 1) \\
&= O\left(18^{(d^+-2)} \cdot s^{M_d}\right) \\
&= O\left(18^{(d^+-2)} \cdot s^{d^2-3}\right) \\
&= s^{O(d^2)}
\end{aligned}$$

Remark 5.1.1. *Similar to the depth-5 case, we can address both small and high degree scenarios: $D_1 \geq s-1$ and $D_1 < s-1$ (see §4.1.3).*

It is important to note that while the automaton for Theorem 6 utilizes an automaton that changes the coefficients of monomials, it does not turn a zero non-commutative polynomial f into a non-zero one. Suppose f is computed by a depth d circuit of size s. We can express this polynomial as:

$$f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{ij}.$$

Now, consider the case where two monomials generated by f cancel each other out. Specifically, suppose a monomial m is generated by two different products, $\prod_{j\in[D_2]} Q_{i_1 j}$ and $\prod_{j\in[D_2]} Q_{i_2 j}$, where $i_1 \neq i_2$ and they cancel each other. Let M be the obtained substitution matrices as above. If we evaluate these monomials on M, the two different ways of generating the monomial transform in the same manner. Therefore, if they cancel each other before the transformation, they will also cancel each other after the transformation.

5.2 Randomized Identity Test for Small Depth +-Regular Circuits

We are now ready to state and prove the main theorem.

Theorem 7. *Let f be a non-commutative polynomial of degree D over $X = \{x_1, \ldots, x_n\}$, computed by a +-regular circuit of size s and depth d. We denote the number of addition (i.e., $\sum$) layers in the circuit by d^+. Then, $f \not\equiv 0$ if and only if f is not identically zero on $\mathbb{M}_N(\mathbb{F})$, where $N = s^{O(d^2)}$ and $|\mathbb{F}|$ is sufficiently large.*

Proof. We use Thoerem 6 to convert the n.c. polynomial f into an $s^{(d^+-1)}$-ordered power-sum polynomial f_{ops}, while preserving its non-zeroness. As discussed above, there is a substitution automaton of size bounded by $s^{O(d^2)}$, which results in substitution matrices of dimension $s^{O(d^2)}$. By Claim 4.0.1, f_{ops} can be treated as a commutative polynomial while preserving its non-zeroness. Using the DeMillo-Lipton-Schwartz-Zippel lemma, we can have a randomized PIT for depth d +-regular circuit of size s using matrices of dimension at most $s^{O(d^2)}$. This completes the proof of the theorem. □

6 Discussion

In this work, we presented a randomized polynomial-time algorithm for black-box polynomial identity testing (PIT) for non-commutative polynomials computed by +-regular circuits. Our method efficiently handles circuits of any constant depth. While our algorithm resolves the PIT problem for +-regular circuits of constant depth, the randomized identity testing problem for general non-commutative circuits, where the degree and sparsity can be exponentially large, remains an open question. We hope that some of the ideas developed in this work will prove useful in addressing the more general case.

Acknowledgements

We would like to extend our sincere gratitude to Prof. Arvind (IMSc and CMI) for his valuable discussions. SR also expresses his sincere gratitude to Prof. Arvind and Prof. Meena Mahajan (IMSc) for facilitating a visit to IMSc, where part of this research was conducted. We also acknowledge

the assistance of ChatGPT in rephrasing sections of this paper to improve clarity and articulation. However, we affirm that no technical ideas or proofs presented in this paper were generated by ChatGPT.

A
Proof of Lemma 4.1.1 (part of Step 1)

Proof. The proof is by induction on D_2. Assume f is non-zero.

- **Base Case: $D_2 = 1$.** For $D_2 = 1$, each Q_{i1} for $i \in [s]$ can be computed by a $\Sigma\Pi^*\Sigma$ circuit. Since the sum of $\Sigma\Pi^*\Sigma$ circuits is also a $\Sigma\Pi^*\Sigma$ circuit, it follows that a $\Sigma\Pi^*\Sigma$ circuit can compute the polynomial f. Furthermore, using the results from [AJMR19], the polynomial f can be transformed into another polynomial $\tilde{f}$ preserving non-zeroness (see Equation 4). Treating the block variables used in [AJMR19] as non-commuting maintains the non-zeroness of $\tilde{f}$. Thus, the resulting polynomial remains non-zero. This completes the proof of the base case.

- **Inductive Step:**
  - **Inductive Hypothesis:** We assume that the lemma holds for polynomials computed by a sum of products of at most (D_2-1) Q_{ij} polynomials. We will show it also holds for D_2.

  Consider the polynomial f expressed as:

$$f = \sum_{i\in[s]} Q_{i1}P_i,$$

  where $P_i = \prod_{j=2}^{D_2} Q_{ij}$. We can expand P_i into a sum of monomials, denoting the coefficient of a monomial m in P_i by $[m]P_i$. Thus, we can rewrite f as:

$$f = \sum_{i\in[s]} Q_{i1} \left(\sum_{m\in X^{D-D_1}} ([m]P_i) \times m \right),$$

  where $D = D_1 \times D_2$ is the degree of the f and $([m]P_i) \in \mathbb{F}$. Since $f \not\equiv 0$, there exists a monomial $m \in X^{D-D_1}$ such that the right derivative $f^{(m)}$ does not vanish. We define $f^{(m)}$ as:

$$f^{(m)} = \sum_{i\in[s]} Q_{i1} \times ([m]P_i),$$

  where $([m]P_i) \in \mathbb{F}$.

  Since $f^{(m)} \not\equiv 0$, it can be computed by a $\Sigma\Pi^*\Sigma$ circuit of size at most s. This is because the sum of $\Sigma\Pi^*\Sigma$ circuits is also a $\Sigma\Pi^*\Sigma$ circuit, it follows that a $\Sigma\Pi^*\Sigma$ circuit can compute the polynomial f. This reduces to the base case. Therefore, $f^{(m)}$ can be transformed into $\hat{f}^{(m)} = \sum_{i\in[s]} \hat{Q}_{i1} \times ([m]P_i)$ while preserving non-zeroness.

Handling Variables in Q_{ij}:
We assume the linear forms in the $Q_{i,j}$ polynomials are indexed from 1 to D_1. For a subset $I \subseteq [D_1]$ of size $s-1$, we define $Q_{i,j,I}$ as the polynomial obtained from $Q_{i,j}$ by treating linear forms indexed by I as non-commuting and the rest of the linear forms as commuting. Let $I = \{i_1, i_2, \ldots, i_{s-1}\}$ with $i_1 < i_2 < \ldots < i_{s-1}$.

Next, we introduce sets of variables: $Z = \{z_1, \ldots, z_n\}$ and let $Y = \{y_{ij} \mid i \in [n] \text{ and } j \in [s-1]\}$, where variables in Y and Z are commutative. The set $\xi = \{\xi_1, \xi_2, \cdots, \xi_s\}$ consists of non-commuting variables.

Replace the variable x_i variable that appears in $[D_1] \setminus I$ with a *new* commuting variable z_i.

The number of non-commuting variables in $Q_{i,j,I} \in \mathbb{F}[Z]\langle X\rangle$ is bounded by $|I| < s$, which is referred to as the n.c. degree of $Q_{i,j,I}$. Since this degree is small, $Q_{i,j,I}$ can be converted into a commutative polynomial while preserving its non-zeroness by replacing x_i variables of linear form that appears at position $i_k \in I$ by $y_{k,i}$. Let $Q^{(c)}_{i,j,I}$ denote the resulting commutative polynomial, which lies in $\mathbb{F}[Y \sqcup Z]$. To ensure all guesses of the set I are distinct, additional variables $\xi = \{\xi_1, \xi_2, \cdots, \xi_s\}$ are introduced in [AJMR19]. We keep ξ variables as *non-commutative*.

The transformed n.c. polynomial $\hat{Q}_{i1} \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ can be expressed as:

$$\hat{Q}_{i1} = \sum_{I \subseteq [D_1], |I| = s-1} Q_{i,1,I} \times \xi_I$$

where $\xi_I = \xi_1^{\ell_1}.\xi_2^{\ell_2 - \ell_1} \cdots \xi_s^{D_1 - \ell_{s-1}}$ with $I = \{\ell_1, \ell_2, \cdots, \ell_{s-1}\}$. The degree of the monomial ξ_I is D_1. By Lemma 6.2 in [AJMR19], there exists a set of indices $I \subseteq [D]$, $|I| < s$, such that $Q_{i,1,I} \neq 0$ implying $\hat{Q}_{i1} \neq 0$.

Since $\hat{f}^{(m)} \in \mathbb{F}[Y \sqcup Z]\langle\xi\rangle$ is non-zero, the polynomial

$$f^\dagger = \sum_{i \in [s]} \hat{Q}_{i1} P_i,$$

is also non-zero. Note that the derivative is only for analysis; we will not compute the right derivative of f.

