On the Complexity of Certain Algebraic and Number Theoretic Problems

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

by

Chandan Saha

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May, 2010

CERTIFICATE

It is certified that the work contained in the thesis entitled "*On the Complexity of Certain Algebraic and Number Theoretic Problems*", by "*Chandan Saha*", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Manindra Agrawal)
Professor,
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

May, 2010

Synopsis

In this thesis we study the deterministic complexity of three problems belonging to the subject of computational algebra and number theory. These problems are - univariate polynomial factoring over finite fields, large integer multiplication and polynomial identity testing. The choice of these problems is primarily motivated by their fundamental nature as mathematical problems and by their important applications in areas like coding theory, cryptography and complexity theory.

Finding an efficient deterministic algorithm to factor univariate polynomials over finite fields is a long standing open problem. Building on earlier work by Evdokimov and Gao, we show that a given polynomial can be factored in deterministic polynomial time, under the assumption of the Extended Riemann Hypothesis, unless the roots of the polynomial satisfy a strong *symmetry* property. Our symmetry property strengthens the symmetry property (*square balance*) defined by Gao. We also give a tight estimate of the fraction of square balanced polynomials (over fields of characteristic $p = 3 \bmod 4$), showing it to be exponentially small. Our main motivation behind this approach to factoring is that checking for inherent asymmetries among the roots can be used to improve the time complexity of the best deterministic algorithms on most input polynomials.

Integer multiplication is ubiquitous in computational number theory. We give an $n \cdot \log n \cdot 2^{O(\log^* n)}$ time algorithm to multiply two n bit integers that uses modular arithmetic for intermediate computations instead of arithmetic over complex numbers as in Fürer's algorithm, which also has the same and so far the best known complexity. The previous best algorithm using modular arithmetic (by Schönhage and Strassen) has complexity $O(n \cdot \log n \cdot \log\log n)$. The advantage of using modular arithmetic as opposed to complex arithmetic is that we can completely evade

the task of bounding the truncation error due to finite approximations of complex numbers, and this imparts more transparency into our algorithm.

Polynomial Identity Testing (PIT) is a fundamental problem lying at the interface of complexity theory and computational algebra. We study the problem of identity testing for depth-2 arithmetic circuits over matrix algebra. We show that identity testing of depth-3 ($\Sigma\Pi\Sigma$) arithmetic circuits over a field $\mathbb{F}$ is polynomial time equivalent to identity testing of depth-2 ($\Pi\Sigma$) arithmetic circuits over $\mathsf{U}_2(\mathbb{F})$, the algebra of upper-triangular 2×2 matrices with entries from $\mathbb{F}$. Such a connection is a bit surprising since we also show that, as computational models, $\Pi\Sigma$ circuits over $\mathsf{U}_2(\mathbb{F})$ are strictly 'weaker' than $\Sigma\Pi\Sigma$ circuits over $\mathbb{F}$. The equivalence further implies that PIT of $\Sigma\Pi\Sigma$ circuits reduces to PIT of width-2 commutative *Algebraic Branching Programs*(ABP). Further, we give a deterministic polynomial time identity testing algorithm for $\Pi\Sigma$ circuits of size s over commutative algebras of dimension $O(\log s/\log\log s)$ over $\mathbb{F}$. Over commutative algebras of dimension $O(s)$, we show that identity testing of $\Pi\Sigma$ circuits is at least as hard as that of $\Sigma\Pi\Sigma$ circuits over $\mathbb{F}$.

Finally, we study the complexity of two particular problems on identity testing. One is a generalization of a problem considered by Kayal and Saxena. Here, we are required to test if the output of a given depth-3 circuit with bounded top fanin equals a given sparse polynomial. The second problem is on checking if a given sparse polynomial equals the product of a given set of other sparse polynomials, a problem that is noted as an open question by von zur Gathen. Using a technique called *dual representation* of polynomials, we give deterministic polynomial time solutions for the first problem and a special case of the second problem where every polynomial in the input set of alleged factors is a sum of univariate polynomials.

Acknowledgements

I am deeply thankful to my advisor Manindra Agrawal for his invaluable guidance and his persistent encouragement throughout the course of this work. His interest in solving fundamental problems, his perseverance in the face of failures, and his pursuit for clarity have always inspired me. Much of the work done in this thesis is greatly influenced by his own views, thoughts and ideas. Thank you Manindra, for all your support. I owe a lot to you, not only for your scientific advice but also for all the generous helps that I have received from you throughout these years.

Studying at IIT Kanpur has been an enjoyable experience for me. I am indebted to the faculties of the Department of Computer Science and Engineering, specially Somenath Biswas, Sumit Ganguly, Piyush Kurur and Surender Baswana for their teaching and their many motivational advice that helped me focus on my work. Piyush has also co-authored the work presented in Chapter 4 and I have enjoyed countless hours of stimulating discussions with him which sometimes went into the realm of philosophy :).

I am grateful to Nitin Saxena for the opportunity to visit Hausdorff Center for Mathematics which started the collaborative work described in Chapter 5. The work presented in Chapter 6 is also done in collaboration with Nitin. I must admit, I have learnt quite a bit of algebra from many hours of illuminating discussions with him. His enthusiasm for algebraic methods is inspiring. I also thank him for his hospitality during my stay in Bonn.

Thanks to my colleagues and friends Anindya De and Ramprasad Saptharishi with whom I have collaborated with for the work in Chapter 4. A special thanks to Ramprasad with whom I have had many hours of insightful conversations. I do appreciate his clear thinking and simple ways of expressing non-trivial ideas.

I am greatly thankful to Marek Karpinski for many insightful discussions during the early stages of the work presented in Chapter 5, and also for his kind hospitality during my stay in Bonn. I am also very much thankful to Joachim von zur Gathen for kindly inviting me to give a talk at b-it, where I also enjoyed a brief but elucidating conversation with Joachim and Dima Grigoriev on the work in Chapter 4. Thanks to Martin Fürer for generously sharing his views on his integer multiplication algorithm and also for pointing out a mistake in referring a paper in our work.

I would also like to thank Comandur Seshadhri for explaining to me his work on identity testing with Nitin Saxena, to Srikanth Srinivasan for stimulating conversations on complexity theory, to Hendrik W. Lenstra for an e-mail communication (with Manindra) clarifying our doubt regarding existence of a certain result in the literature, and to Andrew Yao and Jayalal Sarma for kindly inviting me to a workshop at Tsinghua University. A special thanks to Neeraj Kayal for sharing with me his insights on several interesting problems, and for kindly inviting me to visit Microsoft Research India. Thanks to Samiran Chattopadhyay from Jadavpur University, the institute of my undergraduate study, for inspiring us to work sincerely.

Many thanks to the anonymous referees who have kindly taken some time out to review this work - their suggestions have improved the presentation of this thesis.

Life at IIT would not have been the same without my friends around me and I feel fortunate to have a good many of them. Without naming them individually, I express my deepest gratitude to all of them for making my stay in IIT a memorable one. Still I feel compelled to especially thank Deepanjan Kesh and Ramprasad Saptharishi for being always there to cheer me up. And thanks also to my office-mate Purushottam Kar - his sincerity is contagious.

Finally, it was all made possible for me by my family's continual and unconditional love and support. I thank my sister for her love and care, and also for originally instilling in me an interest for research. Thanks to my brother (in-law) for being a wonderful person whom I have always found refreshing to talk to. I am deeply indebted to my parents for being what they are, for loving me, caring for me and encouraging me to pursue whatever I am interested in. I do not know whether it is possible for me to return an iota of what I have received from them - perhaps it is not. Still, I will feel content to dedicate this little piece of work to them, and I do so with humility.

To my parents

Contents

Chapter 1

Introduction

Much of our endeavor in the theoretical study of computation is aimed towards either finding an efficient algorithm for a problem or gauging the *hardness* of a problem. And in meeting both these goals mathematical insights and ingenuities are constant companions. In particular, two branches of mathematics - combinatorics, and algebra and number theory, have found extensive applications in theoretical computer science. In this thesis, our focus is on problems belonging to the latter branch.

For the past few decades there has been a growing interest among computer scientists and mathematicians, in the field of computational number theory and algebra. Computational number theory is the branch of computer science that involves finding efficient algorithms for algebraic and number theoretic problems. Since its inception in the early 1960s, this field has continued to grow with ever-rising interest among researchers from diverse disciplines that resulted in a fruitful union of different areas in mathematics and computer science, especially algebra, number theory and computational complexity theory.

Factoring large integers, checking if an integer is prime, factoring polynomials, multiplying large integers and matrices, and solving polynomial equations are a few among a plethora of problems that have made this area so rich and fascinating. Unlike numerical analysis, here we are interested in exact solutions to problems instead of approximate solutions. Owing to the fundamental nature of the problems involved, this is a subject of intense theoretical pursuit. And the tools and techniques developed to solve these problems have provided researchers with deep mathematical

insights. But interest in them has escalated in recent time because of their important applications in key areas like cryptography, coding theory and complexity theory.

To cite a few examples, the security and efficiency of cryptographic protocols such as the RSA cryptosystem and the Diffie-Hellman key exchange protocol, rely on the hardness of problems like integer factoring and discrete logarithm and on the efficiency of prime number generation and large integer multiplication. Further, the efficiency of some of the decoding algorithms for error correcting codes like Reed-Solomon and BCH codes, hinge on fast algorithms for solving a system of linear equations and factoring polynomials over finite fields. Another important application of algorithmic algebra is the development of computer algebra systems. These systems are indispensable tools for research in many computation intensive fields of physics, chemistry, biology, mathematics, geology and meteorology.

The basic computer algebra operations can be broadly classified as -

- **Polynomial operations:** Polynomial addition, multiplication, gcd computation, factoring, interpolation, multipoint evaluation, identity testing.
- **Integer operations:** Addition, multiplication, gcd computation, square root finding, primality testing, integer factoring, etc.
- **Linear algebra operations:** Matrix addition, multiplication, inverse and determinant computation, solving a system of linear equations, etc.
- **Abstract algebra operations:** Finding the order of a group element, computing discrete logarithm, etc.

In this thesis, we study three such operations namely, univariate polynomial factoring over finite fields, large integer multiplication and polynomial identity testing. Our attempt in understanding the computational complexity of these problems is driven both by the fundamental nature of these problems as well as by their striking applications in other areas. In this chapter we formally define these problems, noting alongside a few motivating applications. We then give an overview of our results with the intent of placing them in the context of earlier work.

1.1 The Problems

We now state the problems that we work with in this thesis. Definition of a finite field can be looked up from Section 2.1.1 in Chapter 2.

Problem 1.1.1. (Polynomial factoring) *Given a univariate polynomial f of degree n with coefficients taken from a finite field $\mathbb{F}_q$, the field with q elements, find all the irreducible factors of f.*

Polynomial factoring is a fundamental problem that has been studied by the research community for over four decades. So far there is no known deterministic polynomial time solution. Polynomial f is given as an input in terms of all its n coefficients. Since $\mathbb{F}_q$ has q elements, each coefficient can be represented by about $\lceil \log q \rceil$ bits. So a polynomial time algorithm must run in time $(n \log q)^c$, where c is an absolute constant independent of n and $\log q$. Problem 1.1.1 is known to admit efficient randomized polynomial time algorithms (Ber67; Ber70; vzGS92; KS98; KU08). However, the popular belief is that 'a problem with a randomized polynomial time solution can also be solved in deterministic polynomial time' and hence our focus is on deterministic solutions to Problem 1.1.1.

Polynomial factoring finds important applications in coding theory, as in the list decoding algorithms of Reed-Solomon codes (Sud97; GS99), and also in designing efficient algorithms for other algebraic problems like polynomial solvability (Kay05; KY08).

Our second problem is integer multiplication.

Problem 1.1.2. (Integer multiplication) *Find an efficient algorithm to multiply two n-bit integers.*

The high school algorithm to multiply two integers is efficient when the integers involved are small, but it quickly becomes inapplicable for larger integers. Multiplication of large integers do arise in practice. For example, the RSA encryption process starts by multiplying two large primes. Many other cryptosystems also require to generate large primes, and choosing primes is usually accompanied by primality testing. The only known deterministic polynomial time primality test is the AKS family (AKS04) of tests. Crandall and Papadopoulos (CP03) remarked on

their AKS implementation, "...in our implementation almost all of the time is spent multiplying (and squaring) large integers."

Our third problem is polynomial identity testing. Arithmetic circuits are defined in Section 2.1.2 of Chapter 2.

Problem 1.1.3. (Polynomial Identity Testing) *Given an arithmetic circuit C with input variables $x_1, \ldots, x_n$ and constants taken from a field $\mathbb{F}$, check if the polynomial $f(x_1, \ldots, x_n)$ computed by C is identically zero.*

Beside being a natural problem in algebraic computation, identity testing appears in important complexity theory results such as, IP = PSPACE (LFKN90; Sha90) and the PCP theorem (ALM+98). It also plays a promising role in proving super-polynomial circuit lower bound for the permanent polynomial (KI03; Agr05). Moreover, algorithms for problems like primality testing (AKS04), graph matching (Lov79) and multivariate polynomial interpolation (CDGK91) also involve identity testing. Several efficient randomized algorithms (Sch80; Zip79; CK97; LV98; AB99; KS01) are already known for identity testing. However, a deterministic polynomial time algorithm has remained elusive. In this thesis, we are particularly interested in a special case of identity testing that has received a lot of attention in recent times. This is the problem of identity testing for circuits of depth 3.

We now move on to give a summary of our contributions to the above mentioned problems.

1.2 Our Contributions

The purpose of this section is to present a brief overview of our results, all of which are directed towards finding efficient deterministic algorithms for the problems stated in the previous section. The definitions of the basic terminologies, like algebra, endomorphism, zero-divisor, Fourier transform, circuits, etc., can be looked up from Sections 2.1 and 2.3 in Chapter 2.

1.2.1 Factoring Polynomials using Balance Test

As stated before, univariate polynomial factoring over finite fields is yet to be solved in deterministic polynomial time, even under the powerful assumption of the Extended Riemann Hypothesis (ERH). Without the assumption of the ERH, it is not even known how to efficiently find square root of an element $a \in \mathbb{F}_p$, which can be equivalently thought of as factoring polynomial $x^2 - a$. The results in this section assume the validity of the ERH.

The best deterministic factoring algorithm, due to Evdokimov (Evd94), runs in time polynomial in $n^{\log n}$ and $\log p$, where n is the degree of the input polynomial $f(x)$ and p is the characteristic of the finite field $\mathbb{F}_p$. Evdokimov's algorithm involves finding factors of polynomials of 'smaller' degree but with coefficients coming from algebras of dimension $O(n^{\log n})$ over $\mathbb{F}_p$. Using these factors eventually a nontrivial endomorphism of the ring $\mathcal{R} = \frac{\mathbb{F}_p[x]}{(f)}$ is found, or in the process a zero-divisor of $\mathcal{R}$ is encountered. Evdokimov showed that both these cases are sufficient to factor f.

An alternative approach, due to Gao (Gao01), is to exploit an inherent asymmetry among the n roots of f to find a zero-divisor in $\mathcal{R}$. This approach has its merit as it avoids computation in algebras of superpolynomial dimension over $\mathbb{F}_p$. However, Gao's algorithm fails to factor f if its roots satisfy a symmetry condition, known as *square balance*. Square balanced polynomials do exist, although we show in our work that the fraction of such polynomials is exponentially small in n over fields of characteristic $p = 3 \bmod 4$.

In our work (Sah08), we propose an extension of Gao's algorithm in a way that attempts to bring together the merits of both Evdokimov's and Gao's approaches. Our primary motivation in unifying the two approaches lies in the following informal observation. If the number of roots, i.e. n is 'large' then they are 'unlikely' to satisfy a sufficiently strong symmetry condition. Else, if n is small then Evdokimov's algorithm is efficient by itself. We briefly sketch how this observation is put to work.

Our algorithm (implicitly) constructs k simple digraphs $G_1, \ldots, G_k$ on n vertices (labelled $1, \ldots, n$) such that G_ℓ is a subgraph (not necessarily a proper subgraph) of $G_{\ell-1}$. There is an edge (i, j) in G_ℓ if the difference of the i^{th} and the j^{th} roots of f satisfy a 'certain' condition. We show that if any of the k graphs is not regular (i.e.

indegree and outdegree not same for all vertices) then a zero-divisor of $\mathcal{R}$ is found. Otherwise, if all the graphs $G_1, \ldots, G_k$ are regular then we require at most $\log_2 n$ of the graphs G_ℓ to be such that $G_\ell \neq G_{\ell-1}$, so that a nontrivial endomorphism of $\mathcal{R}$ is obtained. As mentioned earlier, by the work of Evdokimov (Evd94), a zero-divisor or an endomorphism in turn produces a factor of f. The time complexity of our algorithm is $k \cdot (n \log p)^{O(1)}$. The graph G_1 is regular exactly for the class of square balanced polynomials, which makes the test that $G_1, \ldots, G_k$ are regular (we call, *balance test*) stronger than Gao's test.

The construction of the graphs is flexible in the sense that any arbitrary but deterministically chosen auxiliary polynomials $q_1(y), \ldots, q_k(y)$ of degree $(n \log p)^{O(1)}$ can be used to form the graphs. For instance, by choosing $q_\ell(y) = (y + 2^{-1}\ell)^2$ a bound of $k \leq \sqrt{p} \log p$ can be shown for $p = 3 \mod 4$, so that our algorithm always succeeds in factoring f for this value of k. However, the task is to show better bounds on k and we leave this question open, noting that for a random choice of $q_\ell(y)$, $G_\ell \neq G_{\ell-1}$ with high probability. The ideal goal is to show that $k = (n \log p)^{O(1)}$ by appropriately fixing the auxiliary polynomials, in which case factoring polynomials under ERH can be solved in deterministic polynomial time.

Theorem 1.2.1. *Univariate polynomials over finite fields can be factored in deterministic polynomial time, under the assumption of the ERH, unless the roots of the polynomial satisfy a strong symmetry condition.*

1.2.2 Integer Multiplication using Modular Arithmetic

How fast can we multiply two n-bit integers? In a seminal paper (SS71), Schönhage and Strassen gave two algorithms to multiply integers - one having a time complexity of $O(n \cdot \log n \cdot \log\log n \ldots 2^{O(\log^* n)})$ bit operations, and another having a complexity of $O(n \cdot \log n \cdot \log\log n)$ bit operations. The first algorithm involves arithmetic over complex numbers while the second one uses modular arithmetic. However, both these algorithms are based on one central theme - reduce integer multiplication to polynomial multiplication and then multiply the polynomials using Fast Fourier Transform. The main technical step here is to suitably encode the integers as polynomials over a ring that has a 'good' principal root of unity which is crucial in making Discrete Fourier Transform (DFT) efficient.

After a period of dormancy in progress, Fürer came up with a breakthrough result (Für07) that multiplies two n-bit integers using $n \cdot \log n \cdot 2^{O(\log^* n)}$ bit operations. While Fürer's overall approach remains the same as in (SS71), he quite importantly introduced the notion of inner and outer DFT and showed how repeated computation of inner DFTs, efficiently, leads to an overall saving in time. For the inner DFT idea to work, an appropriate root of unity is required which is less of an issue here as the underlying field of complex numbers is algebraically closed. However, all intermediate complex numbers need to be approximated suitably during computation and this introduces the added task of bounding the total truncation error in the analysis of the algorithm. The question that motivated us is - Can we dispense with the elaborate error analysis by giving a 'discrete' adaption of Fürer's algorithm?

In our work (DKSS08), we show that two n-bit integers can be multiplied in $n \cdot \log n \cdot 2^{O(\log^* n)}$ time using modular arithmetic. Our algorithm closely follows Fürer's algorithm, but departs from it by using modular arithmetic for intermediate computations instead of complex arithmetic. The primary technical hurdle here is to make the inner DFTs efficient while working over a discrete ring. This we overcome by invoking a deep result in analytic number theory on the least prime in an arithmetic progression (Linnik's theorem), and by using FFT over multivariate polynomials as opposed to univariate-FFT, which is the case in both Fürer's and Schönhage-Strassen's algorithms. This also means that the notion of inner and outer DFT has to be translated properly to make it work in the multivariate world. Overall, the algorithm becomes more transparent and simple (without any truncation error analysis), making it plausible that a suitable variant of it might lead to an $O(n \log n)$ algorithm, thereby matching the popular belief on integer multiplication complexity.

Theorem 1.2.2. *There is an $n \cdot \log n \cdot 2^{O(\log^* n)}$ time algorithm to compute the product of two n bit integers using only modular arithmetic for intermediate computations.*

1.2.3 Identity Testing via Depth 2 Circuits over Algebras

Recall that, Polynomial Identity Testing (PIT) is the problem of efficiently deciding if a multivariate polynomial given as an arithmetic circuit is identically zero.

The goal is to find a deterministic polynomial time algorithm. One specific case of PIT is identity testing for depth-3 ($\Sigma\Pi\Sigma$) arithmetic circuits. In the last few years depth-3 PIT has been intensely studied (DS05; KS07; SS09; KS09; SS10b; SS10a) by the research community. The best result known along this line of research is a deterministic polynomial time 'blackbox' algorithm for PIT of $\Sigma\Pi\Sigma$ circuits with bounded top fan-in. However, PIT of general $\Sigma\Pi\Sigma$ circuits remains a long standing open problem.

In our work (SSS09), we take a different approach to depth-3 identity testing by reducing the depth of the circuit to 2 while increasing the dimension of the underlying algebra. We show that identity testing of depth 3 ($\Sigma\Pi\Sigma$) arithmetic circuits over a field $\mathbb{F}$ is polynomial time equivalent to identity testing of depth 2 ($\Pi\Sigma$) arithmetic circuits over $\mathsf{U}_2(\mathbb{F})$, the algebra of upper-triangular 2×2 matrices with entries from $\mathbb{F}$. Such a connection is a bit surprising since we also show that, as computational models, $\Pi\Sigma$ circuits over $\mathsf{U}_2(\mathbb{F})$ are strictly 'weaker' than $\Sigma\Pi\Sigma$ circuits over $\mathbb{F}$. The equivalence further implies that PIT of $\Sigma\Pi\Sigma$ circuits reduces to PIT of width-2 commutative *Algebraic Branching Programs*(ABP). This is in contrast to the fact that identity testing of any 'non-commutative' ABP can be done in deterministic polynomial time (RS04) and the reduction justifies to some extent the lack of progress for commutative ABPs.

We also show that PIT of a depth-3 circuit of size s reduces in polynomial time to PIT of a depth-2 circuit of size $O(s)$ over a commutative algebra of dimension $O(s)$. We make some progress along this line by giving a deterministic polynomial time identity testing algorithm for depth-2 circuits of size s over commutative algebras of dimension $O(\log s/\log\log s)$. The main technical ingredient of our algorithm is an effective version of a structure theorem that states that a finite dimensional commutative algebra splits into local rings.

Thus, if we can extend the above result to $O(s)$ dimension or use deeper algebraic insight into the ring of 2×2 matrices to solve PIT of $\Pi\Sigma$ circuits over $\mathsf{U}_2(\mathbb{F})$, then depth-3 PIT can be solved in polynomial time.

Theorem 1.2.3. (a) *Identity testing of depth-3 circuits is polynomial time equivalent to identity testing of depth-2 circuits over 2×2 upper-triangular matrices,*

although the latter model is 'computationally weaker' than the former.
(b) *Also, PIT of depth-3 circuits of size s reduces to PIT of depth-2 circuits of size $O(s)$ over commutative algebras of dimension $O(s)$. There is a deterministic polynomial time identity testing algorithm for depth-2 circuits of size s over commutative algebras of dimension $O(\frac{\log s}{\log\log s})$.*

1.2.4 Applying Duality to Two Identity Testing Problems

Finally, we study the complexity of two particular problems on identity testing. The first problem is a natural generalization of a problem on depth-3 identity testing studied before by Kayal and Saxena (KS07).

Problem 1.2.4. *Given a depth-3 circuit C with bounded top fanin and given a sparse polynomial f explicitly, check if $p(C) = f$, where $p(C)$ is the polynomial computed by C.*

Kayal and Saxena (KS07) showed that identity testing of a depth-3 circuit with bounded top fanin can be solved in deterministic polynomial time, which thereby solves the case when $f = 0$ in Problem 1.2.4 (Recently, Saxena and Seshadhri (SS10a) gave a blackbox polynomial time algorithm to check if $p(C) = 0$). We generalize their result using a technique known as *dual representation* of polynomials (introduced in (Sax08)), to show that Problem 1.2.4 can be solved in deterministic polynomial time for any given sparse polynomial f.

The second problem we study is also a very natural case of identity testing.

Problem 1.2.5. *Given a polynomial f explicitly, and also given t other polynomials $g_1, \ldots, g_t$ explicitly (g_i's need not be distinct), check if $f = g_1 \ldots g_t$.*

This problem has been mentioned in a work by von zur Gathen (vzG83) as a problem with no known efficient deterministic solution. Using the same tool of 'duality', we give a deterministic polynomial time algorithm to solve Problem 1.2.5 for the case when the input factors $g_1, \ldots, g_t$ are of the form of sum of univariates.

Roughly speaking, duality is an efficient technique to express a polynomial of a special kind, we call a *semidiagonal* polynomial, as a 'small' sum of product of univariates. This particular representation of semidiagonal polynomials turns out to be quite useful in solving both the above problems.

1.3 Organization

The rest of the thesis is organized as follows. In Chapter 2 we give a brief overview of the basic algebraic concepts and tools used in our work. Chapter 3 and Chapter 4 are devoted to our results on polynomial factoring and integer multiplication, respectively. The results on identity testing are covered in Chapter 5 and Chapter 6.

Chapter 2

Preliminaries

In this chapter we give a concise introduction to the basic mathematical constructs and tools that we allude to throughout the rest of this thesis. In the first section, we define the algebraic and the computational structures and note some of their relevant properties. While in the second section, we provide a more or less self contained exposition to some of the standard mathematical tools that play an important role in our results. The details provided in this section are primarily for completeness and the reader's convenience.

2.1 Basic Structures

2.1.1 Rings and Fields

In this section, we collect the definitions of some of the basic algebraic structures, often accompanying them with simple representative examples. Extensive treatment of these concepts can be found in any standard textbook (Lan02; DF99; Her75; Art91) on abstract algebra.

Definition 2.1.1. (Group) *A group $\mathcal{G}$ is a set of elements along with a binary operator $\bullet$ that satisfies the following properties -*

1. (Closure) *For every two elements $a, b \in \mathcal{G}$, $a \bullet b$ also belongs to $\mathcal{G}$.*

2. (Associativity) *For every three elements $a, b, c \in \mathcal{G}$, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.*

3. (Identity) *There is an element* $e \in \mathcal{G}$ *such that for all* a, $a \bullet e = e \bullet a = a$.

4. (Inverse) *Every element* a *has an inverse* $b \in \mathcal{G}$ *satisfying* $a \bullet b = b \bullet a = e$.

Example - (*Symmetric groups*) The set of all bijections from a set onto itself is a group, where the binary operator $\bullet$ is composition of mappings.

A group, often written as a tuple $(\mathcal{G}, \bullet)$, is *commutative* or *abelian* if for every $a, b \in \mathcal{G}$, $a \bullet b = b \bullet a$. If $\mathcal{G}$ satisfies only the closure and the associativity properties under $\bullet$, then $(\mathcal{G}, \bullet)$ is called a *semigroup*.

Definition 2.1.2. (Ring) *A ring* $\mathcal{R}$ *is a set of elements along with two binary operators* $+$ *and* $\bullet$ *such that -*

1. $(\mathcal{R}, +)$ *is an abelian group.*

2. $(\mathcal{R} - \{0\}, \bullet)$ *is a semigroup, where* 0 *is the identity of* $(\mathcal{R}, +)$.

3. For every a, b *and* $c \in \mathcal{R}$, $(a + b) \bullet c = a \bullet c + b \bullet c$ *and* $a \bullet (b + c) = a \bullet b + a \bullet c$.

Example - (*Integer and polynomial ring*) The set of integers $\mathbb{Z}$ and the set of polynomials with integer coefficients $\mathbb{Z}[x]$ are rings under normal integer and polynomial addition and multiplication, respectively.

Some ring-theoretic concepts - A ring $\mathcal{R}$ is *commutative* if for all $a, b \in \mathcal{R}$, $a \bullet b = b \bullet a$. Ring $\mathcal{R}$ is said to have a *unity*, if there is an element $e \in \mathcal{R}$ for which $a \bullet e = e \bullet a = a$, for all $a \in \mathcal{R}$. By convention, the unity in $\mathcal{R}$ is denoted by 1 and the identity element of $(\mathcal{R}, +)$ is denoted by 0. An element $a \neq 0$ is a *zero divisor* if there exist a $b \neq 0$ such that $a \bullet b = 0$; and a is *nilpotent* if $a^n = 0$ for a positive integer n. A commutative ring with no zero divisor is called an *integral domain*. Elements, in a ring $\mathcal{R}$ with unity, which are invertible in $(\mathcal{R} - \{0\}, \bullet)$ are called *units*. For two different rings $(\mathcal{R}_1, +, \bullet)$ and $(\mathcal{R}_2, +, \bullet)$, although we use the same notation to denote the different rings operations $+$ and $\bullet$, they should be interpreted appropriately from the context of their usage. The *direct sum* of two rings $\mathcal{R}_1$ and $\mathcal{R}_2$, denoted as $\mathcal{R}_1 \oplus \mathcal{R}_2$, is a ring $\mathcal{R}$ with elements of the form $r = (r_1, r_2)$ for all $r_1 \in \mathcal{R}_1$ and $r_2 \in \mathcal{R}_2$. Addition and multiplication in $\mathcal{R}$ is defined as componentwise addition and multiplication over $\mathcal{R}_1$ and $\mathcal{R}_2$ i.e. if $r = (r_1, r_2)$ and $r' = (r'_1, r'_2)$ then

$r + r' = (r_1 + r'_1, r_2 + r'_2)$ and $r \cdot r' = (r_1 \cdot r'_1, r_2 \cdot r'_2)$. Surely, this definition extends to direct sum of multiple rings.

Definition 2.1.3. (Ring homomorphism) *A mapping ϕ from a ring $(\mathcal{R}_1, +, \cdot)$ to a ring $(\mathcal{R}_2, +, \cdot)$ is a ring homomorphism if for every a and b in $\mathcal{R}_1$,*

1. $\phi(a + b) = \phi(a) + \phi(b)$.
2. $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$.
3. $\phi(1) = 1$.

The set of elements of $\mathcal{R}_1$ which map to the element 0 in $\mathcal{R}_2$, under the action of a ring homomorphism ϕ, is called the *kernel* of ϕ denoted by $\mathsf{ker}(\phi)$. When a ring homomorphism ϕ is one-one and onto, it is called an *isomorphism* and the rings are then said to be *isomorphic*. A homomorphism from $\mathcal{R}$ to itself is called an *endomorphism*, and an endomorphism that is also an isomorphism is called an *automorphism*.

Example - (*Residues modulo n*) Given a positive integer n, let $\mathbb{Z}/n\mathbb{Z} = \{0, \ldots, n-1\}$ be the ring of *residues* under addition and multiplication modulo n. Then the map which takes $a \in \mathbb{Z}$ to $a \pmod n$ i.e. the (positive) remainder when a is divided by n, is a ring homomorphism from $\mathbb{Z}$ to $\mathbb{Z}/n\mathbb{Z}$.

Definition 2.1.4. (Ideal) *A subset $\mathcal{I}$ of a ring $(\mathcal{R}, +, \cdot)$ is an ideal of $\mathcal{R}$ if $(\mathcal{I}, +)$ is a subgroup of $(\mathcal{R}, +)$ and for every $a \in \mathcal{I}$ and $b \in \mathcal{R}$, $b \cdot a$ and $a \cdot b$ belong to $\mathcal{I}$.*

An ideal of $\mathcal{R}$ which is not properly contained in any ideal other than $\mathcal{R}$ itself, is called a *maximal* ideal. We say that an ideal $\mathcal{I}$ is *generated* by the elements $a_1, \ldots, a_k \in \mathcal{I}$, denoted as $\mathcal{I} = (a_1, \ldots, a_k)$, if and only if every $b \in \mathcal{I}$ can be expressed as $b = b_1 \cdot a_1 + \ldots + b_k \cdot a_k$ where b_i's belong to $\mathcal{R}$.

Definition 2.1.5. (Quotient ring) *Given an ideal $\mathcal{I}$ of a ring $\mathcal{R}$, the quotient ring $\mathcal{R}/\mathcal{I}$ is defined on the set of elements $\{a+\mathcal{I} \mid a \in \mathcal{R}\}$ satisfying the following relations for all $a, b \in \mathcal{R}$:*

1. $a + \mathcal{I} = b + \mathcal{I}$ *if and only if* $a - b \in \mathcal{I}$.
2. $(a + \mathcal{I}) + (b + \mathcal{I}) = (a + b) + \mathcal{I}$.

3. $(a + \mathfrak{I}) \bullet (b + \mathfrak{I}) = (a \bullet b) + \mathfrak{I}$.

Example - The set of integer multiples of a number $n \in \mathbb{Z}$, denoted as $n\mathbb{Z}$, is an ideal of $\mathbb{Z}$. If n is a prime then $n\mathbb{Z}$ is a maximal ideal. The ring $\mathbb{Z}/n\mathbb{Z}$ of residues modulo n is a quotient ring with respect to the ideal $n\mathbb{Z}$.

Definition 2.1.6. (Primitive root of unity) *Let $\mathcal{R}$ be a commutative ring with unity 1. An element $\omega \in \mathcal{R}$ is called a primitive n^{th} root of unity if $\omega^n = 1$, the element $n = 1 + 1 + \ldots n$ times is a unit in $\mathcal{R}$, and for every $1 \leq m < n$, $\omega^m - 1$ is nonzero and not a zero divisor.*

Observation 2.1.7. *If ω is a primitive n^{th} root of unity in $\mathcal{R}$ then for every $1 \leq m < n$, $\sum_{j=0}^{n-1} \omega^{mj} = 0$.*

Proof. Notice that, $(\omega^m - 1) \cdot \sum_{j=0}^{n-1} \omega^{mj} = \omega^{mn} - 1 = 0$ and $\omega^m - 1 \neq 0$ is not a zero divisor, implying that $\sum_{j=0}^{n-1} \omega^{mj} = 0$. □

Example - In the ring $\mathbb{Z}/5\mathbb{Z}$, the element 2 is a primitive 4^{th} root of unity.

Definition 2.1.8. (Local ring) *A commutative ring $\mathcal{R}$ is local if it has a unique maximal ideal.*

Example - Let $\mathcal{R} = \mathbb{Q}[x]$ be the ring of univariate polynomials in x with rational coefficients, and $x^n\mathcal{R}$ be the ideal of all polynomials that are divisible by x^n, where n is a positive integer. Then the quotient ring $\mathcal{R}/x^n\mathcal{R}$ is a local ring with exactly one maximal ideal consisting of all polynomials that are divisible by x.

Definition 2.1.9. (Field) *A ring $(\mathcal{R}, +, \bullet)$ is a field if $(\mathcal{R} - \{0\}, \bullet)$ is an abelian group. In other words, a commutative ring $\mathcal{R}$ with unity is a field if every nonzero element in $\mathcal{R}$ is invertible.*

Example - (*Polynomial ring and rational functions*) The sets of rationals, reals and complex numbers are fields. The set of polynomials in the indeterminates $x_1, \ldots, x_n$ with coefficients taken from a field $\mathbb{F}$, denoted as $\mathbb{F}[x_1, \ldots, x_n]$, is a ring under 'natural' polynomial addition and multiplication. This is called the *polynomial ring*. Then the set $\mathbb{F}(x_1, \ldots, x_n) = \{f/g : f, g \in \mathbb{F}[x_1, \ldots, x_n] \text{ and } g \neq 0\}$ is a field also known as the field of *rational functions* or the quotient field of the polynomial ring.

Definition 2.1.10. (Finite field) *A field with finite number of elements is called a finite field.*

Example - For every prime p, the quotient ring $\mathbb{Z}/p\mathbb{Z}$ is a field consisting of p elements. Let $\mathbb{F}$ be a finite field with q elements and $f(x) \in \mathbb{F}[x]$ be an *irreducible polynomial* (see the definition below) of degree n. Then $\mathbb{F}[x]/(f)$ is a field with q^n elements, where (f) is the ideal of $\mathbb{F}[x]$ consisting of all multiples of f.

If p is the prime for which $1+1+\ldots p$ times $= 0$ in $\mathbb{F}$ then p is called the *characteristic* of $\mathbb{F}$. If no such prime exists then field $\mathbb{F}$ is said to have zero characteristic.

Definition 2.1.11. (Unique factorization domain) *Let $\mathcal{R}$ be an integral domain with unity. An element $p \in \mathcal{R}$ is irreducible if whenever $p = a \cdot b$ with $a, b \in \mathcal{R}$, either a or b is a unit. Ring $\mathcal{R}$ is a unique factorization domain if every nonunit $r \in \mathcal{R}$ can be uniquely expressed as a finite product of irreducible elements. The uniqueness of the product is up to multiplication by units.*

Example - The polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ over a field $\mathbb{F}$ is a unique factorization domain where every polynomial can be factored uniquely into irreducible factors.

Definition 2.1.12. (Vector space) *A commutative group $(\mathcal{V}, +)$ is a vector space over a field $\mathbb{F}$ if for every $a \in \mathbb{F}$ and $v \in \mathcal{V}$ there is an element $av \in \mathcal{V}$ satisfying the following relations. For all $a, b \in \mathbb{F}$ and $v, w \in \mathcal{V}$,*

1. $a(v + w) = av + aw$.
2. $(a + b)v = av + bv$.
3. $a(bv) = (ab)v$.
4. $1v = v$.

Example - The polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ is a vector space over $\mathbb{F}$. The set of $n \times n$ matrices with entries from a field $\mathbb{F}$, denoted as $\mathsf{M}_n(\mathbb{F})$, is also a vector space over $\mathbb{F}$. Also, the set of complex numbers is a vector space over reals.

Vectors $v_1, \ldots, v_m \in \mathcal{V}$ are said to be *linearly independent* if any vector of the form $a_1v_1 + \ldots a_mv_m$, with a_i's in $\mathbb{F}$, cannot be 0 unless every a_i is zero. Elements

$e_1, \ldots, e_n \in \mathcal{V}$ are said to form a *basis* of $\mathcal{V}$ if every $v \in \mathcal{V}$ can be expressed as $a_1e_1 + \ldots a_ne_n$, where a_i's in $\mathbb{F}$ are unique depending only on v. In this case, n is called the dimension of the vector space $\mathcal{V}$.

Definition 2.1.13. (Algebra over fields) *A ring $(\mathcal{A}, +, \bullet)$ is called an algebra over a field $\mathbb{F}$ if $(\mathcal{A}, +)$ forms a vector space over $\mathbb{F}$ and for every $v, w \in \mathcal{A}$ and $a \in \mathbb{F}$, $a(v \bullet w) = (av) \bullet w = v \bullet (aw)$.*

Example - The quotient ring $\mathbb{F}[x]/(f)$, where f is a polynomial of degree n is an algebra over $\mathbb{F}$ of dimension n. The set of $n \times n$ matrices, $\mathsf{M}_n(\mathbb{F})$, forms an algebra over $\mathbb{F}$ of dimension n^2.

