Exp(n+d)-time Algorithms for Computing Division, GCD and Identity Testing of Polynomials

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

Master of Technology

by

Kartik Kale

Roll No. : 15111019

under the guidance of

Prof. Nitin Saxena

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

June, 2017

Statement of Thesis Preparation

1. Thesis title: Exp(n+d)-time Algorithms for Computing Division, GCD and Identity Testing of Polynomials.
2. Degree for which the thesis is submitted: M. Tech.
3. Thesis Guide was referred to for preparing the thesis. ✓
4. Specifications regarding thesis format have been closely followed. ✓
5. The contents of the thesis have been organized based on the guidelines. ✓
6. The thesis has been prepared without resorting to plagiarism. ✓
7. All sources used have been cited appropriately. ✓
8. The thesis has not been submitted elsewhere for a degree. ✓

(Signature of the student)

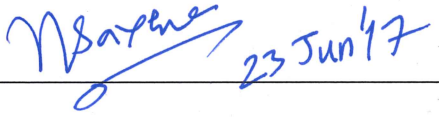
Name: KARTIK KALE

Roll No.: 15111019

Department/~~IDP~~: CSE

CERTIFICATE

It is certified that the work contained in the thesis titled "Exp(n+d)-time Algorithms for Computing Division, GCD and Identity Testing of Polynomials", by "Kartik Kale", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Prof. Nitin Saxena

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

June, 2017

Abstract

Agrawal and Vinay showed that a poly(s) hitting set for $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuits of size s, where a is $\omega(1)$ and b is $O(\log s)$, gives us a quasipolynomial hitting set for general VP circuits [AV08]. Recently, improving the work of Agrawal and Vinay, it was showed that a poly(s) hitting set for $\Sigma \wedge^a \Sigma\Pi^b(n)$ circuits of size s, where a is $\omega(1)$ and n, b are $O(\log s)$, gives us a quasipolynomial hitting set for general VP circuits [AFGS17]. The inputs to the $\wedge$ gates are polynomials whose arity and total degree is $O(\log s)$. (These polynomials are sometimes called 'tiny' polynomials).

In this thesis, we will give a new $2^{O(n+d)}$-time algorithm to divide an n-variate polynomial of total degree d by its factor. Note that this is not an algorithm to compute division with remainder, but it finds the quotient under the promise that the divisor completely divides the dividend. The intuition behind allowing exponential time complexity here is that we will later apply these algorithms on 'tiny' polynomials, and $2^{O(n+d)}$ is just poly(s) in case of tiny polynomials. We will also describe an $2^{O(n+d)}$-time algorithm to find the GCD of two n-variate polynomials of total degree d.

Using these algorithms, we will derandomize the whitebox PIT problem in the restricted case of $\Sigma^2\Pi\Sigma\Pi^b(n)$ circuits, where n, b are $O(\log s)$. In other words, the problem is to check whether

$$\prod_{i=0}^{s} f_i = \prod_{j=0}^{s} g_j$$

are equal, where f_i and g_j are sparse polynomials of degree $O(\log s)$ over $O(\log s)$ variables. Note that while we restrict the top fan-in to just 2, the fan-in of the upper Π gate does not have any restriction. The rough idea here is to divide f_i and g_j by their GCD for all pairs (f_i, g_j), until all the GCDs become 1.

Further, we will also derandomize the question of checking in blackbox whether the polynomial defined as

$$C := f - \prod_{i=0}^{s} g_i$$

is zero or not, where f and g_i's have arity and total degree $O(\log s)$ and their sparsity is poly(s).

Acknowledgments

I am tremendously grateful to my thesis adviser Prof. Nitin Saxena for mentoring me over the past two years. His contribution and experience in this field is immense, and he used it well to guide me. I found his optimism and discipline to be particularly inspiring.

Most of the work in this thesis is a joint work with Pranav Bisht. I would like to thank him not only for his significant contributions in this thesis, but also for being a good friend and lab-mate.

I would like to thank my family for constantly motivating me and guiding me throughout my life. No words are enough to express my gratefulness towards them for the things they have done for me – which are too numerous to list here.

I would like to thank IIT Kanpur for the excellent facilities provided to me over the past two years. I would like to thank the Department of Computer Science of IIT Kanpur for giving me this opportunity to work with some of the best minds in the country. I am grateful to the faculty members of the Department working in the field of theoretical computer science. I would like to thank Prof. Shashank Mehta, Prof. Raghunath Tewari and Prof. Satyadev Nandakumar for the courses that they taught me, and Prof. Mehta for the helpful advice that he lent to me during my days here. I would also like to thank the SIGTACS for the lectures that they arranged over the course of last two years. I found them to be very helpful for developing my knowledge in the field of theoretical computer science.

I would like to thank Sumanta for allowing me to pester him with the silliest of my questions, and clearing many of my misconceptions regarding the problem of polynomial identity testing. I would also like to thank Amit Sinhababu and Ashish Dwivedi for the illuminating discussions about various topics in theoretical computer science. I would also like to thank Ashish for having many insightful philosophical discussions with me.

I would like to thank all my friends and classmates, without whom, these two years would have been rather tedious. I would also like to apologize and thank everyone that I left out of this list because of any oversight.

Kartik Kale

Dedicated to

My family, for their unconditional love, support and guidance.

Contents

Chapter 1

Notations and Preliminaries

We will denote the set of numbers $\{1, 2, \cdots, n\}$ as $[n]$. We will denote the set of natural numbers as $\mathbb{N}$, and we will define 0 to be a natural number.

We will assume the usual definitions of algebraic objects like groups, rings, fields, vector spaces etc. A ring, unless mentioned otherwise, should be assumed to be a commutative ring with unit element (as we will hardly use non-commutative rings in this thesis).

Vectors will be denoted with a bar on top of the letter. Given a vector $\bar{v} \in \mathbb{R}^n$, defined as $\bar{v} = (v_1, v_2, \cdots, v_n)$, $\|\bar{v}\|$ will denote the ℓ_1-norm of $\bar{v}$, defined as $\|\bar{v}\| := \sum_{i\in[n]} |v_i|$. (Here $|v_i|$ is the absolute value of v_i.)

A polynomial ring is a ring formed by polynomials over given set of variables, whose coefficients come from another ring. The addition and multiplication operations in the polynomial ring are usual polynomial addition and multiplication operations. We will denote a polynomial ring as $R[x_1, x_2, \cdots x_n]$, where $x_1, x_2, \cdots x_n$ are the variables, and the coefficients come from the ring R. We will assume the usual definitions of a monomial and a polynomial.

We will often write a monomial $x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$ as $\bar{x}^{\bar{e}}$, when the number of variables is clear from the context. Here $\bar{e}$ denotes the vector $(e_1, e_2, \cdots e_n)$. Given a monomial $x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$, e_i is called the individual degree of the monomial with respect to the variable x_i. $\sum_{i\in[n]} e_i$, which is the sum of all individual degrees, is called the total degree (or degree) of the monomial. We will denote the total degree of monomial m as $\deg(m)$ and the individual degree with respect to the variable x_1 as $\deg_{x_1}(m)$.

One can see that the set of all monomials in variables $x_1, x_2, \cdots x_n$ is a countable set. Thus, there exists a bijection from $\mathbb{N}$ to the set of monomials in n variables. The bijections that also respect the monomial multiplication are called monomial orderings. More formally, a well-ordering $>$ on the set of monomials in $R[x_1, x_2, \cdots x_n]$ is called a monomial ordering if for all monomials m, m_1, m_2 in $R[x_1, x_2, \cdots x_n]$; $m_1 > m_2$, implies that $m_1 \times m > m_2 \times m$. Monomial orderings are useful tools for algorithms that deal with polynomials.

Definition 1.1 (Lex Ordering)**.** *We say that $x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n} > x_1^{e'_1} x_2^{e'_2} \cdots x_n^{e'_n}$ under the lexicographical monomial ordering (often called the lex ordering) if for some i in $[n]$, $e_1 = e'_1$, $e_2 = e'_2$, $\cdots$, $e_{i-1} = e'_{i-1}$ and $e_i > e'_i$.*

Definition 1.2 (Grlex Ordering)**.** *Given two monomials m_1 and m_2, we say that $m_1 > m_2$ under the graded lexicographical monomial ordering (often called the grlex ordering) if deg$(m_1) >$ deg(m_2); or if deg$(m_1) =$ deg(m_2) and $m_1 > m_2$ under the lex ordering.*

The total (or individual) degree of a polynomial is the maximum among the total (or individual) degrees of all monomials in it. The sparsity of a polynomial is defined as the number of monomials that have non-zero coefficient in the polynomial. The number of monomials over n variables of total degree at most d is $\binom{n+d}{d}$. So, the sparsity of an n-variate degree d polynomial is at most $\binom{n+d}{d}$. We will denote the coefficient of a monomial $\bar{x}^{\bar{e}}$ in a polynomial p as coeff$_{\bar{x}^{\bar{e}}}(p)$. A homogeneous polynomial of degree d is a polynomial that consists solely of monomials that have degree d.

Definition 1.3. *Given an n-variate degree-d polynomial $p = \sum c_{\bar{e}} \bar{x}^{\bar{e}}$; the degree $\leq t$ part of p is defined as $\sum_{\|\bar{e}\| \leq t} c_{\bar{e}} \bar{x}^{\bar{e}}$, and denoted as deg$_{\leq t}(p)$.*

In other words, degree $\leq t$ part of a polynomial is the sum of terms in the polynomial which have total degree at most t.

Definition 1.4. *Given an n-variate degree-d polynomial $p = \sum c_{\bar{e}} \bar{x}^{\bar{e}}$; the individual degree $\leq t$ part of p is defined as $\sum_{\forall i, e_i \leq t} c_{\bar{e}} \bar{x}^{\bar{e}}$, and denoted as ideg$_{\leq t}(p)$.*

In other words, degree $\leq t$ part of a polynomial is the sum of terms in the polynomial which have individual degree at most t with respect to each of the variables.

GCD, or the greatest common divisor, of two elements a and b of a ring R is defined as the element c of R, such that c divides both a and b, and every element of R that

divides both a and b also divides c. We will denote the GCD of $a, b \in R$ as (a, b), or sometimes as $\gcd(a, b)$.

A unique factorization domain is defined as an integral domain where each non-zero element of the domain can be written as a product of irreducible elements of the integral domain in unique manner, upto reordering or multiplication by units. Every field is also a unique factorization domain. GCD of two elements of a unique factorization domain is well defined. The domain of multivariate polynomials over a field (or over any unique factorization domain) is a unique factorization domain. Thus every pair of multivariate polynomials have a unique GCD (upto unit multiples).

Chapter 2

Introduction

This thesis primarily concerns the Polynomial Identity Testing (PIT) problem, which is an important problem in theoretical computer science. Given some representation of a multivariate polynomial, the problem asks us to determine whether the polynomial is identically zero or not. The way of representing the polynomial controls the difficulty of the problem. If the polynomial is given as a list of monomial-coefficient pairs, the problem can be trivially solved deterministically in linear time.This method of representing a polynomial is called the sparse representation of the polynomial. More interesting is the case when the polynomial is given as an arithmetic circuit, or as blackbox which takes the values of variables and outputs the evaluation of the polynomial at that point. We will discuss these representations of polynomials before proceeding further.

2.1 Arithmetic Circuits

Definition 2.1 (Arithmetic Circuits)**.** *An arithmetic circuit over a field $\mathbb{F}$, computing a polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ is a directed acyclic graph as follows:*
The vertices of the graph are called gates. Each gate with in-degree 0 is labelled with one of the variables $x_1, x_2, \cdots, x_n$ or with the constant 1. Every other gate is labelled with either $\times$ or $+$. The edges of the graph are called wires. The wires are labelled with constants of the field.

Each gate of a circuit is thought to compute a polynomial- the gates with in-degree 0 are thought to compute the polynomial that is their label. The gates with the label

$+$ (or $\times$) are thought to compute the polynomial that is the sum (or product) of the polynomials, multiplied by the constant given by the label of the connecting wire, computed by the gates that send a wire to this gate. For example, the circuit in Figure 2.1 shows an arithmetic circuit, which computes the polynomial $(x+5)(x+y)+3(x+y)+(-1)(x+y)(y+5)$, which reduces to $x^2-y^2+3x+3y$.

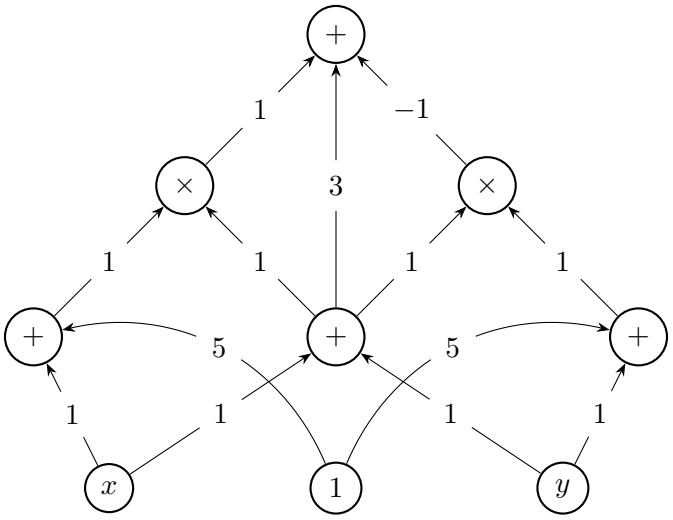


FIGURE 2.1: A circuit computing the polynomial $x^2-y^2+3x+3y$

The gates with out-degree zero are called the output gates, and the gates with in-degree zero are called the input gates or leaves. The in-degree of a gate is called its fan-in and the out-degree is called fan-out. The **size** of a circuit is defined to be the number of wires in the circuit. The maximum of the total degrees of the polynomials computed by the gates in the circuit is defined to be the **degree** of the circuit. Note that the degree of the polynomials computed at the output gates can all be less than the degree of the circuit, due to the possibility of cancellation of higher degree terms.

Without loss of generality, one can assume the circuit to be a layered directed acyclic graph. In this case, a $+$ gate directly connected to another $+$ gate can be combined together, to form a $+$ gate of larger fan-in. Thus, we can safely assume an arithmetic circuit to be a layered directed acyclic graph with alternating layers consisting of $+$ gates and $\times$ gates exclusively. The number of layers in such a circuit is equal to the depth of the circuit.

Sometimes some variations of this model are studied instead. If a gate with fan-in one in an arithmetic circuit is allowed to have the label $\wedge$ and which computes the exponentiation of the input, such a circuit is called a diagonal circuit. The $\wedge$ gate is just a special case of $\times$ gate, where all the input wires are connected to the same

gate. The circuits which are directed trees are called formulas. Arithmetic circuits are inspired from Boolean circuits, which have $\wedge$ and $\vee$ gates to compute conjunction and disjunction respectively.

Arithmetic circuits provide a more concise way to represent a polynomial. A polynomial which can be computed by an arithmetic circuit of size s can have degree $2^{O(s)}$ (for example $x_1^{2^d}$ has a circuit of size $2d$, by repeated squaring) or sparsity $2^{O(s)}$ (for example $\prod_{i\in[n]}(x_i+1)$ has sparsity 2^n, but circuit size $3n$).

Agrawal and Vinay in 2008 proved that for every arithmetic circuit of arbitrary depth, there exists an equivalent circuit of depth 4 with a slightly larger size [AV08]. So arithmetic circuits with constant depth are studied quite intensively, and a convention to describe a constant depth circuit in shorthand has developed.

Definition 2.2. *A $\Sigma^a\Pi^b\Sigma^c\Pi^d(n)$ circuit is an arithmetic circuit that computes an n-variate polynomial, which has 4 alternating layers of sum and product gates, with the output gate being a sum gate, such that the topmost layer has sum gates with fan-in at most a, the next layer has product gates with fan-in at most b, the next layer has sum gates with fan-in at most c, and the bottom layer has product gates with fan-in at most d.*

When there is no non-trivial restriction on fan-ins of gates in a particular level, we will often drop the superscript. When the number of variables can also be implicit. So, for example, a $\Sigma\Pi\Sigma\Pi$ circuit is a general depth-4 circuit, and a $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuit is a general depth-4 circuit computing an n-variate polynomial, such that the bottom layer of product gates has gates with fan-in b, and the topmost layer of product gates has gates with fan-in a.

2.2 Algebraic Complexity Theory

Arithmetic circuits can be thought of as another model of computation, like Turing machines. Turing machines solve decision problems, and the number of steps that a Turing machine takes is considered the fundamental computing resource. In the case of arithmetic circuits, the problems solved are computing a polynomial, and the fundamental resource is the size and depth of the circuit that computes the polynomial.

Many important decision problems can be reduced to the computation of some polynomial, and thus arithmetic circuits give rise to a new branch of complexity theory, called algebraic complexity theory.