Now, expand the polynomial $\hat{Q}_{i1}$ as a sum of monomials in $f^\dagger$:

$$f^\dagger = \sum_{i \in [s]} \sum_{m \in \xi^{D_1}} \Big([m]\hat{Q}_{i1}\Big)\, m \times P_i.$$

The variables ξ are the only non-commuting variables in $\hat{Q}_{i1}$, and the n.c. degree of $\hat{Q}_{i1}$ is exactly D_1. Since $f^\dagger \not\equiv 0$, there exists a monomial $m \in \xi^{D_1}$ such that the left derivative of $f^\dagger$ with respect to m does not vanish. Let $f^\dagger_{(m)}$ be this polynomial. Again, note that this derivative is for analysis only, and we will not compute it. In this case, the coefficient

$\left([m]\hat{Q}_{i1}\right)$ is a polynomial in $\mathbb{F}[Y \sqcup Z]$.

$$f^{\dagger}_{(m)} = \sum_{i\in[s]} \left([m]\hat{Q}_{i1}\right) \times P_i,$$

$\left([m]\hat{Q}_{i1}\right) \in \mathbb{F}[Y \sqcup Z]$.

Clearly, $f^{\dagger}_{(m)} \not\equiv 0$. To analyze this, we can simplify the polynomial $f^{\dagger}_{(m)}$ using the DeMillo-Lipton-Schwartz-Zippel Lemma for commutative polynomials to eliminate the commuting variables in $Y \sqcup Z$. We can replace these commuting variables with scalar substitutions from $\mathbb{F}$ or an extension field, preserving non-zeroness. Let $f^{''}$ be the resulting polynomial, which will have only variables non-commuting variables left:

$$f^{''} = \sum_{i\in[s]} \beta_i \times P_i = \sum_{i\in[s]} \beta_i \times \prod_{j=2}^{D_2} Q_{ij}, \text{ where } \beta_i \in \mathbb{F}.$$

Observe that $f^{''} \not\equiv 0$ and the number of product terms in the polynomial P_i is exactly (D_2-1).

By induction hypothesis, each Q_{ij} in $f^{''}$ can be transformed into $\hat{Q}_{ij}$ polynomial such that the resulting polynomial is non-zero. Since $f^{\dagger}_{(m)} \not\equiv 0$, the following polynomial is also non-zero:

$$\hat{f} = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij}.$$

This completes the proof of the inductive step and therefore the lemma.

□

B Proof of Lemma 4.2.1 (Step 2: Product Sparsification Lemma)

Here, we will present the proof of the *product sparsification lemma*. This proof follows the proof of Lemma 6.2 in [AJMR19], but crucially uses Claim 4.0.1.

Proof. We use induction on s.

- **Base Case:** For $s = 1$, we have $\hat{f} = \prod_{j\in[D_2]} \hat{Q}_{1j}$. By applying Claim 4.0.1, we can treat each n.c. polynomial $\hat{Q}_{1j}$ as commutative. Since the product of non-zero commutative polynomials is non-zero, we conclude that $J = \emptyset$ suffices.
- **Induction hypothesis:** Assume the lemma holds for sums of products of *s-ordered power-sum polynomials* with at most $s - 1$ terms.

  There are two directions to the proof. The backward direction is straightforward: by contrapositive, if we have a zero n.c. polynomial, then treating some of its variables as commuting does not yield a non-zero polynomial.

Now, we prove the forward direction. Let $P_i = \prod_{j\in[D_2]} \hat{Q}_{ij}$ for all $i \in [s]$. Suppose $\sum_{i\in[s]} \beta_i P_i \neq 0$. We want to show there exists a subset $J \subseteq [D_2]$ of size at most $s-1$ such that $\sum_{i\in[s]} \beta_i P_{i,J} \neq 0$, where $P_{i,J} = \left(\prod_{j\in\overline{J}} \hat{Q}_{ij}\right)\left(\prod_{j\in J} \hat{Q}_{ij}\right)$.

Let $j_0 \in [D_2]$ be the smallest index such that $dim\{\hat{Q}_{1,j_0}, \hat{Q}_{2,j_0}, \cdots, \hat{Q}_{s,j_0}\} > 1$. If no such index j_0 exists, then all P_i are scalar multiples of each other, leading to $\sum_{i\in[s]} \beta_i P_i = \alpha P_1$ for some non-zero α, which reduces to the base case.

Assume j_0 exists, we can renumber the polynomials such that $\{\hat{Q}_{1,j_0}, \hat{Q}_{2,j_0}, \cdots, \hat{Q}_{t,j_0}\}$ are the only linearly independent polynomials at j_0, where $1 < t \leq s$. Each polynomial P_i can be expressed as:

$$P_i = c_i P \cdot \hat{Q}_{i,j_0}\hat{Q}_{i,j_0+1}\cdots\hat{Q}_{i,D_2}, \quad i \in [t] \text{ and } c_i \in \mathbb{F}$$

$$P_i = c_i P \cdot \left(\sum_{\ell\in[t]} \alpha_\ell^{(i)} \hat{Q}_{\ell,j_0}\right)\hat{Q}_{i,j_0+1}\cdots\hat{Q}_{i,D_2}, \quad i \in [t+1,s] \text{ and } \alpha_\ell^{(i)} \in \mathbb{F}$$

where P is a product of $\Sigma\Pi^*\Sigma$ circuits. Let $P'_i = c_i \prod_{j=j_0+1}^{D_2} \hat{Q}_{ij}$, for $i \in [s]$.

The polynomial $\hat{f}$ can be expressed as:

$$\sum_{i=1}^{s} \beta_i P_i = P \times \left(\sum_{i=1}^{t} \beta_i Q_{i,j_0} P'_i\right) + P \times \left(\sum_{i=t+1}^{s} \beta_i Q_{i,j_0} P'_i\right).$$

Note that:

$$P \times \left(\sum_{i=t+1}^{s} \beta_i Q_{i,j_0} P'_i\right) = P \times \left(\sum_{i=t+1}^{s} \beta_i \left(\sum_{\ell\in[t]} \alpha_\ell^{(i)} \hat{Q}_{\ell,j_0}\right) P'_i\right).$$

$$\begin{aligned}\sum_{i=1}^{s} \beta_i P_i &= P \times \left(\sum_{i=1}^{t} \beta_i Q_{i,j_0} P'_i\right) + P \times \left(\sum_{i=t+1}^{s} \beta_i \left(\sum_{\ell\in[t]} \alpha_\ell^{(i)} \hat{Q}_{\ell,j_0}\right) P'_i\right) \\ &= P \times \sum_{k=1}^{t} Q_{k,j_0} P''_k\end{aligned}$$

where $P''_k = \beta_k P'_k + \beta_{t+1}\alpha_k^{(t+1)} P'_{t+1} + \beta_{t+2}\alpha_k^{(t+2)} P'_{t+2} + \cdots + \beta_s \alpha_k^{(s)} P'_s$, where $k \in [t]$.

Note that since $t > 1$, each P''_k is a sum of at most $s-1$ polynomials and each of these polynomials is a product of linear forms.

Since we treat all polynomials $Q_{k,j_0}, k \in [t]$ as n.c. and all Q_{ij} are *homgeneous* polynomials of the same degree D_1, with each Q_{k,j_0} being *s-ordered power-sum polynomials*, we can view them as linear forms as follows. This is only for analysis. Let $Y = \{y_{\overline{\ell}} \mid \ell = (\ell_1, \ldots, \ell_s) \text{ and } \sum_i \ell_i = D_1\}$ be the set of *new* non-commuting variables. Each monomial m_a with coefficient β_{m_a} in Q_{k,j_0} can be expressed as $m_a = \xi_1^{a_1} \cdot \xi_2^{a_2} \cdots \xi_s^{a_s}$ and can represented by $(\beta_{m_a} \cdot z_{\overline{a}})$. Thus, we can envision Q_{k,j_0} polynomial as a linear form over Y

variables, denoted by L_{k,j_0}. Note that there is a bijection between monomials of Q_{k,j_0} and L_{k,j_0} (which consists of only variables from Y).

Since all polynomials Q_{k,j_0} for $k \in [t]$ are linearly independent, it follows that the linear forms $L_{k,j_0}, k \in [t]$ are also linearly independent.

Let A be an invertible linear transform such that $A : L_{k,j_0} \mapsto x_k$ for $k \in [t]$. The dimension of A corresponds to the cardinality of Y, i.e., $|Y|$. Applying the map A to the j_0-th factor of polynomial $\left(P \times \sum_{k=1}^{t} L_{k,j_0} P_k^{''}\right)$, we obtain:

$$R_{j_0} = \left(P \times \sum_{k=1}^{t} x_k P_k^{''}\right)$$

Since $\sum_{i=1}^{s} \beta_i P_i = \left(P \times \sum_{k=1}^{t} Q_{k,j_0} P_k^{''}\right)$ is non-zero, by applying Proposition 3.1 in [AJMR19], we conclude that $R_{j_0} \neq 0$. Consequently, there exists $k \in [t]$ such that $P_k^{''} \neq 0$.