We say that an $\mathbb{F}$-basis of an algebra is *explicitly* given if we know the basis elements (say) $\{e_1, \ldots, e_n\}$ and we also know how $e_i \bullet e_j$ expands as an $\mathbb{F}$-linear combination of $\{e_1, \ldots, e_n\}$, for every i and j. The *direct sum* of two algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ is an algebra $\mathcal{A} = \mathcal{A}_1 \oplus \mathcal{A}_2$, which is the same as the direct sum of rings $\mathcal{A}_1$ and $\mathcal{A}_2$ with the added property that if $v = (v_1, v_2)$, where $v_1 \in \mathcal{A}_1$, $v_2 \in \mathcal{A}_2$ and $v \in \mathcal{A}$, then $av = (av_1, av_2)$ for every $a \in \mathbb{F}$.

2.1.2 Arithmetic Circuits

Arithmetic circuits form a natural model for computing multivariate polynomials. It is the arithmetic analogue of boolean circuits, the nonuniform version of Turing machines. Problems involving arithmetic circuits, in particular proving explicit circuit lower bounds, have been studied since the early 70s. Formally, an arithmetic circuit is defined as follows.

Definition 2.1.14. (Arithmetic circuit) *An arithmetic circuit over a field $\mathbb{F}$ is a directed acyclic graph with nodes labelled by the two operations $+$ and $\times$, while nodes with indegree zero are labelled by the input variables $x_1, \ldots, x_n$ and by field elements. The edges are labelled only by field elements (no label indicates a label by 1). Circuit C computes polynomials in $\mathbb{F}[x_1, \ldots, x_n]$ in a natural way. The output of nodes labelled by x_i (or, a constant) are x_i (or, the constant). An edge (v, w) which is labelled by $\alpha \in \mathbb{F}$ multiplies the output of v with α and feeds in to the input of w. Nodes labelled by $+$ and $\times$ output the sum and the product of the*

corresponding input polynomials, respectively. Nodes with outdegree zero output the final polynomials computed by the circuit.

The following figure shows an example of a circuit. The *size* of a circuit is the total number of nodes and edges in the underlying directed graph. The *depth* of a circuit is the length of the longest path from an input to an output node. A circuit is called a *formula* if every node has outdegree at most one.

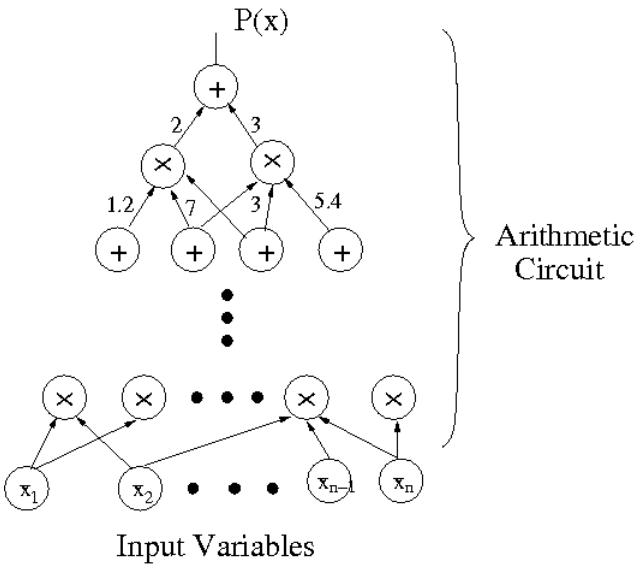

Figure 2.1: An arithmetic circuit.

Bounded depth circuits - An infinite family of circuits whose depths are bounded by a constant is called a family of bounded depth circuits. Interestingly enough, bounded depth circuits capture a great deal about circuits in general. This point will be explained in Chapter 5. The following is an example of a depth-3 circuit computing the polynomial $(x_1+x_2+3x_3)(x_2-2x_3)+x_1(4x_2-3x_3)+(x_1-x_2)(x_1-2x_3)$. The fanin of the topmost '+' gate is called the *top fanin* of the depth-3 circuit. In this case, the top fanin is 3. In Chapter 5 we will be interested in identity testing of depth-3 circuits.

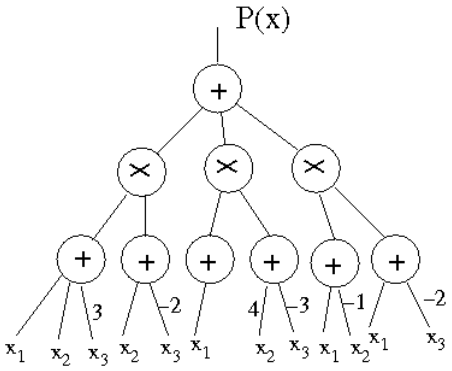

Figure 2.2: A depth-3 circuit.

2.2 Notations and Conventions

Useful sets

The sets of integers, rationals, reals and complex numbers are denoted by $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$, respectively. $\mathbb{Z}^+$ is the set of positive integers and $\mathbb{Z}_N$ is the ring of integers modulo $N \in \mathbb{Z}^+$. The multiplicative subgroup of $\mathbb{Z}_N$, consisting of all $m \in \mathbb{Z}^+$ with $\gcd(m, N) = 1$, is denoted by $\mathbb{Z}_N^\times$. $\mathbb{F}$ represents any arbitrary field, and $\mathbb{F}_q$ is the finite field with q elements. $\mathbb{F}_q^\times = \mathbb{F}_q \backslash \{0\}$ is the multiplicative subgroup of $\mathbb{F}_q$.

The Order notation

Given two functions $t_1(n)$ and $t_2(n)$ from $\mathbb{Z}^+ \to \mathbb{Z}^+$, we write $t_1(n) = O(t_2(n))$ if there exist positive constants c and n_0 such that $t_1(n) \leq c \cdot t_2(n)$ for every $n \geq n_0$. We write $t_1(n) = o(t_2(n))$ if for every constant c, there is an $n_0 > 0$ such that $t_1(n) < c \cdot t_2(n)$ for all $n \geq n_0$. We use the notation $\mathsf{poly}(n_1, \ldots, n_k)$ to mean some arbitrary but fixed polynomial in the parameters $n_1, \ldots, n_k$. Throughout this thesis, log refers to logarithm base 2 and ln is the natural logarithm. Sometimes, for the sake of brevity we use the notation $\tilde{O}(t(n))$ to mean $O(t(n) \cdot \mathsf{poly}(\log t(n)))$.

Other conventions

In this thesis, a ring $\mathcal{R}$ always means a commutative ring with unity 1. We write $\mathcal{R}_1 \cong \mathcal{R}_2$ to mean that the rings $\mathcal{R}_1$ and $\mathcal{R}_2$ are isomorphic. The characteristic of

a field $\mathbb{F}$ is denoted by $\text{char}(\mathbb{F})$. Given two polynomials $f, g \in \mathbb{F}[x]$, where $\mathbb{F}$ is any field, $\gcd(f, g)$ refers to the unique monic largest common divisor of f and g over $\mathbb{F}$. For any $f \in \mathbb{F}[x]$, f' denotes the formal derivative of f with respect to x i.e. $\frac{df}{dx}$. For a positive real a, $\lfloor a \rfloor$ is the largest integer less than a, and $\lceil a \rceil$ is the smallest integer greater than a. The determinant of a square matrix M is denoted by $\det(M)$. Given two polynomials $f, g \in \mathcal{R}[x]$, where $\mathcal{R}$ is an integral domain, $S(f, g)$ denotes the Sylvester matrix of f and g over $\mathcal{R}$. The resultant of f and g over $\mathcal{R}$ is $\mathsf{Res}_x(f, g) = \det(S(f, g))$. Refer to Appendix A.1, for a discussion on resultant.

2.3 Basic Tools

Our results rely heavily on certain fundamental mathematical results or *tools*, namely - Chinese Remaindering Theorem, Hensel Lifting, Discrete Fourier Transform, and a structure theorem on finite dimensional commutative algebra. This section is devoted to a brief study of these tools. More details on them can be found in the texts (GG03; Sho09; AM69) or in the lecture notes (Sud).

2.3.1 Chinese Remaindering

This is a structural result about rings which is used for speeding up computation over integers and polynomials, and also for arguing over rings as in some of the proofs in Chapter 3. For convenience, we present the theorem in a general form and then apply it to the ring of integers and the ring of polynomials.

Two ideals $\mathcal{I}$ and $\mathcal{J}$ of a ring $\mathcal{R}$ are *coprime* if there are elements $a \in \mathcal{I}$ and $b \in \mathcal{J}$ such that $a + b = 1$. The product of two ideals $\mathcal{I}$ and $\mathcal{J}$, denoted by $\mathcal{IJ}$, is the ideal generated by all elements of the form $a \cdot b$ where $a \in \mathcal{I}$ and $b \in \mathcal{J}$. The theorem states the following.

Theorem 2.3.1. (Chinese Remaindering Theorem) *Let $\mathcal{I}_1, \ldots, \mathcal{I}_r$ be pairwise coprime ideals of $\mathcal{R}$ and $\mathcal{I} = \mathcal{I}_1 \ldots \mathcal{I}_r$ be their product. Then,*

$$\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$$

Moreover, this isomorphism map is given by,

$$a \bmod \mathcal{I} \longrightarrow (a \bmod \mathcal{I}_1, \ldots, a \bmod \mathcal{I}_r)$$

for all $a \in \mathcal{R}$.

Proof. The proof uses induction on the number of coprime ideals. Let $\mathcal{J} = \mathcal{I}_2 \ldots \mathcal{I}_r$. Since $\mathcal{I}_1$ is coprime to $\mathcal{I}_j$ for every j, $2 \leq j \leq r$, there are elements $y_j \in \mathcal{I}_j$ and $x_j \in \mathcal{I}_1$ such that $x_j + y_j = 1$. Therefore, $\prod_{j=2}^{r}(x_j + y_j) = x + y' = 1$ where $x \in \mathcal{I}_1$ and $y' \in \mathcal{J}$, implying that $\mathcal{I}_1$ and $\mathcal{J}$ are coprime.

We claim that $\mathcal{I} = \mathcal{I}_1 \cap \mathcal{J}$. By definition, $\mathcal{I} = \mathcal{I}_1\mathcal{J}$ and it is easy to see that $\mathcal{I}_1\mathcal{J} \subseteq \mathcal{I}_1 \cap \mathcal{J}$. If $z \in \mathcal{I}_1 \cap \mathcal{J}$ then, from $x + y' = 1$ we have $zx + zy' = z$. The left hand side of the last equation is an element of $\mathcal{I}_1\mathcal{J}$ as both $zx, zy' \in \mathcal{I}_1\mathcal{J}$. Therefore, $\mathcal{I}_1 \cap \mathcal{J} = \mathcal{I}_1\mathcal{J} = \mathcal{I}$.

Consider the map $\phi : \frac{\mathcal{R}}{\mathcal{I}} \to \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \frac{\mathcal{R}}{\mathcal{J}}$ defined as $\phi(a \bmod \mathcal{I}) = (a \bmod \mathcal{I}_1, a \bmod \mathcal{J})$. It is easy to check that ϕ is well-defined and is in fact a homomorphism. Let $a_1 = a \bmod \mathcal{I}_1$ and $a' = a \bmod \mathcal{J}$. We will abuse notation slightly and write $\phi(a) = (a_1, a')$.

If $\phi(a) = \phi(b) = (a_1, a')$ then $a_1 = a \bmod \mathcal{I}_1 = b \bmod \mathcal{I}_1$, implying that $a - b \in \mathcal{I}_1$. Similarly, $a - b \in \mathcal{J}$. This means $a - b \in \mathcal{I}_1 \cap \mathcal{J} = \mathcal{I}$ and hence ϕ is a one-one map. Also, since $x + y' = 1$ for $x \in \mathcal{I}_1$ and $y' \in \mathcal{J}$, we have $\phi(a_1y' + a'x) = (a_1, a')$ implying that ϕ is onto. Therefore, ϕ is an isomorphism i.e. $\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \frac{\mathcal{R}}{\mathcal{J}}$. Inductively, we can show that $\frac{\mathcal{R}}{\mathcal{J}} \cong \frac{\mathcal{R}}{\mathcal{I}_2} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$ and hence, $\frac{\mathcal{R}}{\mathcal{I}} \cong \frac{\mathcal{R}}{\mathcal{I}_1} \oplus \ldots \oplus \frac{\mathcal{R}}{\mathcal{I}_r}$. □

In $\mathbb{Z}$ (or $\mathbb{F}[x]$), two elements m_1 and m_2 are coprime integers (or polynomials) if and only if the ideals (m_1) and (m_2) are coprime. Applying the above theorem to the ring of integers (or polynomials) we get the following result.

Corollary 2.3.2. *Let* $m \in \mathcal{R} = \mathbb{Z}$ *(or* $\mathbb{F}[x]$*) be such that* $m = \prod_{j=1}^{r} m_j$ *where* $m_1, \ldots, m_r$ *are pairwise coprime integers (or polynomials). Then* $\frac{\mathcal{R}}{(m)} \cong \frac{\mathcal{R}}{(m_1)} \oplus \ldots \oplus \frac{\mathcal{R}}{(m_r)}$.

Thus every element of the ring $\frac{\mathcal{R}}{(m)}$ can be uniquely written as an r-tuple with the i^{th} component belonging to the ring $\frac{\mathcal{R}}{(m_i)}$. Addition and multiplication in $\frac{\mathcal{R}}{(m)}$ are just component-wise addition and multiplication in the rings $\frac{\mathcal{R}}{(m_i)}$.

2.3.2 Hensel Lifting

Given a ring $\mathcal{R}$ and an element $m \in \mathcal{R}$, Hensel (Hen18) designed a method to compute factorization of any element of $\mathcal{R}$ modulo m^ℓ (for an integer $\ell > 0$), given its factorization modulo m. This method, known as Hensel lifting, is used in many algorithms including multivariate polynomial factoring and polynomial division. In this thesis, we use this tool in Chapter 4 to 'lift' a root of unity, as stated in Lemma 2.3.5. Once again, we present the general theorem first and then apply it to prove the case (Lemma 2.3.5) we need.

Lemma 2.3.3. (Hensel lifting) *Let $\mathcal{I}$ be an ideal of ring $\mathcal{R}$. Given elements $f, g, h, s, t \in \mathcal{R}$ with $f = gh \bmod \mathcal{I}$ and $sg + th = 1 \bmod \mathcal{I}$ there exist $g', h' \in \mathcal{R}$ such that,*

$$\begin{aligned} f &= g'h' \mod \mathcal{I}^2 \\ g' &= g \mod \mathcal{I} \\ h' &= h \mod \mathcal{I}. \end{aligned}$$

Further, any g' and h' satisfying the above conditions also satisfy:

1. *$s'g' + t'h' = 1 \bmod \mathcal{I}^2$ for some $s' = s \bmod \mathcal{I}$ and $t' = t \bmod \mathcal{I}$.*
2. *g' and h' are unique in the sense that any other solutions g^* and h^* satisfying the above conditions also satisfy, $g^* = (1 + u)g' \bmod \mathcal{I}^2$ and $h^* = (1 - u)h' \bmod \mathcal{I}^2$, for some $u \in \mathcal{I}$.*

Proof. Let $f - gh = e \bmod \mathcal{I}^2$. Verify that $g' = g + te \bmod \mathcal{I}^2$ and $h' = h + se \bmod \mathcal{I}^2$ satisfy the conditions $f = g'h' \bmod \mathcal{I}^2, g' = g \bmod \mathcal{I}$ and $h' = h \bmod \mathcal{I}$. We refer to these three conditions together by C.

For any g', h' satisfying C, let $d = sg' + th' - 1 \bmod \mathcal{I}^2$. Verify that $s' = (1 - d)s \bmod \mathcal{I}^2$ and $t' = (1 - d)t \bmod \mathcal{I}^2$ satisfy the conditions $s'g' + t'h' = 1 \bmod \mathcal{I}^2$, $s' = s \bmod \mathcal{I}$ and $t' = t \bmod \mathcal{I}$.

Suppose g^*, h^* be another solution satisfying C. Let $v = g^* - g'$ and $w = h^* - h'$. The relation $g^*h^* = g'h' \bmod \mathcal{I}^2$ implies that $g'w + h'v = 0 \bmod \mathcal{I}^2$, as $v, w \in \mathcal{I}$. Since $s'g' + t'h' = 1 \bmod \mathcal{I}^2$, multiplying both sides by v we get $(s'v - t'w)g' = v \bmod \mathcal{I}^2$. By taking $u = s'v - t'w \in \mathcal{I}$, $g^* = (1 + u)g' \bmod \mathcal{I}^2$. Similarly, $h^* = (1 - u)h' \bmod \mathcal{I}^2$. □

When applied to the ring of polynomials, Hensel lifting always generates a 'unique factorization' modulo an ideal. The following lemma clarifies this point.

Lemma 2.3.4. *Let $f, g, h \in \mathcal{R}[x]$ be monic polynomials and $\mathcal{I}$ be an ideal of $\mathcal{R}[x]$ generated by some set $\mathcal{S} \subseteq \mathcal{R}$. If $f = gh \bmod \mathcal{I}$ and $sg + th = 1 \bmod \mathcal{I}$ for some $s, t \in \mathcal{R}[x]$ then,*

1. *there exist monic $g', h' \in \mathcal{R}[x]$ such that $f = g'h' \bmod \mathcal{I}^2$, $g' = g \bmod \mathcal{I}$ and $h' = h \bmod \mathcal{I}$.*

2. *if g^* is any other monic polynomial with $f = g^*h^* \bmod \mathcal{I}^2$, for some $h^* \in \mathcal{R}[x]$, and $g^* = g \bmod \mathcal{I}$ then $g^* = g' \bmod \mathcal{I}^2$ and $h^* = h' \bmod \mathcal{I}^2$.*

Proof. Applying Hensel's lemma, we can get $\tilde{g}, \tilde{h} \in \mathcal{R}[x]$ such that $f = \tilde{g}\tilde{h} \bmod \mathcal{I}^2$, $\tilde{g} = g \bmod \mathcal{I}$ and $\tilde{h} = h \bmod \mathcal{I}$. But $\tilde{g}$ and $\tilde{h}$ need not be monic. Let $v = \tilde{g} - g \in \mathcal{I}$. Since g is monic, we can divide v by g to obtain $q, r \in \mathcal{R}[x]$ such that $v = qg + r$ and $\deg_x(r) < \deg_x(g)$. Note that $q, r \in \mathcal{I}$. Define $g' = g + r$ and $h' = \tilde{h} + qh$ and verify that $f = g'h' \bmod \mathcal{I}^2$. Since $\deg_x(r) < \deg_x(g)$, g' is monic which also implies that h' is monic as f is monic.

Let g^* be any other monic polynomial with $f = g^*h^* \bmod \mathcal{I}^2$, for some h^* and $g^* = g \bmod \mathcal{I}$. This means, $gh = g^*h^* \bmod \mathcal{I}$ implying that $g(h - h^*) = 0 \bmod \mathcal{I}$. Since g is monic, $h^* = h \bmod \mathcal{I}$. Therefore, by Hensel's lemma, $g^* = (1+u)g' \bmod \mathcal{I}^2$ for some $u \in \mathcal{I}$. Since g^* is monic this can only mean $g^* = g' \bmod \mathcal{I}^2$ and also $h^* = h' \bmod \mathcal{I}^2$. □

Let us now use Lemma 2.3.3 and 2.3.4 to show how a root of unity can be *lifted* to the ring $\mathbb{Z}/p^s\mathbb{Z}$, starting from a root in $\mathbb{Z}/p\mathbb{Z}$.

Lemma 2.3.5. *Let ζ_s be a primitive $(p-1)$-th root of unity in $\mathbb{Z}/p^s\mathbb{Z}$. Then there exists a unique primitive $(p-1)$-th root of unity ζ_{2s} in $\mathbb{Z}/p^{2s}\mathbb{Z}$ such that $\zeta_{2s} \equiv \zeta_s \pmod{p^s}$. This unique root is given by $\zeta_{2s} = \zeta_s - \frac{f(\zeta_s)}{f'(\zeta_s)} \pmod{p^{2s}}$ where $f(x) = x^{p-1} - 1$.*

Proof. In the above two lemmas, take the ring $\mathcal{R} = \mathbb{Z}$, $f = x^{p-1} - 1$ and $\mathcal{I} = (p^s)$. Since ζ_s is a root of $f(x)$ modulo p^s, $f = (x - \zeta_s)h \pmod{\mathcal{I}}$ for a certain polynomial h. Applying Lemma 2.3.4, there are unique monic polynomials g' and h' such that $g' = (x - \zeta_s) \pmod{p^s}$ and $f = g'h' \pmod{p^{2s}}$. Hence, $g' = (x - \zeta_{2s})$ for a certain ζ_{2s} with $\zeta_{2s} = \zeta_s \pmod{p^s}$ and ζ_{2s} is a primitive $(p-1)^{th}$ root of unity in $\frac{\mathbb{Z}}{p^{2s}\mathbb{Z}}$. We need to show that $\zeta_{2s} = \zeta_s - \frac{f(\zeta_s)}{f'(\zeta_s)} \pmod{p^{2s}}$.

To see this, let us revisit the proofs of Lemma 2.3.3 and 2.3.4. Going by the same notation as in Lemma 2.3.4, $\tilde{g} = g + te \pmod{\mathcal{I}^2}$, where $s(x-\zeta_s)+th = 1 \pmod{p^s}$ and $e = f - gh \pmod{p^{2s}}$. Let a and b be the quotient and remainder, respectively, when h is divided by $(x - \zeta_s)$. Then $b = h(\zeta_s) = f'(\zeta_s) \pmod{p^s}$ and hence in the above relation s can be taken as $-a$ and t as $\frac{1}{f'(\zeta_s)}$ (note that, $f'(\zeta_s) \neq 0 \pmod{p}$). This gives, $\tilde{g} = g + \frac{e}{f'(\zeta_s)} \pmod{\mathcal{I}^2}$. Now notice that, in the proof of Lemma 2.3.4 by taking $v = \tilde{g} - g$, we get $r = \frac{e(\zeta_s)}{f'(\zeta_s)} = \frac{f(\zeta_s)}{f'(\zeta_s)} \pmod{p^{2s}}$. Therefore, $g' = g + r$ implies that $\zeta_{2s} = \zeta_s - \frac{f(\zeta_s)}{f'(\zeta_s)} \pmod{p^{2s}}$. □

Thus, starting from a primitive root in $\mathbb{Z}/p\mathbb{Z}$ one can apply Lemma 2.3.5 repeatedly to find a primitive root in $\mathbb{Z}/p^s\mathbb{Z}$, for any s.

2.3.3 Discrete Fourier Transform

We use this tool extensively in Chapter 4 to design the integer multiplication algorithm. Suppose $f : [0, n-1] \to \mathcal{R}$ be a function 'selecting' n elements of the ring $\mathcal{R}$. And let ω be a primitive n^{th} root of unity in $\mathcal{R}$. Then the Discrete Fourier Transform (DFT) of f is defined to be the map,

$$\begin{aligned} \mathcal{F}_f : [0, n-1] &\to \mathcal{R} \quad \text{given by} \\ \mathcal{F}_f(j) &= \sum_{i=0}^{n-1} f(i)\omega^{ij}. \end{aligned}$$

Computing the DFT of f is the task of finding the vector $(\mathcal{F}_f(0), \ldots, \mathcal{F}_f(n-1))$. This task can be performed efficiently using an algorithm called the Fast Fourier Transform (FFT), which was first found by Gauss in 1805 and later (re)discovered by Cooley and Tukey (CT65). The algorithm (given below) uses a divide and conquer strategy to compute the DFT of a function f, with domain size n, using $O(n \log n)$ ring operations. For simplicity, assume that n is a power of 2.

Correctness of the algorithm - This is immediate from the following two observations,

$$\begin{aligned} \mathcal{F}_f(2j) &= \sum_{i=0}^{n-1} f(i)\omega^{2ij} = \sum_{i=0}^{\frac{n}{2}-1} (f(i) + f(n/2+i)) \cdot (\omega^2)^{ij} \quad \text{and} \\ \mathcal{F}_f(2j+1) &= \sum_{i=0}^{n-1} f(i)\omega^{i(2j+1)} = \sum_{i=0}^{\frac{n}{2}-1} (f(i) - f(n/2+i))\omega^i \cdot (\omega^2)^{ij} \end{aligned}$$

Algorithm 1 : Fast Fourier Transform

1. If $n = 1$ return $f(0)$.
2. Define $f_0 : [0, \frac{n}{2} - 1] \to \mathcal{R}$ as $f_0(i) = f(i) + f(\frac{n}{2} + i)$.
3. Define $f_1 : [0, \frac{n}{2} - 1] \to \mathcal{R}$ as $f_1(i) = (f(i) - f(\frac{n}{2} + i))\omega^i$.
4. Recursively, compute DFT of f_0 with ω^2 as the root of unity.
5. Recursively, compute DFT of f_1 with ω^2 as the root of unity.
6. Return $\mathcal{F}_f(2j) = \mathcal{F}_{f_0}(j)$ and $\mathcal{F}_f(2j+1) = \mathcal{F}_{f_1}(j)$ for all $0 \leq j < \frac{n}{2}$.

Thus, the problem of computing the DFT of f reduces to computing the DFT of two functions f_0 and f_1 (as defined in the algorithm) with $n/2$ as the domain size.

Time complexity - Computing f_0 takes $n/2$ additions in $\mathcal{R}$, while computing f_1 takes $n/2$ additions in $\mathcal{R}$ and $n/2$ multiplications by powers of ω. Each of Step 4 and 5 computes the DFT of a function with $n/2$ as the domain size. By solving the recurrence, we get the following lemma.

Lemma 2.3.6. (DFT complexity) *Algorithm* 1 *computes the DFT of* f *using* $n \log n$ *additions in* $\mathcal{R}$ *and* $\frac{n}{2} \log n$ *multiplications by powers of* ω.

The reason the addition and the multiplication complexities are stated separately and explicitly, instead of just saying $O(n \log n)$, will be clear in Chapter 4.

Application: Polynomial multiplication

Suppose $f, g \in \mathcal{R}[x]$ be two polynomials of degree less than $n/2$. We will assume that $\mathcal{R}$ contains a primitive n^{th} root of unity ω and the element $n \cdot 1 = 1 + 1 + \ldots$ n times, is not zero in $\mathcal{R}$. Since ω is a primitive root, it satisfies the property $\sum_{j=0}^{n-1} \omega^{ij} = 0$ for every $1 \leq i \leq n-1$.

Let $f = \sum_{i=0}^{n-1} c_i x^i$ where $c_{n/2}, \ldots, c_{n-1}$ are all zeroes, as $\deg(f) < n/2$. Associate a function $\hat{f} : [0, n-1] \to \mathcal{R}$ with f given by $\hat{f}(i) = c_i$. Define the DFT of f to be the DFT of $\hat{f}$ i.e. $\mathcal{F}_f(j) \triangleq \mathcal{F}_{\hat{f}}(j) = \sum_{i=0}^{n-1} c_i \omega^{ij} = f(\omega^j)$, for all $0 \leq j \leq n-1$. Similarly define the DFT of g as $\mathcal{F}_g(j) = g(\omega^j)$, for all j. The product polynomial $h = fg$ has degree less than n and hence we can also define the DFT of h as $\mathcal{F}_h(j) = h(\omega^j)$ for

all j. Let $h = \sum_{i=0}^{n-1} r_i x^i$ and $\mathsf{D}(\omega)$ be the following matrix.

$$\mathsf{D}(\omega) = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix}$$

Define two vectors $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ and $\mathbf{h} = (h(1), h(\omega), \dots, h(\omega^{n-1}))$. Then, $\mathbf{r} \cdot \mathsf{D}(\omega) = \mathbf{h}$, implying that $n \cdot \mathbf{r} = \mathbf{h} \cdot \mathsf{D}(\omega^{-1})$. This is because $\mathsf{D}(\omega) \cdot \mathsf{D}(\omega^{-1}) = n \cdot I$, which follows from the property $\sum_{j=0}^{n-1} \omega^{ij} = 0$ for every $1 \leq i \leq n-1$. Here I is the $n \times n$ identity matrix. Now observe that, computing the expression $\mathbf{h} \cdot \mathsf{D}(\omega^{-1})$ is equivalent to computing the DFT of the polynomial $\tilde{h} = \sum_{i=0}^{n-1} h(\omega^i) x^i$ using ω^{-1} as the primitive n^{th} root of unity. We call this DFT of $\tilde{h}$ the *inverse*-DFT of h. This observation suggests the following polynomial multiplication algorithm.

Algorithm 2 : Polynomial multiplication using FFT

1. Compute the DFT of f to find the vector $(f(1), f(\omega), \dots, f(\omega^{n-1}))$.
2. Compute the DFT of g to find the vector $(g(1), g(\omega), \dots, g(\omega^{n-1}))$.
3. Multiply the above two vectors component-wise.
4. Obtain $(h(1), h(\omega), \dots, h(\omega^{n-1}))$.
5. Compute the inverse-DFT of h to get the vector $n \cdot \mathbf{r}$.
6. Divide $n \cdot \mathbf{r}$ by n to get $\mathbf{r} = (r_0, \dots, r_{n-1})$.
7. Return $h = \sum_{i=0}^{n-1} r_i x^i$.

Time complexity - In Steps 1, 2 and 5 the algorithm computes three DFTs, each with domain size n. The component-wise multiplication in Step 3 require n multiplications in $\mathcal{R}$. This will be elaborated upon in Chapter 4.

2.3.4 Structure of Commutative Algebras

Our next tool is a structure theorem from the theory of commutative algebra. We use this result in Chapter 5 to give a deterministic identity testing algorithm for depth-2 circuits over commutative algebras of small dimension. The structure theorem states how a finite dimensional commutative algebra decomposes into local sub-algebras.

Theorem 2.3.7. (Structure theorem) *A finite dimensional commutative algebra $\mathcal{R}$ over $\mathbb{F}$ is isomorphic to a direct product of local rings i.e.*

$$\mathcal{R} \cong \mathcal{R}_1 \oplus \ldots \oplus \mathcal{R}_\ell$$

where each $\mathcal{R}_i$ is a local ring contained in $\mathcal{R}$ and any non-unit in $\mathcal{R}_i$ is nilpotent.

Our algorithm in Chapter 5 requires an effective and efficient version of this theorem. Therefore, for the sake of convenience in presentation, we defer its proof to Section 5.4.1.

2.4 Randomized vs. Deterministic Algorithms

We end this chapter with a brief discussion of our motive in studying the deterministic complexity of problems as opposed to their randomized complexity. Randomized algorithms are often conceptually simpler and more efficient in practice compared to their deterministic counterparts. A randomized algorithm takes as input a random string in addition to an input string and does some computation to decide the output. Randomized polynomial time computation is formally defined by the complexity class BPP. A language L is in BPP if there is an algorithm which on input (x, r), where $x, r \in \{0,1\}^*$ and $|r| = \mathsf{poly}(|x|)$, runs in polynomial time and correctly decides the membership of x in L with high probability over the random choice of r.

As to whether truly random strings can be generated in practice remains a debatable issue. It is also not clear if randomness is absolutely necessary when restricted to polynomial time computation. In fact, study of *pseudo-random generators* has given rise to the general conjectural belief that $\mathsf{BPP} = \mathsf{P}$ (NW94). One way to derandomize all randomized poly-time algorithms is to show efficient construction of strong pseudo-random generators. But with our present state of knowledge this appears to be a difficult task. So, to support this conjecture it seems justified to look for derandomization of specific problems. However, finding deterministic solutions to specific problems has another important justification. It follows from the work of Impagliazzo and Kabanets (KI03) and Agrawal (Agr05) that certain efficient derandomization of polynomial identity testing implies that the Permanent polynomial

requires super-polynomial sized arithmetic circuit, which if true will settle the conjecture that $\mathsf{VP} \neq \mathsf{VNP}$ (the arithmetic analogue of the famous $\mathsf{P} \neq \mathsf{NP}$ problem). In addition to these profound complexity theoretic implications, derandomization of a problem often provide us with deeper and revealing mathematical insights. These are our primary motivations in studying the deterministic complexity of the problems in this thesis.

Chapter 3

Polynomial Factoring over Finite Fields

3.1 Introduction

The problem of computing the irreducible factors of a given polynomial is one of the most fundamental and well-studied problem in algebraic computation. In this chapter, we study the deterministic complexity of factoring a univariate polynomial with coefficients taken from a finite field. This problem reduces in polynomial time to factoring a monic, square-free and completely splitting polynomial f with coefficients in a prime field $\mathbb{F}_p$ (see (Ber70), (LN94), and Section 3.1.2). Therefore, the factoring problem can be stated simply as follows: given a polynomial $f(x) \in \mathbb{F}_p[x]$ in terms of all its n coefficients, where f splits as

$$f(x) = \prod_{i=1}^{n} (x - \xi_i), \quad \xi_1, \ldots, \xi_n \in \mathbb{F}_p \text{ are distinct roots,}$$

find $\xi_1, \ldots, \xi_n$. Although there are efficient polynomial time randomized algorithms for factoring f (see Section 3.1.1), as yet there is no deterministic polynomial time (i.e. $\mathsf{poly}(n, \log p)$ time) algorithm even under the assumption of the Extended Riemann Hypothesis (ERH). This is in contrast to the state of the affair for its sister problem - factoring polynomials over rationals. Indeed, the classical LLL algorithm, given by Lenstra, Lenstra and Lovász (LJL82), factors a polynomial $f \in \mathbb{Z}[x]$ with

coefficients f_i, $0 \leq i \leq n$, in time $\mathsf{poly}(n, \log A)$, where $A = \max_{0 \leq i \leq n} \mid f_i \mid$.

We begin this chapter by giving a brief account of the known results on polynomial factoring over finite fields. Assume throughout this chapter that n is the degree of the input polynomial f.

3.1.1 Previous Work

Randomized algorithms - The first randomized factoring algorithm dates back to the work of Berlekamp (Ber70). Several improvements in the running time came up subsequently due to Cantor and Zassenhaus (CZ81), von zur Gathen and Shoup (vzGS92), Kaltofen and Shoup (KS98), and others (refer to the survey by von zur Gathen and Panario (vzGP01)). The current best randomized algorithm was given recently by Kedlaya and Umans (KU08). Using a fast modular composition algorithm along with ideas from Kaltofen and Shoup (KS98), they achieved a running time of $\tilde{O}(n^{1.5} + n \log q)$ field operations, where $\mathbb{F}_q$ is the underlying finite field. Note that, this time complexity is indeed polynomial in the input size (which is about $n \log q$ bits), since a field operation takes $(\log q)^{O(1)}$ bit operations.

A substantial amount of work has been directed towards achieving efficient deterministic factoring algorithms. Such algorithms can be broadly classified into two categories - ERH-free results and results assuming the validity of the ERH.

ERH-free results - Without the assumption of the ERH, it is not even known how to find square root of an element $a \in \mathbb{F}_p$ efficiently. However, Schoof (Sch85) showed that if a is fixed then square root can be computed in time polynomial in $\log p$. Another result (without assuming ERH) by Shoup (Sho90) showed that the roots of f can be computed in $O(p^{\frac{1}{2}}(\log p)^2 n^{2+\epsilon})$ time. ERH-free results have been largely limited until recently when Ivanyos, Karpinski, Rónyai and Saxena (IKRS08) showed that either a nontrivial factor of f or a nontrivial automorphism of $\frac{\mathbb{F}_p[x]}{(f)}$ of order n can be found in deterministic $\mathsf{poly}(n^{\log n}, \log p)$ time. They also gave a deterministic polynomial time algorithm to find a nontrivial factor of the r^{th} cyclotomic

polynomial, where $r > 4$ is such that either $4|r$ or r has two distinct prime factors.

Results assuming the ERH - The results obtained under the assumption of the ERH can be broadly classified into two categories: 1) results that assume some special property of the field characteristic p and 2) results that assume some special property of the input polynomial f. Under the first category there are results by von zur Gathen (vzG87), Mignotte and Schnorr (MS88), Rónyai (Rón89) and Shoup (Sho91), which showed that f can be factored in $(n \log p)^{O(1)}$ time if $p-1$ is smooth. Another result by Bach, von zur Gathen and Lenstra (BvzGL95) showed the time complexity of factoring to be $(\ell n \log p)^{O(1)}$ where ℓ is the largest prime factor of $\Phi_k(p)$, the k^{th} cyclotomic polynomial evaluated at p.

Under the second category of results, Huang (Hua84) showed that the factors of the n^{th} cyclotomic polynomial and n^{th} roots of any $a \in \mathbb{F}_p$ can be found in $(n \log p)^{O(1)}$ time (see also (AMM77)). Rónyai (Rón88) showed that f can be factored in a nontrivial way in time polynomial in n^r and $\log p$, where r is a prime divisor of n. It follows immediately that every even degree polynomial can be factored efficiently. However, in the worst case the algorithm takes $(n^n \log p)^{O(1)}$ time. Later, Huang (Hua91), Rónyai (Rón92) and Evdokimov (Evd92) gave deterministic polynomial time algorithms for factoring special kinds of f using polynomial factorizations over rationals and algebraic number fields (see (LJL82) and (Lan85)). Huang (Hua91) showed that a poly-time factoring algorithm exists when the roots of f generate an Abelian extension over $\mathbb{Q}$. A more general result was given by Rónyai (Rón92), where f can be factored in time polynomial in n, the degree of the splitting field of f over $\mathbb{Q}$ and $\log p$. Evdokimov (Evd92) gave a deterministic poly-time algorithm when f is solvable over $\mathbb{Q}$.

In 1994, Evdokimov (Evd94) gave the first deterministic sub-exponential time algorithm that works in $(n^{\log n} \log p)^{O(1)}$ time. It is worth noting that Evdokimov's algorithm works for all finite fields and for all univariate polynomials. Cheng and Huang (CH00) also developed an algorithm, with $(n^{\log n} \log p)^{O(1)}$ time complexity, by exploiting interesting connections to the combinatorial problem of stable coloring of tournaments. They further showed that under a purely combinatorial conjecture, the algorithm runs in polynomial time. Ivanyos, Karpinski and Saxena (IKS09) related the problem of factoring to certain combinatorial objects called m-schemes

and showed that a nontrivial factor of a polynomial with prime degree n can be obtained in poly-time if $n-1$ is smooth. Gao (Gao01) gave a deterministic poly-time factoring algorithm that fails to find nontrivial factors if f belongs to a restricted class of polynomials, namely *square balanced polynomials*.

Our work is mainly motivated by the work of Gao (Gao01) and Evdokimov (Evd94). Although it shares some common concepts with (CH00) and (IKS09), the work presented here appears to be incomparable to them. Before we move on to the main content of this chapter let us briefly sketch how factoring gets reduced to root finding.

3.1.2 Berlekamp's Reduction to Root Finding

We use the Chinese remaindering theorem (Theorem 2.3.1) to show that polynomial factoring reduces to the problem of root finding over finite fields.

Square free factoring - Let $f \in \mathbb{F}_q[x]$ be a polynomial of degree n that factors as $f = f_1 \dots f_k$, where f_i's are irreducible polynomials over $\mathbb{F}_q$. We can assume that f is square-free or else we can take gcd of f and $\frac{df}{dx}$, the formal derivative of f with respect to x, and find nontrivial factors of f. The process can be continued until we are left with only square-free polynomials to factor. It is easy to see that this step, also known as square-free factorization, takes polynomial in n field operations.

Distinct degree factoring - Suppose that the irreducible factors of f are not of the same degree. Then, there are two factors, say, f_1 and f_2 such that $\deg(f_1) = d_1$ is minimum and $\deg(f_2) = d_2 > d_1$. Since f_1 and f_2 are irreducible, f_1 divides the polynomial $x^{q^{d_1}} - x$ whereas f_2 does not. Therefore, $\gcd(x^{q^{d_1}} - x, f)$ yields a non-trivial factor of f. As d_1 is unknown, we iteratively compute $x^{q^t} - x$ starting from $t = 1$ till we hit $t = d_1$. This can be done by using the repeated squaring method to compute $x^{q^t} \bmod f$. This step, known as distinct-degree factorization, takes $(n \log q)^{O(1)}$ field operations.

