Computing GCDs and Resultants of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions

by

Mahsa Ansari

M.Sc., Beheshti University, 2016
B.Sc., Chamran University, 2013

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

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Winter 2026

Declaration of Committee

Name:	**Mahsa Ansari**
Degree:	**Doctor of Philosophy**
Thesis title:	**Computing GCDs and Resultants of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions**
Committee:	**Chair:** Peter Lisonek Professor, Mathematics
	Michael Monagan Supervisor Professor, Mathematics
	Stephen Choi Committee Member Professor, Mathematics
	Imin Chen Examiner Professor, Mathematics
	Mark Giesbrecht External Examiner Professor Faculty of Mathematics University of Waterloo

Abstract

Let $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ be an algebraic number field. Let f_1 and f_2 be two non-zero polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$. This thesis presents two probabilistic new modular algorithms for computing the monic greatest common divisor (GCD) and the resultant of f_1 and f_2 for $k \geq 1$, namely MGCDNF and MRESNF, respectively. To reduce the cost of computing over multiple extensions $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, we perform the GCD and resultant computations over a simple extension $\mathbb{Q}(\gamma)$ where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, and then map the results back to the original field. This conversion is achieved by constructing an isomorphism between $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and $\mathbb{Q}(\gamma)$ using linear algebra, and it is carried out modulo a prime to avoid expression swell.

We reduce the multivariate setting to the univariate case by evaluating the polynomials at a sequence of randomly chosen points, enabling us to employ the monic Euclidean algorithm. We then use dense interpolation to recover the variables in both the GCD and the resultant. To reconstruct the rational coefficients of the target GCD and resultant, we apply the Chinese remaindering and the rational number reconstruction.

Let p be a prime. To compute the monic GCD and the resultant over the finite field $\mathbb{Z}_p$ we present two sub algorithms, namely PGCDNF and PRESNF. These two algorithms may fail under certain conditions, such as when encountering zero divisors. Moreover, MGCDNF and MRESND may have to restart when using some categories of primes or evaluation points. We provide a classification of these failure cases for both the GCD and resultant computations, along with a probabilistic analysis of their occurrence. Furthermore, we analyze the expected time complexity of our algorithms. All algorithms are implemented in Maple, using a recursive dense representation for multivariate polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ modulo a prime p.

Keywords: GCD; Resultant; Modular Algorithms; Algebraic Number Fields; Primitive Elements

Dedication

This work is dedicated to

- All the paths I did not walk,
  to all the paintings I did not paint,
  to all the songs I did not sing,
  to all the books I did not write,
  and to all the lives I did not live.
  One day, I may yet dare to pursue them.

- To the people of Iran, who have long fought for freedom, to those who have fallen, and to those who still endure, breathing on with hope or despite its absence.
- To my soulmate, Payam Ahmadvand, whose love has illuminated my path and brought color and warmth to my life.

Acknowledgements

This thesis would not have been achievable without the constant guidance and support of my supervisor, Professor Michael Monagan. I am deeply grateful for the time, care, and intellectual energy he devoted to my work. I would also like to thank Dr. Jake Levinson for his thoughtful comments on my Ph.D. proposal, and Professor Stephen Choi for kindly agreeing to join my committee after Dr. Levinson moved to Montreal. My sincere thanks also go to my internal examiner, Professor Imin Chen, and my external examiner, Professor Mark Giesbrecht, for evaluating this thesis and offering invaluable feedback.

I would also like to thank my undergraduate professors, who first inspired my love for mathematics and guided me toward this field. I am especially grateful to Professor Behnaz Koochak Shoushtari, whose passing is still difficult for me to believe. I also warmly thank Professor Alborz Azarang and Professor Omidali Karamzadeh for their teaching, encouragement, and lasting influence on my path.

To undertake this work, I crossed seas and oceans from Iran to Canada. Over time, Canada has become my second homeland. I am profoundly grateful to this kind and generous land for all that it has given me: education, safety, kindness, and the fresh air of freedom to breathe.

Writing and defending this thesis coincided with one of the darkest chapters in the recent history of my motherland, Iran, marked by uprisings and ongoing wars. I offer my deepest gratitude to all those who continue to resist darkness in the imposed silence of digital blackouts in Iran. I honour those stars who fell, and those who still stand and continue the struggle to reclaim our land.

I would like to thank my mother, who always believed in me and filled my childhood with science books that kept my curiosity alive. I also thank her for bringing music into my life by buying me a violin after I began my undergraduate studies in mathematics. I thank my father, who always encouraged me to keep going and to endure, and who gave me all the ice creams along the way. I thank my brother, who taught me to dare to be myself; I earned this Ph.D. and am still alive. I also thank my uncle, who was my first mathematics tutor.

I would also like to thank Fatima Delia for her precious friendship. Her vibrant and colorful spirit has brought light, warmth, and beauty into my world. I am profoundly grateful for her companionship through moments of sadness and happiness alike, and for the quiet grace with which she stood beside me in all seasons of my journey.

I would like to thank my husband, Payam Ahmadvand, for his love and unwavering support. His presence has been one of the quiet strengths beneath this journey. I am deeply grateful for the care with which he stood beside me, for the meals he learned to cook and offered with love, and for the many rides to the university that became, in their own way, small acts of devotion carrying me toward the end of this path.

Contents

List of Tables

List of Algorithms

Chapter 1

Introduction

Let $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ be an algebraic number field. Let $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$, where $n > 1$ and $k \geq 1$. In this thesis, we consider two fundamental problems in computational algebra:

1) the computation of the **monic greatest common divisor (GCD)** of f_1 and f_2, and

2) the computation of the **resultant** of f_1 and f_2 with respect to a variable x_i, where $1 \leq i \leq k$.

The central objective of this work is to design, analyze, and implement algorithms that solve these two problems. Polynomial GCD algorithms can be categorized into two classes: those designed for dense inputs or dense GCD (output), and those tailored for sparse inputs and sparse GCD (output). Thus, it is first essential to clarify the notions of sparsity and density.

Definition 1.0.1. *Let f be a polynomial in variables $x_1, \ldots, x_n$ with a total degree of $d = \deg(f)$ (see Definition 2.1.3). Let $\#f$ denote the number of non-zero terms of f. The maximum number of possible terms in f is $D = \binom{n+d}{d}$. We say f is **sparse** if $\#f \ll D$. Otherwise, f is a **dense** polynomial.*

Definition 1.0.1 is imprecise. To be more precise, we call a polynomial f sparse if $\#f \leq \sqrt{D}$ and dense otherwise.

Example 1.0.1. *Let $f = x_3^4 x_4 + x_4^3 x_5 + x_1^3 x_3 + x_2^2 x_4 + x_5 x_1 + 7x_6^3 + 2 \in \mathbb{Z}[x_1, x_2, x_3, x_4, x_5, x_6]$. The number of possible monomials for a polynomial with $n = 6$ variables of total degree $d = 5$ (see Definition 2.1.3) is $D = \binom{n+d}{d} = 462$ and $\sqrt{462} \approx 21$. Since $\#f = 7 \leq \sqrt{D}$, the polynomial f is sparse.*

Example 1.0.2. *Let $f = -5x_1^2 - 7x_1x_2 - x_1 + 3x_2^2 - 4x_2 + 2 \in \mathbb{Z}[x_1, x_2]$. The number of possible monomials for a polynomial with $n = 2$ variables of total degree $d = 2$ is $D = \binom{n+d}{d} = 6$. Since $\#f > \sqrt{6}$, we call f a dense polynomial.*

Our GCD and resultant algorithms are designed for dense polynomials.

1.1 GCDs and resultants of polynomials: A brief history

Computing the polynomial GCD and resultant are two fundamental problems in Computer Algebra systems and arise as subproblems in many applications. In particular, computing the polynomial GCD and the resultant over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ plays a prominent role in Trager's polynomial factorization algorithm [32].

Computing the GCD of two polynomials has a long history. The Euclidean algorithm, which dates back more than two thousand years, is the most fundamental algorithm for computing the GCD of two univariate polynomials over a field. However, it has a fundamental flaw for problems that arise in $R[x]$, where R is not a finite field. In such cases, the size of the coefficients in intermediate remainders grows significantly. In particular, the Euclidean algorithm is slow when the degree of the GCD is much smaller than the degree of the inputs, with the worst case occurring when the GCD is 1. This inefficiency motivated the development of modular GCD algorithms.

For dense polynomials, in 1971, Collins [10] (for polynomials in $\mathbb{Z}[x]$) and Brown [7] (for polynomials in $\mathbb{Z}[x_1, \ldots, x_k]$) introduced algorithms for computing gcds using homomorphic reductions and the Chinese remainder theorem. Let p be a prime number. Applying homomorphic reduction, Collins and Brown converted the GCD problem over $\mathbb{Z}$ into one over $\mathbb{Z}_p$, a simpler domain in which the coefficients of the intermediate remainders in the Euclidean algorithm do not grow. Brown's algorithm is an efficient method for computing the GCD of dense multivariate polynomials. This algorithm handles multivariate polynomials in $\mathbb{Z}[x_1, \ldots, x_k]$ recursively by treating x_1 as the main variable and evaluating x_k in $\mathbb{Z}_p[x_k][x_1, \ldots, x_{k-1}]$. We review Brown's algorithm in Section 2.2.5. In 1989, Char, Geddes, and Gonnet introduced a heuristic algorithm named GCDHEU [9] for dense polynomials. This algorithm works by evaluating the input polynomials over the integer set $\mathbb{Z}$ instead of $\mathbb{Z}_p$. It then determines a single GCD over the integers, from which the actual GCD can be derived. Although the GCDHEU algorithm can be generalized to handle multivariate polynomials, it is inefficient when handling sparse inputs. Section 2.2.4 provides a brief overview of GCDHEU. For further details, refer to [18, Section 7.7].

For sparse case, in 1973, Moses and Yun [29] introduced the EZ-GCD algorithm, a multivariate GCD algorithm built upon multivariate Hensel lifting [18, Section 6.8]. The Hensel lifting technique, initially proposed by Zassenhaus, is used for factoring univariate polynomials over $\mathbb{Z}$. In honor of Zassenhaus, Moses and Yun called their algorithm the Extended Zassenhaus GCD (EZ-GCD). Their algorithm transforms multivariate polynomials into univariate ones using evaluations and modular homomorphisms. After determining the univariate GCD, it applies Hensel's lemma to incrementally lift the coefficients and variables back to the multivariate form using the Newton iteration process. Further information can be found in [18, Section 7.6]. The EZ-GCD algorithm has several flaws, such as the bad zero problem, which often results in significant growth of intermediate expressions, the leading coefficient problem, and the non-unit common factor problem. In 1980, Wang [36] made a significant contribution to EZ-GCD by developing the Extended EZ-GCD (EEZ-GCD) algorithm. In particular, this algorithm was designed for handling sparse problems. The EEZ-GCD algorithm processes one variable at a time, and Wang's heuristic approach for pre-determining the leading coefficient addresses the leading coefficient issue. Keith Geddes implemented Wang's EEZ-GCD Algorithm in Maple during the 1980s, and it remained Maple's main GCD routine for polynomials with three or more variables until 2005. It is also used by Maxima and SageMath.

Since 2005, Maple has utilized Zippel's [40] algorithm to compute the polynomial GCD over $\mathbb{Z}$. Introduced in 1979, Zippel's GCD algorithm was among the first to offer a probabilistic approach for sparse problems. The implementation of Zippel's algorithm in Maple is detailed in [12]. Additionally, this algorithm is employed in other Computer Algebra systems, such as Fermat, Magma, and Mathematica.

In 1989, Langemyr and McCallum [22] adopted Brown and Collins' algorithm to design a modular GCD algorithm for polynomials in $\mathbb{Q}(\alpha)[x]$. In the same year, Smedley [31], using a different approach, designed a modular GCD algorithm for multivariate polynomials in $\mathbb{Q}(\alpha)[x_1, \ldots, x_k]$. In 1995, Encarnacion [14] improved Langemyr and McCallum's algorithm for $\mathbb{Q}(\alpha)[x]$ and made the algorithm output sensitive, that is, the number of primes used depends on the size of the integers in the GCD and not on bounds based on the input polynomials. Encarnacion achieved this by using rational number reconstruction [25, 38] to recover the rational coefficients of the monic GCD instead of scaling GCD images by a denominator. We review Wang's rational number reconstruction algorithm and Encarnacion's GCD algorithm in Sections 2.2.6 and 4.3, respectively. In 2002, Monagan and Van Hoeij [34] generalized Encarnacion's algorithm to handle polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x]$ for $n \geq 1$. Their algorithm is used by Maple and Pari. Later, in 2009, Lin, Moreno Maza, and Schost [23] used the fast Fourier transform (FFT) to speed up arithmetic in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ modulo a prime in Monagan and Van Hoeij's algorithm.

On the other hand, computing the resultant of two polynomials plays a significant role in various areas of mathematics. Resultants arise as subproblems in solving systems of multivariate polynomials, elimination theory [11], and factorization of polynomials over algebraic fields [32]. Furthermore, they are fundamental algebraic tools for determining whether a system of polynomials has a common root without explicitly solving the system. In 1971, Collins [10] introduced a modular algorithm to compute the resultant of multivariate polynomials over $\mathbb{Z}$. Later, in 1976, Brown and Traub [7] established the fundamental theorem of subresultants, which they used to derive a much simpler formulation of the subresultant polynomial remainder sequence (PRS). In 2005, Monagan [26] adapted Collins' resultant algorithm to develop an output sensitive modular algorithm for determining the monic resultant in $\mathbb{Q}[x]$. This algorithm proves to be more efficient when the bounds required by Collins' algorithm for both the coefficients and the degree of the resultant are not precise.

1.2 Motivating application: Polynomial factorization over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$

Consider two polynomials f_1 and f_2 in $\mathbb{Q}[x_1, \ldots, x_k]$, with $g = \gcd(f_1, f_2)$. We run the following example in Maple.

```
>f1:=(x^2 +3*x*y^2 -15)*(7*x+5*y +7) ;
>f2:=(x^2 +3*x*y^2 -15)*(x+5*y);
>simplify(f_1/f_2);
                        f1:= (3*x*y^2 + x^2 - 15)*(7*x + 5*y + 7)
                        f2:=(3*x*y^2 + x^2 - 15)*(x + 5*y)
                        (7*x + 5*y + 7)/(x + 5*y)
```

In Maple, the command *simplify* initially computes g, then provides the result as $\frac{f_1/g}{f_2/g}$. When handling fractions of polynomials, Maple performs various operations, including addition, multiplication, division, and GCD computations. Among these operations, computing the GCD is the most expensive operation, often becoming the bottleneck. Therefore, speeding up polynomial GCD algorithms is crucial for increasing Maple's overall speed [27]. As another motivating application, we can consider the problem of factoring $f \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x]$ by using Trager's algorithm [32]. In 1882, Kronecker [21] suggested the use of norms for factorization. A similar idea was later presented by Van der Waerden in [33]. In 1976, Trager improved upon this idea and presented an algorithm for factoring polynomials

over an algebraic number field $\mathbb{Q}(\alpha)$ with one field extension. In what follows, we provide a brief explanation of how the computation of the GCD and the resultant influences Trager's algorithm.