Given $t > 1$, $P_k^{''}$ is a sum of at most $s-1$ polynomials. By the induction hypothesis, there exists a subset $J' \subseteq [j_0+1, D_2]$ with size at most $s-2$ such that the resulting polynomial remains non-zero:

$$P_{k,J'}^{''} \neq 0$$

where

$$P_{k,J'}^{''} = \beta_k P_{k,J'}^{\prime} + \beta_{t+1}\alpha_k^{(t+1)} P_{t+1,J'}^{\prime} + \beta_{t+2}\alpha_k^{(t+2)} P_{t+2,J'}^{\prime} + \cdots + \beta_s \alpha_k^{(s)} P_{s,J'}^{\prime}.$$

Let $J = \{j_0\} \cup J'$. We now show that the polynomial

$$\sum_{i=1}^{s} \beta_i P_{i,J} = P^{(c)} \times \sum_{k=1}^{t} Q_{k,j_0} P_{k,J'}^{''}$$

remains non-zero, where $P^{(c)}$ is the commutative polynomial obtained by replacing x_i by commuting variable z_i in P. Since P is a product of *s-ordered power-sum polynomials* and by Claim 4.0.1, we can treat each of these *s-ordered power-sum polynomials* as commutative, while preserving the non-zeroness of P.

Thus, it suffices to demonstrate that $\sum_{k=1}^{t} Q_{k,j_0} P_{k,J'}^{''}$ is non-zero. By Proposition 3.1 in [AJMR19], applying the linear transform A to the first position of the polynomial $\sum_{k=1}^{t} Q_{k,j_0} P_{k,J'}^{''}$ yields $\sum_{k=1}^{t} x_k P_{k,J'}^{''}$. This sum is zero if and only if each $P_{k,J'}^{''}$ is zero. However, we established that there exists $k \in [t]$ such that $P_{k,J'}^{''} \neq 0$.

This concludes the inductive step and completes the proof.

□

B.1 Substitution Automaton for Product Sparsification (see §4.2)

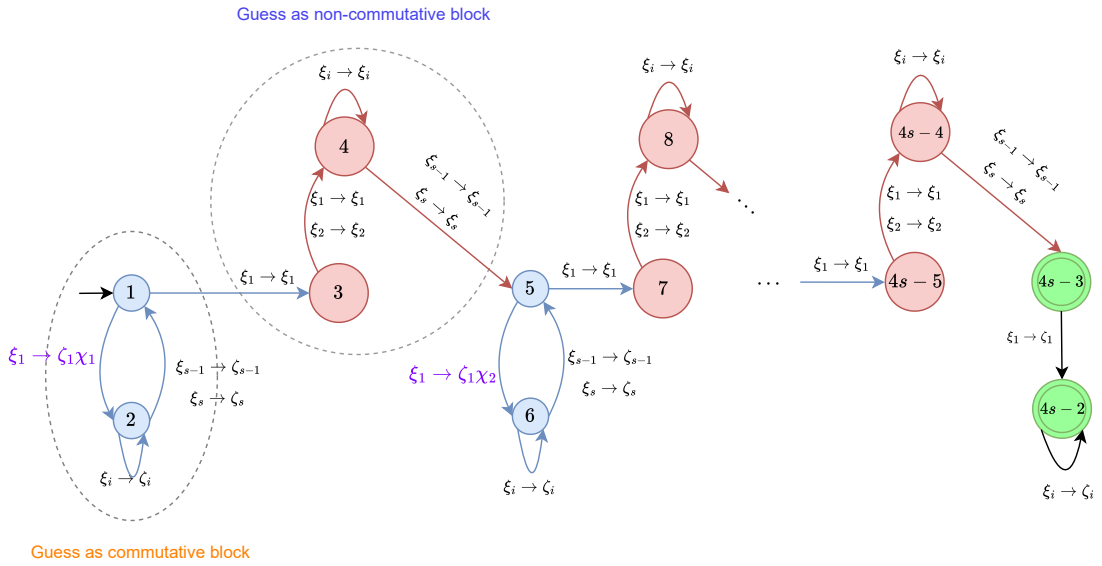

Figure 2: Substitution automaton for product sparsification

Description of the automaton in Figure 2

Recall that the polynomial $\tilde{f}$ can be expressed as:

$$\tilde{f} = \hat{f}_1 + F_1.$$

where

$$\hat{f}_1 = \sum_{i\in[s]} \prod_{j\in[D_2]} \hat{Q}_{ij}.$$

According to product sparsification lemma 4.2.1, there exists an index set $I \subseteq [D_2]$ of size $s-1$ such that for each $i \in [s]$, considering only the $\hat{Q}_{ij}$ where $j \in [I]$, as non-commutative and the remaining $\hat{Q}_{ij}$ where $j \notin I$ as commutative preserves the non-zeroness of $\hat{f}$. Since I is unknown, the automaton guesses the set I.

As previously mentioned (see Remark 4.1.4), each monomial m of the polynomial $\tilde{f}$ is a product of ξ-patterns (see Definition 4.0.2). This holds for both $\hat{f}_1$ and F_1. The automaton can identify the boundary of each ξ-pattern because each ξ-pattern ends either with the variable ξ_{s-1} or ξ_s.

For the structured polynomial $\hat{f}_1$, which is a sum of products of s-ordered power-sum polynomials, the boundary of each $\hat{Q}_{ij}$ can be identified. This allows us to explain the automaton's process of guessing the set I in terms of the $\hat{Q}_{ij}$ polynomials.

The automaton applies its transformation to both $\hat{f}_1$ and F_1. For any given monomial m in $\tilde{f}$, the automaton classifies some of the sub-monomials (i.e., ξ-patterns) as non-commutative and the rest as commutative. If the monomial m belongs to $\hat{f}_1$ it gives rise to set $\hat{Q}_{ij}$ polynomials that correspond to an index set I, that are considered non-commutative. However, if m belongs to F_1, no such index set I can be assigned, because the polynomial F_1 is unstructured. Nevertheless, for

each monomial in the unstructured polynomial F_1, exactly $s-1$ ξ-patterns are considered non-commutative and the rest as commutative. We now explain the automaton's guessing process using the structured part $\hat{f}_1$.

In Figure B.1, the states labeled $4i+1$ (for $i \geq 0$) serve as guessing states. If a $\hat{Q}_{ij}$ polynomial is guessed as non-commutative by one of these guessing states, it will be processed by one of the blocks marked as "Guess as non-commutative block". Similarly, if a $\hat{Q}_{ij}$ polynomial is guessed as commutative, it will processed by one of the blocks marked as "Guess as commutative block".

It is crucial to note that if a $\hat{Q}_{ij}$ polynomial is guessed as non-commutative, all its monomials are treated as non-commutative by replacing the non-commutative variable ξ_i in this polynomial by the same non-commutative variable ξ_i. In particular, this $\hat{Q}_{ij}$ polynomial is fully treated as non-commutative before moving on to process the next factor, $\hat{Q}_{i,j+1}$.

Similarly, if a $\hat{Q}_{ij}$ polynomial is guessed as commutative, all its monomials are treated as commutative by replacing the non-commutative variable ξ_i by the commutative variable ζ_i. In particular, this $\hat{Q}_{ij}$ polynomial is fully treated as commutative before processing the next factor $\hat{Q}_{i,j+1}$.

When a $\hat{Q}_{ij}$ polynomial is guessed as commutative, the non-commutative variable ξ_i appearing at the first position of each monomial in $\hat{Q}_{ij}$ is replaced by a commutative monomial $\zeta_i\chi_k$ where k represents the count of previously guessed non-commutative $\hat{Q}_{i,k'}$ polynomials (for $k' < k$).The commutative χ_k variables are used to distinguish different guesses made by the automaton. If ξ_i appears in positions other than the first, it will be replaced by the commutative variable ζ_i.

Given the structured nature of all $\hat{Q}_{ij}$ polynomials, the automaton can easily identify the boundaries of each $\hat{Q}_{ij}$, facilitating the guessing of the index set I.

C Proof of Lemma 4.3.1 (Step 3: Commutative Transformation)

Proof. The degree of the n.c. polynomial g is $s \times D$, where D can be exponential in s. However, g is in a more structured form as it is a sum of products of s s-ordered power-sum polynomials.

To define the position of substrings within the monomial m, we consider m as a string. Recall that m can be thought of as a string since m is n.c..

For the commutative transformation, we introduce s^3 fresh commutative variables $W = \{w_{ij} \mid i \in [s^2] \text{ and } j \in [s]\}$. Each monomial $m = m_1.m_2 \dots m_s$ can be transformed as follows: for all $i \in [s]$, if $m_i = \xi_1^{i_1}\xi_2^{i_2}\dots\xi_s^{i_s}$, we convert it to $m_i' = w_{(i-1)s+1,1}^{i_1} w_{(i-1)s+2,2}^{i_2} \dots w_{(i-1)s+s,s}^{i_s}$. Suppose the degree of ξ_s in m_i is 0 (i.e., $i_s = 0$), then we skip the variable $w_{(i-1)s+s,s}$ and continue from $w_{is+1,1}$ for the next monomial m_{i+1}. This adjustment simplifies our automaton.