Equal degree factoring - We are now left with the task of factoring a square-free polynomial $f = f_1 \dots f_k$ such that all the irreducible factors f_i's have the

same degree, say, d. This step is called equal-degree factorization. By Chinese remaindering theorem,

$$\mathcal{R} = \frac{\mathbb{F}_q[x]}{(f)} \cong \oplus_{i=1}^{k} \frac{\mathbb{F}_q[x]}{(f_i)} \cong \oplus_{i=1}^{k} \mathbb{F}_{q^d}.$$

Let $g \in \mathcal{R} \backslash \mathbb{F}_q$ be such that $g^q = g$ in $\mathcal{R}$. First, let us show that such a g exists in $\mathcal{R}$. By the above isomorphism, any g whose direct-sum representation belongs to $\oplus_{i=1}^{k} \mathbb{F}_q$ satisfies the condition $g^q = g \bmod f$. Also, if f is not irreducible then such a $g \notin \mathbb{F}_q$ exists. This means, there exists $c_i, c_j \in \mathbb{F}_q$ $(i \neq j)$ such that $c_i = g \bmod f_i$, $c_j = g \bmod f_j$ and $c_i \neq c_j$. This also implies that there is a $c \in \mathbb{F}_q$ such that the $\gcd(g - c, f)$ yields a non-trivial factor of f. For instance, for $c = c_i$, f_i divides the $\gcd(g - c, f)$ but f_j does not.

To compute g, start with a generic element $\sum_{i=0}^{n-1} a_i x^i \in \mathcal{R}$, where $n = \deg(f)$ and a_i's are variables, and solve for $a_i \in \mathbb{F}_q$ such that $\sum_{i=0}^{n-1} a_i x^{qi} = \sum_{i=0}^{n-1} a_i x^i \bmod f$. Solving this equation reduces to solving a system of linear equations in the a_i's. This reduction follows once we compute x^{qi} rem f for all i and equate the coefficients of x^j, for $0 \leq j \leq n-1$, from both sides of the equation. Now all we need to do, while solving the linear equations, is to choose a solution for the a_i's such that $\sum_{i=0}^{n-1} a_i x^i \notin \mathbb{F}_q$. Take $g = \sum_{i=0}^{n-1} a_i x^i$ for that choice of a_i's. Taking into account that x^{qi} rem f can be computed using repeated squaring, we conclude that g can be found in polynomial time.

Reduction to root finding - The only task that remains is to find a $c \in \mathbb{F}_q$ such that the $\gcd(g - c, f)$ gives a nontrivial factor of f. This is where the problem gets reduced to root finding. The fact that the $\gcd(g - c, f) \neq 1$ means resultant of the polynomials $g - c = \sum_{i=1}^{n-1} a_i x^i + (a_0 - c)$ and f is zero. This means, we need to solve for a $y \in \mathbb{F}_q$ such that $h(y) = \mathsf{Res}_x(\sum_{i=1}^{n-1} a_i x^i + (a_0 - y), f) = 0$, treating $a_0 - y$ as the constant term of the polynomial $g - y$. We can find h by computing the determinant of the Sylvester matrix, $S(\sum_{i=1}^{n-1} a_i x^i + (a_0 - y), f)$, of $g - y$ and f. Although there are entries in S containing variable y, we can find $\det(S)$ in polynomial time using interpolation. In this way, factoring polynomial $f(x)$ reduces to finding a root of the polynomial $h(y)$. Finally, root finding over $\mathbb{F}_q$ reduces to root finding over $\mathbb{F}_p$, where $p = \mathrm{char}(\mathbb{F})$. Details of this last step can be found in Chapter 4 of (LN94).

3.2 Our Approach

An Overview - Our approach to factoring can be summarized as follows. Let f be a monic, square-free and completely splitting polynomial in $\mathbb{F}_p[x]$ with n roots $\xi_1, \ldots, \xi_n$. Our factoring algorithm uses an arbitrary (but deterministically chosen) collection of $k = (n \log p)^{O(1)}$ small degree auxiliary polynomials $p_1(.), \ldots, p_k(.)$, and from each $p_\ell(\cdot)$ $(1 \leq \ell \leq k)$ and f it implicitly constructs a simple n-vertex digraph G_ℓ such that, (for $\ell > 1$) G_ℓ is a subgraph (but not necessarily a proper subgraph) of $G_{\ell-1}$. A proper factor of f is efficiently retrieved if any one of the graphs is either not regular, or is regular with in degree and out degree of every vertex less than a chosen constant c. This condition of regularity of all the k graphs imposes a tight symmetry condition on the roots of f, and we point out that this may be exploited to improve the worst case time complexity of the best known deterministic algorithms. Further, we show that if the polynomials $p_\ell(\cdot)$ $(1 \leq \ell \leq k)$ are randomly chosen then the symmetry breaks with high probability and our algorithm works in randomized polynomial time. We call the checking of this symmetry condition a *balance test*.

We now present a little more details.

Square Balance property - Gao (Gao01) designed a polynomial time algorithm that fails to factor only if the input polynomial satisfies a strong symmetry property, namely square balance. Square balance property is defined as follows. Define the sets Δ_i for $1 \leq i \leq n$ as,

$$\Delta_i = \{1 \leq j \leq n : j \neq i, \sigma((\xi_i - \xi_j)^2) = -(\xi_i - \xi_j)\},$$

where σ is the square root algorithm described in (Gao01) (see Section 3.3.6). The polynomial f is called a square balanced polynomial if $\#\Delta_1 = \ldots = \#\Delta_n$. (The square root algorithm σ has the property that for $p = 3 \bmod 4$, $\sigma(a^2) = a$ if and only if a is a quadratic residue.)

Example 3.2.1. (Square Balance) *Let $p = 19 = 3$ mod 4 and $f = (x-2)(x-6)(x-9)(x-12)(x-15)$ be the input polynomial. Then verify using the above definition that f is square balanced. (See also Example 3.2.4.)*

In Section 3.5, we show that for $p = 3 \bmod 4$ about $n^{-n/2}$ fraction of all polynomials of degree n are square balanced when p is sufficiently large.

3.2.1 The Main Theorem

We propose an extension of Gao's algorithm that fails only under an even stronger symmetry property that we call *cross balance*. Cross balance property is defined as follows. For $\ell > 1$, define the polynomial f_ℓ as,

$$f_\ell = \prod_{i=1}^{n} (x - p_\ell(\xi_i)),$$

where $p_\ell(.)$ is an arbitrary but deterministically chosen polynomial with degree bounded by $(n \log p)^{O(1)}$. Further, $p_{\ell_1}(.) \neq p_{\ell_2}(.)$ for $\ell_1 \neq \ell_2$, and f_1 is taken to be f i.e. $p_1(y) = y$. Assume that, for a given $k = (n \log p)^{O(1)}$, for every ℓ, $1 \leq \ell \leq k$, the polynomial $f_\ell = \tilde{f}_\ell^{d_\ell}$, where $\tilde{f}_\ell$ is a square-free and square balanced polynomial and $d_\ell > 0$. Later, we show that, if f_ℓ is not of the above form then a proper factor of f can be retrieved efficiently. For each polynomial f_ℓ, $1 \leq \ell \leq k$, define the sets $\Delta_i^{(\ell)}$ for $1 \leq i \leq n$ as,

$$\Delta_i^{(\ell)} = \{1 \leq j \leq n : p_\ell(\xi_i) \neq p_\ell(\xi_j), \sigma((p_\ell(\xi_i) - p_\ell(\xi_j))^2) = -(p_\ell(\xi_i) - p_\ell(\xi_j))\}$$

Further, define the sets ${D_i}^{(\ell)}$ iteratively over ℓ as,

$$\begin{aligned} D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } \ell > 1,\ D_i^{(\ell)} &= D_i^{(\ell-1)} \cap \Delta_i^{(\ell)} \\ \text{If } D_i^{(\ell)} &= \phi \text{ for all } i,\ 1 \leq i \leq n, \text{ then redefine } D_i^{(\ell)} \text{ as } D_i^{(\ell)} = D_i^{(\ell-1)}. \end{aligned}$$

For $1 \leq \ell \leq k$, let G_ℓ be a directed graph with n vertices $v_1, \ldots, v_n$, such that there is an edge from v_i to v_j if and only if $j \in D_i^{(\ell)}$. Note that, G_ℓ is a subgraph of $G_{\ell-1}$ for $1 < \ell \leq k$. Denote the in degree and out degree of a vertex v_i by $\text{indeg}(v_i)$ and $\text{outdeg}(v_i)$, respectively. We say that the graph G_ℓ is regular (or t-regular) if $\text{indeg}(v_1) = \text{outdeg}(v_1) = \ldots = \text{indeg}(v_n) = \text{outdeg}(v_n) = t$. Call t the *regularity* of G_ℓ.

Definition 3.2.2. (Cross Balance) *A polynomial f is called k-cross balanced, for $k > 0$, if for every ℓ, $1 \leq \ell \leq k$, the polynomial $f_\ell = \tilde{f}_\ell^{d_\ell}$, where $\tilde{f}_\ell$ is a square-free, square balanced polynomial with $d_\ell > 0$, and the graph G_ℓ is regular.*

The following theorem is proved in this chapter.

Theorem 3.2.3. (Main Theorem) *Polynomial f can be factored into nontrivial factors in time $k \cdot (n \log p)^{O(1)}$ if f is not k-cross balanced. Further, if $G_1, \ldots, G_k$ are all regular then we require at most $\log_2 n$ of the graphs G_ℓ to be such that $G_\ell \neq G_{\ell-1}$ $(1 < \ell \leq k)$, so that f can be factored in $k \cdot (n \log p)^{O(1)}$ time.*

3.2.2 The Motivation

Let us motivate the usefulness of generalizing the square balance property to cross balance. Suppose $f(y)$ splits as $f(y) = (y - X) \cdot \hat{f}(y)$ in the quotient ring $\mathcal{R} = \frac{\mathbb{F}_p[x]}{(f)}$ where $X = x \bmod f$. Our algorithm iteratively tests the graphs $G_1, G_2, \ldots$ so on, to check if any one of them is not regular. (Note that, G_1 is regular if and only if f is square balanced.) If at the ℓ^{th} iteration the graph G_ℓ turns out to be not regular, then a proper factor of f is obtained in polynomial time. However, if G_ℓ is regular, then the algorithm returns a nontrivial monic factor $g_\ell(y)$ of $\hat{f}(y)$ with degree equal to the regularity of G_ℓ. Moreover, $g_\ell(y)$ is also a factor of (although may be equal to) $g_{\ell-1}(y)$, the factor obtained at the $(\ell-1)^{th}$ iteration, and it can be ensured that if $g_\ell(y)$ is a proper factor of $g_{\ell-1}(y)$ (which happens iff $G_\ell \neq G_{\ell-1}$) then $\deg(g_\ell(y)) \leq \frac{1}{2} \cdot \deg(g_{\ell-1}(y))$. Thus, if the graphs repeatedly turn out to be regular (which in itself is a stringent condition) and about $\log_2 n$ times it happen that $G_\ell \neq G_{\ell-1}$, for $1 < \ell \leq k$, then we obtain a nontrivial linear factor $g(y)$ of $\hat{f}(y)$. The element $-g(0)$ defines a nontrivial endomorphism in the ring $\mathcal{R}$, and by using a result from (Evd94) (see Section 3.3.5) we can find a proper factor of f in polynomial time. Further, if for only $\epsilon\lceil \log_2 n \rceil$ times we get $G_\ell \neq G_{\ell-1}$ $(1 < \ell \leq k)$ for some ϵ, $0 < \epsilon \leq 1$, then we obtain a nontrivial factor $g(y)$ of $\hat{f}(y)$ with degree at most $\frac{n^{1-\epsilon}}{2}$. Now if we apply Evdokimov's algorithm on $g(y)$ (instead of $\hat{f}(y)$), we can get a proper factor of f in time $(n^{\frac{(1-\epsilon)^2}{2}\log n + \epsilon + c_1} \log p)^{c_2}$ (c_1 and c_2 are constants). For most polynomials $\epsilon > 0$ (i.e. at least about $\frac{1}{\log n}$) and this gives an improvement over the time complexity of $(n^{\frac{1}{2}\log n + c_1} \log p)^{c_2}$ in (Evd94) (c_1, c_2 are the same constants).

Assuming $n << p$, all the best known deterministic algorithms (e.g. (Evd94), (CH00)) use computations in rings with large dimensions over $\mathbb{F}_p$ to get smaller degree factors of $\hat{f}(y)$. Unlike these approaches, the balance test is an attempt to exploit an asymmetry among the roots of the input polynomial to obtain smaller

degree factors of $\hat{f}(y)$ without having to carry out computations in rings with large dimensions over $\mathbb{F}_p$. This attribute of our approach yields a better time complexity for most polynomials in a way as discussed in the previous paragraph.

We now demonstrate the usefulness of the cross balance property with a simple example. Let H_ℓ $(1 \leq \ell \leq k)$ be a digraph with n vertices $v_1, \ldots, v_n$ such that there is an edge from v_i to v_j iff $j \in \Delta_i^{(\ell)}$. Then, graph $G_\ell = G_{\ell-1} \cap H_\ell$ or $G_\ell = G_{\ell-1}$ (if $G_{\ell-1} \cap H_\ell = \Phi$, where Φ is the null graph with n vertices but no edge). Here the symbol $\cap$ denotes the edge intersection of graphs defined on the same set of vertices. Suppose we choose the auxiliary polynomials $p_\ell(y) = y^\ell$ for $1 \leq \ell \leq k$. (In fact, Gao (Gao01) used this choice of auxiliary polynomials to define a restricted class of square balanced polynomials called super square balanced polynomials.)

Example 3.2.4. *Let $p = 19$ and $f = f_1 = (x-2)(x-6)(x-9)(x-12)(x-15)$ be the input polynomial. Then, $f_2 = (x-4)(x-17)(x-5)(x-11)(x-16)$. Since $p = 3 \bmod 4$, there is an edge from v_i to v_j in H_1 (resp. H_2) iff $\xi_i - \xi_j$ (resp. $\xi_i^2 - \xi_j^2$) is a quadratic nonresidue. The nonresidues in F_{19} are $\{2, 3, 8, 10, 12, 13, 14, 15, 18\}$. The graphs G_1, H_2 and G_2 are depicted in Figure 3.1. Note that, both f_1 and f_2 are square free and square balanced. But, since G_2 is irregular, f is not 2-cross balanced and hence by Theorem 3.2.3 f can be factored in polynomial time.*

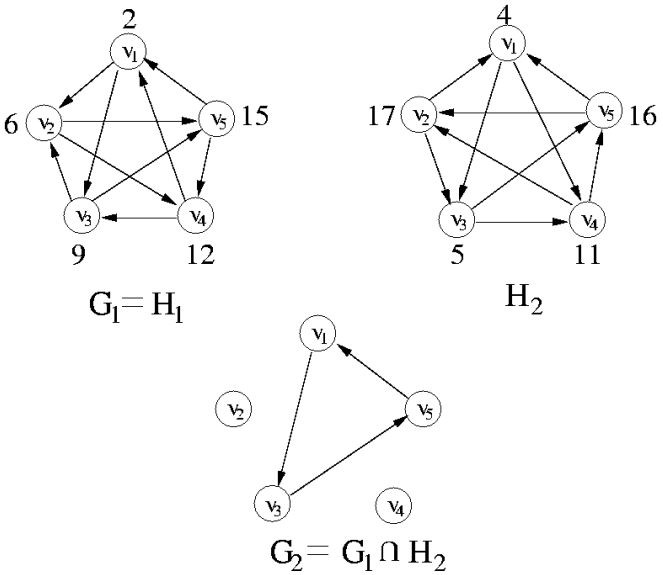

Figure 3.1: An example showing the usefulness of the cross balance property.

3.3 Background Concepts

In this section, we explain the background concepts and results required to prove the main theorem (Theorem 3.2.3). Assume throughout that f is a monic, square-free and completely splitting polynomial over $\mathbb{F}_p$, and $\mathcal{R} = \frac{\mathbb{F}_p[x]}{(f)}$ is the quotient ring consisting of all polynomials modulo f. Denote the element x mod f in $\mathcal{R}$ by X.

3.3.1 Primitive Idempotents

Elements $\chi_1, \ldots, \chi_n$ of the ring $\mathcal{R}$ are called the primitive idempotents of $\mathcal{R}$ if, $\sum_{i=1}^{n} \chi_i = 1$ and for $1 \leq i, j \leq n$, $\chi_i \cdot \chi_j = \chi_i$ if $i = j$ and 0 otherwise. By the Chinese Remaindering theorem, $\mathcal{R} \cong \mathbb{F}_p \oplus \ldots \oplus \mathbb{F}_p$ (n times), such that every element in $\mathcal{R}$ can be uniquely represented by an n-tuple of elements in $\mathbb{F}_p$. Addition and multiplication between two elements in $\mathcal{R}$ can be viewed as componentwise addition and multiplication of the n-tuples. Any element $\alpha = (a_1, \ldots, a_n) \in \mathcal{R}$ can be equated as, $\alpha = \sum_{i=1}^{n} a_i \chi_i$ where $a_i \in \mathbb{F}_p$. Let $g(y)$ be a polynomial in $\mathcal{R}[y]$ given by,

$$\begin{aligned} g(y) &= \sum_{i=0}^{m} \gamma_i y^i \quad \text{where } \gamma_i \in \mathcal{R} \text{ and} \\ \gamma_i &= \sum_{j=1}^{n} g_{ij} \chi_j \quad \text{where } g_{ij} \in \mathbb{F}_p \text{ for } 0 \leq i \leq m \text{ and } 1 \leq j \leq n. \end{aligned}$$

Then $g(y)$ can be alternatively expressed as,

$$g(y) = \sum_{j=1}^{n} g_j(y) \chi_j \text{ where } g_j(y) = \sum_{i=0}^{m} g_{ij} y^i \in \mathbb{F}_p[y] \text{ for } 1 \leq j \leq n.$$

The usefulness of this representation is that, operations on polynomials in $\mathcal{R}[y]$ (multiplication, gcd etc.) can be viewed as componentwise operations on polynomials in $\mathbb{F}_p[y]$.

3.3.2 Characteristic Polynomial

Consider an element $\alpha = \sum_{i=1}^{n} a_i \chi_i \in \mathcal{R}$ where $a_i \in \mathbb{F}_p$, $1 \leq i \leq n$. The element α defines a linear transformation on the vector space $\mathcal{R}$ (over $\mathbb{F}_p$), mapping an

element $\beta \in \mathcal{R}$ to $\alpha\beta \in \mathcal{R}$. The characteristic polynomial of α (viewed as a linear transformation) is independent of the choice of a basis in $\mathcal{R}$ and is equal to

$$c_\alpha(y) = \prod_{i=1}^{n} (y - a_i).$$

In order to find c_α, take $\{1, X, X^2, \ldots, X^{n-1}\}$ as the basis of $\mathcal{R}$ and form the matrix (m_{ij}) where $\alpha \cdot X^{j-1} = \sum_{i=1}^{n} m_{ij} X^{i-1}$, $m_{ij} \in \mathbb{F}_p$, $1 \leq i, j \leq n$. Then c_α is constructed by evaluating the $\det(y \cdot I - (m_{ij}))$ at n distinct values of y and solving the n coefficients of c_α using linear algebra. This process takes polynomial time. The notion of characteristic polynomial extends to higher dimensional algebras over $\mathbb{F}_p$.

3.3.3 GCD of Polynomials over Algebras

Let $g(y) = \sum_{i=1}^{n} g_i(y)\chi_i$ and $h(y) = \sum_{i=1}^{n} h_i(y)\chi_i$ be two polynomials in $\mathcal{R}[y]$, where $g_i, h_i \in \mathbb{F}_p[y]$ for $1 \leq i \leq n$. Then, the gcd of g and h is defined as,

$$\gcd(g, h) = \sum_{i=1}^{n} \gcd(g_i, h_i)\chi_i$$

We note that, the concept of gcd of polynomials does not make sense in general over any arbitrary algebra. However, the fact that $\mathcal{R}$ is a completely splitting, semisimple algebra over $\mathbb{F}_p$ allows us to work component-wise over $\mathbb{F}_p$ and this makes the notion of gcd meaningful in this context. (Note that, $\mathcal{R}$ is completely splitting as $\mathcal{R} \cong \mathbb{F}_p \oplus \ldots \oplus \mathbb{F}_p$ (n times), and semisimple as it has no nonzero nilpotent element.)

Lemma 3.3.1. *Given two polynomials $g, h \in \mathcal{R}[y]$, $\gcd(g, h)$ can be computed in time polynomial in the degrees of g and h, n and $\log p$.*

Proof. Apply the extended Euclidean algorithm to the polynomials g and h pretending that they are polynomials over $\mathbb{F}_p$. Either the algorithm goes through and finds a gcd or during the process a zero divisor $z \in \mathcal{R}$ is encountered. Let $v = z^{p-1}$ in $\mathcal{R}$, implying that $v^2 = v$ in $\mathcal{R}$. Using v split the algebra as $\mathcal{R} = \mathcal{R}v \oplus \mathcal{R}(1-v)$, where v and $(1-v)$ are the identity elements of the orthogonally complement subalgebras $\mathcal{R}v$ and $\mathcal{R}(1-v)$, respectively. Let $g_1 = gv; g_2 = g(1-v)$ and similarly $h_1 = hv; h_2 = h(1-v)$. Now notice that $\gcd(g, h) = \gcd(g_1, h_1) + \gcd(g_2, h_2)$ (assuming $\gcd(0, 0) = 0$). Compute the $\gcd(g_1, h_1)$ and $\gcd(g_2, h_2)$ by recursively applying the Euclidean algorithm

in the subalgebras $\mathcal{R}v$ and $\mathcal{R}(1-v)$ using v and $(1-v)$ as the identity elements, respectively. To carry out computations in the subalgebras, find explicit $\mathbb{F}_p$-bases of $\mathcal{R}v$ and $\mathcal{R}(1-v)$ from a basis of $\mathcal{R}$. Since $\dim_{\mathbb{F}_p} \mathcal{R} = \dim_{\mathbb{F}_p} \mathcal{R}v + \dim_{\mathbb{F}_p} \mathcal{R}(1-v) = n$, the recursion can finally give rise to at most n subalgebras. Therefore, the gcd algorithm runs in time $\mathsf{poly}(n, \log p, \deg(g,h))$. □

3.3.4 ERH: Ankeny-Bach Estimates and ℓ^{th} Root Finding

Many of the results mentioned in Section 3.1.1 use the assumption of the Extended Riemann Hypothesis. The ERH appears in these factoring algorithms mainly through the following result on the least nonresidue in a prime field. An element $a \in \mathbb{F}_p$ is called an ℓ^{th} nonresidue if there is no $b \in \mathbb{F}_p$ for which $b^\ell = a$.

Theorem 3.3.2. (Least nonresidue estimate) *Assuming the ERH, the value of the least ℓ^{th} nonresidue in $\mathbb{F}_p^\times$ is bounded by $c \log^2 p$, where ℓ is a prime dividing $p-1$ and c is an effectively computable constant independent of ℓ and p.*

Finding ℓ^{th} power roots - It is known from the work of Vinogradov (Vin72) (see also Proposition 7 in (Evd94)) that given an ℓ^{th} nonresidue, all the ℓ^{th} power roots of an element $a \in \mathbb{F}_p$ i.e. all x such that $x^\ell = a$, can be found in time $(\ell \log p)^{O(1)}$. In fact, it is this step of finding ℓ^{th} roots in $\mathbb{F}_p$ that appears in many factoring algorithms (including our algorithm), and it is here that the assumption of the ERH is used. The ℓ^{th} root finding algorithm also extends to completely splitting semisimple algebras over $\mathbb{F}_p$ (like the $\mathbb{F}_p$-algebra $\mathcal{R}$).

A restricted version of Theorem 3.3.2 was first shown by Ankeny (Ank52).
Ankeny's result - The least quadratic nonresidue in $\mathbb{F}_p$ has value $O(\log^2 p)$.
This result was later generalized by Bach (Bac82), who gave the following estimate.

Theorem 3.3.3. (Bach) *If $\mathcal{G}$ is a proper subgroup of $\mathbb{Z}_n^\times$ (where n is sufficiently large), then assuming the ERH there is a positive integer $m \in \mathbb{Z}_n^\times \backslash \mathcal{G}$ such that $m \leq c \log^2 n$, where c is a constant.*

Extension to general finite fields - Over general finite fields, Evdokimov (Evd92) gave a deterministic algorithm, under the assumption of the ERH, to construct an ℓ^{th} nonresidue in $\mathbb{F}_q$ in time $(\ell \log q)^{O(1)}$.

3.3.5 From Endomorphism to Factors

In order to see how an endomorphism of $\mathcal{R}$ can be useful in finding factors of f, let us examine its structure a little more closely. Let ϕ be an endomorphism of $\mathcal{R}$ that fixes $\mathbb{F}_p$, i.e. $\phi(a) = a$ for all $a \in \mathbb{F}_p$. Since any element in $\mathcal{R}$ can be expressed as a polynomial in X, the map ϕ is completely specified by its action on X. For any element $g \in \mathcal{R}$, use the Chinese remaindering isomorphism and write $g = (c_1, \ldots, c_n)$ to mean $g = c_i \bmod (x - \xi_i)$ for every $1 \leq i \leq n$. For example, in this direct sum notation the element $X \in \mathcal{R}$ is $(\xi_1, \ldots, \xi_n)$. Let $\phi(X) = (\alpha_1, \ldots, \alpha_n)$ for some α_i's in $\mathbb{F}_p$.

Since $\phi(f(X)) = f(\phi(X)) = 0$ in $\mathcal{R}$, each $\alpha_i = \xi_j$ for some j. Now suppose that there exist k and ℓ, $1 \leq k < \ell \leq n$, such that $\alpha_k = \alpha_\ell$. Then $g \triangleq$ characteristic polynomial of $\phi(X) = \prod_{i=1}^{n}(x - \alpha_i)$ is not square free and the $\gcd(f, g)$ yields a nontrivial factor of f. Clearly, $g \neq 0$ in $\mathcal{R}$. But, $\phi(g(X)) = g(\phi(X)) = 0$ in $\mathcal{R}$ which means kernel of ϕ is nontrivial and so ϕ is not an automorphism.

Not let ϕ be an automorphism of $\mathcal{R}$ fixing $\mathbb{F}_p$. Then, $\alpha_k \neq \alpha_\ell$ for every pair k and ℓ with $k \neq \ell$. In other words, ϕ induces a permutation ρ among the n coordinates of the direct sum representation, i.e. $\alpha_i = \xi_{\rho(i)}$ for all $1 \leq i \leq n$, where $\rho \in S_n$, the symmetric group of degree n. It is also easy to verify that any such permutation defines an automorphism, implying that there are a total of $n!$ automorphisms of $\mathcal{R}$ fixing $\mathbb{F}_p$.

Among these automorphisms suppose we could compute $n + 1$ distinct automorphisms $\phi_1, \ldots, \phi_{n+1}$. Here computing these automorphisms means finding the elements $\phi_s(X) \in \mathcal{R}$, for $1 \leq s \leq n + 1$. By the Pigeonhole principle, there is a pair of automorphisms ϕ_s and ϕ_t with $s \neq t$, such that the first coordinates of the direct sum representations of $\phi_s(X)$ and $\phi_t(X)$ are the same. Since ϕ_s and ϕ_t are distinct, $\phi_s(X) - \phi_t(X) \neq 0$ in $\mathcal{R}$, and hence the $\gcd(\phi_s(x) - \phi_t(x), f)$ gives a nontrivial factor of f.

Evdokimov's result - Evdokimov showed that assuming the ERH, finding even *one* nontrivial automorphism of $\mathcal{R}$ (instead of n of them) is sufficient to factor f. Since any automorphism $\phi(X)$ is a root of the polynomial $f(y) \in \mathcal{R}[y]$ (as $f(\phi(X)) = 0$ in

$\mathcal{R}$) and any root $\alpha(X)$ of $f(y)$ defines an endomorphism of $\mathcal{R}$ (mapping X to $\alpha(X)$), we have the following theorem (see Lemma 9 in (Evd94)).

Theorem 3.3.4. (Evd94) *A root of the polynomial $f(y)$ in $\mathcal{R}$ other than X yields a nontrivial factor of $f(x)$ over $\mathbb{F}_p$.*

The proof of the above theorem goes via finding ℓ^{th} roots of elements in $\mathcal{R}$ and this is where the assumption of the ERH is used (see Section 3.3.4).

3.3.6 Gao's Algorithm

In this section, we briefly present Gao's factoring algorithm (Gao01), based on which our algorithm has been designed. Suppose, the polynomial $f(y)$ splits in $\mathcal{R}$ as $f(y) = (y - X)\hat{f}(y)$. Define the quotient ring $\mathcal{S} = \frac{\mathcal{R}[y]}{(\hat{f})} = \mathcal{R}[Y]$, where $Y = y \bmod \hat{f}$. Like $\mathcal{R}$, ring $\mathcal{S}$ is also a completely splitting semisimple algebra over $\mathbb{F}_p$ with dimension $n' = n(n-1)$. Gao (Gao01) described an algorithm σ for taking square root of an element in $\mathcal{S}$. If $p - 1 = 2^e w$ where w is odd and η is a primitive 2^e-th root of unity in $\mathbb{F}_p$, then σ has the following properties:

1. Let $\mu_1, \ldots, \mu_{n'}$ be the primitive idempotents of $\mathcal{S}$ and $\alpha = \sum_{i=1}^{n'} a_i \mu_i \in \mathcal{S}$ where $a_i \in \mathbb{F}_p$. Then, $\sigma(\alpha) = \sum_{i=1}^{n'} \sigma(a_i)\mu_i$.

2. Let $a = \eta^u \theta$ where $\theta \in \mathbb{F}_p$ with $\theta^w = 1$ and $0 \leq u < 2^e$. Then $\sigma(a^2) = a$ if and only if $u < 2^{e-1}$.

Notice that, when $p = 3 \bmod 4$, $\eta = -1$ and property 2 implies that $\sigma(a^2) = a$ if and only if a is a quadratic residue in $\mathbb{F}_p$. The following is Gao's algorithm to factor f. It either finds a proper factor of f or outputs that "f is square balanced".

Algorithm 3 : Gao's algorithm

1. Form X, Y, $\mathcal{R}$ and $\mathcal{S}$ as before.
2. Compute $C = \frac{1}{2}(X + Y + \sigma((X - Y)^2)) \in \mathcal{S}$..
3. Compute the characteristic polynomial $c(y)$ of C over $\mathcal{R}$.
4. Factor $c(y) = h(y)(y - X)^t$, where t is the largest possible.
5. If $h(X)$ is a zero divisor in $\mathcal{R}$, find a proper factor of f.
6. Otherwise output that 'f is square balanced'.

Gao's result - It was shown in (Gao01) that Algorithm 3 fails to find a proper factor of f if and only if f is square balanced. Moreover, it follows from the analysis in (Gao01) (see Theorem 3.1 in (Gao01)) that when f is square balanced the polynomial $h(y)$ takes the form,

$$h(y) = \sum_{i=1}^{n} \left[\prod_{j \in \Delta_i} (y - \xi_j) \right] \chi_i$$

where $\Delta_i = \{j : j \neq i, \sigma((\xi_i - \xi_j)^2) = -(\xi_i - \xi_j)\}$ and $\#\Delta_i = \frac{n-1}{2}$ for all i, $1 \leq i \leq n$.

3.4 Our Algorithm and Analysis

We are now ready to present our algorithm and analyse it. The algorithm involves k polynomials, $f_1 = f, f_2, \ldots, f_k$, where the polynomial f_ℓ, $1 < \ell \leq k$, is defined as,

$$f_\ell = \prod_{i=1}^{n} (x - p_\ell(\xi_i)),$$

$p_\ell(.)$ is an arbitrary but deterministically fixed polynomial with degree bounded by $(n \log p)^{O(1)}$ and $p_{\ell_1}(.) \neq p_{\ell_2}(.)$ for $\ell_1 \neq \ell_2$. The polynomial f_ℓ can be constructed in polynomial time by considering the element $p_\ell(X)$ in $\mathcal{R}$, and then computing its characteristic polynomial over $\mathbb{F}_p$. We begin by simplifying the form of f_ℓ.

3.4.1 A Simplifying Lemma

Lemma 3.4.1. *If f_ℓ is not of the form $f_\ell = \tilde{f}_\ell^{\,d_\ell}$, where $\tilde{f}_\ell$ is a square-free, square balanced polynomial and $d_\ell > 0$, then a proper factor of f can be retrieved in polynomial time.*

Proof. By definition, $f_\ell = \prod_{i=1}^{n} (x - p_\ell(\xi_i))$. Define the sets E_i, for $1 \leq i \leq n$, as $E_i = \{1 \leq j \leq n : p_\ell(\xi_j) = p_\ell(\xi_i)\}$. Consider the following gcd in the ring $\mathcal{R}[y]$,

$$g(y) = \gcd\left(p_\ell(y) - p_\ell(X), f(y)\right) = \sum_{i=1}^{n} \left[\prod_{j \in E_i} (y - \xi_j) \right] \chi_i$$

The leading coefficient of $g(y)$ is a zero-divisor in $\mathcal{R}$, unless $\#E_1 = \ldots = \#E_n = d_\ell$ (say). Therefore, we can assume that,

$$\begin{aligned} f_\ell &= \prod_{j=1}^{m_\ell} \big(x - p_\ell(\xi_{s_j})\big)^{d_\ell} \quad \text{where } p_\ell(\xi_{s_1}), \ldots, p_\ell(\xi_{s_{m_l}}) \text{ are all distinct and } m_\ell = \frac{n}{d_\ell} \\ &= \tilde{f}_\ell^{\,d_\ell} \quad \text{where } \tilde{f}_\ell = \prod_{j=1}^{m_\ell} \big(x - p_\ell(\xi_{s_j})\big) \text{ is square-free.} \end{aligned}$$

If polynomial $\tilde{f}_\ell$ (obtained by square-freeing f_ℓ) is not square balanced then a proper factor $\tilde{g}_\ell$ of $\tilde{f}_\ell$ is returned by Algorithm 3. But then,

$$\gcd\left(\tilde{g}_\ell(p_\ell(x)), f(x)\right) = \prod_{j:\tilde{g}_\ell(p_\ell(\xi_j))=0} (x - \xi_j)$$

is a proper factor of f. □

On input $\tilde{f}_\ell = \prod_{j=1}^{m_\ell} \big(x - p_\ell(\xi_{s_j})\big)$, Algorithm 3 returns a polynomial $h_\ell(y)$ such that,

$$h_\ell(y) = \sum_{j=1}^{m_\ell} \left[\prod_{r \in \tilde{\Delta}_j^{(\ell)}} (y - p_\ell(\xi_{s_r})) \right] \chi_j^{(\ell)} \tag{3.1}$$

where $\chi_j^{(\ell)}$'s are the primitive idempotents of the ring $\mathcal{R}_\ell = \frac{\mathbb{F}_p[x]}{(\tilde{f}_\ell)}$,

$$\tilde{\Delta}_j^{(\ell)} = \{1 \le r \le m_\ell : r \ne j, \sigma((p_\ell(\xi_{s_j}) - p_\ell(\xi_{s_r}))^2) = -(p_\ell(\xi_{s_j}) - p_\ell(\xi_{s_r}))\}$$

and $\#\tilde{\Delta}_j^{(\ell)} = \frac{m_\ell - 1}{2}$ for $1 \le j \le m_\ell$.

3.4.2 Our Algorithm

The algorithm consists of four main steps. For convenience of presentation, we first deal with each of these steps separately, explaining how they are implemented. Finally, we combine these steps together to present the factoring algorithm. In what follows, assume that $p > n^2$ and n is odd, as even degree polynomials can be factored in polynomial time. Also, parameter k is taken to be a fixed polynomial in n and $\log p$ and c is a fixed constant.

Step 1: Constructing polynomial f_ℓ and checking if f factors

Construct polynomial f_ℓ - Compute the characteristic polynomial, $c_\alpha(x)$, of element $\alpha = p_\ell(X) \in \mathcal{R}$, over $\mathbb{F}_p$. Then $f_\ell = c_\alpha(x)$.

Check if f can be factored using f_ℓ - Check if f_ℓ is of the form $f_\ell = \tilde{f}_\ell^{\,d_\ell}$, where $\tilde{f}_\ell$ is a square-free, square balanced polynomial and $d_\ell > 0$. If not, then find a proper factor of f as in Lemma 3.4.1.

Step 2: Constructing graph G_ℓ implicitly

Apply Algorithm 3 to f_ℓ - From Step 1, $\tilde{f}_\ell$ is square balanced and Algorithm 3 returns a polynomial $h_\ell(y) = y^t + \alpha_1 y^{t-1} + \ldots + \alpha_t$ (as in equation 3.1), where $t = \frac{m_\ell - 1}{2}$ and $\alpha_u \in \mathcal{R}_\ell$ for $1 \leq u \leq t$.

Change to a common ring - Each $\alpha_u \in \mathcal{R}_\ell$ is a polynomial $\alpha_u(x) \in \mathbb{F}_p[x]$ of degree less than m_ℓ. Compute α'_u as, $\alpha'_u = \alpha_u(p_\ell(x)) \bmod f$, for $1 \leq u \leq t$, and construct the polynomial $h'_\ell(y) = y^t + \alpha'_1 y^{t-1} + \ldots + \alpha'_t \in \mathcal{R}[y]$. This step is to ensure that taking gcd is feasible.

Construct graph G_ℓ implicitly - If $\ell = 1$ then assign $g_\ell(y) = h'_\ell(y) \in \mathcal{R}[y]$ and continue with the next value of ℓ. Else, construct the polynomial $h'_\ell(p_\ell(y))$ by replacing y by $p_\ell(y)$ in $h'_\ell(y)$ and compute $g_\ell(y)$ as,

$$g_\ell(y) = \gcd(g_{\ell-1}(y), h'_\ell(p_\ell(y))) \in \mathcal{R}[y].$$

Check if G_ℓ is a null graph - Let $g_\ell(y) = \beta_{t'} y^{t'} + \ldots + \beta_0$, where t' is the degree of $g_\ell(y)$ and $\beta_u \in \mathcal{R}$ for $0 \leq u \leq t'$. If $t' = 0$ then make $g_\ell(y) = g_{\ell-1}(y)$ and continue with the next value of ℓ.

Step 3: Checking for equal out degrees of the vertices of G_ℓ

Check if out degrees are equal - Say, $t' > 0$. If $\beta_{t'}$ is a zero divisor in $\mathcal{R}$, construct a proper factor of f from $\beta_{t'}$ and stop.

Factor if out degrees are small - Else, if $t' \leq c$ then use Evdokimov's algorithm (Evd94) on $g_\ell(y)$ to find a proper factor of f in $(n \log p)^{O(1)}$ time.

Step 4: Checking for equal in degrees of the vertices of G_ℓ

Obtain the values of a nice polynomial at multiple points - If $t' > c$, evaluate $g_\ell(y) \in \mathcal{R}[y]$ at $n \cdot t'$ distinct points $y_1, \ldots, y_{nt'}$ taken from $\mathbb{F}_p$. Find the characteristic polynomials of the elements $g_\ell(y_1), \ldots, g_\ell(y_{nt'}) \in \mathcal{R}$ over $\mathbb{F}_p$ as $c_1(x), \ldots, c_{nt'}(x) \in \mathbb{F}_p[x]$, respectively. Collect the terms $c_i(0)$ for $1 \leq i \leq nt'$.

