

New Algorithmic Results using Noncommutative Algebraic Complexity

By

Abhranil Chatterjee

MATH10201604004

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

for the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

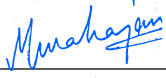


May, 2022

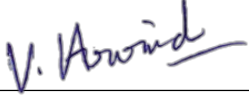
Homi Bhabha National Institute

Recommendations of the Viva Voce Committee

As members of the Viva Voce Committee, we certify that we have read the dissertation prepared by Abhranil Chatterjee entitled “New Algorithmic Results using Noncommutative Algebraic Complexity” and recommend that it may be accepted as fulfilling the thesis requirement for the award of Degree of Doctor of Philosophy.




Chairperson - Meena Mahajan Date: May 11, 2022




Guide/Convenor - V. Arvind Date: May 11, 2022



Co-guide - Partha Mukhopadhyay Date: May 11, 2022



Examiner - Jayalal Sarma Date: May 11, 2022



Member 1 - Saket Saurabh Date: May 11, 2022



Member 2 - Vikram Sharma Date: May 11, 2022



Member 3 - Venkatesh Raman Date: May 11, 2022

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to HBNI.

We hereby certify that we have read this thesis prepared under our direction and recommend that it may be accepted as fulfilling the thesis requirement.

Date: May 11, 2022

Place: Chennai


Co-guide


Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.



Abhranil Chatterjee

DECLARATION

I hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.



Abhranil Chatterjee

LIST OF PUBLICATIONS ARISING FROM THE THESIS

Journals

1. [\[ACDM20b\]](#) **On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, **Chicago Journal of Theoretical Computer Science**, 2020, Vol. 2020.
2. [\[ACDM22\]](#) **Fast Exact Algorithms Using Hadamard Product of Polynomials**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, **Algorithmica**, 2022, Vol. 84, p.436-463.

Conferences

1. [\[ACDM19a\]](#) **Efficient Black-Box Identity Testing for Free Group Algebras**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, **APPROX/RANDOM 2019**, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA, pages 57:1–57:16, 2019.
2. [\[ACDM19c\]](#) **Fast Exact Algorithms Using Hadamard Product of Polynomials**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, In Arkadev Chattopadhyay and Paul Gastin, editors, 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, **FSTTCS 2019**, December 11-13, 2019, Bombay, India,

volume 150 of LIPIcs, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

3. [\[ACDM19b\]](#) **On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, In Pinyan Lu and Guochuan Zhang, editors, 30th International Symposium on Algorithms and Computation, **ISAAC 2019**, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China, volume 149 of LIPIcs, pages 38:1–38:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
4. [\[ACDM20a\]](#) **A Special Case of Rational Identity Testing and the Brešar-Klep Theorem**, V. Arvind, *Abhranil Chatterjee*, Rajit Datta, and Partha Mukhopadhyay, In Javier Esparza and Daniel Král', editors, 45th International Symposium on Mathematical Foundations of Computer Science (**MFCS 2020**), volume 170 of Leibniz International Proceedings in Informatics (LIPIcs), pages 10:1–10:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.


Abhranil Chatterjee

ACKNOWLEDGEMENTS

It was 2017, it was the second semester of my coursework at IMSc. I was yet to decide on my research area. For the first time, I was introduced to the Complexity Theory course and I found it interesting. Coincidentally, a workshop on recent progress on algebraic complexity was organized at IMSc at the same time. It was enough to make my mind, I wanted to be a complexity theorist.

I am fortunate enough to have Arvind as my advisor. I was scared to discuss with an omniscient person with my limited knowledge. I told him that I would like to read for the first year before starting my research journey. I remember the exact words he said to me: *“Don’t think that I am sitting at the top of the mountain and you will climb there. Research is a continuous journey and if you find something you don’t know, just learn it!”* Those simple words boosted my confidence. Thank you Arvind for having faith in me. He is more than an academic advisor to me. His clarity of thinking, way of attacking problems, deep insights into the subject, and his ability to switch instantly between academic and non-academic subjects (being the Director of the institute, he had to do that frequently!) have enkindled me greatly.

Secondly, I wish to thank Partha Mukhopadhyay. He is officially my co-guide, but more than that he is like an elder brother to me who is always there for you with any help, whether it is academic or not. Thank you Partha for all your support. We spent a lot of hours together discussing a wide range of topics. It ranges from the recently published paper at eccc to the taste of Biriyani. His continuous encouragement is instrumental in my research.

I am really grateful to have Rajit, my gurubhai as my collaborator. Working together with Arvind, Partha and Rajit have been a great experience. It felt like a family,

all of us spending time together. We frequently had a heated argument in the office followed by some poor jokes. These people made my Ph.D. days very much enjoyable.

I am really indebted to Meena, Venkatesh, Jam, Saket, Vikram, Kamal, CRS — all the faculty members at IMSc, for their teaching, continuous support, and for providing a nice academic environment. My sincere thanks to Meena for her encouragement, many interesting discussions, and organizing seminar series even during the pandemic.

I would like to thank all the members at IMSc. I am fortunate enough to have some nice friends at IMSc and to spend time with them. I won't be able to mention all of you, but you are really special to me. My sincere thanks to Ashwin and Gaurav, spending time with these lovely idiots was really special.

I have spent a lot of time at CMI. I would like to thank all the members of CMI for their hospitality.

I would like to thank my parents and all my family members. This journey would have been impossible without their love and blessings, especially during the pandemic. Most importantly, I would like to thank Ananya, my better half. Her continuous support gave me confidence in the tough times. Thank you for always standing by my side. Finally, I am thankful to my son, Anukameen, without whom this thesis would have been submitted six months ago!

Contents

Summary	i
List of Figures	iii
1 Introduction	1
2 Background on Noncommutative Algebraic Complexity	19
3 Image of Noncommutative Polynomials and Applications to Rational Identity Testing	37
3.1 Invertible Image of Noncommutative ABPs	38
3.2 Trace of Image of Noncommutative ABPs	44
3.3 Special Instances of Rational Identity Testing	49
4 Efficient Identity Testing of Free Group Algebras	61
4.1 An Amitsur-Levitzki Type Theorem	65
4.1.1 Black-box identity test	72
4.1.2 Reconstruction of sparse expressions	72

4.2	Exponential Degree and Exponential Sparsity	73
4.3	Over Small Finite Fields	81
5	Fast Exact Algorithms using Hadamard Product of Polynomials	83
5.1	Hadamard Product Framework	89
5.2	The Sum of Coefficients of Multilinear Monomials	93
5.2.1	Some Applications	96
5.3	Multilinear Monomial Detection	102
5.4	Deterministic Algorithms for Depth Three Circuits	107
5.5	A Comparison to Related Works	109
6	On Explicit Branching Programs for the Rectangular Determinant	
	and Permanent Polynomials	113
6.1	Explicit construction of ABPs for $S_{n,k}^*$ and Noncommutative Rectan- gular Permanent	115
6.2	Explicit ABP construction for Noncommutative Determinant and Related Polynomials	121
6.3	Hardness of Rectangular Determinant Over Matrix Algebras	127
6.3.1	Computing over Small Dimensional Algebras	131
7	Conclusion	135
	Bibliography	139

Summary

Algebraic Complexity is the study of the complexity of computing multivariate polynomials where the complexity of a polynomial is the number of arithmetic operations such as additions, and multiplications required to compute it. The broad goal of this thesis is to explore the power of noncommutative computation, a subarea of algebraic complexity. We present new algorithmic results using tools and techniques from noncommutative algebraic complexity in this thesis.

In the first part of the thesis, we focus on the image of noncommutative polynomials in various general settings. We explore the algorithmic questions centered around the invertibility and the trace of the image of a noncommutative polynomial. We show that we can leverage ideas from algebraic automata theory to answer these questions. We also study the image of free group algebra functions. It is a generalization of noncommutative polynomials where we allow noncommuting variables as well as their inverses as inputs. We obtain efficient identity testing algorithms for such functions.

The second part of the thesis shows new algorithmic and arithmetic circuit upper bound results, mainly in the context of parameterized complexity, that utilizes new ideas from the noncommutative computation. We address two well-studied algorithmic problems in this area, multilinear monomial detection, and multilinear monomial counting, and show their connection to the computation of noncommutative Hadamard product. This connection leads to a new approach

called symmetrization that yields faster algorithms for these problems. Finally, we study new upper bound results for the noncommutative rectangular permanent, the noncommutative rectangular determinant, and some related polynomials. It complements the rank-based lower bound results for the corresponding polynomials.

Overall this thesis makes some progress in our understanding of the power of noncommutative algebraic complexity.

List of Figures

2.1	An arithmetic circuit computing $x_1^2x_2 + x_1x_2^2 + x_1x_2 + x_2^2 + x_2$	20
2.2	An ABP computing $10x_1^2 - 18x_2^2 - 4x_1x_2 + 46x_2x_1$	21
3.1	Example of the automata when $d = 6$.	47
3.2	Edge Labels having linear form and inverses	53
3.3	Edge Labels rewritten as *-rational expression after applying the shift	
	$x_i \mapsto 1 - x_i$.	53
3.4	Edge Labels replaced by an appropriate automaton.	54
4.1	The transition diagram of the automaton for x_i variables for degree-4	
	functions	69
4.2	The transition diagram of the automaton for x_0 variables for degree-4	
	functions	71
4.3	The transition diagram of the automaton	71
4.4	The transition diagram of the automaton	76
4.5	The transition diagram of the automaton at <i>Encode</i> stage	77
4.6	The transition diagram of the automaton at <i>Skip</i> stage	78

Chapter 1

Introduction

Theoretical computer science is, broadly speaking, the study of computation in different computational models at a suitable level of abstraction that separates it from the messy details of real-world computation. One ubiquitous theoretical model of computation is the Turing machine.

Computational complexity theory is a prominent subfield of theoretical computer science that deals with *efficient computation*. There are different measures of efficient computation, depending on the problem and the computational model being studied and the broad goal is to minimize the computational resources required to solve a given problem.

To understand the computational complexity of a problem, we seek answers to the following questions: (a) what is the most efficient algorithm to solve it? (b) what makes the problem hard or what is the lower bound on the resources required to solve the problem? These two objectives lead to classifying problems into different complexity classes according to the required resources and studying the relative power of these classes.

The running time of an algorithm for a given problem is the most natural and

well-studied measure of efficiency, where the algorithm’s running time is bounded by a function $T(n)$ where n is the input size ($T(n)$ is the number of steps taken by the algorithm, where the notion of a single “step” naturally depends on the model).

In practice, the only known algorithms for many natural optimization problems of interest take exponential time because they essentially solve the problem by brute-force search, it is natural to consider *polynomial-time bounded* algorithms as efficient. Indeed, this turns out to be a robust notion of efficiency as natural models of computation can be simulated on each other with at most a polynomial slow down. This leads to the definition of the complexity class \mathbf{P} , the class of decision problems¹ solvable in polynomial time in a Turing machine.

It turns out that many of the natural optimization problems of interest have the property that the problem instances have efficiently verifiable solutions. It leads to the definition of complexity class \mathbf{NP} , the class of decision problems whose instances have polynomial-size solutions/certificates that can be verified in polynomial time in a Turing machine. Clearly, $\mathbf{P} \subseteq \mathbf{NP}$. The central open problem in computational complexity is $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ or not, independently appeared in the works of Cook [Coo71] and Levin [Lev73]. However, even after five decades, the solution remains elusive. They also identified the “hardest” problems inside the class \mathbf{NP} . This was a breakthrough discovery leading to the notion of \mathbf{NP} -completeness that gave a hardness theory (an explanation for why known algorithms for these problems are exponential-time bounded). This hardness theory, complementing the quest for polynomial-time algorithms is the foundation of modern computational complexity.

In this thesis, we focus on algebraic complexity theory, a subfield of complexity theory, as we discuss next.

¹Considering decision problems rather than optimization problems is a convenient idea that helps with conceptual clarity in understanding hardness of computation: \mathbf{NP} -completeness and other notions.

Algebraic Complexity Theory

Multivariate polynomials are fundamental objects of study in mathematics. Algebraic Complexity is the study of the complexity of computing multivariate polynomials. The complexity of a polynomial is the number of basic arithmetic operations such as additions, and multiplications required to compute it. The natural models for computing polynomials are arithmetic circuits, arithmetic formulas, and algebraic branching programs (formally defined in Chapter 2).

To understand the algebraic complexity of polynomials, polynomial families are classified into complexity classes according to their size of arithmetic circuits. The study of algebraic complexity was initiated by Valiant in his seminal paper [Val79] in which he developed a theory that parallels the NP-completeness theory, now in the setting of polynomials. In Valiant's theory, instead of decision problems we consider polynomial families $\{f_n\}_n$, where each f_n is an n -variate polynomial over \mathbb{F} of $\text{poly}(n)$ degree.

It is easy to see from a simple counting argument that the number of monomials in an n -variate polynomial of degree d is at most $\binom{n+d}{d}$. Therefore any n -variate polynomial of degree $\text{poly}(n)$ trivially has an arithmetic circuit of size exponential in n by writing it as a sum of monomials. It is natural to consider *polynomial-size bounded* arithmetic circuits as efficient computation of a polynomial. This leads to the definition of the complexity class VP, the algebraic analog of the class P, the class of polynomial families $\{f_n\}_n$, where each f_n is an n -variate polynomial over \mathbb{F} of $\text{poly}(n)$ degree which is computable by an arithmetic circuit of size $\text{poly}(n)$. For example, the determinant polynomial and the elementary symmetric polynomial are in VP. See [Sap15] for a detailed exposition.

It turns out that many polynomials of interest, for example, the permanent polynomial, are not known to have small circuits. It leads to the definition of

complexity class VNP , the algebraic analog of the class NP^2 in the following way. A class of polynomial family $\{f_n\}$ is in the class VNP if there exists a polynomial family $\{g_n\}$ in VP such that, for some polynomial $p(n)$,

$$f_n(x_1, \dots, x_n) = \sum_{(a_1, \dots, a_{p(n)}) \in \{0,1\}^{p(n)}} g_{n+p(n)}(x_1, \dots, x_n, a_1, \dots, a_{p(n)}).$$

Clearly, $\text{VP} \subseteq \text{VNP}$. The fundamental question in algebraic complexity is to separate VP and VNP . The permanent polynomial plays an important role as it is identified as the “hardest” polynomial family inside the class VNP i.e. the VNP -complete polynomial family. Therefore, proving that the permanent polynomial does not have an arithmetic circuit of polynomial-size will separate VP and VNP .

The best known lower bound on general circuits for explicit polynomials in VP is due to Baur and Strassen [BS83] who have shown an $\Omega(n \log n)$ lower bound for the polynomial $\sum_{i=1}^n x_i^n$. Unfortunately, no better lower bounds are known still. This absence of progress has led to the study of lower bounds for restricted models such as monotone circuits, multilinear circuits, bounded depth circuits, and noncommutative circuits. For more details on the lower bound results, see the survey of [Sap15].

A fundamental algorithmic problem, closely related to proving arithmetic circuit lower bounds, is Polynomial Identity Testing (PIT): given an arithmetic circuit as input, the problem is to decide whether or not it computes the identically zero polynomial. This can be studied both in the black-box model where the algorithm can access the circuit only by evaluating it at different inputs or in the white-box model where the algorithm has the circuit as input. A well-known and simple randomized black-box algorithm for the problem is by evaluation at a random input

² VNP is algebraic analog of the class $\#\text{P}$ which is the class of the counting problems associated with the decision problems in the class NP : counting the number of solutions/certificates for input instances.

due to the Polynomial Identity Lemma [DL78, Zip79, Sch80]. No subexponential-time deterministic algorithm is known except for some special cases. Devising an efficient deterministic algorithm for PIT for general circuits appears to be a challenging problem in the field. Relatedly, derandomization of the above-mentioned simple randomized algorithm has interesting consequences in proving lower bounds [KI04, Agr05]. See the survey of [SY10] for a detailed exposition on deterministic PIT algorithms and their connection to proving lower bounds.

Noncommutative Computation

The complexity-theoretic study of noncommutative computation was initiated by Nisan in his seminal paper [Nis91]. In the arithmetic circuit model for noncommutative computation, the arithmetic operations are addition and multiplication. The input gates are labeled by indeterminates x_1, x_2, \dots, x_n or scalars from a prescribed field \mathbb{F} . Multiplication gates respect the order of their inputs. That is, for each $i, j \in [n]$, such that $i \neq j$, $x_i x_j \neq x_j x_i$. Such circuits compute *noncommutative polynomials*. These are elements of the *free algebra* $\mathbb{F}\langle x_1, x_2, \dots, x_n \rangle$, where the x_i are free noncommuting variables. To illustrate how commutativity makes a difference, consider the polynomial $x_1^2 - x_2^2$. If x_1 and x_2 commute, the polynomial can be computed as $(x_1 + x_2)(x_1 - x_2)$ using one multiplication only. However, we require at least two multiplications if x_1 and x_2 do not commute.

Analogous to commutative arithmetic computation, the central questions are to show circuit size lower bounds for explicit noncommutative polynomials and the derandomization of polynomial identity testing (PIT) for noncommutative circuits (or subclasses of circuits). Surprisingly, for general circuits no better results are known than in the commutative setting. There is, however, nontrivial progress for

polynomials computable by noncommutative ABPs. Nisan [Nis91] has shown an explicit polynomial that has a linear size noncommutative circuit but requires a noncommutative algebraic branching program (and formula) of exponential size, thus separating noncommutative ABPs and circuits.

Polynomial identity testing (PIT) for noncommutative models of computation is motivated by the hope that efficient deterministic PIT algorithms should be easier than their commutative counterparts. Indeed, Raz and Shpilka [RS05] have shown a deterministic polynomial-time PIT algorithm for noncommutative ABPs in the white-box model. A quasi-polynomial time derandomization is also known for the black-box model [FS13]. However, we do not have a polynomial-time, even randomized, PIT algorithm for general polynomial-sized noncommutative circuits. The randomized polynomial-time PIT algorithm for noncommutative circuits computing a polynomial of polynomially bounded degree [BW05] follows from the Amitsur-Levitzki theorem [AL50] which states that a nonzero polynomial $p \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ of degree $< 2k$ cannot be an identity for the ring $\text{Mat}_k(\mathbb{F})$ of $k \times k$ matrices over \mathbb{F} . It is also known [AJMR17] that a nonzero noncommutative polynomial does not vanish on matrices of dimension logarithmic in the sparsity of the polynomial. This yields a randomized polynomial-time identity test for noncommutative circuits computing polynomials of exponential degree and exponential sparsity.

Rational Identity Testing

Hrubeš and Wigderson [HW15] initiated the study of noncommutative computation with inverses. These computations produce *noncommutative rational functions* which are elements of the *free skew-field*. They introduce the *rational identity testing* (RIT) problem [HW15]: Given a noncommutative rational formula, determine if it is zero in the free skew-field of noncommutative rational functions. By definition, a

rational expression r computes the zero function in the free skew-field if and only if r has a nonempty domain of definition, and for each $d \in \mathbb{N}$ and substitution from $\text{Mat}_d(\mathbb{F})$, the expression evaluates to the zero matrix if it is defined. Using techniques based on operator scaling and invariant theory, the RIT problem for noncommutative rational formulas is shown [GGOW16] [IQS18] to be in deterministic polynomial time in the white-box model. It is in randomized polynomial time in the black-box model [DM17].

The complexity of RIT for general noncommutative rational circuits remains open. Only an exponential-time upper bound is known. In particular, even for rational circuits of inversion height one (inversion height of a circuit is the maximum number of inverse gates present in any input to output path in the circuit [HW15]), we do not know a subexponential-time randomized algorithm.

This Thesis

This thesis is, broadly, an exploration of noncommutative arithmetic computation from an algorithmic perspective. We present new algorithmic results that use tools from algebraic complexity theory, especially noncommutative computation. We now briefly overview the results.

A. Image of Noncommutative Polynomials and Applications to Rational Identity Testing

For the algebra $\text{Mat}_k(\mathbb{F})$ of $k \times k$ matrices over field \mathbb{F} , the image set of a noncommutative polynomial $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ is defined as

$\text{Im}_k(f) = \{f(M_1, \dots, M_n) : (M_1, \dots, M_n) \in \text{Mat}_k^n(\mathbb{F})\}$ for some k . A *polynomial identity* for the matrix algebra $\text{Mat}_k(\mathbb{F})$ is a noncommutative polynomial f such

that $\text{Im}_k(f)$ contains only the zero matrix. The PIT problem of noncommutative

polynomials is closely related to the study of image of noncommutative polynomials. In this thesis, we study the invertibility and span of the image set of noncommutative polynomials. We study the following problem first: Given a noncommutative ABP, find a matrix substitution such that the output matrix is invertible. Our goal is to obtain a deterministic subexponential-time algorithm for this problem. Interestingly, Forbes and Shpilka [FS13] have shown a deterministic quasi-polynomial-time algorithm to find a matrix substitution such that the output matrix is nonzero. Indeed, they show a hitting set for noncommutative ABPs. A hitting set for a class of noncommutative polynomials is a set of matrix tuples such that for each nonzero polynomial in that class, there exists a matrix substitution such that the output matrix is nonzero. However, the output matrix obtained from the Forbes-Shpilka hitting set is always singular. We ask for a stronger notion of hitting set, which we define as an *invertible hitting set* that ensures for each nonzero polynomial there exists a matrix substitution such that the output matrix is invertible. It is not hard to see that the existence of an invertible hitting set follows from Amitsur's theorem on universal division algebra [Ami66] which promises that if f is nonzero on $k \times k$ matrices then f has invertible matrices in its image. We show the following:

Theorem. *There exists an invertible hitting set $\mathcal{H} \subseteq \text{Mat}_d^n(\mathbb{F})$ of size $s^{O(\log d)}$ for n -variate polynomials of degree- d computable by noncommutative ABPs of size s . Moreover, we can construct it in deterministic $s^{O(\log d)}$ -time.*

Another algorithmic question related to the linear span of the image of noncommutative polynomials is motivated by an interesting theorem due to Brešar and Klep [BK08]. Let $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ be any noncommutative polynomial, where \mathbb{F} is a field of zero characteristic. Their theorem states that one of the following is true: (a) $\text{Im}_k(f) = 0$, (b) the span of $\text{Im}_k(f)$ consists of all scalar multiples of the identity matrix I_k , (c) the span of $\text{Im}_k(f)$ is all trace zero matrices over $\text{Mat}_k(\mathbb{F})$,

(d) the span of $\text{Im}_k(f)$ is $\text{Mat}_k(\mathbb{F})$.

It raises a natural algorithmic question: Given a noncommutative polynomial f and the matrix algebra $\text{Mat}_k(\mathbb{F})$, to efficiently determine which of the four cases occur.

A randomized polynomial-time algorithm follows easily by substituting the noncommuting variables with generic $k \times k$ size matrices and evaluating the commuting generic variables randomly. We show the following result which yields an efficient deterministic algorithm for $k \geq \deg(f)$.

Theorem. *Given a noncommutative ABP A of size s computing a polynomial $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ of degree d , there is a deterministic $\text{poly}(n, s, d)$ -time algorithm to check if $\text{Im}_k(f)$ is trace zero over $\text{Mat}_k(\mathbb{F})$ for all $k \geq d$. If A is given by black-box access, there is a deterministic $(ns)^{O(\log d)}$ -time algorithm to check if $\text{Im}_k(f)$ is trace zero for all $k \geq d$ where $|\mathbb{F}| \geq \text{poly}(n, s, d)$.*

As we show in Chapter [3](#) this theorem yields a deterministic polynomial-time algorithm to check which of the four conditions of the Brešar-Klep theorem holds for matrix algebras of dimension $k \geq d$ for a noncommutative polynomial f given by an ABP.

Applications to Rational Identity Testing

We note that the invertible hitting set of noncommutative ABPs solves a very special case of RIT where we have a single inverse gate that is at the output of an ABP.

In general, a noncommutative rational circuit of inversion height 1 can be expressed as a polynomial expression where some of the variables are replaced with inverses of noncommutative polynomials. Thus, the simplest next case to consider for identity testing is to allow inverses on linear forms. As we do not have a subexponential-time algorithm even in this case, it naturally leads us to consider the easier case of RIT for algebraic branching programs whose multi-edges are labeled by affine linear

forms or inverses of affine linear forms. Such ABPs compute rational expressions of inversion height one in the free skew field. The rational expression computed by the ABP is the sum over each source-to-sink path P of the ordered product of affine linear forms or their inverses labeling P . The size of the ABP is defined as the total number of nodes and multi-edges. For this model, a deterministic quasi-polynomial time white-box algorithm and a randomized quasi-polynomial time black-box algorithm already follow from the results of Garg et. al. [GGOW16, IQS18] and Derksen and Makam [DM17], respectively. We improve these bounds in our special case by showing a deterministic polynomial time white-box algorithm and a deterministic quasi-polynomial time algorithm for the black-box model.

Theorem. *Given an ABP (in white-box) of size s where each edge is labeled by an affine linear form or inverse of an affine linear form over \mathbb{Q} , there is a deterministic $\text{poly}(s, n)$ time algorithm to decide if it computes the zero function in $\mathbb{Q}\langle x_1, \dots, x_n \rangle$. If such an ABP is given as a black-box then there is a deterministic $(ns)^{O(\log(ns))}$ -time algorithm for it.*

B. Efficient Identity Testing of Free Group Algebras

Our next result is a continuation of the study of the image of noncommutative polynomials in a more general setting. Let x_1, \dots, x_n be n indeterminates and Γ be the free group generated by x_1, \dots, x_n ³. We consider free group algebra functions in $\mathbb{F}[\Gamma]$ and define a notion of degree and sparsity for free group algebra functions. This can be seen as a natural special case of rational identity testing, as rational circuits with inverse gates only at the bottom-most layer compute free group algebra expressions. Our goal is efficient black-box identity testing for rational expressions in the free group algebra $\mathbb{F}[\Gamma]$.

Our first result is an Amitsur-Levitzki type theorem [AL50] for $\mathbb{F}[\Gamma]$. Let A be an

³The free group Γ consists of the *reduced* words i.e. words of $\langle x_1, \dots, x_n, x_1^{-1}, \dots, x_n^{-1} \rangle$ with the group axioms that for each $i \in [n]$, $x_i x_i^{-1} = 1$.

associative algebra with identity over \mathbb{F} . An element $f \in \mathbb{F}[\Gamma]$ is an *identity* for A if

$$f(a_1, \dots, a_n) = 0,$$

for all $a_i \in A$ such that a_i^{-1} is defined for each $i \in [n]$.

Theorem. *Let \mathbb{F} be any field of characteristic zero and $f \in \mathbb{F}[\Gamma]$ be a nonzero free group algebra function of degree at most d . Then f is not an identity for the matrix algebra $\text{Mat}_{2d}(\mathbb{F})$.*

The following corollary is immediate.

Corollary (black-box identity testing in free group algebras). *There is a black-box randomized $\text{poly}(n, d)$ identity test for degree- d free group algebra functions in $\mathbb{F}[\Gamma]$.*

If the black-box contains a sparse function, we show efficient deterministic algorithms for identity testing and interpolation algorithm.

Theorem (reconstruction of sparse functions). *Let \mathbb{F} be any field of characteristic zero and f is a free group algebra function in $\mathbb{F}[\Gamma]$ of degree- d and sparsity- s given as black-box. Then we can reconstruct f in deterministic $\text{poly}(n, d, s)$ time with matrix-valued queries to the black-box.*

Nonzero polynomials in $\mathbb{F}\langle x_1, \dots, x_n \rangle$ of sparsity- s cannot vanish on $O(\log s)$ dimensional matrix algebras [AJMR17]. We obtain a similar result for $\mathbb{F}[\Gamma]$: nonzero functions in $\mathbb{F}[\Gamma]$ of degree D and sparsity s do not vanish on $O(\log s)$ dimensional matrices. This yields a randomized polynomial-time identity test if the black-box contains a free group algebra function f of exponential degree and exponential sparsity.

Theorem. *Let \mathbb{F} be any field of characteristic zero. Then, a degree- D function $f \in \mathbb{F}[\Gamma]$ of sparsity s is not an identity for the matrix algebra $\text{Mat}_k(\mathbb{F})$ for $k \geq c \log s$ for a constant c .*

Corollary. *Given a degree- D free group algebra function $f \in \mathbb{F}[\Gamma]$ of sparsity s as black box, we can check whether f is identically zero or not in randomized $\text{poly}(n, \log D, \log s)$ time.*

We have stated our results for fields of characteristic zero for simplicity. With suitable modifications, the results easily extend to fields of positive characteristics.

C. Fast Exact Algorithms using Hadamard Product of Polynomials

An interesting aspect of noncommutative computation is that the noncommutative determinant can be used to define an unbiased estimator for the commutative permanent polynomial as initially discovered by Godsil and Gutman [\[GG81\]](#)⁴. Nevertheless, it turns out that we can use techniques from noncommutative algebraic complexity to design efficient algorithms for combinatorial problems. In this thesis, we explore one such application of using noncommutative computation. We define a notion of scaled Hadamard product of commutative polynomials and show its connection to the computation of noncommutative Hadamard product. The main objective is to use this connection to design faster algorithms for multilinear monomial detection and related problems as we discuss now.

Let \mathbb{F} be any field and x_1, \dots, x_n be n commuting variables. Koutis and Williams [\[Kou08\]](#) [\[Wil09\]](#) [\[KW16\]](#) introduced and studied two natural algorithmic problems in arithmetic circuits:

1. Given as input an arithmetic circuit C of $\text{poly}(n)$ size computing a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$, the k -multilinear monomial counting problem, denoted (k, n) -MLC is to compute the sum of the coefficients of all degree- k multilinear monomials in the polynomial f .

⁴Although this direction is not fruitful for the design of approximation schemes as the noncommutative determinant turns out to be hard to compute [\[JSV04\]](#) [\[AS18\]](#)

2. The k -multilinear monomial detection problem, denoted k -MMD, is to test if there is a degree- k multilinear monomial in the polynomial f with a non-zero coefficient.

These problems have attracted attention in recent times. These are natural generalizations of the well-studied k -path detection and counting problems in a given graph [Kou08]. Moreover, some other combinatorial problems like k -Tree, m -Dimensional k -Matching [KW16], well-studied in parameterized complexity, reduces to these problems. In general, the exact counting versions of these counting problems are $\#W[1]$ -hard⁵. For these counting problems, improvements to the trivial $O^*(n^k)$ time exhaustive search algorithm are known only in some cases (like counting k -paths) [BHKK09]. Whether one can design an algorithm for (k, n) -MLC that avoids exhaustive search, is an interesting problem as it would yield a faster algorithm for all these counting problems. This was explicitly mentioned by Koutis and Williams [KW16] as an open problem. In the same paper, they give an algorithm of run time $O^*(n^{k/2})$ to compute the *parity* of the sum of coefficients of degree- k multilinear monomials.

We give a new approach to the k -MMD, (k, n) -MLC problems, and related problems. Our algorithms are based on computing the *Hadamard product of polynomials*. The Hadamard product of polynomials $f, g \in \mathbb{F}[x_1, \dots, x_n]$ is defined as $f \circ g = \sum_m ([m]f \cdot [m]g) \cdot m$, where $[m]f$ denotes the coefficient of the monomial m in f . The Hadamard product has proven useful in noncommutative computation [AJS09, AS18]. A contribution of this thesis is an efficient implementation of the Hadamard product in the *commutative* setting and its application in the design of efficient FPT and exact algorithms. In general, transferring techniques from circuit complexity to algorithm design is an interesting

⁵In parameterized complexity theory, if a parameterized counting problem with parameter k is $\#W[1]$ -hard, then it is unlikely to have an $f(k)\text{poly}(n)$ -time algorithm for that problem under reasonable complexity-theoretic assumptions.

research direction. We refer the reader to the articles of Williams [Wil14b, Wil14a].

Consider the elementary symmetric polynomial $S_{n,k}$ of degree k over the n variables x_1, x_2, \dots, x_n . By definition, $S_{n,k}$ is the sum of all the degree- k multilinear monomials. Computing the Hadamard product of $S_{n,k}$ and a polynomial f sieves out precisely the degree- k multilinear part of f . This connection with the symmetric polynomial gives the following result.

Theorem. *The (k,n) -MLC problem for an input polynomial in $\mathbb{F}[x_1, \dots, x_n]$, given as input by an algebraic branching program of size s , has a deterministic $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ -time algorithm. When f is given as input by an arithmetic circuit C of size s , has a deterministic $O^*\left(\binom{n}{\lfloor k/2 \rfloor} \cdot s^{c \cdot \log k}\right)$ -time algorithm where c is a constant.*

For the above theorem, the underlying field \mathbb{F} could be any field whose elements can be efficiently represented, with efficiently computable field operations. We note that the above run time beats the naive $O^*(n^k)$ bound, answering the question asked by Koutis and Williams [KW16]. The notation $\binom{n}{\lfloor i \rfloor}$ stands for $\sum_{j=0}^i \binom{n}{j}$.

Our next algorithmic result is the following.

Theorem. *The k -MMD problem for any arithmetic circuit C of $\text{poly}(n)$ size has a randomized $O^*(4.32^k)$ -time and polynomial space-bounded algorithm.*

The underlying field \mathbb{F} could be any field whose elements can be efficiently represented, with efficiently computable field operations.

Next, we state the results showing fast *deterministic* algorithms for depth-three circuits. We use the notation $\Sigma^{[s]}\Pi^{[k]}\Sigma$ to denote depth three circuits of top Σ gate fan-in s and the Π gates compute the product of k homogeneous linear forms over x_1, \dots, x_n .

Theorem. *Given any homogeneous depth three $\Sigma^{[s]}\Pi^{[k]}\Sigma$ circuit of degree k , the (k,n) -MLC problem can be solved in deterministic $O^*(2^k)$ -time. Over \mathbb{Z} , the k -MMD*

problem can be solved in deterministic $O^*(4^k)$ -time. Over finite fields, k -MMD problem can be solved in deterministic $e^k k^{O(\log k)} O^*(2^{ck} + 2^k)$ time, where $c \leq 5$.

D. On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials

We study the rank upper bound of Nisan's matrix for the noncommutative polynomial $S_{n,k}^*$ (a noncommutative symmetrized version of the elementary symmetric polynomial), and some related polynomials to design explicit ABPs. We then discuss the algorithmic applications of this construction using the Hadamard product. The rank of Nisan's matrix for a noncommutative polynomial is used to show lower bounds for ABPs computing that polynomial. We use the rank to obtain new upper bound results.