Trager's algorithm

Currently, Maple employs Trager's algorithm for factoring polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$. Nevertheless, factorization over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ continues to pose a significant challenge in Maple. This section provides a summary of Trager's algorithm. For more detailed information about this algorithm, see [18, Section 8.8]. Consider a square-free polynomial $f \in \mathbb{Q}(\alpha)[x]$, that is $\gcd(f, \frac{df}{dx}) = 1$. The fundamental idea behind Trager's algorithm is to transform the polynomial $f \in \mathbb{Q}(\alpha)[x]$ from the algebraic number field into a square-free polynomial $N \in \mathbb{Q}[x]$ in such a way that each factor of f can be derived from a factor of N by performing a GCD computation over $\mathbb{Q}(\alpha)$. To do so, we need to find some $s \in \mathbb{Z}_{\geq 0}$ s.t. $N = \text{Norm}(f(x - s\alpha))$ is square-free. Let f be a monic polynomial. Let $\mathbb{Q}(\alpha) \cong \mathbb{Q}[z]/\langle M(z)\rangle$. where $M(z)$ is the minimal polynomial of α over $\mathbb{Q}$. The diagram below demonstrates the framework of Trager's algorithm.

$$\begin{array}{ccc} \text{monic}(f) \in \mathbb{Q}[z]/\langle M(z)\rangle[x] & \longrightarrow & f = \prod_{i=1}^{k} g_i \in \mathbb{Q}[z]/\langle M(z)\rangle[x] \\ \Big\downarrow \text{Norm} & & \Big\uparrow g_i = \text{monic}(\gcd(f(x - s\alpha), N_i)) \\ N = \text{Norm}(f(x - sz)) \in \mathbb{Q}[x] & \xrightarrow{Factor(N) \text{ over } \mathbb{Q}} & N = N_1 \cdots N_k \in \mathbb{Q}[x] \end{array}$$

The two main steps of Trager's algorithm are computing $\text{Norm}(f(x - sz)) \in \mathbb{Q}[x]$ using the $\text{res}(f(x - sz), M(z), z)$ and computing the monic GCDs $\gcd(N, f_i) \in \mathbb{Q}(\alpha)[x]$. Therefore, improving the speed of the resultant and GCD computation over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ directly enhances the efficiency of the factorization process. Specifically, in Trager's algorithm, $\text{Norm}(f(x - sz))$ can be computed using our modular resultant algorithm, and $\gcd(f, N_i)$ can be computed for $1 \leq i \leq k$ using our modular GCD algorithm.

1.3 Contributions

Building upon prior work, we designed modular algorithms to compute the monic GCD and the resultant of two polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$, referred to as MGCDNF and MRESNF, respectively [2, 3]. To improve efficiency, our algorithms convert the input polynomials f_1 and f_2 to their corresponding polynomials over $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. This approach reduces computational costs and accelerates the process by reducing the coefficients modulo the minimal polynomial of γ, instead of reducing modulo the n minimal polynomials of $\alpha_1, \ldots, \alpha_n$. This conversion is performed modulo a prime to prevent coefficient growth during intermediate computations. Next, our algorithms employ evaluation and dense interpolation to reduce the problem to computing the monic GCD and the resultant of two univariate polynomials, for which we apply the monic Euclidean algorithm [34]. Finally, our modular algorithms apply Chinese remaindering and rational number reconstruction [25] to recover the rational coefficients of the monic GCD and resultant

from their modular images. Moreover, employing the monic Euclidean algorithm, we present a new formula for computing the resultant of univariate polynomials.

Since our MGCDNF and MRESNF algorithms map the input polynomials $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to their images over $\mathbb{Q}(\gamma)$ modulo primes, the monic GCD and the resultant are computed in a finite quotient ring modulo a prime p, rather than in a field. Consequently, the algorithms may encounter zero divisors when computing modulo p. If MGCDNF or MRESNF encounters a zero divisor while working modulo p, the computation for that prime is aborted, a new prime is chosen, and the computation is repeated for the new prime. To date, there has been no analysis of the occurrence of zero divisors beyond the observation that only finitely many primes can cause them. In this thesis, we determine the exact number of zero divisors and analyze their occurrence in our algorithms.

Published work

This thesis is based on the following published papers co-authored with my Ph.D. supervisor, Professor Michael Monagan:

1. Mahsa Ansari and Michael Monagan. Computing GCDs of multivariate polynomials over algebraic number fields presented with multiple extensions. In *Computer Algebra in Scientific Computing*, volume 14139 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2023

2. Mahsa Ansari and Michael Monagan. A modular algorithm to compute the resultant of multivariate polynomials over algebraic number fields presented with multiple extensions. In *Computer Algebra in Scientific Computing*, volume 14938 of *Lecture Notes in Computer Science*, pages 27–46. Springer, 2024

3. Mahsa Ansari and Michael Monagan. A failure probability analysis of a modular algorithm to compute the monic GCD of multivariate polynomials over algebraic number fields. In *Computer Algebra in Scientific Computing*, volume 16235 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2025

We also presented our work on computing the monic GCD at the **Conference on Intelligent Computer Mathematics (CICM), 2023,** held at the **University of Cambridge**, UK. In addition, our work on both computing the monic GCD and the resultant was presented at the **Maple 2024 Conference**.

1.4 Thesis outline

This thesis is organized as follows. Chapter 2 provides a general introduction and motivation for the problem, along with preliminary concepts and notations. Chapter 3 describes essential tools for computing over algebraic number fields, including the new algorithm LAminpoly and the primitive isomorphism used to map polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to their corresponding polynomials over $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. In addition, this chapter counts the number of zero divisors that may arise in our modular algorithms. In Chapter 4, we introduce a new modular GCD algorithm, MGCDNF, to compute the monic GCD of polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ with high probability. This chapter also presents a worst-case complexity analysis of MGCDNF and

benchmark results. Chapter 5 presents a new modular probabilistic resultant algorithm, MRESNF, for polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$, along with benchmarks and a worst-case complexity analysis. The detailed failure probability analysis of both MGCDNF and MRESNF is presented in Chapter 6. In Chapter 7, using the recursive dense data structure (recden package in Maple) to represent polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, some implementation details of MGCDNF and MRESNF are presented. Lastly, Chapter 8 concludes the thesis with a summary of results and a discussion of open problems for future research.

Chapter 2

Background tools, ideas, and algorithms

2.1 Preliminary concepts and notations

In this thesis, R denotes a commutative ring with identity $1 \neq 0$, and $X = x_1, \ldots, x_k$ unless stated otherwise. The set of integers is denoted by $\mathbb{Z}$, and the field of rational numbers by $\mathbb{Q}$. Let $p \in \mathbb{Z}$ be a prime number. For any $a \in \mathbb{Z}$, we denote by $\bar{a}$ the equivalence class of all integers congruent to a mod p. The set $\mathbb{Z}_p = \{\overline{0}, \overline{1}, \ldots, \overline{p-1}\}$ forms a finite field of characteristic p.

Definition 2.1.1. *A **monomial** in $X = x_1, \ldots, x_k$ is a product of the form $\prod_{i=1}^{k} {x_i}^{e_i}$, where all the exponents are non-negative integers. Moreover, the **total degree** of $\prod_{i=1}^{k} {x_i}^{e_i}$, denoted by $\deg(\prod_{i=1}^{k} x_i^{e_i})$, is the sum $e_1 + \cdots + e_k$.*

If $e_1 = \cdots = e_k = 0$, then ${x_1}^{e_1} \ldots {x_k}^{e_k} = 1$. Let $e = (e_1, \ldots, e_k) \in \mathbb{Z}_{\geq 0}^k$. To simplify the notation for monomials, we set $x^e = \prod_{i=1}^{k} {x_i}^{e_i}$.

Definition 2.1.2. *A **polynomial** f in X with coefficients in R is a finite linear combination of monomials with coefficients in R. We present a polynomial in the form*

$$f = \sum_{e \in \mathbb{Z}_{\geq 0}^k} a_e x^e,$$

*where $e = (e_1, \ldots, e_k) \in \mathbb{Z}_{\geq 0}^k$ and $a_e \in R$. If $k > 1$, we call f a **multivariate polynomial**.*

We use the following terminology when dealing with polynomials.

Definition 2.1.3. *Let $f = \sum_{e \in \mathbb{Z}_{\geq 0}^k} a_e x^e$ be a polynomial over R.*

*(i) We call a_e the **coefficient** of the monomial x^e.*

*(ii) We call $a_e x^e$ a **term** of f if $a_e \neq 0$.*

*(iii) We call e the **exponent vector** of the monomial x^e.*

*(iv) If $f \neq 0$, the **total degree** of f, $\deg(f)$, is the maximum of the total degree of the monomials.*

(v) If $f = 0$, we define $\deg(0) = -\infty$.

(vi) The notation $R[x_1, \ldots, x_k]$ denotes the set of all multivariate polynomials in X over R.

Since the rings $R[x_1, \ldots, x_k]$ and $R[x_1, \ldots, x_{k-1}][x_k]$ are *isomorphic*, we can view the multivariate polynomial domain $R[x_1, \ldots, x_k]$ as a univariate polynomial domain $R[x_1, \ldots, x_{k-1}][x_k]$ where x_k is chosen as the main variable and the coefficients are in $R[x_1, \ldots, x_{k-1}]$.

In analyzing the division algorithm for multivariate polynomials in $R[x_1, \ldots, x_k]$, a monomial order is essential (see [11, Chapter 2]). The following definitions follow [11, Chapter 2, Section 2].

Definition 2.1.4. *Let M be a set of monomials in $R[x_1, \ldots, x_k]$. An **order relation**, denoted by $<$, on M is a total ordering if $\forall x^a, x^b, x^c$ the following conditions hold:*

(i) Either $x^a < x^b$, $x^b < x^a$, or $x^a = x^b$.

(ii) If $x^b < x^a$ and $x^c < x^b$, then $x^c < x^a$.

Definition 2.1.5. *A **monomial ordering** on $R[x_1, \ldots, x_k]$ is a relation $>$ on $\mathbb{Z}^k_{\geq 0}$ that satisfies the following conditions:*

(i) $>$ is a total ordering,

(ii) $\forall a, b, c \in \mathbb{Z}^k_{\geq 0}$, we have $a > b \Longrightarrow c + a > c + b$

(iii) Every non-empty subset $S \subset \mathbb{Z}^k_{\geq 0}$ has a least element under $>$.

Definition 2.1.6. *Let $e = (e_1, \ldots, e_k)$ and $d = (d_1, \ldots, d_k)$ be two exponent vectors in $\mathbb{Z}^k_{\geq 0}$.*

*(i) **Lexicographic Order**: We say $e >_{lex} d$ if the left-most non-zero entry in the vector difference $e - d \in \mathbb{Z}^k$ is positive. We write $x^e >_{lex} x^d$ if $e >_{lex} d$.*

*(ii) **Graded Lexicographic Order**: We say $e >_{grlex} d$ if $\deg(x^e) > \deg(x^d)$ or $\deg(x^e) = \deg(x^d)$ and $e >_{lex} d$. We write $x^e >_{grlex} x^d$ if $e >_{grlex} d$.*

Example 2.1.1. *Given the above definition, if $x_1 > x_2 > x_3$, then*

$$x_1^3 x_2 x_3 >_{lex} x_1^2 x_2^3 x_3 \quad \text{and } x_1^2 x_2^3 x_3 >_{grlex} x_1^3 x_2 x_3.$$

Knowing the ordering, we identify a polynomial's greatest monomial and coefficient. We use the following terminology.

Definition 2.1.7. *Let $f = \sum_{e \in \mathbb{Z}^k_{\geq 0}} a_e x^e$ be a non-zero polynomial in $R[x_1, \ldots, x_k]$, and $>$ be a monomial order.*

*(i) The **multidegree** of f is*

$$multideg(f) = max(e \in \mathbb{Z}^k_{\geq 0} : a_e \neq 0)$$

(the maximum is taken with respect to $>$).

*(ii) The **leading coefficient** of f, denoted by $\mathrm{lc}(f)$, is*

$$\mathrm{lc}(f) = a_{multideg(f)}$$

(iii) The ***leading monomial*** *of f, denoted by $\mathrm{lm}(f)$, is*

$$\mathrm{lm}(f) = x^{multideg(f)}$$

(iv) The ***leading term*** *of f, denoted by $\mathrm{lt}(f)$, is*

$$\mathrm{lt}(f) = \mathrm{lm}(f) \cdot \mathrm{lc}(f)$$

Example 2.1.2. *Let $f = -9x^3 + 7x^2z^2 + 6xy^2z \in \mathbb{Z}[x, y, z]$ and let $>_{lex}$ be the lexicographic order with $x > y > z$. Here, $\deg(f) = 4$, $multideg(f) = (3, 0, 0)$, $\mathrm{lc}(f) = -9$, $\mathrm{lm}(f) = x^3$, and $\mathrm{lt}(f) = -9x^3$.*

Remark 2.1. *We define $\mathrm{lc}(0) = 0$, $\mathrm{lm}(0) = 1$, and $\mathrm{lt}(0) = 0$.*

We state the following lemma and corollary without proof.

Lemma 2.1.1. *[11, Lemma 8, Chapter 2] Let $f_1, f_2 \in R[x_1, \ldots, x_k]$ be two non-zero polynomials.*

(i) If $f_1 \cdot f_2 \neq 0$, then $multideg(f_1 \cdot f_2) \leq multideg(f_1) + multideg(f_2)$.

(ii) If $f_1 + f_2 \neq 0$, then $multideg(f_1 + f_2) \leq \max\{multideg(f_1), multideg(f_2)\}$.

Corollary 2.1. *Let $f_1, f_2 \in R[x_1, \ldots, x_k]$ be two non-zero polynomials. Then*

(i) $\mathrm{lm}(f_1 \cdot f_2) \leq \mathrm{lm}(f_1) \cdot \mathrm{lm}(f_2)$.

(ii) $\mathrm{lt}(f_1 \cdot f_2) \leq \mathrm{lt}(f_1) \cdot \mathrm{lt}(f_2)$.

Remark 2.2. *If R is an integral domain, or if $\mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2) \neq 0$, then equality holds in Lemma 2.1.1(i) and Corollary 2.1.*

Definition 2.1.8. *Let $f \in R[x_1, \ldots, x_n]$. For $1 \leq i \leq n$, we can represent f as a polynomial in x_i with coefficients in $R[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$, so that*

$$f = \sum_{j=0}^{d} a_j x_i^j, \qquad a_j \in R[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n], \quad a_d \neq 0.$$

We define $\deg(f, x_i) = d$ and $\mathrm{lc}(f, x_i) = a_d$.

Notation 2.1. *If the variable is not specified, then $\deg(f)$ denotes the total degree of f (Definition 2.1.3, part (iv)). Likewise, if the variable is not specified, then $\mathrm{lc}(f)$ denotes the leading coefficient defined in part (ii) of Definition 2.1.7 (w.r.t. the fixed monomial order stated there).*

Example 2.1.3. *Let $f \in \mathbb{Q}[x_1, x_2, x_3]$ s.t.*

$$f = 3x_1^2x_2 + x_1^2x_3 + 5x_1x_2^3 - 7x_2^2x_3 + 4x_3^5 \ \in \mathbb{Q}[x_1, x_2, x_3].$$

Representing f as a univariate polynomial w.r.t. x_1, we have

$$f = (3x_2 + x_3)x_1^2 + (5x_2^3)x_1 + (-7x_2^2x_3 + 4x_3^5) \in \mathbb{Q}[x_2, x_3][x_1].$$

Hence, $\deg(f, x_1) = 2$, $\mathrm{lc}(f, x_1) = 3x_2 + x_3 \in \mathbb{Q}[x_2, x_3]$, and $\deg(f) = 5$. Consider the lexicographic order with $x_3 > x_1 > x_2$, then $\mathrm{lc}(f) = 4$.

2.2 GCD Computation of polynomials over $\mathbb{Z}$

In this section, we discuss some algorithms for computing the GCD g of two polynomials f_1 and f_2 in $\mathbb{Z}[x_1, \ldots, x_k]$. These include the Euclidean algorithm and the monic Euclidean algorithm over $\mathbb{Q}[x]$, as well as the GCDHEU algorithm and Brown's algorithm over $\mathbb{Z}[x]$. We also review rational number reconstruction and Newton's interpolation algorithm, which are two essential tools for rebuilding the target GCD from its images. Later, in Section 4.3, we will discuss polynomial GCD algorithms over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$.