The central questions of complexity theory revolve around separating various classes of problems. In the world of decision problems, this translates to proving lower bounds to the number of steps that a Turing machine would need to solve a given problem, or proving lower bounds on the size of Boolean circuit that computes the given problem. Similarly, in the world of arithmetic circuits, this translates to proving lower bounds on the size (and depth) of an arithmetic circuit computing a certain polynomial. More progress has been made on lower bounds for arithmetic circuits than their Boolean counterparts, and thus it is hoped that using arithmetic circuits to separate complexity classes will prove to be more productive.

2.3 Polynomial Identity Testing

2.3.1 The Problem Definition

As discussed earlier, PIT is the problem of checking whether a given multivariate polynomial is identically zero or not. Identically zero here means that the coefficient of each monomial in this polynomial is zero. Note that the polynomial evaluating to zero on all possible points does not always imply that the polynomial is identically zero (for example the polynomial $x^2 + x \in \mathbb{F}_2[x]$ evaluates to zero at all points in $\mathbb{F}_2$). However, this problem only appears if the polynomial is over a finite field, and if we consider the polynomial as a polynomial over a large enough field extension of the base field, this problem goes away. For the rest of the discussion in this thesis, we will not worry about this problem.

There are two important versions of the problem that have seen the most of the development on PIT. In the first version, the arithmetic circuit for the polynomial is given to us, and we need to check whether the circuit computes a zero polynomial. This version is called the whitebox PIT problem. Solving the whitebox PIT problem in deterministic polynomial time means finding a deterministic algorithm, that takes a size-s, degree-d circuit computing an n-variate polynomial and correctly outputs

whether the circuit computes a zero polynomial or not, in time less than $s^\alpha n^\beta d^\gamma$ for some constants α, β, γ.

The second version of PIT is called the blackbox PIT problem, which is possibly harder than the whitebox version. Here we are given a blackbox, which computes the evaluations of an n-variate arithmetic circuit of size s, degree d at any given point in one step. Using this blackbox, we need to determine whether the underlying arithmetic circuit computes the zero polynomial, in time poly(s, n, d). Note that we allow the running time to be polynomial in the parameters of the circuit, but the input to our algorithm will be a blackbox which evaluates the circuit, alongwith some information about the circuit like the values of s, n, d and not the actual circuit.

In a blackbox PIT algorithm, all that one is allowed to do is evaluating the blackbox at a few (poly(s, n, d)) points. If one finds that the underlying polynomial takes a non-zero value at some point, we can safely say that the circuit is not identically zero. Thus, finding a set of points, such that any non-zero polynomial will take a non-zero value at at least one of the points in the set, is the heart of the problem. We formalize this by defining hitting sets.

Definition 2.3 (Hitting Set)**.** *We say that a set $\mathcal{H} \subseteq \mathbb{F}^n$ is a hitting set for a class of polynomials (or arithmetic circuits) $\mathcal{C} \subseteq \mathbb{F}[x_1, x_2, \cdots, x_n]$, if for every non-zero polynomial $f \in \mathcal{C}$, $\exists \mathbf{v} \in \mathcal{H}$, s.t. $f(\mathbf{v}) \neq 0$.*

Finding a hitting set of size poly(s, n, d) for a class of circuits $\mathcal{C}$ immediately gives us a deterministic polynomial time blackbox PIT algorithm for $\mathcal{C}$. One can also prove that finding a deterministic polynomial time algorithm for blackbox PIT is equivalent to finding a hitting set of polynomial size, in polynomial amount of time.

Similar to hitting sets, the generator is another concept that captures the difficulty of blackbox PIT.

Definition 2.4 (Generator)**.** *A polynomial map $\bar{\mathcal{G}} \in (\mathbb{F}[x_1, x_2, \cdots, x_s])^n$ is called a generator for a class of polynomials (or circuits) $\mathcal{C} \subseteq \mathbb{F}[x_1, x_2, \cdots, x_n]$, if for every non-zero polynomial $f \in \mathcal{C}$, $f(\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n) \in \mathbb{F}[x_1, x_2, \cdots, x_s]$ is a non-zero polynomial.*

A generator reduces the n-variate blackbox PIT problem to an s-variate blackbox PIT problem. Naturally, it makes sense to only consider generators where s is quite small compared to n (Often $\log n$ or constant). There is a strong connection between a hitting set and a generator, with either of them implying the other.

2.3.2 Motivation

The PIT is an interesting natural problem in computer science. It is one of the few problems that are not known to be NP-complete, and that are not known to be in P. Further, it has a randomized algorithm which puts it in coRP, and thus if the widely believed conjecture P=BPP is true, PIT should also have a deterministic polynomial time algorithm. It is arguably the most popular problem in coRP that is not known to be in P.

Apart from this, a major motivating factor behind the study of PIT is its intricate connection with complexity theory – Derandomizing PIT has more implications than merely bringing another problem from coRP to P. There has been a long line of work exploring the connections between PIT and arithmetic circuit lower bounds.

In 2004, Kabanets and Impagliazzo showed that a non-deterministic sub-exponential time algorithm for PIT will mean that either the complexity class NEXP is not contained in P/Poly, or that VNP$\neq$VP [KI03]. Note that the PIT problem is in coRP, which means that it is in coNP and it has an exponential time deterministic algorithm. VP and VNP are considered to be the arithmetic analogues of P and NP, and the result gives an explicit example of a problem that will be in VNP but not in VP, if VNP is indeed not contained in VP. P/Poly can be thought of as a class of problems that have polynomial sized Boolean circuits. Since a circuit has fixed number of inputs, a different circuit for each input size is required. But P/Poly does not enforce any relationship between the circuits for each input size, and this allows very difficult problems to be in P/Poly. If we put the additional restriction that the circuits must be computable by an algorithm that takes the input size as an input, this class reduces to P. The lack of restriction on relationship between circuits for different input sizes is called non-uniformity. It is another powerful computational resource, like non-determinism or randomness.

In 2005, Agrawal showed that if a generator $\bar{\mathcal{G}}$ exists for a class of n-variate, degree-d and size-s circuits $\mathcal{C}$ and that each $\mathcal{G}_i$ has degree polynomial in n and is computable in time polynomial in n; then there exists a polynomial that is not in $\mathcal{C}$ which is computable in polynomial time [Agr05]. His proof looks at the annihilating polynomial of $\mathcal{G}_1, \mathcal{G}_2, \cdots$, which cannot be in $\mathcal{C}$. Since this result requires generators, we need black-box PIT algorithm to apply this result. Note that the previous result by Kabanets and

Impagliazzo can work with even whitebox PIT algorithm (or even a non-deterministic sub-exponential time algorithm).

Recently, the results of Kabanets and Impagliazzo [KI03] were greatly strengthened, in proving that a PIT algorithm that solves the blackbox PIT problem for the special case of $\Sigma\wedge^a\Sigma\Pi^b(n)$ circuits in time poly$(2^{n+b}, \mu(a))$ for any μ, implies that either E$\not\subseteq$ #P/Poly or some polynomials in VNP need $2^{\Omega(n)}$ circuits[AFGS17]. They also show that such a PIT algorithm will also give us a quasipolynomial time algorithm for the blackbox PIT of general VP circuits.

2.4 Derandomizing the PIT Problem

As we mentioned earlier, there exists a randomized algorithm to solve the blackbox PIT proble. First we will discuss that alogrithm here in in this section.

There has been a lot of work done to derandomize the PIT problem, especially in the last decade. The problem of PIT for general circuits has been reduced to many restricted classes of circuits, and the problem for many other restricted classes of circuits has been solved. We will discuss the most relevant of the results for this thesis in this section.

As we know that a univariate polynomial of degree d over a field $\mathbb{F}$ has at most d roots, the univariate blackbox PIT can be solved by checking whether the polynomial evaluates to zero at more than d distinct points in $\mathbb{F}$. If the field size is less than d, one can work with a sufficiently large field extension of $\mathbb{F}$. This idea is generalized to multivariate polynomials in the following section.

2.4.1 The Randomized Algorithm

The bound on number of roots of a univariate polynomial (as discussed above) does not directly generalize to multivariate polynomial. However, DeMillo and Lipton in 1978 gave a probabilistic result generalizing this property [DL78], and Schwartz and Zippel gave a slightly improved version subsequently [Zip79, Sch80].

Lemma 2.5 (Schwartz-Zippel Lemma [DL78, Zip79, Sch80]). *Let f be a non-zero polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d. Let $S \subseteq \mathbb{F}$. Then for a point $\bar{c}$ drawn from S^n uniformly at random,* $\Pr[f(c_1, c_2, \cdots, c_n) = 0] \leq \frac{d}{|S|}$.

We will not prove this lemma here. Lemma 2.5 immediately gives us a randomized algorithm for PIT – we pick a set $S \subseteq \mathbb{F}$ of size at least $2d$, pick a point $\bar{c}$ from S^n uniformly at random, test whether f is zero at $\bar{c}$ and declare f to be zero if $f(\bar{c})$ is zero and declare f to be non-zero otherwise.

The algorithm always gives the correct answer if the polynomial is zero, and gives the correct answer with probability at least $1/2$ if the polynomial is non-zero. This puts the PIT problem in coRP. Note that this algorithm solves even the blackbox PIT problem. Similar to univariate PIT algorithm, problems arise if the degree of the polynomial is larger than the field size, and these problems can be circumvented by looking at a sufficiently large field extension of $\mathbb{F}$.

From Lemma 2.5, we can also get a deterministic blackbox PIT for n-variate and polynomials with degree at most d, that runs in time $(d+1)^n$. So, if the number of variables n is a constant, this algorithm solves the blackbox PIT in polynomial time. The following theorem states this explicitly:

Theorem 2.6. *The blackbox PIT problem for polynomials in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ with degree at most d has a deterministic algorithm that runs in time $(d+1)^n$.*

Proof. Suppose $S \subseteq \mathbb{F}$ of size at least $d+1$. Then, by Lemma 2.5, for a point $\bar{c} \in S^n$, $\Pr[f(c_1, c_2, \cdots, c_n) = 0] \leq \frac{d}{d+1} < 1$. So, there must exist a point $\bar{\alpha} \in S^n$ such that $f(\bar{\alpha}) \neq 0$, for any polynomial f of degree at most d. So by evaluating f on all points of S^n, and checking if all of them evaluate to zero, one can check whether $f = 0$. □

It has also been proven that for the set of n-variate polynomials that are computable by an arithmetic circuit of size s and degree d, there always exists a hitting set of size at most poly(s, n, d). The proof is not constructive though, and thus we cannot use this result for an algorithm for PIT of such polynomials. Note that for the complexity theoretic purposes, all we need is the proof that a polynomial time deterministic PIT algorithm exists, for example in the case of result by Agrawal [Agr05]. Unfortunately, the existence of a hitting set of small size does not directly translate into the existence of a PIT algorithm, as for a PIT algorithm to exists, we need to know that a hitting

set that is also computable in polynomial time. This proof of existence of hitting sets goes via a counting argument, by counting the number of small non-zero circuits of small degree, and then estimating the probability that a random set $S \subseteq \mathbb{F}$ of size x will be such that all points $p \in S^n$ will be roots for some polynomial computed by such circuits. This method can also be extended for infinite fields. We will not prove this here, but an interested reader can refer to the survey on arithmetic circuit complexity by Shpilka and Yehudayoff [SY10].

2.4.2 Structural Results for Arithmetic circuits

Much work has been done on proving various structural results for arithmetic circuits, i.e., given an arithmetic circuit computing a polynomial f, proving the existence of (or constructing) an equivalent arithmetic circuit with some additional property which computes the same polynomial f. Usually applying a structural result increases the size of the resulting circuit. We will just state the structural results here relevant for this thesis without proof, but an interested reader can refer to the survey [SY10] by Shpilka and Yehudayoff for a comprehensive analysis of these results.

If the proof for a structural result is constructive, and the increase in the size of the resulting circuit is reasonable, a whitebox PIT algorithm can first convert the input circuit into a circuit with the additional property as a preprocessing step, allowing us to focus only on finding a whitebox PIT algorithm for circuits promised to have this additional property. Further, even if the proof is not constructive, a blackbox PIT algorithm can just assume that the circuit hidden under the blackbox has this additional property, allowing us to focus on finding a blackbox PIT algorithm for polynomials computable by circuits with this additional property.

We will discuss the technique of homogenization first, which is derived from the work in [Str73]. We now state in the following theorem the version of this technique as given in the survey of Shpilka and Yehudayoff [SY10]:

Theorem 2.7 (Homogenization [Str73])**.** *Given an arithmetic circuit C of size s, computing a polynomial f, one can construct an arithmetic circuit C' of size $d^2 s$, in time poly(d, s), such that C' has d outputs, with i^{th} output giving the degree-i homogeneous part of f, and such that each gate of C' computes a homogeneous polynomial.*

The proof of this theorem goes by induction on the gates, by showing that the homogeneous parts of a gate can be computed using the homogeneous parts of its inputs. Using this result, one can reduce the PIT problem for general polynomials to the PIT problem for homogeneous polynomials.

Many structural results for arithmetic circuits concern themselves with constructing a circuit with smaller depth. Such results are often called circuit depth reduction results. We will take a look at two of the depth reduction results here.

Theorem 2.8 ($\log d$-Depth Reduction [VSBR83])**.** *For every arithmetic circuit C of size s computing a homogeneous n-variate polynomial f of degree d, there exists another arithmetic circuit C' of size bounded by poly$(s, n, \log d)$, computing the polynomial f, such that C' has alternating layers of sum and product gates, and the number of layers is at most $O(\log d)$.*

The proof of this theorem also gives us a randomized polynomial time (in s, n, d) algorithm to construct C' from C. So this result, in conjunction with the homogenization result (Theorem 2.7) helps us in reducing the blackbox PIT problem for general circuits to the blackbox PIT problem for circuits with depth $\log d$, where d is the degree of the underlying polynomial.

Agrawal and Vinay in 2008 gave a depth reduction result that reduces the depth of a circuit to 4 [AV08]. Although this makes the size of the resulting circuit exponential, it still gives us a size bound for constant depth circuits that is better than the trivial size bound. A trivial size bound can be obtained by considering that a ΣPi circuit that computes an n-variate degree-d polynomial can have size as large as $\binom{n+d}{d}$. It is also not very hard to see that a $\Sigma\Pi\Sigma\Pi$ circuit of size $\binom{s+d}{d}$, equivalent to a circuit of size s computing an n-variate degree-d polynomial can also be constructed. The following theorem, however, gives a much better size bound:

Theorem 2.9 (Reduction to Depth 4 [AV08, Koi12, Tav15])**.** *Suppose an arithmetic circuit C of size s computes an n-variate polynomial f of degree d. Then, for every $t \in [d]$, there exists an arithmetic circuit C' with depth 4 and size $s^{O(t+d/t)}$ computing the polynomial f. Further, each gate of C' computes a homogeneous polynomial, and C' has the form $\Sigma^{s^{O(d/t)}}\Pi^{O(d/t)}\Sigma\Pi^{t}$.*

Using this theorem, they also proved that a poly(s) hitting set for a $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuit, where a is $\omega(1)$ and b is $O(\log s)$, gives us a quasipolynomial time hitting set for

general arithmetic circuits. Since this result, most of the work towards derandomizing PIT has concerned itself with derandomizing PIT for the special case of constant depth circuits. In 2013, Gupta, Kamath, Kayal and Saptharishi gave a depth reduction result that reduces the depth of a circuit to 3, for polynomials over fields of characteristic zero [GKKS13]. Improving the result of Agrawal and Vinay, recently Sumanta and others proved that a poly(s) hitting set for a $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuit, where a is $\omega(1)$ and n, b are $O(\log s)$, gives us a quasipolynomial time hitting set for general arithmetic circuits [AFGS17].

2.4.3 The Sparse PIT Algorithm

Now we will discuss arguably the most important of the deterministic PIT algorithms in this section. This algorithm was also one of the first deterministic algorithms for PIT. This algorithm solves the blackbox PIT problem for a n variate polynomial with degree d and sparsity s, in time poly(s, n, d). Note that for a general n-variate degree-d polynomial has sparsity $\binom{n+d}{d}$ in the worst case, so this algorithm does not solve the blackbox PIT problem in polynomial time. However, it does solve the problem in time poly(z, n, d) when the input circuit is promised to be a $\Sigma\Pi$ circuit of size z. Various versions of this algorithms have been published at various times, for example [KS01, BHLV09].

This algorithm employs a set $S = \{\phi_1, \phi_2, \cdots \phi_k\}$ of homomorphisms where for all $i \in [k]$, $\phi_i : \mathbb{F}[x_1, x_2, \cdots x_n] \to \mathbb{F}[y]$, is a homomorphism that maps all n-variate polynomials to univariate polynomials. The size of S is only poly(s, n, d). Further, the set S is constructed in such a way that for each set T of size s consisting of n-variate degree-d monomials, there exists some $\phi_i \in S$, such that at least one monomial of T gets mapped to a different (non-zero) univariate than all the univariate images of other monomials of T. This effectively means that, there exists a $\phi_i \in S$ for each non-zero polynomial $p \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of sparsity at most s, such that $\phi_i(p) \neq 0$. So our algorithm will declare p to be zero if $\phi_i(p) = 0$ for all i, and non-zero otherwise.