This conversion is implemented by the substitution automaton given in Figure 3 and can be achieved using substitution matrices of dimension s^2. Importantly, this process does not introduce new cancellations, as there is a bijection between the monomials in g and $g^{(c)}$.

Consider the mapping of monomials of g into the monomials of $g^{(c)}$ by the substitution automaton in Figure 3. Note that every monomial of g gets mapped into exactly one monomial in $g^{(c)}$. We establish that this mapping is a bijection as follows:

- **One-to-One:**
  Let m and m' be distinct monomials of g. Let $m^{(c)}$ and m'^{c} be the corresponding transformed commutative monomial (by the automaton). We can express these monomials as follows: $m = m_1.m_2 \dots m_s$ and $m' = m_1'.m_2' \dots m_s'$.

  We first convert the monomials m and m' as follows: for each $i \in [s]$, if m_i does not include the variable ξ_s (i.e., $i_s = 0$), we append ξ_s^0 to m_i, resulting in m_i becoming $m_i \cdot \xi_s^0$. While

$\xi_s^0 = \epsilon$ (empty string), we retain it as a placeholder for clarity in our proof. We apply the same conversion to m'. We can refer to the modified monomials as m and m' again.

After this conversion, each monomial m can be expressed as $m = m_1.m_2 \dots m_s$ where $m_i = \xi_1^{i_1}\xi_2^{i_2}\cdots\xi_s^{i_s}$ with $i_s \geq 0$ and $i_j > 0$ for $j \in [s-1]$. After these conversions, we still have $m \neq m'$.

Next, we consider any monomial m of g as a string over ξ and break this string into segments by identifying maximal substrings containing the same variable ξ_i for $i \in [s]$. We encode the position of each segment within m using the first index of the variables in W. It is important to note that in any monomial m of g, there are at most s^2 such segments. In particular, following the conversion that adds ξ_s^0 when it is absent, there are exactly s^2 segments present.

If $m \neq m'$, then there exists a segment $k \in [s^2]$ where the exponents of the variable in that segment differ between m and m'. Assuming the variable in the k-th segment is ξ_j, we conclude that the exponents of $w_{k,j}$ in $m^{(c)}$ and m'^c are not the same. For any variable $w \in W$, we replace w^0 by 1 in the transformed commutative monomial. This shows that $m \neq m'$ implies $m^{(c)} \neq m'^c$.

- **Onto :**
  Let $m^{(c)}$ be a monomial in $g^{(c)} \in \mathbb{F}[W]$. We first order the variables in $m^{(c)}$ according to the first index of the W variables appearing in $m^{(c)}$. Next, we break this into segments by identifying maximal substrings that contain W variables, ensuring that the second subscript of these variables is non-decreasing, either from 1 to $s-1$ or from 1 to s. Each segment can be represented in one of two forms:
  (1) $w_{i,1}^{\ell_1} \cdot w_{i+1,2}^{\ell_2} \cdots w_{i+s-2,s-1}^{\ell_{s-1}}$ or
  (2) $w_{i,1}^{\ell_1} \cdot w_{i+1,2}^{\ell_2} \cdots w_{i+s-1,s}^{\ell_s}$.
  We then convert each of these segments into segments over the ξ variables. For instance, if we take the segment $w_{i,1}^{\ell_1} \cdot w_{i+1,2}^{\ell_2} \cdots w_{i+s-2,s-1}^{\ell_{s-1}}$, we can convert it to $\xi_1^{\ell_1} \cdot \xi_2^{\ell_2} \cdots \xi_{s-1}^{\ell_{s-1}}$.
  This converts $m^{(c)}$ into a monomial m over ξ variables. It is clear that if we consider the output of the automaton in Figure 3 on the monomial m, then it is the monomial $m^{(c)}$.

Thus, the n.c. polynomial g can be transformed into a commutative polynomial $g^{(c)}$ while preserving its non-zeroness. □

C.1 Substitution Automaton for Commutative Transformation

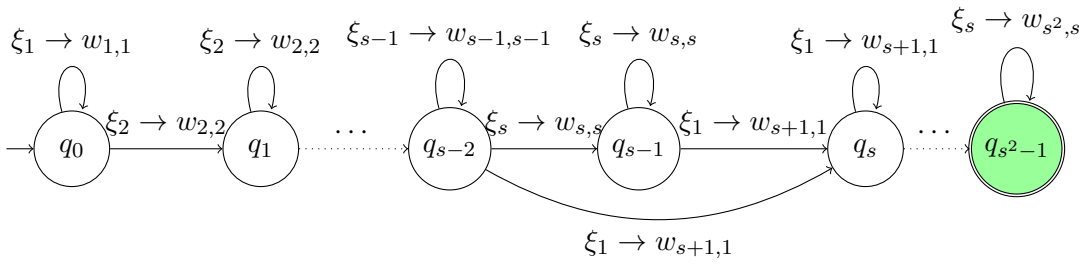

Figure 3: Automaton that converts n.c. variables to commutative variables

As an example, we provide the substitution matrix M_{ξ_1} obtained from Figure C.1 for the variable ξ_1.

$$\mathbf{M_{\xi_1}} = \begin{array}{c} \\ q_0 \\ q_1 \\ \vdots \\ q_{s-2} \\ q_{s-1} \\ q_s \\ \vdots \\ q_{s^2-1} \end{array} \begin{array}{c} \begin{array}{cccccccc} q_0 & q_1 & \cdots & q_{s-2} & q_{s-1} & q_s & \cdots & q_{s^2-1} \end{array} \\ \begin{pmatrix} w_{1,1} & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & w_{s+1,1} & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & w_{s+1,1} & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & w_{s+1,1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{pmatrix} \end{array}$$

D Proof of Lemma 4.4.1 (Coefficient Modification by Automaton)

We recall the following proposition from [DESW11] (see Proposition 1).

Proposition D.0.1. *Let $n \in \mathbb{N}$. Suppose we have two strings, x and y such that $|x| \neq |y|$ and $|x|, |y| \leq n$. There exists a DFA with a number of states bounded by $O(\log n)$ that can distinguish between the two strings. Specifically, when processing the strings x and y from the initial state, the DFA will reach different states for each string. This DFA essentially computes the string length modulo p.*

This above proposition relies on a lemma from [SB96] (also see Lemma 1 from [DESW11]).

Lemma D.0.1. *Let $n \in \mathbb{N}$. If $k \neq m$, and both k and m are less than or equal to n, then there is a prime $p \leq 4.4 \log n$ such that $k \not\equiv m \pmod p$.*

Proof of Lemma 4.4.1:

Proof. Recall that $\tilde{f} = \hat{f} + F \in \mathbb{F}\langle\xi\rangle$ is the polynomial obtained after Step (1). It is important to note that any monomial in $\tilde{f}$, including all monomials in $A_{\hat{f}}^{\tilde{m}}$ and $B_F^{\tilde{m}}$, is formed by concatenating ξ-patterns (see Defintion 4.0.2).

Consider a monomial $m_j = m_{j,1}m_{j,2}\cdots m_{j,N_j} \in B_F^{\tilde{m}}$, where $m_{j,1}, m_{j,2}, \dots, m_{j,N_j}$ are ξ-patterns. While the degree of m_j matches the overall degree $D = D_1 \times D_2$ of the polynomial, we cannot assert whether N_j is equal to D_2 or not (See Case 2 in 4.1.1). However, we can affirm that there is at least one ξ-pattern in m_j for which the sum of the exponents does not equal D_1.

For $B_F^{\tilde{m}}$, there are two possibilities:

In each of the two cases, we construct an automaton $\mathcal{M}$ to evaluate the polynomial $\tilde{f}$ using substitution matrices obtained from the automaton. Let M_{ξ_i} denote the substitution matrix corresponding to the variable ξ_i obtained from the automaton. We then evaluate the polynomial $\tilde{f}$ on these matrices, leading to the resulting matrix: $\tilde{f}(M_{\xi_1}, M_{\xi_2}, \dots, M_{\xi_s})$. It is important to note that this can be expressed as:

$$\tilde{f}(M_{\xi_1}, M_{\xi_2}, \dots, M_{\xi_s}) = \hat{f}(M_{\xi_1}, M_{\xi_2}, \dots, M_{\xi_s}) + F(M_{\xi_1}, M_{\xi_2}, \dots, M_{\xi_s}).$$

- **Case 1:** There exists a monomial $m_l \in B_F^{\tilde{m}}$ such that $N_l \neq D_2$. Let $\lambda \equiv D_2 \mod p$.

  Let's fix such a monomial $m_l \in B_F^{\tilde{m}}$ represented as $m_\ell = m_{\ell,1} m_{\ell,2} \cdots m_{\ell,N_\ell}$. As previously noted, an automaton can identify the boundaries between any two sub-monomials $m_{\ell,i}$ and $m_{\ell,i+1}$ (for $i < N_\ell$) in the monomial m_ℓ. Therefore, for any natural number p we can compute $N_\ell \mod p$ using an automaton.