It is well-known that the k^{th} elementary symmetric polynomial $S_{n,k}$ can be computed by an algebraic branching program of size $O(nk)$. We consider the noncommutative symmetrized version $S_{n,k}^*$, in the ring $\mathbb{F}\langle y_1, \dots, y_n \rangle$ where y_1, \dots, y_n are n noncommuting variables, defined as:

$$S_{n,k}^*(y_1, \dots, y_n) = \sum_{T \subseteq [n]: |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)}.$$

Nisan [Nis91] shows that any ABP for $S_{n,k}^*$ is of size $\Omega\left(\binom{n}{\lfloor k/2 \rfloor}\right)$. Recall that, we use $\binom{n}{\lfloor r \rfloor}$ to denote $\sum_{i=0}^r \binom{n}{i}$. Furthermore, Nisan also shows the *existence* of an ABP of size $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for $S_{n,k}^*$. However, it does not give an algorithm to construct such an ABP in time $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$. The main upper bound question is whether we can achieve any constant factor saving of the parameter k in the exponent, in terms of ABP size and the run time of the construction.

The next polynomial we consider is the rectangular permanent. Given a $k \times n$ rectangular matrix $X = (x_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of commuting variables or a $k \times n$

rectangular matrix $Y = (y_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of noncommuting variables, the rectangular permanent in commutative and noncommutative domains are defined as follows

$$\text{rPer}(X) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k x_{i,\sigma(i)}, \quad \text{rPer}(Y) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k y_{i,\sigma(i)}.$$

Here, $I_{k,n}$ denotes the set of all injections from $[k] \rightarrow [n]$. Alternatively, $\text{rPer}(X) = \sum_{S \subset [n]: |S|=k} \text{Per}(X_S)$ where X_S is the $k \times k$ submatrix whose columns are indexed by the set S . Of course, such a polynomial can be computed in time $O^*(n^k)$ using a circuit of similar size. Can the dependence on k be improved? It is implicit in the work of Williams and Williams [WW13] that the commutative $\text{rPer}(X)$ polynomial has an algebraic branching program of size $O^*(2^k)$. This problem originates from its connection to exact algorithms for certain combinatorial problems [WW13]. We note here that there is an algorithm of run time $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for computing the rectangular permanent over rings and semirings [BHKK10]. The challenge is to obtain an $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for it.

Theorem. *The family of symmetrized elementary polynomials $\{S_{n,k}^*(y_1, \dots, y_n)\}_{n>0}$ and noncommutative rectangular permanent family $\{\text{rPer}(Y)\}_{n>0}$, where Y is a $k \times n$ symbolic matrix has $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABPs over any field.*

The noncommutative determinant polynomial of a symbolic matrix $Y = (y_{i,j})_{1 \leq i,j \leq k}$ is defined as below, with variables in the monomials ordered left to right:

$$\text{Det}(Y) = \sum_{\sigma \in S_k} \text{sgn}(\sigma) y_{1,\sigma(1)} \cdots y_{k,\sigma(k)}.$$

It is known as the *Cayley determinant*. Nisan [Nis91] has also shown that ABPs for the *noncommutative determinant* of a $k \times k$ symbolic matrix require size $\Omega(2^k)$. In this thesis, we give an explicit construction of such an ABP in time $O^*(2^k)$.

Motivated by the aforementioned result of Williams and Williams [WW13] for

rectangular permanent, we also study the complexity of the rectangular determinant polynomial (in the commutative domain) defined as follows.

$$\text{rDet}(X) = \sum_{S \in \binom{[n]}{k}} \text{Det}(X_S).$$

We prove that the rectangular determinant polynomial can be computed using $O^*(2^k)$ -size explicit ABP.

Theorem. *We show that (a) the family of noncommutative determinants $\{\text{Det}(Y)\}_{k>0}$ has 2^k -explicit ABPs over any field, (b) there is a family $\{f_n\}$ of noncommutative degree- k polynomials f_n such that f_n has the same support as $S_{n,k}^*$, and it has 2^k -explicit ABPs, (c) the commutative rectangular determinant family $\{\text{rDet}(X)\}_{k>0}$, where X is a $k \times n$ matrix of variables has 2^k -explicit ABPs.*

Finally, we show the problem of evaluating the noncommutative rectangular determinant over matrix algebras is $\#\mathbf{W}[1]$ -hard for polynomial dimensional matrices. Hence the noncommutative rectangular determinant is unlikely to have an explicit $O^*(n^{o(k)})$ -size ABP.

Theorem. *For any fixed $\epsilon > 0$, evaluating the $k \times n$ rectangular determinant polynomial over $n^\epsilon \times n^\epsilon$ rational matrices is $\#\mathbf{W}[1]$ -hard, treating k as fixed parameter.*

Interestingly, for small dimensional algebra, we can obtain an FPT algorithm as we show that the rectangular determinant (and the rectangular permanent), whose entries are $r \times r$ matrices over any field, can be computed in time $O^*(2^k r^{2k})$.

Organization

In Chapter [2](#) we revisit the lower bounds results and polynomial identity testing algorithms for noncommutative polynomials. The subsequent chapters are presented in the same order as they have been described above. In Chapter [3](#), we study the invertibility and span of the image of noncommutative polynomials and show some applications to the rational identity testing problem. An efficient identity testing algorithm for free group algebras is presented in Chapter [4](#). In Chapter [5](#), we show a connection of noncommutative computation in designing fast exact algorithms using the Hadamard product of polynomials. In Chapter [6](#) we present the construction of explicit branching programs for rectangular permanent, rectangular determinant, and some related polynomials. Finally, we conclude in Chapter [7](#) by summarizing the main results of this thesis and listing down some interesting open problems.

Chapter 2

Background on Noncommutative Algebraic Complexity

In this chapter, we recall some lower bound results and polynomial identity testing (PIT) results for noncommutative algebraic complexity. The aim of this chapter is not to present the state-of-the-art progress in lower bounds or PIT. Rather, it serves as a building block for the subsequent chapters. We start with the definitions of some basic models of computing polynomials. We then mainly focus on the exponential lower bounds and deterministic PIT algorithms for noncommutative ABPs, randomized polynomial-time PIT algorithm for noncommutative circuits, and the notion of Hadamard product of noncommutative polynomials. We then discuss the connection of noncommutative algebraic complexity to algebraic automata theory. Finally, we conclude with an introduction to the rational identity testing problem.

Noncommutative arithmetic complexity deals with the complexity of computing noncommutative polynomials. For instance, noncommutative arithmetic circuits have addition and multiplication gates, and circuit inputs are either variables from $X = \{x_1, x_2, \dots, x_n\}$ or scalars from the field \mathbb{F} . Multiplication gates respect its

input order since the variables are noncommuting. The *free noncommutative ring* $\mathbb{F}\langle X \rangle$ is the ring of all noncommutative polynomials in X -variables over the field \mathbb{F} .

Basic Computational Models for Computing Polynomials

The two most natural models of computing polynomials extensively studied in algebraic complexity are arithmetic circuits and arithmetic formulas.

Definition 1 (Arithmetic Circuit). *Let \mathbb{F} be a field and x_1, \dots, x_n be n indeterminates. An arithmetic circuit computing a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$ is a directed acyclic graph computing a polynomial at each node. The source nodes are labeled by input variables x_1, \dots, x_n or scalars in \mathbb{F} . Each internal node is labeled by a $+$ -gate or a \times -gate computing the sum or the product respectively of the polynomials computed by its children and f is computed at some sink node.*

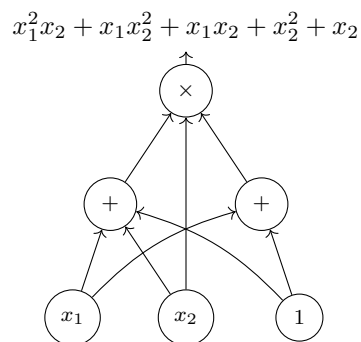


Figure 2.1: An arithmetic circuit computing $x_1^2 x_2 + x_1 x_2^2 + x_1 x_2 + x_2^2 + x_2$

If the underlying directed acyclic graph is indeed a tree, we define it as an *arithmetic formula*.

Another well-known model which is of particular interest in this thesis is an algebraic branching program. It is defined as follows.

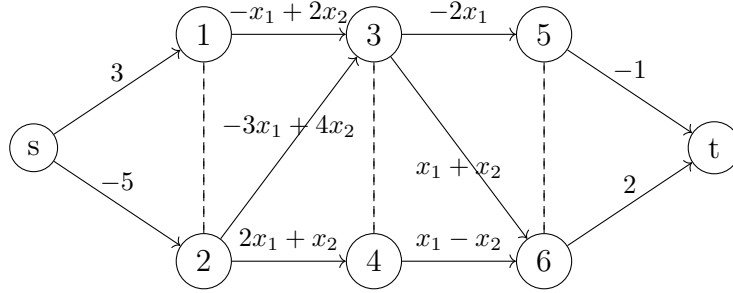


Figure 2.2: An ABP computing $10x_1^2 - 18x_2^2 - 4x_1x_2 + 46x_2x_1$

Definition 2 (Algebraic Branching Programs (ABP)). *An algebraic branching program (ABP) is a directed acyclic graph with one in-degree-0 vertex called the source, and one out-degree-0 vertex called the sink. The vertex set of the graph is partitioned into layers $0, 1, \dots, \ell$, with directed edges only between adjacent layers (i to $i + 1$). The source and the sink are at layers zero and ℓ respectively. Each edge is labeled by a linear form over variables x_1, x_2, \dots, x_n . Let $P = (e_1, e_2, \dots, e_\ell)$ be a source-to-sink directed path and let L_i be the linear form labeling the edge e_i on the path. Then the polynomial computed by the ABP is defined as the sum of products*

$$(2.1) \quad \sum_P \prod_{i=1}^{\ell} L_i,$$

where the sum is over all source-to-sink directed paths P . An ABP is homogeneous if all edge labels are homogeneous linear forms.

It is known that an arithmetic formula (namely, an arithmetic circuit where each gate has fanout one) of size s can be transformed into an ABP of size $\text{poly}(s)$. Furthermore, ABPs of size s can be converted to arithmetic circuits of size $\text{poly}(s)$. A major open problem is to show a separation between any of these models. One can also define these models in the noncommutative setting by fixing an ordering on the multiplications.

Lower Bound of Noncommutative ABPs

The study of noncommutative computation is motivated by the hope that exponential lower bounds in noncommutative models of computation are substantially easier to prove than their commutative counterparts. Indeed, Nisan has shown an exponential lower bound on noncommutative ABPs in his seminal paper [Nis91]. Interestingly, the exponential lower bound on the size of the noncommutative ABP was shown for the palindrome polynomial which is known to have a polynomial-size noncommutative circuit. Therefore, it also yields an exponential separation between noncommutative circuits and noncommutative ABPs. Before we state the main lower bound result, we recall some definitions.

Definition 3 (ABP complexity). *For a noncommutative polynomial, $f \in \mathbb{F}\langle X \rangle$, the ABP complexity of f , $B(f)$ is defined as the minimum size of the noncommutative ABP computing the polynomial f .*

If a noncommutative polynomial of degree d has a small ABP, then any of its degree i (for any $i \leq d$) homogeneous component can also be computed by a small ABP [RS05]. Therefore, to prove an ABP lower bound for a noncommutative polynomial, it suffices to prove it for a homogeneous component of that polynomial (indeed it is true even for commutative polynomials and for circuits also).

We now define a matrix corresponding to every (homogeneous) noncommutative polynomial that plays a crucial role in proving the lower bound.

Definition 4 (Nisan's matrix [Nis91]). *For any degree- d homogeneous noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$, for each $0 \leq k \leq d$, the Nisan's matrix of f for k , $H_k(f)$ is defined as the following $n^k \times n^{d-k}$ matrix. Each row of $H_k(f)$ is indexed by a degree- k word over X^k and each column of $H_k(f)$ is indexed by a degree- $(d-k)$ word over X^{d-k} . For a row indexed by m_1 and a column indexed by m_2 , the $(m_1, m_2)^{th}$ entry of $H_k(f)$ is defined as the coefficient of $m_1 m_2$ in the*

polynomial f , i.e.

$$H_k(f)[m_1, m_2] = [m_1 m_2] f.$$

We are now ready to state the main theorem of [Nis91].

Theorem 1. [Nis91] For any degree- d homogeneous noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$,

$$B(f) = \sum_{k=0}^d \text{rank}(H_k(f)).$$

Now the exponential lower bound for noncommutative polynomials such as the palindrome polynomial, determinant polynomial, permanent polynomial, and many others follows from showing an exponential lower bound on the rank of the corresponding Nisan’s matrix.

We point out that Theorem 1 is extensively used to prove lower bound results mostly. In this thesis, we show some upper bound results and some interesting combinatorial applications of this theorem.

Noncommutative Polynomial Identity Testing

The noncommutative polynomial identity testing problem is an algorithmic problem that asks to determine whether a given noncommutative polynomial is zero in the free algebra. The PIT problems are of two types: *white-box* PIT and *black-box* PIT. For white-box PIT, we are allowed to “see” the circuit structure computing the noncommutative polynomial. In the black-box setting, we can only “see” the output matrix evaluated on some tuple of matrices from a suitable matrix algebra.

If the polynomial is computable by a noncommutative ABP, then the PIT problem is *almost* settled both in white-box and black-box. When the polynomial is computable by a noncommutative circuit or we only have black-box access to the

noncommutative polynomial, then we only have a randomized polynomial-time algorithm in two special cases.

PIT of noncommutative ABPs

For noncommutative algebraic branching programs (ABPs) there is a deterministic polynomial-time PIT algorithm in the white-box model [RS05]. In the black-box model, there is a quasi-polynomial-time deterministic algorithm given by a quasi-polynomial-size hitting set construction [FS13]. In contrast, for commutative algebraic branching programs, efficient deterministic PIT algorithms are known only in very restricted cases. We first recall the white-box PIT algorithm of Raz and Shpilka [RS05].

Theorem 2 (Raz-Shpilka [RS05]). *Given an ABP of width w and d many layers computing a noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$, there is a deterministic $\text{poly}(w, d, n)$ time algorithm to test whether $f \equiv 0$ or not.*

The main idea of the Raz-Shpilka algorithm is that when we merge two consecutive layers of a noncommutative ABP of width w over x_1, x_2, \dots, x_n , even though there can be n^2 many degree-2 monomials, essentially it suffices to maintain only w^2 many monomials as each of the coefficient matrices can be thought of as an w^2 -dimensional vector over \mathbb{F} .

For the black-box case, Forbes and Shpilka [FS13], have shown an efficient construction of quasi-polynomial size hitting set for noncommutative ABPs. Consider the class of noncommutative ABPs of width w , and depth d computing polynomials in $\mathbb{F}\langle X \rangle$. The result of Forbes and Shpilka provide an explicit construction (in quasi-polynomial-time) of a set $\mathcal{H}_{w,d,n}$ contained in $\text{Mat}_{d+1}(\mathbb{F})$, such that for any ABP (with parameters w and d) computing a nonzero polynomial f , there always exists $\alpha \in \mathcal{H}_{w,d,n}$ such that $f(\alpha) \neq 0$.

Theorem 3 (Forbes-Shpilka [FS13](#)). *For all $w, d, n \in \mathbb{N}$, if $|\mathbb{F}| \geq \text{poly}(d, n, w)$, then there is a hitting set $\mathcal{H}_{w,d,n} \subset \text{Mat}_{d+1}(\mathbb{F})$ for noncommutative ABPs of parameters w, d, n such that $|\mathcal{H}_{w,d,n}| \leq (wdn)^{O(\log d)}$ and there is a deterministic algorithm to output the set $\mathcal{H}_{w,d,n}$ in time $(wdn)^{O(\log d)}$.*

Indeed, in the same paper [FS13](#), the authors also obtain a deterministic quasi-polynomial-time PIT algorithm for commutative set-multilinear ABPs and read-once oblivious ABPs. It will be useful for this thesis to present these models and the corresponding PIT results.

Definition 5 (Set-multilinear ABP). *A set-multilinear ABP in the variable set $Y = \{y_{1,1}, \dots, y_{1,d}, \dots, y_{n,1}, \dots, y_{n,d}\}$ is an ABP of depth d , such that each edge between layer $\ell - 1$ and layer ℓ is labeled using a linear form in variables $y_{i,\ell}$ for $i \in [n]$.*

Definition 6 (Read-once oblivious ABP (ROABP)). *An read-once oblivious ABP in the variable set $Z = \{z_1, \dots, z_d\}$ is an ABP of depth d , such that each edge between layer $\ell - 1$ and layer ℓ is labeled using a univariate polynomial in variable z_i of degree at most n .*

Hitting set generator for commutative ROABPs

Now we recall the definition of a hitting set generator and commutative ROABPs.

Let us first define a hitting set generator.

Definition 7 (Hitting Set Generator). *Let \mathcal{C} be a class of circuits computing polynomials on n variables, a polynomial map $\mathcal{G} : \mathbb{F}^m \rightarrow \mathbb{F}^n$ where $m < n$ is a generator for \mathcal{C} if for every polynomial f computed by a circuit in \mathcal{C} , $f \equiv 0$ if and only if $f \circ \mathcal{G} \equiv 0$. If the degree of \mathcal{G} is bounded by r and degree of each polynomial in \mathcal{C} is bounded by d , then clearly, for any $S \subseteq \mathbb{F}$ such that $|S| > (rd)^m$, $\mathcal{G}(S)$ is a hitting set of \mathcal{C} .*

Observe that, a hitting set generator $\mathcal{G} : \mathbb{F}^m \rightarrow \mathbb{F}^n$ can be thought of as an m -input and n -output circuit. The degree of a generator is the maximum degree of an input variable.

Theorem 4 (Forbes-Shpilka). *Let \mathcal{C} be the class of polynomials in $\mathbb{F}[Z]$ which are computable by width s depth d commutative ROABPs. If $|\mathbb{F}| \geq \text{poly}(n, s, d)$ then there is a hitting set generator $\mathcal{G} : \mathbb{F}^m \mapsto \mathbb{F}^d$ where $m = O(\log d)$ and the degree of the generator is at most dns^4 .*

It immediately yields a PIT algorithm for commutative ROABPs.

Connection to the PIT of set-multilinear ABPs and noncommutative ABPs

Given a set-multilinear ABP over Y variables, it is easy to observe that by replacing $y_{i,j}$ by z_j^i , we obtain a ROABP. Moreover, this substitution is identity preserving¹. Therefore, we also obtain a deterministic quasi-polynomial-time PIT algorithm for set-multilinear ABPs.

Let us define a map $\text{SM} : \mathbb{F}\langle X \rangle \rightarrow \mathbb{F}[Y]$ such that for any degree- d word $w \in X^d$, let $w = x_{i_1} \cdots x_{i_d}$, then,

$$\text{SM}(x_{i_1} \cdots x_{i_d}) = y_{i_1,1} \cdots y_{i_d,d}.$$

We extend the definition for any noncommutative polynomial by linearity. Notice that SM is identity preserving. Therefore, given a noncommutative ABP computing a polynomial $f \in \mathbb{F}\langle X \rangle$ as input, if we can compute $\text{SM}(f)$, we can then reduce the PIT problem to the PIT of set-multilinear ABPs. In [\[FS13\]](#), the authors use this connection by substituting the following matrices.

¹Let C_1 and C_2 be two class of polynomials. A map $\sigma : C_1 \rightarrow C_2$ is *identity preserving* if the following is true: $f \in C_1$ is a polynomial identity if and only if $\sigma(f) \in C_2$ is a polynomial identity.

$$\text{For each } i \in [n], \text{ define } M_i = \begin{bmatrix} 0 & y_{i,1} & & & \\ & 0 & y_{i,2} & & \\ & & \ddots & & \\ & & & 0 & y_{i,d} \\ & & & & 0 \end{bmatrix}.$$

PIT of Noncommutative Circuits (in Black-Box)

Let A be an associative algebra with identity over \mathbb{F} . A noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$ is an *identity* for A if

$$f(a_1, \dots, a_n) = 0,$$

for all $a_i \in A$. The study of noncommutative black-box PIT is essentially the study of the complexity of the polynomial identities as we describe next. We have a randomized polynomial-time black-box PIT algorithm for two restricted cases, noncommutative polynomials with polynomially-bounded degree [\[BW05\]](#) and exponentially-bounded sparsity [\[AJMR17\]](#).

Ring of Generic Matrices

Let $\{z_{ij}^{(k)}\}$ be a collection of distinct commuting variables for each integer k and $i, j \in [d]$. Let us define $T_k = (z_{ij}^{(k)})$ for each k . We call these T_k matrices as the $d \times d$ generic matrices. Let $\mathbb{F}[\{z_{ij}^{(k)}\}]$ be the polynomial ring in $\{z_{ij}^{(k)}\}$ variables. Observe that, each T_i is in $\text{Mat}_d(\mathbb{F}[\{z_{ij}^{(k)}\}])$. The ring of generic matrices R_d is defined as the subalgebra generated by these T_k matrices.

Noncommutative PIT for bounded degree

Bogdanov and Wee [BW05] showed a randomized polynomial-time PIT algorithm for noncommutative circuits computing a polynomial of polynomially bounded degree, based on the Amitsur-Levitzki theorem [AL50]. Amitsur-Levitzki theorem studies the complexity of polynomial identities. It states that a nonzero polynomial $f \in \mathbb{F}\langle X \rangle$ of degree $< 2k$ cannot be an identity for the ring $\text{Mat}_k(\mathbb{F})$ of $k \times k$ matrices over \mathbb{F} . In [BW05], the authors use this theorem to obtain a randomized $\text{poly}(n, d)$ -time PIT algorithm for noncommutative polynomials in $\mathbb{F}\langle X \rangle$ of degree d . It simply follows from substituting each x_i by $d' \times d'$ generic matrices where $d' = 2d + 1$. As f is of degree $< 2d'$, $f(T_1, \dots, T_n)$ must be non-zero in R_d , therefore for some i, j , the polynomial g_{ij} computed in the $(i, j)^{\text{th}}$ entry of $f(T_1, \dots, T_n)$ must be non-zero in $\mathbb{F}[\{z_{ij}^{(k)}\}]$. It now reduces to commutative PIT.

Noncommutative PIT for bounded sparsity

Similar to the Amitsur-Levitzki theorem, in [AJMR17], the authors study the complexity of a polynomial identity with respect to the sparsity². They show that a nonzero noncommutative polynomial does not vanish on matrices of dimension logarithmic in the sparsity of the polynomial. This yields a randomized $\text{poly}(n, \log s)$ -time PIT algorithm for noncommutative polynomials in $\mathbb{F}\langle X \rangle$ of sparsity s . It follows from substituting $k \times k$ generic matrices where $k = \lceil \log s \rceil + 1$.

Remark 1. *Let $f \in \mathbb{F}\langle X \rangle$ be a noncommutative polynomial that is computable by a $\text{poly}(n)$ -size circuit. The degree of f can be as large as exponential in n and the sparsity of f can be doubly exponential in n . Therefore, obtaining a $\text{poly}(n)$ -time PIT algorithm (even randomized) for noncommutative polynomials without any restriction in degree or sparsity remains open.*

²Sparsity of a polynomial is the number of monomials in the polynomial with non-zero coefficient.

Hadamard Product of Noncommutative Polynomials

We first define the Hadamard product of two noncommutative polynomials.

Definition 8. *Hadamard product of polynomials* Given two noncommutative polynomials f and g in $\mathbb{F}\langle X \rangle$, the Hadamard product of f and g is defined as the following noncommutative polynomial,

$$f \circ g = \sum_{m \in X^*} [m]f \cdot [m]g \cdot m,$$

where $[m]f$ and $[m]g$ is the coefficient of the monomial m in the polynomial f and g respectively.

The notion of Hadamard product was well-known in algebraic automata theory [BR11, Theorem 5.5]. In [AJS09], [AMS10], [AS10], the authors use this notion for noncommutative polynomials and obtain interesting new results. In mathematics, Hadamard product of two matrices generally refers to their entry-wise product i.e. for two matrices A and B of same dimension m , $A \circ B$ is an $m \times m$ matrix where for every $1 \leq i, j \leq m$, the $(i, j)^{th}$ entry of $A \circ B$ is the product of the $(i, j)^{th}$ entry of A and the $(i, j)^{th}$ entry of B . Recall that, H_f denotes the Nisan's matrix corresponding to a noncommutative polynomial f . Observe that, $H_k(f \circ g) = H_k(f) \circ H_k(g)$.

It is known that in the noncommutative domain, computing the Hadamard product of two polynomials is easy when the polynomials are computable by ABPs.

Theorem 5. [AJS09] *Given a noncommutative ABP of size S' for a degree k polynomial $f \in \mathbb{F}\langle X \rangle$ and a noncommutative ABP of size S for another degree k polynomial $g \in \mathbb{F}\langle X \rangle$, we can compute a noncommutative ABP of size SS' for $f \circ g$ in deterministic $SS' \cdot \text{poly}(n, k)$ -time.*

Let C be a circuit and B an ABP computing homogeneous degree- k polynomials $f, g \in \mathbb{F}\langle X \rangle$ respectively. Then their Hadamard product $f \circ g$ has a noncommutative circuit of polynomially bounded size which can be computed efficiently [AJS09, AS18].

Theorem 6. [AS18] Corollary 4] *Given a noncommutative ABP of size s_1 computing a degree- d polynomial $g \in \mathbb{F}\langle Y \rangle$ and another degree- d polynomial $f \in \mathbb{F}\langle Y \rangle$ by an arithmetic circuit of size s_2 we can compute an arithmetic circuit of size $O(s_1^3 \cdot s_2)$ for $f \circ g$ in time polynomial in s_1, s_2 and d .*

Furthermore, if C is given by black-box access then $f \circ g(a_1, \dots, a_n)$ for $a_i \in \mathbb{F}, 1 \leq i \leq n$ can be evaluated by evaluating C on matrices defined by the ABP B [AS18] as follows: For each $i \in [n]$, the transition matrix $M_i \in \text{Mat}_s(\mathbb{F})$ are computed from the noncommutative ABP B (which is of size s) that encode layers. We define $M_i[k, \ell] = [x_i]L_{k, \ell}$, where $L_{k, \ell}$ is the linear form on the edge (k, ℓ) . Now to compute $(f \circ g)(a_1, a_2, \dots, a_n)$ where $a_i \in \mathbb{F}$ for each $1 \leq i \leq n$, we compute $C(a_1M_1, a_2M_2, \dots, a_nM_n)$. The value $(f \circ g)(a_1, a_2, \dots, a_n)$ is the $(1, s)^{\text{th}}$ entry of the matrix $f(a_1M_1, a_2M_2, \dots, a_nM_n)$.

Theorem 7. [AS18] *Given a circuit C and an ABP B computing homogeneous noncommutative polynomials f and g in $\mathbb{F}\langle Y \rangle$, the Hadamard product $f \circ g$ can be evaluated at any point $(a_1, \dots, a_n) \in \mathbb{F}^n$ by evaluating $C(a_1M_1, \dots, a_nM_n)$ where M_1, \dots, M_n are the transition matrices of B , and the dimension of each M_i is the size of B .*

Connecting Noncommutative Algebraic Complexity to Algebraic Automata Theory

In this section, we explore an interesting connection between noncommutative algebraic complexity and algebraic automata theory [\[AMS10\]](#). Throughout this thesis, we use this connection as a fundamental tool to obtain new algorithmic results.

Background on Algebraic Automata Theory

We recall some basic algebraic automata theory. More details can be found in the book of Berstel and Reutenauer [\[BR11\]](#).

Let K be a semiring and X be an alphabet³. A K -weighted automaton over X is a 4-tuple, $\mathcal{A} = (Q, I, E, T)$, where Q is a finite set of states, and the mappings $I, T : Q \rightarrow K$ are weight functions for entering and leaving a state respectively, and $E : Q \times X \times Q \rightarrow K$ is the weight of each transition. We define $|Q|$, the number of states, to be the size of the automaton. A path is a sequence of edges :

$(q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{t-1}, a_t, q_t)$. The weight of the path is the product of the weights of the edges. The formal series $S \in K\langle\langle X \rangle\rangle$ which is the (possibly infinite) sum of the weights over all the paths that are *recognized* by \mathcal{A} . Then, for each word $w = a_1 a_2 \dots a_t \in X^*$, the contribution of all the paths for the word w is given by $[w]S = \sum_{q_0, \dots, q_t \in Q} I(q_0) \cdot E(q_0, a_1, q_1) \dots E(q_{t-1}, a_t, q_t) \cdot T(q_t)$.

A K -weighted automaton \mathcal{A} with ϵ -*transitions* over X is defined with E modified, such that $E : Q \times \{X \cup \epsilon\} \times Q \rightarrow K$. Let $A_0 \in \text{Mat}_{|Q|}(K)$ be the transition matrix for the ϵ -transitions. It is well-known that if $\sum_k A_0^k$ converges, then another automaton \mathcal{A}' without ϵ -transitions computing the same series can be

³We interchangeably use X as a variable set of ABPs and as alphabet symbol of weighted automata.

constructed [LS12]. The automaton is said to be *valid* if $\sum_k A_0^k$ converges.

The following basic result by Schützenberger [Sch61] is the key to transforming zeroness testing of weighted automata up to a finite length only. For the proof, see [Eil74, Corollary 8.3].

Theorem 8 (Schützenberger). *Let K be a subring of a division ring and \mathcal{A} be a K -weighted automaton without any ϵ -transition with s states computing a series S in $K\langle\langle X \rangle\rangle$. Then S is a nonzero series if and only if there is a word $w \in X^*$ of length at most $s - 1$, such that $w \in \text{supp}(S)$.*

Running Automaton over a Noncommutative Polynomial

Let $f \in \mathbb{F}\langle X \rangle$ be a polynomial of degree bounded by d . We consider monomials in variables X as strings over the alphabet X . Let $\mathcal{A} = (Q, X, \delta, q_0, q_f)$ be a finite automaton over the alphabet X of size s recognizing a language $L : X^* \rightarrow \{0, 1\}$. Define $M_i \in \text{Mat}_s(\{0, 1\})$ to be the transition matrix corresponding to x_i in \mathcal{A} i.e. for each $q, q' \in [s]$, $M_i[q, q'] = 1$ if $\delta(q, x_i) = q'$ otherwise $M_i[q, q'] = 0$. We are interested in the output matrix obtained when each input x_i to the polynomial f is replaced by the matrix M_i . The output matrix of f on automaton \mathcal{A} , denoted M_{out} , is the matrix $f(M_1, \dots, M_n)$. Suppose $f(x_1, \dots, x_n) = \alpha x_{j_1} \cdots x_{j_k}$, with a non-zero coefficient $\alpha \in \mathbb{F}$. Clearly,

$$M_{out} = \alpha M_{j_1} \cdots M_{j_k} = \alpha M_w,$$

where $w = x_{j_1} \cdots x_{j_k}$. Thus, the matrix entry $M_{out}(q_0, q_f)$ is 0 when \mathcal{A} rejects w , and α when \mathcal{A} accepts w . In general, if $f = \sum_{w \in X^*} \alpha_w w$, then

$$M_{out}(q_0, q_f) = \sum_{w:L(w)=1} \alpha_w.$$

We can generalize this for a *substitution automaton* \mathcal{A} recognizing a language $L : X^* \rightarrow R$ for some commutative ring R . For each transition $\delta(q, x_i) = q'$, it substitutes some $a \in R$ and the corresponding transition matrix $M_i[q, q'] = a$. Now, for any noncommutative polynomial $f = \sum_{w \in X^*} \alpha_w w$, we obtain,

$$M_{out}(q_0, q_f) = \sum_{w: L(w) \neq 0} \alpha_w \cdot L(w).$$

Let $f \in \mathbb{F}\langle x_1, x_2, \dots, x_n \rangle$ be a noncommutative polynomial and let $M_i \in \text{Mat}_t(F)$ be a matrix substitution for the variables $x_i, 1 \leq i \leq n$.

It is useful to interpret the evaluation of $f(M_1, M_2, \dots, M_n)$ as the run of a t -state (nondeterministic) substitution automaton \mathcal{A} by treating M_i as its transition matrices on input variables x_i . More precisely, we can interpret the $(i, j)^{th}$ entry of the matrix $f(M_1, M_2, \dots, M_n)$ as follows:

- i is treated as the start state of \mathcal{A} and j as the final state.
- The contribution of each nonzero monomial $x_{i_1} x_{i_2} \cdots x_{i_r}$ to the $(i, j)^{th}$ entry of the matrix product $M_{i_1} M_{i_2} \cdots M_{i_r}$ is obtained as a sum of products over all i to j transition paths of the form

$$\alpha \cdot M_{i_1}[i, j_1] M_{i_2}[j_1, j_2] \cdots M_{i_r}[j_{r-1}, j],$$

where $\alpha \in \mathbb{F}$ is the coefficient of the monomial in f .

- These products correspond precisely to the nondeterministic i to j paths of the automaton on input $x_{i_1} x_{i_2} \cdots x_{i_r}$.
- Finally, the $(i, j)^{th}$ entry of the matrix $f(M_1, M_2, \dots, M_n)$ is a linear combination over all monomials occurring in f .

This automata-theoretic interpretation has proven useful in designing PIT algorithms for noncommutative polynomials in various settings [AMS10, AJMR17].

Rational Identity Testing

Arithmetic circuit complexity is mainly concerned with the computation of polynomials and rational functions using basic arithmetic operations: additions, multiplications, and inverses. Inverse gates can be efficiently eliminated in commutative circuits [Str73], but their role in the noncommutative computation is more complicated.

The study of noncommutative rational expressions is a classical subject [Ami66] and their computational aspects are of interest in algebraic automata theory. Formally, these are the elements of the universal skew-field of fractions $\mathbb{F}\langle X \rangle$ where $X = (x_1, \dots, x_n)$, and x_1, x_2, \dots, x_n are n free noncommuting variables and \mathbb{F} is a scalar field. Two rational expressions r and s are defined *equivalent* if they agree on any $d \times d$ tuple of matrices for all d whenever both are defined. The elements of the universal free skew-field consist of rational expressions modulo this equivalence relation [Ami66]. More recently, Hrubeš and Wigderson [HW15] initiated a complexity-theoretic study of noncommutative rational expressions. The complexity of a rational expression is defined as the size of an efficient representation of that expression such as a rational circuit, a rational formula, etc. In [HW15], the authors initiated the algorithmic study of *rational identity testing* (RIT): determine if a given noncommutative rational expression computes the zero function in the free skew-field. For example, the rational expression $(x + xy^{-1}x)^{-1} + (x + y)^{-1} - x^{-1}$, known as Hua's identity [Hua49], is zero in the free skew-field.