The Kronecker map provides the inspiration for this set of maps. The Kronecker map $\psi_d : \mathbb{F}[x_1, x_2, \cdots, x_n] \to \mathbb{F}[y]$ is defined for a given d as $\psi_d(f(x_1, x_2, \cdots, x_n)) = f(y^{d^0}, y^{d^1}, \cdots, y^{d^{n-1}})$. So, under this map, a monomial $x_1^{e_1}x_2^{e_2}\cdots x_n^{e_n}$ will be mapped to $y^{e_1d^0+e_2d^1+\cdots+e_nd^{n-1}}$. Thus, the images under Kronecker map of all monomials of individual degree less than d will be different, and thus for a non-zero polynomial p,

the image $\psi_d(p)$ will also be non-zero. Unfortunately, the degree of the image of a monomial of degree at most d can be as high as d^n, which is too large. (After applying this map, we intend to check whether the univariate polynomial obtained is zero or not, by evaluating it at d' distinct points where d' is the degree of the univariate.)

So as a workaround, the Kronecker map was slightly modified, by taking the exponent of y modulo a prime p. Now the problem with this approach is that the exponent of two distinct monomials in the polynomial can sometimes be equal modulo p, for some primes p. However, any number n has at most $\log_2 n$ factors, so we try to limit the number of such 'bad' primes. The number of such 'bad' primes comes out to be small, so if we try out many different primes, one of them must work for a given polynomial. We formalize this idea in the following theorem.

Lemma 2.10 ([KS01]). *Suppose $m_1, m_2 \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are two distinct monomials such that individual degree of both m_1 and m_2 is less than d in all the variables. For each $i \in \mathbb{Z}$, the homomorphism $\phi_i : \mathbb{F}[x_1, x_2, \cdots, x_n] \to \mathbb{F}[y]$ is defined as $\phi_i(f(x_1, x_2, \cdots, x_n)) = f(y^{w_1}, y^{w_2}, \cdots, y^{w_n})$, where p_i is the i^{th} prime number, and where $w_j \equiv d^{j-1} \pmod{p_i}$, $0 \leq w_j < p_i$ for all $j \in [n]$. Then $\phi_i(m_1)$ can be equal to $\phi_i(m_2)$ for no more than $n \log d$ values of i.*

Proof. Suppose $m_1 = x_1^{\gamma_1} x_2^{\gamma_2} \cdots x_n^{\gamma_n}$ and $m_2 = x_1^{\delta_1} x_2^{\delta_2} \cdots x_n^{\delta_n}$. First observe that the image of a monomial under the map ϕ_i is a monomial in y, for any i. Suppose $\phi_i(m_1) = y^\alpha$, $\phi_i(m_2) = y^\beta$. Then, $\alpha = \gamma_1 w_1 + \gamma_2 w_2 + \cdots + \gamma_n w_n$. So, by the definition of w_j, $\alpha \equiv \gamma_1 d^0 + \gamma_2 d^1 + \cdots + \gamma_n d^{n-1} \pmod{p_i}$. Similarly, $\beta \equiv \delta_1 d^0 + \delta_2 d^1 + \cdots + \delta_n d^{n-1} \pmod{p_i}$. So if $\alpha = \beta$, p_i must divide $(\gamma_1 - \delta_1)d^0 + (\gamma_2 - \delta_2)d^1 + \cdots + (\gamma_n - \delta_n)d^{n-1}$. Without loss of generality, we can assume that $(\gamma_1 - \delta_1)d^0 + (\gamma_2 - \delta_2)d^1 + \cdots + (\gamma_n - \delta_n)d^{n-1} \geq 0$. Now, $\gamma_i < d$ and $\delta_i < d$ for all i, as m_1 and m_2 have individual degree at most d in all the variables. So, $(\gamma_1 - \delta_1)d^0 + (\gamma_2 - \delta_2)d^1 + \cdots + (\gamma_n - \delta_n)d^{n-1} < d^n$. Now, a positive integer less than d^n can have no more than $\log d^n = n \log d$ many distinct prime factors. So, $\phi_i(m_1)$ can be equal to $\phi_i(m_2)$ for no more than $n \log d$ values of i. □

The above theorem bounds the number of ϕ_i's that we need to try out to ensure that the images of two different monomials remain different under at least one of the ϕ_i's. So, for a polynomial p of sparsity s that contains a monomial m, the number of ϕ_i's that we need to try out, to ensure that the image of m is different from all other monomials appearing in p, is at most $s \times n \log d$. Further, if i is such that the image

of m under ϕ_i is different from the images of all other monomials appearing in p, then one can see that $\phi_i(p)$ cannot be zero for this i – the image of m under this map is a non-zero polynomial, and no monomial in $\phi_i(p-m)$ can cancel $\phi_i(m)$.This gives us the following theorem:

Theorem 2.11 ([SY10]). *Suppose p is a polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of sparsity at most s such that $\deg_{x_i}(p) < d$ for all $i \in [n]$. For each $i \in \mathbb{Z}$, we define the homomorphism ϕ_i as in Lemma 2.10. Then, if $p = 0$, then $\phi_i(p) = 0$ for all $i \in \mathbb{Z}$, and if $p \neq 0$ then $\phi_i(p) = 0$ for no more than $sn \log d$ values of i.*

Proof. Since each ϕ_i is a homomorphism, $\phi_i(0) = 0$ for all i. For the converse, using Lemma 2.10, and by the argument above, one can see that $\phi_i(p)$ cannot be zero for more than $sn \log d$ values of i. □

To use this theorem for a blackbox PIT algorithm, one can simply pick the first $sn \log d + 1$ ϕ_i's, and check if the image of the given blackbox under each of them is zero. The degree of the image of a polynomial p having individual degree less than d in each variable under the map ϕ_i is at most $d \times (w_1 + w_2 + \cdots + w_n)$, which is less than $d \times p_i$. So to check if $\phi_i(p) = 0$, we need to evaluate $\phi_i(p)$ at $d \times p_i$ many points of the field. For this to be considered efficient, we need a upper bound on the value of p_i. Now, the prime number theorem tells us that the n^{th} prime number is less than $n \log n + n \log \log n$. One can easily generate the list of first k prime numbers in time polynomial in k.

So, to summarize, the sparse PIT algorithm for a polynomial f first generates a list of m prime numbers, where $m = sn \log d + 1$; then constructs $\phi_1, \phi_2, \cdots, \phi_m$ and evaluates each $\phi_i(f)$ at $dm \log m + dm \log \log m$ many points; and if all of these evaluations are zero, then outputs $f = 0$, and outputs $f \neq 0$ otherwise.

This technique of solving PIT via homomorphism(s) that satisfy some special property has provided an inspiration for many other results that derandomize some special case of PIT. Our algorithms for derandomizing PIT for certain kinds of polynomials too, follow the same method, but we require the homomorphisms to satisfy different properties.

An important implication of Lemma 2.10 is the existence of a homomorphism that keeps the images of any two monomials of given degree separate. Such a homomorphism will prove to be useful for the algorithms that we give in later chapters. So we will formally state this in the following theorem.

Theorem 2.12. *For all d, there exists a homomorphism $\phi_d : \mathbb{F}[x_1, x_2, \cdots, x_n] \to \mathbb{F}[y]$ such that $\phi_d(m_1) \neq \phi_d(m_2)$, $\phi_d(m_1) \neq 0$ and $\phi_d(m_2) \neq 0$, for any two distinct non-zero monomials $m_1, m_2 \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ which have degree at most d. Further, ϕ_d can be constructed in time polynomial in $\binom{n+d}{d}$.*

Proof. Let s be the number of monomials that have individual degree less than d in each of the variables. Then, $s = \binom{n+d}{d}$. Now, the number of monomial pairs (m_1, m_2) such that m_1, m_2 have degree at most d is $\binom{s}{2} < s^2$. Now, Lemma 2.10 tells us that for any such pair of distinct non-zero monomials, there can be at most $n \log d$ values of $i \in \mathbb{Z}$ where $\phi_i(m_1 - m_2) = 0$, where ϕ_i's are defined as in Lemma 2.10. So, if we look at first $s^2 \times n \log d$ values of i, at least one of the corresponding ϕ_i's will be such that $\phi_i(m_1) \neq \phi_i(m_2)$ for any two distinct non-zero monomials m_1, m_2 with degree less than d. This will give us the desired homomorphism.

Constructing first $s^2 n \log d$ homomorphisms will take no more than poly(s, n, d) time. Checking, for each of the $\binom{s}{2}$ pairs of monomials, and for each of the homomorphisms under consideration, whether the image of the monomials in the pair is different under the homomorphism also takes no more than poly(s, n, d) time. Thus the homomorphism is constructible in time in $\binom{n+d}{d}$. □

The above theorem does not explicitly describe the map ϕ_d, but gives an algorithm that can be run to construct ϕ_d. However, we know that, by the construction in Lemma 2.10, that ϕ_d is is defined as $\phi_i(f(x_1, x_2, \cdots, x_n)) = f(y^{w_1}, y^{w_2}, \cdots, y^{w_n})$, where p is some prime number, and where $w_j \equiv d^{j-1} \pmod{p}$, $0 \leq w_j < p$ for all $j \in [n]$. One can also bound the value of p using prime number theorem. As the proof above requires $s^2 n \log d$ many primes, where s is $\binom{n+d}{d}$, the maximum value of p can be $k \log k + k \log \log k$, where $k = s^2 n \log d$. Asymptotically, p is of the order of $2^{O(n+d)}$.

Since all the monomials of degree at most d are mapped to distinct monomials under ϕ_d, by arguments similar to those in the proof of Theorem 2.11, we can say that $\phi_d(f) \neq 0$ for all non-zero polynomials f of degree at most d. So this gives us a single

homomorphism that preserves the non-zeroness of all polynomials of degree at most d, although the construction of this homomorphism takes time exponential in n and d.

2.5 Our Work

So far, we have seen the problem of polynomial identity testing, its applications, some algorithms that solve it for some special cases, and some methods that may help us in making progress towards solving it completely. In this section, we will briefly summarize the contributions of this thesis.

In this thesis, we give a deterministic algorithm that solves the whitebox PIT problem for arithmetic circuits of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$ and size s in time $\text{poly}(s) \cdot 2^{O(n+b)}$ (Algorithm 4). Further, we give a deterministic algorithm that solves the blackbox PIT problem for polynomials of the form $f - \prod_{i=1}^{k} g_i$ where f, g_i are polynomials in n variables with degree less than d, in time $\text{poly}(k) \cdot 2^{O(n+b)}$ (Algorithm 3). This second model is essentially a special case of the first model, where one of the two product gates in the second layer has fan-in one. The sparsity of a polynomial computed by a circuit of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$ and size s can be as large as $\binom{n+b}{b}^s$, which is approximately $2^{s(n+b)}$. The sparsity of a polynomial of the form $f - \prod_{i=1}^{k} g_i$ can be as large as $\binom{n+d}{d}^k$, which is approximately $2^{k(n+b)}$. So these PIT algorithms perform better than a naïve application sparse PIT algorithm.

More importantly, in light of the recent result [AFGS17] of Sumanta and others, we hope that the techniques used in these algorithms will help us in understanding the depth-4 model and in discovering an algorithm for blackbox PIT of general depth-4 circuits (without the top fan-in restriction) that runs in time $\text{poly}(s) \cdot 2^{O(n+b)}$. Note that our algorithm only requires time polynomial in the number of the upper Π gates, while the result by Sumanta and others allows for an algorithm that runs in time proportional to $t(a)$ where t is any bounded function and a is the said number of product gates.

Also, this thesis gives an algorithm to test the divisibility of a polynomial by another polynomial, and an algorithm to compute the GCD of two polynomials. These algorithms both run in time $2^{O(n+d)}$. These algorithms are used in our PIT algorithms.

The blackbox algorithm works by applying a homomorphism on the polynomial ring in n varaibles that maps an n variate polynomial to a bivariate polynomial. We discuss the properties that a useful homomorphism must satisfy in Chapter 3. It is shown that a homomorphism that preserves non-zeroness and indivisibility of a pair of polynomials will be useful for Algorithm 3.

The properties that Chapter 3 suggests are proven for a homomorphism that we will describe in Chapter 4, by means of an algorithm that tests divisibility of a polynomial by another polynomial. The homomorphism that we describe preserves each step of this algorithm, and thus preserves indivisibility of two polynomials. All this is explained in greater details in Chapter 4.

The divisibility testing algorithm can be slightly modified to give us a GCD algorithm (Algorithm 2). That algorithm is described in Chapter 5. This algorithm is used in the whitebox PIT algorithm that we will present in Chapter 6.

Chapter 6 finally states the two PIT algorithms, using all the things proven in earlier chapters.

Chapter 3

Homomorphisms And PIT

3.1 Homomorphisms for Blackbox PIT

Like numerous other PIT algorithms, we try to find a homomorphism from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to polynomials in fewer variables, which preserves some algebraic property of the polynomials, and such that this property distinguishes the zero polynomial from other polynomials in some way. In the case of polynomials of the form $f - \prod_{i=1}^{k} g_i$, the properties that we look at are the indivisibility and non-zeroness preservation – that is, we identify a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2]$, such that for any $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $f \nmid g$ then $\psi(f) \nmid \psi(g)$ in $\mathbb{F}[x_1, x_2]$ and if $f \neq 0$ then $\psi(f) \neq 0$. In this chapter, we will explain why such a homomorphism will be of any use for the blackbox PIT of polynomials of the form mentioned above.

Suppose we are given a blackbox computing a polynomial $p \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ which is promised to be equal to $f - g_1 g_2 \cdots g_k$ where $f, g_1, g_2, \cdots, g_k$ are polynomials in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d. We are to test whether $p = 0$ or not, in blackbox, and in time poly$(s) \cdot 2^{(n+d)}$ where s is the size of the arithmetic circuit hidden inside the blackbox, computing p. First, we claim that we can safely assume that $g_1, g_2, \cdots, g_k$ are irreducible polynomials. In other words, a blackbox PIT algorithm that runs in time poly$(k) \cdot 2^{(n+d)}$ and tests polynomials of the form $f - g_1 g_2 \cdots g_k$ where $g_1, g_2, \cdots, g_k$ are irreducibles; yields an algorithm that solves blackbox PIT for the same model but without the irreducibility condition.

Lemma 3.1. *Suppose there exists a blackbox PIT algorithm A for polynomials of the form $f - g_1 g_2 \cdots g_k$, where $g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are irreducible polynomials of degree at most d, and f is a polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d. Further, suppose that the algorithm A runs in time at most $\alpha k^\beta 2^{\gamma(n+d)}$ for constants α, β, γ. Then, there exists an algorithm A' that solves blackbox PIT for polynomials of the form $f - g_1 g_2 \cdots g_k$, where $f, g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are some polynomials of degree at most d. Further, the algorithm A' runs in time at most $\alpha(kd)^\beta 2^{\gamma(n+d)}$.*

Proof. Suppose we are given a polynomial p such that $p = f - g_1 g_2 \cdots g_n$ for some $f, g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d. Then, look at the irreducible factorization of each g_i. Suppose $g_i = g_{i1} g_{i2} \cdots g_{in_i}$ for all $i \in [k]$, such that g_{ij} are irreducible polynomials of degree at least 1 for all i, j. Now, since degree of g_i at at most d, and g_{ij} is non-costant for all $j \in [n_i]$, n_i can be at most d. So, p can be written as $f - \prod g_{ij}$, where g_{ij} are all irreducible polynomials, and there are at most kd of them. Thus the algorithm A can do blackbox PIT of p in time at most $\alpha(kd)^\beta 2^{\gamma(n+d)}$. □

Note here that the algorithm A' is essentially the same as the algorithm A, but with different parameters. The algorithm does not actually factorize the polynomials. This kind of technique is frequently used in blackbox algorithms, and can be sometimes a bit confusing at first glance. Further, note that $\alpha(kd)^\beta 2^{\gamma(n+d)}$ is indeed poly$(k) \cdot 2^{O(n+d)}$.

Now we prove that a non-zeroness and indivisibility preserving map, as mentioned above, will give us PIT for polynomials of the form $f - g_1 g_2 \cdots g_k$, where $g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are irreducible polynomials of degree at most d, and f is a polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d.

Lemma 3.2. *Suppose there exists a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2]$ such that for any $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, if $f \nmid g$ then $\psi(f) \nmid \psi(g)$, and such that for any $h \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $h \neq 0$ then $\psi(h) \neq 0$. Suppose a polynomial p is such that $p = f - g_1 g_2 \cdots g_k$, where $g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are irreducible polynomials of degree at most d, and f is some polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d. Then $\psi(p) = 0$ if and only if $p = 0$.*

Proof. We will prove this by induction on k. For the base case, suppose $p = f - g_1$. Then, p has degree at most d, and non-zeroness preserving property of ψ means

that if $p \neq 0$ then $\psi(p) \neq 0$. If $p = 0$ then $\psi(p)$ must be zero anyway, as ψ is a homomorphism. This completes the base case.