  Recall that for each monomial in $A_{\hat{f}}^{\tilde{m}}$, the number of sub-monomials (i.e., the number of ξ-*patterns*) is exactly D_2. We can compute the number of ξ-*patterns* modulo p as boundaries between any two sub-monomials $m_{\ell,i}$ and $m_{\ell,i+1}$ can be identified by the automaton.

  Since $N_j, D_2 \leq D$, by Proposition D.0.1, there exists an automaton $\mathcal{M}$ with a number of states bounded by $O(\log D)$. If we consider the output state of $\mathcal{M}$ when processing m_l, it will differ from the output state of all monomials in $A_{\hat{f}}^{\tilde{m}}$. The automaton $\mathcal{M}$ does not change the monomial; it simply maps the n.c. variable ξ_i to itself while computing the number of ξ-*patterns* modulo p. Notably, all monomials in $A_{\hat{f}}^{\tilde{m}}$ reach the same state in the automaton, which we denote as λ (with $\lambda = O(\log D)$).

  By Proposition D.0.1, at least one monomial $m_l \in B_F^{\tilde{m}}$ will not reach the state λ. If we consider the output of the automaton $\mathcal{M}$ in Figure 4 on $\tilde{f}$ as (q_0, q_λ)-th entry of the resulting matrix, then the n.c. polynomial at this entry is given by

  $$f_\lambda = \hat{f} + \tilde{F}$$

  where the coefficient of the monomial m_l in the polynomial $\tilde{F}$ is 0. Importantly, since $m_l \in B_F^{\tilde{m}}$ the coefficient of m_l in F is non-zero.

  It is crucial to note that $f_\lambda \neq 0$ because $\hat{f} \neq 0$ (by Lemma 4.1.1), and the non-zero monomials of $\tilde{F}$ are a subset of the non-zero monomials of F, with no common non-zero monomials between $\hat{f}$ and F (see Claims 4.1.1 and 4.1.2).

  Additionally, while $\tilde{f} \neq f_\lambda$, all monomials of $A_{\hat{f}}^{\tilde{m}}$ are included in f_λ with the same coefficients as in $\tilde{f}$. However, at least one monomial $m_l \in B_F^{\tilde{m}}$ is absent from $\tilde{F}$, thus from f_λ. Since only monomials in $A_{\hat{f}}^{\tilde{m}} \sqcup B_F^{\tilde{m}}$ are transformed into the commutative monomial $\tilde{m}$ during product sparsification and commutative transformation (Steps 2 and 3), performing these steps on f_λ will yield a non-zero commutative polynomial, specifically ensuring that the coefficient of $\tilde{m}$ is non-zero in the transformed commutative polynomial.

- **Case 2:** For each monomial $m_\ell \in B_m^F$, $N_\ell = D_2$. Let $\lambda \equiv D_1 \mod p$.

  Let's fix a monomial $m_\ell \in B_m^F$ expressed as $m_\ell = m_{\ell,1} m_{\ell,2} \cdots m_{\ell,D_2}$. For any monomial m_ℓ and $r \in [D_2]$, we define

  $$\mathcal{K}_{m_{\ell,r}} = \sum_{k=1}^{s} \ell_k.$$

  We know that there exists an ξ-*pattern* $m_{\ell,r}$ in the monomial m_ℓ, such that $\mathcal{K}_{m_{\ell,r}} \neq D_1$ by Claim 4.1.2. For all $m_t \in A_{\hat{f}}^{\tilde{m}}$, it follows that $\mathcal{K}_{m_{t,r}} = D_1$ for each $r \in [D_2]$ (as $N_t = D_2$) by Claim 4.1.1.

As previously noted, an automaton can identify the boundaries between any two sub-monomials $m_{\ell,i}$ and $m_{\ell,i+1}$ (for $i < N_\ell$) in the monomial m_ℓ. Since we do not know which sub-monomial $m_{\ell,r}$ has the property $\mathcal{K}_{m_{\ell,r}} \neq D_1$, the automaton guesses $r \in [D_2]$ and computes $\left(\mathcal{K}_{m_{\ell,r}} \mod p\right)$ for a given natural number p.

Given that $\mathcal{K}_{m_{\ell,r}}, D_1 \leq D$, by Proposition D.0.1, there exists an automaton $\mathcal{M}$ that computes $\left(\mathcal{K}_{m_{\ell,r}} \mod p\right)$. The automaton we construct has the number of states bounded by $O(s^2)$, that is $O(\log^2 D)$ as D is exponential in s for us. The output state of $\mathcal{M}$ when processing m_l will differ from the output state of all monomials in $A_{\hat{f}}^{\tilde{m}}$.

As in Case 1, the automaton $\mathcal{M}$ does not modify the monomial; it simply computes the sum of the exponents of the r-th sub-monomial $m_{\ell,r}$ of the monomial m_ℓ modulo p. The substitution automaton $\mathcal{M}$ given in Figure 5 does exactly this, with the initial state q_0 guessing the sub-monomial number $r \in [D_2]$ and the remainder of the automaton calculating $\left(\mathcal{K}_{m_{\ell,r}} \mod p\right)$.

For any monomial $m_t \in A_{\hat{f}}^{\tilde{m}}$, for every guess $r \in [D_2]$, we have $\lambda \equiv \mathcal{K}_{m_{t,r}} \mod p$, where $\mathcal{K}_{m_{t,r}} = D_1$. Thus, when considering the output of the automaton $\mathcal{M}$ on the monomial m_ℓ as the sum of two entries

$$M[q_0, q_{f_1}] + M[q_0, q_{f_2}],$$

the coefficient α_{m_t} of the monomial m_t is scaled by a factor of D_2.

Conversely, for the above fixed monomial $m_\ell \in B_F^{\tilde{m}}$, there exists a guess $r \in [D_2]$ such that $\mathcal{K}_{m_{\ell,r}} \neq D_1$ (by Claim 4.1.2). Therefore, this computation path will not reach either of the final states q_{f_1} or q_{f_2}. Thus, the coefficient α_{m_ℓ} of the monomial m_ℓ is scaled by at most a factor of $(D_2 - 1)$ in $M[q_0, q_{f_1}] + M[q_0, q_{f_2}]$.

Then the resulting output n.c. polynomial is given by

$$f_\lambda = D_2 \cdot \hat{f} + \tilde{F},$$

where the coefficient of the monomial m_l in the polynomial $\tilde{F}$ is at most $\alpha_{m_\ell} \cdot (D_2 - 1)$.

It is important to note that $f_\lambda \neq 0$. While $\tilde{f} \neq f_\lambda$, all monomials of $A_{\hat{f}}^{\tilde{m}}$ are present in f_λ with their coefficients scaled by exactly D_2. However, at least one monomial $m_l \in B_F^{\tilde{m}}$ has its coefficient in $\tilde{f}$ scaled by at most $(D_2 - 1)$.

Since only monomials in $A_{\hat{f}}^{\tilde{m}} \sqcup B_F^{\tilde{m}}$ are transformed into the commutative monomial $\tilde{m}$ after product sparsification and commutative transformation (Steps 2 and 3), performing these steps on f_λ, will yield a non-zero commutative polynomial. In particular, the coefficient of $\tilde{m}$ is non-zero.

As only monomials in $A_{\hat{f}}^{\tilde{m}} \sqcup B_F^{\tilde{m}}$ are transformed into the commutative monomial $\tilde{m}$ after product sparsification and commutative transformation (Steps 2 and 3), if we carry out product sparsification and commutative transformation on f_λ, the resulting commutative polynomial is non-zero. In particular, the coefficient of $\tilde{m}$ is non-zero in the transformed commutative polynomial.

This completes the proof of the lemma. □

D.1 Substitution Automaton for Case 1

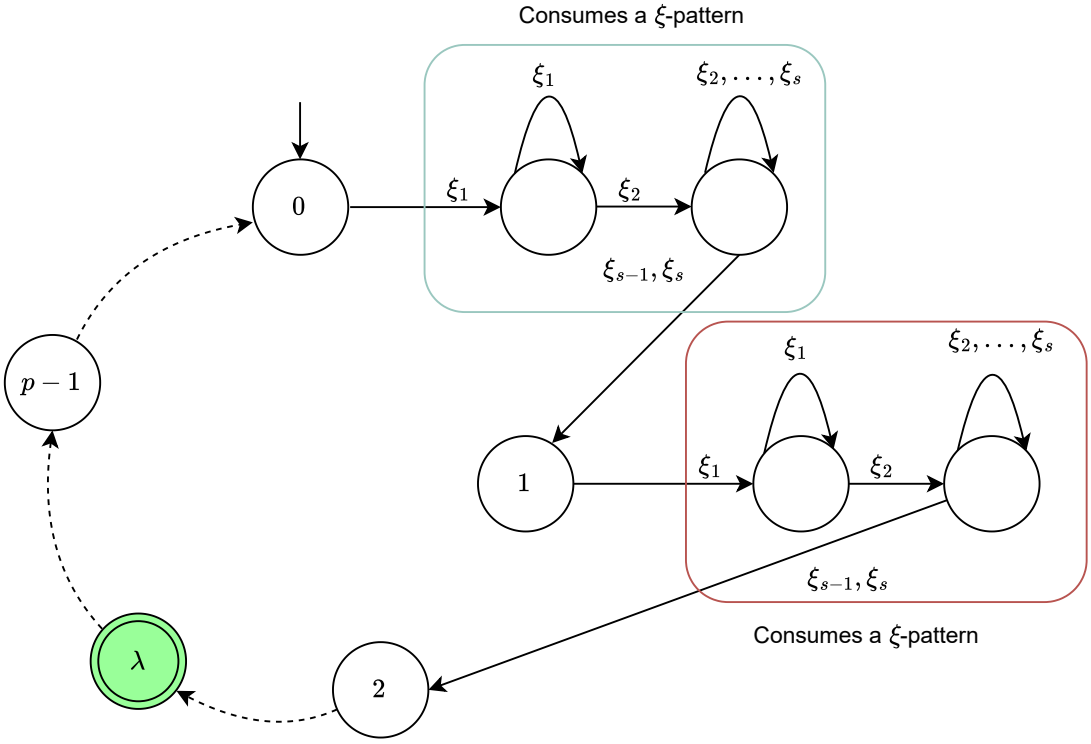

Figure 4: Substitution automaton that computes number of ξ-patterns modulo p.