Construct the nice polynomial from the values - Construct the polynomial $r(x) = x^{nt'} + r_1 x^{nt'-1} + \ldots + r_{nt'} \in \mathbb{F}_p[x]$ such that $r(y_i) = -c_i(0)$ for $1 \leq i \leq nt'$. Solve for $r_i \in \mathbb{F}_p$, $1 \leq i \leq nt'$, using linear algebra.

Check if in degrees are equal - For $0 \leq i < t'$, if $f^i(x)$ divides $r(x)$ then compute the gcd $\left(\frac{r(x)}{f^i(x)}, f(x)\right) \in \mathbb{F}_p[x]$. If a proper factor of f is found, stop. Else, continue with the next value of ℓ.

Algorithm 4 : Cross Balance Factoring Algorithm

```
0.  Choose the auxiliary polynomials p_2, ..., p_k.  Take p_1(y) = y.
    for ℓ = 1 to k do
1.     Construct the polynomial f_ℓ.
2.     Construct the graph G_ℓ (implicitly).
3.     Check if out degrees of the vertices of G_ℓ are equal.
4.     Check if in degrees of the vertices of G_ℓ are equal.
    If no proper factor is found in Steps 1-4, return 'Failure'.
```

3.4.3 Proof of the Main Theorem

The proof of Theorem 3.2.3 follows from the next theorem and the lemma thereafter.

Theorem 3.4.2. *Algorithm 4 fails to find a proper factor f in $k \cdot (n \log p)^{O(1)}$ time if and only if f is k-cross balanced and regularity of the graph G_k is greater than c.*

Proof. We show that Algorithm 4 fails to find a proper factor of f at the ℓ^{th} iteration of the loop if and only if f is ℓ-cross balanced and the regularity of G_ℓ is greater than c. Recall the definitions of the sets $\Delta_i^{(\ell)}$ and $D_i^{(\ell)}$, $1 \leq i \leq n$, from Section 3.2.1. The set $\Delta_i^{(\ell)}$ is defined as,

$$\Delta_i^{(\ell)} = \{1 \leq j \leq n : p_\ell(\xi_i) \neq p_\ell(\xi_j), \sigma((p_\ell(\xi_i) - p_\ell(\xi_j))^2) = -(p_\ell(\xi_i) - p_\ell(\xi_j))\}$$

And the set $D_i^{(\ell)}$ is defined iteratively over ℓ as,

$$\begin{aligned} D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } \ell > 1,\ D_i^{(\ell)} &= D_i^{(\ell-1)} \cap \Delta_i^{(\ell)} \\ \text{If } D_i^{(\ell)} &= \phi \text{ for all } i,\ 1 \leq i \leq n, \text{ then } D_i^{(\ell)} \text{ is redefined as } D_i^{(\ell)} = D_i^{(\ell-1)}. \end{aligned}$$

Graph G_ℓ, with n vertices $v_1, \ldots, v_n$, has an edge from v_i to v_j iff $j \in D_i^{(\ell)}$.

Algorithm 4 fails at the first iteration ($\ell = 1$) if and only if f is square balanced. In this case, $D_i^{(1)} = \Delta_i^{(1)} = \Delta_i$, the polynomial $g_1(y)$ is,

$$g_1(y) = h(y) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(1)}} (y - \xi_j) \right] \chi_i$$

and G_1 is regular with in degree and out degree of a vertex v_i equal to $\#D_i^{(1)} = \#\Delta_i = \frac{n-1}{2}$. Thus, the polynomial f is 1-cross balanced and $\deg(g_1(y)) = \frac{n-1}{2}$. If Algorithm 4 fails at the ℓ^{th} iteration, then we can assume that the polynomials $f = \tilde{f}_1, \ldots, \tilde{f}_\ell$ are square free and square balanced (by Lemma 3.4.1).

Suppose that, Algorithm 4 fails at the ℓ^{th} iteration. Then, $\tilde{f}_\ell = \prod_{j=1}^{m_\ell} \left(x - p_\ell(\xi_{s_j})\right)$ is square free and square balanced, and Algorithm 3 returns the polynomial $h_\ell(y) \in \mathcal{R}_\ell[y]$ such that,

$$h_\ell(y) = \sum_{j=1}^{m_\ell} \left[\prod_{r \in \tilde{\Delta}_j^{(\ell)}} (y - p_\ell(\xi_{s_r})) \right] \chi_j^{(\ell)} \tag{3.2}$$

where $\chi_j^{(\ell)}$'s are the primitive idempotents of the ring $\mathcal{R}_\ell = \frac{\mathbb{F}_p[x]}{(\tilde{f}_\ell)}$ and,

$$\tilde{\Delta}_j^{(\ell)} = \{1 \leq r \leq m_\ell : r \neq j, \sigma((p_\ell(\xi_{s_j}) - p_\ell(\xi_{s_r}))^2) = -(p_\ell(\xi_{s_j}) - p_\ell(\xi_{s_r}))\}$$

Let, $h_\ell(y) = y^t + \alpha_1 y^{t-1} + \ldots + \alpha_t$, where $t = \frac{m_\ell - 1}{2}$ and $\alpha_u \in \mathcal{R}_\ell$ for $1 \leq u \leq t$. Each $\alpha_u \in \mathcal{R}_\ell$ is a polynomial $\alpha_u(x) \in \mathbb{F}_p[x]$ with degree less than m_ℓ and if

$\alpha_u = \sum_{j=1}^{m_\ell} a_{uj}\chi_j^{(\ell)}$ for $a_{uj} \in \mathbb{F}_p$, then by the Chinese Remaindering theorem (and assuming the correspondence between $\chi_j^{(\ell)}$ and the factor $(x - p_\ell(\xi_{s_j}))$ of $\tilde{f}_\ell$) we get,

$$\begin{aligned}
\alpha_u(x) &= q(x)(x - p_\ell(\xi_{s_j})) + a_{uj} \quad \text{for some polynomial } q(x) \in \mathbb{F}_p[x] \\
\Rightarrow \alpha_u(p_\ell(x)) &= q(p_\ell(x))(p_\ell(x) - p_\ell(\xi_{s_j})) + a_{uj} \\
\Rightarrow \alpha_u(p_\ell(x)) &= a_{uj} \bmod (x - \xi) \quad \text{for every } \xi \in \{\xi_1, \ldots, \xi_n\} \text{ such that } p_\ell(\xi) = p_\ell(\xi_{s_j}).
\end{aligned}$$

Suppose that, for a given i $(1 \leq i \leq n)$, $j(i)$ $(1 \leq j(i) \leq m_\ell)$ is a unique index such that, $p_\ell(\xi_i) = p_\ell(\xi_{s_{j(i)}})$. Then, the polynomial $\alpha'_u(x) = \alpha_u(p_\ell(x)) \bmod f$ has the following direct sum (or canonical) representation in the ring $\mathcal{R}$,

$$\alpha'_u(x) = \sum_{i=1}^{n} a_{uj(i)}\chi_i.$$

This implies that the polynomial $h'_\ell(y) = y^t + \alpha'_1 y^{t-1} + \ldots + \alpha'_t \in \mathcal{R}[y]$ has the canonical representation,

$$h'_\ell(y) = \sum_{i=1}^{n} \left[\prod_{r \in \tilde{\Delta}_{j(i)}^{(\ell)}} (y - p_\ell(\xi_{s_r})) \right] \chi_i. \tag{3.3}$$

Inductively, assume that $g_{\ell-1}(y)$ has the form,

$$g_{\ell-1}(y) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(\ell-1)}} (y - \xi_j) \right] \chi_i.$$

Then,

$$\begin{aligned}
g_\ell(y) &= \gcd\left(g_{\ell-1}(y), h'_\ell(p_\ell(y))\right) \\
&= \sum_{i=1}^{n} \gcd\left(\prod_{j \in D_i^{(\ell-1)}} (y - \xi_j), \prod_{r \in \tilde{\Delta}_{j(i)}^{(\ell)}} (p_\ell(y) - p_\ell(\xi_{s_r})) \right) \chi_i \\
&= \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(\ell-1)} \cap \Delta_i^{(\ell)}} (y - \xi_j) \right] \chi_i \quad (\text{as } r \in \tilde{\Delta}_{j(i)}^{(\ell)} \Leftrightarrow s_r \in \Delta_i^{(\ell)})
\end{aligned}$$

Therefore,

$$\begin{aligned} g_\ell(y) &= \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(\ell)}} (y - \xi_j) \right] \chi_i \\ &= \beta_{t'} y^{t'} + \ldots + \beta_0 \quad \text{(say)} \end{aligned}$$

where $t' = \max_i \left(\#D_i^{(\ell)} \right)$ and $\beta_u \in \mathcal{R}$ for $1 \leq u \leq t' \leq \frac{n-1}{2}$. The element $\beta_{t'}$ is not a zero divisor in $\mathcal{R}$ if and only if $\#D_1^{(\ell)} = \ldots = \#D_n^{(\ell)} = t'$. If $t' \leq c$ then a factor of f can be retrieved from $g_\ell(y)$ in polynomial time using already known methods (Evd94). The condition $\#D_i^{(\ell)} = t'$ for all $i, 1 \leq i \leq t'$, makes the out degree of every vertex in G_ℓ equal to t'. However, this may not necessarily imply that the in degree of every vertex in G_ℓ is also t'. Checking for identical in degrees of the vertices of G_ℓ is handled in Step 4 of the algorithm. Consider evaluating the polynomial $g_\ell(y)$ at a point $y_s \in \mathbb{F}_p$.

$$g_\ell(y_s) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(\ell)}} (y_s - \xi_j) \right] \chi_i \in \mathcal{R}.$$

The characteristic polynomial of $g_\ell(y_s)$ over $\mathbb{F}_p$ is,

$$\begin{aligned} c_s(x) &= \prod_{i=1}^{n} \left(x - \prod_{j \in D_i^{(\ell)}} (y_s - \xi_j) \right) \\ \Rightarrow -c_s(0) &= \prod_{j=1}^{n} (y_s - \xi_j)^{k_j} \quad \text{(since } n \text{ is odd)}, \end{aligned}$$

where k_j is the in degree of vertex v_j in G_ℓ. Let $r(x) = x^{nt'} + r_1 x^{nt'-1} + \ldots + r_{nt'} \in \mathbb{F}_p[x]$ be a polynomial of degree nt', such that,

$$r(y_s) = -c_s(0) = \prod_{j=1}^{n} (y_s - \xi_j)^{k_j},$$

for nt' distinct points $\{y_s\}_{1 \leq s \leq nt'}$ taken from $\mathbb{F}_p$. Since we have assumed that $p > n^2 > \frac{n(n-1)}{2} \geq nt'$, we can solve for the coefficients $r_1, \ldots, r_{nt'}$ using any nt' distinct points from $\mathbb{F}_p$. Then,

$$r(x) = \prod_{j=1}^{n} (x - \xi_j)^{k_j}.$$

If $k_j \neq t'$ for some j, then there is an $i = \min\{k_1, \ldots, k_n\} < t'$ such that $f^i(x)$ divides $r(x)$ and the gcd $\left(\frac{r(x)}{f^i(x)}, f(x)\right)$ yields a nontrivial factor of $f(x)$. This shows that the graph G_ℓ is regular if the algorithm fails at the ℓ^{th} step. Since $\deg(g_\ell(y))$ equals the regularity of G_ℓ, if the latter quantity is less than c then we can apply Evdokimov's algorithm (Evd94) on $g_\ell(y)$ and get a non trivial factor of f in polynomial time. □

Recall the definition of graph H_ℓ from Section 3.2.2. We say that an auxiliary polynomial $p_\ell(\cdot)$ is good if either H_ℓ is not regular or $G_\ell \neq G_{\ell-1}$. We show that, only a few good auxiliary polynomials are needed.

Lemma 3.4.3. *Algorithm* 4 *(with a slight modification) requires at most* $\log_2 n$ *good auxiliary polynomials to find a proper factor of* f*.*

Proof. Consider the following modification of Algorithm 4. For $\ell > 1$, take $g_\ell(y)$ to be either $\gcd(g_{\ell-1}(y), h'_\ell(p_\ell(y)))$ or $g_{\ell-1}(y)/\gcd(g_{\ell-1}(y), h'_\ell(p_\ell(y)))$, whichever has the smaller nonzero degree. Accordingly, we modify the definition of graph G_ℓ. Define the set $\bar{\Delta}_i^{(\ell)}$ $(1 \leq i \leq n)$ as,

$$\begin{aligned}\bar{\Delta}_i^{(\ell)} &= \{1 \leq j \leq n : j \neq i, \sigma((p_\ell(\xi_i) - p_\ell(\xi_j))^2) = (p_\ell(\xi_i) - p_\ell(\xi_j))\} \\ &= \{1 \leq j \leq n : j \neq i\} - \Delta_i^{(\ell)}\end{aligned}$$

and modify the definition of the sets $D_i^{(\ell)}$ $(1 \leq i \leq n)$ as,

$$\begin{aligned}D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } \ell > 1,\ D_i{}^{(\ell)} &= D_i{}^{(\ell-1)} \cap \Delta_i^{(\ell)} \text{ if } g_\ell(y) = \gcd(g_{\ell-1}(y), h'_\ell(p_\ell(y))) \\ &= D_i{}^{(\ell-1)} \cap \bar{\Delta}_i^{(\ell)} \text{ else if } g_\ell(y) = g_{\ell-1}(y)/\gcd(g_{\ell-1}(y), h'_\ell(p_\ell(y)))\end{aligned}$$

As before, an edge (v_i, v_j) is present in G_ℓ iff $j \in D_i^{(\ell)}$. This modification ensures that, if $g_\ell(y) \neq g_{\ell-1}(y)$ has an invertible leading coefficient (i.e if $g_\ell(y)$ is monic) then the degree of $g_\ell(y)$ is at most half the degree of $g_{\ell-1}(y)$. Hence, for every good choice of polynomial $p_\ell(\cdot)$ if $G_{\ell-1}$ and G_ℓ are $t_{\ell-1}$-regular and t_ℓ-regular, respectively, then $t_\ell \leq \frac{t_{\ell-1}}{2}$. Therefore, at most $\log_2 n$ good choices of polynomials $p_\ell(\cdot)$ are required by the algorithm. □

Theorem 3.2.3 follows as a corollary to Theorem 3.4.2 and Lemma 3.4.3. As already pointed out in Section 3.2.2, if only $\epsilon\lceil\log_2 n\rceil$ good auxiliary polynomials are

available for some ϵ, $0 < \epsilon \leq 1$, then we obtain a nontrivial factor $g(y)$ of $\hat{f}(y)$ with degree at most $\frac{n^{1-\epsilon}}{2}$. If we apply Evdokimov's algorithm on $g(y)$ instead of $\hat{f}(y)$, then the maximum dimension of the rings considered is bounded by $n^{\frac{(1-\epsilon)^2}{2}\log n+\epsilon+O(1)}$ instead of $n^{\frac{\log n}{2}+O(1)}$ (as is the case in (Evd94)).

3.5 Density of Square Balanced Polynomials

In this section, we give a bound on the fraction of square balanced polynomials for the case of $p = 3 \bmod 4$, when p is sufficiently large. We need a result by Weil (see Schmidt (Sch76), page - 43) on character sums to prove the following lemma.

Weil's theorem - Let χ be the quadratic character, mapping an $a \in \mathbb{F}_p$ to 1 if a is a quadratic residue and -1 otherwise. If $h \in \mathbb{F}_p[x]$ is a degree d monic polynomial that is not a square of another polynomial then,

$$\left|\sum_{a \in F_p} \chi(h(a))\right| \leq (d-1)\sqrt{p}.$$

Lemma 3.5.1. *If $p = 3$ mod 4 and $p \geq n^6 2^{2n}$ then about $\frac{(1+o(1))^n}{(\frac{\pi}{2}n)^{\frac{n}{2}}}$ fraction of all completely splitting, square-free polynomials of degree n are square balanced.*

Proof. With every completely splitting, square-free polynomial $f = \prod_{i=1}^{n}(x - a_i)$, we can associate a tournament H on n vertices $\{v_1, \ldots, v_n\}$ with the natural correspondence between vertex v_i and root a_i for all i, $1 \leq i \leq n$, such that (v_i, v_j) is an edge in H iff $(a_i - a_j)$ is a quadratic non-residue. Such a tournament is also known as *Paley tournament*. Since $p = 3 \bmod 4$, f is a square balanced polynomial iff H is a labeled regular tournament on n vertices. We now bound the fraction of polynomials that can be associated to a particular labeled tournament.

Let T be a fixed labeled tournament. Suppose that, for some $i > 1$, the elements in $S_{i-1} = \{a_1, \ldots, a_{i-1}\}$ are already chosen in a way that is compatible to the subgraph of T induced by the vertices $v_1, \ldots, v_{i-1}$. Let n_i be the number of possible values of a_i such that the edge constraints from v_i to vertices $v_1, \ldots, v_{i-1}$

are maintained. Then n_i can be equated as,

$$2^{i-1} \cdot n_i = \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \prod_{j=1}^{i-1} (1 +_j \chi(x_i - a_j)),$$

where χ is the quadratic character and the symbol $+_j$ is $+$ if (v_j, v_i) is an edge in T and $-$ if (v_i, v_j) is an edge in T. By expanding the inner product we get,

$$\begin{aligned} 2^{i-1} \cdot n_i &= \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \sum_{0 \le e_1, \ldots, e_{i-1} \le 1} \operatorname{sign}(e_1, \ldots, e_{i-1}) \cdot \chi\left(\prod_{j=1}^{i-1} (x_i - a_j)^{e_j}\right) \\ &= \sum_{0 \le e_1, \ldots, e_{i-1} \le 1} \operatorname{sign}(e_1, \ldots, e_{i-1}) \cdot \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi\left(\prod_{j=1}^{i-1} (x_i - a_j)^{e_j}\right). \end{aligned}$$

The term corresponding to $e_1 = \ldots = e_{i-1} = 0$ is $(p - i)$. For any other fixed $e_1, \ldots, e_{i-1}$, the inner sum is of the form,

$$\operatorname{sign}(e_1, \ldots, e_{i-1}) \cdot \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi(q(x_i)),$$

where $q(x_i) = \prod_{j=1}^{i-1} (x_i - a_j)^{e_j}$ is not a perfect square as $a_1, \ldots, a_{i-1}$ are distinct, and $1 \le \deg(q(x_i)) \le (i-1)$. By applying Weil's theorem we get,

$$\begin{aligned} \operatorname{sign}(e_1, \ldots, e_{i-1}) \cdot \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi(q(x_i)) &\le \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi(q(x_i)) \\ &\le \left| \sum_{x_i \in \mathbb{F}_p} \chi(q(x_i)) \right| + i \\ &\le (i-2)\sqrt{p} + i. \end{aligned}$$

Therefore,

$$\begin{aligned} 2^{i-1} \cdot n_i &\le (p-i) + (2^{i-1} - 1)((i-2)\sqrt{p} + i) \\ \Rightarrow \quad n_i &\le \frac{p-i}{2^{i-1}} + (i-2)\sqrt{p} + i \\ \Rightarrow \frac{n_i}{p-i} &\le \frac{1}{2^{i-1}} \left(1 + \frac{2^{i-1}((i-2)\sqrt{p} + i)}{p-i}\right) \\ &\le \frac{1}{2^{i-1}} \left(1 + \frac{1}{n^2}\right) \qquad \text{if } p \ge n^6 2^{2n} \end{aligned}$$

And hence,

$$\Pr_{a_1,\ldots,a_n}\{T \text{ is induced by } \{a_1,\ldots,a_n\}\} \leq \frac{e^{\frac{1}{n}}}{2^{\binom{n}{2}}} = \frac{1+o(1)}{2^{\binom{n}{2}}}$$

By a similar argument we get,

$$\begin{aligned}\text{sign}(e_1,\ldots,e_{i-1}) \cdot \sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi(q(x_i)) &\geq -\sum_{x_i \in \mathbb{F}_p^\times \setminus S_{i-1}} \chi(q(x_i)) \\ &\geq -((i-2)\sqrt{p}+i)\end{aligned}$$

which implies that,

$$\begin{aligned}\frac{n_i}{p-i} &\geq \frac{1}{2^{i-1}}\left(1-\frac{1}{n^2}\right) \qquad \text{if } p \geq n^6 2^{2n} \\ \Rightarrow \Pr_{a_1,\ldots,a_n}\{T \text{ is induced by } \{a_1,\ldots,a_n\}\} &\geq \frac{e^{-\frac{1}{n}}}{2^{\binom{n}{2}}} = \frac{1+o(1)}{2^{\binom{n}{2}}} \\ \Rightarrow \Pr_{a_1,\ldots,a_n}\{T \text{ is induced by } \{a_1,\ldots,a_n\}\} &= \frac{1+o(1)}{2^{\binom{n}{2}}}\end{aligned}$$

The polynomial f is square balanced iff the associated tournament H is regular. The number of regular tournaments on n vertices is given by (Spe74) as,

$$R_n = 2^{\binom{n}{2}} \frac{(1+o(1))^n}{(\frac{\pi}{2}n)^{\frac{n}{2}}}$$

Therefore,

$$\Pr_{a_1,\ldots,a_n}\{\text{polynomial } f = \prod_{i=1}^{n}(x-a_i) \text{ is square balanced}\} \approx \frac{(1+o(1))^n}{(\frac{\pi}{2}n)^{\frac{n}{2}}}$$

□

Corollary 3.5.2. *If $p = 3$ mod 4, $p > n^6 2^{2n}$ and $p_\ell(y)$ is a uniformly randomly chosen polynomial of degree $(n-1)$ then the probability that f_ℓ is either not square-free or is a square-free and square balanced polynomial is upper bounded by $\frac{(1+o(1))^n}{(\frac{\pi}{2}n)^{\frac{n}{2}}}$.*

It follows that, for $p = 3 \bmod 4$ and $p > n^6 2^{2n}$, if the auxiliary polynomials $p_\ell(\cdot)$'s are uniformly randomly chosen then Algorithm 4 works in randomized polynomial time. However, the argument used in the proof of Lemma 3.5.1 does not seem to apply to the case $p = 1 \bmod 4$. Therefore, we resort to a different analysis in the following section, although in the process we get a slightly weaker probability bound.

3.6 Choice of Auxiliary Polynomials

3.6.1 Random Auxiliary Polynomials

We show that random choices of auxiliary polynomials break the cross balance symmetry with high probability thereby making the randomized variant of Algorithm 4 run in polynomial time.

Lemma 3.6.1. *If G_ℓ $(1 \leq \ell < k)$ is regular and $p_{\ell+1}(y) = \sum_{m=1}^{n-1} c_m y^m \in F_p[x]$ is a randomly chosen polynomial of degree $(n-1)$ then $G_{\ell+1} \neq G_\ell$ with probability at least $1 - \frac{1}{2^{0.9n-2}}$.*

Proof. Suppose that, G_ℓ is t-regular where $t \geq 5$. Otherwise, if $t = \deg(g_\ell(y)) < 5$ then we can efficiently solve for a root of $g_\ell(y)$ (or $f(y)$, since $g_\ell(y)$ divides $f(y)$) in $\mathcal{R}$ using radicals. In the process either a nontrivial endomorphism of $\mathcal{R}$ or a zero divisor in $\mathcal{R}$ is obtained.

Assume that G_ℓ has c connected components. Surely, $c \leq \frac{n}{2t+1}$ as G_ℓ is t-regular. Consider any connected component G_ℓ^s $(1 \leq s \leq c)$ with n_s vertices and T_ℓ^s be a tree formed by a depth-first traversal of G_ℓ^s with any particular vertex in G_ℓ^s as the root. Since G_ℓ is a regular digraph, any connected component is also strongly connected and hence T_ℓ^s is a spanning tree of G_ℓ^s containing n_s vertices with $n_s - 1$ edges. Let the vertices of T_ℓ^s be $v_1, \ldots, v_{n_s}$ and the roots of f associated with these vertices be $\xi_1, \ldots, \xi_{n_s}$, respectively. For every edge $e_{ij} = (v_i, v_j)$ in T_ℓ^s, $1 \leq s \leq c$, consider the equation,

$$p_{\ell+1}(\xi_i) - p_{\ell+1}(\xi_j) = \sum_{m=1}^{n-1} c_m(\xi_i^m - \xi_j^m) = a_{ij} \tag{3.4}$$

for a given $a_{ij} \in \mathbb{F}_p$ with $\bar{c} = (c_1, \ldots, c_{n-1})$ as the variables. Let $p - 1 = 2^e w$, where w is odd. Recall from Section 3.3.6, Gao (Gao01) showed that given a primitive 2^e-th root of unity η, if $a = \eta^u \theta$ where $\theta^w = 1$ then $\sigma(a^2) = a$ iff $u < 2^{e-1}$. We say that two elements $a, b \in \mathbb{F}_p$ have the *same character* if, $\sigma(a^2) = a$ if and only if $\sigma(b^2) = b$. For convenience, assume that 0 has the same character as any other non-zero element a iff $\sigma(a^2) = a$.

An edge e_{ij} in G_ℓ is also present in $G_\ell \cap H_{\ell+1}$ if and only if $a_{ij} = \eta^u \theta$ with $u \geq 2^{e-1}$. Therefore, all the trees T_ℓ^s, $1 \leq s \leq c$, are either totally present or totally absent in $G_\ell \cap H_{\ell+1}$ iff all a_{ij}'s (corresponding to all the edges of the trees T_ℓ^s's)

have the same character. Writing down the equation (as in (3.4)) for every edge in every tree T_ℓ^s, $1 \leq s \leq c$, gives a set of $(n-c)$ linear equations in $n-1$ variables, namely $\bar{c} = \{c_1, \ldots, c_{n-1}\}$. Let $M_{n-c,n-1}$ be the coefficient matrix of the set of linear equations. It is not difficult to verify that $\text{rank}(M_{n-c,n-1}) = n-c$. Otherwise, the determinant of the matrix $(\xi_i^j)_{i,j}$, $1 \leq i \leq n$, $0 \leq j \leq n-1$ is 0, which is not possible since $\xi_i \neq \xi_j$ for $i \neq j$. This means, for every choice of values of the a_{ij}'s we have at most p^{c-1} choices of the polynomials $p_{\ell+1}(.)$ satisfying the linear equations. A polynomial $p_{\ell+1}(.)$ is a bad choice only if all a_{ij}'s have the same character. This can happen for at most,

$$\left(\frac{p-1}{2}\right)^{n-c} + \left(\frac{p+1}{2}\right)^{n-c} \leq 3 \cdot \left(\frac{p}{2}\right)^{n-c} \quad (\text{assuming } n-1 \leq \frac{p}{2})$$

different choices of the a_{ij}'s. Therefore, the probability that a random choice of $p_{\ell+1}(.)$ is bad is at most,

$$\Pr_{\bar{c}}\{\ p_{\ell+1}(.) \text{ is bad}\} < \frac{p^{c-1} \cdot 3 \cdot \left(\frac{p}{2}\right)^{n-c}}{p^{n-1}} < \frac{1}{2^{n-c-2}}$$

Since G_ℓ is t-regular $c \leq \frac{n}{2t+1}$. Therefore,

$$\Pr_{\bar{c}}\{\ p_{\ell+1}(.) \text{ is bad}\} < \frac{1}{2^{n\cdot\left(\frac{2t}{2t+1}\right)-2}} < \frac{1}{2^{0.9n-2}} \quad \text{assuming } t \geq 5$$

□

Thus, if polynomials $p_\ell(y)$, $1 < \ell \leq \lceil \log_2 n \rceil$, are randomly chosen, then the probability that f is not factored by Algorithm 4 within $\lceil \log_2 n \rceil$ iterations is less than $\frac{\lceil \log_2 n \rceil}{2^{0.9n-2}}$.

3.6.2 A Deterministic Choice with a Weak Bound

We show that if $p = 3 \bmod 4$ and the auxiliary polynomials are chosen as $p_{\ell+1}(y) = (y + 2^{-1}\ell)^2$ for $1 \leq \ell < k$ then it is sufficient to take $k = \lceil \sqrt{p} \log p \rceil$ to factor f. This can be argued as follows.

Suppose graph G_1 has the edges (v_i, v_j) and (v_i, v_r) where $j \neq r$. Then, $\xi_i - \xi_j$ and $\xi_i - \xi_r$ must have the same quadratic character. Also,

$$\begin{aligned} p_{\ell+1}(\xi_i) - p_{\ell+1}(\xi_j) &= (\xi_i - \xi_j)(\xi_i + \xi_j + \ell) \text{ and} \\ p_{\ell+1}(\xi_i) - p_{\ell+1}(\xi_r) &= (\xi_i - \xi_r)(\xi_i + \xi_r + \ell). \end{aligned}$$

Therefore, edges (v_i, v_j) and (v_i, v_r) are present in graph $G_{\ell+1}$ only if $\xi_i + \xi_j + \ell$ and $\xi_i + \xi_r + \ell$ have the same quadratic character. Shoup (Sho90) showed that if a and b are distinct elements of $\mathbb{F}_p$ and the quadratic characters of $a + i$ and $b + i$ are the same for all $0 \leq i \leq M$ then $M < \sqrt{p} \log p$. (This result holds for any odd prime p not just $p = 3 \bmod 4$.) Since $\xi_i + \xi_j$ and $\xi_i + \xi_r$ are distinct elements of $\mathbb{F}_p$, it follows from Shoup's result that if all the graphs $G_2, \ldots, G_k$ have both the edges (v_i, v_j) and (v_i, v_r) then $k \leq \sqrt{p} \log p$. In other words, within $\sqrt{p} \log p$ iterations Algorithm 4 finds a proper factor of f for this particular choice of auxiliary polynomials.

3.7 Conclusion

In this work, we have extended the square balance test by Gao (Gao01) and showed a direction towards improving the time complexity of the best previously known deterministic factoring algorithms. Using certain auxiliary polynomials, our algorithm attempts to exploit an inherent asymmetry among the roots of the input polynomial f in order to efficiently find a proper factor. The advantage of using auxiliary polynomials is that, unlike (Evd94), it avoids the need to carry out computations in rings with large dimensions, thereby saving overall computation time to a significant extent. Motivated by the stringent symmetry requirement from the roots of f, we pose the following question:

- Is it possible to construct good auxiliary polynomials in deterministic polynomial time?

An affirmative answer to the question will immediately imply that factoring polynomials over finite fields can be done in deterministic polynomial time under the assumption of the ERH.

Chapter 4

Integer Multiplication

-Joint work with Anindya De, Piyush Kurur and Ramprasad Saptharishi.

4.1 Introduction

In this chapter, we study the complexity of another important problem in computational number theory namely, integer multiplication. Being a basic arithmetic operation, it is no surprise that multiplications of integers occur as intermediate steps of computation in algorithms from every possible domain of computer science. But seldom do the complexity of such multiplication influence the overall efficiency of the algorithm as the integers involved are relatively small in size and their multiplications can often be implemented as a few fast hardware operations. However, with the advent of modern cryptosystems, the study of the bit complexity of integer multiplication received a significant impetus. Indeed, large integer multiplication forms the foundation of many modern day public-key cryptosystems, like RSA, El-Gamal and Elliptic Curve cryptosystems. One of the most notable applications is the RSA cryptosystem, where it is required to multiply two primes that are hundreds or thousands of bits long. The larger these primes the harder it is to factor their product, and this makes the RSA extremely secure in practice.

In this work, our focus is more on the theoretical aspects of integer multiplication, it being a fundamental problem in its own right. This is to say, we will be concerned with the asymptotic bit complexity of multiplying two N-bit integers with little emphasis on optimality in practice. It is worth mentioning that in most

cryptographic applications the key sizes are still sufficiently small to make simpler algorithms, like Karatsuba's algorithm (see Section 4.1.1), more efficient in practice than FFT based approaches that are nonetheless asymptotically better than the former. We begin with a brief account of the earlier work on integer multiplication algorithms.

4.1.1 Previous Work

The naive approach to multiply two N-bit integers leads to an algorithm that uses $O(N^2)$ bit operations. Karatsuba (KO63) showed that some multiplication operations of such an algorithm can be replaced by less costly addition operations which reduces the overall running time of the algorithm to $O(N^{\log_2 3})$ bit operations. Shortly afterwards, this result was improved by Toom (Too63) who showed that for any $\varepsilon > 0$, integer multiplication can be done in $O(N^{1+\varepsilon})$ time. This led to the question as to whether the time complexity can be improved further by replacing the term $O(N^{\epsilon})$ by a poly-logarithmic factor. In a major breakthrough, Schönhage and Strassen (SS71) gave two efficient algorithms for multiplying integers using Fast Fourier Transform (FFT). One of the algorithms achieved a running time of $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ using arithmetic over complex numbers (approximated to suitable precisions), while the other used arithmetic modulo carefully chosen integers to improve the complexity further to $O(N \cdot \log N \cdot \log\log N)$ bit operations. The modular algorithm remained the best for a long period of time until a recent remarkable result by Fürer (Für07) (see also (Für09)). Fürer gave an algorithm that uses arithmetic over complex numbers and runs in $N \cdot \log N \cdot 2^{O(\log^* N)}$ time. Till date this is the best time complexity known for integer multiplication and indeed our result is inspired by Fürer's algorithm.

4.1.2 The Motivation

Schönhage and Strassen introduced two seemingly different approaches to integer multiplication – using complex and modular arithmetic. Fürer's algorithm improves the time complexity in the complex arithmetic setting by cleverly reducing some costly multiplications to simple shift operations. However, the algorithm needs to approximate the complex numbers to certain precisions during computation. This

introduces the added task of bounding the total truncation errors in the analysis of the algorithm. On the contrary, in the modular setting the error analysis is virtually absent or rather more implicit, which in turn simplifies the overall analysis. In addition, modular arithmetic gives a discrete approach to a discrete problem like integer multiplication. Therefore, it seems natural to ask whether we can achieve a similar improvement in time complexity of this problem in the modular arithmetic setting. In this work, we answer this question affirmatively. We give an $N \cdot \log N \cdot 2^{O(\log^* N)}$ time algorithm for integer multiplication using only modular arithmetic, thus matching the improvement made by Fürer.

4.1.3 Overview of Our Result

As is the case in both Schönhage-Strassen's and Fürer's algorithms, we start by reducing the problem to polynomial multiplication over a ring $\mathcal{R}$ by properly encoding the given integers. Polynomials can be multiplied efficiently using the Discrete Fourier Transforms (DFT). However, in order that we are able to use the Fast Fourier Transform (FFT), the ring $\mathcal{R}$ should have some special roots of unity. For instance, to multiply two polynomials of degree less than M using the FFT, we require a *principal* $2M$-th root of unity (see Definition 4.2.1 for principal roots). One way to construct such a ring in the modular setting is to consider rings of the form $\mathcal{R} = \mathbb{Z}/(2^M + 1)\mathbb{Z}$ as in Schönhage and Strassen's work (SS71). In this case, the element 2 is a $2M$-th principal root of unity in $\mathcal{R}$. This approach can be equivalently viewed as attaching an 'artificial' root to the ring of integers. However, this makes the size of $\mathcal{R}$ equal to 2^M and thus a representation of an arbitrary element in $\mathcal{R}$ takes M bits. This means an N-bit integer is encoded as a polynomial of degree M with every coefficient about M bits long, thereby making $M \approx \sqrt{N}$ the optimal choice. Indeed, the choice of such an $\mathcal{R}$ is the basis of Schönhage and Strassen's modular algorithm in which they reduce multiplication of N-bit integers to multiplication of $\sqrt{N}$-bit integers and achieve a complexity of $O(N \cdot \log N \cdot \log\log N)$ bit operations.

It turns out that such rings are a little too expensive in our setting. We would rather like to find a ring whose size is bounded by some polynomial in M and which still contains a principal $2M$-th root of unity. In fact, it is this task of choosing a

suitable ring that poses the primary challenge in adapting Fürer's algorithm and making it work in the discrete setting.

We choose the ring to be $\mathcal{R} = \mathbb{Z}/p^c\mathbb{Z}$, for a prime p and a constant c such that $p^c = \mathsf{poly}(M)$. The ring $\mathbb{Z}/p^c\mathbb{Z}$, has a principal $2M$-th root of unity if and only if $2M$ divides $p-1$, which means that we need to find a prime p from the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$. To make this search computationally efficient, we also need the degree of the polynomials, M to be sufficiently small. This we can achieve by encoding the integers as multivariate polynomials instead of univariate ones. It turns out that the choice of the ring as $\mathcal{R} = \mathbb{Z}/p^c\mathbb{Z}$ is still not quite sufficient and needs a little more refinement. This is explained in Section 4.2.1.

The use of multivariate polynomial multiplications along with a small base ring are the main steps where our algorithm differs from earlier algorithms by Schönhage-Strassen and Fürer. Towards understanding the notion of the *inner* and *outer* DFTs in the context of multivariate polynomials, we also present a group theoretic interpretation of the Discrete Fourier Transform (DFT). The use of inner and outer DFTs play a central role in both Fürer's as well as our algorithm. Arguing along the line of Fürer (Für07), we show that repeated applications of efficiently computable inner DFTs, using some special roots of unity in $\mathcal{R}$, make the overall process efficient and leads to an $N \cdot \log N \cdot 2^{O(\log^* N)}$ time algorithm.

4.2 The Basic Setup

4.2.1 The Underlying Ring

Rings of the form $\mathcal{R} = \mathbb{Z}/(2^M + 1)\mathbb{Z}$ have the nice property that multiplications by powers of 2, the $2M$-th principal root of unity, are mere shift operations and are therefore very efficient. Although by choosing the ring $\mathcal{R} = \mathbb{Z}/p^c\mathbb{Z}$ we ensure that the ring size is small, it comes with a price: multiplications by principal roots of unity are no longer just shift operations. Fortunately, this can be redeemed by working with rings of the form $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ for some m whose value will be made precise later. Elements of $\mathcal{R}$ are thus $(m-1)$-degree polynomials over α with coefficients from $\mathbb{Z}/p^c\mathbb{Z}$. By construction, α is a $2m$-th root of unity and multiplication of any element in $\mathcal{R}$ by any power of α can be achieved by shift

operations — this property is crucial in making some multiplications in the FFT less costly (see Section 4.3.2).

Given an N-bit number a, we encode it as a k-variate polynomial over $\mathcal{R}$ with degree in each variable less than M. The parameters M and m are powers of two such that M^k is roughly $\frac{N}{\log^2 N}$ and m is roughly $\log N$. The parameter k will be ultimately chosen a constant (see Section 4.4.2). We now explain the details of this encoding process.

4.2.2 Encoding Integers into k-variate Polynomials

Given an N-bit integer a, we first break these N bits into M^k blocks of roughly $\frac{N}{M^k}$ bits each. This corresponds to representing the number a in base $q = 2^{\frac{N}{M^k}}$. Let $a = a_0 + \ldots + a_{M^k-1}q^{M^k-1}$, where every $a_i < q$. The number a is converted into a polynomial as follows:

1. Express i in base M as $i = i_1 + i_2 M + \cdots + i_k M^{k-1}$.

2. Encode each term $a_i q^i$ as the monomial $a_i \cdot X_1^{i_1} X_2^{i_1} \cdots X_k^{i_k}$. As a result, the number a gets converted to the polynomial $\sum_{i=0}^{M^k-1} a_i \cdot X_1^{i_1} \cdots X_k^{i_k}$.