The linear pencil model for computing rational expressions has played an important role in designing RIT algorithms. A *linear pencil* L of size s over x variables is a

$s \times s$ matrix whose entries are linear forms in x . That is, $L = A_0 + \sum_{i=1}^n A_i x_i$, where each A_i is an $s \times s$ matrix over \mathbb{F} . A rational function r in $\mathbb{F}\langle X \rangle$ has a *linear pencil representation* L of size s , if for some $i, j \in [s]$, $r = (L^{-1})_{i,j}$ where the inverse of the matrix L is defined in a precise sense [HW15]. It turns out that RIT for linear pencils can be efficiently reduced to the noncommutative singularity problem Singular [HW15]. Since Singular has a deterministic polynomial-time algorithm in the white-box model [GGOW16, IQS18] and has a randomized polynomial-time algorithm in the black-box model [DM17], we obtain, as consequence, RIT algorithms for rational expressions with small linear pencils. The RIT algorithm for noncommutative rational formulas follows from the fact that an rational formula has a small linear pencil representation which can be efficiently computed [HW15].

Chapter 3

Image of Noncommutative Polynomials and Applications to Rational Identity Testing

In this chapter, we focus on algorithmic questions concerning the image of noncommutative polynomials. Any noncommutative polynomial $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ defines a map from n -tuples of $d \times d$ matrices over \mathbb{F} to $d \times d$ matrices, for every dimension d . As we have seen in Chapter 2, identity testing of noncommutative polynomials is closely related to the complexity of polynomial identities. A polynomial identity is a noncommutative polynomial whose image set contains only the zero matrices for any dimension d . In this chapter, we explore the invertibility and trace of the image set of noncommutative polynomials. The image of noncommutative polynomials has been explored in mathematics [Šp12] [BK08]. For example, Brešar and Klep [BK08] have shown that the span of the image of any noncommutative polynomials can have only four possibilities: either zero, scalar multiple of the identity matrix or trace zero, or full matrix algebra. Our contribution is to address the algorithmic questions associated with these problems

and show some interesting applications. We also study the Rational Identity Testing (RIT) problem for a generalization of ABPs, where we allow a sum of linear forms and inverses of linear forms as edge labels. In order to obtain a deterministic black-box RIT in this model, we apply some ideas from algebraic automata theory.

Let $X = (x_1, x_2, \dots, x_n)$ be a set of n free noncommuting variables and \mathbb{F} be any scalar field. The *free noncommutative ring* $\mathbb{F}\langle X \rangle$ is the ring of all noncommutative polynomials in X -variables over the field \mathbb{F} .

Let us first formally define the image set of noncommutative polynomials.

Definition 9 (Image Set of a Noncommutative Polynomial). *For the matrix algebra $\text{Mat}_k(\mathbb{F})$ for some integer k , the image set of a noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$ is defined as the set*

$$\text{Im}_k(f) = \{f(M_1, \dots, M_n) \mid (M_1, \dots, M_n) \in \text{Mat}_k^n(\mathbb{F})\}.$$

3.1 Invertible Image of Noncommutative ABPs

In this section, we study the invertibility of an image set of a noncommutative polynomial. More precisely, we study the following algorithmic problem: Given a noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$, find an invertible matrix in the image set $\text{Im}_k(f)$ of f for any k , if it exists. Notice that, this problem is harder than the PIT problem. If we are only interested to know the existence of an invertible matrix in the image set, this problem reduces to the PIT problem following Amitsur's theorem on the universal division algebra [\[Row80\]](#), [\[LZ09\]](#).

Amitsur's theorem on universal division algebra

Recall from Chapter 2 that R_d is the ring of $d \times d$ generic matrices. The central quotient of R_d is denoted as $\text{UD}(d)$. Amitsur showed that $\text{UD}(d)$ is a division ring, also known as the universal division algebra of degree d [Ami66]. It follows from Amitsur's theorem that any non-zero noncommutative polynomial (even rational expression) evaluated on $d \times d$ generic matrices for some large enough d must be invertible. In other words, if a noncommutative polynomial f is not an identity of $\text{Mat}_d(\mathbb{F})$ then $\text{Im}_d(f)$ contains an invertible matrix for large enough d . However, it only argues the existence of such an invertible matrix in the image set and a randomized polynomial-time algorithm follows. However, finding an invertible image in $\text{Im}_d(f)$ in deterministic sub-exponential-time (even if we know it exists) remains open.

We solve this problem when the input noncommutative polynomial is computable by a noncommutative ABP. Recall from Chapter 2 that for this class we already have a deterministic polynomial-time white-box PIT algorithm and a deterministic quasi-polynomial-time black-box PIT algorithm. We show that we can also find an invertible matrix in the image set in a similar running time.

Theorem 9. *Given black-box access to a noncommutative polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d , computable by an ABP A of size at most s , there is a deterministic quasi-polynomial-time algorithm of running time $(snd)^{O(\log d)}$ that computes a matrix tuple $(M_1, \dots, M_n) \in \text{Mat}_d^n(\mathbb{F})$ of $d \times d$ matrices such that $f(M_1, \dots, M_n)$ is invertible where $|\mathbb{F}| > \text{poly}(d, n, s)$. In white-box, we can compute a matrix tuple $(M_1, \dots, M_n) \in \text{Mat}_d^n(\mathbb{F})$ of $d \times d$ matrices such that $f(M_1, \dots, M_n)$ is invertible where $|\mathbb{F}| > \text{poly}(d, n, s)$ in deterministic $\text{poly}(s, n, d)$ time.*

Proof. The idea is to reduce the problem of finding an invertible matrix in the image set of the input polynomial to solving PIT of product commutative ROABPs.

Let us first fix some notations. Suppose S_d denotes the set of permutations $\{\sigma : [d] \rightarrow [d]\}$. For a degree- d word $m \in X^d$, let $m = x_{i_1} \cdots x_{i_d}$. We define σ -permuted word $m^\sigma = x_{i_{\sigma(1)}} \cdots x_{i_{\sigma(d)}}$. For a degree- d homogeneous noncommutative polynomial $g \in \mathbb{F}\langle X \rangle$, g^σ is defined as $g^\sigma = \sum_{m \in \text{supp}(g)} [m]g \cdot m^\sigma$. For each $j \in \{0, 1, \dots, d-1\}$, $\sigma_j \in S_d$ denotes the permutation that cyclically rotates a monomial right to left by j steps. As a permutation,

$$\sigma_j = (j+1, j+2, \dots, d, 1, \dots, j).$$

As preparation, we show in the following lemma that each *cyclic shift* of a noncommutative homogeneous ABP of size s can be computed by an ABP of size polynomial in s .

Lemma 1. *Let A be a homogeneous ABP of size s computing a noncommutative polynomial $g \in \mathbb{F}\langle X \rangle$ of degree d . For each $j \in [d-1]$, there is a $O(s^2)$ size ABP computing g^{σ_j} .*

Proof. Let $w \leq s$ be the width of the ABP. We can write $g = \sum_{i=1}^w p_i q_i$, where for $i \in [w]$, p_i is the polynomial of degree j computed from the source to the i^{th} node of layer j and q_i is the polynomial of degree $d-j$ computed from the i^{th} node of layer j to the sink. It is easy to see that $g^{\sigma_j} = \sum_{i=1}^w q_i p_i$. For each $p_i q_i$, one can extract out an ABP of size at most s from the ABP for g by surgery. Notice that, we can obtain an ABP of size s for each $q_i p_i$ by rearranging the layers. Therefore, putting them together to obtain $g^{\sigma_j} = \sum_{i=1}^w q_i p_i$ gives us an ABP of size at most $ws = O(s^2)$. \square

We first explain the proof for homogenous degree d ABPs. For each $i \in [n]$ construct the following matrices:

$$(3.1) \quad M_i = \begin{bmatrix} 0 & y_{i,1} & 0 & \cdots & 0 \\ 0 & 0 & y_{i,2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y_{i,d-1} \\ y_{i,d} & 0 & 0 & \cdots & 0 \end{bmatrix}_{d \times d}.$$

It is possible to view these matrices as the transition matrices of a labeled automaton. We observe that for a monomial $m = x_{i_1}x_{i_2} \cdots x_{i_d}$, the matrix $m(M_1, \dots, M_n)$ is a diagonal matrix. Moreover, for any $j \in [d]$, the $(j, j)^{th}$ entry is given by $(y_{i_1, j} \cdots y_{i_{d-(j-1)}, d})(y_{i_{d-(j-2)}, 1} \cdots y_{i_d, j-1})$. Since $Y = \{y_{i, j}\}$ is a set of commutative variables, the above is same as $\text{SM}(m^{\sigma_{d-(j-1)}})$ ^[1].

Thus, for any homogeneous degree d polynomial f , by linearity we get that $f(M_1, \dots, M_n)$ is also a diagonal matrix and the $(j, j)^{th}$ entry is $\text{SM}(f^{\sigma_{d-(j-1)}})$.

The image of the polynomial f is invertible on a point (M_1, \dots, M_n) , if and only if $\det(f(M_1, \dots, M_n)) \neq 0$. Further if the shape of each M_i is as described in Equation [3.1](#), we have,

$$\det(f(M_1, \dots, M_n)) = \prod_{j=0}^{d-1} \text{SM}(f^{\sigma_j}).$$

Note that, if the noncommutative polynomial f is nonzero then for each $\sigma_j \in S_d$, f^{σ_j} is also nonzero. Recall that, for any f , f is nonzero if and only if $\text{SM}(f)$ is nonzero. Hence, given a nonzero polynomial f , $\det(f(M_1, \dots, M_n))$ is a nonzero polynomial as every diagonal entry evaluates to a nonzero commutative polynomial. Since f^{σ_0} has an ABP of size s , each cyclic shift f^{σ_j} has an ABP of size $O(s^2)$ by

¹For any homogeneous degree- d noncommutative polynomial $f = \sum_{m \in X^d} [m]fm$, recall that $\text{SM}(f) = \sum_{m \in X^d} [m]f\text{SM}(m)$ where if $m = x_{i_1} \cdots x_{i_d}$ then $\text{SM}(m) = y_{i_1, 1} \cdots y_{i_d, d}$ where Y is a set of commuting variables.

Lemma [1](#). Therefore, the set-multilinearization $\text{SM}(f^{\sigma_0})$ has an ABP of size s , and each $\text{SM}(f^{\sigma_j})$ has an ABP of size $O(s^2)$, over the same variable partition $Y = Y_1 \sqcup Y_2 \sqcup \dots \sqcup Y_d$. It is obtained by making the input ABP set-multilinear i.e. by replacing each x_i variable in the j^{th} layer by $y_{i,j}$.

Let $Z = \{z_1, \dots, z_d\}$ be a set of commuting variables. We can now replace each $y_{i,j}$ by z_j^i . As discussed in Chapter [2](#), by replacing $y_{i,j}$ by z_j^i in a set-multilinear ABP over the variable partition $Y = Y_1 \sqcup Y_2 \sqcup \dots \sqcup Y_d$ yields an ROABP with the variable partition $z_1 < \dots < z_d$. Therefore, $\det(f(M_1, \dots, M_n))$ can be expressed as a product of commutative ROABPs each of size $O(s^2)$ and over the same variable partition. Let h_k be the ROABP by replacing $y_{i,j}$ by z_j^i in $\text{SM}(f^{\sigma_k})$. Therefore we may write

$$(3.2) \quad \det(f(M_1, \dots, M_n)) = \prod_{k=0}^{d-1} h_k(z_1, \dots, z_d).$$

Now we briefly discuss how to use a generator of Forbes-Shpilka [FS13](#) to complete the algorithm. Let $\mathcal{G} : \mathbb{F}^m \mapsto \mathbb{F}^d$ be the hitting set generator for the commutative ROABPs of size $O(s^2)$ over the variable partition $z_1 < \dots < z_d$ with d layers where $m = O(\log d)$ as promised by the result in [FS13](#) (see Chapter [2](#) for more details). The map \mathcal{G} is a polynomial map where each z_i is substituted by a polynomial $p_i(\alpha_1, \dots, \alpha_m)$ of degree at most $D = dns^8$ with the property that each $h_k \circ \mathcal{G}$ is non-zero if and only if $h_k \neq 0$. Thus, to prove that $f(M_1, \dots, M_n)$ is invertible, it suffices to show,

$$(3.3) \quad \prod_{k=0}^{d-1} (h_k \circ \mathcal{G}) = \left(\prod_{k=0}^{d-1} h_k \right) \circ \mathcal{G} = \det(f(M_1, \dots, M_n)) \circ \mathcal{G} \neq 0.$$

Now we note that $\det(f(M_1, \dots, M_n)) \circ \mathcal{G}(z)$ is a m -variate polynomial of degree at

most d^2D . Thus to test Equation [3.1](#) for identity, it suffices to go over $(d^2D)^m$ distinct values of Z variables.

Invertible image in white-box

We have shown that finding an invertible matrix in the image set of a noncommutative ABP can be reduced to the PIT of a product of ROABPs (see Equation [3.2](#)). It is known that the white-box PIT of a ROABP is reducible to identity testing of a low-degree univariate (for more details, see [For14](#) Section 6.3)]. Now the algorithm to find an invertible image of a noncommutative ABP follows from the deterministic $\text{poly}(s, n, d)$ -time white-box PIT algorithm for a product of ROABPs.

Invertible image of inhomogeneous polynomial

In case, the given ABP of degree d is not homogeneous, then the substitution $x_i = tM_i$ is performed where t is a commutative variable. The words (monomials) of degree- i produce terms with t -degree i . Now $\det(f(tM_1, tM_2, \dots, tM_n))$ will have a term with t -degree d^2 which is produced by the identity permutation and no other permutations can produce a term of same t -degree². Thus using Forbes-Shpilka generator \mathcal{G} , we know that for some $\alpha = (\alpha_1, \dots, \alpha_m)$, the polynomial $\det(f(tM_1, tM_2, \dots, tM_n)) \circ \mathcal{G}|_{\alpha}$ is a nonzero polynomial in t of degree d^2 and hence it suffices to try the $d^2 + 1$ distinct substitution for t such that the final output becomes nonzero on one such substitution $t = \beta$. □

²This can be easily seen by observing that the non-diagonal entries of the output matrix contain the terms with t degree $\leq d - 1$.

3.2 Trace of Image of Noncommutative ABPs

We next turn to another algorithmic question related to the image set of noncommutative polynomials motivated by the following interesting theorem due to Brešar and Klep [BK08].

The Brešar-Klep Theorem

It categorizes the span of the image set of noncommutative polynomials.

Theorem 10 (Brešar-Klep Theorem [BK08]). *Let $f \in \mathbb{F}\langle X \rangle$ be any noncommutative polynomial, where \mathbb{F} is any field of zero characteristic. Then precisely one of the following is true:*

1. $\text{Im}_k(f) = 0$, which means f is an identity for $\text{Mat}_k(\mathbb{F})$.
2. The span of $\text{Im}_k(f)$ consists of all scalar multiples of the identity matrix I_k (i.e., f is central for $\text{Mat}_k(\mathbb{F})$).
3. The span of $\text{Im}_k(f)$ is all trace zero matrices over $\text{Mat}_k(\mathbb{F})$.
4. The span of $\text{Im}_k(f)$ is $\text{Mat}_k(\mathbb{F})$, the full matrix algebra.

The Brešar-Klep theorem naturally raises an algorithmic question: Given a noncommutative polynomial f and the matrix algebra $\text{Mat}_k(\mathbb{F})$, to efficiently determine which of the four cases occur.

Proposition 1. *Let $f \in \mathbb{Q}\langle X \rangle$ be a noncommutative polynomial of degree d over rationals given by an arithmetic circuit of size s . For any matrix algebra $\text{Mat}_k(\mathbb{Q})$ we can check in randomized $\text{poly}(s, d, k)$ -time which of the four conditions of the Brešar-Klep theorem hold for f over $\text{Mat}_k(\mathbb{Q})$.*

This is easily observed by substituting the noncommuting variables with generic $k \times k$ size matrices and evaluating the commuting generic variables randomly. We show the following result which yields an efficient deterministic algorithm.

Theorem 11. *Given a noncommutative ABP A of size s computing a polynomial $f \in \mathbb{F}\langle X \rangle$ of degree d , there is a deterministic $\text{poly}(n, s, d)$ -time algorithm to check if $\text{Im}_k(f)$ is trace zero over $\text{Mat}_k(\mathbb{F})$ for all $k \geq d$. If A is given by black-box access, there is a deterministic $(ns)^{O(\log d)}$ -time algorithm to check if $\text{Im}_k(f)$ is trace zero for all $k \geq d$ where $|\mathbb{F}| \geq \text{poly}(n, s, d)$.*

The above theorem yields a deterministic polynomial-time algorithm to check which of the four conditions of the Brešar-Klep theorem holds for matrix algebras of dimension $k \geq d$ for a noncommutative polynomial f given by an ABP.

We now prove Theorem [11](#). By the following lemma, it suffices to show it for homogeneous ABPs. $\text{Trace}(\text{Im}_k(f))$ is the set of traces of the matrices in $\text{Im}_k(f)$. Use \underline{M} to denote the matrix tuple (M_1, \dots, M_n) . We say $\text{Trace}(\text{Im}_k(f)) = \{0\}$ if and only if $\text{Trace}(f(\underline{M})) = 0$ for each $\underline{M} \in \text{Mat}_k^n(\mathbb{F})$.

Lemma 2. *Let $f \in \mathbb{F}\langle X \rangle$ be a noncommutative polynomial of degree d , and f_i be its homogeneous degree- i component for each $i \in \{0, 1, \dots, d\}$. Then for all $k \in \mathbb{N}$, $\text{Trace}(\text{Im}_k(f)) = \{0\}$ if and only if $\text{Trace}(\text{Im}_k(f_i)) = \{0\}$ for each i .*

Proof. Consider the substitution $x_i \mapsto z \cdot x_i$ for a commuting variable z . We can now write,

$$\text{Trace}(f(zM_1, \dots, zM_n)) = \text{Trace} \left(\sum_{i=1}^d f_i(\underline{M})z^i \right) = \sum_{i=1}^d \text{Trace}(f_i(\underline{M}))z^i.$$

Now if $\text{Trace}(\text{Im}_k(f_i)) = \{0\}$ for each $i \in [d]$, then clearly $\text{Trace}(\text{Im}_k(f)) = \{0\}$. Now suppose $\text{Trace}(\text{Im}_k(f_i)) \neq \{0\}$ for some i . Then there is a matrix tuple $\underline{M} = (M_1, \dots, M_n)$ such that $\text{Trace}(f(zM_1, \dots, zM_n))$ is a nonzero univariate in z .

Hence, there is a substitution $z = \alpha$ for which $\text{Trace}(f(\alpha M_1, \dots, \alpha M_n)) \neq 0$ which shows that $\text{Trace}(\text{Im}_k(f)) \neq \{0\}$. \square

Proof of Theorem 11. By Lemma 2 and the fact that homogeneous components can be extracted efficiently both in the black-box and in the white-box setting, we can assume the given ABP is homogeneous of degree d w.l.o.g. First, we prove that if $\text{Im}_k(f)$ is trace zero for some $k \geq d$, then the coefficients of f have the following symmetry property.

Lemma 3. *For a homogeneous polynomial $g \in \mathbb{F}\langle X \rangle$ of degree d ,*

$\text{Trace}(\text{Im}_k(g)) = \{0\}$ at any dimension $k \geq d$, if and only if for every monomial m we have $\sum_{\sigma \in C_d} [m^\sigma]g = 0$, where $C_d = \{\sigma_0, \sigma_1, \dots, \sigma_{d-1}\}$ is the set of all d cyclic shift permutations.

Proof. Let M_d be the set of all monomials in g . Let M'_d denote a maximal subset of M_d constructed as follows: Group the monomials in M_d such that monomials in the same group are cyclic shifts of each other. Now define M'_d by taking one monomial from each such group. For any matrix tuple \underline{M} , by the cyclic property of trace, for each $\sigma \in C_d$, $\text{Trace}(m^\sigma(\underline{M}))$ is same. Hence, we may write,

$$\text{Trace}(g(\underline{M})) = \sum_{m \in M_d} [m]g \cdot \text{Trace}(m(\underline{M})) = \sum_{m \in M'_d} \left(\sum_{\sigma \in C_d} [m^\sigma]g \right) \cdot \text{Trace}(m(\underline{M})).$$

If for every monomial m we have $\sum_{\sigma \in C_d} [m^\sigma]g = 0$ then $\text{Trace}(\text{Im}_k(g)) = \{0\}$ for each k .

For the converse direction, suppose there is a monomial m such that

$\sum_{\sigma \in C_d} [m^\sigma]g \neq 0$. Let $m = x_{i_1}x_{i_2} \dots x_{i_d}$. We now construct an automaton that accepts only those degree- d words which are the cyclic shifts of m . Below, we give an illustrative example for $d = 6$.

The permutation σ_0 is the identity permutation. When the start state and final state

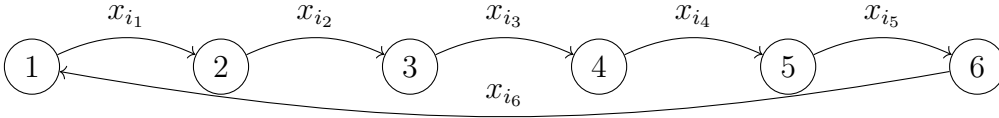


Figure 3.1: Example of the automata when $d = 6$.

are both $j \in [d]$, then only the word $m^{\sigma_{j-1}} = x_{i_j}x_{i_{j+1}} \cdots x_{i_d}x_{i_1}x_{i_2} \cdots x_{i_{j-1}}$ is accepted (also note that if start state and final state are different then no word of length d is accepted). The transition of the automata gives us $d \times d$ matrices $M_{x_{i_1}}, \dots, M_{x_{i_d}}$ where $M_{x_{i_j}}(k, \ell) = 1$ if $k = j$ and $\ell = j + 1 \pmod{d}$ and 0 otherwise. Substituting x_{i_j} by $M_{x_{i_j}}$ and setting $M_{x_t} = [0]$ if $x_t \notin \{x_{i_1}, x_{i_2}, \dots, x_{i_d}\}$, we observe that the matrix $g(M_{x_1}, \dots, M_{x_n})$ is a diagonal matrix and the $(j, j)^{th}$ entry is $[m^{\sigma_{j-1}}]g$,

$$g(M_{x_1}, M_{x_2}, \dots, M_{x_n}) = \begin{bmatrix} [m^{\sigma_0}]g & 0 & \cdots & 0 \\ 0 & [m^{\sigma_1}]g & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & [m^{\sigma_{d-1}}]g \end{bmatrix},$$

and thus $\text{Trace}(g(M_{x_1}, \dots, M_{x_n})) = \sum_{\sigma \in C_d} [m^\sigma]g \neq 0$. \square

Remark 2. Notice that, Lemma [3](#) holds only when $k \geq d$. One could improve the dependency by constructing an automaton of smaller size accepting a word and all its cyclic shifts only. However, it is not clear to us how to improve this construction.

As a corollary of Lemma [3](#) we obtain the following.

Corollary 1. For all matrix substitution of dimension $k \geq d$, $\text{Trace}(\text{Im}_k(g)) = \{0\}$ if and only if $\sum_{\sigma \in C_d} g^\sigma \equiv 0$.

Proof. Observe that $g^\sigma = \sum_m [m]g \cdot m^\sigma$. Hence,

$$\sum_{\sigma \in C_d} g^\sigma = \sum_{\sigma \in C_d} \sum_m [m]g \cdot m^\sigma = \sum_m \left(\sum_{\sigma \in C_d} [m^\sigma]g \right) \cdot m.$$

Then the proof follows from Lemma 3. □

Now we would like to check if the input polynomial f computed by the given ABP has the above property. It turns out that if the input polynomial f is given as a white-box, a combination of Lemma 1 and Theorem 2 can easily yield a deterministic polynomial-time algorithm.

The White-Box Case

By Lemma 1 we see that, for each $\sigma \in C_d$, f^σ can be computed by an algebraic branching program of size $O(s^2)$ and hence $\hat{f} = \sum_{\sigma \in C_d} f^\sigma$ can be computed by an algebraic branching program of size $\text{poly}(s, d)$. Given the algebraic branching program A the algorithm computes the algebraic branching program $\hat{A} = \sum_{\sigma \in C_d} f^\sigma$ using Lemma 1 and runs the algorithm of Raz and Shpilka [RS05] on the ABP \hat{A} and outputs *trace zero* if $\hat{A} \equiv 0$. The correctness of the algorithm follows from Lemma 3, and the run time of the algorithm is $\text{poly}(n, s, d)$.

The Black-Box Case

The main idea is to obtain black-box access to $\text{SM}(\hat{f})$ where $\hat{f} = \sum_{\sigma \in C_d} f^\sigma$ (following notation of Corollary 1). Thereafter, one can use the standard hitting set [FS13] for set-multilinear ABPs over the variable partition $Y = Y_1 \sqcup Y_2 \sqcup \dots \sqcup Y_d$. Now, for each $i \in [n]$, we construct the $d \times d$ matrix M_i as shown in the proof of Theorem 9 (If $k > d$, we adjust each M_i by padding zeros).

Lemma 4. $\text{Trace}(f(M_1, \dots, M_n)) = \text{SM}(\hat{f})$.

The proof of the lemma follows quite easily.

Using Lemma 1, the ABP size of \hat{f} is at most $\text{poly}(s, d)$. Also we conclude that $\text{SM}(\hat{f})$ has a set-multilinear ABP of depth d in the variable partition

$Y_1 \sqcup Y_2 \sqcup \cdots \sqcup Y_d$ of size at most $\text{poly}(s, d)$. Now the algorithm substitutes $y_{i,j}$ from the hitting set of the set-multilinear ABPs of size s over the variable partition $Y_1 \sqcup Y_2 \sqcup \cdots \sqcup Y_d$ with d many layers [ES13] and evaluates the polynomial on the matrices M_i and checks whether the trace of the output matrix is always zero or not. The correctness follows from Corollary 1 and the running time follows from Theorem 4 when applied to the set-multilinear case.

Corollary 2. *Let f be a degree- d noncommutative polynomial in $\mathbb{F}\langle X \rangle$ computed by a size s ABP. For $k \geq d$, when f is given by an ABP (the white-box case) we can check in deterministic polynomial time which of the four cases of the Brešar-Klep theorem holds for f . For $k \geq d$, when f is given only by black-box access, we can check all the possibilities in deterministic quasi-polynomial-time.*

Proof. If $k \geq d$, by the Amitsur-Levitzki theorem a nonzero f is not an identity for $\text{Mat}_k(\mathbb{F})$. To rule out the second case notice that if f is a *central polynomial* for $\text{Mat}_k(\mathbb{F})$ then $g = zf - fz$ is an identity for $\text{Mat}_k(\mathbb{F})$ where z is a new noncommutative variable. This is also not possible by Amitsur-Levitzki theorem as degree of $zf - fz$ is $d + 1$ and as a nonzero polynomial it cannot vanish on $\text{Mat}_k(\mathbb{F})$ as $k \geq (d + 1)/2 + 1$. If $\text{Im}_k(f)$ is trace zero over $\text{Mat}_k(\mathbb{F})$, then the span of the image of f can not be $\text{Mat}_k(\mathbb{F})$ which can be checked efficiently by Theorem 11. Otherwise, if $\text{Im}_k(f)$ is not trace zero over $\text{Mat}_k(\mathbb{F})$, its span must be the entire algebra $\text{Mat}_k(\mathbb{F})$ as promised by the Brešar-Klep theorem. \square

3.3 Special Instances of Rational Identity

Testing

In general, a noncommutative rational expression of inversion height one can be obtained as the composition of a noncommutative polynomial with inverses of

noncommutative polynomials (following Bergman’s definition [Ber76]). This naturally leads us to consider an easier case of rational identity testing for a generalization of algebraic branching programs whose multi-edges are labeled by affine linear forms or inverses of affine linear forms. Clearly, such ABPs compute rational expressions of inversion height one in the free skew field. The rational expression computed by the ABP is the sum over each source-to-sink path P of the ordered product of affine linear forms or their inverses labeling P . The size of the ABP is defined as the total number of nodes and multi-edges. For this model, a deterministic quasi-polynomial time white-box algorithm and a randomized quasi-polynomial time black-box algorithm follow respectively from [GGOW16], [IQS18] and [DM17]. In this section, we obtain a deterministic polynomial time white-box algorithm and a deterministic quasi-polynomial time algorithm for the black-box RIT.

The *algebraic branching program* (ABP) we consider are layered directed acyclic graphs. Layer 0 has a single source node and layer $d + 1$ has a single sink node. The ABP has directed multi-edges from nodes in layer i to layer $i + 1$ (we allow multiple edges between the same pair of nodes), where each edge is labeled by some affine linear form or the inverse of an affine linear form. The size s of an ABP is the sum of the number of nodes and edges in it. The rational expression computed by the ABP is the sum over each source-to-sink path P of the ordered product of affine linear forms or their inverses labeling P .

Theorem 12. *Given black-box access to an ABP of size s where each edge is labeled by an affine linear form or inverse of an affine linear form over \mathbb{Q} , there is a deterministic $(ns)^{O(\log(ns))}$ -time algorithm to decide if the rational expression computed by it is zero in $\mathbb{Q}\langle X \rangle$. If such an ABP is given as a white-box then there is a deterministic $\text{poly}(s, n)$ -time algorithm for it.*

In this section, we prove Theorem [12]. In the generalized ABP model, we allow

directed multi-edges from nodes in layer i to layer $i + 1$ (we allow multiple edges between the same pair of nodes), where each edge is labeled by some affine linear form or the inverse of an affine linear form. Recall that, the size s is the total number of nodes and the multi-edges present in the ABP.

A simple fact about formal power series that we use is replacing the rational expression $(1 - x)^{-1}$ by the equivalent power series $\sum_{k \geq 0} x^k$, denoted as x^* which is used to convert an ABP where edges are labeled by linear forms and its inverses to an automaton computing a formal series. In general, an affine linear form may have zero constant terms. In order to apply the above, we require a linear shift $x_j \mapsto \alpha_j - x_j$, $j \in [n]$, enabling power series expansion of the inverses of the linear forms. The following lemma explains how to find such linear shifts efficiently.

Lemma 5. *Let \mathbb{F} be a field such that $|\mathbb{F}| \geq nr + 1$. We can efficiently construct a subset $S \subseteq \mathbb{F}^n$ of size $nr + 1$ such that for any r affine linear forms L_1, \dots, L_r over X , there is a point $\underline{\alpha} \in S$ such that for all i , $L_i(\underline{\alpha}) \neq 0$.*

Proof. By substituting x_i by y^i , we obtain the univariate polynomial $p(y) = L_1(y)L_2(y) \dots L_r(y)$. The degree of the polynomial p is bounded by nr . Hence, for any set T of $nr + 1$ distinct points from \mathbb{F} , there is a $t \in T$ such that $p(t) \neq 0$. The set S can be defined as $S = \{(t, t^2, \dots, t^n) : t \in T\}$. \square

The following lemma shows that rational identity testing of such ABPs is efficiently reducible to zero-testing of a weighted automaton computing a formal series in $\mathbb{F}\langle\langle X \rangle\rangle$.

Lemma 6. *Let A be a generalized ABP of size s with each edge labeled by either an affine linear form or the inverse of an affine linear form, computing a rational expression f in $\mathbb{F}\langle\langle X \rangle\rangle$. Let r be the total number of multi-edges of A . Then, there is an automaton A' without ϵ -transitions of size at most $s + r$ computing a formal*

series in $\mathbb{F}\langle\langle X \rangle\rangle$ such that f is an identity in $\mathbb{F}\langle\langle X \rangle\rangle$ if and only if A' computes a zero series in $\mathbb{F}\langle\langle X \rangle\rangle$. Moreover, A' can be constructed in $\text{poly}(n, s, r)$ time.

Proof. We present the proof in two parts. We first explain the automaton construction. Then, we show that this construction is identity preserving.

Construction of the Automaton

Let L_1, L_2, \dots, L_r be all the linear forms appearing as L_i or L_i^{-1} in A . By Lemma 5, we can efficiently compute a set $S \subseteq \mathbb{F}^n$ of size $nr + 1$ such that there exists a point $\underline{\alpha} \in S$ such that for each $i \in [r]$, $L_i(\alpha_1 - x_1, \dots, \alpha_n - x_n)$ has a nonzero constant term. Fix such a tuple $\underline{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n) \in S$. We apply the linear shift $x_j \mapsto \alpha_j - x_j$ to each edge label of A , and let g denote the rational expression in $\mathbb{F}\langle\langle X \rangle\rangle$ computed by the resulting ABP.

As each $L_i(\alpha_1 - x_1, \dots, \alpha_n - x_n)$ has a constant term, we may write it as $\beta_i(1 - \tilde{L}_i)$, for a homogeneous linear form \tilde{L}_i , where $\beta_i \neq 0$. We can convert $(1 - \tilde{L}_i)^{-1}$ to the formal power series \tilde{L}_i^* to obtain $L_i^{-1} = \beta_i^{-1}\tilde{L}_i^*$. Thus, any edge labeled L_i^{-1} can be labeled by a Kleene-* expression. From this observation, we now show that g can also be converted to a formal series in $\mathbb{F}\langle\langle X \rangle\rangle$ computed by a small automaton.

This is a standard adaptation of Kleene's original construction. We locally substitute each *-expression with a small automaton. We illustrate this with an example. Consider the edge shown in Figure 3.2 having linear form and inverses. In Figure 3.3, we convert the linear forms with inverses to *-expressions by the linear shift $x_j \mapsto 1 - x_j$. Finally, in Figure 3.4, we show the transitions of an equivalent automaton by replacing the *-rational expressions with their corresponding automata.

It is useful to consider the transition matrix M for the final automaton. In the

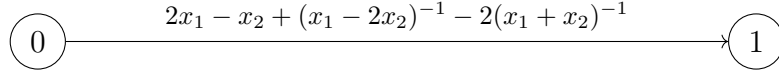


Figure 3.2: Edge Labels having linear form and inverses

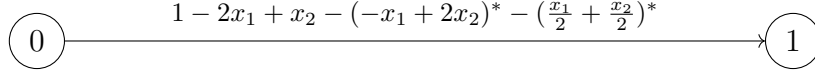


Figure 3.3: Edge Labels rewritten as *-rational expression after applying the shift $x_i \mapsto 1 - x_i$.

current example, this is given by the following matrix.

$$M = \begin{bmatrix} 0 & -1 & -1 & 1 - 2x_1 + x_2 \\ 0 & -x_1 + 2x_2 & 0 & 1 \\ 0 & 0 & \frac{x_1 + x_2}{2} & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Clearly, applying the above transformation to each edge of the input ABP A produces an automaton \tilde{A} of size at most $s + r$, because we introduce a new node in the automaton for each L^{-1} term. Moreover, \tilde{A} can be constructed in $\text{poly}(n, s, r)$ time.