Description of the automaton in Figure 4

Recall that the polynomial $\tilde{f}$ can be expressed as:

$$\tilde{f} = \hat{f}_1 + F_1.$$

Given that each monomial of $\tilde{f}$ is a product of ξ-patterns (see Definition 4.0.2), we can explain the automaton's workings in terms of these ξ-patterns. The automaton in Figure 4 computes the number of ξ-patterns modulo p in a given monomial m. Each ξ-pattern is fully processed by a section of the automaton marked as *Consumes a ξ-pattern* before the next one is considered. Through this mechanism, the automaton keeps track of the number of ξ-patterns encountered, maintaining a count modulo p.

D.2 Substitution Automaton for Case 2

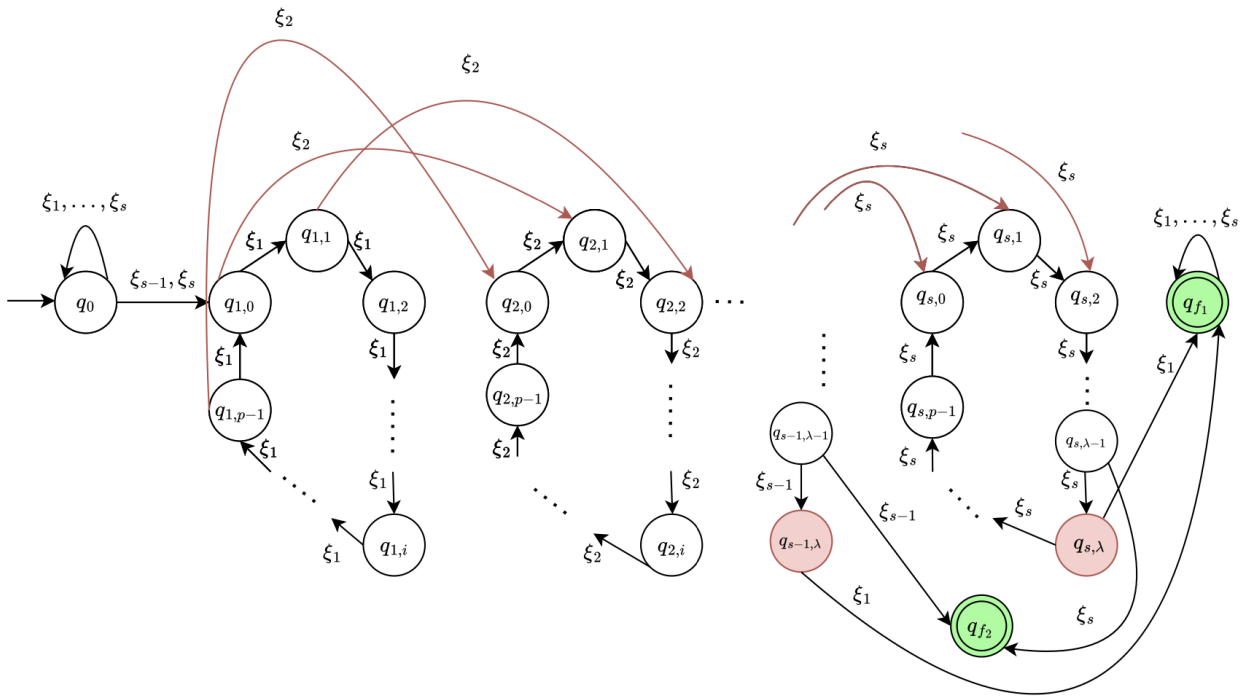

Figure 5: Modulo counting automaton that changes coefficients. Here, $\lambda \equiv D_1 \mod p$

Description of the automaton in Figure 5

Recall that the polynomial $\tilde{f}$ can be expressed as:

$$\tilde{f} = \hat{f}_1 + F_1.$$

Given that each monomial of $\tilde{f}$ is a product of ξ-patterns (see Defintion 4.0.2), it is logical to explain the automaton's guesses in terms of these ξ-patterns.

The substitution automaton in Figure 5 guesses a ξ-pattern to compute the sum of the exponents of this guessed ξ-pattern modulo a prime number p. The initial state q_0 is the only guessing state in the automaton, where the automaton selects a ξ-pattern for calculating the sum of exponents mod p.

Suppose the sum of the exponents for the guessed ξ-pattern is D_1. This guess will lead to either state q_{f_1} or state q_{f_2} based on the following two conditions:

- It reaches the final state q_{f_2} if the guessed ξ-pattern is the last ξ-pattern appearing in the given monomial m (the last variable could be either ξ_{s-1} or ξ_s, influencing the transition from either state $q_{s-1,\lambda-1}$ or $q_{s,\lambda}$) *or*
- It reaches the final state q_{f_1} if the guessed ξ-pattern is not the last one, indicating that there exists at least one other ξ-pattern following it.

We use two different final states, q_{f_1} and q_{f_2}, to ensure that each ξ-pattern is fully processed before considering the next one.

Similarly, suppose the sum of the exponents for the guessed ξ-pattern is denoted by $\mathcal{K}$. In that case, the automaton will transition to the states based on the value $\mathcal{K} \mod p$ and the variable appearing in the last position (either ξ_{s-1} or ξ_s).

The correct guess happens on the variable occurring at the last position of the ξ-pattern, which can be either ξ_{s-1} or ξ_s. In this case, the automaton will transition to the state $q_{1,0}$. Since the first variable of any ξ-pattern is ξ_1, and upon making a correct guess, the automaton continues to process the guessed ξ-pattern further.

However, if the guess is incorrect— meaning the automaton transitions to the state $q_{1,0}$ while still processing the current ξ-pattern— it indicates that additional variables (ξ_{s-1} or ξ_s) still needs to be processed. Once the automaton reaches the state $q_{1,0}$, there will be no transition for the variables ξ_{s-1} or ξ_s, resulting in the discarding of this particular guess.

E
Proof of Lemma 3.0.1 (Composing K Substitution Matrices)

Proof. The proof is by induction on K.

- **Base Case:** K = 1
  We have $f_1 = f_0(A_{11}, \ldots, A_{1n_1}) = f(A_{11}, \ldots, A_{1n})$. The lemma holds for $C_i = A_{1i}, i \in [n]$.

- **Base Case:** K = 2
  Let $f = \sum_{m \in X^D} \alpha_m . m$. Each matrix A_{ij} can be expressed as

$$A_{ij} = \sum_{k=1}^{n_{i+1}} A_{ij}^{(k)} z_{i+1,k},$$

  where $A_{ij}^{(k)} \in \mathbb{F}^{d_i \times d_i}$ for all $k \in [n_i + 1]$.

  Consider a non-zero monomial m defined as $m = x_{\ell_1} x_{\ell_2} \ldots x_{\ell_D}$, where $\ell_i \in [n]$ for all $i \in [D]$. If we replace each variable x_{ℓ_i} by $A_{1\ell_i}$ in m, let $f_{1,m}$ denote the $(1, d_1)$ entry of the matrix product $\prod_{j=1}^{D} A_{1\ell_j}$. We note that $f_{1,m} \in \mathbb{F}\langle Z_2 \rangle$.

  Next, we substitute z_{2i} with A_{2i} in $f_{1,m}$, yielding $f_{2,m}$ as the $(1, d_2)$ entry of the matrix $f_{1,m}(A_{21}, A_{22}, \ldots, A_{2n_2})$.