Further, we break each a_i into $\frac{m}{2}$ equal sized blocks where the number of bits in each block is $u = \frac{2N}{M^k \cdot m}$. Each coefficient a_i is then encoded as a polynomial in α of degree less than $\frac{m}{2}$. The polynomials are then padded with zeroes to stretch their degrees to m. Thus, the N-bit number a is converted to a k-variate polynomial $a(X)$ over $\mathbb{Z}[\alpha]/(\alpha^m + 1)$.

Given integers a and b, each of N bits, we encode them as polynomials $a(X)$ and $b(X)$ and compute the product polynomial. The product $a \cdot b$ can be recovered by substituting $X_s = q^{M^{s-1}}$, for $1 \leq s \leq k$, and $\alpha = 2^u$ in the polynomial $a(X) \cdot b(X)$. The coefficients in the product polynomial could be as large as $M^k \cdot m \cdot 2^{2u}$ and hence it is sufficient to do arithmetic modulo p^c where $p^c > M^k \cdot m \cdot 2^{2u}$. Our choice of the prime p ensures that c is in fact a constant (see Section 4.4.2).

4.2.3 Choosing the Prime

The prime p should be chosen such that the ring $\mathbb{Z}/p^c\mathbb{Z}$ has a *principal* $2M$-th root of unity, which is required for polynomial multiplication using the FFT. A principal root of unity is defined as follows.

Definition 4.2.1. (Principal root of unity) *An n-th root of unity $\zeta \in \mathcal{R}$ is said to be primitive if it generates a cyclic group of order n under multiplication. Furthermore, it is said to be principal if n is coprime to the characteristic of $\mathcal{R}$ and ζ satisfies $\sum_{i=0}^{n-1} \zeta^{ij} = 0$ for all $0 < j < n$.*

(Note the similarity of the above definition with Definition 2.1.6. We are using the terminology 'principal' instead of 'primitive' just to be consistent with some of the existing literature.)

In $\mathbb{Z}/p^c\mathbb{Z}$, a $2M$-th root of unity is principal if and only if $2M \mid p-1$ (see also Section 4.5). As a result, we need to choose the prime p from the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$, which is potentially the main bottleneck of our approach. We now explain how to circumvent this bottleneck.

An upper bound for the least prime in an arithmetic progression is given by the following theorem (Lin44):

Theorem 4.2.2. (Linnik) *There exist absolute constants ℓ and L such that for any pair of coprime integers d and n, the least prime p such that $p \equiv d \bmod n$ is less than ℓn^L.*

Heath-Brown (HB92) showed that the *Linnik constant* $L \leq 5.5$ (a recent work by Xylouris (Xyl09) showed that $L \leq 5.2$). Recall that M is chosen such that M^k is $O\left(\frac{N}{\log^2 N}\right)$. If we choose $k = 1$, that is if we use univariate polynomials to encode integers, then the parameter $M = O\left(\frac{N}{\log^2 N}\right)$. Hence the least prime $p \equiv 1 \pmod{2M}$ could be as large as N^L. Since all known deterministic sieving procedures take at least N^L time this is clearly infeasible (for a randomized approach see Section 4.4.2). However, by choosing a larger k we can ensure that the least prime $p \equiv 1 \pmod{2M}$ is $O(N^\varepsilon)$ for some constant $\varepsilon < 1$.

Remark 4.2.3. If k is any integer greater than $L+1$, then $M^L = O\left(N^{\frac{L}{L+1}}\right)$ and hence the least prime $p \equiv 1 \pmod{2M}$ can be found in $o(N)$ time.

4.2.4 Finding the Root of Unity

We require a principal $2M$-th root of unity $\rho(\alpha)$ in $\mathcal{R}$ to compute the Fourier transforms. This root $\rho(\alpha)$ should also have the property that its $\left(\frac{M}{m}\right)$-th power is α, so as to make some multiplications in the FFT efficient (see Section 4.3.2). The root $\rho(\alpha)$ can be computed by interpolation in a way similar to that in Fürer's algorithm (Für07, Section 3), except that we need a principal $2M$-th root of unity ω in $\mathbb{Z}/p^c\mathbb{Z}$ to start with. To obtain such a root, we first obtain a $(p-1)$-th root of unity ζ in $\mathbb{Z}/p^c\mathbb{Z}$ by lifting a generator of $\mathbb{F}_p^\times$. The $\left(\frac{p-1}{2M}\right)$-th power of ζ gives us the required $2M$-th root of unity ω. A generator of $\mathbb{F}_p^\times$ can be computed by brute force, as p is sufficiently small. Having obtained a generator, we use Hensel Lifting.

Lemma 4.2.4 (Lemma 2.3.5 restated)**.** *Let ζ_s be a primitive $(p-1)$-th root of unity in $\mathbb{Z}/p^s\mathbb{Z}$. Then there exists a unique primitive $(p-1)$-th root of unity ζ_{s+1} in $\mathbb{Z}/p^{s+1}\mathbb{Z}$ such that $\zeta_{s+1} \equiv \zeta_s \pmod{p^s}$. This unique root is given by $\zeta_{s+1} = \zeta_s - \frac{f(\zeta_s)}{f'(\zeta_s)}$ where $f(x) = x^{p-1} - 1$.*

It can be shown that the root ζ in $\mathbb{Z}/p^c\mathbb{Z}$ thus obtained by lifting is principal. Furthermore, different powers of ζ are distinct modulo p. Therefore, the difference between any two of them is a unit in $\mathbb{Z}/p^c\mathbb{Z}$ and this makes the following interpolation feasible in our setting.

Finding $\rho(\alpha)$ from ω - Since ω is a principal $2M$-th root of unity, $\gamma = \omega^{\frac{2M}{2m}}$ is a principal $2m$-th root of unity in $\mathbb{Z}/p^c\mathbb{Z}$. Notice that, $\alpha^m + 1$ uniquely factorizes as, $\alpha^m + 1 = (\alpha - \gamma)(\alpha - \gamma^3)\ldots(\alpha - \gamma^{2m-1})$, and the ideals generated by $(\alpha - \gamma^i)$ in $\mathcal{R}$ are mutually coprime as $\gamma^i - \gamma^j$ is a unit for $i \neq j$. Therefore, using Chinese Remaindering, α has the direct sum representation $(\gamma, \gamma^3, \ldots, \gamma^{2m-1})$ in $\mathcal{R}$. Since we require $\rho(\alpha)^{\frac{2M}{2m}} = \alpha$, it is sufficient to choose a $\rho(\alpha)$ whose direct sum representation is $(\omega, \omega^3, \ldots, \omega^{2m-1})$. Now use Lagrange's formula to interpolate $\rho(\alpha)$ as,

$$\rho(\alpha) = \sum_{i=1,\ i \text{ odd}}^{2m-1} \omega^i \cdot \prod_{j=1,\ j\neq i,\ j \text{ odd}}^{2m-1} \frac{\alpha - \gamma^j}{\gamma^i - \gamma^j}$$

The inverses of the elements $\gamma^i - \gamma^j$ can be easily computed in $\mathbb{Z}/p^c\mathbb{Z}$.

4.3 Fourier Transform

4.3.1 Inner and Outer DFT

Suppose that $a(x) \in \mathcal{S}[x]$ is a polynomial of degree less than M, where $\mathcal{S}$ is a ring containing a $2M$-th principal root of unity ρ. Let us say that we want to compute the $2M$-point DFT of $a(x)$ using ρ as the root of unity. In other words, we want to compute the elements $a(1), a(\rho), \ldots, a(\rho^{2M-1})$ is $\mathcal{S}$. This can be done in two steps.

Step 1 - Compute the following polynomials using $\alpha = \rho^{2M/2m}$.

$$\begin{aligned} a_0(x) &= a(x) \mod (x^{2M/2m} - 1) \\ a_1(x) &= a(x) \mod (x^{2M/2m} - \alpha) \\ &\vdots \\ a_{2m-1}(x) &= a(x) \mod (x^{2M/2m} - \alpha^{2m-1}), \end{aligned}$$

where $\deg(a_j(x)) < \frac{2M}{2m}$ for all $0 \leq j < 2m$.

Step 2 - Note that, $a_j(\rho^{k \cdot 2m+j}) = a(\rho^{k \cdot 2m+j})$ for every $0 \leq j < 2m$ and $0 \leq k < \frac{2M}{2m}$. Therefore, all we need to do to compute the DFT of $a(x)$ is to evaluate the polynomials $a_j(x)$ at appropriate powers of ρ.

The idea is to show that both Step 1 and Step 2 can be performed by computation of some 'smaller' DFTs. Let us see how.

Performing Step 1 - The crucial observation here is the following. Fix an integer ℓ in the range $[0, \frac{2M}{2m} - 1]$. Then the ℓ^{th} coefficients of $a_0(x), a_1(x), \ldots, a_{2m-1}(x)$ are exactly $e_\ell(1), e_\ell(\alpha), \ldots, e_\ell(\alpha^{2m-1})$, respectively, where $e_\ell(y)$ is the polynomial,

$$e_\ell(y) = \sum_{j=0}^{2m-1} a_{j \cdot \frac{2M}{2m} + \ell} \cdot y^j.$$

But then, finding $e_\ell(1), e_\ell(\alpha), \ldots, e_\ell(\alpha^{2m-1})$ is essentially computing the $2m$-point DFT of $e_\ell(y)$ using α as the $2m^{th}$ root of unity. Therefore, all we need to do to find $a_0(x), \ldots, a_{2m-1}(x)$ is to compute the DFTs of $e_\ell(y)$ for all $0 \leq \ell < \frac{2M}{2m}$. These $\frac{2M}{2m}$

many $2m$-point DFTs are called the *inner* DFTs.

Performing Step 2 - In order to find $a_j(\rho^{k\cdot 2m+j})$, for $0 \leq k < \frac{2M}{2m}$ and a fixed j, we first compute the polynomial $\tilde{a}_j(x) = a_j(x \cdot \rho^j)$ followed by a $\frac{2M}{2m}$-point DFT of $\tilde{a}_j(x)$ using ρ^{2m} as the root of unity. These $2m$ many $\frac{2M}{2m}$-point DFTs (j running from 0 to $2m-1$) are called the *outer* DFTs. The polynomials $\tilde{a}_j(x)$ can be computed by multiplying the coefficients of $a_j(x)$ by suitable powers of ρ. Such multiplications are termed as *bad* multiplications.

The above discussion is summarized in the following lemma.

Lemma 4.3.1. (DFT time = Inner DFTs + Bad multiplications + Outer DFTs) *Time taken to compute a $2M$-point DFT over $\mathcal{S}$ is sum of:*

1. *Time taken to compute $\frac{2M}{2m}$ many $2m$-point inner DFTs over $\mathcal{S}$ using α as the $2m$-th root of unity.*
2. *Time to do $2M$ multiplications in $\mathcal{S}$ by powers of ρ (bad multiplications).*
3. *Time taken to compute $2m$ many $\frac{2M}{2m}$-point outer DFTs over $\mathcal{S}$ using ρ^{2m} as the $\frac{2M}{2m}$-th root of unity.*

4.3.2 Analysis of the FFT

We are now ready to analyse the complexity of multiplying the two k-variate polynomials $a(X)$ and $b(X)$ (see Section 4.2.2) using the Fast Fourier Transform. Treat $a(X)$ and $b(X)$ as univariate polynomials in variable X_k over the ring $\mathcal{S} = \mathcal{R}[X_1, \ldots, X_{k-1}]$. We write $a(X)$ and $b(X)$ as $a(X_k)$ and $b(X_k)$, respectively, where $\deg(a(X_k))$ and $\deg(b(X_k))$ are less than M. Multiplication of $a(X)$ and $b(X)$ can be thought of as multiplication of the univariates $a(X_k)$ and $b(X_k)$ over $\mathcal{S}$. This makes Algorithm 2 (in Section 2.3.3, Chapter 2) applicable here with parameter $n = 2M$. Also note that, the root $\rho(\alpha)$ (constructed in Section 4.2.4) is a primitive $2M$-th root of unity in $\mathcal{S} \supset \mathcal{R}$. Denote the multiplication complexity of $a(X_k)$ and $b(X_k)$ by $\mathcal{F}(2M, k)$.

Algorithm 2 computes three $2M$-point DFTs over $\mathcal{S}$ and does $2M$ pointwise (or componentwise) multiplications in $\mathcal{S}$. Let $\mathcal{D}(2M, k)$ be the time taken to compute a

$2M$-point DFT over $\mathcal{S}$. By Lemma 4.3.1, the time to compute a DFT is the sum of the time for the inner DFTs, the bad multiplications and the outer DFTs. Let us analyse these three terms separately. We will go by the notation in Section 4.3.1, using $\mathcal{S} = \mathcal{R}[X_1, \ldots, X_{k-1}]$ and $\rho = \rho(\alpha)$.

Inner DFT time - By Lemma 2.3.6, computing a $2m$-point DFT requires $2m \log(2m)$ additions in $\mathcal{S}$ and $m \log(2m)$ multiplications by powers of α. The important observation here is: since $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$, multiplication by a power of α with an element in $\mathcal{R}$ can be readily computed by simple cyclic shifts (with possible negations), which takes only $O(m \cdot \log p)$ bit operations. An element in $\mathcal{S}$ is just a polynomial over $\mathcal{R}$ in variables $X_1, \ldots, X_{k-1}$, with degree in each variable bounded by M. Hence, multiplication by a power of α with an element of $\mathcal{S}$ can be done using $\mathcal{N}_\mathcal{S} = O(M^{k-1} \cdot m \cdot \log p)$ bit operations. A total of $m \log(2m)$ multiplications takes $O(m \log m \cdot \mathcal{N}_\mathcal{S})$ bit operations. It is easy to see that $2m \log(2m)$ additions in $\mathcal{S}$ also require the same order of time.

Since there are $\frac{2M}{2m}$ many $2m$-point DFTs, the total time spent in the inner DFTs is $O(2M \cdot \log m \cdot \mathcal{N}_\mathcal{S})$ bit operations.

Bad multiplication time - Suppose that two arbitrary elements in $\mathcal{R}$ can be multiplied using $\mathcal{M}_\mathcal{R}$ bit operations. Multiplication in $\mathcal{S}$ by a power of ρ amounts to $c_\mathcal{S} = M^{k-1}$ multiplications in $\mathcal{R}$. Since there are $2M$ such bad multiplications, the total time is bounded by $O(2M \cdot c_\mathcal{S} \cdot \mathcal{M}_\mathcal{R})$.

Outer DFT time - By Lemma 4.3.1, the total outer DFT time is $2m \cdot \mathcal{D}\left(\frac{2M}{2m}, k\right)$.

Total DFT time - Therefore, the net DFT time is bounded as,

$$\begin{aligned}
\mathcal{D}(2M, k) &= O\left(2M \cdot \log m \cdot \mathcal{N}_\mathcal{S} + 2M \cdot c_\mathcal{S} \cdot \mathcal{M}_\mathcal{R}\right) + 2m \cdot \mathcal{D}\left(\frac{2M}{2m}, k\right) \\
&= O\left(2M \cdot \log m \cdot \mathcal{N}_\mathcal{S} + 2M \cdot c_\mathcal{S} \cdot \mathcal{M}_\mathcal{R}\right) \cdot \frac{\log 2M}{\log 2m} \\
&= O\left(M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} \cdot \mathcal{M}_\mathcal{R}\right),
\end{aligned}$$

putting the values of $\mathcal{N}_{\mathcal{S}}$ and $c_{\mathcal{S}}$.

Pointwise multiplications - Finally, Algorithm 2 does $2M$ pointwise multiplications in $\mathcal{S}$. Since elements of $\mathcal{S}$ are $(k-1)$-variate polynomials over $\mathcal{R}$, with degree in every variable bounded by M, the total time taken for pointwise multiplications is $2M \cdot \mathcal{F}(2M, k-1)$ bit operations.

Total polynomial multiplication time - This can be expressed as,

$$\begin{aligned}\mathcal{F}(2M,k) &= O\left(M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right) + 2M \cdot \mathcal{F}(2M, k-1) \\ &= O\left(M^k \log M \cdot m \log p + \frac{M^k \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right), \qquad (4.1)\end{aligned}$$

as k is a constant.

We now present an equivalent group theoretic interpretation of the above process of polynomial multiplication which could be of independent interest.

4.3.3 A Group Theoretic Interpretation

A convenient way to study polynomial multiplication is to interpret it as multiplication in a *group algebra*.

Definition 4.3.2. (Group Algebra) *Let G be any group. The* group algebra *of G over a ring R is the set of formal sums $\sum_{g \in G} \alpha_g g$ where $\alpha_g \in R$ with addition defined point-wise and multiplication defined via convolution as follows*

$$\left(\sum_g \alpha_g g\right)\left(\sum_h \beta_h h\right) = \sum_u \left(\sum_{gh=u} \alpha_g \beta_h\right) u$$

Multiplying univariate polynomials over R of degree less than n can be seen as multiplication in the group algebra $R[G]$ where G is the cyclic group of order $2n$. Similarly, multiplying k-variate polynomials of degree less than n in each variable can be seen as multiplying in the group algebra $R[G^k]$, where G^k denotes the k-fold product group $G \times \ldots \times G$.

In this section, we study the Fourier transform over the group algebra $R[E]$ where E is an *additive abelian group*. Most of this, albeit in a different form, is well known but is provided here for completeness (Sha99, Chapter 17).

In order to simplify our presentation, we will fix the base ring to be $\mathbb{C}$, the field of complex numbers. Let n be the *exponent* of E, that is the maximum order of any element in E. A similar approach can be followed for any other base ring as long as it has a principal n-th root of unity.

We consider $\mathbb{C}[E]$ as a vector space with basis $\{x\}_{x\in E}$ and use the Dirac notation to represent elements of $\mathbb{C}[E]$ — the vector $|x\rangle$, x in E, denotes the element $1.x$ of $\mathbb{C}[E]$.

Definition 4.3.3. (Characters) *Let E be an additive abelian group. A* character *of E is a homomorphism from E to $\mathbb{C}^*$.*

An example of a character of E is the trivial character, which we will denote by 1, that assigns to every element of E the complex number 1. If χ_1 and χ_2 are two characters of E then their product $\chi_1.\chi_2$ is defined as $\chi_1.\chi_2(x) = \chi_1(x)\chi_2(x)$.

Proposition 4.3.4. *(Sha99, Chapter 17, Theorem 1) Let E be an additive abelian group of exponent n. Then the values taken by any character of E are n-th roots of unity. Furthermore, the characters form a* multiplicative abelian group $\hat{E}$ *which is isomorphic to E.*

An important property that the characters satisfy is the following (Isa94, Corollary 2.14).

Proposition 4.3.5. (Schur's Orthogonality) *Let E be an additive abelian group. Then*

$$\sum_{x\in E}\chi(x) = \begin{cases} 0 & \text{if } \chi \neq 1, \\ \#E & \text{otherwise} \end{cases} \quad \text{and} \quad \sum_{\chi\in\hat{E}}\chi(x) = \begin{cases} 0 & \text{if } x \neq 0, \\ \#E & \text{otherwise.} \end{cases}$$

It follows from Schur's orthogonality that the collection of vectors $|\chi\rangle = \sum_x \chi(x)\,|x\rangle$ forms a basis of $\mathbb{C}[E]$. We will call this basis the *Fourier basis* of $\mathbb{C}[E]$.

Definition 4.3.6. (Fourier Transform) *Let E be an additive abelian group and let $x \mapsto \chi_x$ be an isomorphism between E and $\hat{E}$. The* Fourier transform *over E is the linear map from $\mathbb{C}[E]$ to $\mathbb{C}[E]$ that sends $|x\rangle$ to $|\chi_x\rangle$.*

Thus, the Fourier transform is a change of basis from the point basis $\{|x\rangle\}_{x\in E}$ to the Fourier basis $\{|\chi_x\rangle\}_{x\in E}$. The Fourier transform is unique only up to the choice of the isomorphism $x \mapsto \chi_x$. This isomorphism is determined by the choice of the principal root of unity.

Remark 4.3.7. Given an element $|f\rangle \in \mathbb{C}[E]$, to compute its Fourier transform it is sufficient to compute the *Fourier coefficients* $\{\langle\chi|f\rangle\}_{\chi\in\hat{E}}$.

Fast Fourier Transform

We now describe the Fast Fourier Transform for general abelian groups in the character theoretic setting. For the rest of the section fix an additive abelian group E over which we would like to compute the Fourier transform. Let A be any subgroup of E and let $B = E/A$. For any such pair of abelian groups A and B, we have an appropriate Fast Fourier transformation, which we describe in the rest of the section.

Proposition 4.3.8. *1. Every character λ of B can be "lifted" to a character of E (which will also be denoted by λ) defined as follows $\lambda(x) = \lambda(x + A)$.*

2. Let χ_1 and χ_2 be two characters of E that when restricted to A are identical. Then $\chi_1 = \chi_2\lambda$ for some character λ of B.

3. The group $\hat{B}$ is (isomorphic to) a subgroup of $\hat{E}$ with the quotient group $\hat{E}/\hat{B}$ being (isomorphic to) $\hat{A}$.

We now consider the task of computing the Fourier transform of an element $|f\rangle = \sum f_x |x\rangle$ presented as a list of coefficients $\{f_x\}$ in the point basis. For this, it is sufficient to compute the Fourier coefficients $\{\langle\chi|f\rangle\}$ for each character χ of E (Remark 4.3.7). To describe the Fast Fourier transform we fix two sets of cosets representatives, one of A in E and one of $\hat{B}$ in $\hat{E}$ as follows.

1. For each $b \in B$, b being a coset of A, fix a coset representative $x_b \in E$ such $b = x_b + A$.

2. For each character φ of A, fix a character χ_φ of E such that χ_φ restricted to A is the character φ. The characters $\{\chi_\varphi\}$ form (can be thought of as) a set of coset representatives of $\hat{B}$ in $\hat{E}$.

Since $\{x_b\}_{b\in B}$ forms a set of coset representatives, any $|f\rangle \in \mathbb{C}[E]$ can be written uniquely as $|f\rangle = \sum f_{b,a}\,|x_b + a\rangle$.

Proposition 4.3.9. *Let $|f\rangle = \sum f_{b,a}\,|x_b + a\rangle$ be an element of $\mathbb{C}[E]$. For each $b \in B$ and $\varphi \in \hat{A}$ let $|f_b\rangle \in \mathbb{C}[A]$ and $|f_\varphi\rangle \in \mathbb{C}[B]$ be defined as follows.*

$$\begin{aligned} |f_b\rangle &= \sum_{a\in A} f_{b,a}\,|a\rangle \\ |f_\varphi\rangle &= \sum_{b\in B} \overline{\chi}_\varphi(x_b)\langle\varphi|f_b\rangle\,|b\rangle \end{aligned}$$

Then for any character $\chi = \chi_\varphi\lambda$ of E the Fourier coefficient $\langle\chi|f\rangle = \langle\lambda|f_\varphi\rangle$.

We are now ready to describe the Fast Fourier transform given an element $|f\rangle = \sum f_x\,|x\rangle$.

1. For each $b \in B$ compute the Fourier transforms of $|f_b\rangle$. This requires $\#B$ many Fourier transforms over A.

2. As a result of the previous step we have for each $b \in B$ and $\varphi \in \hat{A}$ the Fourier coefficients $\langle\varphi|f_b\rangle$. Compute for each φ the vectors $|f_\varphi\rangle = \sum_{b\in B} \overline{\chi}_\varphi(x_b)\langle\varphi|f_b\rangle\,|b\rangle$. This requires $\#\hat{A}.\#B = \#E$ many multiplications by roots of unity.

3. For each $\varphi \in \hat{A}$ compute the Fourier transform of $|f_\varphi\rangle$. This requires $\#\hat{A} = \#A$ many Fourier transforms over B.

4. Any character χ of E is of the form $\chi_\varphi\lambda$ for some $\varphi \in \hat{A}$ and $\lambda \in \hat{B}$. Using Proposition 4.3.9 we have at the end of Step 3 all the Fourier coefficients $\langle\chi|f\rangle = \langle\lambda|f_\varphi\rangle$.

If the quotient group B itself has a subgroup that is isomorphic to A then we can apply this process recursively on B to obtain a divide and conquer procedure to compute Fourier transform. In the standard FFT we use $E = \mathbb{Z}/2^n\mathbb{Z}$. The subgroup A is $2^{n-1}E$ which is isomorphic to $\mathbb{Z}/2\mathbb{Z}$ and the quotient group B is $\mathbb{Z}/2^{n-1}\mathbb{Z}$.

Analysis of the Fourier Transform

Our goal is to multiply k-variate polynomials over $\mathcal{R}$, with the degree in each variable less than M. This can be achieved by embedding the polynomials into the algebra of the product group $E = \left(\frac{\mathbb{Z}}{2M \cdot \mathbb{Z}}\right)^k$ and multiplying them as elements of the algebra. Since the exponent of E is $2M$, we require a principal $2M$-th root of unity in the ring $\mathcal{R}$. We shall use the root $\rho(\alpha)$ (as defined in Section 4.2.4) for the Fourier transform over E.

For every subgroup A of E, we have a corresponding FFT. We choose the subgroup A as $\left(\frac{\mathbb{Z}}{2m \cdot \mathbb{Z}}\right)^k$ and let B be the quotient group E/A. The group A has exponent $2m$ and α is a principal $2m$-th root of unity. Since α is a power of $\rho(\alpha)$, we can use it for the Fourier transform over A. As multiplications by powers of α are just shifts, this makes Fourier transform over A efficient.

Let $\mathcal{F}(M, k)$ denote the complexity of computing the Fourier transform over $\left(\frac{\mathbb{Z}}{2M \cdot \mathbb{Z}}\right)^k$. We have

$$\mathcal{F}(M,k) = \left(\frac{M}{m}\right)^k \mathcal{F}(m,k) + M^k \mathcal{M}_{\mathcal{R}} + (2m)^k \mathcal{F}\left(\frac{M}{2m}, k\right) \tag{4.2}$$

where $\mathcal{M}_{\mathcal{R}}$ denotes the complexity of multiplications in $\mathcal{R}$. The first term comes from the $\#B$ many Fourier transforms over A (Step 1 of FFT), the second term corresponds to the multiplications by roots of unity (Step 2) and the last term comes from the $\#A$ many Fourier transforms over B (Step 3).

Since A is a subgroup of B as well, Fourier transforms over B can be recursively computed in a similar way, with B playing the role of E. Therefore, by simplifying the recurrence in Equation 4.2 we get:

$$\mathcal{F}(M,k) = O\left(\frac{M^k \log M}{m^k \log m}\mathcal{F}(m,k) + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right) \tag{4.3}$$

Lemma 4.3.10. $\mathcal{F}(m,k) = O(m^{k+1} \log m \cdot \log p)$

Proof. The FFT over a group of size n is usually done by taking 2-point FFT's followed by $\frac{n}{2}$-point FFT's. This involves $O(n \log n)$ multiplications by roots of unity and additions in base ring. Using this method, Fourier transforms over A can be computed with $O(m^k \log m)$ multiplications and additions in $\mathcal{R}$. Since each multiplication is between an element of $\mathcal{R}$ and a power of α, this can be efficiently

achieved through shifting operations. This is dominated by the addition operation, which takes $O(m \log p)$ time, since this involves adding m coefficients from $\mathbb{Z}/p^c\mathbb{Z}$. □

Therefore, from Equation 4.3,

$$\mathcal{F}(M,k) = O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m}\mathcal{M}_{\mathcal{R}}\right).$$

4.4 Algorithm and Analysis

4.4.1 Integer Multiplication Algorithm

We are given two integers $a, b < 2^N$ to multiply. We fix constants k and c whose values are given in Section 4.4.2. The algorithm is as follows:

1. Choose M and m as powers of 2 such that $M^k \approx \frac{N}{\log^2 N}$ and $m \approx \log N$. Find the least prime $p \equiv 1 \pmod{2M}$ (Remark 4.2.3).

2. Encode the integers a and b as k-variate polynomials $a(X)$ and $b(X)$, respectively, over the ring $\mathcal{R} = \mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$ (Section 4.2.2).

3. Compute the root $\rho(\alpha)$ (Section 4.2.4).

4. Use $\rho(\alpha)$ as the principal $2M$-th root of unity to compute the Fourier transforms of the k-variate polynomials $a(X)$ and $b(X)$. Multiply component-wise and take the inverse Fourier transform to obtain the product polynomial. (Sections 4.3.1 and 4.3.2)

5. Evaluate the product polynomial at appropriate powers of two to recover the integer product and return it (Section 4.2.2).

4.4.2 Complexity Analysis

The choice of parameters should ensure that the following constraints are satisfied:

1. $M^k = O\left(\frac{N}{\log^2 N}\right)$ and $m = O(\log N)$.

2. $M^L = O(N^\varepsilon)$, where L is the Linnik constant (Theorem 4.2.2) and ε is any constant less than 1. Recall that this makes picking the prime by brute force feasible (see Remark 4.2.3).

3. $p^c > M^k \cdot m \cdot 2^{2u}$ where $u = \frac{2N}{M^k m}$. This is to prevent overflows during modular arithmetic (see Section 4.2.2).

It is straightforward to check that $k > L + 1$ and $c > 5(k + 1)$ satisfy the above constraints. Since $L \leq 5.2$, it is sufficient to choose $k = 7$ and $c = 42$.

Let $T(N)$ denote the time complexity of multiplying two N bit integers. This consists of:

- Time required to pick a suitable prime p,
- Computing the root $\rho(\alpha)$,
- Encoding the input integers as polynomials,
- Multiplying the encoded polynomials,
- Evaluating the product polynomial.

As argued before, the prime p can be chosen in $o(N)$ time. To compute $\rho(\alpha)$, we need to lift a generator of $\mathbb{F}_p^\times$ to $\mathbb{Z}/p^c\mathbb{Z}$ followed by an interpolation. Since c is a constant and p is a prime of $O(\log N)$ bits, the time required for Hensel Lifting and interpolation is $o(N)$.

The encoding involves dividing bits into smaller blocks, and expressing the exponents of q in base M (Section 4.2.2) and all these take $O(N)$ time since M is a power of 2. Similarly, evaluation of the product polynomial takes linear time as well. Therefore, the time complexity is dominated by the time taken for polynomial multiplication.

Time complexity of Polynomial Multiplication

From Equation 4.1, the complexity of polynomial multiplication is given by,

$$\mathcal{F}(2M,k) = O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right).$$

Proposition 4.4.1. *(Sch82) Multiplication in $\mathcal{R}$ reduces to multiplying $O(\log^2 N)$ bit integers and hence $\mathcal{M}_{\mathcal{R}} = T\left(O(\log^2 N)\right)$.*

Proof. Elements of $\mathcal{R}$ can be viewed as polynomials in α over $\mathbb{Z}/p^c\mathbb{Z}$ with degree at most m. Given two such polynomials $f(\alpha)$ and $g(\alpha)$, encode them as follows: Replace α by 2^d, transforming the polynomials $f(\alpha)$ and $g(\alpha)$ to the integers $f(2^d)$ and $g(2^d)$ respectively. The parameter d is chosen such that the coefficients of the product $h(\alpha) = f(\alpha)g(\alpha)$ can be recovered from the product $f(2^d) \cdot g(2^d)$. For this, it is sufficient to ensure that the maximum coefficient of $h(\alpha)$ is less than 2^d. Since f and g are polynomials of degree m, we would want 2^d to be greater than $m \cdot p^{2c}$, which can be ensured by choosing $d = O(\log N)$. The integers $f(2^d)$ and $g(2^d)$ are bounded by 2^{md} and hence the task of multiplying in $\mathcal{R}$ reduces to $O(\log^2 N)$-bit integer multiplication. □

Therefore, the complexity of our integer multiplication algorithm $T(N)$ is given by,

$$\begin{aligned} T(N) &= O(\mathcal{F}(2M,k)) = O\left(M^k \log M \cdot m \cdot \log p + \frac{M^k \log M}{\log m} \cdot \mathcal{M}_{\mathcal{R}}\right) \\ &= O\left(N \log N + \frac{N}{\log N \cdot \log\log N} \cdot T(O(\log^2 N))\right) \end{aligned}$$

Solving the above recurrence leads to the following theorem.

Theorem 4.4.2. *Given two N bit integers, their product can be computed using $N \cdot \log N \cdot 2^{O(\log^* N)}$ bit operations.*

Choosing the Prime Randomly

To ensure that the search for a prime $p \equiv 1 \pmod{2M}$ does not affect the overall time complexity of the algorithm, we have considered multivariate polynomials to restrict the value of M; an alternative is to use randomization.

Proposition 4.4.3. *Assuming the ERH, a prime $p \equiv 1 \pmod{2M}$ can be computed by a randomized algorithm with expected running time $\tilde{O}(\log^3 M)$.*

Proof. Titchmarsh (Tit30) (see also Tianxin (Tia90)) showed that, assuming the ERH, that the number of primes less than x in the arithmetic progression $\{1 + i \cdot 2M\}_{i>0}$ is given by,

$$\pi(x, 2M) = \frac{Li(x)}{\varphi(2M)} + O(\sqrt{x} \log x)$$

for $2M \leq \sqrt{x} \cdot (\log x)^{-2}$, where $Li(x) = \Theta(\frac{x}{\log x})$ and φ is the Euler totient function. In our case, since M is a power of two, $\varphi(2M) = M$, and hence for $x \geq 4M^2 \cdot \log^6 M$, we have $\pi(x, 2M) = \Omega\left(\frac{x}{M \log x}\right)$. Therefore, for an i chosen uniformly randomly in the range $1 \leq i \leq 2M \cdot \log^6 M$, the probability that $i \cdot 2M + 1$ is a prime is at least $\frac{d}{\log x}$ for a constant d. Furthermore, primality test of an $O(\log M)$ bit number can be done in $\tilde{O}(\log^2 M)$ time using Rabin-Miller primality test (Mil76; Rab80). Hence, with $x = 4M^2 \cdot \log^6 M$ a suitable prime for our algorithm can be found in expected $\tilde{O}(\log^3 M)$ time. □

4.5 A Different Perspective

Our algorithm can be seen as a p-adic version of Fürer's integer multiplication algorithm, where the field $\mathbb{C}$ is replaced by $\mathbb{Q}_p$, the field of p-adic numbers (for a quick introduction, see Baker's online notes (Bak07)). Much like $\mathbb{C}$, where representing a general element (say in base 2) takes infinitely many bits, representing an element in $\mathbb{Q}_p$ takes infinitely many p-adic digits. Since we cannot work with infinitely many digits, all arithmetic has to be done with finite precision. Modular arithmetic in the base ring $\mathbb{Z}[\alpha]/(p^c, \alpha^m + 1)$, can be viewed as arithmetic in the ring $\mathbb{Q}_p[\alpha]/(\alpha^m + 1)$ keeping a precision of $\varepsilon = p^{-c}$.

Arithmetic with finite precision naturally introduces some errors in computation. However, the nature of $\mathbb{Q}_p$ makes the error analysis simpler. The field $\mathbb{Q}_p$ comes with a norm $|\cdot|_p$ called the p-adic norm, which satisfies the stronger triangle inequality $|x + y|_p \leq \max\left(|x|_p, |y|_p\right)$ (Bak07, Proposition 2.6). As a result, unlike in $\mathbb{C}$, the errors in computation do not compound.

Recall that the efficiency of FFT crucially depends on a special principal $2M$-th root of unity in $\mathbb{Q}_p[\alpha]/(\alpha^m + 1)$. Such a root is constructed with the help of a primitive $2M$-th root of unity in $\mathbb{Q}_p$. The field $\mathbb{Q}_p$ has a primitive $2M$-th root of unity if and only if $2M$ divides $p - 1$ (Bak07, Theorem 5.12). Also, if $2M$ divides $p - 1$, a $2M$-th root can be obtained from a $(p - 1)$-th root of unity by taking a suitable power. A primitive $(p - 1)$-th root of unity in $\mathbb{Q}_p$ can be constructed, to sufficient precision, using Hensel Lifting starting from a generator of $\mathbb{F}_p^\times$.

4.6 Conclusion

As mentioned earlier, there has been two approaches to multiplying integers - one uses arithmetic over complex numbers while the other uses modular arithmetic. Using complex numbers, Schönhage and Strassen (SS71) gave an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ algorithm. Fürer (Für07) improved this complexity to $N \cdot \log N \cdot 2^{O(\log^* N)}$ using some special roots of unity. The other approach, that is modular arithmetic, can be seen as arithmetic in $\mathbb{Q}_p$ with certain precision. A direct adaptation of the Schönhage-Strassen's algorithm in the modular setting leads to an $O(N \cdot \log N \cdot \log\log N \ldots 2^{O(\log^* N)})$ time algorithm. In this work, we show that by choosing an appropriate prime and a special root of unity, a running time of $N \cdot \log N \cdot 2^{O(\log^* N)}$ can be achieved through modular arithmetic as well. Therefore, in a way, we have unified the two paradigms. The important question that remains open is:

- Can N-bit integers be multiplied using $O(N \cdot \log N)$ bit operations?

Even an improvement of the complexity to $O(N \cdot \log N \cdot \log^* N)$ operations will be a significant step forward towards answering this question.

Chapter 5

Identity Testing: Depth 2 Circuits over Algebras

-Joint work with Ramprasad Saptharishi and Nitin Saxena.

5.1 Introduction

Quite often in high school algebra we come across polynomial identities like, $x^3+y^3+z^3-3xyz = (x+y+z)(x+\omega y+\omega^2 z)(x+\omega^2 y+\omega z)$ and $x^3-y^3 = (x-y)(x-\omega y)(x-\omega^2 y)$, where ω is a complex cube root of unity. Such identities equate two 'different looking' polynomials, thereby revealing that they are just different expressions of the same polynomial. It is natural to ask, given two polynomial expressions $f(X)$ and $g(X)$, where X is the set of variables $\{x_1, \ldots, x_n\}$, how fast can we check if $f(X) = g(X)$? Stated differently, the problem is to find out if $f(X) - g(X)$ is identically zero. It is this problem that is known as Polynomial Identity Testing (PIT, for short), when the polynomial expressions are given in the concise form of arithmetic circuits (see Section 2.1.2). Recalling the formal definition from Section 1.1, PIT is the following problem.

Problem 5.1.1. (Polynomial Identity Testing) *Given an arithmetic circuit C with input variables $x_1, \ldots, x_n$ and constants taken from a field $\mathbb{F}$, check if the polynomial $f(x_1, \ldots, x_n)$ computed by C is identically zero.*

This natural algebraic problem finds widespread applications in complexity theory and general algorithm design. Some of the high profile results in complexity theory, like IP = PSPACE (LFKN90; Sha90) and the PCP theorem (ALM+98), involve identity testing. More interestingly, it has been shown that (KI03; Agr05) if certain efficient derandomization of PIT is possible then the class VNP (as defined by Valiant (Val79)) is different from the class VP. This can be viewed as proving the arithmetic analogue of the much coveted result, P $\neq$ NP. It is perhaps not an exaggeration to say that identity testing came into the limelight after a remarkable derandomization of primality testing, by Agrawal, Kayal and Saxena (AKS04), that is based on identity testing over a certain ring. Several other algorithms like graph matching (Lov79), multivariate polynomial interpolation (CDGK91) also use identity testing.

The idea of randomly choosing a point and evaluating the polynomial at it to see if it is zero, was first proposed by Schwartz (Sch80) and Zippel (Zip79), and this naturally gave a randomized polynomial time algorithm. Several other efficient randomized algorithms (CK97; LV98; AB99; KS01) came up subsequently, resulting in a significant improvement in the number of random bits used. However, so far identity testing is yet to be derandomized even for the case of depth 3 arithmetic circuits (the problem is trivial for depth 2 and depth 1 circuits). It is this problem of depth 3 identity testing that motivated our work.