Claim 1. \tilde{A} computes a valid formal series in $\mathbb{F}\langle\langle X \rangle\rangle$.

Proof of Claim. Consider the transition matrix of the automaton A_0 corresponding to the ϵ -transitions. To show that \tilde{A} computes a valid formal series in $\mathbb{F}\langle\langle X \rangle\rangle$, it suffices to prove that $\sum_k A_0^k$ converges (Proposition 2 in [LS12]). As the automaton introduces self-loops labeled by homogeneous linear forms only, and it does not have back-edges, the matrix A_0 is strictly upper triangular (see the above example). Hence, A_0 is nilpotent and $\sum_k A_0^k$ converges. \square

As mentioned in Chapter 2, by a standard construction we can compute an automaton A' without ϵ -transitions equivalent of \tilde{A} [LS12]. The overall time to

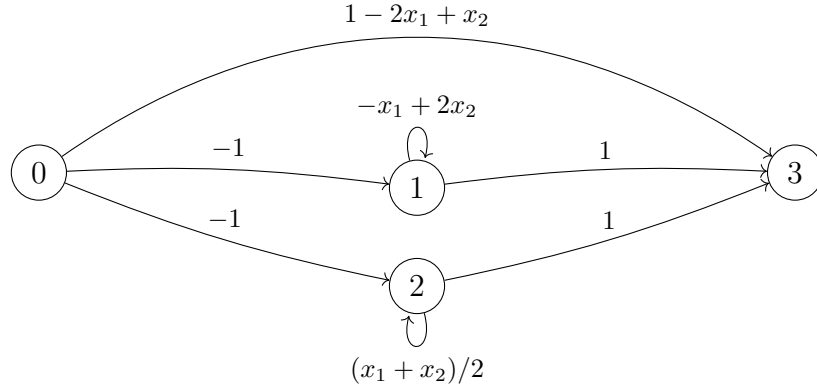


Figure 3.4: Edge Labels replaced by an appropriate automaton.

construct A' is bounded by $\text{poly}(n, s, r)$.

Identity Preserving

Claim 2. $A \neq 0$ in $\mathbb{F}\langle\langle X \rangle\rangle$ if and only if A' does not compute a zero series in $\mathbb{F}\langle\langle X \rangle\rangle$.

Proof of Claim. Let f be a nonzero rational expression in $\mathbb{F}\langle\langle X \rangle\rangle$ computed by A . Then, for some $t \in \mathbb{N}$ and matrix tuple $(M_1, \dots, M_n) \in (\text{Mat}_t(\mathbb{F}))^n$, we have $f(M_1, \dots, M_n) \neq 0$. Therefore, $g = f(\alpha_1 - x_1, \dots, \alpha_n - x_n)$ is also a nonzero rational expression in $\mathbb{F}\langle\langle X \rangle\rangle$ as $g(M'_1, \dots, M'_n) \neq 0$, where $M'_j = \alpha_j I_t - M_j$ for each $j \in [n]$.

To prove that A' computes a nonzero series, it suffices to show that for some matrix substitution A' is defined and outputs a nonzero matrix on that substitution. In g , each affine linear form with inverse looks like $\beta_i(1 - \tilde{L}_i)^{-1}$ where β_i is nonzero and \tilde{L}_i is a homogeneous linear form. Now, let $N_i = \tilde{L}_i(M'_1, \dots, M'_n)$. Since g is defined and nonzero at the point (M'_1, \dots, M'_n) , the matrix $(I_t - N_i)$ is invertible for each i . But it may happen that for some $j \in [r]$, the matrix $\sum_k N_j^k$ does not converge and hence A' is not defined at this matrix tuple. To avoid this problem, we can choose $\gamma \in \mathbb{Q}$ sufficiently small ensuring that $g(\gamma M'_1, \dots, \gamma M'_n)$ is still defined and nonzero,

moreover, for each $i \in [r]$, the matrix $\sum_k N_i^k$, thus obtained, also converges. The following fact is classical and a proof of it is, for example, in [\[Wer05\]](#).

Fact 1. *For any matrix B over \mathbb{Q} , the Neumann series $\sum_k B^k$ converges if the spectral norm of B is less than 1.*

Observation 1. *Let g be a rational expression in $\mathbb{F}\langle X \rangle$ and suppose $g(M'_1, \dots, M'_n) \neq 0$ for some $t \times t$ matrices M'_i . Then there are only finitely many $\gamma \in \mathbb{F}$ for which $g(\gamma M'_1, \dots, \gamma M'_n)$ is not defined or $g(\gamma M'_1, \dots, \gamma M'_n) = 0$.*

Proof. Let us think the parameter γ as indeterminate and note that the output matrix $g(\gamma M'_1, \dots, \gamma M'_n)$ is a $t \times t$ matrix, where each entry is a commutative rational function of form $\frac{h_1}{h_2}$ and h_1 and h_2 are univariate polynomials in γ . The degree of each such h_1, h_2 is some finite value depending on the rational expression g . Clearly, it is not a zero matrix in $\text{Mat}_k(\mathbb{F}(\gamma))$, as for $\gamma = 1$, it is nonzero. Hence, to ensure that $g(\gamma M'_1, \dots, \gamma M'_n)$ is defined and nonzero, it suffices to avoid the roots of the univariates of each entry. □

By [Observation 1](#) we can choose γ small enough such that, for each $i \in [r]$, spectral norm of N_i is less than 1. By [Fact 1](#), the automaton A' is also defined and nonzero on $(\gamma M'_1, \dots, \gamma M'_n)$. Therefore, A' computes a nonzero series.

Conversely, suppose that A' computes a nonzero series in $\mathbb{F}\langle\langle X \rangle\rangle$. Consider any word w such that $[w]A' \neq 0$. Then consider the automaton \mathcal{A} that accepts only the word w and let A_1, \dots, A_n be the transition matrices of the automaton \mathcal{A} for the variables x_1, \dots, x_n . It can be easily observed that $A'(A_1, \dots, A_n)$ is well-defined and a nonzero matrix whose top right-most entry is $[w]A'$ [\[AMS10\]](#) (see [Chapter 2](#) for details). Since whenever A' converges on a point, so does g , we conclude that $g(A_1, \dots, A_n) \neq 0$, which also implies that $f(\alpha_1 I_t - A_1, \dots, \alpha_n I_t - A_n) \neq 0$. Hence f is nonzero in $\mathbb{F}\langle X \rangle$. □

Now the proof of the lemma follows. □

Proof of Theorem 12. We now present the algorithms for white-box and black-box models.

The White-Box Case

Let r be the total number of linear forms or inverses of linear forms in A . Clearly, r is bounded by s . Using Lemma 6 we reduce the problem of deciding whether the ABP A is zero in $\mathbb{F}\langle X \rangle$ to the problem of deciding whether the automaton A' is computing a zero series or not in $\mathbb{F}\langle\langle X \rangle\rangle$. From Lemma 6, A' is of size W which is at most $2s$. Now invoking Theorem 8 we conclude that $A' \neq 0$ if and only if there is a word of length at most $W - 1$ which has nonzero coefficient in A' . Consider the corresponding transition matrix $M_{A'}$ of A' . For each $\ell \leq W - 1$, we construct the branching program $B^{(\ell)} = \underline{u}^T M_{A'}^\ell \underline{v}$ where $\underline{u}, \underline{v}$ are the vectors corresponding to the initial states and final states respectively. As A' does not have any ϵ -transitions, $B^{(\ell)}$ computes words in A' of length exactly ℓ . It suffices to check for each $\ell \leq W - 1$, whether $B^{(\ell)}$ computes an identically zero polynomial. The identity testing algorithm is obtained by applying Theorem 2 on the ABPs $B^{(\ell)}$. The running time of the algorithm is clearly bounded by $\text{poly}(n, s)$.

The Black-Box Case

We now present a deterministic quasi-polynomial time black-box algorithm. Let r be the total number of linear forms or inverses of linear forms in A . Clearly, r is bounded by s . Lemma 5 yields a set S of size $nr + 1$ such that for some $\alpha \in S$, the linear shift $x_j \mapsto \alpha_j - x_j$ ensures that for every edge label, each of the r many L^{-1} in A , the linear form L has a nonzero constant term. Let us fix such $\alpha \in S$. From the proof of Lemma 6, we conclude that there is an automaton A' of size at most $2s$ such that A is zero in $\mathbb{F}\langle X \rangle$ if and only if A' computes a zero series in $\mathbb{F}\langle\langle X \rangle\rangle$.

Let $A^{(\ell)}$ denotes the series computed by A' truncated to the words of length at most ℓ . Let $W = 2s$. Now, by Theorem 8 $A' \neq 0$ if and only if $A^{(W-1)} \neq 0$. We now discuss the effect of $\mathcal{H}_{W^2, W-1, n}$, hitting set from Theorem 4 on A' . It is well known from the proof of Theorem 4 that for each $(h_1, \dots, h_n) \in \mathcal{H}_{W^2, W-1, n}$, each h_i is a $W \times W$ matrix of the following form [FS13]:

$$h_i = \begin{bmatrix} 0 & a_1 & 0 & \cdots & 0 \\ 0 & 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{W-1} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

Using the shape of the matrices h_i , it can be easily checked that for all words $w \in X^*$ of length at least W , $w(h_1, \dots, h_n) = 0$. Hence, evaluating A' at some $(h_1, \dots, h_n) \in \mathcal{H}_{W^2, W-1, n}$ is equivalent to evaluating $A^{(W-1)}$ at (h_1, \dots, h_n) . As already discussed in the previous section for white-box case, we can construct the branching program $B^{(\ell)} = \underline{u}^T M_{A'}^\ell \underline{v}$ computing words of length exactly ℓ in A' , for each $\ell \leq W - 1$, where $M_{A'}$ is the corresponding transition matrix of automaton A' and $\underline{u}, \underline{v}$ are the vectors corresponding to the initial states and final states respectively. Hence, $A^{(W-1)}$ can be computed by an ABP of width at most W^2 . Therefore, A' computes a zero series if and only if for each $(h_1, \dots, h_n) \in \mathcal{H}_{W^2, W-1, n}$, $A'(h_1, \dots, h_n)$ outputs a zero matrix. Hence, by evaluating A on $(\alpha_1 I_W - h_1, \dots, \alpha_n I_W - h_n)$ for each $\alpha \in S$ and $(h_1, \dots, h_n) \in \mathcal{H}_{W^2, W-1, n}$, we can decide A is zero in $\mathbb{F}\langle X \rangle$ or not.

Pseudocode for Theorem 12

The White-Box Algorithm

INPUT: An ABP A of size s whose multi-edge labels are either linear forms or inverse of linear forms over X .

1. Using Lemma 5, construct a set $S \subseteq \mathbb{F}^n$ of size at most $ns + 1$. For each $\underline{\alpha} \in S$, apply the shift $x_j \mapsto \alpha_j - x_j$ for $1 \leq j \leq n$ to the edge labels of A . Fix an $\underline{\alpha} \in S$ such that $L_i(\underline{\alpha}) \neq 0$ for each $i \in [s]$.
2. Construct automaton A' as described in Lemma 6 of size W which is at most $2s$.
3. For each $\ell \leq W - 1$, construct ABP $B^{(\ell)}$ which computes words in A' of length exactly ℓ .
 - (a) For each ℓ , use Theorem 2 to check if $B^{(\ell)} \equiv 0$. In case a nonzero is found, report that $A \neq 0$.
4. If for all $\ell \leq W - 1$, $B^{(\ell)} \equiv 0$ then declare that $A \equiv 0$.

The Black-Box Algorithm

INPUT: A black-box containing an ABP A of size s with each multi-edge label as a linear form or inverse of linear form.

1. Let $\mathcal{H}_{W^2, W-1, n} \subset \text{Mat}_W^n(\mathbb{F})$ be the hitting set from Theorem 4 for ABP of width W^2 and $W - 1$ many layers where $W = 2s$.
2. Using Lemma 5, construct a set $S \subseteq \mathbb{F}^n$ of size $ns + 1$. For each $\underline{\alpha} \in S$, $\underline{h} \in \mathcal{H}_{W^2, W-1, n}$, construct the matrices $M_i = \alpha_i I_W - h_i$ for $1 \leq i \leq n$.
3. Query the black-box at the points (M_1, M_2, \dots, M_n) constructed above.

4. If the black-box ever outputs a nonzero matrix then declare that $A \neq 0$.
Otherwise, declare that $A \equiv 0$.

Conclusion

We have explored the algorithmic questions related to the invertibility and trace of the image of noncommutative polynomials in this chapter. Our algorithm to obtain an invertible image of a noncommutative ABP in a black-box is a special instance of derandomizing Amitsur's theorem on universal division algebra. We also obtain a black-box quasi-polynomial RIT algorithm for ABP with inverses. The main idea is to use the connection of zero-testing of an algebraic automaton to the RIT of this model. To generalize this idea of using algebraic automata for a more general model would be interesting.

Chapter 4

Efficient Identity Testing of Free Group Algebras

In the last chapter, we have studied the invertibility and trace of the image of noncommutative polynomials. In this chapter, we continue the study of the image of noncommutative polynomials in a more general setting. Let f be a noncommutative polynomial in the free noncommutative ring $\mathbb{F}\langle x_1, \dots, x_n, y_1, \dots, y_n \rangle$. Consider the image of f obtained by mapping $y_i \rightarrow x_i^{-1}$ for each i . The resulting object is an \mathbb{F} -linear combination of elements of the free group algebra generated by $\{x_1, \dots, x_n\}$. In this chapter, we explore the problem of identity testing for elements of this free group algebra with black-box access. In order to evaluate the free group algebra expression over $d \times d$ matrices, we must ensure that each matrix substitution M_i for x_i is invertible. This restriction imposes difficulties in obtaining an identity testing algorithm for such expressions. In this chapter, we first formally define such expressions as the free group algebra expressions and define a notion of the degree and sparsity of such expressions. We then show that the known identity testing algorithms for noncommutative polynomials can actually be generalized for free group algebra expressions (i.e. expressions with a polynomially-bounded degree and

exponentially-bounded sparsity).

Free Group Algebras Let $(X \cup X^{-1})^*$ denote the words generated by n (free group) generators $X = \{x_1, x_2, \dots, x_n\}$ and their inverses

$$X^{-1} = \{x_1^{-1}, x_2^{-1}, \dots, x_n^{-1}\}.$$

Given a word $w \in (X \cup X^{-1})^*$, we can *reduce* it repeatedly by replacing with 1 all occurrences of $x_i x_i^{-1}$ or $x_i^{-1} x_i$ in it, $1 \leq i \leq n$. We call $w \in (X \cup X^{-1})^*$ a *reduced word* if we cannot reduce it any further.

Formally, this reduction process is a function $\text{red} : (X \cup X^{-1})^* \rightarrow \Gamma$ where $\text{red}(w)$ is the reduced word corresponding to w . Two words $w_1, w_2 \in (X \cup X^{-1})^*$ are *equivalent* if $\text{red}(w_1) = \text{red}(w_2)$. The elements of the *free group algebra* $\mathbb{F}[\Gamma]$ can be expressed as a finite linear combination

$$f = \sum_w \alpha_w w, \quad \alpha_w \in \mathbb{F},$$

where each $w \in (X \cup X^{-1})^*$ is a reduced word. The *degree* of a free group algebra function f is defined as the maximum length of a word w such that $\alpha_w \neq 0$.

Moreover, f has *sparsity* s if the number of reduced words w with $\alpha_w \neq 0$ is s .

Clearly, by linearity we can extend the mapping to $\text{red} : \mathbb{F}\langle X \cup X^{-1} \rangle \rightarrow \mathbb{F}[\Gamma]$.

Two expressions $f_1, f_2 \in \mathbb{F}\langle X \cup X^{-1} \rangle$ are *equivalent* in $\mathbb{F}[\Gamma]$ if and only if $\text{red}(f_1) = \text{red}(f_2)$. We also use the notation $[w]f$ to denote the coefficient α_w of the reduced word w in the function f .

Connection to Rational Identity Testing: The complexity of identity testing for general rational circuits remains open. For example, given a noncommutative

rational circuit involving addition, multiplication, and division gates, no efficient algorithm (even randomized!) is known to check if the resulting a rational function is zero in the free skew-field. To precisely formulate the problem, we define classes of rational expressions based on Bergman's definition [Ber76] of *inversion height* which we now explain with some notation.

Definition 10. [Ber76] *Let X be a set of free noncommuting variables.*

Polynomials in the free ring $\mathbb{F}\langle X \rangle$ are defined to be rational expressions of height 0. A rational expression of height $i + 1$ is inductively defined to be a polynomial in rational expressions of height at most i , and inverses of such expressions.

Let $\mathcal{E}_{d,0}$ denote all polynomials of degree at most d in the free ring $\mathbb{F}\langle X \rangle$. We inductively define rational expressions in $\mathcal{E}_{d,i+1}$ as follows: Let f_1, f_2, \dots, f_r and g_1, g_2, \dots, g_s be rational expressions in $\mathcal{E}_{d,i}$ in the variables x_1, x_2, \dots, x_n . Let $f(y_1, y_2, \dots, y_s, z_1, z_2, \dots, z_r)$ be a degree- d polynomial in $\mathbb{F}\langle X \rangle$. Then $f(g_1, g_2, \dots, g_s, f_1^{-1}, f_2^{-1}, \dots, f_r^{-1})$ is a rational expression (of inversion height $i + 1$) in $\mathcal{E}_{d,i+1}$.

As already mentioned in the last chapter, black-box identity testing for rational expressions is not well understood in general. In particular, no efficient randomized algorithm is known even for identity testing of the class $\mathcal{E}_{d,1}$. One source of difficulty is the subtle behavior of rational expressions when evaluated on matrix algebras. For example, a surprising result of Bergman [Ber76, Proposition 5.1] shows that there are rational expressions that are nonzero over a dense subset of 2×2 matrices but evaluate to zero on dense subsets of 3×3 matrices.

In this connection, we note that Hrubeš and Wigderson [HW15] have observed that testing if a *correct* rational expression Φ (see [HW15], Section 2) is not identically zero is equivalent to testing if the rational expression Φ^{-1} is *correct*. I.e. testing if a correct rational expression of *inversion height* i is identically zero or not can be reduced to testing if a rational expression of *inversion height* $i + 1$ is correct or not.

Furthermore, testing if a rational expression of *inversion height one* is correct can be done by applying (to each inversion operation in this expression) Amitsur's theorem on universal division algebra (see Chapter 3 for more details) which implies that a nonzero degree $2d - 1$ noncommutative polynomial evaluated on $d \times d$ matrices will be invertible with high probability. However, this does not yield an efficient randomized identity testing algorithm for rational expressions of inversion height one. Because that requires testing the correctness of expressions of inversion height two which is a question left open in their paper [HW15, Section 9].

We also note that the difficulty of designing even an efficient randomized algorithm for the class of noncommutative rational circuits of inversion height 1 comes as no surprise. As already mentioned, designing a polynomial-time, even randomized, PIT algorithm for polynomial-sized noncommutative circuits is an interesting open problem. As the class of noncommutative circuits is a sub-class of rational circuits of inversion height one, RIT for inversion height one also yields a PIT algorithm for general circuits.

Elements of $\mathbb{F}[\Gamma]$ are clearly a special kind of rational expressions of *inversion height one*. Precisely, a rational circuit where we allow the inverse gates only at the bottom-most layer computes a free group algebra expression.

Proposition 2. $\mathbb{F}[\Gamma] \subset \cup_{d>0} \mathcal{E}_{d,1}$.

We note that Proposition 2 is a strict containment. A rational function of inversion height one is not necessarily a free group algebra function. For example, consider $(1 - x)^{-1}$. Suppose there is a free group algebra function $f = \sum_{i=1}^k \alpha_i x^i + \sum_{i=1}^{k'} \beta_i x^{-i}$ for some finite k, k' such that $f = (1 - x)^{-1}$. Now, $(1 - x)^{-1} = \sum_{i \geq 0} x^i$. Let e be the maximum exponent such that x^{-e} is in the support of f . Now, multiplying both sides by x^e we have, $x^e f = \sum_{i \geq e} x^i$ which can not be true.

4.1 An Amitsur-Levitzki Type Theorem

First we show an Amitsur-Levitzki type theorem [AL50] for $\mathbb{F}[\Gamma]$. Let A be an associative algebra with identity over \mathbb{F} . An element $f \in \mathbb{F}[\Gamma]$ is an *identity* for A if

$$f(a_1, \dots, a_n) = 0,$$

for all $a_i \in A$ such that a_i^{-1} is defined for each $i \in [n]$.

Theorem 13. *Let \mathbb{F} be any field of characteristic zero and $f \in \mathbb{F}[\Gamma]$ be a nonzero free group algebra function of degree at most d . Then f is not an identity for the matrix algebra $\text{Mat}_{2d}(\mathbb{F})$.*

Remark 3. *We have stated our results for fields of characteristic zero only for the simplicity of the proof. With suitable modifications, this result extends to fields of positive characteristics as discussed in Section 4.3.*

The following corollary is an immediate consequence.

Corollary 3 (black-box identity test). *There is a black-box randomized $\text{poly}(n, d)$ identity test for degree- d free group algebra expressions in $\mathbb{F}[\Gamma]$.*

The aim of this section is to prove that a nonzero free group algebra expression $f \in \mathbb{F}[\Gamma]$ of degree d does not vanish on the algebra of $2d \times 2d$ matrices over \mathbb{F} . This will give us a randomized polynomial-time black-box identity test for such expressions.

As explained in the previous section, our proof will be guided by automata-theoretic ideas. The transition matrix M_i for variable x_i need to be chosen keeping in mind that x_i^{-1} will be substituted by M_i^{-1} .

Definition 11. *Define a map $\varphi : \mathbb{F}[\Gamma] \rightarrow \mathbb{F}[Y, Z]$ such that φ is identity on \mathbb{F} , and*

for each reduced word $w = x_{i_1}^{b_1} x_{i_2}^{b_2} \cdots x_{i_d}^{b_d}$,

$$\varphi(x_{i_1}^{b_1} x_{i_2}^{b_2} \cdots x_{i_d}^{b_d}) = \xi_1 \xi_2 \cdots \xi_d,$$

where $\xi_j = y_{i_j, j}$ if $b_j = 1$ and $\xi_j = z_{i_j, j}$ if $b_j = -1$. Here, the y_{ij} and z_{ij} are all commuting variables.

The map φ is now defined by linearity on all free group algebra expressions in $\mathbb{F}[\Gamma]$.

We observe some properties.

1. The map φ is injective on the reduced words in $(X \cup X^{-1})^*$. I.e., it maps each reduced word $w \in (X \cup X^{-1})^*$ to a unique monomial over the commuting variables $Y \cup Z$.
2. Consequently, φ is identity preserving. An expression f in $\mathbb{F}[\Gamma]$ is identically zero if and only if its image $\varphi(f)$ is the zero polynomial in $\mathbb{F}[Y, Z]$.
3. φ preserves the sparsity of f . That is, f in $\mathbb{F}[\Gamma]$ is s -sparse if and only if $\varphi(f)$ in $\mathbb{F}[Y, Z]$ is s -sparse.
4. Given the image polynomial $\varphi(f) \in \mathbb{F}[Y, Z]$ in its sparse description (i.e., as a linear combination of monomials), we can efficiently recover the sparse description of $f \in \mathbb{F}[\Gamma]$.

Our goal now is to implement the map φ through matrix substitutions M_i for the variables x_i . As explained before, it is useful to think of matrix M_i as the transition matrix on the input symbol x_i of a substitution automaton. In any reduced word w the automaton must replace x_i occurring in position j by y_{ij} , and must replace x_i^{-1} by z_{ij} .

This transition is handled by the following 2×2 matrix and its inverse. These are

like the basic “building blocks” for the transition matrix M_i :

$$\begin{bmatrix} 0 & y_{i,j} \\ \frac{1}{z_{i,j}} & 0 \end{bmatrix},$$

and its inverse

$$\begin{bmatrix} 0 & z_{i,j} \\ \frac{1}{y_{i,j}} & 0 \end{bmatrix}.$$

In order to ensure that each M_i has a block-diagonal structure, we will introduce a new variable x_0 to play the role of a separator between the letters of a reduced word by defining a new encoding of the free group algebra expressions. Let Γ' be the free group generated by $\{x_0, x_1, \dots, x_n\}$.

Definition 12. *The encoding $\sigma : \mathbb{F}[\Gamma] \rightarrow \mathbb{F}[\Gamma']$ is defined by substituting for each x_i the word $x_0^i x_i x_0^i$ (therefore, x_i^{-1} by $x_0^{-i} x_i^{-1} x_0^{-i}$).*

Consider a degree- d reduced word $w = x_{i_1}^{b_1} x_{i_2}^{b_2} \cdots x_{i_d}^{b_d} \in \Gamma, b_i \in \{-1, 1\}$. Then,

$$\sigma(w) = x_0^{b_1 i_1} x_{i_1}^{b_1} x_0^{b_1 i_1 + b_2 i_2} \cdots x_0^{b_{d-1} i_{d-1} + b_d i_d} x_{i_d}^{b_d} x_0^{b_d i_d}.$$

The following observation shows that σ is an identity preserving map.

Observation 2. *For any $f_1, f_2 \in \mathbb{F}[\Gamma]$, $f_1 \neq f_2$ if and only if $\sigma(f_1) \neq \sigma(f_2)$.*

Proof. It suffices to prove for every reduced word in $(X \cup X^{-1})^*$, σ is injective.

Consider two reduced words w_1 and w_2 . We note that the x_0 variable works as a separator between the letters. Consequently, in $\sigma(w_1)$ and $\sigma(w_2)$, we have at least one x_0 or x_0^{-1} between any two letters of w_1 and w_2 . Therefore, $\sigma(w_1) \neq \sigma(w_2)$. \square

Two polynomials $g, h \in \mathbb{F}[Y, Z]$ are said to be *weakly equivalent* if they have the same *support*. That is, a monomial is nonzero in g if and only if it is nonzero in h .

For $f \in \mathbb{F}[\Gamma]$, let f_ℓ denote the degree- ℓ homogeneous part of f . For an n -tuple $\underline{M} = (M_1, M_2, \dots, M_n)$ of $t \times t$ invertible matrices M_i we denote by $f_{\underline{M}}[1, t]$ the $(1, t)^{th}$ entry of the matrix $f(M_1, M_2, \dots, M_n)$.

The next lemma is the main construction in this section.

Lemma 7. *Let $f \in \mathbb{F}[\Gamma]$ be a nonzero expression of degree d . There is an n -tuple of $2d \times 2d$ invertible matrices $\tilde{\underline{M}} = (\tilde{M}_1, \tilde{M}_2, \dots, \tilde{M}_n)$ whose entries are either scalars, or variables $u \in Y \cup Z$, or their reciprocals $1/u$, such that*

$$f_{\tilde{\underline{M}}}[1, 2d] = \left(f(\tilde{M}_1, \dots, \tilde{M}_n) \right)_{1, 2d} \text{ is a polynomial in } \mathbb{F}[Y, Z],$$

and is weakly equivalent to $\varphi(f_d)$. Furthermore, for each degree- d reduced word $w = x_{i_1}^{b_1} x_{i_2}^{b_2} \dots x_{i_d}^{b_d}$ in $\mathbb{F}[\Gamma]$,

$$(4.1) \quad [\varphi(w)]f_{\tilde{\underline{M}}}[1, 2d] = [w]f \prod_{j=1}^{d-1} (b_j i_j + b_{j+1} i_{j+1}).$$

In other words, Equation [4.1](#) states that the coefficient of the monomial $\varphi(w)$ in the commutative polynomial $f_{\tilde{\underline{M}}}[1, 2d]$ is an integer multiple of the coefficient of the reduced word w in the group algebra expression f , where the multiplicative factor is explicitly given in the equation.

Proof. We first apply the σ -map on f which means substituting for each variable x_i the word $x_0^i x_i x_0^i$. The purpose of the x_0 variable has it as a separator between two consecutive variables from $\{x_i \mid i \in [n]\}$ (or their inverses). This separator variable enables us to keep the transition matrices nicely structured. We now describe the substitution automaton and corresponding (invertible) transition matrices. The run of the substitution automaton on a σ -encoded word $\sigma(w)$ is composed of two steps that are applied in succession repeatedly: the encoding step and the *skip* step. The encoding step deals with variables $\{x_1, \dots, x_n\}$ and the skip step deals with x_0 .

Encoding Step: Recall the structure of the 2×2 building blocks for the transition matrix. When the 2×2 block is the j^{th} diagonal block in transition matrix M_i for variable x_i , the automaton moves from state $2j - 1$ to state $2j$ replacing x_i at position j by $y_{i,j}$, or x_i^{-1} at position j by $z_{i,j}$. The transition matrix M_i will be a block diagonal matrix with such 2×2 invertible blocks as the principal minors along the diagonal:

$$M'_{i,j} = \begin{bmatrix} 0 & y_{i,j} \\ \frac{1}{z_{i,j}} & 0 \end{bmatrix}, \quad M_i = \begin{bmatrix} M'_{i,1} & 0 & 0 & \dots & 0 \\ 0 & M'_{i,2} & 0 & \dots & 0 \\ 0 & 0 & M'_{i,3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & M'_{i,d} \end{bmatrix}.$$

$$M'^{-1}_{i,j} = \begin{bmatrix} 0 & z_{i,j} \\ \frac{1}{y_{i,j}} & 0 \end{bmatrix}, \quad M_i^{-1} = \begin{bmatrix} M'^{-1}_{i,1} & 0 & 0 & \dots & 0 \\ 0 & M'^{-1}_{i,2} & 0 & \dots & 0 \\ 0 & 0 & M'^{-1}_{i,3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & M'^{-1}_{i,d} \end{bmatrix}.$$

The corresponding transitions of the automaton are shown in Figure [4.1](#).

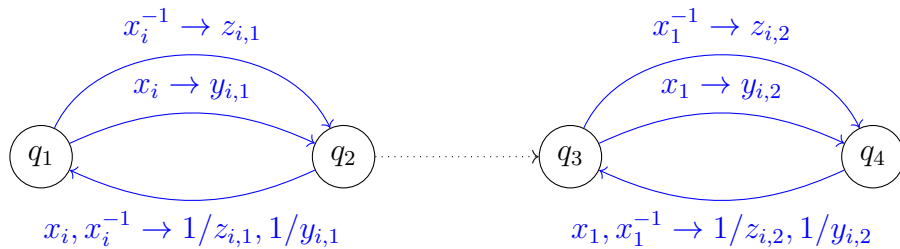


Figure 4.1: The transition diagram of the automaton for x_i variables for degree-4 functions

Skip Step: We now describe the transition matrix M_0 for x_0 . It does not play any role in the commutative encoding φ . The substitution automaton will skip x_0 (or x_0^{-1}) with a nice structure-preserving invertible transition matrix. As explained already, the purpose of x_0 as a separator variable is to ensure that the matrices M_i have a block-diagonal structure.

$$\begin{aligned}
M'_0 &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, & M_0 &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & M'_0 & 0 & \dots & 0 & 0 \\ 0 & 0 & M'_0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & M'_0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \\
M_0'^{-1} &= \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}, & M_0^{-1} &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & M_0'^{-1} & 0 & \dots & 0 & 0 \\ 0 & 0 & M_0'^{-1} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & M_0'^{-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \\
\text{For any } k \in \mathbb{Z}, & M_0'^k &= \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}, & M_0^k &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & M_0'^k & 0 & \dots & 0 & 0 \\ 0 & 0 & M_0'^k & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & M_0'^k & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}.
\end{aligned}$$

The corresponding transitions of the automaton is depicted in Figure [4.2](#).

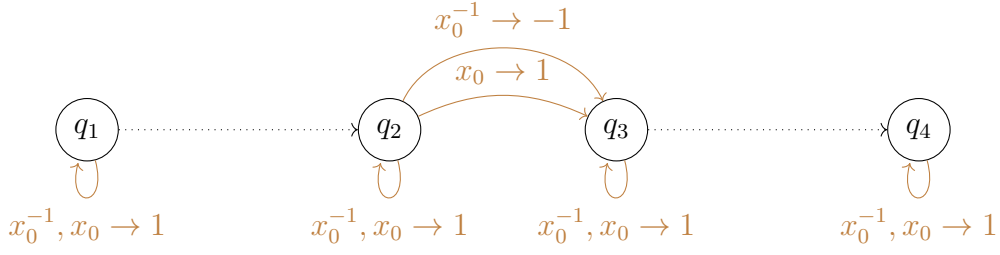


Figure 4.2: The transition diagram of the automaton for x_0 variables for degree-4 functions

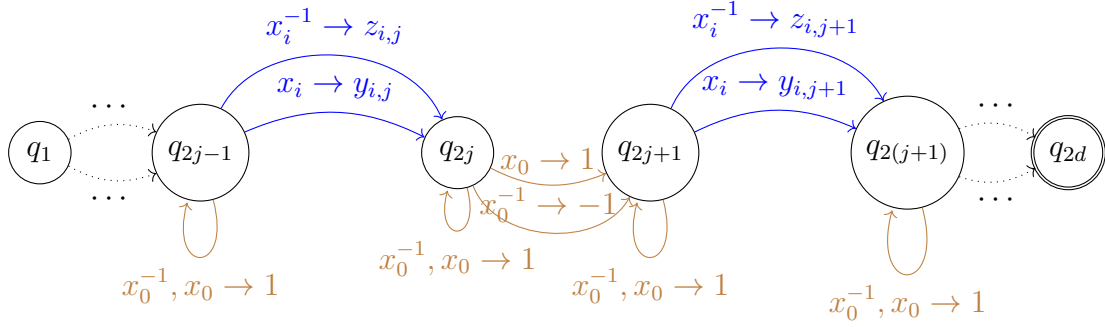


Figure 4.3: The transition diagram of the automaton

Correctness: Consider a degree- d reduced word $w = x_{i_1}^{b_1} x_{i_2}^{b_2} \cdots x_{i_d}^{b_d}$ with nonzero coefficient in the expression f , where $d = \deg(f)$.