  Now, consider $a = (a_1, \ldots, a_D) \in [n_2]^D$ and define the n.c. monomial $m_{a,Z_2} = \prod_{i=1}^{D} z_{2,a_i}$. We can express:

$$\begin{aligned} \prod_{i=1}^{D} A_{1\ell_i} &= \prod_{i=1}^{D} \left(\sum_{k=1}^{n_2} A_{1\ell_i}^{(k)} z_{2k} \right) \\ &= \sum_{(a_1,\ldots,a_D) \in [n_2]^D} \left(\prod_{i=1}^{D} A_{1\ell_i}^{(a_i)} \right) \times m_{a,Z_2} \end{aligned}$$

This factorization holds because m_{a,Z_2} is generated only by the terms $\prod_{i=1}^{D}\left(A_{1\ell_i}^{(a_i)} z_{2a_i}\right)$, where $A_{1\ell_i}^{(a_i)} \in \mathbb{F}^{d_1 \times d_1}$ for all $i \in [D]$.

Let α_{1,d_1} be the $(1, d_1)$-th entry of the matrix $\prod_{i=1}^{D} A_{1\ell_i}^{(a_i)}$. This $\alpha_{1,d_1} \in \mathbb{F}$ represents the coefficient of the monomial m_{a,Z_2} in the $(1, d_1)$ entry of $\prod_{i=1}^{D} A_{1\ell_i}$.

Recall that $A_{1j} = \sum_{k=1}^{n_2} A_{1j}^{(k)} z_{2k}$ for all $j \in [n_1]$. For each $i \in [n]$, define:

$$C_i = \sum_{k=1}^{n_2} A_{1i}^{(k)} \otimes A_{2k}.$$

Thus, C_i is a matrix of dimension $(d_1 \cdot d_2)$.

We aim to show that f_2 corresponds to the $(1, d_1 \times d_2)$-th entry of the matrix obtained by substituting x_{ℓ_i} with C_{ℓ_i} for all $i \in [D]$. Substituting x_{ℓ_i} by C_{ℓ_i} for all $i \in [D]$ in m, we have

$$C_{\ell_1} C_{\ell_2} \ldots C_{\ell_D} = \prod_{i=1}^{D} \left(\sum_{k=1}^{n_2} A_{1\ell_i}^{(k)} \otimes A_{2k} \right).$$

Expanding this, we have:

$$= \sum_{(a_1,\ldots,a_D) \in [n_2]^D} \left(\prod_{i=1}^{D} A_{1\ell_i}^{(a_i)} \otimes A_{2a_i} \right).$$

By the mixed product property of $\otimes$ and matrix multiplication (See Lemma 2.2.1), we have:

$$= \sum_{(a_1,\ldots,a_D) \in [n_2]^D} \left(\left(\prod_{i=1}^{D} A_{1\ell_i}^{(a_i)} \right) \otimes \left(\prod_{i=1}^{D} A_{2a_i} \right) \right).$$

Let $M_{a,Z_2} = \prod_{i=1}^{D} A_{2a_i}$ and β_{1,d_2} be the $(1, d_2)$-th entry of the matrix M_{a,Z_2}. This $\beta_{1,d_2} \in \mathbb{F}$ corresponds to evaluating the monomial m_{a,Z_1} using matrices $A_2 = (A_{21}, A_{22}, \ldots, A_{2n_2})$ by replacing z_{2j} with A_{2j}.

The $(1, d_1.d_2)$-th entry of the matrix $\left(\prod_{i=1}^{D} A_{1\ell_i}^{(a_i)}\right) \otimes \left(\prod_{i=1}^{D} A_{2a_i}\right)$ is equal to $(1, d_2)$-th entry of M_{a,Z_2} scaled by α_{1,d_1}. Summing over all $a \in [n_2]^D$, the $(1, d_1 \cdot d_2)$-th entry of this matrix $\prod_{i=1}^{D} C_{\ell_i}$ equals $f_{2,m}$.

By linearity, this extends to $f(C_1, C_2, \ldots, C_n)$, showing that the $(1, d_1 \cdot d_2)$-th entry of the matrix $f(C_1, C_2, \ldots, C_n)$ is equal to the polynomial f_2.

This completes the base case, where K = 2.

- **Inductive Step:**
  Induction Hypothesis: Assume that for K − 1, we can express f evaluated at matrices $A_i = (A_{i1}, \ldots, A_{in_i})$ as

$$f_i = f_{i-1}(A_{i1}, A_{i2}, \ldots, A_{in_i})$$

for all $i \geq 1$. Let $f_{\mathrm{K}-1}$ be the resulting polynomial over the n.c. variables $Z_{\mathrm{K}} = \{z_{\mathrm{K}1}, \ldots, z_{\mathrm{K}n_{\mathrm{K}}}\}$.

For the induction hypothesis, we assume that there is a matrix substitution $B = (B_1, B_2, \ldots, B_n)$ such that each B_i has dimensions $\prod_{i\in[\mathrm{K}-1]} d_i$ and the polynomial $f_{\mathrm{K}-1}$ is equal to the $(1, \prod_{i\in[\mathrm{K}-1]} d_i)$-th entry of the matrix $f(B_1, B_2, \ldots, B_n)$. Each matrix B_i can be written as:

$$B_i = \sum_{j=1}^{n_{\mathrm{K}}} B_i^{(j)} z_{\mathrm{K},j}.$$

By the inductive hypothesis, we find that $f_{\mathrm{K}-1}$ is equivalent to the $(1, \prod_{i\in[\mathrm{K}-1]} d_i)$-th entry of the matrix $f(B_1, B_2, \ldots, B_n)$. Define:

$$f_{\mathrm{K}} = f_{\mathrm{K}-1}(A_{\mathrm{K}1}, \ldots, A_{\mathrm{K}n_{\mathrm{K}}})[1, d_{\mathrm{K}}].$$

This reduces to the base case.

Consequently, there exists a matrix substitution $C = (C_1, C_2, \ldots, C_n)$, where each C_i has dimensions $\prod_{i\in[\mathrm{K}]} d_i$ and is given by:

$$C_i = \sum_{j=1}^{n_{\mathrm{K}}} B_i^{(j)} \otimes A_{\mathrm{K},j}.$$

Thus, the polynomial f_{K} matches the $(1, \prod_{i\in[\mathrm{K}]} d_i)$-th entry of the matrix $f(C_1, C_2, \ldots, C_n)$. This concludes the proof of the lemma.

□

F Missing Proofs from §4.1

F.1 Proof of Proposition 4.1.1

Proof. The path ρ starts at q_0 and ends at the state q_{s-1}, labeled by the monomial m. When this path ρ returns to q_0 for the k-th time, it has converted some initial part of the monomial m into $\alpha_k m'_1 \cdot m'_2 \cdots m'_k$, where $\alpha_k \in \mathbb{F}[Y \sqcup Z]$. The number of sub-monomials, denoted by N, depends on the number of times the path ρ returns to the initial state q_0 before eventually reaching either q_{s-1} or q_0. We can see from the automaton given in Figure 1 that each m'_ℓ is of the form $m'_\ell = \xi_1^{\ell_1}.\xi_2^{\ell_2} \cdots \xi_s^{\ell_s}$, where $\ell_k > 0, k \in [s-1]$ and $\ell_s \geq 0$. The commutative part is grouped into the coefficient α_k. Note that $\ell_k > 0, k \in [s-1]$. This is because to return to q_0, the path ρ has to use transitions involving $\xi_1, \ldots, \xi_{s-1}$ variables. But the exponent $\ell_s \geq 0$, because to return to q_0 the path ρ may use the transition from q_{s-2} to q_0 instead of going to the state q_{s-1}, and ξ_s variable only appears at the transition from q_{s-1} to itself. This completes the proof. □

F.2 Proof of Claim 4.1.1

Proof. The path ρ respects the boundary between m_j and m_{j+1} in m for all $j < D_2$. The proof follows by noting that each transition of the substitution automaton $\mathcal{A}$ has exactly one ξ variable

in it and each transition of the automaton consists of reading a variable appearing at some position in m and substituting it according to transition rules of $\mathcal{A}$. □

F.3 Proof of Claim 4.1.2

Proof. We will consider the following two possibilities for ρ:

- **Case 1:** Suppose the path ρ visits the state q_0 in the middle of processing some sub-monomial m_j of m (that is when the automaton visits the state q_0 when it reads some variable appearing in a position other than the first position). At this point, the length of the prefix of m that has been processed is not a multiple of D_1. Additionally, at this stage, the sum of exponents of ξ variables in at least one of the generated m'_ℓ up to this point is not equal to D_1.

- **Case 2:** Suppose ρ is in a state $q_r, r \neq 0$, while replacing the variable appearing at the 1st position of the sub-monomial m_j. That is ρ starts replacing the sub-monomial m_j from a state q_r that is not q_0. If ρ reaches q_0 again in the middle of m_j, then this scenario is already handled by Case 1. Otherwise, it fully replaces m_j before returning to the state q_0. Note that since ρ started processing the sub-monomial m_j from a state $q_r \neq q_0$ and it fully replaces at least m_j before returning q_0, we can make the following observation. At this point, the sum of exponents of ξ variables in the sub-monomial m'_ℓ, which is generated by ρ after fully processing m_j and returning to q_0, is strictly greater than D_1. This is because when ρ is in state q_r, the path ρ has recently transitioned from q_0 while processing some earlier sub-monomial $m_{j'}$ where $j' < j$. Between the time ρ returning to q_0 after starting from q_0 for $m_{j'}$ and the completion of m_j, at least all sub-monomials $m_{j'}, m_{j'+1}, \ldots, m_j$ of the monomial m are processed to produce a single m'_ℓ in m_ρ. Thus, the sum of exponents of ξ variables in the sub-monomial m'_ℓ is strictly greater than D_1 and in particular we can say that it is $c \times D_1$ where $c > 1$.