Now assume that the lemma is true for $k < l$. We need to prove that it will also be true for $k = l$. Suppose $p = f - g_1 g_2 \cdots g_l$. Again, if p is zero, then $\psi(p)$ is zero, because ψ is a homomorphism. So suppose $p \neq 0$. Since ψ is a homomorphism, $\psi(p) = \psi(f) - \psi(g_1)\psi(g_2) \cdots \psi(g_l)$. If $g_l \nmid f$, by the indivisibility preserving property of ψ, $\psi(g_l) \nmid \psi(f)$. So, $\psi(f)$ cannot be equal to $\psi(g_l) \times \psi(g_1)\psi(g_2) \cdots \psi(g_{l-1})$, and thus $\psi(p) \neq 0$.

On the other hand, if $g_l \mid f$, suppose $f' = f/g_l$. Then, $p = g_l(f' - g_1 g_2 \cdots g_{l-1})$. Now since p is non-zero, both of these factors must be non-zero. After applying ψ we get $\psi(p) = \psi(g_l) \times \psi(f' - g_1 g_2 \cdots g_{l-1})$. $\psi(g_l)$ cannot be zero by the non-zeroness preserving property of ψ. $\psi(f' - g_1 g_2 \cdots g_{l-1})$ is non-zero by induction hypothesis (as f' is also a polynomial with degree at most d). Thus $\psi(p) \neq 0$. □

Note that this proves that $\psi(p)$ is non-zero if and only if p is zero, and thus reduces the n-variate PIT problem (of the restricted model) to bivariate PIT problem. In the previous chapter, we have seen ways to solve PIT problem for constant variate polynomials. To combine the two, one must have explicit homomorphism ψ. Merely proving the existence of such a ψ will not suffice. In fact, we believe that proving the existence of such a ψ will not be very difficult. Further, the ψ that we find must be computable in time less than poly$(k) \cdot 2^{O(n+d)}$.

Combining Lemmas 3.1 and 3.2, we get the following theorem, which will prove to be the main building block of our blackbox PIT algorithm for the model being discussed.

Theorem 3.3. *Suppose there exists a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2]$ such that for any $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, if $f \nmid g$ then $\psi(f) \nmid \psi(g)$, and such that for any $h \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $h \neq 0$ then $\psi(h) \neq 0$. Suppose a polynomial p is such that $p = f - g_1 g_2 \cdots g_k$, where $f, g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are some polynomials of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d. Then $\psi(p) = 0$ if and only if $p = 0$.*

In the next chapter, we will define a homomorphism ψ, and prove that it has the aforementioned properties. Here, however, we will now prove that the blackbox PIT of polynomials computable by arithmetic circuits of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$ can also be done by coming up with a homomorphism with some algebraic property. Unlike

the previous model, we failed to come up with a homomorphism that satisfies the property required by this model, and thus we could not manage to solve this problem in blackbox. But the following lemmas are still interesting as they give some insights for anyone attempting to solve blackbox PIT for this model. We will present an algorithm to solve the whitebox PIT for this model later in this thesis.

The properties required of a homomorphism that will help in devising an algorithm for blackbox PIT of circuits of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$ will be coprimeness preservation and non-zeroness preservation, or alternatively, squarefreeness preservation and non-zeroness preservation.

A polynomial computed by a circuit of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$ has the form $\prod_{i\in[k]} f_i - \prod_{j\in[k']} g_j$ where f_i, g_j are polynomials in $\mathcal{F}[x_1, x_2, \cdots, x_n]$ of total degree at most b. Similar to the case of previous model, we can prove that a PIT algorithm for such polynomials where the polynomial f_i is assumed to be an irreducible for all i yields us a PIT algorithm without the irreducibility restriction.

Lemma 3.4. *Suppose there exists a blackbox PIT algorithm A for polynomials of the form $f_1 f_2 \cdots f_m - g_1 g_2 \cdots g_k$, where $g_1, g_2, \cdots, g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are some polynomials of degree at most d, and $f_1, f_2, \cdots f_m \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are irreducible polynomials of degree at most d. Further, suppose that the algorithm A runs in time at most $\alpha k^\beta m^\gamma 2^{\delta(n+d)}$ for constants $\alpha, \beta, \gamma, \delta$. Then, there exists an algorithm A' that solves blackbox PIT for polynomials of the form $f_1 f_2 \cdots f_m - g_1 g_2 \cdots g_k$, where for all i, j, $f_i, g_j \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are some polynomials of degree at most d. Further, the algorithm A' runs in time at most $\alpha k^\beta (md)^\gamma 2^{\delta(n+d)}$.*

We will not prove this as the proof is very similar to the proof of Lemma 3.1. Note that assuming all f_i's to be irreducible increases the time complexity by a factor of d^γ, which is small as compared to $2^{\delta(n+d)}$, and thus we can safely assume f_i's to be irreducible polynomials. Further, note that $\alpha k^\beta (md)^\gamma 2^{\delta(n+d)}$ is asymptotically less than $\alpha s^{\beta+\gamma} d^\gamma 2^{\delta(n+d)}$ where s is the size of the circuit, and thus can be considered to be poly$(s) \cdot 2^{O(n+d)}$, as $s > m$ and $s > k$ due to the structure of the circuit.

We now formally state and prove that an explicit non-zeroness and coprimeness preserving homomorphism yields a blackbox PIT algorithm for $\Sigma^2\Pi\Sigma\Pi^b(n)$ circuits.

Lemma 3.5. *Suppose there exists a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2, \cdots x_c]$ such that for any $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, if*

$\gcd(f,g) = 1$ then $\gcd(\psi(f), \psi(g)) = 1$, and such that for any $h \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $h \neq 0$ then $\psi(h) \neq 0$. Suppose a polynomial p is such that $p = f_1 f_2 \cdots f_m - g_1 g_2 \cdots g_k$, where $f_1, f_2, \cdots, f_m \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are non-constant irreducible polynomials of degree at most d, and $g_1, g_2, \cdots g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ have degree at most d. Then $\psi(p) = 0$ if and only if $p = 0$.

Proof. If $p = 0$, then as ψ is a homomorphism, $\psi(p) = 0$. So now let us assume that $p \neq 0$. We will prove that $\psi(p) \neq 0$ by induction on m.

For the base case, we will assume $m = 1$.

If $\exists i \in [k]$ s.t. $f_1 \mid g_i$, we can write $p = f_1 \times (1 - g_1' g_2' \cdots g_k')$, where $g_i' = g_i / f_1$ and $g_j' = g_j, \forall j \neq i$. Thus $\psi(p) = \psi(f_1) \times (1 - \psi(g_1')\psi(g_2') \cdots \psi(g_k'))$. (Note that $\psi(1) = 1$ as ψ is a homomorphism). By the non-zeroness preserving property of ψ, we know that $\psi(f_1) \neq 0$, so all we need to prove is that $\psi(g_1')\psi(g_2') \cdots \psi(g_k')) \neq 1$. Now if $g_1' g_2' \cdots g_k'$ is a constant, then p has degree at most d, and thus $\psi(p) \neq 0$ by the non-zeroness preserving property of ψ. If it is not a constant, there exists some g_j' of degree at least 1. We claim that $\psi(g_j')$ cannot be a constant. To see this, assume that $\psi(g_j') = c$ where c is a constant. Then, $\psi(g_j' - c)$ must be 0, which contradicts the non-zeroness preserving property of ψ. Now since $\psi(g_j')$ has degree at least 1, $\psi(g_1')\psi(g_2') \cdots \psi(g_k'))$ also has degree at least 1, and thus cannot be equal to 1.

If, on the other hand, $f_1 \nmid g_i$ for all $i \in [k]$, then it means that $\gcd(f_1, g_i) = 1$ for all i, as f_1 is an irreducible. Thus $\gcd(\psi(f_1), \psi(g_i)) = 1$ for all i, and thus $\gcd(\psi(f_1), \psi(g_1)\psi(g_2) \cdots \psi(g_k)) = 1$. Thus $\psi(f_1) \neq \psi(g_1)\psi(g_2) \cdots \psi(g_k)$, and thus $\psi(p) \neq 0$. This completes the base case.

Now let us assume the lemma to be true for $m < l$. We need to prove it for $m = l$.

Again, if $f_l \mid g_j$ for some j, then we can write p as $f_l \times (f_1 f_2 \cdots f_{l-1} - g_1' g_2' \cdots g_k')$, where $g_j' = g_j / f_l$ and $g_i' = g_i, \forall j \neq i$. Now $\psi(f_l) \neq 0$ by non-zeroness preserving property of ψ, and $\psi(f_1 f_2 \cdots f_{l-1} - g_1' g_2' \cdots g_k') \neq 0$ by induction hypothesis. Thus $\psi(p) \neq 0$.

If $f_l \nmid g_i$ for all i, then $\gcd(f_l, g_i) = 1$ for all i. Thus $\gcd(\psi(f_l), \psi(g_i)) = 1$ for all i. Thus $\gcd(\psi(f_l), \psi(g_1)\psi(g_2) \cdots \psi(g_k)) = 1$. So $\psi(f_l) \nmid \psi(g_1)\psi(g_2) \cdots \psi(g_k)$ as f_l is not a constant and thus $\psi(f_l)$ is not a constant. Since $\psi(f_l)$ divides $\psi(f_1)\psi(f_2) \cdots \psi(f_m)$, $\psi(f_1)\psi(f_2) \cdots \psi(f_m) \neq \psi(g_1)\psi(g_2) \cdots \psi(g_k)$ and thus $\psi(p) \neq 0$. □

Again, note that for this result to yield a blackbox PIT algorithm, we need an explicit ψ, and the running time of the algorithm must include time spent in computing ψ. Lemmas 3.4 and 3.5 combine to give us a blackbox PIT algorithm for $\Sigma^2\Pi\Sigma\Pi^b(n)$ circuits, provided such an explicit ψ exists. This model of $\Sigma^2\Pi\Sigma\Pi^b(n)$ circuits is a generalization of earlier model of polynomials of the form $f - \prod g_i$, and similarly, the condition of coprimeness preservation is indeed a generalization of indivisibility preservation, as one would expect.

Lemma 3.6. *Suppose there exists a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2, \cdots x_c]$ such that for any $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, if $\gcd(f, g) = 1$ then $\gcd(\psi(f), \psi(g)) = 1$, and such that for any $h \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $h \neq 0$ then $\psi(h) \neq 0$. Then for all $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$, if $f \nmid g$ then $\psi(f) \nmid \psi(g)$.*

Proof. Suppose $\exists f, g$ such that $f \nmid g$ but $\psi(f) \mid \psi(g)$. Let $h = \gcd(f, g)$ and $f' = f/h$, $g' = g/h$. Now $\gcd(f', g') = 1$. Thus $\gcd(\psi(f'), \psi(g')) = 1$. $\psi(f) = \psi(f')\psi(h)$ and $\psi(g) = \psi(g')\psi(h)$. Suppose α is the highest power of $\psi(f')$ that divides $\psi(h)$. That is, $\psi(f')^\alpha \mid \psi(h)$ but $\psi(f')^{\alpha+1} \nmid \psi(h)$. Then $\psi(f')^{\alpha+1} \mid \psi(f)$ but $\psi(f')^{\alpha+1} \nmid \psi(g)$ as $\gcd(\psi(f'), \psi(g')) = 1$. So $\psi(f)$ cannot divide $\psi(g)$. □

The resultant is an important algebraic tool when dealing with coprimeness of polynomials. The resultant of two polynomials is zero if and only if they are not coprime. So, if a homomorphism is to preserve the coprimeness of any two coprime polynomials, it should essentially preserve the non-zeroness of their resultant. This gives us another way of looking at this problem. However, the resultant of the polynomials f and g, considering them as $f, g \in \mathbb{F}(x_1, x_2, \cdots x_{n-1})[x_n]$, has degree at most $2d^2$. Using a homomorphism that is based on the sparse PIT map, and that preserves all polynomials of degree $2d^2$ will require time of the order $\binom{n+d^2}{n}$, which is not acceptable. (The recent result [AFGS17] by Sumanta and others dictates what is acceptable – a factor of $2^{O(n+d)}$ is acceptable but anything more than that is not.)

A polynomial f is called a squarefree polynomial if for all polynomials p of degree at least 1, $p^2 \nmid f$. If f and g are coprime, fg is squarefree, and if f and g are not coprime then fg is not squarefree. A well known fact about squarefree polynomials (over characteristic zero fields) helps us in relating squarefreeness and coprimeness in the other direction.

Fact 3.7. *If $f \in R[x]$ is a squarefree univariate polynomial over a zero-characteristic ring R, then $\gcd(f, f') = 1$ where f' denotes the derivative of f.*

So now if f, g are coprime polynomials in $\mathbb{F}[x_1, x_2, \cdots x_n]$, then considering fg as a univariate polynomial in $\mathbb{F}(x_1, x_2, \cdots x_{n-1})[x_n]$, and assuming $\mathbb{F}$, we can say that $\gcd(fg, (fg)') = 1$ if and only if fg is squarefree. So, in conclusion, in the case of zero-characteristic fields, a squarefreeness preserving homomorphism for polynomials with degree at most $2d$ is also a coprimeness preserving homomorphism for polynomials with degree at most d; and a coprimeness preserving homomorphism for polynomials with degree at most d also preserves the squarefreeness of polynomials with degree at most d. This gives us the following lemma:

Theorem 3.8. *Suppose $\mathbb{F}$ is a characteristic zero field. Suppose there exists a homomorphism ψ from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[x_1, x_2, \cdots x_c]$ such that for any $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most $2d$, if f is squarefree then $\psi(f)$ is squarefree, and such that for any $h \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, if $h \neq 0$ then $\psi(h) \neq 0$. Suppose a polynomial $p \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ is computed by an arithmetic circuit of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$. Then $\psi(p) = 0$ if and only if $p = 0$.*

3.2 Finding "Good" Homomorphisms

Now that we have identified the properties that a homomorphism must have, if it is to be useful for blackbox PIT of the models under consideration, the next task is to find a homomorphism that preserves these properties. A good candidate to begin with is the homomorphism used in blackbox PIT for sparse polynomials, which reduces an n-variate polynomial to a univariate polynomial. This map already preserves the non-zeroness of polynomials of degree d, if we define a sparse polynomial as a polynomial having sparsity at most $\binom{n+d}{d}$, which is the maximum sparsity an n-variate polynomial of degree d can have. Unfortunately, this homomorphism does not exhibit the indivisibility preserving property, let alone the coprimeness preserving property.

So, we will slightly modify this map to come up with a map that preserves indivisibility. We could not prove or disprove that it preserves coprimeness. Proving that it indeed preserves coprimeness will yield us a blackbox PIT for arithmetic circuit of the form $\Sigma^2\Pi\Sigma\Pi^b(n)$. The actual description of this homomorphism is given in Section 4.2.

The way we will prove that this homomorphism preserves indivisibility is that we will devise a deterministic algorithm to test divisibility of n-variate degree d polynomials, and then we will show that the homomorphism (say, ψ), in some sense, 'preserves' each step of the algorithm, thus implying that the output of the algorithm will not change if the inputs f, g to the algorithm are replaced by $\psi(f), \psi(g)$. This gives us a brief and vague outline of the proof that the homomorphism preserves indivisibility. We will be able to formally prove this in the next chapter after we describe the divisibility testing algorithm and the homomorphism. The divisibility testing algorithm can be further modified to get an algorithm to compute GCD of two polynomials. We discuss the two algorithms in the next few chapters.

Chapter 4

The Divisibility Testing Algorithm

In this chapter, we will describe our division algorithm. From the discussion in the previous chapter, it is clear that we need some homomorphism that will preserve each step of this algorithm. Further, the running time of the algorithm is not especially important for us, as we will not be using this algorithm as a subroutine of our PIT algorithm. Still, we will give an algorithm to check for divisibility of n-variate degree d polynomials in time poly$\left(\binom{n+d}{d}\right)$. We need this algorithm to be deterministic, and not necessarily blackbox. We assume the polynomials are given to us in fully expanded form, that is, as a list of monomial-coefficient pairs. Given an arithmetic circuit of size s for an n-variate degree d polynomial, it takes poly$\left(s \cdot \binom{n+d}{d}\right)$ time to compute the polynomial in its expanded form. So, even if we are given (whitebox) arithmetic circuits for the polynomials, we can run our algorithm on them by first computing their expanded form, and this will only incur extra cost of the order poly$\left(\binom{n+d}{d}\right)$ (assuming s to be small enough – which is not an unreasonable assumption), so the running time of our algorithm will still be poly$\left(\binom{n+d}{d}\right)$. Our algorithm is a divisibility testing algorithm that outputs whether the divisor divides the dividend, but it can also be modified to output the division when the divisor divides the dividend. This algorithm can also be converted into a blackbox divisibility testing algorithm.