In many fields, including coding theory and cryptography, calculations modulo an integer play a crucial role. As we will demonstrate in this thesis, employing modular homomorphisms enhances the efficiency of computing greatest common divisors, resultants, and consequently, polynomial factorizations [32]. The fundamental tool for a modular approach is **division with remainder**. Given the integers a and b (where $b \neq 0$), we aim to find two integers, q (the quotient) and r (the remainder), s.t.

$$a = qb + r, \quad 0 \leq r < |b|.$$

Similarly, for polynomials f_1 and f_2 in $R[x]$, we seek polynomials q and r in $R[x]$ that satisfy

$$f_1 = f_2 q + r, \quad \deg(r) < \deg(f_2).$$

Example 2.2.1. *Let $f_1 = x^3 + 2x + 1$ and $f_2 = 3x^2 + 5$ be two polynomials in $\mathbb{Q}[x]$. Then we can divide f_1 by f_2 with remainder $r = \frac{1}{3}x + 1 \in \mathbb{Q}[x]$ and quotient $q = \frac{1}{3}x \in \mathbb{Q}[x]$.*

However, such q and r do not always exist. For instance, if we consider the polynomials f_1 and f_2 from Example 2.2.1 as polynomials in $\mathbb{Z}[x]$, the division of f_1 by f_2 with remainder in $\mathbb{Z}[x]$ is not possible since $\mathrm{lc}(f_2) = 3$ is not invertible in $\mathbb{Z}$. This problem can be addressed using *pseudo-division* (refer to Section 2.2.3). For simplicity, at this stage, we assume the leading coefficient $\mathrm{lc}(f_2)$ is a unit in R, allowing us to present Theorem 2.2.1.

Theorem 2.2.1. *[35, Algorithm 2.5 Polynomial division with remainder] Let f_1 and f_2 be two polynomials in $R[x]$ with $f_2 \neq 0$. If $lc(f_2)$ is a unit, there exist unique polynomials q (the quotient) and r (the remainder) in $R[x]$ s.t. $f_1 = qf_2 + r$ with $r = 0$ or $\deg(r) < \deg(f_2)$.*

Definition 2.2.1. *Let $f_1, f_2 \in U[X]$, where U is a UFD. A greatest common divisor (GCD) of f_1 and f_2, denoted by $g = \gcd(f_1, f_2)$, is a polynomial in $U[X]$ s.t.:*

(i) $g \mid f_1$ and $g \mid f_2$, and

(ii) if $h \in U[X]$ s.t. $h \mid f_1$ and $h \mid f_2$, then $h \mid g$.

Definition 2.2.2. *Let $f_1, f_2 \in U[X]$, where U is a UFD and $g = \gcd(f_1, f_2)$. Let h_1 and h_2 be two polynomials in $U[x]$ s.t. $f_1 = g \cdot h_1$ and $f_2 = g \cdot h_2$. Then we call h_1 and h_2 the cofactors of f_1 and f_2, respectively.*

Example 2.2.2. *Let $f_1 = (x-1)(x^2y - y + 1)$ and $f_2 = (x-1)(x^5y - 2y + 6) \in \mathbb{Q}[x, y]$. Then $g = \gcd(f_1, f_2) = x - 1$. Also, $h_1 = x^2y - y + 1$ and $h_2 = x^5y - 2y + 6$ are the cofactors of f_1 and f_2, respectively. Notice that $\gcd(h_1, h_2) = 1$.*

In this example, $1-x = -(x-1)$ and $2x-2 = 2(x-1)$ are also GCDs of f_1 and f_2. More generally, if g is a GCD of two polynomials over U, then for any unit $c \in U$, the polynomial $c \cdot g$ is also a GCD of these polynomials. Thus, the GCD of two polynomials is unique up to multiplication by a unit. We will return to this point in more detail in Section 2.2.2.

Definition 2.2.3. *Let U be a unique factorization domain (UFD) and let $f \in U[x_1, \ldots, x_k]$. Consider a set X that consists of some or all of the variables $x_1, \ldots, x_k$.*

*(i) The **content** of f w.r.t X, denoted by $\operatorname{cont}(f, X)$, is the GCD of the coefficients of f, when f is considered as a polynomial in the variables X, with any remaining variables being part of the coefficient ring. If $X = \{x_1, \ldots, x_k\}$, then we write $\operatorname{cont}(f, X) = \operatorname{cont}(f)$.*

*(ii) We define the **primitive part** of f, w.r.t. X, denoted by $\operatorname{pp}(f, X)$, as the quotient of the polynomial by its content. More precisely, $\operatorname{pp}(f, X) = \frac{f}{\operatorname{cont}(f,X)}$. If $X = \{x_1, \ldots, x_k\}$, then we write $\operatorname{pp}(f, X) = \operatorname{pp}(f)$.*

(iii) The polynomial f is called primitive if $\operatorname{cont}(f) = 1$.

Example 2.2.3. *Let $f = 4x^2yz + 24xy^2z + 16z \in \mathbb{Z}[x, y, z]$. Then*

$$\operatorname{cont}(f) = \gcd(4, 24, 16) = 4 \text{ and}$$
$$\operatorname{pp}(f) = x^2yz + 6xy^2z + 4z.$$

Now, consider f as a polynomial in $\mathbb{Z}[z][x, y]$, i.e., $f = (4z)x^2y + (24z)xy^2 + 16z$. Then,

$$\operatorname{cont}(f, \{x, y\}) = \gcd(4z, 24z, 16z) = 4z \text{ and}$$
$$\operatorname{pp}(f, \{x, y\}) = x^2y + 6xy^2 + 4.$$

2.2.1 The Euclidean algorithm

The Euclidean algorithm, first described in Euclid's Elements (Book VII) [15], is among the oldest algorithms still in common use. It is presented as Algorithm 1 and serves as the fundamental algorithm for computing univariate polynomial gcds over a field. The Euclidean algorithm is based on the following lemma.

Lemma 2.2.2. *Let $f_1, f_2 \in \mathbb{F}[x]$ with $f_1, f_2 \neq 0$, where $\mathbb{F}$ is a field. Let q and r be the quotient and remainder when dividing f_1 by f_2. Then $\gcd(f_1, f_2) = \gcd(f_2, r)$.*

Example 2.2.4. *Let $f_1, f_2 \in \mathbb{Q}[x]$ s.t.*

$$f_1 = 6x^5 + 6x^4 - 2x^3 - x^2 + 2x + 1,$$
$$f_2 = 5x^4 + 5x^3 - 3x^2 + 2x + 5.$$

Let $r_1 = f_1$ and $r_2 = f_2$. The Euclidean algorithm proceeds as illustrated in Table 2.1. Given that $r_6 = 0$, we conclude that $\gcd(f_1, f_2) = r_5 \in \mathbb{Q}[x]$.

Remark 2.3. *The Example 2.2.4 illustrates two drawbacks of the Euclidean algorithm:*

Algorithm 1: Euclidean Algorithm (EA)

Input: $f_1, f_2 \in \mathbb{F}[x]$ s.t. $0 \leq \deg(f_2) \leq \deg(f_1)$ and $\mathbb{F}$ is a field.
Output: $\gcd(f_1, f_2)$.

```
r_1, r_2 = f_1, f_2
i = 1
while r_i ≠ 0 do
    Set r_{i+1} to be the remainder of r_{i-1} divided by r_i
    Set i = i + 1
l = i - 1
return(r_l)
```

Table 2.1: Euclidean algorithm

Dividend	Divisor	Quotient	Remainder
r_1	r_2	$q_3 = \frac{6}{5}x$	$r_3 = \frac{8}{5}x^3 - \frac{17}{5}x^2 - 4x + 1$
r_2	r_3	$q_4 = \frac{25}{8}x + \frac{625}{64}$	$r_4 = \frac{2733}{64}x^2 + \frac{607}{16}x - \frac{305}{64}$
r_3	r_4	$q_5 = \frac{512}{13665}x - \frac{843328}{7469289}$	$r_5 = \frac{3450304}{7469289}x + \frac{3450304}{7469289}$
r_4	r_5	$q_6 = \frac{20413566837}{220819456}x - \frac{2278133145}{220819456}$	$r_6 = 0$

1. *The size of coefficients in intermediate remainders grows exponentially. This makes the Euclidean algorithm inefficient in practice.*

2. *Although the input polynomials are in $\mathbb{Z}[x]$, the remainders lie in $\mathbb{Q}[x]$.*

To address the first problem in our modular algorithms, we run the Euclidean algorithm over $\mathbb{F} = \mathbb{Z}_p$. To resolve the second problem, we employ pseudo-division, as described in Section 2.2.3.

Definition 2.2.4. *Let $f_1(x)$ and $f_2(x)$ be two polynomials in $U[x]$ with $\deg(f_1) \geq \deg(f_2) \geq 0$, where U is a UFD. A **Natural Euclidean Polynomial Remainder Sequence (p.r.s.)** generated by the polynomials f_1 and f_2 is a sequence $r_1(x), r_2(x), \ldots, r_l(x)$ in $U[x]$ obtained from the execution of the Euclidean Algorithm s.t.:*

(i) $r_1(x) = f_1(x)$, $r_2(x) = f_2(x)$,

(ii) $r_{i+1} = r_{i-1} - r_i q_i$ *with* $\deg(r_{i+1}) < \deg(r_i)$ *for* $2 \leq i \leq l-1$, *and*

(iii) the remainder of r_{l-1} divided by r_l is zero, i.e., $r_{l+1} = 0$ in the Euclidean algorithm.

Example 2.2.5. *In Example 2.2.4, the sequence $r_1, r_2, \ldots, r_5$ is the p.r.s. generated by f_1 and f_2.*

2.2.2 The monic Euclidean algorithm

Let $f_1, f_2 \in R[x]$ with $g = \gcd(f_1, f_2) \in R[x]$. If $c \in R$ is a unit, then $h = c \cdot g$ is also a GCD of f_1 and f_2. To make the GCD unique, it must be normalized (refer to Theorems 2.2.4 and 2.2.5). This normalization is based on the concept of monic polynomials, which is explained below.

Definition 2.2.5. *Let $f \in R[x]$. If $f = 0$, we define* $\text{monic}(f) = 0$. *Otherwise, we define*

$$\text{monic}(f) = \text{lc}(f)^{-1} f.$$

If $\text{lc}(f)$ *is not a unit over* R*, then* $\text{monic}(f) =$ "$failed$"*. A polynomial* f *is called* ***monic*** *if* $f = \text{monic}(f)$*.*

Example 2.2.6. *For* $f = 2x + 3 \in \mathbb{Q}[x]$*, we have*

$$\text{monic}(f) = \text{lc}(f)^{-1} \cdot f = x + \frac{3}{2}$$

If $f = 2x + 3 \in \mathbb{Z}[x]$*, then* $\text{monic}(f) =$ "$failed$" *because* $\text{lc}(f) = 2$ *is not invertible in* $\mathbb{Z}$*.*

In our modular algorithms, MGCDNF (Algorithm algorithm 8) and MRESNF (Algorithm algorithm 11), the ring R is an algebraic number field modulo a prime. In this setting, every element of R is either zero, a unit, or a zero divisor. Therefore, if $\text{monic}(f) =$ "$failed$", it indicates that $\text{lc}(f)$ is a zero divisor.

Given $f_1, f_2 \in R[x]$, the **monic GCD** of f_1 and f_2 is a monic polynomial $g \in R[x]$, which is a GCD of f_1 and f_2. To compute the monic GCD, we employ the monic Euclidean algorithm (Algorithm 2) [34]. This algorithm takes two polynomials $f_1, f_2 \in R[x]$ and returns either $\text{monic}(\gcd(f_1, f_2))$ or FAIL if the leading coefficient of any remainder is not a unit. In this thesis, Algorithm 2 is used to determine both the GCD and the resultant of two univariate polynomials.

Algorithm 2: Monic Euclidean Algorithm (MEA)

Input: $f_1, f_2 \in R[x]$ s.t. $\deg(f_1) \geq \deg(f_2) \geq 0$ and R is a commutative ring with identity $1 \neq 0$.
Output: Either the monic $\gcd(f_1, f_2)$ or FAIL.

```
r_1, r_2 = f_1, f_2
M_1, i = r_1, 2
while r_i ≠ 0 do
    M_i = monic(r_i)
    if M_i = failed then return(FAIL)
    Set r_{i+1} to be the remainder of M_{i-1} divided by M_i
    Set i = i + 1
l = i − 1
return(M_l)
```

Example 2.2.7. *Let* $f_1, f_2 \in \mathbb{Q}[x]$ *be the polynomials given in Example 2.2.4. Let* $M_1 = f_1$ *and* $M_2 = \text{monic}(f_2)$*. The monic Euclidean algorithm executes as follows.*

Table 2.2: Monic Euclidean algorithm

Dividend	**Divisor**	**Quotient**	**Remainder**
$f_1 = 6x^5 + 6x^4 - 2x^3 - x^2 + 2x + 1$	$f_2 = 5x^4 + 5x^3 - 3x^2 + 2x + 5$	$(-)$	$(-)$
$M_1 = f_1$	$M_2 = x^4 + x^3 - \frac{3}{5}x^2 + \frac{2}{5}x + 1$	$q_3 = 6x$	$r_3 = \frac{8}{5}x^3 - \frac{17}{5}x^2 - 4x + 1$
M_2	$M_3 = x^3 - \frac{17}{8}x^2 - \frac{5}{2}x + \frac{5}{8}$	$q_4 = x + \frac{25}{8}$	$r_4 = \frac{2733}{320}x^2 + \frac{607}{80}x - \frac{61}{64}$
M_3	$M_4 = x^2 + \frac{2428}{2733}x - \frac{305}{2733}$	$q_5 = x - \frac{65885}{21864}$	$r_5 = \frac{2156440}{7469289}x + \frac{2156440}{7469289}$
M_4	$M_5 = x + 1$	$q_6 = x - \frac{305}{2733}$	$r_6 = 0$

Since the final remainder is $r_6 = 0$*, the algorithm terminates after* 6 *iterations, and the monic GCD is* $M_5 = x + 1$*.*

Notation 2.2. *In this thesis,*

- ***EA*** *stands for the Euclidean algorithm.*
- ***MEA*** *stands for the monic Euclidean algorithm.*

Lemma 2.2.3. *Let $f_1, f_2 \in R[x]$ with $n = \deg(f_1)$ and $m = \deg(f_2)$ s.t. $n \geq m$. The MEA does $\mathcal{O}(mn)$ arithmetic operations in R.*

In Theorem 2.2.4, we demonstrate that the MEA terminates, and in cases where it does not result in a FAIL, it delivers the monic GCD of the provided inputs. Following this, in Theorem 2.2.5, we prove that if a monic GCD exists, it is unique.

Theorem 2.2.4. *Let $f_1, f_2 \in R[x]$ with $\deg(f_1) \geq \deg(f_2) \geq 0$. Algorithm 2 terminates for the input f_1 and f_2. Furthermore, if it does not return FAIL, it outputs a monic GCD of f_1 and f_2.*

Proof. First, we demonstrate that Algorithm 2 terminates. If any of the remainders r_i produced in Line 6 have a leading coefficient that is not a unit, the algorithm will terminate and return FAIL. If this is not the case, then from Lines 1 and 2, we have $r_1 = M_1 = f_1$ and $r_2 = f_2$, implying that

$$\deg(M_1) = \deg(r_1) \geq \deg(r_2) = \deg(M_2) \geq 0.$$

Additionally, according to the division algorithm, for each $i \geq 2$, we have

$$M_{i-1} = M_i q_{i+1} + r_{i+1}$$

with $0 \leq \deg(M_{i+1}) < \deg(M_i)$. As a result, we observe a strictly decreasing sequence of non-negative integers

$$\deg(M_1) \geq \deg(M_2) > \deg(M_3) > \ldots \geq 0.$$

According to the well-ordering principle, this sequence must terminate, which guarantees that the monic Euclidean algorithm terminates. We now demonstrate that if Algorithm 2 does not return FAIL, the output it generates, M_l, is the monic$(\gcd(f_1, f_2))$. The steps of Algorithm 2 are outlined below.

$$\begin{aligned}
M_1 &= r_1 \\
M_2 &= \text{monic}(r_2) && M_1 = M_2 q_3 + r_3 \Rightarrow r_3 = M_1 - M_2 q_3 && (1)\\
M_3 &= \text{monic}(r_3) && M_2 = M_3 q_4 + r_4 \Rightarrow r_4 = M_2 - M_3 q_4 && (2)\\
&\vdots \\
M_{l-1} &= \text{monic}(r_{l-1}) && M_{l-2} = M_{l-1} q_l + r_l \Rightarrow r_l = M_{l-2} - M_{l-1} q_l && (l-2)\\
M_l &= \text{monic}(r_l) && M_{l-1} = M_l q_{l+1} + r_{l+1} \Rightarrow M_{l-1} = M_l q_{l+1} && (l-1)
\end{aligned}$$

By construction, the output M_l is monic. From the iteration $(l-1)$, it follows that $M_l \mid M_{l-1}$. Furthermore, given that $M_l = \text{monic}(r_l)$, we also have $M_l \mid r_l$. From the $(l-2)$th iteration, since $M_l \mid r_l$ and $M_l \mid M_{l-1}$, we conclude that $M_l \mid M_{l-2}$. Continuing this reasoning, we have $M_l \mid M_2$; hence $M_l \mid r_2 = f_2$. From the iteration (1), $M_l \mid M_1 = f_1$. Consequently, M_l is a monic common divisor of f_1 and f_2.