If $x_i^{b_i}$ occurs at position j in w , then the automaton (see Figure 4.3) substitutes it by y_{ij} if $b_i = 1$ or z_{ij} if $b_i = -1$. We can compactly express the substitution by

$$\mathbb{I}_{[b_i=1]} y_{i,j} + \mathbb{I}_{[b_i=-1]} z_{i,j},$$

where

$$\mathbb{I}_{[b_i=b]} = \begin{cases} 1 & \text{if } b_i = b \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, for each position $j \in [d-1]$, the adjacent pair $x_{i_j}^{b_j} x_{i_{j+1}}^{b_{j+1}}$ in the reduced word w produces $x_0^{b_j i_j + b_{j+1} i_{j+1}}$ (after reduction) in $\sigma(w)$. Therefore, there is a scalar factor of $(b_j \cdot i_j + b_{j+1} \cdot i_{j+1})$ due to the product $M_0^{b_j i_j} M_0^{b_{j+1} i_{j+1}}$. Consequently, it

follows that

$$\left(w(\tilde{M}_1, \dots, \tilde{M}_n)\right)_{1,2d} = \prod_{j=1}^{d-1} (b_j i_j + b_{j+1} i_{j+1}) \prod_{j=1}^d (\mathbb{I}_{[b_j=1]} y_{i_{jj}} + \mathbb{I}_{[b_j=-1]} z_{i_{jj}}).$$

As φ is a linear map, the lemma follows. \square

4.1.1 Black-box identity test

Proof of Theorem 13. The proof follows easily from Lemma 7. Lemma 7 says that if $f \in \mathbb{F}[\Gamma]$ is nonzero of degree d then the $(1, 2d)$ entry of the matrix $p(M_0 M_1 M_0, \dots, M_0^n M_n M_0^n)$ is a nonzero polynomial in $\mathbb{F}[Y, Z]$. Hence f can not be an identity for $M_{2d}(\mathbb{F})$. \square

Proof of Corollary 3. The identity testing algorithm follows from Theorem 13. We can randomly substitute for the variables in $Y \cup Z$ from a suitably large finite subset of \mathbb{F} and apply the Polynomial Identity Lemma [Sch80, Zip79, DL78]. This completes the proof of the Corollary 3. \square

4.1.2 Reconstruction of sparse expressions

If the black-box contains an s -sparse expression in $\mathbb{F}[\Gamma]$, we give a $\text{poly}(s, n, d)$ deterministic interpolation algorithm (which also gives a deterministic identity testing for such expressions). We use a result of Klivans-Spielman [KS01, Theorem 11] that constructs a test set in deterministic polynomial time for sparse commutative polynomials, which are used for the interpolation algorithm.

Theorem 14 (reconstruction algorithm). *Let \mathbb{F} be a field of characteristic zero and $f \in \mathbb{F}[\Gamma]$ given by black-box be s -sparse and of degree- d . Then we can reconstruct f in deterministic $\text{poly}(n, d, s)$ time with matrix-valued queries to the black-box.*

Proof. Let the black-box expression f be s -sparse of degree d . By Lemma 7 a polynomial $\widehat{\varphi(H_d(f))}$ in $\mathbb{F}[Y, Z]$ is obtained at the $(1, 2d)^{th}$ entry of the matrix $f(M_1, \dots, M_n)$, where $M_i \in \text{Mat}_{2d}(\mathbb{F}[Y, Z])$ is as defined in Lemma 7. By Definition 11, $\varphi(f) \in \mathbb{F}[Y, Z]$ is s -sparse and has $2nd$ variables. Let $\mathcal{H}_{2nd,d,s}$ be the corresponding test set from [KS01] to interpolate a polynomial of degree d and s -sparse over $2nd$ variables. Querying the black-box on $M_1(\underline{h}), M_2(\underline{h}), \dots, M_n(\underline{h})$ for each $\underline{h} \in \mathcal{H}_{2nd,d,s}$ we can interpolate the commutative polynomial $\widehat{\varphi(H_d(f))}$ and obtain an expression for $\widehat{\varphi(H_d(f))} = \sum_{t=1}^s c_{m_t} m_t$ as a sum of monomials.

We will now adjust the extra scalar factors for each monomial in $\widehat{\varphi(H_d(f))}$ to obtain $\varphi(H_d(f))$. We can adjust this for each monomial as Lemma 7 shows that the extra scalar factor for the word $m = x_{i_1}^{b_1} x_{i_2}^{b_2} \dots x_{i_\ell}^{b_\ell}$ is just $\alpha_{\varphi(m)} = \prod_{j=1}^{\ell-1} (b_j \cdot i_j + b_{j+1} \cdot i_{j+1})$. So we construct $\varphi(H_d(f)) = \sum_{t=1}^s \frac{c_{m_t}}{\alpha_{m_t}} m_t$ by removing the factors α_{m_t} for each monomial m_t . We now *invert* the map φ (using the 4th property of Definition 11) on every monomial m_t to obtain $H_d(f)$ as a sum of degree d reduced words. This yields the expression for highest degree homogeneous component of f . We can repeat the above procedure on $f - H_d(f)$ and reconstruct the remaining homogeneous components of f . □ □

4.2 Exponential Degree and Exponential Sparsity

It is known that s -sparse nonzero polynomials in $\mathbb{F}\langle X \rangle$ cannot vanish on $O(\log s)$ dimensional matrix algebras [AJMR17]. We obtain a similar result for $\mathbb{F}[\Gamma]$: nonzero functions in $\mathbb{F}[\Gamma]$ of sparsity s do not vanish on $O(\log s)$ dimensional matrices. It yields a randomized polynomial-time identity test if the black-box contains a free group algebra function f of exponential degree and exponential sparsity. We show the following.

Theorem 15. *Let \mathbb{F} be any field of characteristic zero. Then, a nonzero free group algebra function $f \in \mathbb{F}[\Gamma]$ of sparsity s is not an identity for the matrix algebra $\text{Mat}_k(\mathbb{F})$ for $k \geq c \log s$ for some small constant c .*

Remark 4. *Similarly, we have stated our results for fields of characteristic zero only for the simplicity of the proof. With suitable modifications, this result also extends to fields of positive characteristics (discussed in Section [4.3](#)).*

The following corollary is immediate.

Corollary 4. *Given a degree- D free group algebra function $f \in \mathbb{F}[\Gamma]$ of sparsity s as black box, we can check whether f is identically zero or not in randomized $\text{poly}(n, \log D, \log s)$ time.*

It is known [\[AJMR17\]](#) that a nonzero noncommutative polynomial of sparsity s cannot be an *identity* for $O(\log s)$ dimensional matrix algebras. In this section, we show a similar result for free group algebra functions. In particular, we prove that the dimension of the matrix algebra for which a nonzero free group algebra function f does not vanish is logarithmic in the sparsity of f . It yields a randomized $\text{poly}(\log D, \log s, n)$ time identity testing algorithm when the black-box contains a free group algebra function of degree D and sparsity s .

We first recall the notion of *isolating index set* from [\[AJMR17\]](#).

Definition 13. *Let $\mathcal{M} \subseteq \{X, X^{-1}\}^D$ be a subset of reduced words of degree D . An index set $I \subseteq [D]$ is an isolating index set for \mathcal{M} if there is a word $m \in \mathcal{M}$ such that for each $m' \in \mathcal{M} \setminus \{m\}$ there is an index $i \in I$ for which $m[i] \neq m'[i]$. I.e. no other word in \mathcal{M} agrees with m on all positions in the index set I . We say m is an isolating word.*

In the following lemma, we show that \mathcal{M} has an isolating index set of size $\log |\mathcal{M}|$. The proof is identical to [\[AJMR17\]](#). Nevertheless, we give simple details for completeness as we deal with both variables and their inverses.

Lemma 8. [AJMR17](#) Let $\mathcal{M} \subseteq \{X, X^{-1}\}^D$ be reduced degree- D words. Then \mathcal{M} has an isolating index set of size k which is bounded by $\log |\mathcal{M}|$.

Proof. The words $m \in \mathcal{M}$ are indexed, where $m[i]$ denotes the variable (or the inverse of a variable) in the i^{th} position of m . Let $i_1 \leq D$ be the first index such that not all words agree on the i_1^{th} position. Let

$$S_j^+ = \{m : m[i_1] = x_j\}$$

$$S_j^- = \{m : m[i_1] = x_j^{-1}\}.$$

For some j , $|S_j^+|$ or $|S_j^-|$ is of size at most $|\mathcal{M}|/2$. Let S_j^b denote that subset, $b \in \{+, -\}$. We replace \mathcal{M} by S_j^b and repeat the same argument for at most $\log |\mathcal{M}|$ steps. Clearly, by this process, we identify a set of indices $I = \{i_1, \dots, i_{k'}\}$, $k' \leq \log |\mathcal{M}|$ such that the set shrinks to a singleton set $\{m\}$. Clearly, I is an isolating index set as witnessed by the *isolating word* m . □

Proof of Theorem [15](#)

Proof. Let $k = 4(k' + 1)$ where k' is the size of the isolating set I . As in Section [4.1](#), we apply σ -encoding on input free group algebra function f . The transition matrices for each x_i is denoted by M_i . Let $I = \{i_1, \dots, i_{k'}\}$ be an isolating set such that $i_1 < \dots < i_{k'}$. Intuitively, the NFA does one of two operations on each symbol (a variable or its inverse) of the input expression: a *skip* or an *encode* similar to the proof of Lemma [7](#). However, the crucial difference is that it encodes only the positions belonging to the isolating set. In a *Skip* stage, the NFA deals with positions that are not part of the (guessed) isolating index set. In this stage, the NFA substitutes the x_0 variables by suitable scalars and x_i variables by block variables $\{\xi_1, \dots, \xi_{k'+1}\}$. The NFA nondeterministically decides whether the *skip*

stage is over and it enters the *encode* stage for a guessed index of the isolating set.

It then substitutes x_i and x_i^{-1} variables by $y_{i,j}$ and $z_{i,j}$ respectively. Fig. 4.4

summarizes the action of the NFA.

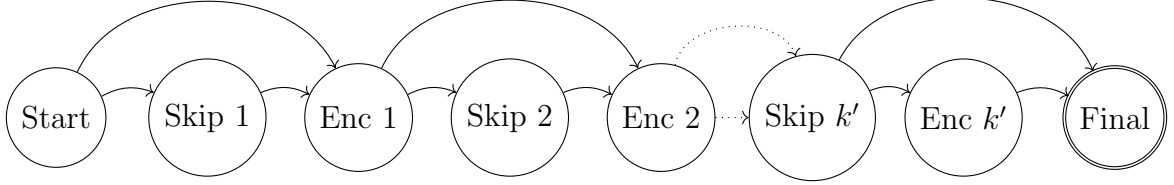


Figure 4.4: The transition diagram of the automaton

We define $k \times k$ matrix M_0 as a block diagonal matrix of $k' + 1$ many copies of a

4×4 matrix M'_0 where $M'_0 = I_4 + i(e_{12} + e_{34} + e_{32} + e_{14})$.

$$M'_0 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad M_0 = \begin{bmatrix} M'_0 & 0 & 0 & \dots & 0 \\ 0 & M'_0 & 0 & \dots & 0 \\ 0 & 0 & M'_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & M'_0 \end{bmatrix},$$

$$M'^{-1}_0 = \begin{bmatrix} 1 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad M_0^{-1} = \begin{bmatrix} M'^{-1}_i & 0 & 0 & \dots & 0 \\ 0 & M'^{-1}_i & 0 & \dots & 0 \\ 0 & 0 & M'^{-1}_i & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & M'^{-1}_i \end{bmatrix}.$$

Notice that

$$M_0^{b_1i+b_2j} = \begin{bmatrix} 1 & (b_1i + b_2j) & 0 & (b_1i + b_2j) \\ 0 & 1 & 0 & 0 \\ 0 & (b_1i + b_2j) & 1 & (b_1i + b_2j) \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We now define the $k \times k$ transition matrix M_i for each $i \in [n]$ as a block diagonal matrix,

$$M'_{i,j} = \begin{bmatrix} 0 & y_{i,j} \\ \frac{1}{z_{i,j}} & 0 \end{bmatrix}, \quad M'_{\xi_i} = \begin{bmatrix} 0 & \xi_i \\ \frac{1}{\xi_i} & 0 \end{bmatrix},$$

$$M_i = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & M_{\xi_1} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & M'_{i,1} & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & M_{\xi_2} & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & M_{\xi_{k'+1}} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}.$$

These matrices can be seen as the transitions of a suitable NFA. We sketch the transitions of this NFA.

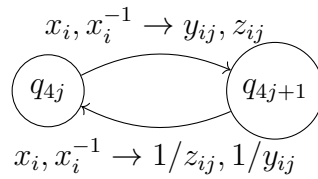


Figure 4.5: The transition diagram of the automaton at *Encode* stage

We refer to an NFA transition $q_i \rightarrow q_j$ as a *forward edge* if $i < j$ and a *backward edge* if $i > j$. We classify the backward edges in three categories based on the substitution on the edge-label. We say, a backward edge is of *type A* if a variable is substituted by a scalar value; a backward edge is of *type B* if a variable is

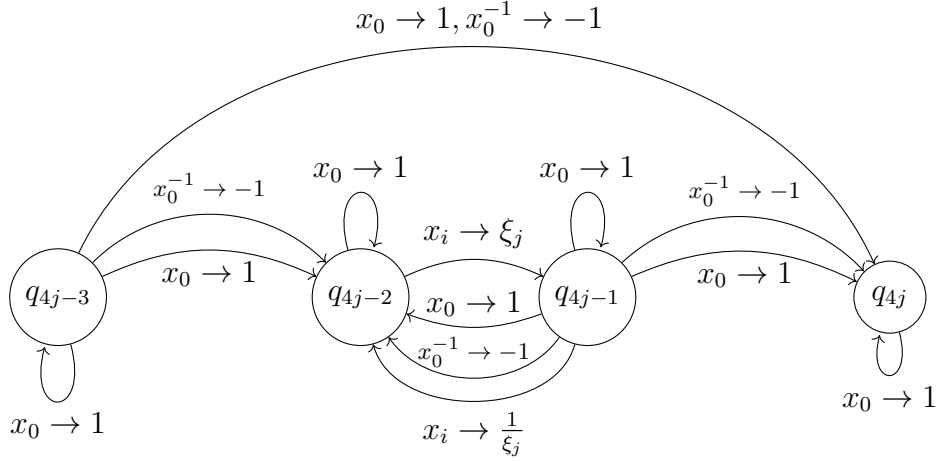


Figure 4.6: The transition diagram of the automaton at *Skip* stage

substituted by $\frac{1}{\xi_j}$ for some j ; a backward edge is of *type C* if a variable is substituted by $\frac{1}{y_{i,j}}$ or $\frac{1}{z_{i,j}}$ for some i, j .

Define \hat{f} in $\mathbb{F}(Y, Z, \xi)$ to be rational function we obtain at the $(1, k)^{th}$ entry by evaluating the expression $f(\tilde{M}_1, \dots, \tilde{M}_n)$ where for each i , $\tilde{M}_i = M_0^i M_i M_0^i$. Let $I = \{i_1, \dots, i_{k'}\}$ be the isolating set such that for each $j \in [k']$,

$$i_j = \sum_{j'=1}^j \ell_{j'} + (j - 1).$$

Notice that, the isolating word m of highest degree D will be of the following form,

$$m = W_1 x_{i_1}^{b_{i_1}} W_2 x_{i_2}^{b_{i_2}} \dots W_k' x_{i_{k'}}^{b_{i_{k'}}} W_{k'+1},$$

where each subword $W_j = x_{j_1}^{b_{j_1}} x_{j_2}^{b_{j_2}} \dots x_{j_{\ell_j}}^{b_{j_{\ell_j}}}$ is of length $\ell_j \geq 0$ and $\ell_{k'+1} = D - i_{k'}$, where some of the W_j could be the empty word as well.

We now define, \hat{m} as a monomial over $\{Y, Z, \xi\}$ of degree D ,

$$\hat{m} = \prod_{j=1}^{k'+1} \xi_j^{\ell_j} \cdot \prod_{j=1}^{k'} (\mathbb{I}_{[b_{i_j}=1]} y_{i_j} + \mathbb{I}_{[b_{i_j}=-1]} z_{i_j}).$$

¹Recall that $k = 4(k' + 1)$ where k' is the size of an isolating set.

Now the proof follows from the following claim.

Claim 3.

$$[\hat{m}]f \neq 0 \text{ if and only if } [m]f \neq 0.$$

Proof. Let us first show that \hat{m} is generated by substituting m in the NFA. Consider a walk of the NFA on an input word m that reaches state k using only *type A* backward edges. In that case, m is substituted by $\alpha \cdot \hat{m}$ where α is some nonzero constant obtained as a product of $[m]f$ with the scalars obtained as substitutions from the edges involving the x_0 variable in the *Skip* stages. Indeed, as we can see from the entries of product matrices $M_0^{b_1 i} \cdot M_0^{b_2 j}$, where $b_1, b_2 \in \{-1, 1\}$, the scalar α is a product of $[m]f$ with terms of the form $b_1 i + b_2 j$, for $i \neq j$, each of which is nonzero for any reduced word.

It suffices to show that for any word $m' \neq m$, where m' has degree $\leq D$, no walks of the NFA accepting m' generate \hat{m} after substitution. For a computation path J , the monomial m_J in \hat{f} has two parts, let us call it $skip_J$ and $encode_J$ where $skip_J$ is a monomial over $\{\xi_1, \dots, \xi_{k'+1}\}$ and $encode_J$ is a monomial over $\{y_{ij}, z_{ij}\}_{i \in [n], j \in [k']}$. If the computation path J (which is different from the computation path described above for \hat{m}) uses only *type A* backward edges, then necessarily $m_J \neq \hat{m}$ from the definition of *isolating index set*. This argument is analogous to the argument given in [\[AJMR17\]](#).

Now consider a walk J which involves backward edges of other types. Let us first consider those walks that take backward edges only of *type A* and *type B*. Such a walk still produces a monomial over $\{y_{ij}, z_{ij}\}_{i \in [n], j \in [k']}$ and $\{\xi_i\}_{1 \leq i \leq k'+1}$ because division only by ξ_i variables occur in the resulting expression. Since \hat{m} is of highest degree, the total degree of these monomials is strictly lesser than degree of \hat{m} . For those walks that take at least one backward edge of *type C*, a rational expression in $\{y_{ij}, z_{ij}\}_{i \in [n], j \in [k']}$ and $\{\xi_i\}_{1 \leq i \leq k'+1}$ is produced (as there is division by y_{ij} or z_{ij}

variables). As the sum of the degree of the numerator and degree of the denominator is bounded by the total degree, the degree of the numerator is smaller than degree of \hat{m} .

Thus the $(1, k)^{th}$ entry of the output matrix is of the form $\sum_{i=1}^{N_1} c_i m_i + \sum_{j=1}^{N_2} r_j$ where $\{m_1, \dots, m_{N_1}\}$ are monomials arising from different walks (w.l.o.g. assume that $m_1 = \hat{m}$) and $\{r_1, \dots, r_{N_2}\}$ are the rational expressions from the other walks (due to the backward edges of *type C*). Note that, denominator in each r_j is a monomial over Y, Z of degree at most D . Let $L = \prod_{i=1}^n \prod_{j=1}^{k'} y_{i,j}^D \cdot z_{i,j}^D$. Now, we have,

$$\sum_{i=1}^{N_1} c_i m_i + \sum_{j=1}^{N_2} r_j = \frac{1}{L} \cdot \left(\sum_{i=1}^{N_1} c_i m_i L + \sum_{j=1}^{N_2} p_j \right).$$

Since $\hat{m}L \neq m_i L$ for any $i \in \{2, \dots, N_1\}$ and degree of each $p_j <$ degree of $\hat{m}L$ for any $j \in \{1, \dots, N_2\}$, the numerator of the final expression is a nonzero polynomial in $\mathbb{F}[Y, Z, \xi]$. \square

The above proof shows that the matrix $f(N_1 M_1 N_1, \dots, N_n M_n N_n)$ is nonzero with rational entries in $\mathbb{F}[Y, Z, \xi]$. Each entry is a linear combination of terms of the form m_1/m_2 , where m_1 and m_2 are monomials in $Y \cup Z \cup \{\xi_1, \dots, \xi_{k'+1}\}$ of degree bounded by D . Note that, the matrix dimension is $k = c \log s$ for some constant c . This completes the proof of Theorem [15](#). \square

To get an identity testing algorithm, we can do random substitutions. The matrix dimension is $\log s$ and the overall running time of the algorithm is $\text{poly}(n, \log s, \log D)$. This also proves Corollary [4](#). \square

Remark 5. For algorithmic purposes, we note that Theorem [13](#) is sometimes preferable to Theorem [15](#). For instance, the encoding used in Theorem [15](#) does not preserve the sparsity of the polynomial as required in the sparse reconstruction result (see Theorem [14](#)).

4.3 Over Small Finite Fields

Let \mathbb{F} be any finite field. We will ensure that the scalar used in the matrix construction and the scalar produced by the automaton (see Equation 4.1) described in Section 4.1 for each word m in the free group algebra is nonzero in \mathbb{F} . We first note that one can modify the Definition 14 of σ -encoding in the following way.

Introduce a new variable \tilde{x}_i corresponding to x_i for each $i \in [n]$. Let Γ' be the free group generated by $\{x_1, \tilde{x}_1, \dots, x_n, \tilde{x}_n\}$.

Definition 14. *The encoding $\sigma : \mathbb{F}[\Gamma] \rightarrow \mathbb{F}[\Gamma']$ is defined by substituting for each x_i the word $\tilde{x}_i x_i \tilde{x}_i$ (therefore, x_i^{-1} by $\tilde{x}_i^{-1} x_i^{-1} \tilde{x}_i^{-1}$).*

For any two free group algebra functions f_1, f_2 , $f_1 \neq f_2$ if and only if $\sigma(f_1) \neq \sigma(f_2)$.

The proof follows from Observation 2.

Now following the proof of Lemma 7, we can modify the matrix substitution in the “skip step” as follows,

$$\tilde{x}_i \leftarrow \tilde{M}_i = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \tilde{M}'_i & 0 & \dots & 0 & 0 \\ 0 & 0 & \tilde{M}'_i & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \tilde{M}'_i & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \quad \text{where, } \tilde{M}'_i = \begin{bmatrix} 1 & i \\ 0 & 1 \end{bmatrix}.$$

Observe that, the basic building block of each \tilde{M}_i is a simple 2×2 matrix with i in the top-right corner. It ensures that for each pair $i, j \in [n]$, each term $(b_i i + b_j j) \neq 0$ where $b_i, b_j \in \{-1, +1\}$. Indeed, for some fixed $\alpha \in \mathbb{F}$, one can redefine each 2×2 block \tilde{M}'_i with α^i in the top-right corner instead of i . It also ensures that for each pair $i, j \in [n]$, each term $(b_i \alpha^i + b_j \alpha^j) \neq 0$ if $\alpha^i \pm \alpha^j \neq 0$ for any $i, j \in [n]$. Observe that, $\alpha^i - \alpha^j \neq 0$ if each α^i is distinct. To ensure $\alpha^i + \alpha^j \neq 0$, we choose α such that

$\alpha^{2(i-j)} \neq 1$ for any $i, j \in [n]$. Therefore, the goal is to find an element $\alpha \in \mathbb{F}$ of order at least $2n$. To make it work over any small finite field \mathbb{F} of characteristic p also, we can construct an extension field \mathbb{F}' of \mathbb{F} such that $|\mathbb{F}'| \geq 2n$.

Thus, for each word m , the scalar produced by the automata is nonzero.

Furthermore, the test set of [KS01] works for all fields. Hence Theorem [14] holds for all finite fields too. To obtain Corollaries [3] and [4], we will do random substitutions from a suitable small degree extension field and use Polynomial Identity Lemma [Sch80, Zip79, DL78]. In summary, our algorithms in this chapter can be adapted to work for all finite fields also.

Conclusion

In this chapter, we have studied the identity testing problem for free group algebra expressions and obtained algorithmic results similar to the known results for noncommutative polynomials. Theorem [13] shows, analogous to the Amitsur-Levitzki theorem [AL50], a degree upper bound for nonzero free group algebra expressions that can vanish on $d \times d$ matrices. However, unlike the Amitsur-Levitzki theorem, we do not know whether or not this upper bound is tight. On the lower bound side, it will be interesting to obtain a nonzero free group algebra expression of degree d that is an identity for $d \times d$ matrices.

Chapter 5

Fast Exact Algorithms using Hadamard Product of Polynomials

In the previous two chapters, we have studied the image of noncommutative polynomials and obtained new algorithmic results in noncommutative algebraic complexity. In this chapter, we obtain new algorithmic results using techniques from noncommutative algebraic complexity. We study two algebraic problems, namely multilinear monomial detection, and multilinear monomial counting, and show that these problems are special instances of computing commutative Hadamard products which we reduce to noncommutative Hadamard product computation of two suitable noncommutative polynomials. Using this connection, we obtain faster algorithms for those algebraic problems.

It turns out that we know only a few examples of designing efficient algorithms for combinatorial problems using techniques from noncommutative algebraic complexity. One interesting aspect of noncommutative computation is that the noncommutative determinant can be used to define an unbiased estimator for the commutative permanent polynomial as initially discovered by Godsil and Gutman [GG81]. In this chapter, we explore one such application of using noncommutative computation.

Introducing Multilinear Monomial Detection and Multilinear Monomial

Counting: Let \mathbb{F} be any field and $X = \{x_1, x_2, \dots, x_n\}$ be a set of commuting variables. Koutis and Williams [Kou08, Wil09, KW16] introduced and studied two natural algorithmic problems in arithmetic circuits:

1. Given as input an arithmetic circuit C of $\text{poly}(n)$ size computing a polynomial $f \in \mathbb{F}[X]$, the k -multilinear monomial counting problem denoted (k, n) -MLC is to compute the sum of the coefficients of all degree- k multilinear monomials in the polynomial f . In this formulation, the problem is a generalization of counting the degree- k multilinear monomials.
2. The k -multilinear monomial detection problem denoted k -MMD, is to test if there is a degree- k multilinear monomial in the polynomial f with a non-zero coefficient.

Observe that (k, n) -MLC and k -MMD can be solved in time $O^*(n^k)$ essentially by *extracting out* the degree k part in the circuit C computing f , and enumerating all degree- k monomials with their coefficients in f .

The above two problems have attracted significant attention in recent times. In particular, Koutis [Kou08], Williams [Wil09], and Koutis-Williams [KW16] have studied (k, n) -MLC and k -MMD problems from the viewpoint of parameterized and exact algorithms. These problems are natural generalizations of the well-studied k -path detection and counting problems in a given graph [Kou08]. Moreover, some other combinatorial problems like k -Tree, m -Dimensional k -Matching [KW16], well-studied in the parameterized complexity, reduces to these problems. In fact, the first randomized FPT algorithms for the decision version of these combinatorial problems were obtained from an $O^*(2^k)$ algorithm for k -MMD for monotone circuits using an algebraic technique based on group algebras [Kou08, Wil09, KW16]. Recently, Brand et al. [BDH18] have given the first randomized FPT algorithm for

k -MMD for general circuits that runs in time $O^*(4.32^k)$. Their method is based on exterior algebra and color coding [AYZ95].

In general, the exact counting versions of the k -path problem and many related problems are $\#\text{W}[1]$ -hard with respect to parameter k . For these counting problems, improvements to the trivial $O^*(n^k)$ time exhaustive search algorithm is known only in some cases (like counting k -paths) [BHKK09]. In this connection, Koutis and Williams [KW16] ask if there is an algorithm for (k, n) -MLC that improves upon the naive $O^*(n^k)$ time algorithm. It would yield faster algorithms for several exact counting problems. Indeed, Koutis and Williams in [KW16] give an algorithm of running time $O^*(n^{k/2})$ to compute the *parity* of the sum of coefficients of degree- k multilinear monomials.

In this chapter, we make progress on the Koutis and Williams problem, mentioned above, by giving an $O^*(n^{k/2})$ algorithm for the (k, n) -MLC problem. Broadly, we develop a new approach to the k -MMD, (k, n) -MLC problems, and related problems. Our algorithms are based on computing the Hadamard product of polynomials. See Definition 8 in Chapter 2 for more details.

Recall from Chapter 2 that the Hadamard product of polynomials has turned out to be a useful tool in *noncommutative* computation [AJS09, AS18]. A contribution of this chapter is to develop a new method for computing the Hadamard product in the *commutative* setting (as defined above), which turns out to be useful for designing efficient FPT and exact algorithms.

We now give some formal basic definitions and set up the notation for this chapter.

Some basic definitions: We begin with the definition of the Hadamard product. Notice that, one can define the Hadamard product of two commutative polynomials also (i.e. a commutative analog of Definition 8).

Definition 15. The (commutative) Hadamard product of polynomials f and g in $\mathbb{F}[X]$ is defined as

$$f \circ g = \sum_m ([m]f \cdot [m]g) \cdot m,$$

where m runs over all monomials nonzero in f or g , and $[m]f$ denotes the coefficient of the monomial m in f .

A polynomial $f \in \mathbb{F}[X]$ is said to be *multilinear* if for every nonzero monomial $m = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$ of f we have $e_i \leq 1$.

An important family of polynomials for this chapter are the *elementary symmetric polynomials* which are defined over any field \mathbb{F} as follows:

The *elementary symmetric polynomial* $S_{n,k} \in \mathbb{F}[X]$ of degree k over the n variables $X = \{x_1, x_2, \dots, x_n\}$ is defined as

$$S_{n,k} = \sum_{S \subset [n]: |S|=k} \prod_{i \in S} x_i.$$

By definition, $S_{n,k}$ is the sum of all the degree- k multilinear monomials.

Polynomial-time as the notion of feasible computation, and the accompanying hardness theory of NP-completeness, is refined in the world of parameterized computation where the input instance is augmented with a fixed parameter k . Feasible parameterized computation means that the running time is of the form $t(k) \cdot n^{O(1)}$ for inputs of size n and fixed-parameter k , where $t(\cdot)$ can be an arbitrary function that depends solely on parameter k . The parameterized analogue of P is denoted FPT. It is the class of *fixed parameter time* solvable problems, and algorithms with such running time are called FPT algorithms. The analogue of NP is denoted W[1], but the hardness theory has more technical details that can be found in the textbook by Downey and Fellows [DF13](#).

The notation $O^*(T(n, k))$ suppresses polynomial factors. Thus, a function in

$O^*(T(n, k))$ is of the form $O(T(n, k) \cdot \text{poly}(n, k))$.

The following is a convenient notation for ascending sums of binomial coefficients:

$$\binom{n}{\downarrow i} \triangleq \sum_{j=0}^i \binom{n}{j}.$$

From Commutative to Noncommutative

A crucial element of our results on Hadamard product computation is going from the commutative to the noncommutative setting.

Let $X = \{x_1, x_2, \dots, x_n\}$ be n commuting variables and $Y = \{y_1, y_2, \dots, y_n\}$ be n corresponding noncommuting variables¹. Suppose $f \in \mathbb{F}[X]$ is a homogeneous degree- k polynomial represented by an arithmetic circuit C . We define its noncommutative version \mathfrak{C} which computes a noncommutative homogeneous degree- k polynomial denoted $\hat{f} \in \mathbb{F}\langle Y \rangle$ as follows.

Definition 16. *Let C be a commutative arithmetic homogeneous circuit C computing a homogeneous degree- k polynomial $f \in \mathbb{F}[X]$. The noncommutative version of C , \mathfrak{C} is the noncommutative circuit obtained from C by fixing an ordering of the inputs to each product gate in C and replacing x_i by the noncommuting variable y_i , $1 \leq i \leq n$. The polynomial computed by \mathfrak{C} is denoted $\hat{f} \in \mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$.*

We similarly define the noncommutative version \mathfrak{B} of an ABP B .

The *arithmetic circuit complexity* of a polynomial f is the size $s(f)$ of the smallest circuit representing f . The *ABP complexity* of f is the size $B(f)$ of the smallest ABP representing f . This notation is used for both commutative and noncommutative polynomials.

Remark 6. *The definition of the noncommutative \mathfrak{C} is entirely dependent on the ordering of the inputs to each product gate of C . This could, for instance, be by*

¹Throughout, we use y_i to denote noncommuting variables associated with the x_i .

increasing the order of the gate names of the circuit C . Since C is a homogeneous circuit, we note that the circuit \mathfrak{C} is also a homogeneous circuit.

We introduce the following notation:

$$X_k \triangleq \{\text{all degree } k \text{ monomials over } X\}.$$

$$Y^k \triangleq \{\text{all degree } k \text{ monomials over } Y\}.$$

For mapping noncommutative polynomials back to commutative polynomials, we use the substitution map:

$$\nu : Y \rightarrow X \text{ defined as } \nu(x_i) = y_i, 1 \leq i \leq n.$$

This map extends to $\nu : Y^k \rightarrow X_k$ and, by linearity, gives a ring homomorphism (it is easily checked that $\nu(f + g) = \nu(f) + \nu(g)$ and $\nu(fg) = \nu(f)\nu(g)$):

$$\nu : \mathbb{F}\langle Y \rangle \rightarrow \mathbb{F}[X],$$

and its kernel, $\ker(\nu)$, is all those noncommutative polynomials over Y that vanish if the variables are allowed to commute.

Each monomial $m \in X_k$ can appear as a different noncommutative monomial $\hat{m} \in \nu^{-1}(m)$ in $\hat{f} \in \mathbb{F}\langle Y \rangle$. We will use the notation $\hat{m} \rightarrow m$ to denote that $\hat{m} \in \nu^{-1}(m)$. Observe that

$$[m]f = \sum_{\hat{m} \in \nu^{-1}(m)} [\hat{m}]\hat{f} = \sum_{\hat{m}: \hat{m} \rightarrow m} [\hat{m}]\hat{f}.$$

The noncommutative circuit \mathfrak{C} is not directly useful for computing Hadamard product. However, the following symmetrization helps. We first explain how permutations $\sigma \in S_k$ act on the set Y^k of degree- k monomials. The action extends,

by linearity, to all homogeneous degree- k polynomials.