The problem of general divisibility testing is as hard as the problem of general PIT. Forbes, in his work in 2015, [For15] gives a demonstration of the reduction of PIT to the divisibility testing problem: $t \mid f$ if and only if $f = 0$, where t is a new variable that does not appear in f. Conversely, the work by Kopparty, Saraf and Shpilka showed

that one can solve the blackbox multivariate factoring problem, and thus blackbox divisibility testing problem, given access to a blackbox PIT algorithm [KSS14].

In 1990, work by Kaltofen and Trager gave a poly(n, d)-time blackbox randomized polynomial factoring algorithm for n-variate degree d polynomials over certain fields [KT90]. Polynomial factorization algorithm factors a polynomial into irreducible polynomials, so to use a polynomial factoring algorithm for divisibility testing, one needs to check whether the multiset of the factors of the first polynomial is fully contained in the multiset of the factors of the second polynomial. Checking this will essentially require whitebox PIT (for the difference between a factor of first polynomial and a factor of the second polynomial), if the inputs and output of the algorithm are arithmetic circuits. The sparse PIT algorithm solves the PIT problem in time $2^{O(n+d)}$, so this may not be a big issue if one is only interested in divisibility testing in time $2^{O(n+d)}$. On the other hand, if we allow the input to the algorithm to be polynomials in expanded form, then the input itself may take size of the order $\binom{n+d}{d}$, and thus this algorithm will be no better than our algorithm. Further, the algorithm by Kaltofen and Trager is a randomized algorithm that consumes $\Omega(d^2)$ random bits, as it crucially applies the randomized PIT algorithm on the resultant of the input polynomial and its derivative. Derandomizing it in the naïve way will make its complexity $2^{\Omega(d^2)}$, which is even worse than our algorithm which tests divisibility in time $2^{O(n+d)}$. For the purpose of PIT, we cannot afford an algorithm taking time more than $2^{O(n+d)}$. The algorithm by Kaltofen and Trager also works for only certain base fields, and it cannot be used for the fields in which univariate polynomial factoring does not have a deterministic polynomial time algorithm. For example for finite fields of characteristic p, we only have a randomized algorithm that takes $O(\log p)$ bits to compute univariate polynomial factoring. The naïve derandomization of this algorithm will make the time complexity $O(p)$, which is considered to be exponential, as representing p only takes $\log p$ bits in the input. We note here that our algorithm does not depend on the characteristic of the field.

Another problem with using the algorithm by Kaltofen and Trager, or any other divisibility testing algorithm for that matter, for our PIT applications is that we also need a homomorphism such that the steps of a divisibility testing algorithm are preserved after applying the homomorphism to apply Theorem 3.3, and we have no reason to believe that there exists such a homomorphism for the algorithm by Kaltofen and Trager.

Michael Forbes, in his paper in 2015, discusses the problem of blackbox divisibility testing of polynomials and its connections with PIT [For15]. He gives a deterministic

polynomial time algorithm for a special case of the problem, when the divisor is a quadratic polynomial, and the dividend is a sparse polynomial. Our algorithm discusses a more general version of the problem, but it has a higher time complexity.

A lot of work has been done on factoring supersparse polynomials in one or two variables. A supersparse polynomial has degree exponential in the size of input, but has small sparsity. In 1977, Plaisted showed that the problem of computing GCD, factoring or divisibility testing of univariate supersparse polynomials, in general, is NP-hard [Pla77]. Kaltofen and Koiran, in 2005, gave an algorithm to find all the linear or quadratic factors of a supersparse bivariate polynomial [KK05]. This was later generalized by Grenet to all the constant-degree factors in 2014 [Gre14], and for multivariates in 2016 [Gre16].

4.1 The Divisibility Testing Algorithm

Our algorithm for divisibility testing is inspired by the techniques used in the theory of resultant, which reduce an algebraic problem concerning polynomials to a problem of linear algebra. We will now describe a rough sketch of the algorithm, before formally describing and proving it. We will first construct an algorithm for divisibility testing where the input polynomials have a a non-zero constant term (that is, the polynomials do not evaluate to zero at the point $(0, 0, \cdots 0)$). Such an algorithm can then be extended to general polynomials by applying a random shift to the variables, where each variable x_i is replaced by $x_i + t_i$. We divide the polynomial modulo a monomial ideal first, and then see the conditions on the result which will imply divisibility in the original ring. If the polynomials given to us are degree d polynomials in $\mathbb{F}[x_1, x_2, \cdots x_n]$, we consider the monomial ideal generated by all the monomials of the polynomial ring of degree $d + 1$ here. We will prove that every polynomial which has a non-zero constant term is invertible modulo this ideal, and thus completely divides every other polynomial modulo this ideal. We will then prove that the division modulo this ideal that we obtain is the actual division if the divisor completely divides the dividend in the polynomial ring. We can then check divisibility by multiplying (in the polynomial ring) the divisor with the result of division modulo the ideal and checking if it is equal to the dividend.

We now formally state the problem of divisibility testing here. Suppose f, g are polynomials of $\mathbb{F}[x_1, x_2, \cdots x_n]$ with degree at most d. Then we need a deterministic algorithm which will check whether $f \mid g$ when the polynomials f, g are given in fully expanded form as input to the algorithm.

The following lemma states that we can safely assume the divisor to have non-zero constant term:

Lemma 4.1. *Suppose there exists a deterministic algorithm A, which takes as input two polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, with the promise that $f(\bar{0}) \neq 0$, and correctly checks whether $f \mid g$ or not, in time at most $\alpha \cdot \binom{n+d}{d}^{\beta}$ for some constants α, β. Then there exists an algorithm A', which takes as input any two polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ both of degree at most d, and correctly checks whether $f \mid g$ or not, in time at most $\alpha \cdot \binom{n+d}{d}^{\beta}$.*

Proof. The sparse PIT algorithm discussed in Section 2.4.3 tells us that for any polynomial $p \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d and sparsity m, there exists a hitting set $\mathcal{H} \subseteq \mathbb{F}^n$ such that $|\mathcal{H}|$ is $O(m^2 n^2 \log m)$. Also, $m \leq \binom{n+d}{d}$. Thus, given a polynomial f, we have a set $\mathcal{H}$ such that there exists a point $\bar{t} \in \mathcal{H}$ such that $f(\bar{t}) \neq 0$.

Now suppose we are given a polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d and sparsity m, which evaluates to 0 at $\bar{0}$. We claim that we can obtain a polynomial which has a constant term, or in other words, which is not zero at $\bar{0}$, from f, by shifting f by a point $\bar{t}$ such that $f(\bar{t}) \neq 0$. That is, we apply the isomorphisms ϕ_j, as defined below, on polynomials of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to get polynomials of $\mathbb{F}[y_1, y_2, \cdots, y_n]$:

$$\phi_i(p) := \begin{cases} p & \text{if } p \in \mathbb{F} \\ y_j + t_{ij} & \text{if } p = x_j \text{ for some } j \in [n], \\ \phi_i(p_1) \cdot \phi_j(p_2) & \text{if } p = p_1 \cdot p_2 \text{ for some } p_1, p_2 \in \mathbb{F}[x_1, x_2, \cdots, x_n] \\ \phi_i(p_1) + \phi_j(p_2) & \text{if } p = p_1 + p_2 \text{ for some } p_1, p_2 \in \mathbb{F}[x_1, x_2, \cdots, x_n] \end{cases}$$

where $\bar{t_i} \in \mathcal{H}$, which is the hitting set of f. We will not prove that ϕ is indeed a ring isomorphism, but we do not believe it to be too difficult. We will now prove that there exists a ϕ_j as defined above, such that $(\phi_j(f))(\bar{0}) \neq 0$.

Since $\mathcal{H}$ is a hitting set for f, there exists some i, such that for $\bar{t_i} \in \mathcal{H}$ $f(\bar{t_i}) \neq 0$. Now look at the corresponding isomorphism ϕ_i. By definition of ϕ_i, $(\phi_i(f))(\bar{0}) = f(\bar{t_i}) \neq 0$. Thus, $\phi_i(f)$ is not zero at $\bar{0}$, or, in other words, it has non-zero constant term.

Now we will describe the algorithm A', which is essentially the same as the algorithm A, but with some additional preprocessing. In the beginning, we check if f has non-zero constant term. If not, then we construct the hitting set $\mathcal{H} = \{\bar{t_1}, \bar{t_2}, \cdots, \bar{t_k}\}$ (say) for f, using the sparse PIT algorithm. Then we pick a point $\bar{t_i} \in \mathcal{H}$ such that $f(t_i) \neq 0$, and construct a homomorphism ϕ_i using $\bar{t_i}$, as defined above. The argument above tells us that $(\phi_i(f)) \neq 0$, and since ϕ_i is an isomorphism, we also know that $\phi_i(f) \mid \phi_i(g)$ if and only if $f \mid g$. So we now apply the algorithm A on $\phi_i(f), \phi_i(g)$, and return the output of A as our output.

The extra preprocessing step has three steps – constructing $\mathcal{H}$, using it to find appropriate shifts and actually shifting f and g. The first step takes time $O(\binom{n+d}{n} \cdot n \log d)$ time, by the analysis of the sparse PIT algorithm. The second step takes $|\mathcal{H}|$ time. In the third step, we need to compute $\phi_i(f)$ and $\phi_i(g)$ in the fully expanded form. Each coefficient of $\phi_i(f)$ can be computed in time poly$\binom{n+d}{d}$, and thus the third step takes time poly$\binom{n+d}{d}$. So, all the preprocessing can be done in time at most poly$\binom{n+d}{d}$, and thus A' runs in time at most poly$(\binom{n+d}{d})$. □

This lemma tells us that we can safely assume the divisor (or even both the polynomials) to have non-zero constant term. Now, we define the monomial ideal I_d of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ as the ideal generated by all the monomials of degree $d+1$.

$$I_d := \langle \bar{x}^{\bar{e}} \colon \sum_{i \in [n]} e_i = d+1 \rangle$$

Now we will claim that every polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ which has non-zero constant term, and which has degree at most d is uniquely invertible modulo I_d.

Theorem 4.2. *Suppose f is a polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ with degree at most d, such that $f(\bar{0}) \neq 0$. Suppose the ideal I_d of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ is defined as $I_d = \langle \bar{x}^{\bar{e}} \colon \sum_{i \in [n]} e_i = d+1 \rangle$. Then there exists a unique $g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ such that degree of g is at most d and $fg \equiv 1 \pmod{I_d}$. Further, $g(\bar{0}) \neq 0$.*

Proof. There are multiple approaches to prove this, each with its own merits. Later in this thesis, we will see another more structured, although slightly more complicated, way of proving this theorem.

The cleanest way to prove this is via formal power series. We can consider f to be a formal power series over n variables. A formal power series over n variable is invertible

(in the domain of formal power series) if it has a non-zero constant term. So, f is invertible in the domain of formal power series. Let g be the inverse of f in this domain. Now, we define the polynomial g' as the degree $\leq d$ part of g. Suppose g'' is the degree greater than d part of g. Then, $g'' \cdot f$ is a power series which has zero as the coefficient of all monomials of degree at most d. But $g \cdot f$ is the power series 1. Thus, $g' \cdot f$ must be 1 modulo I_d.

However, since we have not formally defined the formal power series, we will prove this only using elementary algebra here, by induction on the degree of f. This proof may seem a bit tedious, but it has the advantage of being constructive, as we will actually construct the inverse od f recursively.

Base Case: When the degree of f is zero, $f \in \mathbb{F}$, and thus there exists a $g \in \mathbb{F}$ such that $fg = 1$, and thus $fg \equiv 1 \pmod{I_0}$. Degree of this g is also 0, and $g \neq 0$, so $g(\bar{0}) \neq 0$. This completes the base case.

Induction Step: Now suppose the theorem is true for all values of d less than some l. We will now prove that it then must be true for $d = l$. Suppose f is a polynomial of degree l. Suppose $f = f_d + f_{<d}$, where $f_{<d}$ has degree strictly less than d, and f_d is a homogeneous polynomial of degree d. By induction hypothesis, there exists a g of degree less than d such that $f_{<d} \cdot g \equiv 1 \pmod{I_{d-1}}$. Now, we define the polynomial h as $h := f \cdot g = f_d \cdot g + f_{<d} \cdot g$. We split h into three parts: $h_{>d}$, which has no monomial of degree less than d (or in other words, $h_{>d} \in I_d$); h_d, which is a homogeneous polynomial of degree d; and $h_{<d}$, which is a polynomial of degree at most $d-1$.

Now recall that $h = f_d \cdot g + f_{<d} \cdot g$. In $f_d \cdot g$ there exists no monomial of degree less than d, as f_d is a homogeneous polynomial of degree d. The only monomial of degree less than d in $f_{<d} \cdot g$ is 1, by the definition of g. So, $h_{<d} = 1$. So, $f \cdot g \equiv 1 + h_d \pmod{I_d}$ where h_d is a homogeneous polynomial of degree d. Now, by assumption, $f_{<d}$ has a non-zero constant term, say c. Then, $h_d f = h_d f_d + h_d f_{<d} \equiv 0 + c h_d \pmod{I_d}$. These two congruences give us a way to construct the inverse of f modulo I_d. We define $g' := g - c^{-1} h_d$. Then, $fg' = fg - c^{-1} f h_d \equiv 1 + h_d - h_d \equiv 1 \pmod{I_d}$.

So, g' is the inverse of f modulo I_d. Also note that degree of g' is at most d, as degree of g is less than d, and $-c^{-1}h_d$ is a homogeneous polynomial of degree d. Further, g' has a non-zero constant term, as g has a non-zero constant term.

We will prove the uniqueness of inverse by contradiction. Suppose there exist two distinct polynomials g_1 and g_2 in $\mathbb{F}[x_1, x_2, \cdots, x_n]$, both of degree at most d, such that $fg_1 \equiv fg_2 \equiv 1 \pmod{I_d}$ Then, $f(g_1 - g_2) \in I_d$. Since $g_1 - g_2$ is a non-zero polynomial of degree at most d, there exists a $d' \in [d]$ such that d' is the smallest degree among the degrees of monomials that are present in $g_1 - g_2$. Let $g_{d'}$ be the degree-d' homogeneous part of $g_1 - g_2$, and let $g_{>d'}$ be the degree greater than d' part of $g_1 - g_2$. Further, c be the constant term of f, and f' be the degree ≥ 1 part of f. Since $f(g_1 - g_2) \equiv 0 \pmod{I_d}$, $(c + f')(g_{d'} + g_{>d'}) \equiv 0 \pmod{I_d}$. Thus, $cg_{d'} + cg_{>d'} + f'g_{d'} + f'g_{>d'} \equiv 0 \pmod{I_d}$. However, $cg_{d'}$ is a non-zero homogeneous polynomial of degree d' for some $d' \leq d$, and none of the other three summands can have a monomial of degree less than $d' + 1$. Thus $cg_{d'} + cg_{>d'} + f'g_{d'} + f'g_{>d'}$ cannot be zero modulo I_d, which is a contradiction. This proves the uniqueness of the inverse. □

Note that the proof of Theorem 4.2 is constructive, and we get a recursive way to construct the inverse. The existence of a unique inverse for each polynomial with non-zero constant term implies that any polynomial can be completely divided by a polynomial with non-zero constant term, modulo I_d. Further, by arguments similar as above, we can say that the division is unique, modulo I_d. Now, if f divides g in the polynomial ring, then there exists a polynomial h such that $g = fh$. This identity will also hold modulo I_d, that is, $g \equiv fh \pmod{I_d}$. Now, the uniqueness of division modulo I_d implies that the recursive algorithm that Theorem 4.2 must output this same h.

The output of the recursive algorithm is a polynomial of degree at most d. So, if we multiply the output of the recursive algorithm with the divisor, we will get a polynomial of degree at most $2d$. This polynomial must be equal to the dividend if the divisor divides the dividend in the polynomial ring. Moreover, if the divisor does not divide the dividend, this polynomial cannot be equal to the dividend. Thus to checking the divisibility of dividend with divisor is equivalent to checking whether this polynomial is equal to the dividend or not. If the polynomials that we are dealing with are provided to us in expanded form, this checking can be done in time $O(\binom{n+2d}{d})$, and even otherwise, the sparse PIT algorithm can do it in time poly$\binom{n+2d}{d}$. This provides us a rough sketch of the divisibility testing algorithm. We now formalize this in the following theorem:

Theorem 4.3. *Suppose $f, g \in \mathbb{F}[x_1, x_2, \cdots x_n]$ are polynomials of degree at most d, such that $f(\bar{0}) \neq 0$. Let the degree of f be d_f and the degree of g be d_g. The*

monomial ideal I_d of $\mathbb{F}[x_1, x_2, \cdots x_n]$ is defined as $I_d = \langle \bar{x}^{\bar{e}} : \sum_{i\in[n]} e_i = d+1 \rangle$. Then, there exists a unique polynomial h such that $fh \equiv g \pmod{I_d}$ and such that degree of h is at most d. Further, $fh = g$ if and only if f divides g in $\mathbb{F}[x_1, x_2, \cdots x_n]$. Also, degree of h is $d_g - d_f$ if f divides g in $\mathbb{F}[x_1, x_2, \cdots x_n]$, and it is more than $d - d_f$ otherwise.