To prove that M_l is the greatest monic common divisor of f_1 and f_2, let $d \in R[x]$ s.t. $d \mid f_1$ and $d \mid f_2$. If $d = 1$, then trivially $d \mid M_l$ and we are done. Assume $d \neq 1$. Since $d \mid f_2 = r_2$, we have $d \mid M_2 = \text{monic}(f_2)$. From $M_1 = M_2 q_3 + r_3$, since $d \mid M_1$ and $d \mid M_2$, it follows that $d \mid r_3$. Thus, $d \mid M_3 = \text{monic}(r_3)$. Iteration (2) along with the fact that $d \mid M_2$ and $d \mid M_3$, gives $d \mid r_4$. Repeating this process, we conclude that $d \mid M_l$. Consequently, $M_l = \text{monic}(\gcd(f_1, f_2))$.

□

Theorem 2.2.5. *Let R be a commutative ring with $1 \neq 0$ and let f_1 and f_2 be two polynomials in $R[x]$. If a monic GCD of f_1 and f_2 exists, then it is unique.*

Proof. Let g and d be monic GCDs of f_1 and f_2. Then $g \mid d$ and $d \mid g$, so there exist non-zero polynomials $u, v \in R[x]$ s.t.

$$d = u \cdot g \quad \text{and} \quad g = v \cdot d.$$

We show that $u = v = 1$. Since g and d are monic, we have

$$\text{lc}(d) = \text{lc}(u) \cdot \text{lc}(g) \Longrightarrow 1 = \text{lc}(u) \cdot 1 \Longrightarrow \text{lc}(u) = 1, \tag{2.1}$$

$$\text{lc}(g) = \text{lc}(v) \cdot \text{lc}(d) \Longrightarrow 1 = \text{lc}(v) \cdot 1 \Longrightarrow \text{lc}(v) = 1. \tag{2.2}$$

Moreover, since $\text{lc}(u) \cdot \text{lc}(g) = 1 \neq 0$ and $\text{lc}(v) \cdot \text{lc}(d) = 1 \neq 0$, it follows that

$$\deg(d) = \deg(u) + \deg(g), \text{ and} \tag{2.3}$$

$$\deg(g) = \deg(v) + \deg(d). \tag{2.4}$$

Substituting (2.3) into (2.4) yields

$$\deg(g) = \deg(v) + \deg(d) = \deg(v) + \deg(u) + \deg(g),$$

hence

$$\deg(u) + \deg(v) = 0.$$

Since $\deg(u), \deg(v) \geq 0$, we obtain $\deg(u) = \deg(v) = 0$, so $u, v \in R$. Together with (2.1) and (2.2), this implies $u = v = 1$, completing the proof. □

Definition 2.2.6. *Let $f_1, f_2 \in R[x]$ with $\deg(f_2) \leq \deg(f_1)$. Assume the MEA does not fail for the inputs f_1 and f_2 and terminates after $l + 1$ iterations. The **Monic Polynomial Remainder Sequence (m.p.r.s.)** generated by f_1 and f_2 is the sequence $r_1, r_2, \ldots, r_l$ obtained from the execution of the Monic Euclidean Algorithm s.t. $r_1 = f_1$, $r_2 = f_2$, $r_3 = r_1 - M_2 q_3$, and $r_{i+2} = M_i - M_{i+1} q_{i+2}$ with $M_i = \text{monic}(r_i)$ and $\deg(r_{i+1}) < \deg(r_i)$ for $2 \leq i \leq l - 1$ and $r_{l+1} = 0$.*

Remark 2.4. *Let $f_1, f_2 \in R[x]$ with $n = \deg(f_1) \geq m = \deg(f_2) \geq 0$.*

(i) The remainders appearing in the m.p.r.s. are polynomials obtained from Line 6 of Algorithm 2. In general, these remainders are not monic polynomials. We refer to this sequence as the Monic Polynomial Remainder Sequence because it is obtained from the MEA.

(ii) The number of division steps in the MEA is bounded by $m+1$. Since the m.p.r.s. contains both $r_1 = f_1$ and $r_2 = f_2$, the number of remainders in the m.p.r.s. generated by f_1 and f_2 is bounded by $l \leq m+2$.

Example 2.2.8. *In Example 2.2.7, the sequence $r_1, r_2 \ldots, r_5$ is the m.p.r.s. generated by f_1 and f_2.*

As with the EA, in the MEA, the size of the coefficients of the remainders in $\mathbb{Q}[x]$ grows rapidly. To address this issue, in our GCD and resultant algorithms, we run the MEA modulo a prime.

2.2.3 Pseudo-division

Let R be a commutative ring with $1 \neq 0$, and let $f_1, f_2 \in R[x]$. The quotient and remainder of f_1 divided by f_2 will not belong to $R[x]$ unless $\mathrm{lc}(f_2)$ is a unit in R (see Theorem 2.2.1). As noted after Example 2.2.1, it is impossible to divide $f_1 = x^3+2x+1$ by $f_2 = 3x^2+5$ with a remainder in $\mathbb{Z}[x]$ since $\mathrm{lc}(f_2) = 3$ is not a unit in $\mathbb{Z}$. In such scenarios, we utilize pseudo-division (refer to Section 2.7 of [18] and Section 6.12 of [35]). Moreover, pseudo-division is a fundamental tool for defining subresultants in Section 6.3.3.

Theorem 2.2.6. *Let $f_1, f_2 \in R[x]$ with $\deg(f_1) \geq \deg(f_2) \geq 0$ and assume that $\mathrm{lc}(f_2)$ is not a zero divisor. Set $\delta = \deg(f_1) - \deg(f_2) + 1$. Then there exist polynomials $\tilde{r}(x), \tilde{q}(x) \in R[x]$ s.t.*

$$\mathrm{lc}(f_2)^\delta f_1(x) = f_2(x)\tilde{q}(x) + \tilde{r}(x),$$

*where either $\deg(\tilde{r}(x)) < \deg(f_2)$ or $\tilde{r}(x) = 0$. We call $\tilde{r}(x) = \mathrm{prem}(f_1, f_2, x)$ the **pseudo-remainder** and $\tilde{q}(x) = \mathrm{pquo}(f_1, f_2, x)$ the **pseudo-quotient**.*

Proof. This is a slight generalization of [39, Theorem 2.2.2], which is stated for polynomials over an integral domain. The pseudo-division algorithm proceeds by repeatedly multiplying the current dividend by $\mathrm{lc}(f_2)$ and subtracting suitable multiples of f_2 to cancel the leading term. In Winkler's proof, the integral domain assumption is used only to ensure that multiplication by $\mathrm{lc}(f_2)$ does not decrease the degree of the current dividend during the pseudo-division process. This conclusion still holds under the weaker assumption that $\mathrm{lc}(f_2)$ is not a zero divisor in R. Therefore, the same argument applies verbatim and yields the asserted representation. □

Example 2.2.9. *Let*

$$f_1 = 6x^5 + 6x^4 - 2x^3 - x^2 + 2x + 1 \text{ and}$$
$$f_2 = 5x^4 + 5x^3 - 3x^2 + 2x + 5$$

be the polynomials given in Example 2.2.4. Then

$$\delta = \deg(f_1) - \deg(f_2) + 1 = 5 - 4 + 1 = 2 \quad \text{and} \quad \mathrm{lc}(f_2) = 5.$$

Multiplying f_1 by $\mathrm{lc}(f_2)^\delta$ and dividing the result by f_2, we obtain

$$\mathrm{prem}(f_1, f_2, x) = 40x^3 - 85x^2 - 100x + 25 \quad \text{and}$$
$$\mathrm{pquo}(f_1, f_2, x) = 30x.$$

Note that in this example we have $\operatorname{prem}(f_1, f_2, x), \operatorname{pquo}(f_1, f_2, x) \in \mathbb{Z}[x]$*, while in Example 2.2.4 the remainder and quotient of* f_1 *divided by* f_2 *are*

$$r_3 = \frac{8}{5}x^3 - \frac{17}{5}x^2 - 4x + 1 \quad \textit{and} \quad q_3 = \frac{6}{5}x \in \mathbb{Q}[x].$$

Remark 2.5. *If* $f_1, f_2 \in R[x]$*, then both* $\operatorname{prem}(f_1, f_2)$ *and* $\operatorname{pquo}(f_1, f_2)$ *lie in* $R[x]$*. In particular, if* $f_1, f_2 \in \mathbb{Z}[x]$*, then applying pseudo-division over* $\mathbb{Z}$ *ensures that no fractions appear in* $\operatorname{prem}(f_1, f_2)$ *and* $\operatorname{pquo}(f_1, f_2)$*. This illustrates how pseudo-division addresses the second issue discussed in Remark 2.3, namely, the appearance of rational coefficients when using the Euclidean algorithm over* $\mathbb{Z}[x]$*.*

Notation 2.3. *Let* $f_1, f_2 \in R[x]$ *be two non-zero polynomials s.t.* $\operatorname{lc}(f_2)$ *and* $\operatorname{lc}(f_1)$ *are units, and* $0 \leq \deg(f_2) \leq \deg(f_1)$*. In this thesis, we use the following notations.*

- *Let* $\operatorname{rem}(f_1, f_2)$ *and* $\operatorname{quo}(f_1, f_2)$ *denote the remainder and quotient of* f_1 *divided by* f_2*, where* $\operatorname{rem}(f_1, f_2) = 0$ *or* $\deg(\operatorname{rem}(f_1, f_2)) < \deg(f_2)$*, i.e.,* $\operatorname{rem}(f_1, f_2) = f_1 - f_2 \operatorname{quo}(f_1, f_2)$*.*
- *Let* $\operatorname{mrem}(f_1, f_2)$ *and* $\operatorname{mquo}(f_1, f_2)$ *be the remainder and quotient of* $\operatorname{monic}(f_1)$ *divided by* $\operatorname{monic}(f_2)$ *i.e.,* $\operatorname{mrem}(f_1, f_2) = \operatorname{monic}(f_1) - \operatorname{monic}(f_2) \operatorname{mquo}(f_1, f_2)$*.*
- *Let* $\operatorname{prem}(f_1, f_2)$ *and* $\operatorname{pquo}(f_1, f_2)$ *be the pseudo-remainder and pseudo-quotient of* f_1 *divided by* f_2*.*

In Lemma 2.2.7, we establish a connection between $\operatorname{prem}(f_1, f_2)$, $\operatorname{rem}(f_1, f_2)$, and $\operatorname{mrem}(f_1, f_2)$.

Lemma 2.2.7. *Let* $f_1, f_2 \in R[x]$ *be two non-zero polynomials s.t.* $\operatorname{lc}(f_2)$ *and* $\operatorname{lc}(f_1)$ *are units, and* $0 \leq \deg(f_2) \leq \deg(f_1)$*. Let* $\delta = \deg(f_1) - \deg(f_2) + 1$*. Then we have*

(i) $\operatorname{rem}(f_1, f_2) = \operatorname{lc}(f_1)\operatorname{mrem}(f_1, f_2)$ *and* $\operatorname{quo}(f_1, f_2) = \operatorname{lc}(f_2)^{-1}\operatorname{lc}(f_1) \operatorname{mquo}(f_1, f_2)$*.*

(ii) $\operatorname{prem}(f_1, f_2) = \operatorname{lc}(f_2)^{\delta}\operatorname{rem}(f_1, f_2)$ *and* $\operatorname{pquo}(f_1, f_2) = \operatorname{lc}(f_2)^{\delta} \operatorname{quo}(f_1, f_2)$*.*

(iii) $\operatorname{prem}(f_1, f_2) = \operatorname{lc}(f_2)^{\delta}\operatorname{lc}(f_1)\operatorname{mrem}(f_1, f_2)$ *and* $\operatorname{pquo}(f_1, f_2) = \operatorname{lc}(f_2)^{\delta-1}\operatorname{lc}(f_1) \operatorname{mquo}(f_1, f_2)$*.*

Proof. (i) Since $\operatorname{lc}(f_1)$ and $\operatorname{lc}(f_2)$ are units, we have

$$\begin{aligned} \operatorname{mrem}(f_1, f_2) &= \operatorname{monic}(f_1) - \operatorname{monic}(f_2) \operatorname{mquo}(f_1, f_2) \\ &= \operatorname{lc}(f_1)^{-1} f_1 - \operatorname{lc}(f_2)^{-1} f_2 \operatorname{mquo}(f_1, f_2). \end{aligned}$$

By multiplying $\operatorname{mrem}(f_1, f_2)$ with $\operatorname{lc}(f_1)$, we obtain

$$\operatorname{lc}(f_1)\operatorname{mrem}(f_1, f_2) = f_1 - f_2 \operatorname{lc}(f_2)^{-1}\operatorname{lc}(f_1) \operatorname{mquo}(f_1, f_2). \tag{2.5}$$

Given that $\operatorname{lc}(f_1)$ and $\operatorname{lc}(f_2)$ are units, it follows that

$$\begin{aligned} \deg(\operatorname{lc}(f_1) \operatorname{mrem}(f_1, f_2)) &= \deg(\operatorname{mrem}(f_1, f_2)) \text{and} \\ \deg(\operatorname{lc}(f_2)^{-1}\operatorname{lc}(f_1) \operatorname{mquo}(f_1, f_2)) &= \deg(\operatorname{mquo}(f_1, f_2)). \end{aligned}$$

By subtracting $\operatorname{rem}(f_1, f_2) = f_1 - f_2 \operatorname{quo}(f_1, f_2)$ from (2.5), we derive

$$\operatorname{lc}(f_1) \operatorname{mrem}(f_1, f_2) - \operatorname{rem}(f_1, f_2) = f_2(\operatorname{quo}(f_1, f_2) - \operatorname{lc}(f_2)^{-1}\operatorname{lc}(f_1) \operatorname{mquo}(f_1, f_2)). \tag{2.6}$$

By definition, $\deg(\text{mrem}\ (f_1, f_2)) < \deg(f_2)$ or $\text{mrem}\ (f_1, f_2) = 0$ and $\deg(\text{rem}\ (f_1, f_2)) < \deg(f_2)$ or $\text{rem}\ (f_1, f_2) = 0$. Therefore, from (2.6), we conclude

$$\text{lc}(f_1)\ \text{mrem}(f_1, f_2) = \text{rem}(f_1, f_2) = 0. \tag{2.7}$$

Since $f_2 \neq 0$ and from (2.7), it follows from (2.6) that

$$\text{quo}(f_1, f_2) - \text{lc}(f_2)^{-1}\text{lc}(f_1)\ \text{mquo}(f_1, f_2) = 0,$$

which completes the proof of part (i).