For each monomial $\hat{m} = y_{i_1} y_{i_2} \cdots y_{i_k}$, the permutation $\sigma \in S_k$ maps \hat{m} to the monomial \hat{m}^σ defined as

$$\hat{m}^\sigma = y_{i_{\sigma(1)}} y_{i_{\sigma(2)}} \cdots y_{i_{\sigma(k)}}.$$

By linearity, the polynomial \hat{f} is mapped by σ to the polynomial

$$\hat{f}^\sigma = \sum_{\hat{m} \in Y^k} [\hat{m}] \hat{f} \cdot \hat{m}^\sigma.$$

Definition 17 (Symmetrized polynomial). *The symmetrized polynomial $f^* \in \mathbb{F}\langle Y \rangle$ obtained from $f \in \mathbb{F}[X]$ is defined as the degree- k homogeneous polynomial*

$$f^* = \sum_{\sigma \in S_k} \hat{f}^\sigma.$$

5.1 Hadamard Product Framework

Given two arithmetic circuits C_1 and C_2 computing polynomials f_1 and f_2 , it is in general unlikely that $f_1 \circ f_2$ can be computed by an arithmetic circuit C of size $\text{poly}(|C_1|, |C_2|)$. This can be observed from the fact that for the symbolic matrix $X = (x_{i,j})_{1 \leq i,j \leq n}$, the Hadamard product of the determinant polynomial $\text{Det}(X)$ with itself is the permanent polynomial $\text{Per}(X)$ which does not have polynomial-size circuit assuming Valiant's $\text{VP} \neq \text{VNP}$ hypothesis [Val79]. However, it is well known that $\text{Det}(X)$ can be computed by a polynomial-size ABP [MV97].

Nevertheless, we develop a method for computing the *scaled* Hadamard product of commutative polynomials in some special cases.

Definition 18. *The scaled Hadamard product of polynomials $f, g \in \mathbb{F}[X]$ is defined*

as

$$f \circ^s g = \sum_m (m! \cdot [m]f \cdot [m]g) \cdot m,$$

where for monomial $m = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_r}^{e_r}$ we define $m! = e_1! \cdot e_2! \cdot \dots \cdot e_r!$.

Computing the scaled Hadamard product is key to our algorithmic results for k -MMD and (k, n) -MLC. Broadly, it works as follows: we transform polynomials f and g to suitable *noncommutative* polynomials. We compute their (noncommutative) Hadamard product (see the discussion in p.29-30, Chapter 2 [AJS09, AS18], and then recover the scaled commutative Hadamard product $f \circ^s g$ (or evaluate it at a desired point $\underline{a} \in \mathbb{F}^n$).

Suppose $f \in \mathbb{F}[x_1, x_2, \dots, x_n]$ is a homogeneous degree- k polynomial given by a circuit C . We have its noncommutative version \mathfrak{C} which computes the noncommutative homogeneous degree- k polynomial $\hat{f} \in \mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$ (see Definition 16).

Recall that X_k denotes the set of all degree- k monomials over X , Y^k denote all degree- k noncommutative monomials over Y , and we have $[m]f = \sum_{\hat{m} \rightarrow m} [\hat{m}] \hat{f}$, where $m \in X_k$ and $\hat{m} \in Y^k$. We will use the *symmetrized polynomial* (see Definition 17), $f^* = \sum_{\sigma \in S_k} \hat{f}^\sigma$, to compute the scaled Hadamard product $f \circ^s g$.

Lemma 9. *For a homogeneous degree- k commutative polynomial $f \in \mathbb{F}[X]$ given by circuit C , and its noncommutative version \mathfrak{C} computing polynomial $\hat{f} \in \mathbb{F}\langle Y \rangle$, consider the symmetrized noncommutative polynomial $f^* = \sum_{\sigma \in S_k} \hat{f}^\sigma$. Then for each monomial $m \in X_k$ and each word $m' \in Y^k$ such that $m' \rightarrow m$, we have:*

$$[m']f^* = m! \cdot [m]f.$$

Proof. Let $f = \sum_m [m]f \cdot m$ and $\hat{f} = \sum_{\hat{m}} [\hat{m}] \hat{f} \cdot \hat{m}$. As already observed, $[m]f = \sum_{\hat{m} \rightarrow m} [\hat{m}] \hat{f}$. Now, we write $f^* = \sum_{m'} [m'] f^* \cdot m'$. The group S_k acts on Y^k by permuting the positions. Suppose $m = x_{i_1}^{e_1} \dots x_{i_q}^{e_q}$ is a type $\underline{e} = (e_1, \dots, e_q)$

monomial over X_k and $m' \rightarrow m$. Then, by the well-known *Orbit-Stabilizer Theorem* the orbit $O_{m'}$ of m' under the action of S_k has size $\frac{k!}{m!}$. It follows that

$$[m']f^* = \sum_{\hat{m} \in O_{m'}} m! \cdot [\hat{m}]f = m! \sum_{\hat{m} \rightarrow m} [\hat{m}]f = m! \cdot [m]f.$$

It is important to note that for some $\hat{m} \in Y^k$ such that $\hat{m} \rightarrow m$, even if $[\hat{m}]f = 0$ then also $[\hat{m}]f^* = m! \cdot [m]f$. \square

Next, we apply Lemma 9 to compute scaled Hadamard product in the commutative setting via noncommutative Hadamard product.

Remark 7. We note that given a commutative circuit C computing f , the noncommutative polynomial \hat{f} computed by \mathfrak{C} depends on the circuit structure of C . In other words, there can be many \hat{f} corresponding to f . However, Lemma 9 shows that f^* depends only on the polynomial f .

Lemma 10. Let $g \in \mathbb{F}[X]$ be a homogeneous degree- k polynomial and C be some arithmetic circuit for g . For any homogeneous degree- k polynomial $f \in \mathbb{F}[X]$ and any point $\underline{a} \in \mathbb{F}^n$

$$(f \circ^s g)(\underline{a}) = (f^* \circ \hat{g})(\underline{a}),$$

where the noncommutative polynomial \hat{g} is defined by the given circuit C .

Proof. We write $f = \sum_m [m]f \cdot m$ and $g = \sum_{m'} [m']g \cdot m'$. By definition, we have $f \circ^s g = \sum_m m! \cdot [m]f \cdot [m]g \cdot m$.

The polynomial computed by \mathfrak{C} is $\hat{g}(Y) = \sum_{m \in X_k} \sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{g} \cdot \hat{m}$. By Lemma 9, the noncommutative polynomial $f^*(Y) = \sum_{m \in X_k} \sum_{\hat{m} \rightarrow m} m! \cdot [m]f \cdot \hat{m}$. Therefore,

$$(f^* \circ \hat{g})(Y) = \sum_{m \in X_k} \sum_{\hat{m} \rightarrow m} m! \cdot [m]f \cdot [\hat{m}]\hat{g} \cdot \hat{m} = \sum_{m \in X_k} m! \cdot [m]f \cdot \left(\sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{g} \right) \cdot \hat{m}.$$

Consequently, for any point $\underline{a} \in \mathbb{F}^n$ we have

$$(f^* \circ \hat{g})(\underline{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot \left(\sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{g} \right) \cdot \hat{m}(\underline{a}).$$

Since $[m]g = \sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{g}$, we have

$$(f^* \circ \hat{g})(\underline{a}) = \sum_{m \in X_k} m! \cdot [m]f \cdot m(\underline{a}) \cdot [m]g = (f \circ^s g)(\underline{a}).$$

□

We note an immediate corollary of the above.

Corollary 5. *Let f_1, f_2 be homogeneous degree- k polynomials in $\mathbb{F}[X]$. Given a noncommutative circuit C computing the polynomial $\hat{f}_1 \circ f_2^* \in \mathbb{F}\langle Y \rangle$, one can obtain a commutative circuit \tilde{C} for $f_1 \circ^s f_2 \in \mathbb{F}[X]$ by replacing the noncommutative variables y_i in C by the commutative variables x_i .*

Proof. Let $f_1 = \sum_m [m]f_1 \cdot m_1$. So, $\hat{f}_1 = \sum_{\hat{m}} [\hat{m}]\hat{f}_1 \cdot \hat{m}$ and $[m]f = \sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{f}$.

Then, $\hat{f}_1 \circ f_2^*(Y) = \sum_{\hat{m}} [\hat{m}]\hat{f}_1 \cdot [\hat{m}]f_2^* \cdot \hat{m} = \sum_{\hat{m}} [\hat{m}]\hat{f}_1 \cdot m! [m]f_2 \cdot \hat{m}$ where $\hat{m} \rightarrow m$.

Now replacing the noncommutative variables by commutative variables, we obtain

$$\hat{f}_1 \circ f_2^*(X) = \sum_m m! \cdot \left(\sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{f}_1 \right) \cdot [m]f_2 \cdot m$$

Since, $[m]f = \sum_{\hat{m} \rightarrow m} [\hat{m}]\hat{f}$, we further simplify and get $\hat{f}_1 \circ f_2^*(X) = f_1 \circ^s f_2(X)$. □

Remark 8. *A key conceptual tool in [\[Pra18\]](#) is the apolar inner product for homogeneous degree- k polynomials f and g in $\mathbb{F}[X]$, which is defined as*

$$\langle f, g \rangle = f(\partial_{x_1}, \dots, \partial_{x_n}) \circ g(x_1, \dots, x_n).$$

We note that in the Hadamard product framework, we can express the apolar inner

product of f and g as $f \circ^s g$ evaluated at the all-ones vector $\mathbf{1} \in \mathbb{F}^n$. In Section [5.5](#) we present more details.

5.2 The Sum of Coefficients of Multilinear Monomials

The main result of this section is the following.

Theorem 16. *The (k, n) -MLC problem for an input polynomial in $\mathbb{F}[x_1, x_2, \dots, x_n]$, given as input by an algebraic branching program of size s , has a deterministic $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ -time algorithm. When f is given as input by an arithmetic circuit C of size s , has a deterministic $O^*\left(\binom{n}{\lfloor k/2 \rfloor} \cdot s^{c \cdot \log k}\right)$ -time algorithm where c is a constant.*

For the above theorem, the underlying field \mathbb{F} could be any field whose elements can be efficiently represented, with efficiently computable field operations. We note that the above run time beats the naive $O^*(n^k)$ bound, answering the question asked by Koutis and Williams [\[KW16\]](#). The notation $\binom{n}{\lfloor i \rfloor}$ stands for $\sum_{j=0}^i \binom{n}{j}$.

The main idea is to apply the symmetrization trick to reduce the (k, n) -MLC problem to the evaluation of the rectangular permanent over a suitable matrix ring. Then we use a result of [\[BHKK10\]](#) to solve the instance of the rectangular permanent evaluation problem.

Before we present the results we recall the definition of ABPs ([Definition 2](#)) and the following different but equivalent formulation of ABPs:

A homogeneous ABP of width w computing a degree- k polynomial over X can be thought of as the $(1, w)^{th}$ entry of the product of $w \times w$ matrices $M_1 \cdots M_k$ where entries of each M_i are homogeneous linear forms over X . By $[x_j]M_i$, we denote the $w \times w$ matrix over \mathbb{F} , such that $(p, q)^{th}$ entry of the

matrix, $([x_j]M_i)(p, q) = [x_j](M_i(p, q))$, the coefficient of x_j in the linear form of the $(p, q)^{th}$ entry of M_i .

Permanent of Rectangular Matrices

We now define the permanent of a rectangular matrix. The permanent of a rectangular $k \times n$ matrix $A = (a_{ij}), k \leq n$, with entries over a ring R is defined as

$$\text{rPer}(A) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k a_{i,\sigma(i)},$$

where $I_{k,n}$ is the set of all injections from $[k]$ to $[n]$. We define the noncommutative polynomial $S_{n,k}^*$ as

$$S_{n,k}^*(y_1, y_2, \dots, y_n) = \sum_{T \subseteq [n]: |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)},$$

which is the symmetrized version of the elementary symmetric polynomial $S_{n,k}$ as defined in Lemma 9. Given a set of $t \times t$ matrices M_1, \dots, M_n over some field \mathbb{F} define the rectangular (block) matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ such that $a_{i,j} = M_j$. Thus, A is a $k \times n$ matrix with entries from the ring of $t \times t$ matrices over the field \mathbb{F} . The following observation is crucial.

Observation 3. $S_{n,k}^*(M_1, \dots, M_n) = \text{rPer}(A)$.

Proof. To see this, observe that ,

$$\text{rPer}(A) = \sum_{T \subseteq [n]: |T|=k} \text{Per}(A_T) = \sum_{T \subseteq [n]: |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} M_{\sigma(i)}.$$

Here A_T is the minor of A such that the columns are indexed by the set T . □

In the sequel, we will apply a result from [BHKK10], showing that over *any* ring R ,

the permanent of a rectangular $k \times n$ matrix can be evaluated with $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ ring operations. In particular, if R is the matrix ring $\text{Mat}_s(\mathbb{F})$, the algorithm runs in time $O(k \binom{n}{\lfloor k/2 \rfloor} \text{poly}(n, s))$. We now present the proof of Theorem [16](#)

Proof. We first prove a special case of the theorem when the polynomial f is given by a homogeneous degree- k ABP B of width w . Notice that we can compute the sum of the coefficients of the degree- k multilinear terms in f by evaluating $(f \circ S_{n,k})(\mathbf{1})$. Now to compute the Hadamard product efficiently, we will transfer the problem to the noncommutative domain. Let \mathfrak{B} define the noncommutative version of the commutative ABP B for the polynomial f . By Lemma [10](#), it suffices to compute $(\mathfrak{B} \circ S_{n,k}^*)(\mathbf{1})$. Now, the following lemma reduces this computation to evaluating $S_{n,k}^*$ over a suitable matrix ring. We recall the following result from [AS18](#) (see, Chapter [2](#) for more details).

Lemma 11. (Theorem 2 of [AS10](#)) *Let f be a homogeneous degree- k noncommutative polynomial in $\mathbb{F}\langle Y \rangle$ and B be an ABP of width w computing a homogeneous degree- k polynomial $g = (M_1 \cdots M_k)(1, w)$ in $\mathbb{F}\langle Y \rangle$.*

Then $(f \circ g)(\mathbf{1}) = (f(A_1^B, \dots, A_n^B))(1, (k+1)w)$ where for each $i \in [n]$, A_i^B is the following $(k+1)w \times (k+1)w$ block superdiagonal matrix,

$$A_i^B = \begin{bmatrix} 0 & [y_i]M_1 & 0 & \dots & 0 \\ 0 & 0 & [y_i]M_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & [y_i]M_k \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

Now, we construct a $k \times n$ rectangular matrix $A = (a_{i,j})_{i \in [k], j \in [n]}$ from the ABP \mathfrak{B} (obtained from the given ABP B) by setting $a_{i,j} = A_j^{\mathfrak{B}}$ as defined. Using

Observation [3](#) we now have,

$$\text{rPer}(A)(1, (k+1)w) = S_{n,k}^*(A_1^{\mathfrak{B}}, \dots, A_n^{\mathfrak{B}})(1, (k+1)w).$$

Now by Lemma [10](#) and Lemma [11](#), we conclude that,

$$S_{n,k}^*(A_1^{\mathfrak{B}}, \dots, A_n^{\mathfrak{B}})(1, (k+1)w) = (S_{n,k}^* \circ \mathfrak{B})(\underline{1}) = (S_{n,k} \circ^s B)(\underline{1}).$$

Hence applying the algorithm of Björklund et al. for evaluating the rectangular permanent over noncommutative ring [\[BHKK10\]](#), we compute the sum of the coefficients deterministically in time $O(k \binom{n}{\lfloor k/2 \rfloor} \text{poly}(n, k, w))$.

Now, we prove the general case. We apply a standard transformation from circuits to ABPs [\[VSB83, SY10\]](#) and reduce the problem to the ABP case. More precisely, given an arithmetic circuit of size s computing a polynomial f of degree k , f can also be computed by a homogeneous ABP of size $s^{O(\log k)}$. In particular, if f is given by an arithmetic circuit of size s , we first get a circuit of $\text{poly}(k, s)$ size for degree- k part of f using a standard method of homogenization [\[SY10, Theorem 2.2\]](#). Then, we convert the homogeneous circuit to a homogeneous ABP of size $s^{O(\log k)}$. The width w of the new ABP is also bounded by $s^{O(\log k)}$. Next, we apply the first part of the proof to the newly constructed ABP. Notice that the entire computation can be done in deterministic $O(k \binom{n}{\lfloor k/2 \rfloor} \text{poly}(n, k, w))$ time which is $O^*\left(\binom{n}{\lfloor k/2 \rfloor} \cdot s^{c \cdot \log k}\right)$ for some constant c . \square

5.2.1 Some Applications

We show some applications of Theorem [16](#) and the technique developed there. The first two are algorithmic applications and the last is a hardness result.

Applying Theorem [16](#), we improve the counting complexity of two combinatorial

problems studied in [KW16]. To the best of our knowledge, nothing better than the brute-force exhaustive search algorithms were known for the counting version of these problems. We start with the k -Tree problem: Let T be a tree with k vertices and G be an n -vertex graph. A *homomorphic embedding* of T in G is defined by an injective map $\varphi : V(T) \rightarrow V(G)$ such that for all $u, v \in V(T)$

$$uv \in E(T) \implies \varphi(u)\varphi(v) \in E(G).$$

A *copy* of T in G is $\varphi(T)$ for some homomorphic embedding φ of T in G . Let $e_{T,G}$ denote the number of homomorphic embeddings of T in G and $c_{T,G}$ denote the number of copies of T in G . Let $\text{Aut}(T)$ denote the automorphism group of T . For two such homomorphic embeddings φ_1 and φ_2 we have $\varphi_1(T) = \varphi_2(T)$ if and only if $\varphi_1^{-1}\varphi_2 \in \text{Aut}(T)$. Hence,

$$(5.1) \quad c_{T,G} = \frac{e_{T,G}}{|\text{Aut}(T)|}.$$

The exact counting version of k -Tree is the problem of counting the number of copies of T in G for an input pair (T, G) . Clearly, there is a trivial $O^*(n^k)$ exhaustive search algorithm for the problem. We apply Theorem [16] to obtain an essentially quadratic speed-up.

Corollary 6. *The exact counting k -Tree problem of counting the number of copies of a given k -vertex tree in a given n -vertex graph can be computed in deterministic $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ time.*

Proof. By Equation [5.1] it suffices to count the number of homomorphic embeddings of T in G , as counting the number of automorphisms $|\text{Aut}(T)|$ of T can be done in $\text{poly}(k)$ time.

To this end, we will use a modification of the construction defined in [KW16, Theorem 2.2]. Let the nodes of T be $\{1, 2, \dots, k\}$ and the nodes of G be $\{1, 2, \dots, n\}$. W.l.o.g., we can consider T to be a tree rooted at 1. As a consequence, every node $i \in [k]$ of T uniquely defines a subtree T_i rooted at i . Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of n commuting variables corresponding to the n vertices of G . We inductively define a polynomial C_{ij} in $\mathbb{F}[X]$ as follows:

- If i is a leaf node of the tree T then $C_{ij} = x_j$.
- Otherwise, let i_1, i_2, \dots, i_ℓ be the children of i in T . Inductively, we can assume that the polynomials $C_{i_t, j}$, $1 \leq t \leq \ell$, $1 \leq j \leq n$ are already defined.

We define

$$C_{ij} = x_j \prod_{t=1}^{\ell} \left(\sum_{j': (j, j') \in E(G)} C_{i_t, j'} \right).$$

Finally, we define the polynomial Q as

$$Q(X) = \sum_{j=1}^n C_{1j}.$$

By definition, it follows that C_{ij} is a homogeneous polynomial of degree $|V(T_i)|$ for each $i \in [k]$. Consequently, $Q(X)$ is a homogeneous degree- k polynomial.

Claim 4. *Let $i \in [k]$ and the subtree T_i have r nodes. Then the number of degree- r multilinear monomials in C_{ij} is exactly the number of homomorphic embeddings of T_i in G that maps i to j . Hence, the number of degree- r multilinear monomials in $\sum_{j \in V(G)} C_{ij}$ is the number of homomorphic embeddings of T_i in G .*

The above claim is easily proved by induction on the size of T_i . It is clearly true for $|V(T_i)| = 1$. In general, suppose i_1, i_2, \dots, i_ℓ are the children of i in tree T . Any homomorphic embedding $\varphi : T_i \rightarrow G$ that maps i to j is defined uniquely by homomorphic embeddings $\varphi_t : T_{i_t} \rightarrow G$ such that the ranges of the φ_t are all disjoint

and the i_t map to distinct G -neighbors of j . Clearly, from the definition of C_{ij} and induction, it follows that there is a unique multilinear monomial of C_{ij} that corresponds to φ . Conversely, each multilinear monomial defines a unique homomorphic embedding $\varphi : T_i \rightarrow G$ that maps i to j .

Thus, the exact counting k -Tree problem is solved by counting the number of multilinear monomials in Q . By Theorem [16](#), it suffices to construct for Q an ABP of size $\text{poly}(n, k)$.

From the recursively defined structure of the noncommutative formula for C_{ij} we can analyze the size. Let size_ℓ bound the noncommutative formula size for the polynomial Q_ℓ defined as above for trees of size ℓ (note that $Q_k = Q$). We note from the formula structure that $\text{size}_1 = n$ and

$$\text{size}_k = nk + \sum_{t=1}^{\ell} \text{size}_{k_t},$$

where k_t are the subtree sizes and $k_1 + k_2 + \dots + k_\ell = k - 1$. Clearly, $\text{size}_k \leq nk^2$. Thus, Q has a formulas of size at most nk^2 which can be converted to an ABP of size $O(nk^2)$ by standard techniques [\[SY10\]](#). \square

We now consider the exact counting version of the m -Dimensional k -Matching problem: Let U_1, U_2, \dots, U_m be mutually disjoint sets, and let C be a collection of m -tuples from their cartesian product $U_1 \times \dots \times U_m$. An m -dimensional k -matching in C is a subcollection of k many m -tuples such that no two of these m -tuples share a common coordinate. Koutis and Williams [\[KW16\]](#) obtain a faster parameterized algorithm for the decision version of this problem. We present an exact counting algorithm as an application of Theorem [16](#).

Corollary 7. *Given mutually disjoint sets $U_i, i \in [m]$, and a collection C of m -tuples from $U_1 \times \dots \times U_m$, we can count the number of m -dimensional k -matchings in C in deterministic $O^* \left(\binom{n}{\lfloor (m-1)k/2 \rfloor} \right)$ time.*

Proof. Following [KW16], encode each element u in $U = \cup_{i=2}^m U_i$ by a variable $x_u \in X$. Encode each m -tuple $t = (u_1, \dots, u_m) \in C \subseteq U_1 \times \dots \times U_m$ by the monomial $M_t = \prod_{i=2}^m x_{u_i}$. Assume $U_1 = \{u_{1,1}, \dots, u_{1,n}\}$, and let $T_j \subseteq C$ denote the subset of m -tuples whose first coordinate is $u_{1,j}$. Consider the polynomial,

$$P(X, z) = \prod_{j=1}^n \left(1 + \sum_{t \in T_j} (z \cdot M_t) \right)$$

Clearly, $P(X, z)$ has an ABP of size $\text{poly}(n, m, |C|)$. Let $Q(X) = [z^k]P(X, z)$. In polynomial time we can obtain an ABP of size $\text{poly}(n, m, |C|)$ for $Q(X)$ by the standard method of Vandermonde matrix based interpolation on the variable z . Clearly, $Q(X)$ is a homogeneous degree- km polynomial and the nonzero multilinear monomials of $Q(X)$ are in 1 – 1 correspondence with the m -dimensional k -matchings in C . Therefore, the number of multilinear terms in $Q(X)$ is the required count. We can now apply the first part of Theorem [16] to count the number of multilinear terms in $Q(X)$.

□

Hardness of the rectangular permanent over general rings

In [BHKK10], it is shown that the $k \times n$ rectangular permanent can be evaluated over commutative rings and commutative semirings in $O(h(k) \cdot \text{poly}(n, k))$ time for some computable function h . In other words, the problem is in FPT, parameterized by the number of rows. An interesting question is to ask whether one can get any FPT algorithm when the entries are from noncommutative rings (in particular, matrix rings). We prove that such an algorithm is unlikely to exist. We show that counting the number of k -paths in a graph G , a well-known #W[1]-complete problem, reduces to this problem. So, unless ETH fails we do not have such an algorithm [DF13].

Theorem 17. *Given a $k \times n$ matrix X with entries $x_{ij} \in \text{Mat}_{t \times t}(\mathbb{Q})$, computing the rectangular permanent of X is $\#\mathbf{W}[1]$ -hard with k as the parameter where $t = (k + 1)n$ under polynomial-time many-one reduction.*

Proof. If we have an algorithm to compute the permanent of a $k \times n$ matrix over noncommutative rings which is FPT in parameter k , that yields an algorithm which is FPT in k for evaluating the polynomial $S_{n,k}^*$ on matrix inputs. This follows from Observation [3](#). Now, given a graph G we can compute a homogeneous ABP of width n and k layers for the graph polynomial C_G defined as follows.

Let $G(V, E)$ be a directed graph with n vertices where $V(G) = \{v_1, v_2, \dots, v_n\}$. A k -walk is a sequence of k vertices $v_{i_1}, v_{i_2}, \dots, v_{i_k}$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $1 \leq j \leq k - 1$. A k -path is a k -walk where no vertex is repeated. Let A be the adjacency matrix of G , and let y_1, y_2, \dots, y_n be noncommuting variables. Define an $n \times n$ matrix B

$$B[i, j] = A[i, j] \cdot y_i, \quad 1 \leq i, j \leq n.$$

Let $\underline{1}$ denote the all 1's vector of length n . Let \underline{y} be the length n vector defined by $\underline{y}[i] = y_i$. The graph polynomial $C_G \in \mathbb{F}\langle Y \rangle$ is defined as

$$C_G(Y) = \underline{1}^T \cdot B^{k-1} \cdot \underline{y}.$$

Let W be the set of all k -walks in G . The following observation is folklore.

Observation 4.

$$C_G(Y) = \sum_{v_{i_1} v_{i_2} \dots v_{i_k} \in W} y_{i_1} y_{i_2} \dots y_{i_k}.$$

Hence, G contains a k -path if and only if the graph polynomial C_G contains a multilinear term.

Clearly the number of k -paths in G is equal to $(C_G \circ S_{n,k})(\underline{1})$. By Lemma [10](#), we know that it suffices to compute $(\mathfrak{C}_G \circ S_{n,k}^*)(\underline{1})$. We construct $kn \times kn$ matrices

A_1, \dots, A_n from the ABP of \mathfrak{C}_G following Lemma [11](#). Then from Lemma [11](#) we know that $(\mathfrak{C}_G \circ S_{n,k}^*)(1) = S_{n,k}^*(A_1, \dots, A_n)(1, t)$ where $t = (k+1)n$. So if we have an algorithm which is FPT in k for evaluating $S_{n,k}^*$ over matrix inputs, we also get an algorithm to count the number of k -paths in G in $\text{FPT}(k)$ time. \square

5.3 Multilinear Monomial Detection

The next algorithmic result we obtain is the following.

Theorem 18. *The k -MMD problem for any arithmetic circuit C of $\text{poly}(n)$ size, has a randomized $O^*(4.32^k)$ -time and polynomial space-bounded algorithm.*

Again, the underlying field \mathbb{F} could be any field whose elements can be efficiently represented, with efficiently computable field operations. First, we give an algorithm for computing the Hadamard product for a special case in the commutative setting. Any depth two $\Pi^{[k]}\Sigma$ circuit computes the product of k homogeneous linear forms over the input set of variables X .

Lemma 12. *Given an arithmetic circuit C of size s computing $g \in \mathbb{F}[X]$, and a homogeneous $\Pi^{[k]}\Sigma$ circuit computing $f \in \mathbb{F}[X]$, and any point $\underline{a} \in \mathbb{F}^n$, we can evaluate $(f \circ^s g)(\underline{a})$ in $O^*(2^k)$ time and in polynomial space.*

Proof. By standard homogenization technique [[SY10](#), Theorem 2.2] we can extract the homogeneous degree- k component of C and thus we can assume that C computes a homogeneous degree- k polynomial. Write $f = \prod_{j=1}^k L_j$, for homogeneous linear forms L_j . The corresponding noncommutative polynomial \hat{f} is defined by the natural order of the j indices (and replacing x_i by y_i for each i).

Claim 5. *The noncommutative polynomial f^* has a (noncommutative) $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ circuit, which we can write as $f^* = \sum_{i=1}^{2^k} C_i$, where each C_i is a (noncommutative) $\Pi^{[k]}\Sigma$ circuit.*

Before we prove the claim, we show that it easily yields the desired algorithm. First, we notice that

$$\mathfrak{C} \circ f^* = \sum_{i=1}^{2^k} \mathfrak{C} \circ C_i.$$

Now, by Theorem [6](#) we can compute in $\text{poly}(n, s, k)$ time a $\text{poly}(n, s, k)$ size circuit for the (noncommutative) Hadamard product $\mathfrak{C} \circ C_i$. As argued in the proof of Lemma [10](#), for any $\underline{a} \in \mathbb{F}^n$ we have

$$(g \circ^s f)(\underline{a}) = (C \circ^s f)(\underline{a}) = (\mathfrak{C} \circ f^*)(\underline{a}).$$

Thus, we can evaluate $(g \circ^s f)(\underline{a})$ by incrementally computing $(\mathfrak{C} \circ C_i)(\underline{a})$ and adding up for $1 \leq i \leq 2^k$. This can be clearly implemented using only polynomial space. \square

Proof of Claim [5](#) By definition,

$$f^* = \sum_{\sigma \in S_k} \hat{L}_{\sigma(1)} \hat{L}_{\sigma(2)} \cdots \hat{L}_{\sigma(k)}.$$

Now define the $k \times k$ matrix T such that the elements in each row of T are the linear forms $\hat{L}_1, \hat{L}_2, \dots, \hat{L}_k$ in this order. Then the (noncommutative) permanent of T is given by

$$\text{Perm}(T) = \sum_{\sigma \in S_k} \prod_{j=1}^k \hat{L}_{\sigma(j)}$$

which is just f^* .

We will now apply Ryser's formula [Rys63](#) to express $\text{Perm}(T)$ as a depth-3 homogeneous noncommutative $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula. We recall Ryser's formula [Rys63](#) for $\text{Per}(A)$, where A is a $k \times k$ matrix with noncommuting entries A_{ij} :

$$\text{Per}(A) = \sum_{S \subseteq [k]} (-1)^{|S|} \prod_{i=1}^k \sum_{j \in S} A_{ij}.$$

It is a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula for the $k \times k$ noncommutative permanent. Now,

substituting \hat{L}_j for A_{ij} $1 \leq i, j \leq k$, it follows that $f^* = \text{Per}(T)$ has a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ noncommutative formula. Note that Ryser's formula is usually given for the commutative permanent. It is easy to observe that the same proof, based on the principle of inclusion-exclusion, also holds for the noncommutative permanent. \square

Remark 9. *Over the rationals, we can get an alternative proof of Lemma [12](#) by using Fischer's identity [\[Fis94\]](#) to obtain a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ formula for f^* .*

Now we are ready to prove Theorem [18](#).

Proof. By homogenization, we can assume that C computes a homogeneous degree k polynomial f .

We will refer to maps $\zeta : [n] \rightarrow [k]$ as *coloring maps*. The map ζ can be seen as assigning colors [2](#) to the elements of $[n]$: $\zeta(i)$ is the *color* of i .

We will pick a collection of coloring maps $\{\zeta_i : [n] \rightarrow [k]\}$ each picked independently and uniformly at random. For each coloring map ζ_i we define a $\Pi^{[k]}\Sigma$ formula

$$P_i = \prod_{j=1}^k \sum_{\ell: \zeta_i(\ell)=j} x_\ell.$$

A monomial is *covered* by a coloring map ζ_i if the monomial is nonzero in P_i . The probability that a random coloring map covers a given degree- k multilinear monomial is

$$k! \cdot k^{-k} \approx e^{-k}.$$

Hence, for a collection \mathcal{C} of $O^*(e^k)$ many coloring maps $\mathcal{C} = \{\zeta_i : [n] \rightarrow [k]\}$ picked independently and uniformly at random, it holds with a constant probability that every multilinear monomial of degree k is covered at least once by some ζ_i in \mathcal{C} .

²The terminology is in keeping with a seminal paper in the field [\[AYZ95\]](#) which introduced the color-coding technique. However, it should be clear that this notion of coloring has nothing to do with graph colorings.

This probability bound is by a simple and standard union bound argument. Now, for each coloring map $\zeta_i \in \mathcal{C}$ we consider the circuit $C'_i = C \circ^s P_i$.

Notice that for each multilinear monomial m , the multiplicative factor $m!$ is 1. Also, the coefficient of each monomial is exactly 1 in each P_i , and if f contains a multilinear term then it is covered by *some* P_i . Now, we perform the randomized polynomial identity test on each circuit C'_i by applying the Polynomial Identity Lemma [DL78] [Zip79], [Sch80] in randomized polynomial time to complete the procedure. More precisely, we pick a random $\underline{a} \in \mathbb{F}^n$ and evaluate C'_i at \underline{a} to check if it is nonzero. By Lemma [12], the computation of $C'_i(\underline{a})$ can be done deterministically in time $O^*(2^k)$ time and $\text{poly}(n, k)$ space³. Hence the total running time of the procedure is $O^*((2e)^k)$.

In order to improve the running time to $O^*(4.32^k)$, we apply the color-coding technique of Hüffner et al. [HWZ08]. The idea is to use more than k colors to reduce the number of coloring maps required to cover the degree- k monomials. But this would increase the formal degree of each depth two circuit which we need to handle.

We will use $1.3k$ colors⁴ and each circuit P_i will now be a $\Pi^{[1.3k]}\Sigma$ circuit. For each coloring map $\zeta_i : [n] \rightarrow [1.3k]$ chosen uniformly at random, we define the following $\Pi^{[1.3k]}\Sigma$ circuit

$$P_i(x_1, x_2, \dots, x_n, z_1, \dots, z_{1.3k}) = \prod_{j=1}^{1.3k} \left(\sum_{\ell: \zeta_i(\ell)=j} x_\ell + z_j \right).$$

Since each P_i is of degree $1.3k$, we need to modify the circuit C to another circuit C' of degree $1.3k$ in order to apply Hadamard products. To that end, we define the

³Since the syntactic degree of the circuit is not bounded here, and if we have to account for the bit level complexity (over \mathbb{Z}) of the scalars generated in the intermediate stage we may get field elements whose bit-level complexity is exponential in the input size. So, a standard technique is to take a random prime of polynomial bit-size and evaluate the circuit modulo that prime.

⁴By $1.3k$ and $0.3k$, we mean the integers $\lceil 1.3k \rceil$ and $\lceil 0.3k \rceil$, respectively.

circuit $C' \in \mathbb{F}[X, Z]$ as follows

$$C'(X, Z) = C(X) \cdot S_{1.3k, 0.3k}(z_1, \dots, z_{1.3k}),$$

where $S_{1.3k, 0.3k}(z_1, \dots, z_{1.3k})$ is the elementary symmetric polynomial of degree $0.3k$ over the variables $z_1, \dots, z_{1.3k}$. By the result of [HWZ08], for $O^*(1.752^k)$ random coloring maps, with high probability each multilinear monomial in C is covered by the monomials of some P_i (over the X variables).