Assume that a circuit C has alternate layers of addition and multiplication gates. A layer of addition gates is denoted by Σ and that of multiplication gates is denoted by Π. Kayal and Saxena (KS07) gave a deterministic polynomial time identity testing algorithm for depth 3 ($\Sigma\Pi\Sigma$) circuits with constant top fan-in. This case of identity testing was subsequently made *blackbox* by a series of interesting work by Kayal and Saraf (KS09), and Saxena and Seshadhri (SS09; SS10b; SS10a). A justification behind the hardness of PIT even for small depth circuits was provided by Agrawal and Vinay (AV08). They showed that a deterministic black box identity test for depth 4 ($\Sigma\Pi\Sigma\Pi$) circuits implies a quasi-polynomial time deterministic PIT algorithm for circuits computing polynomials of *low* degree. A polynomial is said to have low degree if its degree is less than the size of the circuit that computes it.

Thus, the non-trivial case for identity testing starts with depth 3 circuits; whereas circuits of depth 4 are *almost* the general case. At this point, it is natural to ask as to what is the complexity of the PIT problem for depth 2 ($\Pi\Sigma$) circuits if we allow the *constants* of the circuit to come from an algebra $\mathcal{R}$ that has dimension over $\mathbb{F}$, $\dim_{\mathbb{F}}(\mathcal{R}) > 1$. Can we relate this problem to the classical PIT problem for depth 3 and depth 4 circuits? In this work, we address and answer this question. We assume that the algebra $\mathcal{R}$ is given in *basis form* i.e. we know an $\mathbb{F}$-basis $\{e_1, \ldots, e_k\}$ of $\mathcal{R}$ and we also know how $e_i e_j$ can be expressed in terms of the basis elements, for all i and j. The problem at hand is the following,

Problem 5.1.2. *Given an expression,*

$$P = \prod_{i=1}^{d} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n)$$

where $A_{ij} \in \mathcal{R}$, an algebra over $\mathbb{F}$ given in basis form, check if P is zero.

How hard is the above problem? At first sight, this problem might look deceptively simple. For instance, if $\mathcal{R}$ is a field or even a division algebra (say, the real quaternion algebra) then it is trivial to check if $P = 0$ using polynomial number of $\mathbb{F}$-operations. However, in general this is far from what might be the case.

Since elements of a finite dimensional algebra, given in basis form, can be expressed as matrices over $\mathbb{F}$ we can equivalently write the above problem as,

Problem 5.1.3. *Given an expression,*

$$P = \prod_{i=1}^{d} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n) \tag{5.1}$$

where $A_{ij} \in \mathsf{M}_k(\mathbb{F})$, the algebra of $k \times k$ matrices over $\mathbb{F}$, check if P is zero using $\mathsf{poly}(k \cdot n \cdot d)$ many $\mathbb{F}$-operations.

It is quite easy to verify that if we allow randomness then it is solvable just like the usual PIT problem (using Schwartz-Zippel test (Sch80; Zip79)). We are only interested in deterministic methods in this work. To avoid confusion we use the

following convention:

Convention - Whenever we say 'arithmetic circuit (or formula)' without an extra qualification, we mean a circuit (or formula) over a field. Otherwise, we explicitly mention 'arithmetic circuit (or formula) over *some* algebra' to mean that the constants of the circuit are taken from 'that' algebra. Also, by depth 3 and depth 2 circuits, we always mean $\Sigma\Pi\Sigma$ and $\Pi\Sigma$ circuits respectively.

5.2 The Depth 2 Model

5.2.1 Depth 2 Circuits over Matrices

A depth 2 circuit C over matrices, as in Equation 5.1, naturally defines a computational model. Assuming $\mathcal{R} = \mathsf{M}_k(\mathbb{F})$, for some k, a polynomial $P \in \mathcal{R}[x_1, \ldots, x_n]$ outputted by C can be viewed as a $k \times k$ matrix of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. We say that a polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ is *computed* by C if one of the k^2 polynomials in matrix P is f. Sometimes we say P *computes* f to mean the same. The following is an example of a depth 2 circuit over $\mathsf{M}_2(\mathbb{F})$.

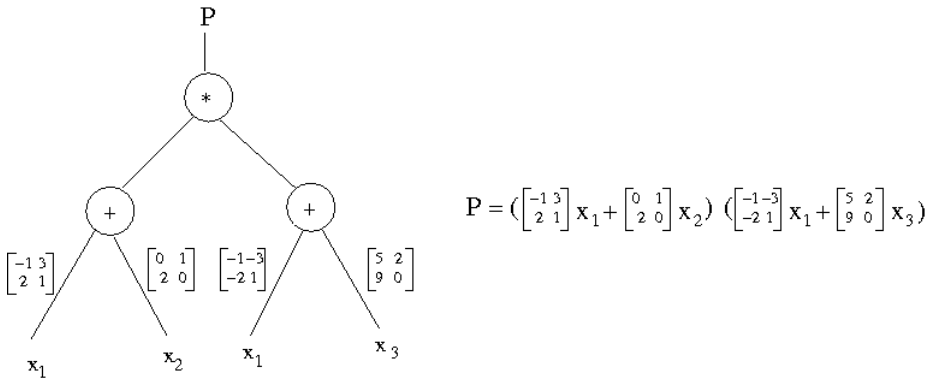

Figure 5.1: A depth-2 circuit over 2×2 matrices.

In the following discussion, we denote the algebra of upper-triangular $k \times k$ matrices by $\mathsf{U}_k(\mathbb{F})$. The algebra $\mathsf{U}_2(\mathbb{F})$ is the *smallest* non-commutative algebra with unity over $\mathbb{F}$, in the sense that $\dim_{\mathbb{F}} \mathsf{U}_2(\mathbb{F}) = 3$ and any algebra of smaller dimension is commutative. We show here that already $\mathsf{U}_2(\mathbb{F})$ captures an open case of identity

testing.

Ben-Or and Cleve (BC88) showed that a polynomial computed by an arithmetic formula E of depth d, and with fan-in (of every gate) bounded by 2, can also be computed by a straight-line program of length at most 4^d using only 3 registers. The following fact can be readily derived from their result (see Appendix A.2): From an arithmetic formula E of depth d and fan-in bounded by 2, we can efficiently compute the expression,

$$P = \prod_{i=1}^{m} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n)$$

where $m \leq 4^d$ and $A_{ij} \in \mathsf{M}_3(\mathbb{F})$ such that P *computes* the polynomial that E does. Thus solving Problem 5.1.3 in polynomial time even for 3×3 matrices yields a polynomial time algorithm for PIT of constant depth circuits, in particular depth 4 circuits. There is an alternative way of arguing that the choice of $\mathcal{R}$ as $\mathsf{M}_3(\mathbb{F})$ is *almost* the general case.

Given an arithmetic circuit of size s, computing a low degree polynomial, Allender, Jiao, Mahajan and Vinay (AJMV98) (see also (VSBR83)) showed how to efficiently construct an equivalent circuit of size $s^{O(1)}$ and depth $O(\log s)$. Such a circuit can be readily converted to a bounded fan-in formula of size $s^{O(\log s)}$ and depth $O(\log^2 s)$. From this formula one can obtain a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ of size $4^{O(\log^2 s)} = s^{O(\log s)}$ (using Ben-Or and Cleve's result) that computes the same polynomial as the formula. Thus, derandomization of PIT for depth 2 circuits over 3×3 matrices yields a quasi-polynomial time PIT algorithm for any circuit computing a low degree polynomial. This means, in essence a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ plays the role of a depth 4 circuit over $\mathbb{F}$ (in the spirit of Agrawal and Vinay's result).

It is natural to ask how the complexity of PIT for depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ relates to PIT for arithmetic circuits. In this work, we provide an answer to this. We show a surprising connection between PIT of depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ and PIT of depth 3 circuits. The reason this is surprising is because we also show that, a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ is not even powerful enough to compute a simple polynomial like, $x_1x_2 + x_3x_4 + x_5x_6$!

5.2.2 Known Related Models

Identity testing and circuit lower bounds have been studied for different algebraic models. Nisan (Nis91) showed an exponential lower bound on the size of any arithmetic formula computing the determinant of a matrix in the non-commutative *free algebra* model. The result was generalized by Chien and Sinclair (CS04) to a large class of non-commutative algebras satisfying polynomial identities, called PI-algebras. Identity testing has also been studied for the non-commutative model by Raz and Shpilka (RS04), Bogdanov and Wee (BW05), and Arvind, Mukhopadhyay and Srinivasan (AMS08). But unlike those models where the variables do not commute, in our setting the variables always commute but the *constant coefficients* are taken from an algebra $\mathcal{R}$. The motivation for studying this latter model (besides it being a natural generalization of circuits over fields) is that, it provides a different perspective to the complexity of the classical PIT problem in terms of the dimension of the underlying algebra. It seems to 'pack' the combinatorial nature of the circuit into a larger base algebra and hence opens up the possibility of using algebra structure results. The simplest nontrivial circuit in this model is a $\Pi\Sigma$ circuit over the non-commutative algebra $\mathcal{R} = \mathsf{U}_2(\mathbb{F})$, and even this, as we show, represents the frontier of our understanding.

5.2.3 Our Results

The results we present are of two types. Some are related to identity testing while the rest are related to the weakness of the depth 2 computational model over $\mathsf{U}_2(\mathbb{F})$ and $\mathsf{M}_2(\mathbb{F})$.

Identity testing

We show the following result.

Theorem 5.2.1. *Identity testing for depth* 3 ($\Sigma\Pi\Sigma$) *circuits is polynomial time equivalent to identity testing for depth* 2 ($\Pi\Sigma$) *circuits over* $\mathsf{U}_2(\mathbb{F})$.

The above theorem has an interesting consequence on identity testing for Algebraic Branching Program (ABP) (Nis91) (see Definition 5.3.2). It is known that identity testing for non-commutative ABP can be done in deterministic polynomial time (a

result due to Raz and Shpilka (RS04)). But no progress is made on identity testing of even width-2 commutative ABP. The following result justifies this.

Corollary 5.2.2. *Identity testing of depth* 3 *circuits* ($\Sigma\Pi\Sigma$) *reduces to identity testing of width-*2 *ABPs.*

We have mentioned before about the prospect of using structural results to solve PIT for depth 2 circuits over algebras. Our next result shows this idea at work for commutative algebras.

Theorem 5.2.3. *Given an expression,*

$$P = \prod_{i=1}^{d} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n),$$

where $A_{ij} \in \mathcal{R}$*, a commutative algebra of dimension* k *over* $\mathbb{F}$*, there is a deterministic algorithm to test if* P *is zero running in time* $\mathsf{poly}\left(k^k, n, d\right)$*.*

The above result gives a polynomial time algorithm for $k = O\left(\log nd / \log\log nd\right)$. This result establishes that the power of depth 2 circuits over *small* algebras is primarily derived from the non-commutative nature of the algebra. However, we show that commutative algebras of polynomial dimension over $\mathbb{F}$ are much more powerful.

Theorem 5.2.4. *Identity testing of a depth* 3 ($\Sigma\Pi\Sigma$) *circuit of size* s *reduces to identity testing of a depth* 2 ($\Pi\Sigma$) *circuit of size* $O(s)$ *over a commutative algebra of dimension* $O(s)$*.*

Our argument for proving Theorem 5.2.1 is relatively simple in nature. Perhaps the reason why such a connection was overlooked before is that, unlike a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$, we do not have the privilege of *exactly* computing a polynomial over $\mathbb{F}$ using a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$. Showing this weakness of the latter computational model constitutes the second part of our results.

Weakness of the depth 2 model over $\mathsf{U}_2(\mathbb{F})$ and $\mathsf{M}_2(\mathbb{F})$

We show that depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ are computationally weaker than depth 3 circuits.

Theorem 5.2.5. *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial such that there are no two linear functions l_1 and l_2 (with $1 \notin (l_1, l_2)$, the ideal generated by l_1 and l_2) which make $f \bmod (l_1, l_2)$ also a linear function. Then f is not computable by a depth 2 ($\Pi\Sigma$) circuit over $\mathsf{U}_2(\mathbb{F})$.*

Even a simple polynomial like $x_1x_2 + x_3x_4 + x_5x_6$ satisfies the condition stated in the above theorem, and so it is not computable by any depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$, no matter how large! This contrast makes Theorem 5.2.1 surprising as it establishes an equivalence of identity testing in two models of different computational strengths. We further show that the computational power of depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ is also severely restrictive. Let P_ℓ denote the partial product $P_\ell = \prod_{i=\ell}^{d} \sum_{j=0}^{n} A_{ij}x_j$, where $A_{ij} \in \mathsf{M}_2(\mathbb{F})$, $x_0 = 1$ and $1 \leq \ell \leq d$.

Definition 5.2.6. *A polynomial f is computed by a depth 2 circuit* ($\Pi\Sigma$) under a degree restriction *of m if the degree of every partial product P_ℓ is bounded by m, for $1 \leq \ell \leq d$.*

Theorem 5.2.7. *There exists a class of polynomials over $\mathbb{F}$ of degree n that cannot be computed by a depth 2 ($\Pi\Sigma$) circuit over $\mathsf{M}_2(\mathbb{F})$, under a degree restriction of n.*

The motivation behind imposing a condition like *degree restriction* comes naturally from depth 2 circuits over $\mathsf{M}_3(\mathbb{F})$. Given a polynomial $f = \sum_i m_i$, where m_i's are the monomials of f, it is easy to construct a depth 2 circuit over $\mathsf{M}_3(\mathbb{F})$ that literally forms these monomials and adds them up one by one. This computation is degree restricted, if we extend our definition of degree restriction to $\mathsf{M}_3(\mathbb{F})$. However, the above theorem shows that this simple scheme fails over $\mathsf{M}_2(\mathbb{F})$.

5.3 Identity Testing over $\mathsf{M}_2(\mathbb{F})$

In this section, we show that PIT of depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ is at least as hard as PIT of depth 3 circuits. This further implies that PIT of a width-2 commutative ABP is also 'harder' than the latter problem.

5.3.1 Equivalence with Depth 3 Identity Testing

Given a depth 3 circuit, assume (without loss of generality) that the fan-in of the multiplication gates are the same. This multiplicative fan-in is referred to as the *degree* of the depth 3 circuit. For convenience, we call a matrix with linear functions as entries, a *linear* matrix.

Lemma 5.3.1. *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial computed by a depth 3 circuit C of degree d and top fan-in s. Given circuit C, it is possible to construct in polynomial time a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ of size $O((d+n)s^2)$ that computes a polynomial $L \cdot f$, where L is a product of non-zero linear functions.*

Proof. A depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ is simply a product sequence of 2×2 upper-triangular linear matrices. We show that there exists such a sequence of length $O((d+n)s^2)$ such that the product 2×2 matrix has $L \cdot f$ as one of its entries.

Since f is computed by a depth 3 circuit, we can write $f = \sum_{i=1}^{s} P_i$, where each summand $P_i = \prod_j l_{ij}$ is a product of linear functions. Observe that a single P_i can be computed using the following product sequence of length d.

$$\begin{bmatrix} l_{i1} & \\ & 1 \end{bmatrix} \cdots \begin{bmatrix} l_{i(d-1)} & \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & l_{id} \\ & 1 \end{bmatrix} = \begin{bmatrix} L' & P_i \\ & 1 \end{bmatrix}, \tag{5.2}$$

where $L' = l_{i1} \cdots l_{i(d-1)}$.

Each matrix of the form $\begin{bmatrix} 1 & l \\ & 1 \end{bmatrix}$, where $l = a_0 + \sum a_i x_i$, can be further expanded as,

$$\begin{bmatrix} 1 & a_0 \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & a_1 x_1 \\ & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & a_n x_n \\ & 1 \end{bmatrix} = \begin{bmatrix} 1 & l \\ & 1 \end{bmatrix}$$

These will be the only type of non-diagonal matrices that would appear in the sequence.

The proof proceeds through induction, where Equation 5.2 serves as the induction basis. A generic intermediate matrix looks like $\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix}$, where each L_i is a product of non-zero linear functions and g is a partial sum of the P_i's. We will inductively double the number of summands in g as follows.

At the i-th iteration, let us assume that we have the matrices $\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix}$ and $\begin{bmatrix} M_1 & M_2 h \\ & M_3 \end{bmatrix}$, each computed by a sequence of n_i linear matrices. We want a sequence that computes a polynomial of the form $L \cdot (g+h)$. Consider the following sequence,

$$\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix} \begin{bmatrix} A & \\ & B \end{bmatrix} \begin{bmatrix} M_1 & M_2 h \\ & M_3 \end{bmatrix} = \begin{bmatrix} AL_1M_1 & AL_1M_2h + BL_2M_3g \\ & BL_3M_3 \end{bmatrix}, \tag{5.3}$$

where A, B are products of linear functions. By setting $A = L_2M_3$ and $B = L_1M_2$ we get the desired sequence,

$$\begin{bmatrix} L_1 & L_2 g \\ & L_3 \end{bmatrix} \begin{bmatrix} A & \\ & B \end{bmatrix} \begin{bmatrix} M_1 & M_2 h \\ & M_3 \end{bmatrix} = \begin{bmatrix} L_1L_2M_1M_3 & L_1L_2M_2M_3(g+h) \\ & L_1L_3M_2M_3 \end{bmatrix}.$$

This way, we have doubled the number of summands in $g + h$. By induction, the length of the sequence computing L_2g and M_2h is n_i, and each L_i and M_i is a product of n_i many linear functions. Therefore, both A and B are products of at most $2n_i$ linear functions and the matrix $\begin{bmatrix} A & \\ & B \end{bmatrix}$ can be written as a product of at most $2n_i$ diagonal linear matrices. The total length of the sequence given in Equation 5.3 is hence bounded by $4n_i$.

As the number of summands in f is s, the above process of doubling the summands needs to be repeated at most $\log s + 1$ times. The final sequence length is hence bounded by $(d+n) \cdot 4^{\log s} = (d+n)s^2$. □

Proof of Theorem 5.2.1: It follows from Lemma 5.3.1 that given a depth 3 circuit C computing f we can efficiently construct a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ that outputs a matrix, $\begin{bmatrix} L_1 & L \cdot f \\ & L_2 \end{bmatrix}$, where L is a product of non-zero linear functions. Multiplying this matrix by $\begin{bmatrix} 1 & 0 \\ & 0 \end{bmatrix}$ to the left and $\begin{bmatrix} 0 & 0 \\ & 1 \end{bmatrix}$ to the right yields another depth 2 circuit D that outputs $\begin{bmatrix} 0 & L \cdot f \\ & 0 \end{bmatrix}$. Thus D computes an identically zero polynomial over $\mathsf{U}_2(\mathbb{F})$ if and only if C computes an identically zero polynomial.

This shows that PIT for depth 3 circuits reduces to PIT of depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$.

The other direction, that is PIT for depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ reduces to PIT for depth 3 circuits, is trivial to observe. The diagonal entries of the output 2×2 matrix is just a product of linear functions whereas the off-diagonal entry is a sum of at most d' many products of linear functions, where d' is the multiplicative fan-in of the depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$. □

5.3.2 Width-2 Algebraic Branching Programs

Algebraic Branching Program (ABP) is a model of computation defined by Nisan (Nis91). Formally, an ABP is defined as follows.

Definition 5.3.2. (Algebraic Branching Program) *An algebraic branching program (ABP) is a directed acyclic graph with one source and one sink. The vertices of this graph are partitioned into levels labelled* 0 *to* d*, where edges may go from level* i *to level* $i+1$*. The parameter* d *is called the degree of the ABP. The source is the only vertex at level* 0 *and the sink is the only vertex at level* d*. Each edge is labelled with a homogeneous linear function of* $x_1, \ldots, x_n$ *(i.e. a function of the form* $\sum_i c_i x_i$*). The width of the ABP is the maximum number of vertices in any level, and the size is the total number of vertices.*

An ABP computes a polynomial by taking sum over all paths from source to sink, the product of all linear functions by which the edges of the path are labelled.

An ABP is said to be *planar* if the underlying graph is planar.

The following argument shows how Corollary 5.2.2 follows easily from Theorem 5.2.1.

Proof of Corollary 5.2.2: In the proof of Theorem 5.2.1 we have constructed a depth 2 circuit D that computes $P = \prod_j (A_{j0} + A_{j1}x_1 + \ldots + A_{jn}x_n)$, where each $A_{ji} \in \mathsf{U}_2(\mathbb{F})$. We can make D homogeneous by introducing an extra variable z, such that $P = \prod_j (A_{j0}z + A_{j1}x_1 + \ldots + A_{jn}x_n)$. This means, the product sequence considered in Lemma 5.3.1, is such that all the linear matrices have homogeneous linear functions as entries and the only non-diagonal linear matrices are of the form $\begin{bmatrix} z & cx_i \\ & z \end{bmatrix}$. It

is now straightforward to construct a width-2 ABP by making the j^{th} linear matrix in the sequence act as the biadjacency matrix between level j and $j+1$ of the ABP. The ABP constructed is planar since it has layers only of the following two kinds:

where l_1. l_2 are homogeneous linear functions. □

As a matter of fact, the above argument actually shows that PIT of depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ reduces to PIT of width-2 ABPs.

5.4 Identity Testing over Commutative Algebras

We will prove Theorem 5.2.3 in this section. The main tool used in this proof is the structure theorem for finite dimensional commutative algebra stated in Section 2.3.4. We will first prove this theorem.

5.4.1 Proof of the Structure Theorem

Theorem 2.3.7 (restated.) *A finite dimensional commutative algebra $\mathcal{R}$ over $\mathbb{F}$ is isomorphic to a direct product of local rings i.e.*

$$\mathcal{R} \cong \mathcal{R}_1 \oplus \ldots \oplus \mathcal{R}_\ell$$

where each $\mathcal{R}_i$ is a local ring contained in $\mathcal{R}$ and any non-unit in $\mathcal{R}_i$ is nilpotent.

Proof. If all non-units in $\mathcal{R}$ are nilpotents then $\mathcal{R}$ is a local ring and the set of nilpotents forms the unique maximal ideal. Suppose, there is a non-nilpotent non-unit z in $\mathcal{R}$. (Any non-unit z in a finite dimensional algebra is a zero-divisor i.e. $\exists y \in \mathcal{R}$ and $y \neq 0$ such that $yz = 0$.) We will later show that using z it is possible to find an *idempotent* $v \notin \{0, 1\}$ (i.e. $v^2 = v$) in $\mathcal{R}$. But at first, let us see what happens if we already have a non-trivial idempotent $v \in \mathcal{R}$.

Let $\mathcal{R}v$ be the sub-algebra of $\mathcal{R}$ generated by multiplying elements of $\mathcal{R}$ with v. Since any $a = av + a(1-v)$ and for any $b \in \mathcal{R}v$ and $c \in \mathcal{R}(1-v)$, $b \cdot c = 0$, we get

$\mathcal{R} \cong \mathcal{R}v \oplus \mathcal{R}(1-v)$ as a non-trivial decomposition of $\mathcal{R}$. (This is the place where we use commutativity of $\mathcal{R}$.) By repeating the splitting process on the sub-algebras we can eventually prove the theorem.

It remains to see how to find an idempotent from a zero-divisor z. An element $a \in \mathcal{R}$ can be equivalently expressed as a matrix in $\mathsf{M}_k(\mathbb{F})$, where $k = \dim_{\mathbb{F}}(\mathcal{R})$, by treating a as the linear transformation on $\mathcal{R}$ that takes $b \in \mathcal{R}$ to $a \cdot b$. Therefore, z is a zero-divisor if and only if z as a matrix is singular. Consider the Jordan normal form of z. Since it is merely a change of basis we can assume, without loss of generality, that z is already in Jordan normal form. (We will not compute the Jordan normal form in our algorithm, it is used only for the sake of argument.) Let,

$$z = \begin{bmatrix} A & 0 \\ 0 & N \end{bmatrix}$$

where A, N are block diagonal matrices and A is non-singular and N is nilpotent. Then,

$$w = z^k = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix}$$

where $B = A^k$ is non-singular. The claim is, there is an identity element in the sub-algebra $\mathcal{R}w$ which can be taken to be the idempotent v that splits $\mathcal{R}$. First observe that the minimum polynomial of w over $\mathbb{F}$ is $m(x) = x \cdot m'(x)$, where $m'(x)$ is the minimum polynomial of B. Also if $m(x) = \sum_{i=1}^{k} \alpha_i x^i$ then $\alpha_1 \neq 0$ as it is the constant term of $m'(x)$ and B is non-singular. Therefore, there exists an $a \in \mathcal{R}$ such that $w \cdot (aw - 1) = 0$. Hence $v = aw$ is the identity element of $\mathcal{R}w$ and is also an idempotent in $\mathcal{R}$. □

We are now ready to prove Theorem 5.2.3.

5.4.2 A Deterministic Algorithm

Theorem 5.2.3 (restated.) *Given an expression,*

$$P = \prod_{i=1}^{d} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n),$$

where $A_{ij} \in \mathcal{R}$, a commutative algebra of dimension k over $\mathbb{F}$, there is a deterministic algorithm to test if P is zero running in time $\mathsf{poly}\left(k^k, n, d\right)$.

Proof. Let $\{e_1, \ldots, e_k\}$ be a basis of $\mathcal{R}$ over $\mathbb{F}$. Since any element in $\mathcal{R}$ can be equivalently expressed as a $k \times k$ matrix over $\mathbb{F}$ (by treating it as a linear transformation), we will assume that $A_{ij} \in \mathsf{M}_k(\mathbb{F})$, for all i and j. Further, since $\mathcal{R}$ is given in basis form, we can find these matrix representations of A_{ij}'s efficiently.

If every A_{ij} is non-singular, then surely $P \neq 0$. This can be argued by fixing an ordering $x_1 \succ x_2 \succ \ldots \succ x_n$ among the variables. The coefficient of the leading monomial of P, with respect to this ordering, is a product of invertible matrices and hence $P \neq 0$. Therefore, assume that $\exists A_{ij} = z$ such that z is a zero-divisor i.e. singular. From the proof of Theorem 2.3.7 it follows that the sub-algebra $\mathcal{R}w$, where $w = z^k$, contains an identity element v which is an idempotent. We now argue that the idempotent v can be found by solving a system of linear equations over $\mathbb{F}$. Let $b_1, \ldots, b_{k'}$ be a basis of $\mathcal{R}w$, which we can find easily from the elements $e_1w, \ldots, e_kw$. To solve for v write it as,

$$v = \nu_1 b_1 + \ldots + \nu_{k'} b_{k'}$$

where $\nu_j \in \mathbb{F}$ are unknowns. Since v is an identity in $\mathcal{R}w$ it satisfies the relation,

$$(\nu_1 b_1 + \ldots + \nu_{k'} b_{k'}) \cdot b_i = b_i \quad \text{for } 1 \leq i \leq k'.$$

Expressing each b_i in terms of $e_1, \ldots, e_k$, we get a system of linear equations in the ν_j's. Find v by solving this linear system.

Since $\mathcal{R} \cong \mathcal{R}v \oplus \mathcal{R}(1-v)$, we can split the identity testing problem into two subproblems. That is, P is zero if and only if,

$$\begin{aligned} Pv &= \prod_{i=1}^{d} (A_{i0}v + A_{i1}v \cdot x_1 + \ldots + A_{in}v \cdot x_n) \quad \text{and} \\ P(1-v) &= \prod_{i=1}^{d} (A_{i0}(1-v) + A_{i1}(1-v) \cdot x_1 + \ldots + A_{in}(1-v) \cdot x_n) \end{aligned}$$

are both zero. What we just did with $P \in \mathcal{R}$ we can repeat for $Pv \in \mathcal{R}v$ and $P(1-v) \in \mathcal{R}(1-v)$ by applying the above process, recursively, to Pv and $P(1-v)$. By decomposing the algebra each time an A_{ij} is a non-nilpotent zero-divisor, we are finally left with the easier problem of checking if,

$$P = \prod_{i=1}^{d} (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n)$$

is zero, where the coefficients A_{ij}'s are either nilpotent or invertible matrices.

Let $T_i = (A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n)$ be a term such that the coefficient of x_j in T_i, i.e. A_{ij} is invertible. And let Q be the product of all terms other than T_i. Then $P = T_i \cdot Q$ (since $\mathcal{R}$ is commutative). Fix an ordering among the variables so that x_j gets the highest priority. The leading coefficient of P, under this ordering, is A_{ij} times the leading coefficient of Q. Since A_{ij} is invertible this implies that $P = 0$ if and only if $Q = 0$. (If A_{i0} is invertible, we can arrive at the same conclusion by arguing with the coefficients of the least monomials of P and Q under some ordering.) In other words, $P = 0$ if and only if the product of all those terms for which all the coefficients are nilpotent matrices is zero. If the number of such terms is greater than k then P is automatically zero. This follows from the fact that commuting matrices (over an algebraically closed field) can be simultaneously triangularized and the product of k upper-triangular $k \times k$ nilpotent matrices is always zero.

Otherwise, treat each term $(A_{i0} + A_{i1}x_1 + \ldots + A_{in}x_n)$ as a $k \times k$ linear matrix. Since, there are at most k such linear matrices in P, the total number of linear functions occurring as entries of these linear matrices is bounded by k^3. Using a basis of these linear functions we can reduce the number of effective variables in P to k^3. Now, checking if P is zero takes only $\mathsf{poly}(k^k)$ field operations and hence the overall time complexity is bounded by $\mathsf{poly}(k^k, n, d)$. □

5.4.3 Reduction from Depth 3 Identity Testing

It is clear from the discussion in Section 5.4.2 that identity testing of depth 2 ($\Pi\Sigma$) circuits over commutative algebras reduces in polynomial time to that over local rings. As long as the dimensions of these local rings are $O(\log nd/\log\log nd)$, we have an efficient algorithm. But what happens for larger dimensions? The following result shows the hardness of this problem.

Theorem 5.4.1. *Given a depth* 3 ($\Sigma\Pi\Sigma$) *circuit* C *of degree* d *and top level fan-in* s*, it is possible to construct in polynomial time a depth* 2 ($\Pi\Sigma$) *circuit* $\tilde{C}$ *over a local ring of dimension* $s(d-1)+2$ *over* $\mathbb{F}$ *such that* $\tilde{C}$ *computes a zero polynomial if and only if* C *does so.*

Proof. Consider a depth 3 ($\Sigma\Pi\Sigma$) circuit computing a polynomial $f = \sum_{i=1}^{s}\prod_{j=1}^{d} l_{ij}$, where l_{ij}'s are linear functions. Consider the ring $\mathcal{R} = \mathbb{F}[y_1, \ldots, y_s]/\mathcal{I}$, where

$\mathcal{I}$ is an ideal generated by the elements $\{y_i y_j\}_{1\le i<j\le s}$ and $\{y_1^d - y_i^d\}_{1<i\le s}$. Observe that $\mathcal{R}$ is a local ring, as $y_i^{d+1} = 0$ for all $1 \le i \le s$. Also the elements $\{1, y_1, \ldots, y_1^d, y_2, \ldots, y_2^{d-1}, \ldots, y_s, \ldots, y_s^{d-1}\}$ form an $\mathbb{F}$-basis of $\mathcal{R}$. Now notice that the polynomial,

$$\begin{aligned} P &= \prod_{j=1}^{d} (l_{j1}y_1 + \ldots + l_{js}y_s) \\ &= f \cdot y_1^d \end{aligned}$$

is zero if and only if f is zero. Polynomial P can indeed be computed by a depth 2 ($\Pi\Sigma$) circuit over $\mathcal{R}$. □

5.5 Weakness of the Depth 2 Model

In Lemma 5.3.1, we have constructed a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ that computes $L \cdot f$ instead of f. Is it possible to drop the factor L and simply compute f? In this section, we show that in *many* cases it is impossible to find a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ that computes f.

5.5.1 Depth 2 Model over $\mathsf{U}_2(\mathbb{F})$

The ideal of $\mathbb{F}[x_1, \ldots, x_n]$ generated by two linear functions l_1 and l_2 is denoted by (l_1, l_2). We say that l_1 is *independent* of l_2 if $1 \notin (l_1, l_2)$.

Theorem 5.2.5 (restated.) *Let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial such that there are no two independent linear functions l_1 and l_2 which make $f \bmod (l_1, l_2)$ also a linear function. Then f is not computable by a depth 2 ($\Pi\Sigma$) circuit over $\mathsf{U}_2(\mathbb{F})$.*

Proof. Assume on the contrary that f can be computed by a depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$. In other words, there is a product sequence $M_1 \cdots M_t$ of 2×2 upper-triangular linear matrices such that f appears as the top-right entry of the final product matrix. Let $M_i = \begin{bmatrix} l_{i1} & l_{i2} \\ & l_{i3} \end{bmatrix}$, then

$$f = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} \\ & l_{13} \end{bmatrix} \begin{bmatrix} l_{21} & l_{22} \\ & l_{23} \end{bmatrix} \cdots \begin{bmatrix} l_{t1} & l_{t2} \\ & l_{t3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5.4}$$

Case 1: Not all the l_{i1}'s are constants.

Let k be the least index such that l_{k1} is not a constant and $l_{i1} = c_i$ for all $i < k$. To simplify Equation 5.4, let

$$\begin{aligned} \begin{bmatrix} B \\ L \end{bmatrix} &= M_{k+1} \cdots M_t \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \begin{bmatrix} d_i & D_i \end{bmatrix} &= \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot M_1 \cdots M_{i-1} \end{aligned}$$

Observe that L is just a product of linear functions, and for all $1 \leq i < k$, we have the following relations.

$$\begin{aligned} d_{i+1} &= \prod_{j=1}^{i} c_j \\ D_{i+1} &= d_i l_{i2} + l_{i3} D_i \end{aligned}$$

Hence, Equation 5.4 simplifies as

$$\begin{aligned} f &= \begin{bmatrix} d_k & D_k \end{bmatrix} \begin{bmatrix} l_{k1} & l_{k2} \\ & l_{k3} \end{bmatrix} \begin{bmatrix} B \\ L \end{bmatrix} \\ &= d_k l_{k1} B + (d_k l_{k2} + l_{k3} D_k) L \end{aligned}$$

Suppose there is some factor l of L with $1 \notin (l_{k1}, l)$. Then $f = 0 \bmod (l_{k1}, l)$, which is not possible. Hence, L must be a constant modulo l_{k1}. For appropriate constants α, β, we have

$$f = \alpha l_{k2} + \beta l_{k3} D_k \pmod{l_{k1}} \tag{5.5}$$

By inducting on k, we argue that the above relation can not be true. If l_{k3} was independent of l_{k1}, then $f = \alpha l_{k2} \bmod (l_{k1}, l_{k3})$ which is not possible. Therefore, l_{k3} must be a constant modulo l_{k1}. We then have the following (reusing α and β to denote appropriate constants):

$$\begin{aligned} f &= \alpha l_{k2} + \beta D_k \pmod{l_{k1}} \\ &= \alpha l_{k2} + \beta \left(d_{k-1} l_{(k-1)2} + l_{(k-1)3} D_{k-1} \right) \pmod{l_{k1}} \\ \implies f &= \left(\alpha l_{k2} + \beta d_{k-1} l_{(k-1)2} \right) + \beta l_{(k-1)3} D_{k-1} \pmod{l_{k1}}. \end{aligned}$$

The last equation can be rewritten in the form of Equation 5.5 with the term $\beta l_{k3} D_k$ replaced by $\beta l_{(k-1)3} D_{k-1}$. Notice that the expression $\left(\alpha l_{k2} + \beta d_{k-1} l_{(k-1)2} \right)$ is linear

just like αl_{k2}. Hence, by using the argument iteratively we eventually get a contradiction at D_1.

Case 2: All the l_{i1}'s are constants.

In this case, Equation 5.4 can be rewritten as

$$\begin{aligned} f &= \begin{bmatrix} d_t & D_t \end{bmatrix} \begin{bmatrix} c_t & l_{t2} \\ & l_{t3} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &= d_t l_{t2} + l_{t3} D_t \end{aligned}$$

The last equation is again of the form in Equation 5.5 (without the mod term) and hence the same argument can be repeated here as well to give the desired contradiction. □

The following corollary provides some explicit examples of functions that cannot be computed.

Corollary 5.5.1. *A depth 2 circuit over $\mathsf{U}_2(\mathbb{F})$ cannot compute the polynomial $x_1x_2 + x_3x_4 + x_5x_6$. Other examples include well known functions like det_n and perm_n, the determinant and permanent polynomials, for $n \geq 3$.*

Proof. It suffices to show that $f = x_1x_2 + x_3x_4 + x_5x_6$ satisfy the requirement in Theorem 5.2.5.

To obtain a contradiction, let us assume that there does exist two linear functions l_1 and l_2 (with $1 \notin (l_1, l_2)$) such that $f \bmod (l_1, l_2)$ is linear. We can evaluate $f \bmod (l_1, l_2)$ by substituting a pair of the variables in f by linear functions in the rest of the variables (as dictated by the equations $l_1 = l_2 = 0$). By the symmetry of f, we can assume that the pair is either $\{x_1, x_2\}$ or $\{x_1, x_3\}$.

If $x_1 = l_1'$ and $x_3 = l_2'$ are the substitutions, then $l_1'x_2 + l_2'x_4$ can never contribute a term to cancel off x_5x_6 and hence $f \bmod (l_1, l_2)$ cannot be linear.

Otherwise, let $x_1 = l_1'$ and $x_2 = l_2'$ be the substitutions. If $f \bmod (l_1, l_2) = l_1'l_2' + x_3x_4 + x_5x_6$ is linear, there cannot be a common x_i with non-zero coefficient in both l_1' and l_2'. Without loss of generality, assume that l_1' involves x_3 and x_5 and l_2' involves x_4 and x_6. But then the product $l_1'l_2'$ would involve terms like x_3x_6 that cannot be cancelled, contradicting linearity again. □

5.5.2 Depth 2 Model over $\mathsf{M}_2(\mathbb{F})$

In this section we show that the power of depth 2 circuits is very restrictive even if we take the underlying algebra to be $\mathsf{M}_2(\mathbb{F})$ instead of $\mathsf{U}_2(\mathbb{F})$. In the following discussion, we call a homogeneous linear function a *linear form*.

Definition 5.5.2. *A polynomial f of degree n is said to be* r-robust *if f does not belong to any ideal generated by r linear forms.*

For instance, it can be checked that det_n and perm_n, the symbolic determinant and permanent of an $n \times n$ matrix, are $(n-1)$-robust polynomials. For any polynomial f, denote the d^{th} homogeneous part of f by $[f]_d$. And let $(h_1, \cdots, h_k)$ denote the ideal generated by $h_1, \cdots, h_k$. Recall the definition of *degree restriction* from Definition 5.2.6.

Theorem 5.5.3. *A polynomial f of degree n, such that $[f]_n$ is 5-robust, cannot be computed by a depth 2 circuit over $\mathsf{M}_2(\mathbb{F})$ under a degree restriction of n.*

We prove this with the help of the following lemma, which basically applies Gaussian column operations to simplify matrices.

Lemma 5.5.4. *Let f_1 be a polynomial of degree n such that $[f_1]_n$ is 4-robust. Suppose there is a linear matrix M and polynomials f_2, g_1, g_2 of degree at most n satisfying*

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = M \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

Then, there is an appropriate invertible column operation A such that

$$M \cdot A = \begin{bmatrix} 1 & h_2 \\ c_3 & h_4 + c_4 \end{bmatrix}$$

where c_3, c_4 are constants and h_2, h_4 are linear forms.