(ii) By Theorem 2.2.6, we have $\text{prem}\ (f_1, f_2) = \text{lc}(f_2)^{\delta} f_1 - f_2 \text{pquo}\ (f_1, f_2)$. Since $\text{lc}(f_2)$ is a unit, we can multiply $\text{prem}\ (f_1, f_2)$ by $\text{lc}(f_2)^{-\delta}$ which implies that

$$\text{lc}(f_2)^{-\delta}\text{prem}\ (f_1, f_2) = f_1 - f_2\ \text{lc}(f_2)^{-\delta}\text{pquo}\ (f_1, f_2). \tag{2.8}$$

As $\text{lc}(f_2)$ is a unit, $\deg(\text{lc}(f_2)^{-\delta}\ \text{prem}\ (f_1, f_2)) = \deg(\text{prem}\ (f_1, f_2))$ and $\deg(\text{lc}(f_2)^{-\delta}\ \text{pquo}\ (f_1, f_2)) = \deg(\text{pquo}\ (f_1, f_2))$. Similar to part (i), we subtract $\text{rem}\ (f_1, f_2) = f_1 - f_2\ \text{quo}(f_1, f_2)$ from (2.8), which implies that

$$\text{lc}(f_2)^{-\delta}\ \text{prem}\ (f_1, f_2) - \text{rem}(f_1, f_2) = f_2\ (\text{quo}\ (f_1, f_2) - \text{lc}(f_2)^{-\delta}\ \text{pquo}\ (f_1, f_2)). \tag{2.9}$$

Since $f_2 \neq 0$, $\deg(\text{prem}\ (f_1, f_2)) < \deg(f_2)$, and $\deg(\text{rem}\ (f_1, f_2)) < \deg(f_2)$, we must have

$$\text{quo}(f_1, f_2) - \text{lc}(f_2)^{-\delta}\ \text{pquo}\ (f_1, f_2) = 0,$$

resulting in the proof of part (ii).

(iii) Part (iii) is a direct consequence of parts (i) and (ii)

□

Example 2.2.10. *Let* $f_1 = 2x^3 + x^2 + 2$ *and* $f_2 = 5x^2 + 3$ *be two polynomials in* $\mathbb{Q}[x]$. *We use Maple to show that Lemma 2.2.7 holds for* f_1 *and* f_2.

```
>f1 := 2*x^3+x^2+2:
>f2 := 5*x^2+3:
>mf1 := f1*lcoeff(f1)^(-1); #Monic(f1)
>mf2 := f2*lcoeff(f2)^(-1); #Monic(f2)
>delta:=degree(f1)-degree(f2)+1;
                                        mf1 := x^3 + 1/2x^2 + 1
                                        mf2 := x^2 + 3/5
                                        delta := 2
```

In Maple, the command `rem(f1,f2,x,'q')` *computes the* $\text{rem}\ (f_1, f_2)$ *w.r.t. the variable* x *with quotient* q. *Also, the command* `prem(f1,f2,x,'m','pquo')` *in Maple computes the* $\text{prem}\ (f_1, f_2)$ *w.r.t. the variable* x, *where* $m = \text{lc}(f_2)^{\delta}$, *and the quotient is pquo. To determine* $\text{mrem}(f_1, f_2)$, *we first compute* $mf_1 = \text{monic}(f_1)$ *and* $mf_2 = \text{monic}(f_2)$, *and then proceed to calculate* $\text{rem}\ (mf_1, mf_2)$.

```
>r := rem(f1, f2, x,'q');
```

```
>q := q; #Quotient of dividing f1 by f2
>mr := rem(mf1,mf2,x,'mq');
>mq := mq; #Quotient of dividing mf1=monic(f1) by mf2=monic(f2)
>pr := prem(f1,f2,x,'m','pquo');
>pquo := pquo; #Pseudo-quotient of dividing f1 by f2
                                        r:=-6/5x +7/5
                                        q:= 2x/5+ 1/5
                                        mr:=-3/5x + 7/10
                                        mq:=x+1/2
                                        pr:=-30x + 35
                                        pquo:=10x + 5
```

Now, we verify if Lemma 2.2.7 holds for the polynomials f_1 and f_2:

```
>#Part i of the lemma
>r - lcoeff(f1)*mr, q - lcoeff(f1)*mq/lcoeff(f2) ;
>#Part ii of the lemma
>pr - lcoeff(f2)^(delta)*r, pquo-lcoeff(f2)^(delta)*q ;
>#Part iii of the lemma
pr - lcoeff(f2)^(delta)*lcoeff(f1)*mr, pquo-lcoeff(f2)^(delta-1)*lcoeff(f1)*mq;
                                        0,0
                                        0,0
                                        0,0
```

In Theorem 6.3.12 of Chapter 6, we establish a connection between p.r.s. and the Subresultant Polynomial Sequence (as defined in Definition 6.3.2). To do so, we employ the following lemma.

Lemma 2.2.8. *Let $f_1, f_2 \in R[x]$ be two non-zero polynomials s.t. $\mathrm{lc}(f_2)$ is a unit. Let $a, b \in R$ be units and $\delta = \deg(f_1) - \deg(f_2) + 1$. Then,*

(i) $\mathrm{rem}\ (af_1, bf_2) = a \cdot \mathrm{rem}\ (f_1, f_2)$ *and* $\mathrm{quo}(af_1, bf_2) = ab^{-1} \cdot \mathrm{quo}(f_1, f_2)$.

(ii) $\mathrm{prem}\ (af_1, bf_2) = ab^{\delta} \cdot \mathrm{prem}\ (f_1, f_2)$ *and* $\mathrm{pquo}(af_1, bf_2) = ab^{\delta-1} \cdot \mathrm{pquo}(f_1, f_2)$.

(iii) $\mathrm{mrem}(af_1, bf_2) = \mathrm{mrem}(f_1, f_2)$ *and* $\mathrm{mquo}(af_1, bf_2) = \mathrm{mquo}(f_1, f_2)$.

Proof. (i) Let $r = \mathrm{rem}\ (f_1, f_2) = f_1 - f_2 q$ and $\bar{r} = \mathrm{rem}\ (af_1, bf_2) = af_1 - bf_2\bar{q}$ where q and $\bar{q}$ are the quotient polynomials in $R[x]$. By multiplying r by a and subtracting this product from $\bar{r}$, we obtain

$$\bar{r} - ar = f_2(aq - b\bar{q}). \tag{2.10}$$

Since $a \in R$ is a unit, we have $\deg(ar) = \deg(r)$. Moreover, by definition, $\deg(r), \deg(\bar{r}) < \deg(f_2)$. Thus (2.10) holds if and only if $aq - b\bar{q} = 0$, which implies that $\bar{r} = ar$ and $\bar{q} = \frac{a}{b}q$.

(ii) Let $r = \mathrm{prem}\ (f_1, f_2) = \mathrm{lc}(f_2)^{\delta} f_1 - f_2 q$ and $\bar{r} = \mathrm{prem}\ (af_1, bf_2) = \mathrm{lc}(bf_2)^{\delta} af_1 - bf_2\bar{q}$, where q and $\bar{q}$ represent the pseudo-quotient polynomials in $R[x]$. Given that b and $\mathrm{lc}(f_2)$ are units, we have $\bar{r} = b^{\delta}\mathrm{lc}(f_2)^{\delta} af_1 - bf_2\bar{q}$. Multiplying r by ab^{δ} and subtracting the result from $\bar{r}$, we have

$$\bar{r} - ab^{\delta} r = f_2(ab^{\delta} q - b\bar{q}).$$

Since b and a are units and $\deg(r), \deg(\bar{r}) < \deg(f_2)$, we must have $\bar{q} = ab^{\delta-1}q$, which implies that $\bar{r} = ab^{\delta} r$.

(iii) The proof follows from Definition 2.2.5.

□

2.2.4 The GCDHEU algorithm

The GCDHEU algorithm, introduced by Char, Geddes, and Gonnet[9], is a heuristic algorithm designed to compute the GCD of two polynomials over $\mathbb{Z}$. The core idea of the GCDHEU algorithm is to convert the problem of computing the GCD over $\mathbb{Z}[x]$ into a problem over $\mathbb{Z}$. Let $f_1 = \sum_{i=1}^{n} a_i x^i$ and $f_2 = \sum_{j=1}^{m} b_j x^j$ be two non-zero polynomials in $\mathbb{Z}[x]$. We aim to determine $g = \gcd(f_1, f_2)$. Define $H = \max_{i,j}(\mid a_i \mid, \mid b_j \mid)$. Heuristically, each coefficient of g is expected to have a magnitude no greater than H. Let $\beta \in \mathbb{Z}$ s.t. $\beta > 2H$. The GCDHEU algorithm starts by evaluating f_1 and f_2 at β, then computes the GCD as

$$g_\beta = \gcd(f_1(\beta), f_2(\beta)) \in \mathbb{Z}.$$

Subsequently, the algorithm converts g_β into g by determining the β-adic representation with a symmetric coefficient form, expressed as

$$g_\beta = c_0 + c_1\beta + c_2\beta^2 + \ldots + c_k\beta^d.$$

Finally, $g(x)$ is obtained by substituting β with x. We demonstrate this algorithm with the following example.

Example 2.2.11. *Let*

$$f_1(x) = (4x^3 + 2x + 1)(6x + 3),$$
$$f_2(x) = (4x^3 + 2x + 1)(7x + 14) \in \mathbb{Z}[x].$$

By inspection, $H = 56$. *To determine* $g = \gcd(f_1, f_2)$, *the GCDHEU algorithm initiates by selecting an evaluation point* $\beta = 1000 > 2H$ *and evaluating*

$$f_1(1000) = 24012012012003,$$
$$f_2(1000) = 28056014035014.$$

Then, by applying the Euclidean algorithm, it determines the integer GCD:

$$G = \gcd(f_1(1000), f_2(1000)) = 12000006003.$$

By calculating the symmetric 1000*-adic representation of* G *(see [18, Section* 6.2*]), we derive the corresponding polynomial as* $h = 12x^3 + 6x + 3$, *with* $x = 1000$. *The GCD* h *might contain an extraneous factor, i.e.* $h = C \cdot g(1000)$, *where* $C = \gcd\left(\frac{f_1}{g}(1000), \frac{f_2}{g}(1000)\right)$. *To extract* g *from* h, *we need to remove* C *from* h. *Here,* $C = \mathrm{cont}(h, x) = 3$. *Thus*

$$g = \frac{h}{3} = 4x^3 + 2x + 1.$$

Since g *divides both* f_1 *and* f_2, *we confirm that* $g = \gcd(f_1, f_2)$.

When C is too large, leading to a failure in division, the algorithm may not succeed. In such cases, it can be effective to choose an alternative, typically larger, evaluation point. The GCDHEU algorithm can be extended to compute multivariate gcds by recursively evaluating each variable at an integer. Despite generating large integer GCDs, the algorithm can be fast if the input polynomials are dense and fast GCD computation over $\mathbb{Z}$ is available. The authors noted that GCDHEU is frequently effective for GCD computations involving up to four variables. Before the release of Maple 11, GCDHEU was used in Maple for dense, small problems. Starting with Maple 11, Maple instead employs Zippel's GCD algorithm [40].

2.2.5 Modular GCD algorithms

The Euclidean algorithm allows us to compute the GCD of univariate polynomials over a field. However, this algorithm is not directly applicable to multivariate polynomials over a field. To address this challenge, we turn to modular algorithms, which provide a powerful framework for managing computation with multivariate polynomials. Modular arithmetic is particularly effective in controlling the growth of integer coefficients in intermediate remainders appearing in the EA and the MEA (see Example 2.2.4 and 2.2.7), as it confines all coefficients to a finite field. In general, the modular algorithms employ two core homomorphisms: the modular homomorphism and the evaluation homomorphism.

Definition 2.2.7. *Let $p \in \mathbb{Z}_{>0}$ and $\beta \in R$.*

(i) The modular homomorphism $\phi_p : \mathbb{Z} \longrightarrow \mathbb{Z}_p$ maps each integer to its residue modulo p, so that $0 \leq \phi_p(a) < p$ for $a \in \mathbb{Z}$.

(ii) Let $R' = R[x_1, \ldots, x_{k-1}]$ and $f \in R'[x_k]$. We define the evaluation homomorphism $\phi_{x_k=\beta} : R'[x_k] \longrightarrow R'$ s.t. $\phi_{x_k=\beta}(f) = f(x_k = \beta)$.

In our modular algorithms, we choose p to be prime, so that $\mathbb{Z}_p$ is a finite field. The modular homomorphism is used to prevent the growth of integer coefficients of algebraic numbers in the EA. Of course, if we only use the two homomorphisms in a modular algorithm, we will lose information. To reconstruct the lost information, we must invert these homomorphisms at some point. To invert the modular homomorphisms, the Chinese Remainder Theorem (CRT) is applied to a collection of modular homomorphic images, $\phi_{p_1} \ldots \phi_{p_s}$. Furthermore, to invert the evaluation homomorphism, we use polynomial interpolation, as explained in Section 2.2.7. The reconstruction process of the target GCD is not guaranteed to succeed for all choices of primes and evaluation points. The following example in $\mathbb{Z}[x]$ illustrates this.

Example 2.2.12. *Let $f_1 = x^2 + 4x + 4$ and $f_2 = x + 5$ be two polynomials in $\mathbb{Z}[x]$. By inspection, f_1 and f_2 are relatively prime, i.e. $g = \gcd(f_1, f_2) = 1$. Reducing f_1 and f_2 modulo $p = 3$, we obtain*

$$\phi_3(f_1) = (x+2)^2 \text{ and } \phi_3(f_2) = x + 2 \in \mathbb{Z}_3[x].$$

Therefore,

$$\gcd(\phi_3(f_1), \phi_3(f_2)) = x + 2 \in \mathbb{Z}_3[x].$$

This result does not provide complete information about the actual GCD $g = 1$.

To identify the primes and evaluation points that are unsuitable for reconstructing the target GCD, such as $p = 3$ in Example 2.2.12, we employ Lemma 2.2.9. In [6], Brown proves Lemma 2.2.9 under the assumption that R and R' are UFDs (see also [18, Lemma 7.3]). Here, we establish Lemma 2.2.9 in the more general setting where R and R' are commutative rings with $1 \neq 0$.

Lemma 2.2.9. *Let R and R' be commutative rings with $1 \neq 0$ and $\phi : R \longrightarrow R'$ be a ring homomorphism with $\phi(1) = 1$. Let f_1 and f_2 be two non-zero polynomials in $R[x_1, \ldots, x_k]$. Fix a monomial ordering on $R[x_1, \ldots, x_k]$. Suppose that the monic $g = \gcd(f_1, f_2)$ and the monic $g_\phi = \gcd(\phi(f_1), \phi(f_2))$ exist. If $\phi(\mathrm{lc}(f_2)) \neq 0$, then*

(i) $\mathrm{lm}(g_\phi) \geq \mathrm{lm}(g)$ *and*

(ii) If $\mathrm{lm}(g_\phi) = \mathrm{lm}(g)$, *then* $g_\phi = \phi(g)$.