Proof. Theorem 4.2 tells us that there exists a unique polynomial p, of degree at most d, such that $fp \equiv 1 \pmod{I_d}$. Now, to prove the existence of h, we just multiply p by g, modulo I_d. Suppose h is a polynomial of degree at most d, such that $pg \equiv h \pmod{I_d}$. Then, $fh \equiv fpg \equiv g \pmod{I_d}$.

Now we prove the uniqueness of h. Suppose there exists a polynomial h' such that $fh' \equiv g \pmod{I_d}$. Then, $f(h - h') \equiv 0 \pmod{I_d}$. But since f has a non-zero constant term, to cancel that term out, $h - h'$ must be zero modulo I_d, by argument similar to that in the proof of Theorem 4.2.

Now suppose $f \mid g$. Then there exists a polynomial p, of degree $d_g - d_f$, such that $fp = g$. Then, $fp \equiv g \pmod{I_d}$. But, by uniqueness of h as discuss above, we have $p \equiv h \pmod{I_d}$. Since both h and p have degree less than d, p must be equal to h, and so $fh = g$, and degree of h is $d_g - d_f$.

f On the other hand, if $f \nmid g$, then there cannot exist a polynomial p such that $fp = g$. Thus, $fh \neq g$. Also, if degree of h is not more than $d - d_f$, then degree of fh is not more than d. But since $fh \equiv g \pmod{I_d}$, and since g has degree less than d, the nature of I_d dictates that $fh = g$, which contradicts our assumption that $f \nmid g$. Thus degree of h must be at least $d - d_f$. □

In the discussion so far, we check the divisibility with the help of the ideal I_d as defined in the previous theorems. However, the Theorems 4.2 and 4.3 can be slightly generalized by generalizing the ideal I_d. The proofs of the Theorems 4.2 and 4.3 will work well with any the monomial ideal, such that it contains all but finitely many monomials of the ring. An example of an such an ideal, other than I_d as defined previously, is the ideal $\langle {x_i}^{d_i} : i \in [n] \rangle$, for some d_i's.

One can prove this generalized version of Theorem 4.3 by considering the polynomials as formal power series, but we will just state it without proof.

Theorem 4.4. *Let I be a monomial ideal of $R[x_1, x_2, \cdots x_n]$ such that the number of monomials of $R[x_1, x_2, \cdots x_n]$ that do not appear in I is finite. Suppose $f, g \in$*

$R[x_1, x_2, \cdots x_n]$ are polynomials such that none of them contain any monomial that is present in I, and f is such that $f(\bar{0})$ is a unit in R. Then, there exists a unique polynomial h such that $fh \equiv g \pmod{I}$ and such that h does not have any monomial of I. Further, $fh = g$ if and only if f divides g in $R[x_1, x_2, \cdots x_n]$.

The discussion so far gives us an for divisibility testing of polynomials, given below as Algorithm 1. Now we determine the running time of this algorithm. First, we assume

Algorithm 1: The Divisibility Testing Algorithm

Input : $n, d \in \mathbb{N}$, two polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, such that $f \neq 0$

Output: 1 if $f \mid g$, 0 otherwise.

1 Find a point $\bar{t}$ such that $f(\bar{t}) \neq 0$, using the sparse PIT algorithm.
2 Shift f and g by $\bar{t}$ to get polynomials f', g' respectively.
3 Let c be the constant term of f'. Then define h_0 as c^{-1}. Define $i = 0$.
4 **while** $i < d$ **do**
5 Compute $f' \cdot h_i$. Define p_i as the degree-$(i+1)$ part of $f' \cdot h_i$.
6 Define h_{i+1} as $h_i - c^{-1} p_i$.
7 Define h as the degree $\leq d$ part of $h_d \cdot g$.
8 **if** $f \cdot h = g$ **then**
9 **return** 1
10 **else**
11 **return** 0

that the inputs given to us are in the expanded form, and not arithmetic circuits. Then the first two steps take time polynomial in $\binom{n+d}{d}$, by Lemma 4.1. Now, i^{th} iteration of the while loop will involve basic operations on polynomials of at most $i+1$, and thus they take time polynomial in $\binom{n+i}{i}$, and thus polynomial in $\binom{n+d}{d}$. Computing h also involves multiplication of two polynomials with degree at most d, and we need not compute terms of degree higher than d in the multiplication, so this also takes time polynomial in $\binom{n+d}{d}$. In the end, we check if $f \cdot h = g$. This requires PIT of an n-variate polynomial of degree $2d$, and the sparse PIT algorithm can solve this in time polynomial in $\binom{n+2d}{n}$, which again can be written as a polynomial in $\binom{n+d}{d}$. Thus Algorithm 1 runs in time $O(\binom{n+d}{d}^c)$, for some constant c.

Now if the inputs are given to us in circuit form, steps 2 can be done in time poly(s, n, d). Further, the while loop can be run in time poly(n, d), when we interpret computing h_i as constructing the circuit for h_i. As we have seen in Section 2.4.2, given a circuit for a polynomial, we can compute its degree k homogeneous part in time poly(s, n, d)-time. However, the preprocessing step and the final step in which we check if $f \cdot h = g$ need PIT, and these steps prevent us from claiming that the

algorithm can run in time poly(s, n, d). So even if the inputs are given as arithmetic circuits, Algorithm 1 requires poly$\binom{n+d}{d}$ time in the worst case.

Although the discussion in this section and the above algorithm only admits polynomials over fields as inputs, all we really need is that the divisor input should have a constant term that is invertible in the base ring, as Theorem 4.4 suggests. For polynomials over fields, this has a nice workaround in that we can shift a polynomial which does not have a non-zero constant term, to get a polynomial which has a non-zero constant term. But if the polynomial provided to us already has invertible constant term in the base ring, then we can still run the same algorithm to test the divisibility for polynomials over rings Note that in such a case, we skip the preprocessing steps in that case (first two steps) and assume $f' = f$ from the third step onwards.

4.2 Divisibility Testing and PIT

In Section 4.1 we described our divisibility testing algorithm. Our main motivation towards testing divisibility, however, was to facilitate a PIT algorithm for polynomials of the form $f - \prod g_i$. Lemma 3.2 tells us that, to solve PIT for polynomials of the above form, we need a homomorphism ψ such that $\psi(f) \mid \psi(g)$ only if $f \mid g$. As discussed earlier, we will prove that a homomorphism that we will define now satisfies this property, and we will do so by showing that each step of the divisibility testing algorithm is 'preserved' by this homomorphism.

Theorem 2.12 tells us that for every $d \in \mathbb{N}$, there exists a homomorphism $\phi_d : \mathbb{F}[x_1, x_2, \cdots, x_n] \to \mathbb{F}[y]$, such that all the monomials of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ with degree at most d are mapped to different monomials of $\mathbb{F}[y]$ under ϕ_d. We know that ϕ_d is such that for all $i \in [n]$, $\phi(x_i) = y^{w_i}$, where $w_i \equiv d^i \pmod{p_d}$ and $w_i < p_d$ for some prime number p_d. The exact value of p_d depends on d, and can only be obtained by running an algorithm, that runs in time poly$\binom{n+d}{d}$. (See the proof of Theorem 2.12 for more details).

We will use a slight modification of ϕ_{2d} as our indivisibility preserving homomorphism. First, we introduce a new variable t, and by mapping x_i to $x_i t$, map each polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to a polynomial of $\mathbb{F}[x_1, x_2, \cdots, x_n, t]$. This map is sometimes called a 'degree-counter', as degree of a polynomial in the domain is equal to the individual degree in t of the image of the polynomial. Then we apply the map ϕ_{2d}

on these polynomials, to get our indivisibility preserving map. Note that ϕ_{2d} will not touch t at all, as it is a map from $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to $\mathbb{F}[y]$.

Formally, the homomorphism $\psi_d : \mathbb{F}[x_1, x_2, \cdots, x_n] \to \mathbb{F}[y, t]$ is such that for all polynomials $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$, $\psi(f)$ is defined to be $f(y^{w_1}t, y^{w_2}t, \cdots, y^{w_n}t)$, where $w_1, w_2, \cdots w_n$ are defined as in the map ϕ_{2d}. Although the definition of ψ_d depends on the value of d, we will drop the subscript d, and call the map ψ henceforth, with the understanding that we mean ψ_d when we say ψ. Usually, the value of d will be equal to the bound on the degree of polynomials under consideration.

We now give a rough sketch of the proof of the fact that ψ preserves indivisibility, that is, $\psi(f) \mid \psi(g)$ if and only if $f \mid g$, for all polynomials f, g of degree at most d.

For any polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree d, and where f_i is defined as the degree-i part of f for all $i \in [d]$, we have $\psi(f) = \psi(f_0) + \psi(f_1) + \cdots + \psi(f_d)$. Now, since f_i is a homogeneous polynomial of degree i, by the definition of ψ, one can see that $\psi(f_i)$ is of the form $t^i \cdot \rho(y)$ for some univariate polynomial ρ. Using this, we will prove that the degree$_t = i$ part of $\psi(f)$ is equal to $\psi(f_i)$, where f_i is the (total) degree-i part of f.

Now we will compare two runs of the divisibility testing algorithm – one with f and g as the inputs, and one with $\psi(f)$ and $\psi(g)$ as the inputs, treating $\psi(f), \psi(g)$ as univariates in the variable t, over the ring $\mathbb{F}[y]$. The polynomials h_i's, used in Algorithm 1, will now be univariates in t, over the ring $\mathbb{F}[y]$ in the second run of the algorithm. We will prove that the polynomial h_i in the second run is exactly the polynomial obtained by applying the map ψ on the corresponding polynomial h_i of the first run, and the polynomial h obtained by the algorithm in the second run is the same as the polynomial obtained by applying ψ on the corresponding polynomial h in the first run. From this, we will be able to conclude that the output of the divisibility algorithm when the input is $\psi(f), \psi(g)$ is same as its output when the input is f, g, and thus we can say that ψ preserves indivisibility.

Now we show that if f is a polynomial with non-zero constant term in $\mathbb{F}[x_1, x_2, \cdots, x_n]$, then $\psi(f)$, when considered to be a univariate in the variable t, has an invertible (in $\mathbb{F}[y]$) constant term. In other words, this mean that $\psi(f)$, evaluated at $t = 0$ gives us an element of $\mathbb{F}$, instead of some polynomial of non-zero degree in $\mathbb{F}[y]$.

Lemma 4.5. *Suppose $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$. We will consider $\psi(f)$ as a univariate polynomial in $R[t]$, where $R = \mathbb{F}[y]$. Then, $\psi(0) = f(\bar{0})$.*

Proof. This is not hard to see, from the definition of ψ. □

Corollary 4.6. *If $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ has a non-zero constant term, then $\psi(f) \in R[t]$ has a constant term invertible in R, where $R = \mathbb{F}[y]$.*

The above corollary helps us in establishing that the divisibility testing algorithm, given as Algorithm 1, can indeed be run of $\psi(f), \psi(g)$, considering them as univariate polynomials over $\mathbb{F}[y]$, for any f, g where $f(\bar{0}) \neq 0$. Note that the algorithm can be run over polynomials over integral domain, as long as the divisor input has a constant term invertible in the integral domain, by skipping the preprocessing steps.

Now we will prove that the degree$_t = i$ part of $\psi(f)$ is equal to $\psi(f_i)$, where f_i is the (total) degree-i part of f.

Lemma 4.7. *Suppose we have a polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, such that $f = f_0 + f_1 + \cdots + f_d$, and for all i, f_i is a homogeneous polynomial of degree i. Then, considering $\psi(f)$ as a univariate in $R[t]$, where $R = [y]$, we have coeff$_{t^i}(\psi(f)) \cdot t^i = \psi(f_i)$ for all i.*

Proof. The image of any homogeneous polynomial under ψ is a monomial in $R[t]$, whose degree is equal to the degree of the homogeneous polynomial, by the construction of ψ. Thus for all i, $\psi(f_i) = t^i \cdot \rho_i(y)$ for some univariate polynomial ρ_i as each f_i is a homogeneous polynomial of degree i. Now, $\psi(f) = \sum_{i \in [d]} \psi(f_i) = \sum_{i \in [d]} \rho_i(y) t^i$. So coeff$_{t^i}(\psi(f)) \cdot t^i = \rho_i(y) t^i = \psi(f_i)$. □

The $\rho_i(y)$'s mentioned in the proof above are actually the polynomials $\phi_{2d}(f_i)$. Further, the discussion in Theorem 2.12 tells us that if p is a polynomial of degree at most d, then $\phi_d(p) = 0$ if and only if $p = 0$. From this, we can conclude that, for a polynomial p, the degree (in t) of $\psi(p)$ is not more than the (total) degree of p, and the two are equal if p has degree at most $2d$. The above lemma also tells us that if $p_{\leq k}$ is the degree $\leq k$ part of a polynomial p, then the degree $\leq k$ part of $\psi(p)$ (when considered as a univariate in $R[t]$, where $R = \mathbb{F}[y]$) is $\psi(p_{\leq k})$.

Now we prove the main theorem of this section, which says that the ψ as defined above preserves the indivisibility of two polynomials, when both of them have degree at most d.

Theorem 4.8. *Suppose we have two polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$, both of degree at most d and such that $f(\bar{0}) \neq 0$. Then $\psi(f) \mid \psi(g)$ if and only if $f \mid g$.*

Proof. If $f \mid g$, then there must exist an h such that $fh = g$. The definition of a homomorphism implies that $\psi(g) = \psi(fh) = \psi(f)\psi(h)$, and thus $\psi(f) \mid \psi(g)$. So all we need to do is to prove that the converse is also true, that is, to prove that if $\psi(f) \mid \psi(g)$ then $f \mid g$. We will do this by running the divisibility testing algorithm on f, g in the first run, and then on $\psi(f), \psi(g)$, considering them as univariates in $R[t]$ in the second run, and then comparing the two runs.

First, we establish the fact that the divisibility testing algorithm can be run on $\psi(f)$, $\psi(g)$, when them as univariates in t. From Corollary 4.6, we know that the constant term in $\psi(f)$ is invertible in $\mathbb{F}[y]$, as $f(\bar{0}) \neq 0$. So we can apply the divisibility testing algorithm on $\psi(f)$, $\psi(g)$ by skipping the preprocessing steps.

Now suppose the polynomials obtained in the while loops of the two runs of the algorithm on inputs f, g, and on inputs $\psi(f), \psi(g)$ (the polynomials described by h_i's in Algorithm 1) are $l_1, l_2, \cdots, l_d$ and $l'_1, l'_2, \cdots, l'_d$, respectively. Further, assume that the polynomial denoted by h in the algorithm is equal to l in the first run, and l' in the second run.

Note that $l_i \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ and $l'_i \in (\mathbb{F}[y])[t]$ for all $i \in d$. Also note that the while loop indeed runs for exactly d steps in both the runs – the while loop runs for as many steps as the degree of the divisor input, and the degree of $\psi(f)$ is equal to the degree of f, as discussed earlier.

Now we will prove that $\psi(l_i) = l'_i$ for all $i \in [d]$. We will do so by induction on i. For the base case, note that $l_0 = l'_0$ is a constant of $\mathbb{F}$, defined by the inverse of the non-zero constant part of f, and thus $\psi(l_0) = l'_0$. Now suppose $\psi(l_i) = l'_i$ for all $i < j$ for some j. We need to show that $\psi(l_j) = l'_j$. Now, l_j is defined as $l_{j-1} - l_0^{-1}p$, where p is the degree-j part of $f \cdot l_{j-1}$, and similarly $l'_j = l'_{j-1} - l_0^{-1}p'$, where p' is the degree-j part of $\psi(f) \cdot l'_{j-1}$. Now note that $l'_{j-1} = \psi(l_{j-1})$, and thus $\psi(f) \cdot l'_{j-1} = \psi(f \cdot l_{j-1})$, and thus by Lemma 4.7, $\psi(p) = p'$. Now as ψ is a homomorphism, $l'_j = l'_{j-1} - l_0^{-1}p' = \psi(l_{j-1}) - l_0^{-1}\psi(p) = \psi(l_j)$. By induction, we conclude that $\psi(l_d) = l'_d$.

Now, the algorithm computes the polynomial $h = h_d \times g$, where $\times$ denotes multiplication modulo the ideal I_d. In our case, the ideal I_d is simply the ideal $\langle t^{d+1} \rangle$. Now l is the degree $\leq d$ part of $l_d \cdot f$, and l' is the degree $\leq d$ part of $\psi(f) \cdot l'_d = \psi(l_d \cdot f)$. So, again by Lemma 4.7, we have $l' = \psi(l)$.