We will defer the proof of this lemma to the end of this section, and shall use it to prove Theorem 5.5.3.

Proof of Theorem 5.5.3. Assume, on the contrary, that there is a sequence of 2×2 linear matrices computing f. Since only one entry is of interest to us, assume without

loss of generality that the first matrix in the sequence is a row vector $\bar{v}$ and the last matrix is a column vector $\bar{w}$. Let,

$$f = \bar{v} \cdot M_1 M_2 \cdots M_d \cdot \bar{w}$$

be a sequence of minimum length computing f. Using Lemma 5.5.4 we repeatedly transform the above sequence by replacing the term $M_i M_{i+1}$ by $(M_i A)(A^{-1} M_{i+1})$ for an appropriate invertible column transformation A. Since entries of A are just constants, $M_i A$ and $A^{-1} M_{i+1}$ continue to be linear matrices.

To begin, let $\bar{v} = [l_1, l_2]$ for two linear functions l_1 and l_2. And let $[f_1, f_2]^T = M_1 \cdots M_d \bar{w}$. Then,

$$\begin{bmatrix} f \\ 0 \end{bmatrix} = \begin{bmatrix} l_1 & l_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}.$$

Applying Lemma 5.5.4, we can assume that $\bar{v} = [1, h]$ and so $f = f_1 + h f_2$. By the minimality of the sequence, $h \neq 0$. This forces $[f_1]_n = [f]_n$ to be 4-robust and the degree restriction makes $[f_2]_n = 0$.

Let $[g_1, g_2]^T = M_2 \cdots M_d \bar{w}$. The goal is to translate the properties that $[f_1]_n$ is 4-robust and $[f_2]_n = 0$ to the polynomials $[g_1]_n$ and $[g_2]_n$ respectively. We use induction and translate these properties to the vectors $M_i \cdots M_d \bar{w}$, for all $i \geq 2$. So, suppose that the relation,

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = M_i \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

holds in general for some i, where $[f_1]_n$ is 4-robust and $[f_2]_n = 0$.

Since $[f_1]_n$ is 4-robust, using Lemma 5.5.4 again, we can assume that

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & h_2 \\ c_3 & c_4 + h_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \tag{5.6}$$

by reusing the symbols g_1, g_2 and others. Observe that in the above equation if $h_4 = 0$ then $M_{i-1} M_i$ still continues to be a linear matrix (since, by induction, M_{i-1} is of the form as dictated by Lemma 5.5.4) and that would contradict the minimality of the sequence. Therefore $h_4 \neq 0$.

Claim: $c_3 = 0$ (by comparing $[f_1]_n$ and $[g_1]_n$, as explained below).

Proof: As $h_4 \neq 0$, the degree restriction forces $[g_2]_n = 0$. And since $[f_2]_n = 0$, we have the relation $c_3[g_1]_n = -h_4[g_2]_{n-1}$. If $c_3 \neq 0$, we have $[g_1]_n \in (h_4)$, contradicting 4-robustness of $[f_1]_n$ as then $[f_1]_n = [g_1]_n + h_2[g_2]_{n-1} \in (h_2, h_4)$. □

Therefore Equation 5.6 gives,

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 1 & h_2 \\ 0 & c_4 + h_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

with $h_4 \neq 0$. Also, since $[f_2]_{n+1} = [f_2]_n = 0$ this implies that $[g_2]_n = [g_2]_{n-1} = 0$. Hence, $[g_1]_n = [f_1]_n$ is 4-robust. Thus, we have translated the properties to $[g_1\, g_2]^T$, showing that $[g_1]_n$ is 4-robust and $[g_2]_n = 0$. However, since there are only finitely many matrices in the sequence, there must come a point when degree of g_1 in $[g_1\, g_2]^T = M_i \cdots M_d \bar{w}$ drops below n for some $i \geq 2$. At this point we get a contradiction as $[g_1]_n = 0$ (reusing symbol) which contradicts robustness. □

We only need to finish the proof of Lemma 5.5.4.

Proof of Lemma 5.5.4. Suppose we have an equation of the form,

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} h_1 + c_1 & h_2 + c_2 \\ h_3 + c_3 & h_4 + c_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}, \tag{5.7}$$

where $c_1, \ldots, c_4$ are constants and $h_1, \ldots, h_4$ are linear forms. On comparing degree $n+1$ terms, we have the relations,

$$\begin{aligned} h_1[g_1]_n + h_2[g_2]_n &= 0 \\ h_3[g_1]_n + h_4[g_2]_n &= 0. \end{aligned}$$

If h_3 and h_4 (a similar reasoning holds for h_1 and h_2) are not proportional (i.e. not multiple of each other), then the above equation implies that $[g_1]_n, [g_2]_n \in (h_3, h_4)$. But this means,

$$[f_1]_n = h_1[g_1]_{n-1} + h_2[g_2]_{n-1} + c_1[g_1]_n + c_2[g_2]_n \in (h_1, h_2, h_3, h_4),$$

contradicting the 4-robustness of $[f_1]_n$. Thus, h_3 and h_4 (as well as h_1 and h_2) are proportional, in the same ratio as $-[g_2]_n$ and $[g_1]_n$. Using an appropriate column operation, Equation 5.7 simplifies to

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} c_1 & h_2 + c_2 \\ c_3 & h_4 + c_4 \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix},$$

reusing symbols g_1, g_2 and others. If $c_1 = 0$ and $[g_2]_n = 0$ then $[f_1]_n = h_2[g_2]_{n-1}$, contradicting robustness. Therefore, either $c_1 \neq 0$, in which case another column transformation gets the matrix to the form claimed, or $[g_2]_n \neq 0$ implying that $h_2 = h_4 = 0$. But then c_1 and c_2 both cannot be zero, $[f_1]_n$ being 4-robust, and hence a column transformation yields the desired form. □

5.6 Conclusion

We give a new perspective to identity testing of depth 3 arithmetic circuits by showing an equivalence to identity testing of depth 2 circuits over $\mathsf{U}_2(\mathbb{F})$ and a reduction to identity testing of depth 2 circuits over commutative algebra. We also show that identity testing of depth 2 circuits over commutative algebra of small dimension can be solved in deterministic polynomial time. For this, our algorithm crucially uses an interesting structural result involving local rings. This naturally poses the following questions:

- Can we exploit properties very specific to the ring of upper-triangular 2×2 matrices to solve PIT of depth 3 circuits?
- Can we use more algebraic insights into commutative algebras or local rings to extend our deterministic algorithm to commutative algebras of *linear* dimension?

Although we show a reduction from depth 3 circuits, the exact power of depth 2 circuits over commutative algebras of polynomial dimension is not very clear. Same is true for depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$, in which case we are able to show its computational limitation only under the assumption of degree restriction. We are therefore interested in answering the following questions:

- Does identity testing of depth 4 circuits reduce to identity testing of depth 2 circuits over commutative algebra of polynomial dimension?
- Can we show that depth 2 circuits over $\mathsf{M}_2(\mathbb{F})$ are not powerful enough to compute depth 3 polynomials?

Chapter 6

Two Problems on Identity Testing

-Joint work with Ramprasad Saptharishi and Nitin Saxena.

6.1 Introduction

6.1.1 The Problems

In this chapter, we give deterministic solutions to two problems on identity testing. The first one is a generalization of the problem considered by Kayal and Saxena (KS07).

Problem 6.1.1. *Given a depth-3 circuit C with bounded top fanin and given a sparse polynomial f explicitly, check if $p(C) = f$, where $p(C)$ is the polynomial computed by C.*

We say that a polynomial is given *explicitly* if it is presented as a list of monomials, where every monomial is given by its exponent vector and its nonzero coefficient. We assume that the integer entries of the exponent vectors are given in unary. (See the paragraph after the statement of Problem 6.1.3).

As mentioned in Chapter 5 (Section 5.1), the Kayal-Saxena test (KS07) checks if $p(C) = 0$, where C is a bounded top fanin depth-3 circuit. Building on their idea of using 'Chinese remaindering over local rings', we show that the general Problem 6.1.1 can also be solved in deterministic polynomial time.

Our second problem is checking the validity of a given factorization. That is,

Problem 6.1.2. *Given a polynomial f explicitly, and also given t other polynomials $g_1, \ldots, g_t$ explicitly (g_i's need not be distinct), check if $f = g_1 \ldots g_t$.*

This natural case of depth-4 identity testing has been mentioned as a problem with no known efficient deterministic solution in a work by von zur Gathen (vzG83) on sparse polynomial factoring. We are not aware of any progress made towards efficiently solving this problem. In this work, we give a deterministic algorithm to solve the following special case of this problem.

Problem 6.1.3. *Given a polynomial f and t other polynomials $g_1, \ldots, g_t$ explicitly, where every g_i is of the form of a sum of univariate polynomials, check if $f = g_1 \ldots g_t$.*

We show that Problem 6.1.3 can be solved in deterministic polynomial time. It is worth noting, the assumption that the exponent vector of every monomial is given in unary, instead of binary, is made to avoid certain unpleasant situations where even basic questions on univariate polynomials like, checking coprimeness of univariates, division of the product of a set of univariates by a single univariate etc., become NP-hard (see the work by Plaisted (Pla77)).

6.1.2 Overview of Our Approach

The Dual Representation

The common tool we use in solving both Problem 6.1.1 and Problem 6.1.3 is something known as the *dual representation* of polynomials (borrowing terminology from (Sax08)). It is a technique to express a power of a sum of univariates as a 'small' sum of product of univariates. The usefulness of this latter representation is that identity testing of sum of product of univariates can be done very efficiently using a dimension argument that we will describe in Section 6.2. It is similar in spirit to the argument on identity testing of noncommutative formulas by Raz and Shpilka (RS04). An added feature of this dimension argument is that it not only tells us if the input polynomial is zero or not, but also gives us the leading monomial of the polynomial under natural lexicographic ordering of the monomials. This observation turns out to be crucial in extending the Kayal-Saxena test to a solution for Problem 6.1.1.

We note that finding the leading monomial of a given depth-3 circuit is supposedly a harder problem than identity testing. In a recent work, Kayal (Kay10) gives it as a challenge problem to either devise a randomized algorithm to find the leading monomial of a $\Sigma\Pi\Sigma$ circuit or show that the existence of any efficient algorithm imply that the polynomial hierarchy collapses. Fortunately, to our benefit, the problem turns out to be easy in the particular case we are interested in.

Approach to Solving Problem 6.1.1

The general idea is to begin by applying the Kayal-Saxena test to the polynomial $p(C) - f$. This test uses some invertible linear maps to transform the polynomial $p(C) - f$, thereby altering f to some nonsparse polynomial at various stages of the algorithm. However, it can be ensured that the transformed f always stays in a special form that we call a *semidiagonal* form.

Definition 6.1.4. (Semidiagonal Polynomial) *A polynomial $f = \sum_{i=1}^{m} Q_i$ is said to be semidiagonal if every Q_i is a product of a monomial and constantly many powers of linear functions in the variables.*

In our case, the constant in 'constantly many powers of linear functions' is bounded by k, the top fanin of the circuit C. Identity testing of a semidiagonal f can be made easy by using the dual representation of f. So if we can also ensure that in the base case the Kayal-Saxena test on $p(C) - f$ always reduces to identity testing of a semidiagonal f then we are done. Let $p(C) = \sum_{i=1}^{k} P_i$, where P_i's are product of linear functions. The original Kayal-Saxena test (where we test if $p(C) = 0$) chooses a P_j such that $LM(P_j) \succeq LM(p(C))$ (LM stands for leading monomial and $\succeq$ refers to the monomial ordering). But now the test is on $p(C) - f$ and not just $p(C)$. So, at an intermediate stage of the algorithm we need to find a P_j such that $LM(P_j) \succeq LM(p(C) - f)$, where f is semidiagonal. This is where we need to find the leading monomial of f. As mentioned before, finding the leading monomial of a general depth-3 polynomial has no known efficient algorithm. But, once again the semidiagonal nature of f and its corresponding dual representation come to our rescue. Finding leading monomial of a semidiagonal f turns out to be efficiently doable.

Approach to Solving Problem 6.1.3

The approach we take to check if $f = g_1 \dots g_t$ is very natural - reduce the problem to divisibility and then use Chinese remaindering. But the issue here is that g_i's need not be distinct. So in general we need to check if g^d divides f for some $d \geq 1$. Since g^d may not be a sparse polynomial we reduce this divisibility problem to checks of the kind, g divides h, where h is a sparse polynomial, using partial derivatives. We show that such divisibility checks reduce to identity testing of a slightly general form of semidiagonal polynomials, if g is a sum of univariates. And this identity test can be done efficiently using dual representation. Finally, for Chinese remaindering to take its effect, we need to say something about the coprimeness of g_i and g_j when $g_i \neq g_j$. Towards this, we show an irreducibility result on polynomials of the form $f(x) + g(y) + h(z)$ that helps us conclude the proof.

Remark - The technique of dual representation of polynomials is used by Saxena (Sax08) to solve identity testing of diagonal depth-3 circuits and some of its generalizations. Very recently, Kayal (Kay10) shows that an argument based on partial derivatives can also be used to solve these problems. Just like dual representations, Kayal's technique can also be applied to solve Problem 6.1.1 and Problem 6.1.3. In fact, it appears to us that Problem 6.1.1 and 6.1.3 are two natural problems where both Saxena's and Kayal's techniques fit in to give something productive.

6.2 The Dual Representation

6.2.1 Finding the Dual Polynomial

In this section, we show how to express a power of a sum of univariate polynomials as a 'small' sum of product of univariates. The argument is already present in (Sax08). We reproduce it here for self-containment.

Let $f = \left(\sum_{i=1}^{n} g_i(x_i)\right)^D$, where $g_i(x_i)$ (or simply g_i) is a univariate in x_i of degree at most d. Assume that the underlying field $\mathbb{F}$ has size greater than $nD + 1$ and $\text{char}(\mathbb{F}) = 0$ or $\text{char}(\mathbb{F}) > D$. Let z be a new variable and $g = \sum_{i=1}^{n} g_i$. Then in the exponential series expansion $e^{gz} = \sum_{j=0}^{\infty} \frac{g^j}{j!} z^j$, $f/D!$ is the coefficient of z^D. On

the other hand, e^{gz} is also the product of the formal series $e^{g_i z}$ for $1 \leq i \leq n$ i.e. $e^{gz} = \prod_{i=1}^{n} e^{g_i z}$. Define the polynomials,

$$E_D(g_i, z) = \sum_{j=0}^{D} \frac{{g_i}^j}{j!} z^j$$

Then the coefficient of z^D in the formal series $\sum_{j=0}^{\infty} \frac{g^j}{j!} z^j$ is exactly equal to the coefficient of z^D in the polynomial $P(z) = \prod_{i=1}^{n} E_D(g_i, z)$. The idea is to use interpolation to express this coefficient of z^D as an $\mathbb{F}$-linear combination of few evaluations of $P(z)$ at different field points.

Suppose $P(z) = \sum_{j=0}^{nD} p_j z^j$, where p_j's are polynomials in $x_1, \ldots, x_n$. Choose $nD + 1$ distinct points $\alpha_0, \ldots, \alpha_{nD}$ from $\mathbb{F}$ and V be the $(nD + 1) \times (nD + 1)$ Vandermonde matrix $(\alpha_j^k)_{0 \leq j,k \leq nD}$. Then,

$$\begin{aligned} \mathsf{V} \cdot (p_0, \ldots, p_{nD})^T &= (P(\alpha_0), \ldots, P(\alpha_{nD}))^T \\ \Rightarrow (p_0, \ldots, p_{nD}) &= \mathsf{V}^{-1} \cdot (P(\alpha_0), \ldots, P(\alpha_{nD}))^T \end{aligned}$$

In other words, p_D can be expressed as an $\mathbb{F}$-linear combination of the $P(\alpha_j)$'s for $0 \leq j \leq nD$. Now, to complete the proof, notice that $p_D = f/D!$ and each $P(\alpha_j)$ is a product of the univariates $E_D(g_i, \alpha_j)$ of degree at most dD. This proves the following theorem (Sax08).

Theorem 6.2.1. *Given $f = \left(\sum_{i=1}^{n} g_i(x_i)\right)^D$, where g_i is a univariate in x_i of degree at most d, f can be expressed as a sum of $nD + 1$ products of univariates of degree dD, using $\mathsf{poly}(n, d, D)$ $\mathbb{F}$-operations.*

6.2.2 Leading Monomial of Sum of Product of Univariates

Let us now see how identity testing of a sum of product of univariates can be performed in deterministic polynomial time. Surely, the result follows as a corollary to Raz and Shpilka's (RS04) algorithm on noncommutative formula identity testing. But we choose to present an argument here not only because of self-containment but also to point out an interesting feature of this identity testing - it finds the leading monomial of the input polynomial (something we will need in Section 6.3). Our argument is similar in spirit to that of Raz and Shpilka (RS04).

Let $f = \sum_{i=1}^{k} \prod_{j=1}^{n} g_{ij}(x_j)$, where $g_{ij}(x_j)$ (or simply g_{ij}) is a univariate in x_j of degree at most d. Denote by $\succeq$ the natural lexicographic ordering among monomials with $x_1 \succ \ldots \succ x_n$. The following discussion suggests an efficient recursive algorithm to find $LM(f)$.

In general, at any ℓ^{th} step of the recursion we need to find the leading monomial of a polynomial of the form $f_\ell = \sum_{i=1}^{k} G_{i\,\ell-1} \cdot \prod_{j=\ell}^{n} g_{ij}$, where $G_{i\,\ell-1}$'s are polynomials in $x_1, \ldots, x_{\ell-1}$ and $LM(f) = LM(f_\ell)$. To begin, $\ell = 1$ and $G_{i0} = 1$ for all $1 \leq i \leq k$.

Fix an integer $\ell \in [1, n]$. Consider the monomials with nonzero coefficients occurring in the products $G_{i\,\ell-1} \cdot g_{i\ell}$ for all i between 1 and k. Let the set of these monomials be $M_\ell = \{m_1, \ldots, m_{\mu(\ell)}\}$ such that $m_1 \succ m_2 \succ \ldots \succ m_{\mu(\ell)}$. Similarly, consider the monomials with nonzero coefficients in the partial products $\prod_{j=\ell+1}^{n} g_{ij}$ for all $1 \leq i \leq k$ and call them $N_\ell = \{n_1, \ldots, n_{\nu(\ell)}\}$ (also assume that $n_1 \succ n_2 \succ \ldots \succ n_{\nu(\ell)}$). Then, for any i, $G_{i\,\ell-1} \cdot g_{i\ell} = \sum_{r=1}^{\mu(\ell)} c_{ir} m_r$ and $\prod_{j=\ell+1}^{n} g_{ij} = \sum_{s=1}^{\nu(\ell)} d_{is} n_s$, where the coefficients c_{ir} and d_{is} belong to $\mathbb{F}$.

Notice that the monomials $m_r \cdot n_s$ are distinct for distinct tuples (r, s). The coefficient of $m_r \cdot n_s$ in f_ℓ is exactly $\sum_{i=1}^{k} c_{ir} d_{is}$. Put differently, if $\mathbf{c}_r$ is the k-dimensional vector $(c_{1r}, \ldots, c_{kr})$ and $\mathbf{d}_s$ is the vector $(d_{1r}, \ldots, d_{ks})$ then $\text{Coeff}(m_r \cdot n_s) = \mathbf{c}_r \boldsymbol{\cdot} \mathbf{d}_s^T$. Consider the $\mu(\ell) \times k$ matrix C with vectors $\mathbf{c}_r$ as rows for all $1 \leq r \leq \mu(\ell)$, and D be the $k \times \nu(\ell)$ matrix with vectors $\mathbf{d}_s^T$ as columns for all $1 \leq s \leq \nu(\ell)$. The coefficient of $m_r \cdot n_s$ is the $(r, s)^{th}$ entry of the product matrix $C \times D$.

We are interested in finding the leading monomial of f_ℓ. Now notice that, the minimum possible r for which there is a nonzero entry (r, s) in $C \times D$, gives the leading monomial $m_r \cdot n_s$ of f_ℓ, where s is also the minimum possible for that choice of r. This is because of the monomial orderings $m_1 \succ m_2 \succ \ldots \succ m_{\mu(\ell)}$ and $n_1 \succ n_2 \succ \ldots \succ n_{\nu(\ell)}$. Therefore, if there is a row vector $\mathbf{c}_r$ in C that is $\mathbb{F}$-linearly dependent on the vectors $\mathbf{c}_1, \ldots, \mathbf{c}_{r-1}$ then the leading monomial of f_ℓ can never be of the form $m_r \cdot n_s$ for any s, and so we can safely drop the row $\mathbf{c}_r$ from C. Since C is a $\mu(\ell) \times k$ matrix, the number of linearly independent rows of C is at most k. Hence, we can iteratively remove rows from C starting from the first row $\mathbf{c}_1$; at the r^{th} iteration dropping $\mathbf{c}_r$ if and only if $\mathbf{c}_r$ is $\mathbb{F}$-linearly dependent on $\mathbf{c}_1, \ldots, \mathbf{c}_{r-1}$. Finally, we are left with a new matrix $\tilde{C}$ with at most k rows.

The dropping of a row $\mathbf{c}_r$ from C corresponds to pruning of the monomial m_r from the product $G_{i\,\ell-1} \cdot g_{i\ell}$. By this we mean the following. Let $\tilde{M}_\ell$ be the subset of those monomials of M_ℓ such that $m_r \in \tilde{M}_\ell$ if and only if $\mathbf{c}_r$ is a row of $\tilde{C}$. Let $G_{i\ell}$ be the product $G_{i\,\ell-1} \cdot g_{i\ell}$ projected to only the monomials in $\tilde{M}_\ell$ i.e. $G_{i\ell} = \sum_{m_r \in \tilde{M}_\ell} c_{ir} m_r$. Then the leading monomial of $f_{\ell+1} = \sum_{i=1}^{k} G_{i\ell} \cdot \prod_{j=\ell+1}^{n} g_{ij}$ is the same as the leading monomial of f_ℓ for reason explained in the last paragraph.

The good part is that $\tilde{M}_\ell$ has at most k monomials and so do the polynomials $G_{i\ell}$ for all $1 \leq i \leq k$. Since the number of monomials in $G_{i\ell}$ does not grow with ℓ, it is easy to see that $LM(f)$ can be found in $\mathsf{poly}(n, k, d)$ time by adapting the above discussion into a recursive algorithm. We have thus shown the following theorem.

Theorem 6.2.2. *Given $f = \sum_{i=1}^{k} \prod_{j=1}^{n} g_{ij}(x_j)$, where g_{ij} is a univariate in x_j of degree at most d, the leading monomial of f under natural lexicographic ordering can be found using $\mathsf{poly}(n, k, d)$ $\mathbb{F}$-operations.*

Note - The above approach actually gives the coefficient of $LM(f)$.

6.3 Generalizing Kayal-Saxena test

In this section, we show how the Kayal-Saxena identity test on bounded top fanin depth-3 circuits can be generalized to tests of the kind: Is $p(C) = f$, where C is a given bounded top fanin depth-3 circuit computing polynomial $p(C)$ and f is a given sparse polynomial. This reduces to testing if the depth-3 polynomial $p(C) - f$ is identically zero. Our algorithm builds on ideas from the Kayal-Saxena test. So, to put things in context, we will review their algorithm first.

To begin with, assume that the circuit C and the polynomial f are homogenized so that all the product terms of $p(C)$ and all the monomials of f have the same degree, which is d (say). Also assume that $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F}) > d$. Let $p(C) = \sum_{i=1}^{k} P_i$, where k is a constant and P_i is a product of d linear forms over $\mathbb{F}$, for all $1 \leq i \leq k$. Let $X = \{x_1, \ldots, x_n\}$ be the underlying set of variables.

6.3.1 Revisiting Kayal-Saxena Test

Kayal and Saxena's algorithm uses recursion (on the top fanin of circuit C) to check if $p(C) = 0$. At an intermediate level of the recursion, the algorithm finds out if a

polynomial of the form,

$$\alpha_1 T_1 + \ldots + \alpha_{k'} T_{k'}$$

is zero over a local ring $\mathcal{R} \supseteq \mathbb{F}$ given explicitly in terms of an $\mathbb{F}$-basis, where $k' \leq k$ and α_i's are elements of $\mathcal{R}$. Each T_i is a product of linear forms over $\mathcal{R}$, where a linear form over $\mathcal{R}$ is of the kind,

$$a_1 x_{v_1} + \ldots + a_{n'} x_{v_{n'}} + m,$$

where $a_i \in \mathbb{F}$ and $m \in \mathcal{M}$, the unique maximal ideal of $\mathcal{R}$. In addition, there is at least one nonzero a_i and $x_{v_i} \in X$ is a free variable over $\mathcal{R}$. At the start of the recursion, $\mathcal{R} = \mathbb{F}$ and $\mathcal{M} = (0)$, the null ideal.

The Algorithm

The following Algorithm **ID** takes as input an $\mathbb{F}$-basis of a local ring $\mathcal{R}$, a set of elements $\langle \alpha_1, \ldots, \alpha_{k'} \rangle$ in $\mathcal{R}$ and a set of products of linear forms (over $\mathcal{R}$), $\langle T_1, \ldots, T_{k'} \rangle$. It checks if the polynomial $\sum_{i=1}^{k'} \alpha_i T_i$ is identically zero over $\mathcal{R}$ and returns YES or NO accordingly.

The description of Algorithm **ID** given below is exactly the same as in (KS07), except a slight modification in Step 3.1. This change is somewhat necessary for our purpose (see the remark on Step 3.1 later).

Algorithm ID$(\mathcal{R}, \langle \alpha_1, \ldots, \alpha_{k'} \rangle, \langle T_1, \ldots, T_{k'} \rangle)$:

Step 1: (Rearranging terms) Order the terms $\alpha_i T_i$ so that $LM(T_1) \succeq LM(T_i)$, for all $2 \leq i \leq k'$. Let $p = \sum_{i=1}^{k'} \alpha_i T_i$.

Step 2: (Base Case) If $k' = 1$ check if $\alpha_1 = 0$. If so, return YES otherwise NO.

Step 3: (Verifying that $p = 0 \mod T_1$) The product T_1 can be split as $T_1 = S_1 \ldots S_r$, with possible change in the constant α_1, such that each S_j is of the form,

$$S_j = (\ell_j + m_1) \cdot (\ell_j + m_2) \ldots (\ell_j + m_t),$$

where $m_i \in \mathcal{M}$ and ℓ_j is a linear form over $\mathbb{F}$. Further, for $i \neq j$, ℓ_i and ℓ_j are coprime linear forms over $\mathbb{F}$.
Check if $p = 0 \mod S_j$, for every $1 \leq j \leq r$. This is done in the following way.

Step 3.1: (Applying a linear transformation) Find a free variable x_u with nonzero coefficient c_u in ℓ_j. Let σ be a linear transformation on the free variables (occuring in $T_1, \ldots T_{k'}$) that sends ℓ_j to x_u. More precisely, $\sigma(x_u) = c_u^{-1}(x_u - \ell_j + c_u x_u)$ and for any other free variable $x_v \neq x_u$, $\sigma(x_v) = x_v$.

Step 3.2: (Recursively verify if $\sigma(p) = 0 \bmod \sigma(S_j)$) Define ring $\mathcal{R}'$ as

$$\mathcal{R}' = \mathcal{R}[x_u]/(\sigma(S_j)),$$

where x_u is the same variable x_u in Step 3.1. Notice that, $\sigma(S_j)$ is of the form,

$$\sigma(S_j) = (x_u + m_1) \cdot (x_u + m_2) \ldots (x_u + m_t),$$

and $\sigma(T_1) = 0 \bmod \sigma(S_j)$. For $i = 2$ to k' compute elements $\beta_i \in \mathcal{R}'$ and T_i' such that,

$$\sigma(T_i) = \beta_i T_i' \bmod \sigma(S_j),$$

where T_i' is a product of linear forms over $\mathcal{R}'$.

Recursively call **ID**$(\mathcal{R}', \langle \beta_2, \ldots, \beta_{k'} \rangle, \langle T_2', \ldots, T_{k'}' \rangle)$. If the recursive call returns NO then output NO and exit, otherwise declare $p = 0 \bmod S_j$.

Declare $p = 0 \bmod T_1$, if $p = 0 \bmod S_j$ for all $1 \leq j \leq r$.

Step 4: Check if Coeff(LM(T_1)) in p is zero. If so, return YES otherwise NO.

Correctness of the Algorithm

Suppose Step 3 ensures that $p = 0 \bmod T_1$, where $LM(T_1) \succeq LM(T_i)$ for all $2 \leq i \leq k'$ which also means that $LM(T_1) \succeq LM(p)$. The way a linear form is defined over $\mathcal{R}$, it follows that Coeff($LM(T_1)$) in T_1 is a nonzero field element. Therefore, $p = \alpha \cdot T_1$ for some $\alpha \in \mathcal{R}$, implying that $p = 0$ if and only if Coeff($LM(T_1)$) in p is zero. This is verified in Step 4.

It remains to show the correctness of Step 3. In order to check if $p = 0 \bmod T_1$, the algorithm finds out if $p = 0 \bmod S_j$ for every $1 \leq j \leq r$. That this is a sufficient condition is implied by the following lemma (also known as 'Chinese Remaindering over local rings'). The way a local ring $\mathcal{R}$ is formed in Algorithm **ID** it has the

property that every element $r \in \mathcal{R}$ can be uniquely expressed as $r = a + m$, where $a \in \mathbb{F}$ and $m \in \mathcal{M}$. Let ϕ be a projection map, taking r to a i.e. $\phi(r) = a$. This map naturally extends to polynomials over $\mathcal{R}$ by acting on the coefficients.

Lemma 6.3.1. [(KS07)] *Let $\mathcal{R}$ be a local ring over $\mathbb{F}$ and $p, g, h \in \mathcal{R}[x_1, \ldots, x_n]$ be multivariate polynomials such that $\phi(g)$ and $\phi(h)$ are coprime over $\mathbb{F}$. If $p = 0 \bmod g$ and $p = 0 \bmod h$ then $p = 0 \bmod gh$.*

Since $\phi(S_j) = \ell_j$ and ℓ_i, ℓ_j are coprime for $i \neq j$, the correctness of Step 3 follows. Finally notice that, $p = 0 \bmod S_j$ if and only if $\sigma(p) = 0 \bmod \sigma(S_j)$ as σ is an invertible linear transformation. The check $\sigma(p) = 0 \bmod \sigma(S_j)$ is done recursively in Step 3.2. The recursion is on the top fanin, as $\sigma(p) = \sum_{i=2}^{k'} \beta_i T'_i$ over the local ring $\mathcal{R}'$ so that the top fanin of $\sigma(p)$ is lowered to $k' - 1$.

Complexity of the Algorithm

Note that, $\deg(T'_i) \leq \deg(T_i)$ in Step 3.2. At the start, Algorithm **ID** is called on polynomial $p(C)$. So, at every intermediate level $\deg(S_j) \leq \deg(T_1) \leq d$. Therefore, $\dim_{\mathbb{F}}(\mathcal{R}') \leq d \cdot \dim_{\mathbb{F}}(\mathcal{R})$. Time spent by Algorithm **ID** is at most $\mathsf{poly}(n, k', d, \dim_{\mathbb{F}}(\mathcal{R}))$ in Steps 1, 2, 3.1 and 4. Whereas, time spent in Step 3.2 is at most d times a smaller problem with top fanin reduced by 1 while dimension of the underlying local ring raised by a factor of at most d. Unfolding the recursion, we get the time complexity of Algorithm **ID** on input $p(C)$ to be $\mathsf{poly}(n, d^k)$.

Remark on Step 3.1 **-** In (KS07), the linear transformation σ is described as a map that takes ℓ_j to some fixed variable x_1 and transforms the remaining variables $x_2, \ldots, x_n$ accordingly so that σ is invertible. In our case, we also need the property that σ maps only one variable to a general linear form, whereas any other variable is mapped to a variable only. To stress upon this property, we have defined σ in a slightly different way. We will need this attribute of σ, in Section 6.3.2, to ensure that certain 'semidiagonal' structure of the given sparse polynomial f is preserved at every intermediate stage of the algorithm.

6.3.2 The Generalization

With the description of the Kayal-Saxena algorithm at hand, and equipped with the tool of dual representation (Section 6.2), it is now easier for us to point out the changes one needs to make in Algorithm **ID** so that it solves Problem 6.1.1 when applied to polynomial $p(C) - f$.

We will work with lexicographic ordering among monomials with $x_1 \succ \ldots \succ x_n$. Let the number of monomials with nonzero coefficients in f be s. At an intermediate level of the recursion, Algorithm **ID** tests if a polynomial of the form,

$$p = \alpha_1 T_1 + \ldots + \alpha_{k'} T_{k'} + \beta_1 \omega_1 + \ldots + \beta_{s'} \omega_{s'}$$

is zero over a local ring $\mathcal{R}$, where $k' \leq k$, $s' \leq s$ and α_i, β_j are elements of $\mathcal{R}$. As before, every T_i is a product of linear forms over $\mathcal{R}$. And each ω_r is a product of a monomial (in the free variables over $\mathcal{R}$) and at most $k - k'$ powers of distinct linear forms over $\mathcal{R}$. Further, it can be ensured that $\deg(T_i)$ and $\deg(\omega_r)$ are bounded by d.

Adapting Algorithm ID - Let us apply Algorithm **ID** to $p(C) - f$ and see what happens. The part $\beta_1 \omega_1 + \ldots + \beta_{s'} \omega_{s'} = \tilde{f}$ (say), in the above expression, shows how the polynomial f in $p(C) - f$ evolves with different levels of the recursion. At the beginning, $\tilde{f} = -f$.

In Step 1 of Algorithm **ID**, we are required to find a term T_1 such that $LM(T_1) \succeq LM(T_i)$ for all $2 \leq i \leq k'$. The purpose of this step is to ensure that $LM(T_1) \succeq LM(p)$, something we need for arguing the correctness of the algorithm. But now, our polynomial p has an added term $\tilde{f}$. So, to ensure $LM(T_1) \succeq LM(p)$, we also need to find the leading monomial of $\tilde{f}$. We will show in a short while how to find $LM(\tilde{f})$. Suppose that we know $LM(\tilde{f})$. If $LM(\tilde{f}) \succ LM(T_i)$ for all $1 \leq i \leq k'$ then surely $p \neq 0$ over $\mathcal{R}$ and the algorithm returns NO. Otherwise, there is a T_1 such that $LM(T_1) \succeq LM(p)$ and Algorithm **ID** proceeds to Step 2 with that T_1.

In Step 2, the base case is no longer $k' = 1$ but instead $k' = 0$. In this case, we have to check if $\tilde{f} = 0$ in $\mathcal{R}$ and this can be done efficiently since we will know shortly how to find $LM(\tilde{f})$.

Step 3 remains the same as before, except that in Step 3.2 we also need to find $\sigma(\tilde{f})$, where σ is the linear transformation. Notice that, σ maps only x_u to a general

linear form and keeps other free variables intact. So, the product term $\sigma(\omega_r)$ has at most one power of a linear form more than that of ω_r (as x_u gets replaced by $\sigma(x_u)$). Since the depth of the recursion of Algorithm **ID** is at most k, every ω_r in $\tilde{f}$ is the product of a monomial and at most k powers of linear forms over $\mathcal{R}$. In other words, $\tilde{f}$ is a semidiagonal polynomial over $\mathcal{R}$ (by extending Definition 6.1.4 to semidiagonal polynomials over rings).

Finally, in Step 4, we need to confirm that Coeff$(LM(T_1))$ in p is zero. For this, we may have to find the coefficient of $LM(\tilde{f})$ in $\tilde{f}$, if $LM(\tilde{f})$ happens to be the same as $LM(T_1)$. This is taken care of by the way we find $LM(\tilde{f})$ which also gives us its coefficient.

We now show how to find leading monomial of $\tilde{f}$.

Dual representation of $\tilde{f}$ - Let $\ell^{d'}$ be a term occuring in the product ω_r, where $\ell = a_1 x_{v_1} + \ldots + a_n x_{v_{n'}} + m$ is a linear form over $\mathcal{R}$. (Assume that $x_{v_1}, \ldots x_{v_{n'}}$ are the free variables over $\mathcal{R}$). Replace m by a formal variable z and use Theorem 6.2.1 to express $(a_1 x_{v_1} + \ldots + a_n x_{v_{n'}} + z)^{d'}$ as,

$$(a_1 x_{v_1} + \ldots + a_n x_{v_{n'}} + z)^{d'} = \sum_{i=1}^{(n'+1)d'+1} g_i(z) \cdot g_{i1}(x_{v_1}) \ldots g_{in'}(x_{v_{n'}}),$$

where g_i and g_{ij} are univariates over $\mathbb{F}$ of degree at most d'. Therefore, $\ell^{d'}$ can be expressed as,

$$\ell^{d'} = \sum_{i=1}^{(n'+1)d'+1} \gamma_i \cdot g_{i1}(x_{v_1}) \ldots g_{in'}(x_{v_{n'}}), \tag{6.1}$$

where $\gamma_i = g_i(m) \in \mathcal{R}$, in time $\mathsf{poly}(n, d, \dim_{\mathbb{F}} \mathcal{R})$. Since ω_r is a product of a monomial (in $x_{v_1}, \ldots x_{v_{n'}}$) and at most k products of powers of linear forms over $\mathcal{R}$, using Equation 6.1, each ω_r can be expressed as,

$$\omega_r = \sum_{i=1}^{O((nd)^k)} \gamma_i \cdot g_{i1}(x_{v_1}) \ldots g_{in'}(x_{v_{n'}}),$$

(reusing symbols γ_i and g_{ij}) where $\gamma_i \in \mathcal{R}$ and g_{ij} is a polynomial over $\mathbb{F}$ of degree $O(kd)$, in time $\mathsf{poly}((nd)^k, \dim_{\mathbb{F}} \mathcal{R})$.

Therefore, given the polynomial $\tilde{f}$,

$$\text{dual } (\tilde{f}) = \sum_{i=1}^{O(s\cdot(nd)^k)} \gamma_i \cdot g_{i1}(x_{v_1}) \dots g_{in'}(x_{v_{n'}}),$$

(again reusing symbols γ_i and g_{ij}) can be computed in time $\mathsf{poly}(s, (nd)^k, \dim_{\mathbb{F}} \mathcal{R})$.

Finding leading monomial of $\tilde{f}$ - Let $\{e_1, \dots, e_{\dim_{\mathbb{F}} \mathcal{R}}\}$ be an $\mathbb{F}$-basis of $\mathcal{R}$. In the sum,

$$\text{dual } (\tilde{f}) = \sum_{i=1}^{O(s\cdot(nd)^k)} \gamma_i \cdot g_{i1}(x_{v_1}) \dots g_{in'}(x_{v_{n'}}),$$

let $\gamma_i = \sum_{j=1}^{\dim_{\mathbb{F}} \mathcal{R}} b_{ij} e_j$, where $b_{ij} \in \mathbb{F}$. Consider the polynomials,

$$q_j = \sum_{i=1}^{O(s\cdot(nd)^k)} b_{ij} \cdot g_{i1}(x_{v_1}) \dots g_{in'}(x_{v_{n'}}),$$

for all $1 \leq j \leq \dim_{\mathbb{F}} \mathcal{R}$. Then, the leading monomial of $\tilde{f}$ is the highest among the leading monomials of the polynomials q_j, with respect to the monomial ordering. From Theorem 6.2.2, the leading monomial (and its coefficient) of every q_j can be computed in time $\mathsf{poly}(s, (nd)^k)$ and so $LM(\tilde{f})$ can also be found in time $\mathsf{poly}(s, (nd)^k, \dim_{\mathbb{F}} \mathcal{R})$.