Proof. (i) Let $p = f_1/g$ and $q = f_2/g$ be two polynomials in $R[x_1, \ldots, x_k]$. By the ring homomorphism property of ϕ, since $f_1 = p \cdot g$ and $f_2 = q \cdot g$, we obtain

$$\phi(f_1) = \phi(p) \cdot \phi(g) \quad \text{and} \quad \phi(f_2) = \phi(q) \cdot \phi(g).$$

Since $\phi(\mathrm{lc}(f_2)) \neq 0$, it follows that $\phi(f_2) \neq 0$. Furthermore, as $\phi(\mathrm{lc}(g)) = \phi(1) = 1 \neq 0$, we have $\phi(g) \neq 0$. Therefore, $\phi(g)$ is a common factor of $\phi(f_1)$ and $\phi(f_2)$, and hence $\phi(g) \mid g_\phi$. Thus, there exists a non-zero polynomial $h \in R'[x_1, \ldots, x_k]$ s.t.

$$g_\phi = h \cdot \phi(g). \tag{2.11}$$

If $\mathrm{lc}(h) \cdot \mathrm{lc}(\phi(g)) = 0$, then $\mathrm{lc}(h) \cdot 1 = 0$, which implies $\mathrm{lc}(h) = 0$, contradicting the assumption $h \neq 0$. Therefore,

$$\mathrm{lc}(g_\phi) = \mathrm{lc}(h \cdot \phi(g)) = \mathrm{lc}(h) \cdot \mathrm{lc}(\phi(g)) \neq 0.$$

From Corollary 2.1, Part (i),

$$\mathrm{lm}(g_\phi) = \mathrm{lm}(h) \cdot \mathrm{lm}(\phi(g)), \tag{2.12}$$

and hence $\mathrm{lm}(g_\phi) \geq \mathrm{lm}(\phi(g))$. Moreover, since g is monic, we have $\phi(\mathrm{lc}(g)) = \phi(1) = 1 \neq 0$, which implies $\mathrm{lc}(\phi(g)) = 1$ and hence $\mathrm{lm}(\phi(g)) = \mathrm{lm}(g)$. Therefore,

$$\mathrm{lm}(g_\phi) \geq \mathrm{lm}(\phi(g)) = \mathrm{lm}(g).$$

(ii) Similar to part (i) as $\phi(\mathrm{lc}(g)) = \phi(1) = 1$, we have $\mathrm{lm}(\phi(g)) = \mathrm{lm}(g)$. Let h be as defined in part (i). Then by (2.11) and (2.12), we obtain

$$\begin{aligned} g_\phi = h \cdot \phi(g) &\Longrightarrow \mathrm{lm}(g_\phi) = \mathrm{lm}(h) \cdot \mathrm{lm}(\phi(g)) = \mathrm{lm}(h) \cdot \mathrm{lm}(g) \\ &\Longrightarrow \mathrm{lm}(g_\phi) = \mathrm{lm}(h) \cdot \mathrm{lm}(g). \end{aligned} \tag{2.13}$$

By assumption, $\mathrm{lm}(g_\phi) = \mathrm{lm}(g)$. Hence, from (2.13) we obtain $\mathrm{lm}(h) = 1$, so $h \in R'$. Since both $\phi(g)$ and g_ϕ are monic, the equality $g_\phi = h \cdot \phi(g)$ implies $h = 1$, completing the proof.

□

Definition 2.2.8. *Let f_1, f_2, g, and g_ϕ be as defined in Lemma 2.2.9. If $\mathrm{lm}(g_\phi) > \mathrm{lm}(g)$ (see Example 2.2.12), then we call ϕ an* ***unlucky homomorphism****.*

The image produced by an unlucky homomorphism must not be utilized in reconstructing the target GCD. However, we do not know $\mathrm{lm}(g)$ in advance, so how can we detect unlucky homomorphisms? To answer this question, for the i-th iteration, we set $g_{\phi_i} = \gcd(\phi_i(f_1), \phi_i(f_2))$, where ϕ_i is the homomorphism used in that iteration. If $\mathrm{lm}(g_{\phi_i}) > \mathrm{lm}(g_{\phi_{i-1}})$, then ϕ_i is an unlucky homomorphism, and g_{ϕ_i} must be ignored. Moreover, if $\mathrm{lm}(g_{\phi_i}) < \mathrm{lm}(g_{\phi_{i-1}})$, then all the previously used homomorphisms up to the i-th one are unlucky, and $g_{\phi_1}, g_{\phi_2}, \ldots, g_{\phi_{i-1}}$ must be ignored. Briefly, we keep the image GCDs with the least leading monomial. This approach was first proposed by Brown [7]. We adopt this strategy in Algorithm 8 in Chapter 4. Furthermore, if $\phi_i(\mathrm{lc}(f_2)) = 0$, then Lemma 2.2.9 does not apply to ϕ_i, and we cannot determine whether ϕ_i is an unlucky homomorphism. In this case, we cannot reconstruct g from its image g_{ϕ_i}, since it may happen that $\mathrm{lm}(g_{\phi_i}) < \mathrm{lm}(g)$. Therefore, in modular GCD algorithms, we require homomorphisms such as ϕ to satisfy $\phi(\mathrm{lc}(f_2)) \neq 0$. The following example illustrates this situation.

Example 2.2.13. *Let $f_1 = (5x^2 - x + 1)(6x^2 + 1)$ and $f_2 = (x + 4)(6x^2 + 1)$ be two polynomials in $\mathbb{Z}[x]$. By inspection, $g = \gcd(f_1, f_2) = 6x^2 + 1$ and $\mathrm{lc}(f_2) = 6$. If we choose $p = 3$, then $\phi_p(\mathrm{lc}(f_2)) = 0$ and*

$$\phi_p(f_1) = 2x^2 + 2x + 1$$
$$\phi_p(f_2) = x + 1 \in \mathbb{Z}_p[x]$$

with

$$g_{\phi_p} = \gcd(\phi_p(f_1), \phi_p(f_2)) = 1.$$

We cannot recover $g = 6x^2 + 1$ from its image $g_{\phi_p} = 1$ since $\mathrm{lm}(g_{\phi_p}) < \mathrm{lm}(g)$.

In Lemma 2.2.9, the ring R' may contain zero divisors. We will discuss this matter in more detail in Sections 4.4.3 and 5.4.2. We fix the lexicographic order on $R = \mathbb{Z}[x_1, \ldots, x_k]$ with $x_1 > x_2 > \cdots > x_k$. In the following example, we illustrate the general structure of modular GCD algorithms.

Example 2.2.14. *Consider the polynomials f_1 and f_2 in $\mathbb{Z}[x]$ given as follows:*

$$f_1 = (2x + 7)(x + 1)(2x^2 + 1) \text{ and}$$
$$f_2 = (2x + 7)(6x + 11).$$

By inspection $g = \gcd(f_1, f_2) = 2x + 7$. We choose the first prime to be $p_1 = 5$. Since $p_1 \nmid \mathrm{lc}(f_2) = 12$, we can use it for modular reduction. By reducing f_1 and f_2 modulo p_1, we obtain the polynomials

$$\phi_5(f_1) = 4(x + 1)^2(x^2 + 3) \text{ and}$$
$$\phi_5(f_2) = 2(x + 1)^2$$

in $\mathbb{Z}_5[x]$, where the monic GCD is

$$g_5 = \gcd(\phi_5(f_1), \phi_5(f_2)) = (x + 1)^2 = 1 \cdot x^2 + 2x + 1 \in \mathbb{Z}_5[x].$$

Next, we select another prime $p_2 = 7$ *s.t.* $p_2 \nmid \mathrm{lc}(f_2)$ *and reduce* f_1 *and* f_2 *modulo* p_2 *to obtain the polynomials*

$$\phi_7(f_1) = 4x(x+1)(x^2+4) \text{ and}$$
$$\phi_7(f_2) = 5x(x+3),$$

in $\mathbb{Z}_7[x]$*, where their monic GCD is*

$$g_7 = \gcd(\phi_7(f_1), \phi_7(f_2)) = 1 \cdot x \in \mathbb{Z}_7[x].$$

Thus, g_7 *could be a candidate for the monic* g *in* $\mathbb{Z}_7[x]$*. Since* $\mathrm{lm}(g_7) = x < \mathrm{lm}(g_5) = x^2$*, part (ii) of Lemma 2.2.9 does not hold for* g_5 *so the homomorphism* ϕ_5 *is identified as an unlucky homomorphism, and its image* g_5 *must be disregarded. Repeating the same process for a new prime* $p_3 = 13$*, we obtain*

$$\phi_{13}(f_1) = 4(x+1)(x+10)(x^2+7) \text{ and}$$
$$\phi_{13}(f_2) = 12(x+4)(x+10)$$

in $\mathbb{Z}_{13}[x]$*, where their monic GCD is given by:*

$$g_{13} = \gcd(\phi_{13}(f_1), \phi_{13}(f_2)) = 1 \cdot x + 10 \in \mathbb{Z}_{13}[x].$$

Hence, the candidate for the monic g *in* $\mathbb{Z}_{13}[x]$ *is* g_{13}*. Since both images,* g_7 *and* g_{13}*, have the same degree, we assume* p_2 *and* p_3 *are not unlucky primes. Therefore, we assume that the target monic GCD also has degree* 1*. Hence,*

$$\mathrm{monic}(g) = \gcd(f_1, f_2) = x + b, \quad b \in \mathbb{Z}.$$

Let $M = p_2 \times p_3 = 91$*. We now apply the Chinese remainder theorem to determine* g_M *from*

$$g_M \equiv g_7 \mod 7, \; g_M \equiv g_{13} \mod 13,$$

that is, we solve

$$\{g_M \equiv x \mod 7, \; g_M \equiv x + 10 \mod 13\},$$

and obtain

$$g_M = x + 49 \in \mathbb{Z}_M[x].$$

The question remains: how can we recover $g = 2x + 7$ *from its monic image* $g_M = x + 49 \in \mathbb{Z}_M[x]$*? By inspection, we know* $\mathrm{monic}(g) = x + \frac{7}{2}$*. Since* g_7 *and* g_{13} *are monic,* g_M *is also monic, and it is a candidate for* $\mathrm{monic}(g)$ *modulo* M *so*

$$g_M = \phi_M(\mathrm{monic}(g)).$$

If we knew $\mathrm{lc}(g) = 2$ *in advance, we would just scale* g_M *by* 2 *modulo* M*. In other words, if we use the symmetric range for the integers modulo* M*,* $\mathbb{Z}_M = \{[z] \text{ s.t } -M/2 < z < M/2\}$*, then*

$$\phi_M(\mathrm{lc}(g) \cdot g_M) = \phi_M(g).$$

However, in general, we do not know $\mathrm{lc}(g)$ *in advance. The approach used by Collins and Brown for* $f_1, f_2 \in \mathbb{Z}[x]$ *is to use the fact that* $\mathrm{lc}(g) \mid \mathrm{lc}(f_1)$ *and* $\mathrm{lc}(g) \mid \mathrm{lc}(f_2)$*. Let* $c = \gcd(\mathrm{lc}(f_1), \mathrm{lc}(f_2))$*, then* $\mathrm{lc}(g) \mid c$*. Thus, instead of scaling* g_M *by* $\mathrm{lc}(g)$*, we scale it by* c *modulo* M*. Provided that* M *is large enough, we will obtain an integer multiple of* g*,*

$$\phi_M(c \cdot g_M) = \frac{c}{\mathrm{lc}(g)} \cdot g.$$

In our example $c = \gcd(4, 12) = 4$ *and we obtain*

$$\phi_M(c \cdot g_M) = 4x + 14.$$

Dividing by the integer multiple of 2*, we get* $g = 2x + 7$*.*

This approach does not extend to $f_1, f_2 \in \mathbb{Q}(\alpha)[x]$*, since* $\gcd(\mathrm{lc}(f_1), \mathrm{lc}(f_2)) \in \mathbb{Q}(\alpha)$ *lies in a field. Instead, we apply Wang's rational number reconstruction algorithm to recover the rational coefficients of* $\mathrm{monic}(g) = x + \frac{7}{2}$ *from their images modulo* M*. This will be discussed in the next section. Encarnacion [14] was the first to apply rational number reconstruction to compute the monic GCD of two polynomials in* $\mathbb{Q}(\alpha)[x]$ *(see Section 4.3).*

2.2.6 Rational number reconstruction

Notation 2.4. *For simplicity, the following notations will remain unchanged and will be used consistently throughout this thesis.*

1. *CRT stands for Chinese Remainder Theorem, and*
2. *RNR stands for Rational Number Reconstruction.*

In our modular algorithms, MGCDNF and MRESNF, once the CRT has been applied, we are left with the task of reconstructing the rational coefficients of the target GCD and the resultant. To do this, we employ the algorithm of Paul S.Wang [37]. Let $\frac{n}{d} \in \mathbb{Q}$ with $\gcd(n, d) = 1$, and let $m \in \mathbb{Z}$ s.t. $\gcd(d, m) = 1$. Suppose we have computed $u \equiv \frac{n}{d} \pmod{m}$. The goal of **rational number reconstruction** is to recover the integers n and d from the known values of u and m. Wang's method solves this problem using the Extended Euclidean Algorithm (EEA).

By running the EEA on the inputs m and u, each remainder r_i produced during the algorithm can be expressed as a linear combination of m and u. That is,

$$r_i = s_i m + t_i u$$

for some integers s_i and t_i. Reducing both sides modulo m, we obtain:

$$r_i \equiv t_i u \pmod{m}.$$

If $\gcd(t_i, m) = 1$, then it follows that

$$\frac{r_i}{t_i} \equiv u \pmod{m}.$$

Thus, the set $S = \left\{ \frac{r_i}{t_i} \right\}_i$ contains candidate rational reconstructions of u modulo m.

Algorithm 3: Extended Euclidean Algorithm (EEA)

Input: Integers a, b with $b \neq 0$.
Output: Integers x, y, d s.t. $g = \gcd(a, b)$ and $ax + by = g$.

$r_0 = a$, $r_1 = b$
$s_0 = 1$, $s_1 = 0$
$t_0 = 0$, $t_1 = 1$
$i = 1$
while $r_i \neq 0$ **do**
  $q = \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor$
  $r_{i+1} = r_{i-1} - q \cdot r_i$
  $s_{i+1} = s_{i-1} - q \cdot s_i$
  $t_{i+1} = t_{i-1} - q \cdot t_i$
  $i = i + 1$
Set $g = r_{i-1}$, $x = s_{i-1}$, $y = t_{i-1}$
return (x, y, g)

Example 2.2.15. *Consider running the EEA with inputs $m = 23$ and $u = 7$. We keep track of the values $\frac{r_i}{t_i}$ at each step. This yields the following set of candidate rational reconstructions:*

$$S = \left\{ \frac{7}{1}, \frac{2}{-3}, \frac{1}{10} \right\}.$$

Each rational number in the set S is congruent to 7 modulo 23.

The following theorem, originally presented by Wang, Guy, and Davenport [38], establishes that the RATCONVERT algorithm developed by Wang [37] reliably reconstructs a rational number $\frac{n}{d}$, provided that the modulus m satisfies the condition $m > 2ND$, where $N, D \in \mathbb{N}$, $N \geq |\, n \,|$, $\gcd(m, d) = 1$, and $D \geq d > 0$.

Theorem 2.2.10. *(Wang, Guy and Davenport, 1982) Let $n, d \in \mathbb{Z}$ with $d > 0$ and $\gcd(n, d) = 1$. Let $m \in \mathbb{N}$ s.t. $\gcd(m, d) = 1$, and suppose $u \equiv \frac{n}{d} \pmod{m}$, where $u \in [0, m)$. Given bounds $N, D \in \mathbb{Z}$ satisfying $N \geq |n|$ and $D \geq d$, the following statements hold:*

- *If $m > 2ND$, then for any $u \in [0, m)$, there exists a unique rational number $\frac{n}{d}$ s.t.*

$$\frac{n}{d} \equiv u \pmod{m}.$$

- *If $m > 2ND$, then given m and u, there exists a unique index i in the extended Euclidean algorithm s.t.*

$$\frac{r_i}{t_i} = \frac{n}{d}.$$

Furthermore, i is the first index for which $r_i \leq N$.

Example 2.2.16. *Let $N = D = 200$, so that the condition $m > 2ND$ is satisfied. Table 2.3 illustrates the iterations of the EEA for the inputs $u = 22234$ and $m = 99991$. We observe that $i = 3$ is the first index for which the remainder satisfie $r_3 = 124 \leq N = 200$. According to Wang's algorithm, this enables us to recover the rational number $\frac{r_3}{t_3} = \frac{n}{d} = \frac{124}{9}$ which satisfies $\frac{124}{9} \equiv u \mod m$.*

i	r_i	t_i	q	$\frac{r_i}{t_i}$
1	22234	1	4	$\frac{22234}{1}$
2	11055	-4	2	$\frac{-11055}{4}$
3	124	9	89	$\frac{124}{9}$
4	19	-805	6	$\frac{-19}{805}$
5	10	4839	1	$\frac{10}{4839}$
6	9	-5644	1	$\frac{-9}{5644}$
7	1	10483	9	$\frac{1}{10483}$

Table 2.3: Extended Euclidean algorithm steps

Maple's command $iratrecon(u, m, N, D)$ defaults to using Wang's algorithm when N and D are specified.

```
>iratrecon(22234,99991,200,200);
```

$$\frac{124}{9}$$

In cases where N and D are not specified, $iratrecon(u, m)$ calls Wang's algorithm with $N = D = k$, where k represents the largest integer s.t. $2k^2 < m$.

```
>iratrecon(22234,99991);
```

$$\frac{124}{9}$$

Currently, our modular algorithms employ Wang's algorithm without specifying N and D. In the following example, we illustrate how CRT and RNR can be combined to reconstruct the rational coefficients of the monic GCD of two polynomials in $\mathbb{Z}[x]$ from its modular images.