Finally, the algorithm concludes that $f \mid g$ if $hf = g$. So, the algorithm, in the first run, say that $f \mid g$ if $g = lf$, and in the second run, say that $\psi(f) \mid \psi(g)$ if $\psi(g) = \psi(l)\psi(f)$. Since we need to prove that $\psi(f) \mid \psi(g)$ implies $f \mid g$, we need to prove that $\psi(g - fl) = 0$ implies that $g - fl = 0$. Now note that the ϕ_d preserves the non-zeroness of polynomials of degree at most d. Also note that, by the definition of ψ, for any polynomial p, $\psi(p)(1) = \phi_{2d}(p)$. So, $\psi(g - fl)(1) = \phi_{2d}(g - fl)$, and thus if $\psi(g - fl) = 0$, then $\phi_{2d}(g - fl) = 0$, and thus $g - fl = 0$. This implies that the output of the two runs of the algorithm must be the same, and thus ψ preserves indivisibility. □

In the Chapter 6, we will finally link all the pieces together, and describe the algorithm for the blackbox PIT problem when the input is promised to be of the form $f - \prod g_i$, for polynomials f, g_i. The main idea for the algorithm will be to reduce the n-varite polynomial to a bivariate polynomial using the map ψ described here, and then use Theorem 2.6 to do PIT of the bivariate thus obtained.

Chapter 5

The GCD Algorithm

In this chapter, we will describe a deterministic algorithm for computing the GCD of two multivariate polynomials. This algorithm runs time poly$(\binom{n+d}{d})$, when the input polynomials are n-variate polynomials of degree at most d, given in their fully expanded form. The algorithm is a slight modification of the divisibility testing algorithm.

Most of the important algorithms to compute the GCD of two multivariate polynomials, like the EZ-GCD algorithm by Moses and Yun [MY73], have running time dependent on the sparsity of the polynomials, and the sparsity of an n-variate degree d polynomial could be as large as $\binom{n+d}{d}$, which is exponential. In that sense, the algorithm that we give in this chapter, which works in time poly$(\binom{n+d}{d})$, does not perform worse than other algorithms for GCD of two polynomials. The main aim for working towards this algorithm was to solve the PIT problem in the special case of $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuits, and this algorithm can be thought of as a byproduct of our work on PIT. The goal was to demonstrate a homomorphism that will preserve each step of the algorithm, and thus preserve the coprimeness of two polynomials, but we could not come up with a homomorphism that does this. Nevertheless, this algorithm still helps us to solve the whitebox PIT problem in the special case of $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuits.

Tateki Sasaki and Masayuki Suzuki gave three new GCD algorithms, that are fast for some special types of polynomials [SS92]. Zippel in 1979 gave a randomized algorithm (also sometimes called the sparse modular algorithm) for GCD that runs in time polynomial in sparsity [Zip79]. Brown and Traub in 1971 gave an algorithm for GCD that requires time exponential in the number of variables [Bro71]. As mentioned in

the previous chapter, Kaltofen and Trager gave a poly(n, d) time randomized algorithm to factorize an n-variate degree d polynomial [KT90]. Naïve derandomization of this algorithm will result in $2^{\text{poly}(n,d)}$ time gcd algorithm, which requires access to a subroutine that solves PIT of polynomials in $2^{\text{poly}(n,d)}$ time. The subroutine for PIT is not a problem, as even the sparse PIT algorithm can solve the blackbox PIT problem in as much time. However, this algorithm only works for base fields where univariate factorization can be done, and, for example, require time exponential in p for polynomials over a field of characteristic p. Further, our algorithm computes the GCD in time poly$(\binom{n+d}{d})$, which can be described as $2^{O(n+d)}$, which is better than $2^{\text{poly}(n,d)}$. (As discussed in the previous chapter, the poly(n, d) for their algorithm is $\Omega(d^2)$). We note here that Kaltofen's factoring algorithm works even for blackboxes of polynomials, unlike our GCD algorithm.

As mentioned in the previous chapter, Plaisted in 1977 showed that the problem of computing the GCD of two univariate supersparse polynomials is NP-hard, when we consider the input size as $s \log d$, where s is the sparsity and d is the degree of the polynomials [Pla77].

5.1 Background

GCD, or the greatest common divisor, of two elements a and b of a ring R is defined as the element c of R, such that c divides both a and b, and every element of R that divides both a and b also divides c.

A unique factorization domain is defined as an integral domain where each non-zero element of the domain can be written as a product of irreducible elements of the integral domain in unique manner, upto reordering or multiplication by units. Every field is also a unique factorization domain. Naturally, GCD of two elements of a unique factorization is well defined. The domain of multivariate polynomials over a field (or over any unique factorization domain) is a unique factorization domain. Thus every pair of multivariate polynomials have a unique GCD (upto unit multiples).

To compute the GCD of integers, we have the famous Euclid's algorithm, which repeatedly divides the larger integer by the smaller integer and replaces the larger integer by the remainder. The algorithm works well for all unique factorization domains where division with remainder is possible, such as univariate polynomials over a field. The

unique factorization domains where division with remainder is possible are thus called Euclidean domains. Unfortunately, multivariate polynomials over a field (or a ring) do not form a Euclidean domain, and thus Euclid's algorithm cannot be applied here for computing the GCD.

We now assume that $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ are two polynomials of degree d_f, d_g respectively, and that $h = \gcd(f, g)$ has degree d_h. Clearly, $0 \leq d_h \leq d_g$ and $0 \leq d_h \leq d_f$. In the 18^{th} century, Bézout showed that if f, g are elements of a Euclidean domain, then there exist a, b in the same Euclidean domain such that $fa + gb = h$ [Béz79]. This is not true for all unit factorization domains however, and in particular, this is not true for multivariate polynomials.

However, the ring of univariate polynomials is a Euclidean domain. So, the Bézout's identity is true for f, g, when we consider them as polynomials in $\mathbb{F}(x_2, x_3, \cdots, x_n)[x_1]$, where $\mathbb{F}(x_2, x_3, \cdots, x_n)$ is the field of fractions of $\mathbb{F}[x_2, x_3, \cdots, x_n]$. Thus, there exist $a, b \in \mathbb{F}(x_2, x_3, \cdots, x_n)$ such that $fa+gb = h$. This further gives rise to concepts like resultants, which can be used to test coprimeness of two polynomials which, though not directly relevant for our discussion on our GCD algorithm, are useful for our PIT applications. Unfortunately, the degree of the resultant of two degree-d multivariates is $2d^2$, so applying the sparse PIT algorithm on the resultant to check coprimeness will only test coprimeness testing algorithm that runs in time $2^{O(d^2)}$.

The following theorem establishes a connection between GCD and LCM of two polynomials. This is a well-established fact in number theory, and has a simple proof. We will not prove it here.

Theorem 5.1. *Suppose $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$. Let h be the GCD of f, g, and let $l = \frac{fg}{h}$. Then if l' is such that $f \mid l', g \mid l'$, then $l \mid l'$.*

Our GCD algorithm find GCD by finding the LCM of two polynomials first, so this theorem is quite useful to us in our analysis of the GCD algorithm in the following section.

5.2 The GCD Algorithm

We will give a rough sketch of the algorithm in this section. We compute the GCD of two polynomials via computing their LCM first. Suppose the input polynomials

$f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ have degrees d_f, d_g respectively, and the polynomial l is such that $f \mid l, g \mid l$ and if some $l' \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ is such that $f \mid l', g \mid l'$, then $l \mid l'$. We will assume, without loss of generality, that $d_f \leq d_g$. Suppose the degree of l is d_l. Theorem 5.1 tells us that $\gcd(f, g) = \frac{fg}{l}$. We also know that $d_g \leq d_l \leq d_f + d_g$. In the i^{th} step of this algorithm, we will assume d_l to be $d_g + i - 1$, and then we will try to find a polynomial of this degree that is divisible by both f and g. If we succeed, we claim that this polynomial is the LCM of f, g. If we fail, we increment the value to be tried for d_l.

To check if there exists a polynomial of given degree such that both f and g divide it, we assume the coefficients of the polynomial to be some variables. Then we apply the divisibility testing algorithm on this polynomial and f, and then on this polynomial and g. An important fact to note here is that we require the dividend polynomial in the very last steps of the divisibility testing algorithm. Using this fact, we will show that the two runs of the divisibility testing algorithm will give us a homogeneous system of linear equations. This homogeneous system of equations has twice as many equations as variables, and thus may or may not have a non-trivial solution. We find one non-trivial solution of this system, if it exists, and this will give us the desired LCM. The number of variables in the system of equation generated in the i^{th} step is $\binom{n+d_g+i-1}{n}$, and $i \leq d_f$, so we will be able find a solution in time $2^{O(n+d)}$.

We will, however, give a proof that is a little more formal than what we argued above. We first define a few terms that we will use in the rest of the discussion in this section. We define the set P_d as the set of all polynomials of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d. Similarly, we define the set M_d as the set of all monomials of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ that have degree at most d. We define $N_{a,b}$ as the number $\binom{a+b}{b}$. So the number of monomials in M_d is $N_{n,d}$.

We will first observe that P_d is a vector space over $\mathbb{F}$, with the usual polynomial addition and scalar multiplication as the vector operations. This vector space has dimension $N_{n,d}$, and the set M_d serves as its basis. Thus there exists an isomorphism, say ψ, between the vector space P_d and $\mathbb{F}^{N_{n,d}}$. We will then prove that the set of all polynomials of P_d that are divisible by a given polynomial f of degree d_f, forms a subspace of P_d. This will imply that there exists a system of linear equations over $\mathbb{F}$, such that its solutions correspond to the polynomials with degree at most d that are divisible by f. We will describe a way to construct this system of linear equations, and show that it has $N_{n,d}$ variables and $N_{n,d-d_f}$ equations. Similarly another polynomial

g of degree d_g will give us another system of linear equations, whose solution set corresponds to all the polynomials of degree at most d that are divisible by g. The intersection of these two subspaces will be the set of polynomials that are divisible by the LCM of f, g. This intersection corresponds to the points in $F^{Nn,d}$ that satisfy both the system of linear equations, or in other words, another system of equations which contains equations from both the systems of linear equations, corresponding to f and g respectively. Solving this combined system of linear equations corresponding to the LCM for a non-trivial solution will give us a multiple of the LCM. If the LCM has degree more than d, the system cannot have a non-trivial solution. So we take $d = d_g$ in the beginning; obtain the system of linear equations corresponding to the LCM; solve it, if possible, for a non-trivial solution; and if no non-trivial solution is found, increment the value of d and repeat the process. This completes the informal overview of the algorithm, and now we will formally show the algorithm.

Lemma 5.2. *P_d is a vector space over $\mathbb{F}$, with the usual polynomial addition and multiplication (by a constant) as the vector operations. This vector space has dimension $N_{n,d}$, and the set of all monomials of degree at most d serves as its basis.*

Proof. One can easily verify that all the axioms for vector spaces are satisfied by the operations for P_d. Now, suppose the monomials of M_d are not linearly independent. Then, there must exist $c_1, c_2, \cdots, c_{m_{n,d}} \in \mathbb{F}$, not all zero, such that $\sum c_i m_i = 0$, where $m_i \in M_d$ is the i^{th} monomial of degree at most d (Once can always order them in some way, or one can just use the graded lexicographical monomial ordering). This can only happen if all c_i's are zero, so we have a contradiction. So the set of all the monomials of degree at most d is indeed a linearly independent set. Now look at some polynomial p in P_d. Since $p \in P_d$, we know that the degree of p cannot be more than d, and then it can always be expressed as a linear combination of some monomials of degree at most d. □

Corollary 5.3. *The vector space $\mathbb{F}^{N_{n,d}}$ is isomorphic to P_d. Suppose the monomials in M_d are $m_1, m_2, \cdots m_{Nn,d}$. For a vector $\bar{v} \in \mathbb{F}^{N_{n,d}}$, we denote the i^{th} component of $\bar{v}$ by $(\bar{v})_i$. Suppose the map $\psi : P_d \to \mathbb{F}^{N_{n,d}}$ is defined componentwise as $(\psi(p))_i = \text{coeff}_{m_i}(p)$. Then ψ is an isomorphism.*

Proof. If a finite dimensional vector space V over a field F has dimension d, then it is isomorphic to the vector space F^d. If $\{b_1, b_2, \cdots, b_d\}$ is a basis of V, and $\{v_1, v_2, \cdots v_d\}$ is a basis of F^d, then the map that maps $c_1 b_1 + c_2 b_2 + \cdots + c_n b_n$

to $c_1v_1 + c_2v_2 + \cdots + c_nv_n$ for any $c_1, c_2, \cdots, c_n \in F$ is an isomorphism from V to F^d. Thus this corollary follows. □

Now we show that for a polynomials p of degree d_p, the set of polynomials in P_d that are divisible by p form a subspace of P_d.

Lemma 5.4. *Suppose $p \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ is a non-zero polynomial of degree d_p. Then the set of all polynomials in P_d that are divisible by p forms a subspace of P_d. When $d \geq d_p$, the set $\mathcal{B} = \{m \cdot p : m \in M_{d-d_p}\}$ is the basis of this subspace.*

Proof. For any $p_1, p_2 \in P_d$ such that $p \mid p_1, p \mid p_2$, and for all constants $\alpha, \beta \in \mathbb{F}$, $p \mid (\alpha p_1 + \beta p_2)$, so the set of all polynomials in P_d that are divisible by p indeed forms a subspace of P_d. The set $\mathcal{B}$ is obtained by multiplying each monomial of degree at most $d - d_p$ with the polynomial p. If a polynomial f is such that $p \mid f$, then degree of $\frac{f}{p}$ can be at most $d - d_p$ and thus, $\mathcal{B}$ spans the set of all polynomials in P_d that are divisible by p. Now, suppose there exist $c_1, c_2, \cdots c_k$ where $k = N_{n,d-d_p}$ such that $c_1m_1p + c_2m_2p + \cdots + c_km_kp = 0$. So $p(c_1m_1 + c_2m_2 \cdots c_km_k) = 0$. Since p is non-zero, this means that $c_1, c_2, \cdots, c_k$ must all be zero. This shows that $\mathcal{B}$ is also linearly independent. Thus $\mathcal{B}$ is a basis for this subspace. □

We will denote the subspace of P_d consisting of all the polynomials that are divisible by a polynomial p, as S_p. From the theorem above, we get the following corollary, using standard facts about vector spaces and isomorphisms.

Corollary 5.5. *One can construct a homogeneous system of linear equations, with $N_{n,d}$ variables and $N_{n,d-d_p}$ equations, such that its solution space in $\mathbb{F}^{n,d}$ is exactly the image of the subspace given in Lemma 5.4 above, under the isomorphism ψ, as defined in Corollary 5.3 above. Constructing this homogeneous system of linear equations takes poly$(N_{n,d})$ time.*

From this, we can construct a divisibility testing algorithm, as testing subspace membership given the basis for the subspace can be done in time polynomial in the size of the basis. In fact, Algorithm 1 that we provided in the previous chapter is just a variation of this method. Given a non-zero polynomial f of degree d_f, and a polynomial g of degree d, we construct a $N_{n,d-d_f} \times N_{n,d}$ matrix A such that $Ax = 0$ is a homogeneous system of linear equations, and such that if $A\bar{v} = 0$ then $\psi^{-1}(\bar{v})$ is

divisible by f. We can then check if $\psi(g)$ satisfies this system of linear equations. If it does, then we say that $f \mid g$, and $f \nmid g$ otherwise.

Corollary 5.5 also tells us that for any $f, g \in P_d$, we have a homogeneous system of linear equations, such that its solution set is the image under ψ of the set of polynomials divisible by both f and g. This system of linear equations has a non-trivial solution if and only if there exists a non-zero polynomial in P_d which is divisible by both f, g. Further, if we know that there does not exist a non-zero polynomial in P_{d-1}, divisible by both f and g, then this non-trivial solution must correspond to the LCM of f and g. This gives us a way to compute the LCM, and thus also the GCD of two multivariate polynomials. The following algorithm computes the GCD of two multivariate polynomials this way. The discussion so far has made sure that the algorithm must return an output, and the output it returns must indeed be the GCD of the input polynomials.

We now observe a few things about the algorithm. First, note that while GCDs and LCMs are not unique, they are unique upto unit multiplication. This algorithm computes some GCD of the input polynomials. If one wishes to compute a standardized form of the GCD, the algorithm can be easily modified slightly, to divide the output polynomial by an appropriate constant in $\mathbb{F}$. Now we observe that the algorithm works well when the inputs are promised to be non-zero, as Lemma 5.4 requires non-zeroness of the input polynomial. But one can remove this restriction from the algorithm if one wishes, by checking if either of the input polynomials are zero, using the sparse PIT algorithm. If one of the inputs is zero, the problem of finding the GCD becomes trivial. Running the sparse PIT algorithm $2^{O(n+d_g)}$ time, so adding this step will still keep the time complexity of the GCD algorithm $2^{O(n+d_g)}$.