Using an analysis similar to the complexity analysis of Algorithm **ID** before and observing that $\dim_{\mathbb{F}} \mathcal{R} \leq d^k$, we can show that Algorithm **ID** adapted to work for $p(C) - f$ takes time $\mathsf{poly}(s, (nd)^k)$. This solves Problem 6.1.1 in deterministic polynomial time. Note that, the complexity remains polynomial time even if f is a sparse semidiagonal polynomial over $\mathbb{F}$ in Problem 6.1.1.

6.4 Checking Sparse Polynomial Factorization

We have mentioned in Chapter 5 (Section 5.1) that solving depth-4 identity testing is almost as hard as solving the general problem of identity testing. One special, but natural case of depth-4 identity testing is given by Problem 6.1.2, where we need to verify the validity of a given sparse polynomial factorization. That is, given

polynomials f and $g_1, \ldots, g_t$ explicitly, check if $f = g_1 \ldots g_t$. This problem is also mentioned in a work by von zur Gathen (vzG83) and we are not aware of any progress in derandomizing it. Note that, the naive approach of multiplying all the factors could be infeasible because of intermediate swelling in the number of monomials. In the view of a lack of progress, we investigate a special case of this problem, which is Problem 6.1.3, where every polynomial g_i is of the form of a sum of univariates. The motivation for considering this special kind of g_i actually comes from the case when g_i's are linear functions. Consider the following example. Let

$$f = (x_1^d - 1) \cdot (x_2^d - 1) \ldots (x_n^d - 1)$$

be the input polynomial that has $s = 2^n$ monomials. Suppose that the factors given to us are,

$$(x_1 - 1), (x_1 - \zeta), \ldots, (x_1 - \zeta^{d-1}), \ldots, (x_n - 1), (x_n - \zeta), \ldots, (x_n - \zeta^{d-1})$$

where ζ is a primitive d^{th} root of unity, and we want to check if f equals the product of these factors. If we do not follow any particular rule in multiplying these factors then we may as well end up with the intermediate product,

$$p = (x_1^{d-1} + x_1^{d-1} + \ldots + 1) \ldots (x_n^{d-1} + x_n^{d-1} + \ldots + 1),$$

which has $d^n = s^{\log d}$ monomials. Thus, given f with s monomials we might have to spend time $O(s^{\log d})$ if we follow this naive approach. However, with a deterministic polynomial time solution to Problem 6.1.3 we can solve this example problem in $\mathsf{poly}(s, d)$ time.

When g_i's are linear functions, Problem 6.1.3 become a special case of Problem 6.1.1 and can therefore be solved in deterministic polynomial time. However, the approach given in Section 6.3 does not seem to generalize directly to the case when, instead of linear functions, g_i's are sums of univariates. It is this latter case that we solve in this section. As a note to the reader we would like to mention that polynomials of the kind of sums of univariates also appear in the work of Saxena (Sax08) and Kayal (Kay10).

Throughout this section, we will assume that either $\mathrm{char}(\mathbb{F}) = 0$ or $\mathrm{char}(\mathbb{F})$ is greater than the total degree of the input polynomial f. Let, $X = \{x_1, \ldots, x_n\}$ be the set of variables.

6.4.1 Reduction to Sparse Divisibility

Given f and $g_1, \ldots, g_t$, group together the polynomials g_i's that are just $\mathbb{F}$-multiples of each other. After this is done, we need to check if f is equal to a product of the form $a \cdot g_1^{d_1} \ldots g_t^{d_t}$ (reusing symbol t), where $a \in \mathbb{F}$ and $g_i \neq b \cdot g_j$ for any $b \in \mathbb{F}$ if $i \neq j$. Suppose that g_i and g_j are irreducible for $i \neq j$. (This assumption will be justified later in Section 6.4.2). Then, Problem 6.1.3 gets reduced to the problem of divisibility of sparse polynomials as follows.

Since g_i and g_j are coprime, if we can confirm that $g_i^{d_i}$ divides f for all $1 \leq i \leq t$ then by unique factorization (or Chinese Remaindering), $f = 0 \bmod g_1^{d_1} \ldots g_t^{d_t}$. Fix a monomial ordering and notice that $LM(g_i^{d_i}) = LM(g_i)^{d_i}$. Then $f = b \cdot g_1^{d_1} \ldots g_t^{d_t}$ for some $b \in \mathbb{F}$ if and only if $LM(f) = LM(g_1^{d_1}) \ldots LM(g_t^{d_t})$. Finally, compare the coefficients of the leading monomials of f and $g_1^{d_1} \ldots g_t^{d_t}$ to conclude that $f = a \cdot g_1^{d_1} \ldots g_t^{d_t}$.

But, how do we check if $g_i^{d_i}$ divides f. We cannot possibly expand out $g_i^{d_i}$ completely as it may not be sparse. This problem can be circumvented by using the following observation.

Observation 6.4.1. *Let f and g be multivariate polynomials and g be irreducible. Suppose x is a variable such that $\frac{\partial f}{\partial x} \neq 0$ and $\frac{\partial g}{\partial x} \neq 0$. Then, g^d divides f if and only if g divides f and g^{d-1} divides $\frac{\partial f}{\partial x}$.*

Proof. Suppose g^d divides f, then $f = g^d \cdot h$ for some polynomial h. Therefore, $\frac{\partial f}{\partial x} = g^d \frac{\partial h}{\partial x} + d \cdot g^{d-1} \frac{\partial g}{\partial x} h$, which is clearly divisible by g^{d-1}.

As for the other direction, suppose $f = g^e \cdot h$ with $e \geq 1$ (assuming $g \mid f$) and let $g \nmid h$. Then

$$\begin{aligned} f' = \frac{\partial f}{\partial x} &= g^e \cdot \frac{\partial h}{\partial x} + e g^{e-1} \frac{\partial g}{\partial x} \cdot h \\ &= g^{e-1} \cdot \left(g \frac{\partial h}{\partial x} + e \frac{\partial g}{\partial x} h \right) \end{aligned}$$

Since, $\mathrm{char}(\mathbb{F}) \geq \deg(f) \geq e$, g does not divide $\left(g \frac{\partial h}{\partial x} + e \frac{\partial g}{\partial x} h\right)$. Therefore, $g^{d-1} \mid f'$ implies that $e \geq d$, in other words $g^d \mid f$. □

Notice that, since f is a sparse polynomial, $\frac{\partial f}{\partial x_j}$ is also sparse and easily computable from an explicitly given f, for every $1 \leq j \leq n$. So, by the above observation, the

check "if $g_i^{d_i}$ divides f" reduces to d_i checks of the kind "if g_i divides $\tilde{f}$" where $\tilde{f}$ is a sparse polynomial. What remains to be answered is - how do we check divisibility of a sparse polynomial by a sum of univariates g_i. Before going into that let us at first justify our assumption that g_i is irreducible.

6.4.2 Irreducibility of Sums of Univariates

In this section, we prove the following theorem. Denote by $\bar{\mathbb{F}}$ the algebraic closure of field $\mathbb{F}$.

Theorem 6.4.2. *Let g be a polynomial over a field $\mathbb{F}$ that is a sum of univariates. If g depends on at least three variables then g is irreducible over $\mathbb{F}$, assuming* $\mathrm{char}(\mathbb{F}) = 0$ *or* $\mathrm{char}(\mathbb{F}) > \deg(g)$. *(In other words, g is absolutely irreducible over any such $\mathbb{F}$.)*

Proof. We need the following observation to prove the theorem. Assume that g depends on at least three variables, say x_1, x_2 and x_3.

Observation 6.4.3. *Let $g = u \cdot v$ where u and v are nontrivial factors. Then both u and v are monic in every variable that g depends on.*

Proof. If u is not monic in, say, x_i then fix an ordering amongst the variables so that x_i is the highest. Then, the product $u \cdot v$ has its highest degree term as a mixed term which is not possible. □

If $g = u \cdot v$ is a nontrivial factorization of g then,

$$\frac{\partial g}{\partial x_1} = \frac{\partial u}{\partial x_1} v + \frac{\partial v}{\partial x_1} u. \tag{6.2}$$

We can assume that degree of x_1 in $\frac{\partial g}{\partial x_1}$ is not zero as otherwise degree of x_1 in g is one and by Observation 6.4.3, g is irreducible. Denote by $h_{x_i=\alpha}$ the polynomial h evaluated at $x_i = \alpha$, where $\alpha \in \bar{\mathbb{F}}$.

Claim 6.4.4. *There exists an $\alpha \in \bar{\mathbb{F}}$ such that $u_{x_1=\alpha}$ and $v_{x_1=\alpha}$ are both non-zero and they share a nontrivial factor.*

Proof. Suppose $g' = \frac{\partial g}{\partial x_1}$, $u' = \frac{\partial u}{\partial x_1}$ and $v' = \frac{\partial v}{\partial x_1}$, and let $w = \gcd(g', u', v')$ which is a univariate in x_1 as g' is a univariate. From Equation 6.2,

$$\frac{g'}{w} = \left(\frac{u'}{w}\right) v + \left(\frac{v'}{w}\right) u$$

Since x_1-degree of u' is less than that of g', the univariate g'/w has degree in x_1 at least one. Let $\alpha \in \bar{\mathbb{F}}$ be a root of g'/w. Substituting $x_1 = \alpha$ in the above expression, we get an equation of the form,

$$\tilde{u} \cdot v_{x_1=\alpha} + \tilde{v} \cdot u_{x_1=\alpha} = 0$$

where $\tilde{u} = \left(\frac{u'}{w}\right)_{x_1=\alpha}$ and $\tilde{v} = \left(\frac{v'}{w}\right)_{x_1=\alpha}$. The above equation is nontrivial as none of $\tilde{u} \cdot v_{x_1=\alpha}$ and $\tilde{v} \cdot u_{x_1=\alpha}$ is zero. This is because none of the polynomials $u_{x_1=\alpha}$ and $v_{x_1=\alpha}$ can be zero by Observation 6.4.3. Also, this means that none of $\tilde{u}$ and $\tilde{v}$ is zero, as g'/w, u'/w and v'/w do not share any common factor, in particular $x_1 - \alpha$. Now notice that, since u is monic in x_2 (by Observation 6.4.3), degree of x_2 in u' is strictly less than that in u. Which means, degree of x_2 in $\tilde{u}$ is also strictly less than degree of x_2 in $u_{x_1=\alpha}$. Therefore, by the above equation, $u_{x_1=\alpha}$ and $v_{x_1=\alpha}$ must share a nontrivial factor. □

Let us see how the above claim implies the theorem. Consider the following relations,

$$\begin{aligned} & & \frac{\partial g}{\partial x_2} &= \frac{\partial u}{\partial x_2} v + \frac{\partial v}{\partial x_2} u \\ \implies & \left.\frac{\partial g}{\partial x_2}\right|_{x_1=\alpha} = \frac{\partial g}{\partial x_2} &= \left.\frac{\partial u}{\partial x_2}\right|_{x_1=\alpha} \cdot v_{x_1=\alpha} + \left.\frac{\partial v}{\partial x_2}\right|_{x_1=\alpha} \cdot u_{x_1=\alpha} \\ & \text{Similarly,} \quad \frac{\partial g}{\partial x_3} &= \left.\frac{\partial u}{\partial x_3}\right|_{x_1=\alpha} \cdot v_{x_1=\alpha} + \left.\frac{\partial v}{\partial x_3}\right|_{x_1=\alpha} \cdot u_{x_1=\alpha} \end{aligned}$$

Since both $\frac{\partial g}{\partial x_2}$ and $\frac{\partial g}{\partial x_3}$ are non-zero, the last two equations force the gcd $(u_{x_1=\alpha}, v_{x_1=\alpha})$ (which is nontrivial by Claim 6.4.4) to divide $\frac{\partial g}{\partial x_2}$ and $\frac{\partial g}{\partial x_3}$. But this leads to a contradiction since the partial derivatives are univariates in x_2 and x_3 respectively. This completes the proof of the theorem. □

Note that, Theorem 6.4.2 cannot hold in general when g is a bivariate. For instance, the polynomial $x_1^d - x_2^d$ is divisible by $x_1 - x_2$ over any field.

Remark - Ehrenfeucht and Pelczynski proved in an unpublished work that polynomials of the form $f(x) + g(y) + h(z)$ are irreducible over fields of characteristic zero. We came to know about their work from here (Dav83) (page 221) after having independently produced the above proof of Theorem 6.4.2. We are not quite aware if the original argument by Ehrenfeucht and Pelczynski is the same as ours, which going by the not so difficult nature of our proof could as well be the same.

6.4.3 From Sparse Divisibility to Identity Testing

Handling bivariates and univariates - In Section 6.4.1, we have assumed that the polynomials $g_1, \ldots, g_t$ are irreducible. Although, Theorem 6.4.2 justifies our assumption when g_i depends on three or more variables, we need to slightly change our strategy for bivariates and univariates. In this case, we first take pairwise gcd of the bivariates (similarly, pairwise gcd of the univariates) and factorize accordingly till all the bivariates (similarly, the univariates) are mutually coprime. Computing the gcd of two bivariate polynomials takes only polynomial time using Hensel lifting. Once coprimeness is ensured, we can directly check if a bivariate $g_i^{d_i}$ divides f by expressing f as $f = \sum_j f_j m_j$, where f_j's are bivariate polynomials depending on the same two variables as g_i and m_j's are monomials in the remaining variables. Then, $g_i^{d_i} \mid f$ if and only if $g_i^{d_i} \mid f_j$ for all j. Once again, just like gcd, bivariate divisibility is also a polynomial time computation. Finally, we can use Chinese remaindering to complete the argument in a similar fashion as in Section 6.4.1.

By the reduction of Section 6.4.1, it remains to show that checking if a sparse polynomial f is divisible by a sum of univariates g can be done in deterministic polynomial time. For this, we once again use the tool of dual representation of polynomials.

Let us extend Definition 6.1.4 and call a polynomial $q = \sum_{i=1}^{m} Q_i$ a *general* semidiagonal polynomial if every Q_i is a product of a monomial and constantly many powers of sums of univariates. Using this extended definition, we show the following.

Theorem 6.4.5. *Checking divisibility of a sparse polynomial f by a sum of univariates g reduces to identity testing of a general semidiagonal polynomial.*

Proof. Let $g = \sum_{i=1}^{n} u_i(x_i)$, where u_i is a univariate in x_i. Assume without loss of generality that $u_1 \neq 0$. Consider replacing the partial sum $\sum_{i=2}^{n} u_i(x_i)$ in g by a new variable y so that we get a bivariate $h = u_1(x_1) + y$. Let $\deg_{x_1} u_1 = e$. Given any power of x_1, say x_1^d, we can employ long division with respect to the variable x_1 to find the remainder when x_1^d is divided by h. This division is possible since h is monic in x_1. It is not hard to see that the remainder, say r_d, thus obtained is a bivariate in x_1 and y with degree in x_1 less than e and degree in y at most d.

To check if g divides f, do the following. For every monomial of f, replace the power of x_1 occuring in the monomial, say x_1^d, by the corresponding remainder r_d. After the replacement process is over, completely multiply out terms in the resulting polynomial, say $\tilde{f}(x_1, x_2, \ldots, x_n, y)$, and express it as sum of monomials in the variables $x_1, \ldots, x_n$ and y. Now notice that,

$$f \bmod g = \tilde{f}(x_1, x_2, \ldots, x_n, \sum_{i=2}^{n} u_i(x_i))$$

Since degree of x_1 in $\tilde{f}$ is at most e, polynomial g divides f if and only if the polynomial $\tilde{f}(x_1, x_2, \ldots, x_n, \sum_{i=2}^{n} u_i(x_i))$ is identically zero. Polynomial $\tilde{f}$ with y evaluated at $\sum_{i=2}^{n} u_i(x_i)$ is a general semidiagonal polynomial. Also, verify that the above reduction takes only polynomial time. □

Checking if $\tilde{f}(x_1, x_2, \ldots, x_n, \sum_{i=2}^{n} u_i(x_i)) = 0$:- When we evaluate $\tilde{f}$ at $y = \sum_{i=2}^{n} u_i(x_i)$, every monomial in $\tilde{f}$ becomes a product of a monomial in $x_1, \ldots, x_n$ and a power of $\sum_{i=2}^{n} u_i(x_i)$. Now use Theorem 6.2.1 to express this power of $\sum_{i=2}^{n} u_i(x_i)$ as a sum of product of univariates. Then multiply out terms to express $\tilde{f}(x_1, x_2, \ldots, x_n, \sum_{i=2}^{n} u_i(x_i))$ as a sum of product of univariates. Finally, use Theorem 6.2.2 to do identity testing on this expression of $\tilde{f}(x_1, x_2, \ldots, x_n, \sum_{i=2}^{n} u_i(x_i))$. It is not hard to see that the whole process takes time $\mathsf{poly}(n, d, s)$, where d is the bound on the degree and s is the bound on the sparsity of the input polynomials f and $g_1, \ldots, g_t$.

This concludes our solution to Problem 6.1.3.

6.5 Conclusion

In this chapter, we have used the trick of expressing polynomials in dual representations to solve two very natural problems on identity testing. We have shown that it can be efficiently checked if a given depth-3 polynomial with bounded top fanin equals a given sparse polynomial, thereby extending the Kayal-Saxena test. However, our algorithm is not blackbox in nature as it heavily relies on explicit information about the input polynomials. On the other hand, due to a recent development (SS10a), it is now known how to check if $p(C) = 0$ in blackbox polynomial time over any field. We wonder if the present techniques can be used to give a blackbox solution to Problem 6.1.1 as well. We leave this as an open question.

- Find a blackbox deterministic polynomial time algorithm to solve Problem 6.1.1.

We have shown that checking sparse polynomial factorization can be done efficiently if the input factors are sums of univariates. Another natural case that we leave open here is the following.

- Can we solve Problem 6.1.2 in deterministic polynomial time if the degrees of the input factors $g_1, \ldots, g_t$ are bounded?

It would be interesting to see if duality yields anything useful for this case.

Further, notice that it follows directly from Theorem 6.4.2 that identity testing of a depth-4 circuit with top fanin 2 can be solved in polynomial time if each of the two '$\times$' gates is just a product of sums of univariates. It is natural to ask the following.

- Can we solve identity testing of depth-4 circuits with bounded top fanin k in deterministic polynomial time if each of the k multiplications is a product of sums of univariates?

We also leave this as an open problem.

Appendix A

Appendix

A.1 The Resultant

Let $\mathcal{R}$ be an integral domain and $\mathbb{F}$ be its field of fractions. Let f and g be two polynomials in $\mathcal{R}[x]$ of degree n and m, respectively. Assume that the $\gcd(f, g)$ is the unique, monic largest common divisor of f and g over $\mathbb{F}$.

Lemma A.1.1. *The* $\gcd(f, g)$ *is nontrivial if and only if there exists polynomials* $s, t \in \mathbb{F}[x]$, *with* $\deg(s) < m$ *and* $\deg(t) < n$, *such that* $sf + tg = 0$.

Proof. Suppose $h = \gcd(f, g)$. If $\gcd(f, g) \neq 1$ then $\deg(h) > 1$. Now, if we take $s = \frac{g}{h}$ and $t = -\frac{f}{h}$ then $\deg(s) < m, \deg(t) < n$ and $sf + tg = 0$.

To show the other direction, suppose that there exist s and t with $\deg(s) < m, \deg(t) < n$ and $sf + tg = 0$. If the $\gcd(f, g) = 1$ then by unique factorization over $\mathbb{F}$, g should divide s. But, this is not possible as $\deg(s) < \deg(g)$. Hence the $\gcd(f, g)$ is nontrivial. □

Let $f = \sum_{i=0}^{n} f_i x^i$ and $g = \sum_{j=0}^{m} g_j x^j$, where $f_i, g_j \in \mathbb{F}$ for $0 \leq i \leq n$ and $0 \leq j \leq m$. Suppose $s = \sum_{k=0}^{m-1} \alpha_k x^k$ and $t = \sum_{\ell=0}^{n-1} \beta_\ell x^\ell$. Treat the coefficients $\alpha_0, \ldots, \alpha_{m-1}$ and $\beta_0, \ldots, \beta_{n-1}$ as variables. Now, consider the relation $sf + tg = 0$. By multiplying s, f and t, g, and then equating the coefficients of x^i to zero for all $0 \leq i \leq n + m - 1$, we get a system of $n + m$ homogeneous linear equations in the variables $\alpha_0, \ldots, \alpha_{m-1}, \beta_0, \ldots, \beta_{n-1}$. The coefficient matrix of this linear system is called the Sylvester matrix of f and g, and is denoted by $S(f, g)$. It is easy to verify that $S(f, g)$ is the following $(n + m) \times (n + m)$ matrix.

$$S(f,g) = \begin{pmatrix}
f_n & & & & g_m & & & & & \\
f_{n-1} & f_n & & & g_{m-1} & g_m & & & & \\
\vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & & \\
\vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & & \\
\vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \ddots & \\
f_1 & \vdots & & \vdots & & g_0 & & & & g_m \\
f_0 & \vdots & & \vdots & & & \ddots & & & \vdots \\
 & f_0 & & \vdots & & & & \ddots & & \vdots \\
 & & \ddots & \vdots & & & & & \ddots & \vdots \\
 & & & f_0 & & & & & & g_0
\end{pmatrix}.$$

The resultant of f and g is defined as, $\mathsf{Res}_x(f,g) = \det(S(f,g))$. By Lemma A.1.1, the above linear system has a nonzero solution if and only if $\gcd(f,g)$ is nontrivial. This has the following implication.

Lemma A.1.2. *The* $\gcd(f,g)$ *is nontrivial if and only if* $\mathsf{Res}_x(f,g) = \det(S(f,g)) = 0$.

Another useful fact about the resultant is the following.

Lemma A.1.3. *There exist* $s,t \in \mathcal{R}[x]$*, with* $\deg(s) < m$ *and* $\deg(t) < n$*, such that* $sf + tg = \mathsf{Res}_x(f,g)$.

Proof. If the $\gcd(f,g) \neq 1$, then from Lemma A.1.1 and A.1.2 it follows that, there exist $s',t' \in \mathbb{F}[x]$, with $\deg(s') < m$ and $\deg(t') < n$, such that $s'f + t'g = 0 = \mathsf{Res}_x(f,g)$. Since a coefficient of s' or t' is of the form $\frac{a}{b}$, where $a,b \in \mathcal{R}$ and $b \neq 0$, by clearing out the denominators of the coefficients of s' and t' we get $s,t \in \mathcal{R}[x]$ such that $sf + tg = 0$. Clearly, $\deg(s) = \deg(s') < m$ and $\deg(t) = \deg(t') < n$.

Suppose, the $\gcd(f,g) = 1$. By extended Euclidean algorithm, there exist $s',t' \in \mathbb{F}[x]$, with $\deg(s') < m$ and $\deg(t') < n$, such that $s'f + t'g = 1$. Let $s' = \sum_{k=0}^{m-1} \alpha_k x^k$ and $t' = \sum_{\ell=0}^{n-1} \beta_\ell x^\ell$. Once again, by multiplying s', f and t', g, and then equating the coefficients of x^i for $0 \leq i \leq n+m-1$, we get a linear system in $\alpha_0, \ldots, \alpha_{m-1}, \beta_0, \ldots, \beta_{n-1}$ with $S(f,g)$ as the coefficient matrix. By Cramer's rule, the polynomials $s = \mathsf{Res}_x(f,g) \cdot s'$ and $t = \mathsf{Res}_x(f,g) \cdot t'$ both belong to $\mathcal{R}[x]$. Therefore, $sf + tg = \mathsf{Res}_x(f,g)$. □

A.2 Ben-Or and Cleve's Result

For the sake of completeness, we provide a proof of the result by Ben-Or and Cleve (BC88).

Theorem A.2.1. *(BC88) Let E be an arithmetic formula of depth d with fan-in (of every gate) bounded by 2. Then, there exists a sequence of 3×3 matrices, whose entries are either variables or constants, of length at most 4^d such that one of the entries of their product is E.*

Proof. The proof is by induction on the structure of E. The base case when $E = c \cdot x_i$ is computed as,

$$\begin{bmatrix} 1 & & \\ & 1 & \\ c \cdot x_i & & 1 \end{bmatrix}$$

Suppose $E = f_1 + f_2$ and that we have inductively constructed sequences computing f_1 and f_2. Then the following equation gives a sequence for E.

$$\begin{bmatrix} 1 & & \\ & 1 & \\ f_1 & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ f_2 & & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ f_1 + f_2 & & 1 \end{bmatrix}$$

If $E = f_1 \cdot f_2$, then the following sequence computes E

$$\begin{bmatrix} 1 & & \\ -f_2 & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & f_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ f_2 & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & -f_1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ f_1 f_2 & & 1 \end{bmatrix}$$

Applying the above two equations inductively, it is clear that E can be computed by a sequence of length at most 4^d. □

References

[AB99] Manindra Agrawal and Somenath Biswas. Primality and Identity Testing via Chinese Remaindering. In *FOCS*, pages 202–209, 1999. 4, 78

[Agr05] Manindra Agrawal. Proving Lower Bounds Via Pseudo-random Generators. In *FSTTCS*, pages 92–105, 2005. 4, 26, 78

[AJMV98] Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-Commutative Arithmetic Circuits: Depth Reduction and Size Lower Bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998. 81

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of Mathematics*, 160(2):781–793, 2004. 3, 4, 78

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof Verification and the Hardness of Approximation Problems. *Journal of the ACM*, 45(3):501–555, 1998. 4, 78

[AM69] M.F. Atiyah and I.G. MacDonald. *Introduction to Commutative Algebra*. Addison-Wesley Publishing Company, 1969. 19

[AMM77] Leonard M. Adleman, Kenneth L. Manders, and Gary L. Miller. On Taking Roots in Finite Fields. In *FOCS*, pages 175–178, 1977. 31

[AMS08] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New Results on Noncommutative and Commutative Polynomial Identity Testing. In *IEEE Conference on Computational Complexity*, pages 268–279, 2008. 82

[Ank52] Nesmith C. Ankeny. The Least Quadratic Nonresidue. *Annals of Mathematics*, 55:65–72, 1952. 40

[Art91] Michael Artin. *Algebra*. Prentice Hall, United States, 1991. 11

[AV08] Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008. 78

[Bac82] Eric Bach. Fast Algorithms under the Extended Riemann Hypothesis: A Concrete Estimate. In *STOC*, pages 290–295, 1982. 40

[Bak07] Alan J. Baker. An introduction to p-adic numbers and p-adic analysis. *Online Notes*, 2007. http://www.maths.gla.ac.uk/~ajb/dvips/padicnotes.pdf. 75, 76

[BC88] Michael Ben-Or and Richard Cleve. Computing Algebraic Formulas Using a Constant Number of Registers. In *STOC*, pages 254–257, 1988. 81, 121

[Ber67] Elwyn Berlekamp. Factoring Polynomials Over Finite Fields. *Bell System Technical Journal*, 46:1853–1859, 1967. 3

[Ber70] E. R. Berlekamp. Factoring Polynomials Over Large Finite Fields. *Mathematics of Computation*, 24(111):713–735, 1970. 3, 29, 30

[BvzGL95] E. Bach, J. von zur Gathen, and H. Lenstra. Deterministic factorization of polynomials over special finite fields. *Preprint*, 1995. 31

[BW05] Andrej Bogdanov and Hoeteck Wee. More on Noncommutative Polynomial Identity Testing. In *CCC*, pages 92–99, 2005. 82

[CDGK91] Michael Clausen, Andreas W. M. Dress, Johannes Grabmeier, and Marek Karpinski. On Zero-Testing and Interpolation of k-Sparse Multivariate Polynomials Over Finite Fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991. 4, 78

[CH00] Qi Cheng and Ming-Deh A. Huang. Factoring Polynominals over Finite Fields and Stable Colorings of Tournaments. *ANTS*, pages 233–246, 2000. 31, 32, 36

[CK97] Zhi-Zhong Chen and Ming-Yang Kao. Reducing Randomness via Irrational Numbers. In *STOC*, pages 200–209, 1997. 4, 78

[CP03] R. Crandall and J. Papadopoulos. On the implementation of AKS-class primality tests, March 2003. Available from `http://www.apple.com/acg/pdf/aks3.pdf`. 3

[CS04] Steve Chien and Alistair Sinclair. Algebras with Polynomial Identities and Computing the Determinant. In *FOCS*, pages 352–361, 2004. 82

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965. 23

[CZ81] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981. 30

[Dav83] James H. Davenport. Factorization of sparse polynominals. In *EUROCAL*, pages 214–224, 1983. 116

[DF99] David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley and Sons, Inc., second edition, 1999. 11

[DKSS08] Anindya De, Piyush P. Kurur, Chandan Saha, and Ramprasad Saptharishi. Fast integer multiplication using modular arithmetic. In *STOC*, pages 499–506, 2008. 7

[DS05] Zeev Dvir and Amir Shpilka. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. In *STOC*, pages 592–601, 2005. 8

[Evd92] Sergie Evdokimov. Factorization of solvable polynomials over finite fields and the generalized Riemann hypothesis. *Journal of Mathematical Sciences*, 59(3):842–849, 1992. 31, 40

[Evd94] Sergei Evdokimov. Factorization of polynominals over finite fields in subexponential time under GRH. In *ANTS*, pages 209–219, 1994. 5, 6, 31, 32, 36, 40, 42, 46, 49, 50, 51, 56

[Für07] Martin Fürer. Faster Integer Multiplication. In *STOC*, pages 57–66, 2007. 7, 58, 60, 63, 76

[Für09] Martin Fürer. Faster Integer Multiplication. *SIAM J. Comput.*, 39(3):979–1005, 2009. 58

[Gao01] Shuhong Gao. On the Deterministic Complexity of Factoring Polynomials. *J. Symb. Comput.*, 31(1/2):19–36, 2001. 5, 32, 34, 37, 42, 43, 54, 56

[GG03] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003. 19

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999. 3

[HB92] D. R. Heath-Brown. Zero-free regions for Dirichlet L-functions, and the least prime in an arithmetic progression. In *Proceedings of the London Mathematical Society, 64(3)*, pages 265–338, 1992. 62

[Hen18] Kurt Hensel. Eine neue Theorie der algebraischen Zahlen. *Mathematische Zeitschift*, 2:433–452, 1918. 21

[Her75] I.N. Herstein. *Topics in Algebra*. Wiley, second edition, 1975. 11

[Hua84] Ming-Deh A. Huang. Factorization of Polynomials over Finite Fields and Factorization of Primes in Algebraic Number Fields. In *STOC*, pages 175–182, 1984. 31

[Hua91] Ming-Deh A. Huang. Generalized Riemann Hypothesis and Factoring Polynomials over Finite Fields. *J. Algorithms*, 12(3):464–481, 1991. 31

[IKRS08] Gábor Ivanyos, Marek Karpinski, Lajos Rónyai, and Nitin Saxena. Trading GRH for algebra: algorithms for factoring polynomials and related structures. *CoRR*, abs/0811.3165, 2008. 30

[IKS09] Gábor Ivanyos, Marek Karpinski, and Nitin Saxena. Schemes for deterministic polynomial factoring. In *ISSAC*, pages 191–198, 2009. 31, 32

[Isa94] I. Martin Isaacs. *Character theory of finite groups*. Dover publications Inc., New York, 1994. 68

[Kay05] Neeraj Kayal. Solvability of a system of bivariate polynomial equations over a finite field. In *ICALP*, pages 551–562, 2005. 3

[Kay10] Neeraj Kayal. Algorithms for Arithmetic Circuits. *Technical Report TR10-073, (ECCC)*, 2010. 101, 102, 112

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, pages 355–364, 2003. 4, 26, 78

[KO63] A Karatsuba and Y Ofman. Multipication of multidigit numbers on automata. *English Translation in Soviet Physics Doklady*, 7:595–596, 1963. 58

[KS98] Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comput.*, 67(223):1179–1197, 1998. 3, 30

[KS01] Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC*, pages 216–223, 2001. 4, 78

[KS07] Neeraj Kayal and Nitin Saxena. Polynomial Identity Testing for Depth 3 Circuits. *Computational Complexity*, 16(2):115–138, 2007. 8, 9, 78, 99, 106, 108

[KS09] Neeraj Kayal and Shubhangi Saraf. Blackbox polynomial identity testing for depth-3 circuits. In *FOCS*, 2009. 8, 78

[KU08] Kiran S. Kedlaya and Christopher Umans. Fast Modular Composition in any Characteristic. In *FOCS*, pages 146–155, 2008. 3, 30

[KY08] Swastik Kopparty and Sergey Yekhanin. Detecting Rational Points on Hypersurfaces over Finite Fields. In *IEEE Conference on Computational Complexity*, pages 311–320, 2008. 3

[Lan85] Susan Landau. Factoring Polynomials Over Algebraic Number Fields. *SIAM J. Comput.*, 14(1):184–195, 1985. 31

[Lan02] Serge Lang. *Algebra*. Springer-Verlag, New York Inc., third edition, 2002. 11

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic Methods for Interactive Proof Systems. In *FOCS*, pages 2–10, 1990. 4, 78

[Lin44] Yuri V. Linnik. On the least prime in an arithmetic progression, I. The basic theorem, II. The Deuring-Heilbronn's phenomenon. *Rec. Math. (Mat. Sbornik)*, 15:139–178 and 347–368, 1944. 62

[LJL82] Arjen K. Lenstra, Hendrik W. Lenstra Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. 29, 31

[LN94] Rudolf Lidl and Harald Neiderreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1994. 29, 33

[Lov79] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979. 4, 78

[LV98] Daniel Lewin and Salil P. Vadhan. Checking Polynomial Identities over any Field: Towards a Derandomization? In *STOC*, pages 438–447, 1998. 4, 78

[Mil76] Gary L. Miller. Riemann's Hypothesis and Tests for Primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976. 75

[MS88] M. Mignotte and C.-P. Schnorr. cacul deteministe des racines d'un polynome dans un corps fini. *Comptes Rendus Academie des Sciences*, 306:467–472, 1988. 31

[Nis91] Noam Nisan. Lower bounds for non-commutative computation. In *STOC*, pages 410–418, 1991. 82, 87

[NW94] Noam Nisan and Avi Wigderson. Hardness vs Randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. 26

[Pla77] David A. Plaisted. Sparse Complex Polynomials and Polynomial Reducibility. *J. Comput. Syst. Sci.*, 14(2):210–221, 1977. 100

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128–138, 1980. 75

[Rón88] Lajos Rónyai. Factoring Polynomials Over Finite Fields. *J. Algorithms*, 9(3):391–400, 1988. 31

[Rón89] Lajos Rónyai. Factoring polynomials modulo special primes. *Combinatorica*, 9(2):199–206, 1989. 31

[Rón92] Lajos Rónyai. Galois groups and factoring over finite fields. *SIAM J. Discrete Math.*, 5:345–365, 1992. 31

[RS04] Ran Raz and Amir Shpilka. Deterministic Polynomial Identity Testing in Non-Commutative Models. In *CCC*, pages 215–222, 2004. 8, 82, 83, 100, 103

[Sah08] Chandan Saha. Factoring Polynomials over Finite Fields using Balance Test. In *STACS*, pages 609–620, 2008. 5

[Sax08] Nitin Saxena. Diagonal Circuit Identity Testing and Lower Bounds. In *ICALP (1)*, pages 60–71, 2008. 9, 100, 102, 103, 112

[Sch76] W. Schimidt. *Equations over Finite Fields.* Springer-Verlag Lecture Notes in Mathematics No. 536, 1976. 51

[Sch80] Jacob T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980. 4, 78, 79

[Sch82] Arnold Schönhage. Asymptotically Fast Algorithms for the Numerical Multiplication and Division of Polynomials with Complex Coefficients. In *Computer Algebra, EUROCAM*, volume 144 of *Lecture Notes in Computer Science*, pages 3–15, 1982. 74

[Sch85] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44:483–494, 1985. 30

[Sha90] Adi Shamir. IP=PSPACE. In *FOCS*, pages 11–15, 1990. 4, 78

[Sha99] Igor R. Shafarevich. *Basic Notions of Algebra.* Springer Verlag, USA, 1999. 68

[Sho90] Victor Shoup. On the Deterministic Complexity of Factoring Polynomials over Finite Fields. *Inf. Process. Lett.*, 33(5):261–267, 1990. 30, 56

[Sho91] Victor Shoup. Smoothness and Factoring Polynomials Over Finite Fields. *Information Processing Letters*, 38(1):39–42, 1991. 31

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, New York, 2009. Available from `http://shoup.net/ntb/`. 19

[Spe74] Joel H. Spencer. Random Regular Tournaments. *Periodica Mathematica Hungarica*, 5(2):105–120, 1974. 53

[SS71] A Schönhage and V Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971. 6, 7, 58, 59, 76

[SS09] Nitin Saxena and C. Seshadhri. An Almost Optimal Rank Bound for Depth-3 Identities. In *IEEE Conference on Computational Complexity*, pages 137–148, 2009. 8, 78

[SS10a] Nitin Saxena and C. Seshadhri. Blackbox identity testing for bounded top fanin depth-3 circuits: the field doesn't matter. *Technical Report TR10-167, (ECCC)*, 2010. 8, 9, 78, 118

[SS10b] Nitin Saxena and C. Seshadhri. From Sylvester-Gallai Configurations to Rank Bounds: Improved Black-box Identity Test for Depth-3 Circuits. *Technical Report TR10-013, (ECCC)*, 2010. 8, 78

[SSS09] Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. The Power of Depth 2 Circuits over Algebras. In *FSTTCS*, pages 371–382, 2009. 8

[Sud] Madhu Sudan. Lecture notes for the course 'Algebra and Computation'. Available from `http://people.csail.mit.edu/madhu/FT98/`. 19

[Sud97] Madhu Sudan. Decoding of Reed Solomon Codes beyond the Error-Correction Bound. *J. Complexity*, 13(1):180–193, 1997. 3

[Tia90] Cai Tianxin. Primes Representable by Polynomials and the Lower Bound of the Least Primes in Arithmetic Progressions. *Acta Mathematica Sinica, New Series*, 6:289–296, 1990. 75

[Tit30] E. C. Titchmarsh. A Divisor Problem. *Rend. Circ. Mat. Palerme*, 54:414–429, 1930. 75

[Too63] A L. Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. *English Translation in Soviet Mathematics*, 3:714–716, 1963. 58

[Val79] Leslie G. Valiant. Completeness Classes in Algebra. In *STOC*, pages 249–261, 1979. 78

[Vin72] I.M. Vinogradov. *Basic Number Theory*. Moscow, 1972. 40

[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. 81

[vzG83] Joachim von zur Gathen. Factoring Sparse Multivariate Polynomials. In *FOCS*, pages 172–179, 1983. 9, 100, 112

[vzG87] Joachim von zur Gathen. Factoring polynomials and primitive elements for special primes. *Theoretical Computer Science*, 52:77–89, 1987. 31

[vzGP01] Joachim von zur Gathen and Daniel Panario. Factoring Polynomials Over Finite Fields: A Survey. *J. Symb. Comput.*, 31(1/2):3–17, 2001. 30

[vzGS92] Joachim von zur Gathen and Victor Shoup. Computing Frobenius Maps and Factoring Polynomials. *Computational Complexity*, 2:187–224, 1992. 3, 30

[Xyl09] Triantafyllos Xylouris. On Linnik's constant, 2009. 62

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. *EUROSAM*, pages 216–226, 1979. 4, 78, 79