Example 2.2.17. *Consider the polynomials $f_1, f_2 \in \mathbb{Z}[x]$ given in Example 2.2.14:*

$$f_1 = (2x + 7)(x + 1)(2x^2 + 1) \text{ and}$$
$$f_2 = (2x + 7)(6x + 11).$$

In this example, we first obtain the monic GCD $x + \frac{7}{2}$ and then clear denominators to obtain the GCD $g = 2x + 7$. From Example 2.2.14, we know that $p = 5$ is an unlucky prime, so we do not use it here. We run this example in Maple. In Maple, the command $GCD(f_1, f_2) \bmod p$ computes the monic GCD of f_1 and f_2 modulo p.

```
>f1:=expand((2*x + 7)*(x+1)*(2*x^2+1)):
>f2:=expand((2*x + 7)*(6*x + 11)):
>#We reduce the inputs modulo p1=7 and compute GCD in Z_p1[x]
>p1:=7:
>F1:=f1 mod p1:
>F2:=f2 mod p1:
>G1:=Gcd(F1,F2) mod p1; #Monic GCD in Z_p1[x]
                          G1:= x
>#We reduce the inputs modulo p2 and compute GCD in Z_p2[x]
```

```
>p2:=11:
>F1:=f1 mod p2:
>F2:=f2 mod p2:
>G2:=Gcd(F1,F2) mod p2; #Monic GCD in Z_p2[x]
                              G2 := x+9
```

Let $P = [p_1, \ldots, p_n]$ *be a list of prime numbers and* $M = \prod_{i=1}^{n} p_i$. *Let* $G = [G_1, \ldots, G_n]$ *be the corresponding list of polynomials s.t. for each* $1 \leq i \leq n$, *the polynomial* G_i *is over* $\mathbb{Z}_{p_i}$. *In Maple, the command chrem*(G, P) *uses CRT to compute the polynomial* $C \in \mathbb{Z}_M$ *s.t.* $C \mod p_i = G_i$ *for each* $1 \leq i \leq n$. *To reconstruct the rational coefficients of the target GCD using RNR, we use the command iratrecon modulo* M.

```
>gM:=chrem([G1,G2],[p1,p2]); #CRT
                                  gM:= x+42
>M:=p1*p2:
>R1:=iratrecon(gM,M); #RNR
                                     R1:= FAIL
```

A failure in the RNR process indicates a requirement for additional modular images and larger moduli. As a result, we repeat the above process for a new prime $p_3 = 13$.

```
>p3:=13:
>F1:=f1 mod p3:
>F2:=f2 mod p3:
>G3:=Gcd(F1,F2) mod p3;
                              G3:= x+10
>gM:=chrem([G1,G2,G3],[p1,p2,p3]);
                              gM := x+504
>M:=p1*p2*p3:
>R2:=iratrecon(gM,M);
                              R2:=x+7/2
```

Repeating the above process for a new prime $p_4 = 17$, *we ensure that the result of the RNR is stabilized.*

```
>p4:=17:
>F1:=f1 mod p4:
>F2:=f2 mod p4:
>G4:=Gcd(F1,F2) mod p4;
                              G4 := x+12
>gM:=chrem([G1,G2,G3,G4],[p1,p2,p3,p4]);
                             gM := x + 8512
>M:=p1*p2*p3*p4:
>R3:=iratrecon(gM,M);
                               R3:= x+7/2
```

Since the result of the RNR stabilized, $R2 = R3$, *we employ the division test to check whether* R_3 *is the monic GCD of* f_1 *and* f_2.

```
>divide(f1,R3);
                  true
divide(f2,R3);
                  true
```

Multiplying R_3 *by* 2, *we clear fractions from the monic GCD and obtain* $g = 2x + 7$.

2.2.7 Newton's interpolation algorithm

To compute the GCD and resultant of two multivariate polynomials via modular algorithms, we recursively evaluate the input polynomials to obtain two univariate polynomials. We then apply the monic Euclidean algorithm to these univariate polynomials to determine the monic GCD and the resultant. The inversion of the evaluation homomorphism corresponds to an interpolation problem. In Algorithms PGCDNF (Algorithm 7) and MRESNF (Algorithm 10), we use Newton interpolation incrementally to reconstruct the integer coefficients of the target monic GCD and the resultant, respectively.

Theorem 2.2.11. *[18, Theorem 5.8] Let D be an arbitrary integral domain. Given distinct evaluation points $\beta_0, \ldots, \beta_d \in D$ and their corresponding values $y_0, \ldots, y_d \in D$, there exists a unique polynomial $f(x) \in F_D[x]$ (where F_D is the quotient field of D) that satisfies the following conditions*

- $\deg(f(x)) \leq d$, *and*
- $f(\beta_i) = y_i$ *for* $0 \leq i \leq d$.

Newton's interpolation algorithm (see Algorithm 5.2 in [18]) is a classical approach for constructing a polynomial such as $f(x)$ in Theorem 2.2.11. In this section, we begin by explaining Newton's interpolation for univariate polynomials and subsequently demonstrate, through an example, its extension to multivariate polynomials.

Newton's interpolation algorithm constructs a univariate polynomial of degree d using a recursive formula based on a set of distinct evaluation points $\beta_0, \ldots, \beta_d \in \mathbb{F}$. It represents the interpolating polynomial in the following form:

$$f(x) = v_0 + v_1(x - \beta_0) + v_2(x - \beta_0)(x - \beta_1) + \cdots + v_d \prod_{i=0}^{d-1}(x - \beta_i) \tag{2.14}$$

where the coefficients $v_0, v_1, \ldots, v_d \in \mathbb{F}$, known as the Newton coefficients, are initially unknown. These coefficients are determined recursively as follows:

$$v_i = \begin{cases} f(\beta_0), & i = 0, \\ \left(f(\beta_i) - \left[v_0 + \cdots + v_{i-1} \prod_{k=0}^{i-2}(\beta_i - \beta_k)\right]\right) \left(\prod_{k=0}^{i-1}(\beta_i - \beta_k)\right)^{-1}, & i = 1, \ldots, d. \end{cases} \tag{2.15}$$

In the univariate case, to interpolate a polynomial of degree at most d, Newton's algorithm requires $d+1$ evaluation points and performs $\mathcal{O}(d^2)$ arithmetic operations in $\mathbb{F}$ (see Chapter 13 of [42]). For a multivariate polynomial f in $x_1, \ldots, x_k$, the algorithm proceeds by interpolating for one variable at a time with v_i and $f(\beta_i)$, which are multivariate polynomials in the remaining variables. We illustrate the multivariate version of Newton's interpolation algorithm through the following example.

Example 2.2.18. *Let $f(x, y) \in \mathbb{Z}_5[x, y]$. Assume that $\deg(f, x) = 2$ and $\deg(f, y) = 1$, and that the values of $f(x, y)$ are given in Table 2.4 below. Our objective is to reconstruct $f(x, y)$ so that it is consistent with these data. We first fix $x = 0$ and interpolate $f(0, y)$ as a univariate polynomial in $\mathbb{Z}_5[y]$. Since $\deg(f, y) = 1$, it suffices to evaluate f at two points. We choose $\beta_0 = 0$ and $\beta_1 = 1$ so from Table 2.4, we have*

$$f(0, \beta_0) = f(0, 0) = 1, \quad \text{and} \quad f(0, \beta_1) = f(0, 1) = 4.$$

x	y	$f(x,y)$
0	0	1
0	1	4
1	0	1
1	1	5
2	0	1
2	1	8

Table 2.4: Values of $f(x,y)$, Example 2.2.18

Using Equation (2.15), the Newton interpolation coefficients are

$$v_0 = f(0,0) = 1, \quad v_1 = (f(0,1) - v_0)(1-0)^{-1} = (4-1) = 3.$$

Substituting into Newton's interpolation formula (see (2.14)), we obtain

$$f(0,y) = v_0 + v_1(y - \beta_0) = 1 + 3(y-0) = 3y + 1.$$

Thus $f(0,y)$ represents the polynomial $f(x,y)$ evaluated at $x = 0$. We now assume the general form:

$$f(x,y) = a_1(x)\, y + a_0(x), \tag{2.16}$$

where $a_1(x), a_0(x) \in \mathbb{Z}_5[x]$. Since $\deg(f,x) = 2$, determining $a_1(x)$ and $a_0(x)$ requires two additional evaluations of $f(x,y)$. Notice that we know the values of $f(1,0)$, $f(1,1)$, $f(2,0)$, and $f(2,1)$ in advance. Therefore, we repeat the process of interpolating f in y with $x = 1$ and $x = 2$ and obtain the results below.

β_i	$f(\beta_i,y) = a_1(\beta_i)\, y + a_0(\beta_i)$
0	3 $y+$ 1
1	4 $y+$ 1
2	2 $y+$ 1

Table 2.5: Values of $f(\beta_i,y)$, Example 2.2.18.

From Table 2.5, we have $a_1(0) = 3$, $a_1(1) = 4$, and $a_1(2) = 2$. Interpolating $a_1(x)$ as a univariate polynomial in $\mathbb{Z}_5[x]$ using Newton's method, we have $a_1(x) = x^2 + 3$. The constant term $a_0(x)$ is the same in all three evaluations, so $a_0(x) = 1$. Finally, substituting $a_1(x)$ and $a_0(x)$ into the general form, Equation (2.16), gives the polynomial

$$f(x,y) = a_1(x)y + a_0(x) = (x^2+3)y + 1 = x^2y + 3y + 1,$$

which is the desired polynomial in $\mathbb{Z}_5[x,y]$.

In [41], Zippel discussed the following lemma.

Lemma 2.2.12. *Let $f \in \mathbb{F}[x_1, \ldots, x_k]$, and let $d_i = \deg(f, x_i)$ for $1 \leq i \leq k$. If each $d_i \leq d$, then the total number of evaluation points required to interpolate f using Newton's interpolation algorithm is*

$$\prod_{i=1}^{k}(d_i + 1) \leq (d + 1)^k. \tag{2.17}$$

As demonstrated in Lemma 2.2.12, the number of evaluation points in $\mathbb{F}$ required for interpolating the polynomial $f \in \mathbb{F}[x_1, \ldots, x_k]$ using Newton's interpolation method is exponential in the number of variables k and the number of arithmetic operations is $\mathcal{O}((d+1)^k)$ which remains independent of the number of terms. Consequently, this algorithm is not efficient for handling sparse polynomials.

2.2.8 Brown's GCD algorithm

The most efficient approach to address the coefficient growth problem in the EA, as discussed in Remark 2.3, is to apply a modular algorithm. In Example 2.2.14, we illustrated how a modular GCD algorithm works for univariate polynomials over $\mathbb{Z}$. Brown's GCD algorithm [6] was the pioneering efficient modular GCD algorithm designed to address the multivariate scenario. It has profoundly impacted the evolution of contemporary polynomial GCD algorithms, with many of the techniques introduced becoming foundational in this field. Brown's algorithm comprises two principal sub-algorithms: **MGCD** and **PGCD**.

Let $f_1, f_2 \in \mathbb{Z}[x_1, \ldots, x_k]$ with $g = \gcd(f_1, f_2)$. Let us fix the lexicographic ordering over $\mathbb{Z}[x_1, \ldots, x_k]$ with $x_1 > x_2 > \cdots > x_k$. The **MGCD** algorithm reduces the GCD computation over $\mathbb{Z}$ to a sequence of computations over finite fields using modular homomorphisms. It chooses a sequence of primes $p_i \in \mathbb{Z}$ s.t. $p_i \nmid \mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2)$. For each such prime p_i, MGCD invokes the PGCD algorithm to compute

$$g_i = \gcd(\phi_{p_i}(f_1), \phi_{p_i}(f_2)) \in \mathbb{Z}_{p_i}[x_1, \ldots, x_k].$$

To reconstruct the integer coefficients of the target GCD g, MGCD incrementally applies the CRT to the sequence of modular gcds g_i with corresponding moduli p_i's. The stabilized image is the GCD if it divides both f_1 and f_2.

To compute g_is in $\mathbb{Z}_{p_i}[x_1, \ldots, x_k]$, Brown used a recursive algorithm called **PGCD**. It employs evaluation homomorphisms to reduce polynomials in $\mathbb{Z}_{p_i}[x_1, \ldots, x_k]$ to their corresponding polynomials in $\mathbb{Z}_{p_i}[x_1]$. Then, it applies the Euclidean algorithm to compute the GCD of univariate polynomials in $\mathbb{Z}_{p_i}[x]$. Since the computation is performed modulo a prime p_i, the polynomial coefficients are bounded by p_i; therefore, the coefficient growth problem will never occur. After computing the GCD, PGCD applies multivariate polynomial interpolation to reconstruct any lost information from the evaluation process as illustrated in Example 2.2.18.

Lemma 2.2.13. *Brown's PGCD performs $\mathcal{O}(kD^{k+1})$ arithmetic operations in $\mathbb{Z}_p$.*

In [7], the stated time complexity of PGCD is missing the factor k, which accounts for the number of variables. This appears to be an error in the complexity statement.

We will modify Brown's method to compute the monic GCD and the resultant over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. To reconstruct the rational coefficients of the output, we employ rational number reconstruction, as explained in Section 2.2.6. To illustrate Brown's algorithm, consider the following example.

Example 2.2.19. *Consider the problem of computing the GCD of $f_1, f_2 \in \mathbb{Z}[x,y]$ w.r.t. the lexicographic order $x > y$, where*

$$f_1 = (x^2 + 3xy - 15)(x + 5y + 7), \quad f_2 = (x^2 + 3xy - 15)(x + 5y).$$

By inspection, $g = \gcd(f_1, f_2) = x^2 + 3xy - 15$. We employ Brown's algorithm to compute g. Brown's algorithm uses Gauss' Lemma,

$$\gcd(f_1, f_2) = \gcd(\mathrm{pp}(f_1, x), \mathrm{pp}(f_2, x)) \cdot \gcd(\mathrm{cont}(f_1, x), \mathrm{cont}(f_2, x)),$$

so it can compute the GCD and cofactors of the content and primitive parts of the inputs separately and combine them at the end of the algorithm. In this example, $\mathrm{pp}(f_1) = f_1$ and $\mathrm{pp}(f_2) = f_2$. Moreover, both f_1 and f_2 are monic, so we do not need to worry about normalizing the images.