We will analyse the algorithm now, and argue that it terminates in time $2^{O(n+d)}$. Steps 4,5 and 6 require time poly$(N_{n,d-d_f} \times N_{n,d})$, which is less than $2^{O(n+d)}$. Step 7 and 8 are matrix operations that take time polynomial in the size of the matrix, so they can also be run in time $2^{O(n+d)}$. Polynomial division in step 10 can be done using the divisibility testing algorithm that we described in the previous chapter, and that also takes time at most $2^{O(n+d)}$. Doing steps 2 to 7 in a loop will increase the time complexity by a factor of d_f, but the total time required will still remain $2^{O(n+d)}$.

Algorithm 2: The GCD Algorithm

Input : $n, d \in \mathbb{N}$, two non-zero polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degrees d_f, d_g respectively, such that $d_f \leq d_g$.

Output: The GCD of f, g.

for $i = 0$ to d_f **do**
  Define $d := i + d_g$.
  Define the isomorphism $\psi : P_d \to \mathbb{F}^{N_{n,d}}$ componentwise as $(\psi(p))_i = \text{coeff}_{m_i}(p)$, where m_i is the i^{th} monomial of M_d.
  Construct a $N_{n,d-d_f} \times N_{n,d}$ matrix A_f, such that $A_f \cdot \bar{v} = 0$ if and only if $\bar{v} = \psi(p)$ for some polynomial $p \in P_d$ divisible by f.
  Similarly construct a matrix A_g corresponding to the polynomial g.
  Using A_f and A_g, construct a $2N_{n,d-d_f} \times N_{n,d}$ matrix A_{fg}, consisting of rows in A_f and the rows in A_g.
  if A_{fg} *has a non-trivial solution* **then**
    Compute a non-zero $\bar{v}$ such that $A_{fg} \cdot \bar{v} = 0$.
    Find a polynomial $p \in P_d$ such that $\psi(p) = \bar{v}$.
    return $\frac{fg}{p}$.

5.3 The GCD Algorithm And PIT

Our initial intention towards finding a new GCD algorithm was to prove that the homomorphism ψ given in the previous chapter preserves the GCD of two polynomials. We intended to do this by proving that this ψ preserves each step of our GCD algorithm. Unfortunately, we could not prove that ψ preserves each step of this algorithm.

Nevertheless, we still get a whitebox PIT algorithm for the model of $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuits, that runs in time $2^{O(n+d)}$ time. We will formally describe the algorithm in the next chapter. This algorithm uses both the GCD algorithm and the divisibility testing algorithm as subroutines, and uses the fact that the divisibility algorithm can output the division if the divisor divides the dividend.

Chapter 6

The PIT Algorithms

In the previous sections, we described our GCD and divisibility testing algorithms. In Theorem 4.8, we demonstrated, using our divisibility testing algorithm, that the map ψ as defined in Section 4.2 preserves indivisibility. We know that this ψ preserves non-zeroness as well. Lemma 3.2 tells us that if a homomorphism ψ preserves the non-zeroness and indivisibility of polynomials of degree at most d, then for all polynomials $f, g_1, g_2, \cdots g_k$ of degree at most d, we have $\psi(f - \prod g_i) = 0$ if and only if $f - \prod g_i = 0$. This gives us a way to solve the blakbox PIT problem on polynomials promised to be of this form. We find the appropriate map ψ, given the value of d and n, and then construct a blackbox that computes $\psi(C)$ where C is the input blackbox. We then do PIT for $\psi(C)$, using the method provided by Theorem 2.6 for PIT of constant variate polynomials. As we know that $\psi(C) = 0$ if and only if $C = 0$, we can give the correct output using the output of the constant-variate PIT algorithm. Algorithm 3 given below describes this algorithm more succinctly.

Now we briefly discuss the time complexity of the algorithm. The first step requires time polynomial in $\binom{n+d}{d}$, as Theorem 2.12 suggests. Constructing a blackbox for f' using a blackbox for f requires time poly$(\log w_{\mathsf{max}}, n)$ time, and as the discussion in Theorem 2.12 suggests, w_{max} is of the order $2^{O(n+d)}$. Step 3 and 4 require time of the order $O((dw_{\mathsf{max}} + 1)^2)$, which will again be of the order $2^{O(n+d)}$. So the PIT problem can be solved in time at most $2^{O(n+d)}$.

An interesting thing to note here is that the time complexity does not involve the size of the circuit at all. So, the value of k does not have an effect on the efficiency of the

Algorithm 3: The Blackbox PIT Algorithm for Polynomials of the Form $f - \prod g_i$

Input : $n, d \in \mathbb{N}$, a blackbox computing a polynomial C, such that there exist $g_0, g_1, \cdots g_k \in \mathbb{F}[x_1, x_2, \cdots, x_n]$,all of degree at most d, with $C = g_0 - \prod_{i\in[k]} g_i$.
Output: 0 if $C = 0$, 1 if $C \neq 0$.

1 Obtain the values for $w_1, w_2, \cdots w_n$, such that $\bar{w} \cdot \bar{e} \neq 0$ for all $\bar{e} \in \mathbb{N}^n$ with $\sum_{i\in[n]} e_i \leq d$, using the method provided by Theorem 2.12
2 Define $C' \in \mathbb{F}[y, t]$ as $C(y^{w_1}t, y^{w_2}t, \cdots, y^{w_n}t)$.
3 Pick an arbitrary set $S \subseteq \mathbb{F}$ of size at least $dw_{\text{max}} + 1$, where w_{max} is such that $w_{\text{max}} \geq w_i$ for all $i \in [n]$. Evaluate C' at all points of S^2.
4 **if** $\exists \alpha, \beta \in S$ *such that* $C'(\alpha, \beta) \neq 0$ **then**
5 **return** 1.
6 **else**
7 **return** 0.

algorithm. Of course, if the polynomials g_i's are to be all non-constant, the value of k cannot be more than d anyway.

Unfortunately, we could not prove that the same ψ also preserves coprimeness or squarefreeness of polynomials of degree at most d, using the GCD algorithm or otherwise. Thus we could not put to use Theorems 3.5 and 3.8, to get a blackbox PIT algorithm for $\Sigma\Pi^a\Sigma\Pi^b(n)$ circuits. However, the GCD algorithm will still give us a whitebox PIT algorithm for this model, running in time $2^{O(n+d)}$ time. We now describe a rough sketch of the algorithm here.

Suppose $C = f_1 f_2 \cdots f_m - g_1 g_2 \cdots g_k$ is the polynomial computed by the given circuit, with $f_1, f_2, \cdots f_m$ and g_1, g_2, g_k being polynomials of degree at most d. The problem can be restated as checking whether one product of polynomials is equal to another product of polynomials, or in other words, checking whether $f_1 f_2 \cdots f_m = g_1 g_2 \cdots g_k$ for f_i's and g_j's as defined above. We will solve this version of the problem. To do so, we compute the GCD of f_1, g_1 in the first step, divide both f_1, g_j by the GCD, and replace them in the circuit by the result of the division. If f_1 becomes 1 after the division, we remove it, and start again with f_2 as our new f_1. If g_1 becomes 1, we simply remove it. If neither is zero, we then proceed to the next g, computing the GCD of f_1 and g_2 and repeating the whole process. If we reach g_k, and after the division of f_1 by $\gcd(f_1, g_k)$, f_1 still does not become zero, it means that f_1 has some power of an irreducible as a factor that does not divide $\prod g_i$. If so, C cannot be equal to zero. If, on the other hand, after such divisions, the problem reduces to checking whether $\prod 1 = \prod 1$, we conclude that $C = 0$, as we divided both the LHS and the RHS by the same polynomial in each step, and thus if they are equal at the end, they must be equal to begin with. If we run out of polynomials with non-zero degree in the

LHS, and the RHS has some polynomial with non-zero degree, or a different constant than the LHS, then we again conclude that the polynomial was non-zero to begin with. Algorithm 4 given below describes this algorithm more succinctly.

The key observation that helps us in this whitebox algorithm is that the divisibility algorithm can actually compute the division of two polynomials, with the same complexity, if the divisor indeed divides the dividend.

Algorithm 4: The Whitebox PIT Algorithm for polynomials computed by a $\Sigma\Pi^a\Sigma\Pi^b(n)$ arithmetic circuit

Input : $n, d \in \mathbb{N}$, an arithmetic circuit of the form $\Sigma^2\Pi^a\Sigma\Pi^b(n)$ of size s computing a polynomial $C \in \mathbb{F}[x_1, x_2, \cdots, x_n]$.

Output: 0 if $C = 0$, 1 if $C \neq 0$.

1 Suppose $C = \prod_{i\in[m]} f_i - \prod_{j\in[k]} g_j$, with f_i's and g_j's having a $\Sigma\Pi$ circuit of size at most s.
2 **for** $i = 0$ to m **do**
3 **for** $j = 0$ to k **do**
4 Compute $h := \gcd(f_i, g_j)$, using the GCD algorithm.
5 Compute $f := \frac{f_i}{h}$, $g := \frac{g_j}{h}$, using the divisibility testing algorithm.
6 Replace f_i with f and g_j with g.
7 **if** *deg*$(f_i) \neq 0$ **then**
8 **return** 1.
9 **for** $i = 0$ to k **do**
10 **if** *deg*$(g_i) \neq 0$ **then**
11 **return** 1.
12 **if** $\prod_{i\in[m]} f_i = \prod_{j\in[k]} g_j$ **then**
13 **return** 0.
14 **else**
15 **return** 1.

Now we look at the time complexity of the algorithm. In the first step, we obtain $m + k$ depth-2 arithmetic circuits from the input circuit, that compute f_i's and g_j's. This takes time poly(s). The steps 4 and 5 take only $2^{O(n+d)}$ time, by the complexity analysis of the GCD and division algorithms. In step 6, we construct two $\Sigma\Pi$ circuits of size at most $N_{n,d}$, computing f and g respectively, and then replace the circuits for f_i and g_j obtained in step 1 by the circuits for f and g. The division algorithm provides us f and g in the expanded form, so constructing $\Sigma\Pi$ circuits for f, g again takes only $2^{O(n+d)}$ time. Checking whether the degree of a $\Sigma\Pi$ circuit of size a is zero requires only poly(a) time as all one has to do is to check if any of the Π gates has a variable as an input. Since each of the circuits for f_i's and g_j's has size at most $\binom{n+d}{d}$, steps 7 and 10 also take only $2^{O(n+d)}$ time. By the time we reach 12, we have ensured that all f_i's and all g_j's have degree 0. So step 12 involves multiplication of at most s

constants of $\mathbb{F}$, so this cannot take more than poly(s) time. So the algorithm indeed runs in time poly$(s, 2^{n+d})$ time.

Chapter 7

Conclusion and Future Work

The PIT problem remains one of the most elusive problems in theoretical computer science, despite a lot of progress made over the last two decades. We hope that our work in this thesis, and the ideas that we used here, will be of help to further our understanding of the PIT problem. We mostly deal with some special cases of top fan-in 2 and depth-4 circuits in this thesis, and the recent progress in depth reduction (See [AFGS17]) does not allow any restriction on top fan-in, so our result cannot be directly used in conjunction with their result. One can, however, attempt to generalize our results to top fan-in s and depth-4 circuits.

One can also attempt to reduce the time complexity of our PIT algorithms from exponential in $n + d$ to polynomial in $n + d$. If this is achieved, one can successfully solve the whitebox PIT problem for general $\Sigma^2\Pi\Sigma\Pi$ circuits, or the blackbox divisibility testing problem for general polynomials in polynomial time. Either of these results should be considered major breakthroughs in this field.

The proverbial lowest hanging fruit according to us, however, seems to be the construction of a blackbox PIT algorithm for the model of $\Sigma^2\Pi^a\Sigma\Pi^b(n)$ circuits, for which we have given a whitebox algorithm in this thesis. We have demonstrated, in Theorems 3.8 and 3.6, that a homomorphism that preserves non-zeroness and either coprimeness or squarefreeness will give us the blackbox PIT for this model. We could not prove that the ψ that we worked with preserves coprimeness or squarefreeness. We believe that with some more work, and perhaps a slight modification to ψ, one might be able to come up with such a homomorphism.

An interesting direction in which one can work to prove that ψ preserves coprimeness, is the concept of resultant. For pair of two polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ of degree at most d, their resultant is a polynomial $\text{res}(f, g)$ of degree $2d^2$, such that $\text{res}(f, g) = 0$ if and only if $\gcd(f, g) = 1$. The resultant is defined for univariate polynomials of degree d over any unit factorization domain, as a determinant of a $2d \times 2d$ matrix (called Sylvester matrix) whose entries are the coefficients of the polynomials. So, there are actually n different resultants for $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$, each of which is a degree-$2d^2$ and $(n-1)$-variate polynomial. Unfortunately, the sparsity of such a polynomial can be as high as $\binom{n-1+2d^2}{2d^2}$, which is more than $2^{O(n+d)}$, so the naïve application of the sparse PIT algorithm will not help us either. The Sylvester matrix of f, g will contain as entries n-variate polynomials of degree d. So if one manages to find a non-zeroness preserving homomorphism ϕ that also preserves the linear independence of a set of vectors, whose entries are polynomials, then we can prove that this homomorphism must preserve the rank of the Sylvester matrix, effectively preserving the coprimeness of two polynomials.

Our approach to proving that ψ preserves a particular property by constructing an algorithm seems rather indirect. However, this approach has an added advantage that it provides us at least a whitebox algorithm for the problem. One can try some more direct approaches for this problem though.

Theorem 3.8 tells us that a squarefreeness and non-zeroness preserving homomorphism yields us a blackbox PIT algorithm for the model of $\Sigma^2\Pi^a\Sigma\Pi^b(n)$ circuits. This gives us an interesting direction to work on, as we now need to work with only one polynomial, instead of two, as in the case of coprimeness preservation.

An important observation in the direction of coprimeness preservation is that one cannot hope to prove that ψ will map every irreducible polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ to an irreducible polynomial in $\mathbb{F}[y, t]$. In fact, finding examples of irreducible polynomials that get map to reducible polynomials under ψ is not too hard. Yet we expect ψ to preserve coprimeness. In other words, we need to prove that the set of factors of images under ψ of two distinct irreducibles is disjoint.

Bibliography

[AFGS17] Manindra Agrawal, Michael Forbes, Sumanta Ghosh, and Nitin Saxena. Small hitting-sets for tiny arithmetic circuits or: How to turn bad designs into good. *arXiv preprint arXiv:1702.07180*, 2017.

[Agr05] Manindra Agrawal. Proving lower bounds via pseudo-random generators. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 92–105. Springer, 2005.

[AV08] Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 67–75. IEEE, 2008.

[Béz79] Etienne Bézout. *Théorie générale des équations algébriques; par m. Bézout.* de l'imprimerie de Ph.-D. Pierres, rue S. Jacques, 1779.

[BHLV09] Markus Bläser, Moritz Hardt, Richard J Lipton, and Nisheeth K Vishnoi. Deterministically testing sparse polynomial identities of unbounded degree. *Information Processing Letters*, 109(3):187–192, 2009.

[Bro71] W Steven Brown. On euclid's algorithm and the computation of polynomial greatest common divisors. *Journal of the ACM (JACM)*, 18(4):478–504, 1971.

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

[For15] Michael A Forbes. Deterministic divisibility testing via shifted partial derivatives. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 451–465. IEEE, 2015.

[GKKS13] Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 578–587. IEEE, 2013.

[Gre14] Bruno Grenet. Computing low-degree factors of lacunary polynomials: a Newton-Puiseux approach. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 224–231. ACM, 2014.

[Gre16] Bruno Grenet. Bounded-degree factors of lacunary multivariate polynomials. *Journal of Symbolic Computation*, 75:171–192, 2016.

[KI03] Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 355–364. ACM, 2003.

[KK05] Erich Kaltofen and Pascal Koiran. On the complexity of factoring bivariate supersparse (lacunary) polynomials. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 208–215. ACM, 2005.

[Koi12] Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448:56–65, 2012.

[KS01] Adam R Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 216–223. ACM, 2001.

[KSS14] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and deterministic multivariate polynomial factorization. In *Computational Complexity (CCC), 2014 IEEE 29th Conference on*, pages 169–180. IEEE, 2014.

[KT90] Erich Kaltofen and Barry M Trager. Computing with polynomials given by-black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[MY73] Joel Moses and David YY Yun. The ez gcd algorithm. In *Proceedings of the ACM annual conference*, pages 159–166. ACM, 1973.

[Pla77] David A Plaisted. New np-hard and np-complete polynomial and integer divisibility problems. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 241–253. IEEE, 1977.

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

[SS92] Tateaki Sasaki and Masayuki Suzuki. Three new algorithms for multivariate polynomial gcd. *Journal of symbolic computation*, 13(4):395–411, 1992.

[Str73] Volker Strassen. Vermeidung von divisionen. *Journal für die reine und angewandte Mathematik*, 264:184–202, 1973.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends® in Theoretical Computer Science*, 5(3–4):207–388, 2010.

[Tav15] Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. *Information and Computation*, 240:2–11, 2015.

[VSBR83] Leslie G. Valiant, Sven Skyum, Stuart Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and algebraic computation*, pages 216–226, 1979.