Now to compute $\widehat{C}' \circ P_i^*$ for each i , we symmetrize the polynomial P_i . Of course, the symmetrization happens over the X variables as well as over the Z variables. But in \widehat{C}' we are only interested in the monomials (or words) where the rightmost $0.3k$ variables are over Z variables. In the noncommutative circuit \widehat{C}' , every subword $z_{i_1} z_{i_2} \dots z_{i_{0.3k}}$ receives a natural ordering $i_1 < i_2 < \dots < i_{0.3k}$.

Notice that

$$P_i^*(Y, Z) = \sum_{\sigma \in S_{1.3k}} \prod_{j=1}^{1.3k} \left(\sum_{\ell: \zeta_i(\ell) = \sigma(j)} y_\ell + z_{\sigma(j)} \right).$$

Our goal is to understand the part of $P_i^*(Y, Z)$ where each monomial ends with a subword of the form $z_{i_1} z_{i_2} \dots z_{i_{0.3k}}$ and the top k symbols are over the X variables. For a fixed set of indices $W = \{i_1 < i_2 < \dots < i_{0.3k}\}$, define the set $T = [1.3k] \setminus W$. Let $S_{[k], T}$ be the set of permutations $\sigma \in S_{1.3k}$ such that $\sigma : [k] \rightarrow T$ and $\sigma(k+j) = i_j$ for $1 \leq j \leq 0.3k$. As we have fixed the last $0.3k$ positions, each $\sigma \in S_{[k], T}$ corresponds to some $\sigma' \in S_k$. Let $Z_W = z_{i_1} z_{i_2} \dots z_{i_{0.3k}}$. Now we notice the following.

Observation 5. *The part of $P_i^*(Y, Z)$ where each monomial ends with the subword Z_W is $P_{i, W}^* \cdot Z_W$, where*

$$P_{i, W}^*(Y) = \sum_{\sigma \in S_{[k], T}} \prod_{j=1}^k \left(\sum_{\ell: \zeta_i(\ell) = \sigma(j)} y_\ell \right) = \sum_{\sigma' \in S_k} \prod_{j=1}^k \left(\sum_{\ell: \zeta_i(\ell) = \sigma'(j)} y_\ell \right).$$

Now, just like the case above, it suffices to perform polynomial identity testing for

$$(\widehat{C}' \circ P_i^*)(Y, Z) = \sum_{W \subseteq [1.3k]: |W|=0.3k} (\mathfrak{C}(Y) \circ P_{i,W}^*(Y)) \cdot Z_W.$$

for each i . But this is same as testing $C' \circ P_i$ for identity. Now we eliminate the Z variables by substituting 1 and evaluate X variables on a random point $\underline{a} \in \mathbb{F}^n$. By Lemma [12](#), $(C' \circ P_i)(\underline{a}, \underline{1})$ can be computed in $O^*(2^{1.3k}) = O^*(2.46^k)$ time and $\text{poly}(n, k)$ space. The bound on the success probability follows from Polynomial Identity Lemma [\[DL78\]](#) [\[Zip79\]](#) [\[Sch80\]](#).

We repeat the above procedure for each coloring map and obtain a randomized $O^*(4.32^k)$ algorithm. This completes the proof of Theorem [18](#). \square

5.4 Deterministic Algorithms for Depth Three Circuits

Next, we state the results showing fast *deterministic* algorithms for depth three circuits. We use the notation $\Sigma^{[s]}\Pi^{[k]}\Sigma$ to denote depth three circuits of top Σ gate fan-in s and the Π gates compute the product of k homogeneous linear forms over X .

Theorem 19. *Given any homogeneous depth three $\Sigma^{[s]}\Pi^{[k]}\Sigma$ circuit of degree k , the (k, n) -MLC problem can be solved in deterministic $O^*(2^k)$ -time. Over \mathbb{Z} , the k -MMD problem can be solved in deterministic $O^*(4^k)$ -time. Over finite fields, k -MMD problem can be solved in deterministic $e^k k^{O(\log k)} O^*(2^{ck} + 2^k)$ time, where $c \leq 5$.*

It is well-known that the elementary symmetric polynomial $S_{n,k}$ can be computed using an ABP of size $\text{poly}(n, k)$. Here the key observation is that we can efficiently compute the commutative Hadamard product of a depth three circuit with *any* circuit. These are obtained by a simple application of Hadamard product combined

with symmetrization. We will require the following.

Theorem 20. [AJS09, Theorem 4] *Let A and B be noncommutative ABPs of sizes s_1 and s_2 , computing homogeneous degree- k polynomials $f_1, f_2 \in \mathbb{F}\langle Y \rangle$, respectively. Then, we can compute an ABP of size $O(s_1 s_2)$ for the Hadamard product $f_1 \circ f_2$ in deterministic $\text{poly}(s_1, s_2)$ time. Furthermore, if A and B are $\Pi^{[k]}\Sigma$ circuits, then we can compute a $\Pi^{[k]}\Sigma$ circuit for $f_1 \circ f_2$ in $\text{poly}(s_1, s_2)$ time.*

We now prove Theorem [19].

Proof. Let C be the given $\Sigma^{[s']}\Pi^{[k]}\Sigma$ circuit computing the polynomial $f \in \mathbb{F}[X]$.

We first consider the k -MMD problem.

Suppose the coefficients are integers (without loss of generality, if we assume the underlying field is rationals). Let $C = \sum_{i=1}^{s'} C_i$ where each C_i is a $\Pi^{[k]}\Sigma$ circuit. We obtain a circuit for $C \circ^s C(X)$ as follows. By Corollary [5] it suffices to obtain a circuit for $\mathfrak{C} \circ C^*(Y)$. Notice that $C^* = \sum_{i=1}^{s'} C_i^*$ and by Claim [5] we know that each C_i^* is a $\Sigma^{[2^k]}\Pi^{[k]}\Sigma$ circuit which can be computed in $O^*(2^k)$ time. By distributivity, the problem of computing $\mathfrak{C} \circ C^*(Y)$ reduces to computing the noncommutative Hadamard product of $s' \cdot 2^k$ many pairs of depth-two $\Pi^{[k]}\Sigma$ circuits. By Theorem [20], each such Hadamard product can be computed in $\text{poly}(n, k)$ time. Hence, we obtain a depth three commutative $\Sigma^{[s' \cdot 2^k]}\Pi^{[k]}\Sigma$ circuit \tilde{C} for $C \circ^s C(X)$ in $O^*(2^k)$ time.

Note that m is a nonzero monomial in C if and only if $[m]\tilde{C} > 0$.

Let B be the $\text{poly}(n, k)$ size ABP for $S_{n,k}$. Now the idea is to compute $\tilde{C} \circ^s B(\underline{1})$, and if it is nonzero, we know that C contains a degree- k multilinear term. Again this reduces to computing $s' \cdot 2^k$ scaled Hadamard products, each of the form $\Pi^{[k]}\Sigma \circ^s B(\underline{1})$. By Lemma [12], each such computation can be done in $O^*(2^k)$ time incurring a overall running time $O^*(4^k)$.

In the case of finite fields, the above proof does not work since $\tilde{C} \circ^s B(\underline{1})$ could be zero modulo the characteristic. The idea is similar to the proof of Theorem [18]. But

instead of random coloring maps we pick $\zeta_i : [n] \rightarrow [k]$ from the explicit (n, k) -family perfect hash functions constructed in [NSS95], which is of size $e^k k^{O(\log k)} \log n$, and define a $\Pi^{[k]}\Sigma$ formula

$$P_i = \prod_{j=1}^k \sum_{\ell: \zeta_i(\ell)=j} x_\ell$$

for each coloring map ζ_i . Now for each i , we construct the circuit $C'_i = C \circ^s P_i$. As already explained each $C'_i(X)$ is a $\Sigma^{[s' \cdot 2^k]}\Pi^{[k]}\Sigma$ circuit and it can be obtained in deterministic $O^*(2^k)$ time. Clearly, if C contains a multilinear monomial, we can detect it by doing identity testing of each C'_i . Now we apply a result of Saxena [Sax08] where he shows that the identity testing of a $\Sigma\Pi^{[k]}\Sigma$ circuit over finite fields can be done in deterministic $O^*(2^{ck})$ time where the constant $c \leq 5$. The final running time is $e^k k^{O(\log k)} O^*(2^{ck} + 2^k)$.

As a by-product of the above technique, we get a fast deterministic algorithm to compute the sum of the coefficients of degree- k multilinear monomials in a depth three circuit, solving the (k, n) -MLC problem. Notice that, we need to compute $C \circ^s B(\underline{1})$. As already explained, it can be obtained in deterministic $O^*(2^k)$ time.

□

Remark 10. *The main result of [Sax08, Theorem 1] is actually a polynomial identity testing algorithm for a larger class of circuits. It also uses the identity testing algorithm for noncommutative ABPs [RS05]. Indeed, the bound $c \leq 5$ is from [RS05, Theorem 4].*

5.5 A Comparison to Related Works

We now broadly compare the Hadamard product used in this chapter with the apolar inner product used in the work of Pratt and others [Pra18, Pra19, BP21].

Recall, given two commutative homogeneous degree d polynomials f and g in $\mathbb{F}[X]$,

the apolar inner product $\langle f, g \rangle$ is defined as follows.

$$\langle f, g \rangle = f \left(\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_n} \right) g = \sum_m m! [m] f \cdot [m] g,$$

where the sum is over all degree- d monomials $m = x_1^{e_1} x_2^{e_2} \dots x_n^{e_n} \in X_d$.

The *Waring rank* of a homogeneous degree- d polynomial $f \in \mathbb{F}[X]$ is the least r such that $f = \sum_{j=1}^r \alpha_j \cdot L_j^d$, where for each j , $\alpha_j \in \mathbb{F}$ and L_j is a homogeneous linear form. If f has Waring rank r and g is given by an arithmetic circuit Pratt [Pra18, Pra19] has shown that the apolar inner product of f and g can be computed in $O^*(r)$ time. Hence, using the Waring decomposition of the elementary symmetric polynomials (over the field of rationals) [Lee15] yields faster algorithms [Pra19] for k -MMD and (k, n) -MLC: $O^*(4.08^k)$ -time for k -MMD and $O^*(n^{k/2})$ -time for (k, n) -MLC) over the field of rationals.

However, the Waring decomposition does not appear to have a finite field analogue. For example, over \mathbb{F}_2 the polynomial xy has no Waring decomposition. In comparison, our algorithms are essentially oblivious to the underlying field.

More recently, Brand and Pratt [BP21] have shown that the apolar inner product of commutative polynomials f and g can be computed in $O^*(r)$ time if the partial derivative space of f has rank r . This strengthens Pratt's result [Pra19] as the Waring rank of f is an upper bound on its partial derivative rank.

It is interesting to compare this with the Hadamard product based approach of this chapter. We first note that the apolar inner product of polynomials $f, g \in \mathbb{F}[X]$ can be computed by first computing their scaled Hadamard product (see Section 5.1) and evaluating the resulting polynomial at $x_i = 1$ for each $i \in [n]$. Furthermore, the computation of the Hadamard product of f and g can be done efficiently, as shown in Lemma 10 (Section 5.1), in time polynomial in the noncommutative algebra branching program complexity $B(f^*)$ (i.e. the minimum size of the ABP computing

f^*) of the noncommutative polynomial f^* . The partial derivative space rank of f essentially coincides with $B(f^*)$:

Fact 2. *For any commutative polynomial f , the ABP complexity $B(f^*)$ of f^* is the dimension of the space of partial derivatives of f .*

Proof. Let $f \in \mathbb{F}[X]$ be a homogeneous degree- k commutative polynomial, where $X = \{x_1, x_2, \dots, x_n\}$ is a set of commuting variables. Let $Y = \{y_1, y_2, \dots, y_n\}$ be a corresponding set of free noncommuting variables.

Recall that Nisan [Nis91] has shown that the ABP complexity $B(g)$ of a noncommutative polynomial $g \in \mathbb{F}\langle Y \rangle$ of degree k is exactly $\sum_{\ell=1}^k \text{rank}(M_\ell(g))$ where the matrix $M_\ell(g)$ is defined as follows. The rows of $M_\ell(g)$ are indexed by words in Y^ℓ and the columns of $M_\ell(g)$ are indexed by words in $Y^{k-\ell}$. For some $w_1 \in Y^\ell$ and $w_2 \in Y^{k-\ell}$, the $(w_1, w_2)^{\text{th}}$ entry of $M_\ell(g)$ is $[w_1 w_2]g$.

We now consider the matrix $M_\ell(f^*)$ for the noncommutative polynomial $f^* \in \mathbb{F}\langle Y \rangle$. Let $w_1 \in Y^\ell$ and $w_2 \in Y^{k-\ell}$. Let $\sigma \in S_\ell$ and $\tau \in S_{k-\ell}$ be any permutations. By definition of f^* , it is clear that

$$M_\ell(f^*)(w_1, w_2) = M_\ell(f^*)(w_1^\sigma, w_2^\tau).$$

In particular, the row of $M_\ell(f^*)$ indexed by w_1 is identical to the row indexed by w_1^σ .

Furthermore, the row of $M_\ell(f^*)$ indexed by w_1 has the following structure for any $w_1 \in Y^\ell$: the w_2 -th entry of this row is the same as the w_2^τ -th entry of this row.

Now consider the partial derivative matrix of f , $\tilde{M}_\ell(f)$ defined as follows. The rows of $\tilde{M}_\ell(f)$ are indexed by commuting degree- ℓ monomials in X_ℓ . For some $m \in X_\ell$, the corresponding row is the coefficient vector of $\frac{\partial f}{\partial m}$.

It follows from the above considerations that a subset of rows of $\tilde{M}_\ell(f)$ labeled by monomials m_1, m_2, \dots, m_r are linearly independent if and only if for noncommuting

monomials w_1, w_2, \dots, w_r , such that $w_i \rightarrow m_i$, the rows of row of $M_\ell(f^*)$ indexed by the w_i are linearly independent.

Therefore, $\text{rank}(M_\ell(f^*)) = \text{rank}(\tilde{M}_\ell(f))$, which completes the proof. □

Conclusion

We conclude with the following arithmetic circuit complexity question, a positive answer to which would have interesting algorithmic implications: Is there a polynomial f over the rationals that is *positively* weakly equivalent⁵ to the elementary symmetric polynomial $S_{n,k}$ such that the ABP complexity $B(f^*)$ is $O^*(2^k)$? It would even suffice to show that f^* has arithmetic circuits of size $O^*(2^k)$ to improve on the current best deterministic algorithm for the k -path problem.

⁵A polynomial g is positively weakly equivalent to f if it has the same set of nonzero monomials, with any positive coefficients.

Chapter 6

On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials

We have so far studied the image of noncommutative polynomials in various general settings and computation of Hadamard product in the commutative setting via a reduction to the noncommutative computation in the previous chapters. In this chapter, we initiate the study of *explicit construction* of noncommutative ABPs (see Definition [19](#) for more details). Recall that, the ABP complexity of a noncommutative polynomial is exactly characterized by the rank of Nisan's matrix of the corresponding polynomial [\[Nis91\]](#). The rank of Nisan's matrix is extensively used to show the lower bound for various noncommutative polynomials. From Nisan's characterization, we can argue the existence of an ABP of size equal to the rank of Nisan's matrix. However, explicit construction of such ABPs of the same size was not known before our work. We study the explicit construction of noncommutative ABPs for the rectangular version of permanent and determinant polynomials and some related polynomials.

The rectangular permanent and the rectangular determinant are natural restrictions and well-studied in literature. However, the motivation to study these polynomials also emerges from the last chapter where we have shown that the multilinear monomial counting problem reduces to the evaluation of rectangular permanent over matrix rings. Whereas, for the decision version, we used a weakly equivalent elementary symmetric polynomial. The Hadamard product framework (see Chapter 5) shows that studying the ABP complexity of these polynomials may be useful for a better understanding of these problems.

In this chapter also we often switch between commutative polynomials and noncommutative polynomials. To avoid any confusion, we use X as a set of commuting variables and Y, Z as a set of noncommuting variables.

Explicit Circuit Upper Bound: We now formally define what we mean by *explicit* circuit upper bounds.

Definition 19 (Explicit Circuit Upper Bound). *A family $\{f_n\}_{n>0}$ of degree- k polynomials in the commutative ring $\mathbb{F}[x_1, x_2, \dots, x_n]$ (or the noncommutative ring $\mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$) is $q(n, k)$ -explicit if there is an $O^*(q(n, k))$ time-bounded uniform algorithm \mathcal{A} that on input $\langle 0^n, k \rangle$ outputs a circuit C_n of size $O^*(q(n, k))$ computing f_n .*

The same definition applies to the case of explicit ABP families as well. Notice that, if an ABP is s -explicit then the number of edges in the ABP is $O(s)$. It is important to note that, if we consider the number of nodes as the size of an ABP then an s -size ABP does not imply that the ABP is s -explicit as the number of edges in the ABP can be $O(s^2)$.

6.1 Explicit construction of ABPs for $S_{n,k}^*$ and Noncommutative Rectangular Permanent

We first recall the definition of k^{th} elementary symmetric polynomial $S_{n,k} \in \mathbb{F}[X]$, over the n variables $X = \{x_1, x_2, \dots, x_n\}$,

$$S_{n,k}(X) = \sum_{S \subseteq [n]: |S|=k} \prod_{i \in S} x_i.$$

Recall that $S_{n,k}(X)$ can be computed by an algebraic branching program of size $O(nk)$. In this chapter, we consider again the noncommutative symmetrized version $S_{n,k}^*$, in the ring $\mathbb{F}\langle Y \rangle$ where $Y = \{y_1, \dots, y_n\}$, defined as:

$$S_{n,k}^*(Y) = \sum_{T \subseteq [n]: |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)}.$$

The complexity of the polynomial $S_{n,k}^*$ is first considered by Nisan in his seminal work in noncommutative computation [Nis91]. Nisan shows that any ABP for $S_{n,k}^*$ is of size $\Omega\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ ¹. Furthermore, Nisan also shows the *existence* of an ABP of size $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for $S_{n,k}^*$. However, it is not clear how to construct such an ABP in time $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$. Note that an ABP of size $O^*(n^k)$ for $S_{n,k}^*$ can be directly constructed in $O^*(n^k)$ time by opening up the expression completely. The main upper bound question is whether we can achieve any constant factor saving of the parameter k in the exponent, in terms of size and run time of the construction. In this chapter, we give such an explicit construction of size $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ and the construction takes $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ time. Recall from the previous chapter that the O^* notation suppresses the $\text{poly}(n, k)$ factor. Note that Nisan's result [Nis91] also rules out any $\text{FPT}(k)$ -size ABP for $S_{n,k}^*$, where $\text{FPT}(k)$ means a bound of the form $f(k)n^{O(1)}$.

The next polynomial of our interest is the rectangular permanent polynomial. Given

¹Recall that, $\binom{n}{\lfloor r \rfloor}$ denotes $\sum_{i=0}^r \binom{n}{i}$.

a $k \times n$ rectangular matrix $X = (x_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of commuting variables or a $k \times n$ rectangular matrix $Y = (y_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of noncommuting variables, the rectangular permanent polynomial in commutative and noncommutative domains are defined as follows

$$\text{rPer}(X) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k x_{i,\sigma(i)}, \quad \text{rPer}(Y) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k y_{i,\sigma(i)}.$$

Here, $I_{k,n}$ is the set of all injections from $[k] \rightarrow [n]$. An alternative view is that $\text{rPer}(X) = \sum_{S \subset [n]; |S|=k} \text{Per}(X_S)$ where X_S is the $k \times k$ submatrix where the columns are indexed by the set S . Of course, such a polynomial can be computed in time $O^*(n^k)$ using a circuit of similar size, the main interesting issue is to understand whether the dependence on the parameter k can be improved. It is implicit in the work of Vassilevska and Williams [WW13] that the $\text{rPer}(X)$ polynomial in the commutative setting can be computed by an algebraic branching program of size $O^*(2^k)$. This problem originates from its connection with combinatorial problems studied in the context of exact algorithm design [WW13].

In the noncommutative setting, we start with the polynomial $S_{n,k}^*(Y)$ and then make the polynomial set-multilinear to obtain $\text{rPer}(Y)$. More precisely, we replace each y_i at position j by the variable $y_{j,i}$. With a slight abuse of notation, it is easy to see that the resulting polynomial is $\text{rPer}(Y)$ where Y is a $k \times n$ symbolic matrix of noncommuting variables. Since we already have an explicit ABP construction for $S_{n,k}^*(Y)$ polynomial which is of size $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$, we just make the ABP set-multilinear to obtain an ABP for $\text{rPer}(Y)$ in the *noncommutative setting*. Clearly the size and the time to construct the resulting ABP for $\text{rPer}(Y)$ are bounded by $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$.

Theorem 21.

1. The family of symmetrized elementary polynomials $\{S_{n,k}^*(Y)\}_{n>0}$ has $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABPs over any field.

2. The noncommutative rectangular permanent family $\{\text{rPer}(Y)\}_{n>0}$, where Y is a $k \times n$ symbolic matrix of variables has $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABPs.

Remark 11. We note here that there is an algorithm of run time $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for computing the rectangular permanent over rings and semirings [BHKK10]. Our contribution in the second part of Theorem 21 is that we obtain an $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for it.

Proof. We now present the construction of explicit ABPs for $S_{n,k}^*(Y)$ and noncommutative $\text{rPer}(Y)$.

Construction of ABP for $S_{n,k}^*$: The construction of the ABP for $S_{n,k}^*(Y)$ is inspired by an inclusion-exclusion-based dynamic programming algorithm for the disjoint sum problem [BHKK09].

Let us denote by F the family of subsets of $[n]$ of size exactly $k/2$. Let $\downarrow F$ denote the family of subsets of $[n]$ of size at most $k/2$. For a subset $S \subset [n]$, we define $m_S = \prod_{j \in S} y_j$ where the product is taken in the natural order. Let us define

$$f_S = \sum_{\sigma \in S_{k/2}} \prod_{j=1}^{k/2} y_{i_{\sigma(j)}}$$

where $S \in F$ and $S = \{i_1, i_2, \dots, i_{k/2}\}$, otherwise for subsets $S \notin F$, we define $f_S = 0$. Note that, for each $S \in F$, f_S is the symmetrization of the monomial m_S .

For each $S \in \downarrow F$, let us define $\hat{f}_S = \sum_{A \in F, S \subset A} f_A$ where $A \in F$. We now show, using the inclusion-exclusion principle, that we can express $S_{n,k}^*$ using an appropriate combination of these symmetrized polynomials for different subsets.

Lemma 13.

$$S_{n,k}^* = \sum_{S \in \downarrow F} (-1)^{|S|} \hat{f}_S^2.$$

Proof. Let us first note that, $S_{n,k}^* = \sum_{A \in F} \sum_{B \in F} (A \cap B = \emptyset) f_A f_B$, where we use (P) to denote that the proposition P is true. By the inclusion-exclusion principle:

$$\begin{aligned}
S_{n,k}^* &= \sum_{A \in F} \sum_{B \in F} (A \cap B = \emptyset) f_A f_B \\
&= \sum_{A \in F} \sum_{B \in F} \sum_{S \in \downarrow F} (-1)^{|S|} (S \subseteq A \cap B) f_A f_B \\
&= \sum_{S \in \downarrow F} (-1)^{|S|} \sum_{A \in F} \sum_{B \in F} (S \subseteq A) (S \subseteq B) f_A f_B \\
&= \sum_{S \in \downarrow F} (-1)^{|S|} \left(\sum_{A \in F} (S \subseteq A) f_A \right)^2 = \sum_{S \in \downarrow F} (-1)^{|S|} \hat{f}_S^2.
\end{aligned}$$

□

Now we describe two ABPs where the first ABP simultaneously computes f_A for each $A \in F$ and the second one simultaneously computes \hat{f}_S for each $S \in \downarrow F$.

Lemma 14. *There is an $\binom{n}{\downarrow k/2}$ -explicit multi-output ABP B_1 that outputs the collection $\{f_A\}$ for each $A \in F$.*

Proof. The construction of the ABP is simple. It consists of $(k/2 + 1)$ layers where layer $\ell \in \{0, 1, \dots, k/2\}$ has $\binom{n}{\ell}$ many nodes indexed by ℓ size subsets of $[n]$. In $(\ell + 1)^{th}$ layer, the node indexed by the set T is connected to the nodes $T \setminus \{j\}$ in the previous layer with an edge label y_j for each $j \in T$. Clearly, in the last layer, the sink node indexed by the set A computes the polynomial f_A . □

Lemma 15. *There is an $\binom{n}{\downarrow k/2}$ -explicit multi-output ABP B_2 that outputs the collection $\{\hat{f}_S\}$ for each $S \in \downarrow F$.*

Proof. To construct such an ABP, we use ideas from [\[BHKK09\]](#). We define $\hat{f}_{i,S} = \sum_{S \subseteq A} f_A$ where $S \subseteq A$ and $A \cap [i] = S \cup [i]$. Note that, $\hat{f}_{n,S} = f_S$ and $\hat{f}_{0,S} = \hat{f}_S$. From the definition, it is clear that $\hat{f}_{i-1,S} = \hat{f}_{i,S} + \hat{f}_{i,S \cup \{i\}}$ if $i \notin S$ and $\hat{f}_{i-1,S} = \hat{f}_{i,S}$ if $i \in S$. Hence, we can take a copy of ABP B_1 from Lemma [14](#) and

then simultaneously compute $\hat{f}_{i,S}$ for each $S \in \downarrow F$ and i ranging from n to 0 .

Clearly, the new ABP B_2 consists of $(n + k/2 + 1)$ many layers and at most $\binom{n}{\downarrow k/2}$ nodes at each layer. The number of edges in the ABP is also linear in the number of nodes. \square

Let $f = \sum_{m \in Y^k} [m]f \cdot m$ be a noncommutative polynomial of degree k in $\mathbb{F}\langle Y \rangle$. The *reverse* of f is defined as the polynomial

$$f^R = \sum_{m \in Y^k} [m]f \cdot m^R,$$

where m^R is the reverse of the word m .

Lemma 16. *[Reversing an ABP] Suppose B is a multi-output ABP with r sink nodes where the i^{th} sink node computes $f_i \in \mathbb{F}\langle Y \rangle$ for each $i \in [r]$. We can construct an ABP of twice the size of B that computes the polynomial $\sum_{i=1}^r f_i \cdot L_i \cdot f_i^R$ where L_i are affine linear forms.*

Proof. Suppose B has ℓ layers, then we construct an ABP of $2\ell + 1$ layers where the first ℓ layers are the copy of ABP B and the last ℓ layers are the *mirror image* of the ABP B , call it B^R . More precisely, the ABP B^R is a r -source and single sink ABP, where the polynomial computed between i^{th} source and the sink is the polynomial f_i^R . In the $(\ell + 1)^{\text{th}}$ layer we connect the i^{th} sink node of ABP B to the i^{th} source node of B^R by an edge with the edge label L_i . Note that, B^R has r source nodes and one sink node and the polynomial computed between i^{th} source node and sink is f_i^R . \square

Now, applying the construction of Lemma 16 to the multi-output ABP B_2 of Lemma 15 with $L_S = (-1)^{|S|}$ we obtain an ABP that computes the polynomial $\sum_S (-1)^{|S|} \hat{f}_S \cdot \hat{f}_S^R$. Since \hat{f}_S is a symmetrized polynomial, we note that $\hat{f}_S^R = \hat{f}_S$ and

using Lemma [13](#) we conclude that this ABP computes $S_{n,k}^*$. The ABP size is $O(k \binom{n}{\lfloor k/2 \rfloor})$. It completes the proof of the first part.

Construction of ABP for rectangular permanent: A $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for the rectangular permanent polynomial can be obtained easily from the $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for $S_{n,k}^*(Y)$ by making the ABP set-multilinear.

More precisely, let A be the $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for $S_{n,k}^*(Y)$ obtained above. Recall that,

$$S_{n,k}^*(Y) = \sum_{T \subseteq [n]; |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)}.$$

To make the ABP set-multilinear, we simply rename the variables $y_i : 1 \leq i \leq n$ at the position $1 \leq j \leq k$ by $y_{j,i}$. It is immediate that the resulting polynomial is $\text{rPer}(Y)$. □

Remark 12. *Given an arithmetic circuit C of size s computing a degree- k polynomial $f \in \mathbb{F}[X]$, it is possible to compute the sum of the coefficients of all the multilinear terms present in f in $O^*(\binom{n}{\lfloor k/2 \rfloor} \cdot n^{c \cdot \log k})$ -time (where c is a constant) as shown in Chapter [5](#). Recall that, the key observation is that $f \circ S_{n,k}(\mathbf{1})$ is the sum of the coefficients of all the multilinear terms. It is also proved that for any $\mathbf{a} \in \mathbb{F}^n : f \circ S_{n,k}(\mathbf{a}) = \mathfrak{C} \circ S_{n,k}^*(\mathbf{a})$ for the noncommutative analog of C . Then the algorithm reduces the computation of $\mathfrak{C} \circ S_{n,k}^*(\mathbf{1})$ to a computation of a rectangular permanent over matrix inputs and appeal to the algorithm in [\[BHKK10\]](#). Now using Theorem [21](#), one gets an alternative viewpoint for this problem. Let A be the ABP obtained for $S_{n,k}^*$ from Theorem [21](#). Using the standard circuit to ABP conversion method [\[VSB83\]](#), one can first construct an ABP B of size $s^{O(\log k)}$ for f . Now recall Theorem [5](#) from Chapter [2](#) that computes the Hadamard product of two noncommutative ABPs efficiently. We can now compute $\mathfrak{B} \circ A(\mathbf{1})$ in time $O^*(\binom{n}{\lfloor k/2 \rfloor} \cdot n^{O(\log k)})$.*

6.2 Explicit ABP construction for Noncommutative Determinant and Related Polynomials

As in the usual commutative case, the noncommutative determinant polynomial of a symbolic matrix $Y = (y_{i,j})_{1 \leq i,j \leq k}$ is defined as follows (the variables in the monomials are ordered from left to right):

$$\text{Det}(Y) = \sum_{\sigma \in S_k} \text{sgn}(\sigma) y_{1,\sigma(1)} \cdots y_{k,\sigma(k)}.$$

Nisan [Nis91] has also shown that any algebraic branching program for the *noncommutative determinant* of a $k \times k$ symbolic matrix must be of size $\Omega(2^k)$. In this chapter, we give an explicit construction of such an ABP in time $O^*(2^k)$.

Remark 13. *In mathematics, noncommutative determinants are studied in various forms. For example, Gelfand, Gelfand, Retakh, and Wilson have studied the quasideterminants and successfully develop a version of Cramer's rule [GGRL05]. Another important notion is Dieudonn'e determinant which originated from the study of the determinant over division rings and local rings [Die43]. However, these notions are different from the notion of the noncommutative determinant polynomial studied in this chapter. Our motivation is mainly to study the complexity-theoretic properties of the noncommutative determinant polynomial as an element in $\mathbb{F}\langle Y \rangle$.*

Recall the results of Vassilevska and Williams [WW13] for commutative rectangular permanent. Motivated by those results, we study the complexity of the rectangular determinant polynomial (in the commutative domain) defined as follows.

$$\text{rDet}(X) = \sum_{S \in \binom{[m]}{k}} \text{Det}(X_S).$$

We prove that the rectangular determinant polynomial can be computed using $O^*(2^k)$ -size explicit ABP.

Theorem 22.

1. *The family of noncommutative determinants $\{\text{Det}(Y)\}_{k>0}$ has 2^k -explicit ABPs over any field.*
2. *There is a family $\{f_n\}$ of noncommutative degree- k polynomials f_n such that f_n has the same support as $S_{n,k}^*$, and it has 2^k -explicit ABPs. This result holds over any field that has at least n distinct elements.*
3. *The commutative rectangular determinant family $\{\text{rDet}(X)\}_{k>0}$, where X is a $k \times n$ matrix of variables has 2^k -explicit ABPs.*

Remark 14. *It is interesting to compare the results stated in Theorem [21](#) and Theorem [22](#). In Theorem [21](#), the polynomial families are defined over two parameters n and k , and the size of the ABPs are bounded by $O^*\binom{n}{\lfloor k/2 \rfloor}$. The first result stated in Theorem [22](#) considers a polynomial family with only one parameter k , and the size of the ABP is $O^*(2^k)$. However, the second and the third results consider the families of polynomials over two parameters n and k (like in Theorem [21](#)) but the ABP sizes show parameterized dependence on the parameter k only.*

Remark 15. *It is important to note that one can also make Nisan's results constructive using existing techniques. For example, it is possible to adapt the learning algorithm in [\[BBB⁺00\]](#) to design uniform algorithms that will construct the ABP families stated in Theorem [21](#). However, the constructions are not $\binom{n}{\lfloor k/2 \rfloor}$ -explicit.*

We divide the proof into three subsections showing the explicit ABPs for noncommutative determinant polynomial, a polynomial having the same support as $S_{n,k}^*$, and commutative rectangular determinant polynomial.

A 2^k -explicit ABP for $k \times k$ noncommutative determinant

In this section, we present an optimal explicit ABP construction for the noncommutative determinant polynomial for the square symbolic matrix.

Proof of Theorem 22.1. The ABP B has $k + 1$ layers with $\binom{k}{\ell}$ nodes at the layer ℓ for each $0 \leq \ell \leq k$. The source of the ABP is labeled \emptyset and the nodes in layer ℓ are labeled by the distinct size ℓ subsets $S \subseteq [k]$, $1 \leq \ell \leq k$, hence the sink is labeled $[k]$. From the node labeled S in layer ℓ , there are $k - \ell$ outgoing edges $(S, S \cup \{j\})$, $j \in [k] \setminus S$.

Define the sign $\text{sgn}(S, j)$ as $\text{sgn}(S, j) = (-1)^{t_j}$, where t_j is the number of elements in S larger than j . Equivalently, t_j is the number of swaps required to insert j in the correct position, treating S as a sorted list.