This completes the proof. □

F.4 Proof of Claim 4.1.3

Proof. Recall that $f = \sum_{i\in[s]} \prod_{j\in[D_2]} Q_{i,j}$. For ease of notation, we assume that the position of linear forms within each $Q_{i,j}$ polynomial starts from 1. For a monomial $m = m_1 \cdot m_2 \cdots m_{D_2}$ with coefficient α_m, the polynomial $\hat{f}_{\alpha_m \cdot m}$ is the sum of all monomials obtained from computation paths ρ labeled by m from Case 1:

$$\begin{aligned} \hat{f}_{\alpha_m \cdot m} &= \alpha_m \cdot \hat{f}_m \\ &= \alpha_m \times \sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} m_\rho. \end{aligned}$$

For each $j < D_2$, the boundary between m_j and m_{j+1} in m is respected in all such computation paths ρ. In particular, for each path ρ and sub-monomial m_j, we can associate a set $I_{\rho,j} \subseteq [D_1]$ such that $r \in I_{\rho,j}$, if the path ρ makes transition from q_{k-1} to q_k (for some $k \in [s-1]$) or from q_{s-2} to q_0 while reading a variable at position r in m_j. The transition is unique for given ρ and $r \in I_{\rho,j}$. Since computational path ρ respects all boundaries of the sub-monomials, it makes exactly $s-1$ transitions and returns to q_0 when starting to process m_{j+1}. Thus, $|I_{\rho,j}| = s-1$.

Let $I_{\rho,j} = \{\ell_1, \ell_2, \cdots, \ell_{s-1}\}$, where $\xi_{I_{\rho,j}} = \xi_1^{\ell_1}.\xi_2^{\ell_2-\ell_1} \cdots \xi_s^{D-\ell_{s-1}}$ $(\ell_1 < \ell_2 < \cdots < \ell_{s-1})$. From the automaton in Figure 1, it follows that $I_{\rho,j}$ can be any subset of size $s-1$. For a sub-monomial

m_j of m, let $m'_{j,\rho}$ be the transformed sub-monomial that includes both commutative variables $Y \sqcup Z$ and n.c. variables ξ. Define $c_{j,\rho}$ as the commutative part of $m'_{j,\rho}$. Then $m'_{j,\rho} = c_{j,\rho} \cdot \xi_{I_{\rho,j}}$.

Given ρ, we can define the sequence of sets $I_{\rho,j} \subseteq [D_1]$ for $j \in [D_2]$ of size $s-1$ and conversely for a sequence of sets $I_j \subseteq [D_1]$ for $j \in [D_2]$ of size $s-1$, we can define the path ρ.

We can then express $\hat{f}_m$ as:

$$\begin{aligned}
\hat{f}_m &= \sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} m_\rho \\
&= \sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} \left(\prod_j m'_{j,\rho} \right) \\
&= \sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} \left(\prod_j c_{j,\rho} \cdot \xi_{I_{\rho,j}} \right) \\
&= \prod_{j=1}^{D_2} \left(\sum_{\rho: q_0 \overset{m}{\rightsquigarrow} q_{s-1}} c_{j,\rho} \cdot \xi_{I_{\rho,j}} \right) \\
&= \prod_{j=1}^{D_2} \left(\sum_{I \subseteq [D_1], |I| = s-1} c_{j,I} \xi_I \right)
\end{aligned}$$

Here, $c_{j,\rho}$ and $c_{j,I}$ are commutative monomials over $Y \sqcup Z$ defined by the path ρ and the set I, respectively. The last two equalities hold because, $I_{\rho,j}$ can be any subset of size $s-1$ and is independent of $I_{\rho,k}$ for $k \neq j$ (where $k \leq D_2$).

Recall that $Mon(f)$ is the set of all monomials computed by the depth-5 +-regular circuit for f. Since coefficients of sub-monomials do not change due to the automaton's operations, we can ignore them in the notation for ease. Therefore, we have:

$$\begin{aligned}
\hat{f} &= \sum_{m \in Mon(f)} \hat{f}_m \\
&= \sum_{m \in Mon(f)} \prod_{j=1}^{D_2} \left(\sum_{I \subseteq [D_1], |I| = s-1} c_{j,I} \xi_I \right) \\
&= \sum_{i \in [s]} \prod_{j \in [D_2]} \left(\sum_{I \subseteq [D_1], |I| = s-1} Q_{i,j,I} \times \xi_I \right)
\end{aligned}$$

The last equality holds because if $m \in Mon(f)$, it can be written as $m = m_1 \cdot m_2 \cdots m_{D_2}$ and for some $i \in [s]$ such that each sub-monomial $m_j \in X^{D_1}$, has a non-zero coefficient in Q_{ij}. Also suppose m is computed by $\prod_{j \in [D_2]} Q_{i,j}$ for some $i \in [s]$ then by definition $m \in Mon(f)$. This completes the proof. □

F.5 Proof of Claim 4.1.4

Proof. Let $m \in X^D$ be a monomial computed by the depth-5 circuit that computes f. For some $i \in [s]$, m has a non-zero coefficient in $\prod_j Q_{ij}$. The output of the substitution automaton $\mathcal{A}$ on m

can be expressed as $\hat{f}_m + F_m$. Consequently, the output of the automaton $\mathcal{A}$ on the polynomial f is given by:

$$f' = \sum_{m \in Mon(f)} (\hat{f}_m + F_m)$$

which is the polynomial $f' = \hat{f} + F$. □

G Additional Proofs and Automaton

G.1 Proof of Claim 4.0.1

Proof. Since g is an ordered power-sum polynomial, we can treat the n.c. variables $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ as commutative variables without introducing any new cancellations. This is because the exponents of the variables $\xi = \{\xi_1, \xi_2, \ldots, \xi_s\}$ are different for any two distinct monomials of g. Thus, g as a n.c. polynomial is non-zero if and only if g as a commutative polynomial is non-zero. □

G.2 Substitution Automaton for Small Degree Case (see §4.1.3)

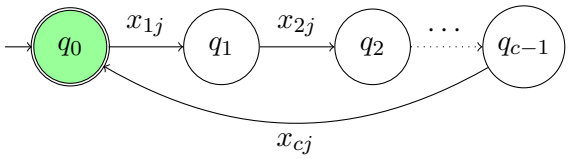

Figure 6: The transition diagram for the variable $x_j : 1 \leq j \leq n$

$$\mathbf{M_{x_j}} = \begin{pmatrix} 0 & x_{1j} & 0 & \ldots & 0 \\ 0 & 0 & x_{2j} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & x_{c-1,j} \\ x_{c,j} & 0 & 0 & \ldots & 0 \end{pmatrix}$$

References

[AGKS15] Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM J. Comput.*, 44(3):669–697, 2015.

[AJMR19] Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial-time identity testing for noncommutative circuits. *Theory of Computing*, 15:1–36, 2019.

[AJR16] Vikraman Arvind, Pushkar S. Joglekar, and S. Raja. Noncommutative valiant's classes: Structure and complete problems. *ACM Trans. Comput. Theory*, 9(1):3:1–3:29, 2016.

[AJR18] Vikraman Arvind, Pushkar S. Joglekar, and Gaurav Rattan. On the complexity of noncommutative polynomial factorization. *Inf. Comput.*, 262:22–39, 2018.

[AL50] Avraham Shimshon Amitsur and Jacob Levitzki. Minimal identities for algebras. *Proceedings of the American Mathematical Society*, 1(4):449–463, 1950.

[BW05] Andrej Bogdanov and Hoeteck Wee. More on noncommutative polynomial identity testing. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 92–99. IEEE, 2005.

[DESW11] Erik D. Demaine, Sarah Eisenstat, Jeffrey O. Shallit, and David A. Wilson. Remarks on separating words. In Markus Holzer, Martin Kutrib, and Giovanni Pighizzini, editors, *Descriptional Complexity of Formal Systems - 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25-27, 2011. Proceedings*, volume 6808 of *Lecture Notes in Computer Science*, pages 147–157. Springer, 2011.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[FS13] Michael A Forbes and Amir Shpilka. Quasipolynomial-time identity testing of noncommutative and read-once oblivious algebraic branching programs. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 243–252. IEEE, 2013.

[Hya77] Laurent Hyafil. The power of commutativity. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 171–174. IEEE Computer Society, 1977.

[Nis] Noam Nisan. Lower bounds for non-commutative computation extended abstract. In *STOC*, pages 410–418. Citeseer.

[RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.

[SB96] Jeffrey O. Shallit and Yuri Breitbart. Automaticity I: properties of a measure of descriptional complexity. *J. Comput. Syst. Sci.*, 53(1):10–25, 1996.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il