For noncommutative determinant polynomial, we connect the set S in the i^{th} layer to a set $S \cup \{j\}$ in the $(i + 1)^{\text{th}}$ layer with the edge label $\text{sgn}(S, j) \cdot y_{i+1, j}$. The source to sink paths in this ABP are in 1-1 correspondence to the node labels on the paths which give subset chains $\emptyset \subset T_1 \subset T_2 \subset \dots \subset T_k = [k]$ such that $|T_i \setminus T_{i-1}| = 1$ for all $i \leq k$. Such subset chains are clearly in 1-1 correspondence with permutations $\sigma \in S_k$ listed as a sequence: $\sigma(1), \sigma(2), \dots, \sigma(k)$, where $T_i = \{\sigma(1), \sigma(2), \dots, \sigma(i)\}$. The following claim spells out the connection between the sign $\text{sgn}(\sigma)$ of σ and the $\text{sgn}(S, j)$ function defined above.

Claim 6. For each $\sigma \in S_k$ and $T_i = \{\sigma(1), \sigma(2), \dots, \sigma(i)\}$, we have

$$\text{sgn}(\sigma) = \prod_{i=1}^k \text{sgn}(T_{i-1}, \sigma(i)).$$

Proof. We first note that $\text{sgn}(\sigma) = (-1)^t$, if there are t transpositions $(r_i \ s_i)$, $1 \leq i \leq t$ such that $\sigma \cdot (r_1 \ s_1) \cdot (r_2 \ s_2) \cdots (r_t \ s_t) = 1$. Equivalently, interpreting this as sorting the list $\sigma(1), \sigma(2), \dots, \sigma(k)$ by swaps $(r_i \ s_i)$, applying

these t swaps will sort the list into $1, 2, \dots, k$. As already noted, $\text{sgn}(T_{i-1}, \sigma(i)) = (-1)^{t_i}$, where t_i is the number of swaps required to insert $\sigma(i)$ in the correct position into the sorted order of T_{i-1} (where $\sigma(i)$ is initially placed to the right of T_{i-1}). Hence, $\sum_{i=1}^k t_i$ is the total number of swaps required for this insertion sort procedure to sort $\sigma(1), \sigma(2), \dots, \sigma(k)$. It follows that $\prod_{i=1}^k \text{sgn}(T_{i-1}, \sigma(i)) = (-1)^{\sum_i t_i} = \text{sgn}(\sigma)$, which proves the claim. \square

Now, it is easy to see that the ABP computes the noncommutative determinant polynomial. We use Claim [6](#) to obtain $\text{sgn}(\sigma)$ for each σ , and the edge labels are already mentioned. \square

Remark 16. In [\[Gre11\]](#), Grenet shows the construction of an ABP of size 2^k for the permanent polynomial of the symbolic $k \times k$ matrix, and the construction is similar to the above construction.

A 2^k -explicit ABP weakly equivalent to $S_{n,k}^*$

Recall from the last chapter that a polynomial $f \in \mathbb{F}[X]$ (resp. $\mathbb{F}\langle Y \rangle$) is said to be *weakly equivalent* to a polynomial $g \in \mathbb{F}[X]$ (resp. $\mathbb{F}\langle Y \rangle$), if for each monomial m over X , $[m]f = 0$ if and only if $[m]g = 0$. For the construction of an ABP computing a polynomial weakly equivalent to $S_{n,k}^*$, we will suitably modify the ABP construction described above.

Proof of Theorem [22](#).2. Let $\alpha_i, 1 \leq i \leq n$ be distinct elements from \mathbb{F} . For each $j \in [k] \setminus S$, the edge $(S, S \cup \{j\})$ is labeled by the linear form $\text{sgn}(S, j) \cdot \sum_{i=1}^n \alpha_i^j y_i$, where $y_i, 1 \leq i \leq n$ are noncommuting variables. This gives an ABP B of size $O^*(2^k)$.

We show that the polynomial computed by ABP B is weakly equivalent to $S_{n,k}^*$. B computes a homogeneous degree k polynomial in the variables $y_i, 1 \leq i \leq n$. We determine the coefficient of a monomial $y_{i_1} y_{i_2} \cdots y_{i_k}$. As noted, each source to sink

path in B corresponds to a permutation $\sigma \in S_k$. Along that path, the ABP computes the product of linear forms

$$\operatorname{sgn}(\sigma)L_{\sigma(1)}L_{\sigma(2)}\cdots L_{\sigma(k)}, \text{ where } L_{\sigma(q)} = \sum_{i=1}^n \alpha_i^{\sigma(q)} y_i,$$

where the sign is given by the previous claim. The coefficient of the monomial $y_{i_1}y_{i_2}\cdots y_{i_k}$ in the above product is given by $\operatorname{sgn}(\sigma)\prod_{q=1}^k \alpha_{i_q}^{\sigma(q)}$. Thus, the coefficient of $y_{i_1}y_{i_2}\cdots y_{i_k}$ in the ABP is given by $\sum_{\sigma \in S_k} \operatorname{sgn}(\sigma)\prod_{q=1}^k \alpha_{i_q}^{\sigma(q)}$, which is the determinant of the $k \times k$ Vandermonde matrix whose q^{th} column is $(\alpha_{i_q}, \alpha_{i_q}^2, \dots, \alpha_{i_q}^k)^T$. That determinant is non-zero if and only if the monomial $y_{i_1}y_{i_2}\cdots y_{i_k}$ is multilinear. Clearly the proof works for any field that contains at least n distinct elements. \square

Remark 17. A polynomial $f \in \mathbb{Q}\langle Y \rangle$ is positively weakly equivalent to $S_{n,k}^*$, if for each multilinear monomial $m \in Y^k$, $[m]f > 0$. In the above proof, let g be the polynomial computed by ABP B that is weakly equivalent to $S_{n,k}^*$. Clearly, $f = g \circ g$ is positively weakly equivalent to $S_{n,k}^*$, and f has a 4^k -explicit ABP, since B is 2^k -explicit. This follows from Theorem [5](#). We leave open the problem of finding a 2^k -explicit ABP for some polynomial that is positively weakly equivalent to $S_{n,k}^*$. Such an explicit construction would imply a deterministic $O^*(2^k)$ time algorithm for the k -path which is a longstanding open problem [\[KW16\]](#).

A 2^k -explicit ABP for $k \times n$ commutative rectangular determinant

In this section, we present the ABP construction for commutative determinant polynomial for $k \times n$ symbolic matrix.

Proof of Theorem [22.3](#). We adapt the ABP presented in Subsection [6.2](#). The main difference is that, for the edge $(S, S \cup \{j\})$, the linear form is

$\text{sgn}(S, j) \cdot (\sum_{i=1}^n x_{j,i} z_i)$, where $z_i : 1 \leq i \leq n$ are fresh noncommuting variables, and the $x_{j,i} : 1 \leq j \leq k, 1 \leq i \leq n$ are commuting variables.

Then by an argument similar to the one in Subsection [6.2](#), the coefficient of the monomial $z_{i_1} z_{i_2} \dots z_{i_k}$ where $i_1 < i_2 < \dots < i_k$ is given by

$\sum_{\sigma \in S_k} \text{sgn}(\sigma) x_{\sigma(1), i_1} \dots x_{\sigma(k), i_k}$ Now for a fixed $\sigma \in S_k$, let τ^σ be the injection $[k] \rightarrow [n]$ such that $\tau^\sigma(j) = i_{\sigma^{-1}(j)} : 1 \leq j \leq k$.

Let (j_1, j_2) be an index pair that is an inversion in σ , i.e. $j_1 < j_2$ and $\sigma(j_1) > \sigma(j_2)$.

Let $\ell_1 = \sigma(j_1)$ and $\ell_2 = \sigma(j_2)$. So $i_{\tau^\sigma(\ell_1)} = i_{\sigma^{-1}(\ell_1)}$ and $i_{\tau^\sigma(\ell_2)} = i_{\sigma^{-1}(\ell_2)}$. Clearly,

$i_{\tau^\sigma(\ell_1)} < i_{\tau^\sigma(\ell_2)}$. Hence:

$$\sum_{\sigma \in S_k} \text{sgn}(\sigma) x_{\sigma(1), i_1} \dots x_{\sigma(k), i_k} = \sum_{\tau^\sigma \in I_{k,n}} \text{sgn}(\tau^\sigma) x_{1, \tau^\sigma(1)} \dots x_{k, \tau^\sigma(k)}.$$

Now the idea is to filter out only the good monomials $z_{i_1} z_{i_2} \dots z_{i_k}$ where

$i_1 < i_2 < \dots < i_k$ from among all the monomials. This can be done by taking the

Hadamard product (using Theorem [5](#)) with the following polynomial,

$$S_{n,k}^{nc}(Z) = \sum_{S=\{i_1 < i_2 < \dots < i_k\}} z_{i_1} z_{i_2} \dots z_{i_k}.$$

Clearly, $S_{n,k}^{nc}$ has a $\text{poly}(n, k)$ -sized ABP which is just the noncommutative version

(see Definition [16](#)) of the well-known ABP for commutative $S_{n,k}$. Finally, we

substitute each $z_i = 1$ to get the desired ABP for $\text{rDet}(X)$. □

6.3 Hardness of Rectangular Determinant Over Matrix Algebras

Finally, we consider the problem of evaluating the noncommutative rectangular determinant over matrix algebras and show that it is $\#\mathbf{W}[1]$ -hard for polynomial dimensional matrices. Hence the noncommutative rectangular determinant is unlikely to have an explicit $O^*(n^{o(k)})$ -size ABP. Recall from Chapter 5 we have shown the $\#\mathbf{W}[1]$ -hardness of computing the noncommutative rectangular permanents over poly-dimensional rational matrices. We note that the noncommutative $n \times n$ determinant over matrix algebras is well-studied, and computing it remains $\#\mathbf{P}$ -hard even over 2×2 rational matrices

[AS10, CHSS11, Blä15]. Our proof technique is based on the Hadamard product of noncommutative polynomials [AS10]. However, the crucial difference is that, to show the $\#\mathbf{P}$ -hardness of the noncommutative determinant, the proof in [AS10] shows a reduction from the evaluation of the commutative permanent to the noncommutative determinant; whereas, the $\#\mathbf{W}[1]$ -hardness of the noncommutative rectangular determinant requires a different proof route because the commutative rectangular permanent is in FPT. We show that the rectangular determinant (and the rectangular permanent), whose entries are $r \times r$ matrices over any field can be computed in time $O^*(2^k r^{2k})$.

Next, we describe the parameterized hardness result for rectangular determinant polynomial when we evaluate over matrix algebras.

Theorem 23. *For any fixed $\epsilon > 0$, evaluating the $k \times n$ rectangular determinant polynomial over $n^\epsilon \times n^\epsilon$ rational matrices is $\#\mathbf{W}[1]$ -hard, treating k as fixed parameter.*

In this section, we prove a hardness result for evaluating the rectangular determinant over matrix algebras. More precisely, if A is a $k \times n$ matrix whose

entries A_{ij} are $n^\epsilon \times n^\epsilon$ rational matrices for a fixed $\epsilon > 0$, then it is $\#\mathbf{W}[1]$ -hard to compute $\text{rDet}(A)$. We show this by a reduction from the $\#\mathbf{W}[1]$ -complete problem of counting the number of simple k -paths in directed graphs.

Let $G(V, E)$ be a directed graph with n vertices where $V(G) = \{v_1, v_2, \dots, v_n\}$. A k -walk is a sequence of k vertices $v_{i_1}, v_{i_2}, \dots, v_{i_k}$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $1 \leq j \leq k - 1$. A k -path is a k -walk where no vertex is repeated. Let A be the adjacency matrix of G , and let z_1, z_2, \dots, z_n be noncommuting variables. Define the $n \times n$ matrix B as follows:

$$B[i, j] = A[i, j] \cdot z_i, \quad 1 \leq i, j \leq n.$$

Let $\underline{1}$ denote the all 1's vector of length n . Let \underline{z} be the length n vector defined by $\underline{z}[i] = z_i$. The polynomial $C_G \in \mathbb{F}\langle Z \rangle$ is defined as

$$C_G(z_1, z_2, \dots, z_n) = \underline{1}^T \cdot B^{k-1} \cdot \underline{z}.$$

Let W be the set of all k -walks in G . The following observation is folklore.

Observation 1.

$$C_G(z_1, z_2, \dots, z_n) = \sum_{v_{i_1} v_{i_2} \dots v_{i_k} \in W} z_{i_1} z_{i_2} \dots z_{i_k}.$$

Hence, G contains a k -path if and only if the polynomial C_G contains a multilinear term.

The Proof of Theorem [23](#)

Let $I_{k,n}$ be the set of injections from $[k] \rightarrow [n]$. Define

$$S := \{f \in I_{2k,2n} \mid \exists g \in I_{k,n} \text{ such that } \forall i \in [k], f(2i-1) = g(i); f(2i) = n + g(i)\}.$$

There is a bijection between S and $I_{k,n}$. We denote each $f \in S$ as f_g where $g \in I_{k,n}$ is the corresponding injection. By a simple counting argument, we observe the following.

Observation 2. For each $f \in S$, $\text{sgn}(f) = (-1)^{\frac{k(k-1)}{2}}$.

Consider a set of noncommuting variables $Y = \{y_{1,1}, y_{1,2}, \dots, y_{2k,2n}\}$ corresponding to the entries of a $2k \times 2n$ symbolic matrix Y . Given $f \in I_{2k,2n}$, define

$$m_f = \prod_{i=1}^{2k} y_{i,f(i)}.$$

Lemma 17. There is an ABP B of $\text{poly}(n, k)$ size that computes a polynomial $F \in \mathbb{F}\langle Y \rangle$ such that for each $f \in I_{2k,2n}$, $[m_f]F = 1$ if $f \in S$. Otherwise $[m_f]F = 0$.

Proof. The ABP B consists of $2k + 1$ layers, labelled $\{0, 1, \dots, 2k\}$. For each even $i \in [0, 2k]$, there is exactly one node q_i at level i . For each odd $i \in [0, 2k]$, there are n nodes $p_{i,1}, p_{i,2}, \dots, p_{i,n}$ at level i . We now describe the edges of B . For each even $i \in [0, 2k - 2]$ and $j \in [n]$, there is an edge from q_i to $p_{i+1,j}$ labeled $y_{i+1,j}$. For each odd $i \in [0, 2k - 1]$ and $j \in [n]$, there is an edge from $p_{i,j}$ to q_{i+1} labeled $y_{i+1,n+j}$. For an injection $f \in I_{2k,2n}$, B contributes a monomial m_f if and only if $f \in S$ and B can be computed in $\text{poly}(n, k)$ time. \square

Suppose Y is a $2k \times 2n$ matrix where the $(i, j)^{\text{th}}$ entry is $y_{i,j}$. By Observation [2](#) and Lemma [17](#),

$$\text{rDet}(Y) \circ F(Y) = \sum_{f_g \in S} \text{sgn}(f_g) m_{f_g} = (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m_{f_g}.$$

Let $Z = \{z_1, \dots, z_n\}$ be a set of noncommuting variables. Define for each $g \in I_{k,n}$, $m'_g = \prod_{i=1}^k z_{g(i)}$. Define a map τ such that $\tau : y_{i,j} \mapsto z_j$ if i is odd, and $\tau : y_{i,j} \mapsto 1$ for even i . In other words, $\tau(m_{f_g}) = m'_g$. Notice that,

$$\begin{aligned} \text{rDet}(Y) \circ F(Y)|_\tau &= (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m_{f_g}|_\tau \\ &= (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m'_g \\ &= (-1)^{\frac{k(k-1)}{2}} S_{n,k}^*(Z). \end{aligned}$$

Given a directed graph G on n vertices, we first construct an ABP for the noncommutative graph polynomial C_G over rationals. From the definition, it follows that C_G has a polynomial-size ABP. Notice that,

$$((\text{rDet}(Y) \circ F(Y)|_\tau) \circ C_G(Z))(\underline{1}) = S_{n,k}^*(Z) \circ C_G(Z)(\underline{1})$$

counts the number of directed k -paths in the graph G , and hence evaluating this term is $\#\text{W}[1]$ -hard. Let us modify the ABP for graph polynomial $C_G(Z)$ by replacing each edge labeled by z_j at i^{th} layer by two edges where the first edge is labeled by $y_{2i-1,j}$ and second one is labeled by $y_{2i,n+j}$. Let $C'_G(Y)$ is the new polynomial computed by the ABP. Notice that, each monomial of the modified graph polynomial looks like $\prod_{i=1}^{2k} y_{i,f(i)}$ for some $f : [2k] \mapsto [2n]$. More importantly, for each k -path $v_{i_1}v_{i_2} \dots v_{i_k}$, if $g \in I_{k,n}$ is the corresponding injection, then $\prod_{i=1}^k z_{g(i)}$ is converted to $\prod_{i=1}^{2k} y_{i,f_g(i)}$ for $f_g \in S$. Notice that, $(\text{rDet}(Y) \circ F(Y)|_\tau) \circ C_G(Z) = (\text{rDet}(Y) \circ F(Y) \circ C'_G(Y))|_\tau$ and hence, evaluating $(\text{rDet}(Y) \circ F(Y) \circ C'_G(Y))(\underline{1})$ is $\#\text{W}[1]$ -hard.

Now, assume to the contrary, we have an FPT algorithm \mathcal{A} to evaluate $\text{rDet}(Y)$ over matrix inputs. As, $C'_G(Y)$ and $F(Y)$ are computed by ABPs, we obtain an ABP B' computing $C'_G \circ F(Y)$. From ABP B' , we construct the $t \times t$ transition

matrices $M_{1,1}, \dots, M_{2k,2n}$ where t is the size of the ABP B' . From Lemma 7 we know that, we are interested in computing $\text{rDet}(Y)$ over the matrix tuple $(M_{1,1}, \dots, M_{2k,2n})$ which is same as invoking the algorithm \mathcal{A} on the $2k \times 2n$ matrix A : $a_{i,j} = M_{i,j}$. By a simple reduction we get a similar hardness over $n^\epsilon \times n^\epsilon$ dimensional matrix algebras for any fixed $\epsilon > 0$. \square

6.3.1 Computing over Small Dimensional Algebras

Finally, in contrast to the above hardness result, we show that there are simple algorithms of run time $O^*(2^k r^{2k})$ to evaluate the rectangular permanent and the rectangular determinant of size $k \times n$ over $r \times r$ matrix algebras. Thus, over constant-dimensional matrix algebras they are fixed-parameter tractable.

Theorem 24. *Let \mathbb{F} be any field and \mathcal{A} be an r dimensional algebra over \mathbb{F} with basis e_1, e_2, \dots, e_r . Let $\{A_{ij}\}_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}}$ be a $k \times n$ matrix with $A_{ij} \in \mathcal{A}$. Then $\text{rPer}(A)$ and $\text{rDet}(A)$ can be computed in deterministic $O^*(2^k r^k)$ time.*

Proof. We present the proof for the rectangular permanent. The proof for the rectangular determinant is identical. The proof follows easily from expressing each entry $A_{i,j}$ on the standard basis and then rearranging terms. Let e_1, e_2, \dots, e_r be the standard basis for \mathcal{A} over \mathbb{F} . First we note that,

$$\begin{aligned}
\text{rPer}(A) &= \sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)} \\
&= \sum_{f \in I_{k,n}} \prod_{i=1}^k \sum_{\ell=1}^r A_{if(i)}^{(\ell)} e_\ell \\
(6.1) \quad &= \sum_{f \in I_{k,n}} \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \prod_{i=1}^k A_{if(i)}^{(t_i)} \prod_{i=1}^k e_{t_i} \\
&= \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \left(\sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)}^{(t_i)} \right) \prod_{i=1}^k e_{t_i}.
\end{aligned}$$

Now we observe that

$$\sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)}^{(t_i)} = \text{rPer}(A^{(t_1, t_2, \dots, t_k)}),$$

where $A^{(t_1, t_2, \dots, t_k)}$ is the $k \times n$ matrix defined as $A_{ij}^{(t_1, t_2, \dots, t_k)} = A_{ij}^{(t_i)}$. Thus we have

$$(6.2) \quad \text{rPer}(A) = \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \text{rPer}(A^{(t_1, t_2, \dots, t_k)}) \prod_{i=1}^k e_{t_i}.$$

For a fixed $(t_1, t_2, \dots, t_k) \in [r]^k$ the value $\text{rPer}(A^{(t_1, t_2, \dots, t_k)})$ can be computed in $O^*(2^k)$ time using the rectangular permanent algorithm [WW13]. Now we can compute $\text{rPer}(A)$ by computing r^k many such rectangular permanents and putting them together according to equation [6.2]. This gives a deterministic $O^*(2^k r^k)$ time algorithm for computing $\text{rPer}(A)$. \square

As a direct corollary, we get the following.

Corollary 8. *Let \mathbb{F} be any field and let A be a $k \times n$ matrix with $A_{ij} \in \text{Mat}_r(\mathbb{F})$. Then $\text{rPer}(A)$ and $\text{rDet}(A)$ can be computed in $O^*(2^k r^{2k})$ time.*

Conclusion

In this chapter, we have presented the construction of explicit algebraic branching programs for the noncommutative symmetrized elementary symmetric polynomial, the noncommutative rectangular permanent polynomial, and the commutative rectangular determinant polynomial. Additionally, we have given an explicit algebraic branching program for the noncommutative square determinant polynomial. The constructions are essentially optimal, in the sense of the lower bound result of Nisan [Nis91]. We have also shown that evaluating the rectangular determinant polynomial over matrix algebras is $\#\text{W}[1]$ -hard.

This chapter opens further avenues of research. A very interesting problem is to tightly classify the complexity of computing the commutative rectangular $k \times n$ determinant polynomial. Is it computable in $\mathbf{poly}(n, k)$ time? If not, can one show a complexity-theoretic hardness of evaluating the commutative determinant polynomial? We feel that the main obstacle is interpreting the rectangular determinant computation combinatorially. Another open question is whether or not there is an explicit algebraic branching program for the *noncommutative* rectangular determinant of size $O^*(n^{ck})$ for some $c < 1$, similar to our construction for the noncommutative rectangular permanent.

Chapter 7

Conclusion

In this chapter, we summarize the main results of the thesis. We also recapitulate the main technical ideas involved; we hope these can be useful for further research on the topics studied. Finally, we list some open problems arising from the thesis.

The results presented in the thesis revolve around two algorithmic problems:

rational identity testing and *multilinear monomial detection*. These problems are well-motivated and well-studied in recent years. For both problems, tools from algebraic complexity theory have proven useful in algorithms.

Rational Identity Testing

As already mentioned, *rational identity testing* (RIT) problem is to determine if a given noncommutative rational formula computes the zero function in the free skew-field. The problem is in deterministic polynomial time in the white-box model [GGOW16, IQS18] and in randomized polynomial time in the black-box model [DM17]. The main open problems are (a) to derandomize the black-box RIT algorithm, and (b) to obtain white-box RIT for more general models than formulas. In Chapter 3 and Chapter 4 we have explored some restricted models and have

obtained *efficient* RIT algorithms.

- In Chapter [3](#), we consider two generalizations of noncommutative ABPs: one that allows inversion at the top (meaning the output), the other that allows inversions at the bottom (i.e. each edge label of the ABP is either an affine linear form or the inverse of an affine linear form). We obtain deterministic polynomial-time algorithms in white-box and a deterministic quasi-polynomial-time algorithm in black-box for these models. The main contribution is to explore the connection between RIT and algebraic automata theory to obtain efficient black-box algorithms.
- In Chapter [4](#), we consider a special case of noncommutative rational circuits of inversion height one that allows inverses only at the bottom-most layer computing a free group algebra function. We obtain a randomized polynomial (in the maximum length of a word in the expression) time algorithm for this problem. Moreover, we generalize the known identity testing results for noncommutative arithmetic circuits [\[AJMR17\]](#), [\[BW05\]](#).

Open Problems: Several questions remain open.

1. Can the connection to algebraic automata theory lead to a deterministic black-box quasi-polynomial time algorithm for rational identity testing of all noncommutative rational formulas? Recall that, for polynomial identity testing of noncommutative formula, we have a deterministic black-box quasi-polynomial time algorithm due to Forbes and Shpilka [\[FS13\]](#). Connections between algebraic automata and rational functions in the free skew field have been studied by Volčič [\[Vol18\]](#) motivated by linear systems and control.
2. Can we obtain deterministic rational identity testing algorithms for models

stronger than rational formulas?

3. As already mentioned in Chapter 4, an interesting problem is to tighten the bound on the minimum degree of a free group algebra identity for matrix algebras.

Multilinear Monomial Detection

Recall that, given an arithmetic circuit computing a polynomial of degree- k as input, the multilinear monomial detection problem (k -MMD) is to check whether an input polynomial (by a circuit or ABP) has a multilinear monomial with a nonzero coefficient. This problem is known to be important in *parameterized complexity* as many graph problems, such as finding the longest path, matching, dominating set etc., are reducible to it. The counting version multilinear monomial counting ((k,n) -MLC) is to compute the sum of the coefficients of all the multilinear terms in the polynomial.

- An open problem in this area was to improve the trivial $O(n^k)$ time exhaustive search algorithm for (k,n) -MLC. In Chapter 5 we solved this problem by obtaining a $O(n^{k/2+c\log k})$ time deterministic algorithm that works over \mathbb{Q} and any finite field. We also studied the k -MMD problem for general arithmetic circuits. Over rationals, Brand et al. [BDH18] obtained a $4.32^k \cdot \text{poly}(n)$ time algorithm for it that requires exponential (in k) space. In Chapter 5 we have presented an algorithm with the same running time that works over finite fields also and requires only polynomial space.

We obtain these upper bounds by efficiently computing *scaled Hadamard product* (or the *apolar inner product*) of two commutative polynomials. Our main technical contribution is to introduce a new technique, *symmetrization* in order to compute this.

- In Chapter [6](#) we have shown ABP constructions for the noncommutative permanent and the noncommutative determinant with some related polynomials. Our results show that the complexity of polynomials of exponential ABP complexity can be explored further in the context of parameterized complexity.

Open Problems: The most interesting open problem here is the following: Is there a deterministic $2^k \cdot \text{poly}(n)$ time algorithm for the k -MMD problem when the input n -variate polynomial is monotone and it is given by an ABP? A positive answer will solve the long-standing open problem of deciding whether a graph of size n has a path of length k in deterministic $O^*(2^k)$ time. From our work, it seems that a first step would be to find a monotone weakly equivalent elementary symmetric polynomial such that the corresponding noncommutative symmetrized polynomial is computable by an ABP of size $2^k \cdot \text{poly}(n)$.

Overall this thesis makes some progress in our understanding of the power of noncommutative algebraic complexity. We believe that connections between algebraic automata theory and algebraic complexity merits further study. The symmetrization technique needs to be explored further.

Bibliography

- [ACDM19a] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Efficient black-box identity testing for free group algebras. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, pages 57:1–57:16, 2019.
- [ACDM19b] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast exact algorithms using hadamard product of polynomials. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPICs*, pages 9:1–9:14. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.
- [ACDM19c] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. On explicit branching programs for the rectangular determinant and permanent polynomials. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*,

pages 38:1–38:13. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

- [ACDM20a] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. A Special Case of Rational Identity Testing and the Brešar-Klep Theorem. In Javier Esparza and Daniel Král, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [ACDM20b] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. On explicit branching programs for the rectangular determinant and permanent polynomials. *Chic. J. Theor. Comput. Sci.*, 2020, 2020.
- [ACDM22] Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast exact algorithms using hadamard product of polynomials. *Algorithmica*, 84(2):436–463, 2022.
- [Agr05] M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. In: Sarukkai S., Sen S. (eds) FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science. FSTTCS 2005, 2005.
- [AJMR17] Vikraman Arvind, Pushkar S. Joglekar, Partha Mukhopadhyay, and S. Raja. Randomized polynomial time identity testing for noncommutative circuits. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 831–841, 2017.

- [AJS09] Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic circuits and the Hadamard product of polynomials. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 25–36, 2009.
- [AL50] A. S. Amitsur and J. Levitzki. Minimal identities for algebras. *Proceedings of the American Mathematical Society*, 1(4):449–463, 1950.
- [Ami66] S.A Amitsur. Rational identities and applications to algebra and geometry. *Journal of Algebra*, 3(3):304 – 359, 1966.
- [AMS10] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010.
- [AS10] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 677–686, 2010.
- [AS18] Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. *Computational Complexity*, 27(1):1–29, 2018.
- [AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
- [BBB⁺00] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

- [BDH18] Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 151–164, 2018.
- [Ber76] George M Bergman. Rational relations and rational identities in division rings. *Journal of Algebra*, 43(1):252 – 266, 1976.
- [BHKK09] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 578–586, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [BHKK10] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Evaluation of permanents in rings and semirings. *Inf. Process. Lett.*, 110(20):867–870, 2010.
- [BK08] Matej Bresar and Igor Klep. Values of noncommutative polynomials, Lie skew-ideals and the Tracial Nullstellensatz. *Mathematical Research Letters*, 2008.
- [Blä15] Markus Bläser. Noncommutativity makes determinants hard. *Information and Computation*, 243:133 – 144, 2015. 40th International Colloquium on Automata, Languages and Programming (ICALP 2013).
- [BP21] Cornelius Brand and Kevin Pratt. Parameterized applications of symbolic differentiation of (totally) multilinear polynomials. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual*

Conference), volume 198 of *LIPICs*, pages 38:1–38:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

- [BR11] J. Berstel and C. Reutenauer. *Noncommutative Rational Series with Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.
- [BS83] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.
- [BW05] Andrej Bogdanov and Hoeteck Wee. More on noncommutative polynomial identity testing. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 92–99, 2005.
- [CHSS11] Steve Chien, Prahladh Harsha, Alistair Sinclair, and Srikanth Srinivasan. Almost settling the hardness of noncommutative determinant. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 499–508, 2011.
- [Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71*, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.
- [DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- [Die43] Jean Dieudonné. Les déterminants sur un corps non commutatif. *Bulletin de la Société Mathématique de France*, 71:27–45, 1943.

- [DL78] Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193 – 195, 1978.
- [DM17] Harm Derksen and Visu Makam. Polynomial degree bounds for matrix semi-invariants. *Advances in Mathematics*, 310:44–63, 2017.
- [DM18] Harm Derksen and Visu Makam. On non-commutative rank and tensor rank. *Linear and Multilinear Algebra*, 66(6):1069–1084, 2018.
- [Eil74] Samuel Eilenberg. *Automata, Languages, and Machines (Vol A)*. Pure and Applied Mathematics. Academic Press, 1974.
- [Fis94] Ismor Fischer. Sums of like powers of multivariate linear forms. *Mathematics Magazine*, 67(1):59–61, 1994.
- [For14] Michael Andrew Forbes. Polynomial identity testing of read-once oblivious algebraic branching programs. 2014.
- [FS13] Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 243–252, 2013.
- [GG81] CD Godsil and I Gutman. On the matching polynomial of a graph, algebraic methods in graph theory I-II, 1981.
- [GGOW16] Ankit Garg, Leonid Gurvits, Rafael Mendes de Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 109–117, 2016.

- [GGRL05] Israel Gelfand, Sergei Gelfand, Vladimir Retakh, and Robert Lee Wilson. Quasideterminants. *Advances in Mathematics*, 193(1):56 – 141, 2005.
- [Gre11] Bruno Grenet. An Upper Bound for the Permanent versus Determinant Problem. Manuscript, 2011.
- [Hua49] Loo-Keng Hua. Some properties of a sfield. *Proceedings of the National Academy of Sciences of the United States of America*, 35(9):533–537, 1949.
- [HW15] Pavel Hrubeš and Avi Wigderson. Non-commutative arithmetic circuits with division. *Theory of Computing*, 11(14):357–393, 2015.
- [HWZ08] Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008.
- [IQS18] Gábor Ivanyos, Youming Qiao, and K. V. Subrahmanyam. Constructive non-commutative rank computation is in deterministic polynomial time. *computational complexity*, 27(4):561–593, Dec 2018.
- [JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, July 2004.
- [KI04] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complex.*, 13(1/2):1–46, December 2004.
- [Kou08] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11,*

2008, *Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008.

- [KS01] Adam R. Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 216–223, New York, NY, USA, 2001. ACM.
- [KW16] Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016.
- [Lee15] Hwangrae Lee. Power sum decompositions of elementary symmetric polynomials, volume 492, 2015. *Linear Algebra and its Applications*.
- [Lev73] L. A. Levin. Universal sequential search problems. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 265–266. Probl. Peredachi Inf., 1973.
- [LS12] Sylvain Lombardy and Jacques Sakarovitch. The removal of weighted ϵ -transitions. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata*, pages 345–352, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [LZ09] Tsiu-Kwen Lee and Yiqiang Zhou. Right ideals generated by an idempotent of finite rank. *Linear Algebra and its Applications*, 431:2118–2126, 11 2009.
- [MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

- [Nis91] Noam Nisan. Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991.
- [NSS95] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 182–191, 1995.
- [Pra18] Kevin Pratt. Faster algorithms via waring decompositions. *CoRR*, abs/1807.06194, 2018.
- [Pra19] Kevin Pratt. Waring rank, parameterized and exact algorithms. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 806–823. IEEE Computer Society, 2019.
- [Row80] Louis Halle Rowen. *Polynomial identities in ring theory*. Pure and Applied Mathematics. Academic Press, 1980.
- [RS05] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non-commutative models. *Computational Complexity*, 14(1):1–19, 2005.
- [Rys63] H.J. Ryser. *Combinatorial mathematics*. Carus mathematical monographs. Mathematical Association of America; distributed by Wiley [New York], 1963.
- [Sap15] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. 2015.
<https://github.com/dasarpmar/lowerbounds-survey/>.

- [Sax08] Nitin Saxena. Diagonal circuit identity testing and lower bounds. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Track A: Algorithms, Automata, Complexity, and Games*, pages 60–71, 2008.
- [Sch61] M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245 – 270, 1961.
- [Sch80] Jacob T. Schwartz. Fast probabilistic algorithm for verification of polynomial identities. *J. ACM.*, 27(4):701–717, 1980.
- [Str73] Volker Strassen. Vermeidung von divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.
- [SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.
- [Val79] Leslie G. Valiant. Completeness classes in algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261, 1979.
- [Vol18] Jurij Volčič. Matrix coefficient realization theory of noncommutative rational functions. *Journal of Algebra*, 499:397–437, 04 2018.
- [VSBR83] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.
- [Wer05] Dirk Werner. *Funktionalanalysis (in German)*. Springer Verlag, 2005.
- [Wil09] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

- [Wil14a] Ryan Williams. Algorithms for circuits and circuits for algorithms. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 248–261, 2014.
- [Wil14b] Ryan Williams. The polynomial method in circuit complexity applied to algorithm design (invited talk). In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 47–60, 2014.
- [WW13] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.
- [Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. of the Int. Sym. on Symbolic and Algebraic Computation*, pages 216–226, 1979.
- [Šp12] Špela Špenko. On the image of a noncommutative polynomial. *Journal of Algebra*, 377, 12 2012.