Let $p_1 = 11$ be the first prime that we use. We compute $g_{11} = \gcd(\phi_{11}(f_1), \phi_{11}(f_2))$. We do this by first evaluating $\phi_{11}(f_1)$ and $\phi_{11}(f_2)$ at some evaluation points for y, computing the corresponding univariate GCD in $\mathbb{Z}_{11}[x]$ using the EA, and then interpolating these images to obtain $g_{11} \in \mathbb{Z}_{11}[x,y]$. Since $g \mid f_1$, and $g \mid f_2$, and $\min(\deg(f_1, y), \deg(f_2, y)) = 2$, we know that $\deg(g, y) \leq 2$. Therefore, we require at least three evaluation points to be able to interpolate y at g. We evaluate $\phi_{11}(f_1)$ and $\phi_{11}(f_2) \in \mathbb{Z}_{11}[x,y]$ at points $y = 1$, $y = 2$, and $y = 3$. Using the EA, we compute the following gcds and then interpolate y.

$$\begin{aligned}
g_{11,1} &= \gcd(\phi_{y=1}(\phi_{11}(f_1)), \phi_{y=1}(\phi_{11}(f_2))) = 1\,x^2 + 3\,x + 7 \in \mathbb{Z}_{11}[x] \\
g_{11,2} &= \gcd(\phi_{y=2}(\phi_{11}(f_1)), \phi_{y=2}(\phi_{11}(f_2))) = 1\,x^2 + 6\,x + 7 \in \mathbb{Z}_{11}[x] \\
g_{11,3} &= \gcd(\phi_{y=3}(\phi_{11}(f_1)), \phi_{y=3}(\phi_{11}(f_2))) = 1\,x^2 + 9\,x + 7 \in \mathbb{Z}_{11}[x] \\
&\quad (\textit{Interpolate } y) \\
g_{11} &= 1\,x^2 + 3y\,x + 7 \in \mathbb{Z}_{11}[x]
\end{aligned}$$

Notice that $g_{11} \mid \phi_{11}(f_1)$ and $g_{11} \mid \phi_{11}(f_2)$. We assume that,

$$g_{11} = g \mod 11.$$

We repeat this process for a new prime $p = 7$. By evaluating $\phi_7(f_1)$ and $\phi_7(f_2) \in \mathbb{Z}_7[x,y]$ at $y = 1$, we obtain

$$\begin{aligned}
\phi_{y=1}(\phi_7(f_1)) &= (x^2 + 3x + 6)(x + 5), \textit{ and} \\
\phi_{y=1}(\phi_7(f_2)) &= (x^2 + 3x + 6)(x + 5) \in \mathbb{Z}_7[x],
\end{aligned}$$

so

$$g_{7,1} = \gcd(\phi_{y=1}(\phi_7(f_1)), \phi_{y=1}(\phi_7(f_2))) = (x^2 + 3x + 6)(x + 5) \in \mathbb{Z}_7[x].$$

Since $\mathrm{lm}(g_{7,1}) = x^3 > \mathrm{lm}(g_{11}) = x^2$, either ϕ_7 or $\phi_{y=1}$ is an unlucky homomorphism but we do not know which one. Therefore, we discard this image, and for convenience, we choose a new prime $p = 13$. We compute 3 univariate GCD images with evaluation points $y = 1$, $y = 2$, and $y = 3$ then interpolate

y in the coefficients:

$$\begin{aligned}
g_{13,1} &= \gcd(\phi_{y=1}(\phi_{13}(f_1)), \phi_{y=1}(\phi_{13}(f_2))) = \boxed{1}\ x^2\ \boxed{+3}\ x\ \boxed{+11} \mod 13 \\
g_{13,2} &= \gcd(\phi_{y=2}(\phi_{13}(f_1)), \phi_{y=2}(\phi_{13}(f_2))) = \boxed{1}\ x^2\ \boxed{+6}\ x\ \boxed{+11} \mod 13 \\
g_{13,3} &= \gcd(\phi_{y=3}(\phi_{13}(f_1)), \phi_{y=3}(\phi_{13}(f_2))) = \boxed{1}\ x^2\ \boxed{+9}\ x\ \boxed{+11} \mod 13 \\
&\quad (\textit{Interpolate } y) \\
g_{13} &= \boxed{1}\ x^2\ \boxed{+3y}\ x\ \boxed{+11} \mod 13.
\end{aligned}$$

Finally, we use the CRT to combine g_{11} and g_{13} and obtain the symmetric representation

$$g = x^2 + 3xy - 15 \mod 11 \times 13. \tag{2.18}$$

Since $g \mid f_1$ and $g \mid f_2$, we conclude that $g = \gcd(f_1, f_2)$.

To terminate, in Brown's algorithm, we can construct new GCD images until the result of the CRT divides both inputs. Once $g \mid f_1$ and $g \mid f_2$, we stop and return g as the desired GCD. Since division is an expensive operation, we can postpone the division test until we are more certain that the result of the CRT is the target GCD. A method to do so is to divide g into f_1 and f_2 only after the CRT result stabilizes for one iteration. Continuing with Example 2.2.19 and employing another prime $p = 17$, we obtain

$$g_{17} = x^2 + 3xy + 2.$$

Applying CRT to g_{11}, g_{13}, and g_{17} modulo $11 \times 13 \times 17$ yields

$$g = x^2 + 3xy - 15 \pmod{11 \times 13 \times 17}. \tag{2.19}$$

Since the CRT outcome remains unchanged, that is (2.19) is equal to (2.18), a division test should only be conducted at this stage. Moreover, because g must divide both f_1 and f_2 regardless of the main variable, it follows that

$$\deg(g, x_i) \leq \min(\deg(f_1, x_i), \deg(f_2, x_i)),$$

where $1 \leq i \leq k$. This could be used as a prior bound for the number of evaluation points needed in Brown's PGCD algorithm. Let $\|f\|_\infty$ indicate the magnitude of the largest coefficient of the polynomial f and

$$B_{MGCD} = 2^{\min(\deg(f_1), \deg(f_2))} \min(\|f_1\|_\infty, \|f_2\|_\infty) \mid \gcd(\mathrm{lc}(f_1), \mathrm{lc}(f_2)) \mid .$$

In Brown's MGCD algorithm, the prior bound could be expressed as:

$$\|g\|_\infty \leq B_{MGCD}.$$

That is, if the product of the used primes in MGCD is greater than B_{MGCD}, then we employ the division test (see Section 7.4 of [18]).

As mentioned, Brown's algorithm employs the CRT to reconstruct the integer coefficients of the GCD from its modular images. However, when the input polynomials are in $\mathbb{Q}[x]$ (see Section 4.3), the CRT alone is insufficient to recover the rational coefficients of the GCD. In this case, to recon-

struct rational coefficients of the target GCD from its modular residues, we can use rational number reconstruction as described in Section 2.2.6.

As demonstrated in Example 2.2.19, the modular algorithm transforms a complicated problem into a sequence of simpler ones by performing arithmetic in a smaller domain. However, this comes at the cost of potential information loss, which can lead to unexpected failures. This includes unlucky primes, unlucky evaluation points, and leading coefficient reconstruction. We elaborate on these difficulties below.

Bad and Unlucky primes and evaluation points

Definition 2.2.9. *Suppose $f_1, f_2 \in \mathbb{Z}[x_2, \ldots, x_k][x_1]$ with $g = \gcd(f_1, f_2)$. A prime p is said to be an lc-bad prime if $\operatorname{lm}(\phi_p(g)) < \operatorname{lm}(g)$. Similarly, an evaluation point $(\beta_1, \ldots, \beta_{k-1}) \in \mathbb{Z}_p^{k-1}$ is an lc-bad evaluation point if $\deg(g(\beta_1, \ldots, \beta_{k-1})(x_1), x_1) < \deg(g, x_1)$.*

In Brown's modular algorithm, to successfully reconstruct the target GCD from its images modulo several primes, lc-bad primes and lc-bad evaluation points are avoided. This ensures that Lemma 2.2.9 can be used to identify the unlucky primes and evaluation points defined below.

Definition 2.2.10. *Suppose $f_1, f_2 \in \mathbb{Z}[x_1, \ldots, x_k]$. Let $g = \gcd(f_1, f_2)$, and $h_1 = f_1/g$, and $h_2 = f_2/g$ be the cofactors of f_1 and f_2, respectively. A prime p is said to be unlucky if*

$$\gcd(\phi_p(h_1), \phi_p(h_2)) \neq 1.$$

Definition 2.2.11. *Suppose $f_1, f_2 \in \mathbb{Z}_p[x_1, \ldots, x_k]$, where p is a prime. Let $g = \gcd(f_1, f_2)$, and $h_1 = f_1/g$, and $h_2 = f_2/g$ be the cofactors of f_1 and f_2, respectively. An evaluation point $x_i = \beta_i \in [0, p)$ is an unlucky evaluation point if*

$$\gcd(\phi_{x_i=\beta_i}(h_1), \phi_{x_i=\beta_i}(h_2)) \neq 1.$$

Example 2.2.20. *Let*

$$\begin{aligned} f_1 &= (x_2 + 2) \cdot (x_1 + 1) \\ f_2 &= (x_2 + 2) \cdot (x_1 + 2x_2 + 10), \end{aligned}$$

be two polynomials in $\mathbb{Z}_{11}[x_1, x_2]$ listed in lexicographic order with $x_1 > x_2$. By inspection,

$$g = \gcd(f_1, f_2) = x_2 + 2.$$

In this example, $x_2 = 9$ is an lc-bad evaluation point, since $\phi_{x_2=9}(\operatorname{lc}(f_2)(x_1)) = 0 \mod 11$. Let

$$h_1 = \frac{f_1}{g} = x_1 + 1 \quad \text{and } h_2 = \frac{f_2}{g} = x_1 + 2x_2 + 10.$$

Then $x_2 = 1$ is an unlucky evaluation point because

$$\begin{aligned} &\phi_{x_2=1}(h_1) = x_1 + 1 \text{ and} \\ &\phi_{x_2=1}(h_2) = x_1 + 1 \in \mathbb{Z}_{11}[x_1], \end{aligned}$$

so $\gcd(\phi_{x_2=1}(h_1), \phi_{x_2=1}(h_2)) = x_1 + 1 \neq 1$.

Example 2.2.21. *Let*

$$f_1 = (5x_1 + x_2)(x_1 + 2x_2 + x_3) \text{ and}$$
$$f_2 = (5x_1 + x_2)(x_1 + 9x_2 + x_3)$$

be polynomials in $\mathbb{Z}[x_1, x_2, x_3]$ *listed in lexicographic order with* $x_1 > x_2 > x_3$. *By inspection* $g = \gcd(f_1, f_2) = 5x_1 + x_2$. *In this example,* $p = 5$ *is an lc-bad prime since* $\operatorname{lm}(\phi_5(g)) = x_2 < \operatorname{lm}(g) = x_1$. *Let*

$$h_1 = \frac{f_1}{g} = (x_1 + 2x_2 + x_3) \text{ and}$$
$$h_2 = \frac{f_2}{g} = (x_1 + 9x_2 + x_3).$$

Then $p = 7$ *is an unlucky prime since*

$$\phi_7(h_1) = (x_1 + 2x_2 + x_3),$$
$$\phi_7(h_2) = (x_1 + 2x_2 + x_3) \in \mathbb{Z}_7[x_1, x_2, x_3],$$

and $\gcd(\phi_7(h_1), \phi_7(h_2)) \neq 1$.

Let $g = \gcd(f_1, f_2)$. In Brown's algorithm, the images of unlucky primes and evaluation points must be avoided because it is essential for the images of the GCD, calculated modulo p, to be able to divide $g \mod p$ for the accurate reconstruction of g (see Lemma 2.2.9). While we can rule out lc-bad primes and evaluation points in advance, it is not feasible to detect unlucky primes and evaluation points before they are used. As noted following Lemma 2.2.9, Brown suggests that if, during the computation of the GCD, an image emerges with a greater degree in the main variable than the other images, it must be considered unlucky and, therefore, must be ignored. The likelihood of a prime or evaluation point being unlucky can be reduced by selecting sufficiently large primes and performing evaluations at random points (refer to Theorems 7.5 and 7.6 in [18]).

Leading coefficient problem

In Example 2.2.19, the input polynomials are monic in the main variable x; therefore, the GCD is monic. When this is not the case, special attention must be paid to reconstructing the leading coefficient of the target GCD properly at each recursive stage. Similarly, in our modular algorithm, Algorithm 8, we compute the monic GCD of the modular images. Therefore, we lose information about the leading coefficient of the target GCD. We call this problem the leading coefficient problem. We explain the leading coefficient problem and its solution through the following example.

Example 2.2.22. *Let* $f_1 = ((y+1)x+1)\cdot(yx+1)$, *and* $f_2 = ((y+1)x+1)\cdot(yx+y+1) \in \mathbb{Z}_{11}[y][x]$ *with*

$$g = \gcd(f_1, f_2) = xy + x + 1.$$

First, we replace f_1 *and* f_2 *with their primitive parts, which in this case are equal to the original polynomials. Then, we evaluate* $f_1(x, y)$ *and* $f_2(x, y)$ *at a random evaluation point* $y = 9$ *to obtain*

$$\phi_{y=9}(f_1) = f_1(x, y = 9) = 2(x + 5)(x + 10)$$
$$\phi_{y=9}(f_2) = f_2(x, y = 9) = 2(x + 6)(x + 10),$$

with monic GCD $g_1 = \gcd(\phi_{y=9}(f_1), \phi_{y=9}(f_2)) = x + 10 \in \mathbb{Z}_{11}[x]$*. How can we recover the actual GCD* $g \in \mathbb{Z}_{11}[y][x]$ *from its monic images, such as* g_1*? To solve this problem, we must recover the leading coefficient of GCD by evaluating the polynomial*

$$c = \gcd(\mathrm{lc}(f_1, x), \mathrm{lc}(f_2, x)) = y \cdot (y + 1) \in \mathbb{Z}_{11}[y]$$

at the point $y = 9$ *to get* $c_1 = c(9) \mod 11 = 2$*. Then we multiply* g_1 *by* c_1 *to obtain the first GCD image*

$$h_1 = c_1 \cdot g_1 = 2x + 9.$$

We set $prod = y - 9$*. To obtain another GCD image, we choose a new appropriate evaluation point in* $[0, 11)$ *randomly. Let us choose* $y = 7$*. Repeating the previous process for* $y = 7$*, we have* $c_2 = c(7) \mod 11 = 1$ *and* $g_2 = \gcd(\phi_{y=7}(f_1), \phi_{y=7}(f_2)) = x + 7 \in \mathbb{Z}_{11}[x]$ *so*

$$h_2 = c_2 \cdot g_2 = x + 7.$$

Now, we can interpolate y *from images* $H = [h_1, h_2]$ *at points* $q = [9, 7]$ *to get*

$$G = (6y + 3)x + y.$$

We set $prod = (y - 9)(y - 7)$*. We still need more evaluation points to construct the target GCD. Let* $y = 6$ *be the next evaluation point. Repeating the above process for* $y = 6$*, we have* $c_3 = 9$ *and* $g_3 = \gcd(\phi_{y=6}(f_1), \phi_{y=6}(f_2)) = x + 8 \in \mathbb{Z}_{11}[x]$ *so*

$$h_3 = c_3 \cdot g_3 = 9x + 6.$$

Then we set $prod = (y - 9)(y - 7)(y - 6)$*. You can see the summary of the above computation in the following table. Please note that we will interpolate the red and blue coefficients of* h_i*'s. Interpolating*

$y = \beta_i$	$f_{1_i} = f_1(x, \beta_i)$	$f_{2_i} = f_2(x, \beta_i)$	$g_i = \gcd(f_{1_i}, f_{2_i}) \mod 11$	$c_i = c(\beta_i)$	$h_i = c_i \cdot g_i$
$\beta_1 = 9$	$2(x + 5)(x + 10)$	$2(x + 6)(x + 10)$	$x + 10$	2	$2x + 9$
$\beta_2 = 7$	$(x + 7)(x + 8)$	$(x + 7)(x + 9)$	$x + 7$	1	$1x + 7$
$\beta_3 = 6$	$9(x + 2)(x + 8)$	$9(x + 3)(x + 8)$	$x + 8$	9	$9x + 6$

y from images $H = [h_1, h_2, h_3]$ *at points* $q = [9, 7, 6]$*, we end up*

$$G = y(y + 1)x + y = y \cdot g.$$

Finally, we will remove the factor y from G and find the actual GCD by computing $pp(G) = g$*. Our PGCDNF algorithm, Algorithm 7, performs the division test once* $\deg(prod, y) > \deg(G, y) + 1$*. We elaborate on this later in Chapter 4.*

Let d be an upper bound on the degree of each variable in the target GCD g and let k represent the number of variables. Brown's algorithm performs dense interpolation of g, for example, by using Newton's interpolation method. This process requires $\mathcal{O}((d+1)^{k-1})$ evaluation points (see Lemma 2.2.12).

2.3 Failure probability of modular GCD algorithms over $\mathbb{Z}$

We have seen that the modular GCD algorithm in $\mathbb{Z}[x]$ may encounter unlucky or lc-bad primes (see Example 2.2.21), and that Brown's modular GCD algorithm in $\mathbb{Z}_p[x_2][x_1]$ may encounter unlucky or lc-bad evaluation points (see Example 2.2.20). If most of the used primes or evaluation points are unlucky, the modular GCD algorithm may take a long time to terminate. Nevertheless, if we use sufficiently large primes (in our algorithms, we use 31-bit primes), such unlucky primes and evaluation points occur rarely. To characterize them, we use the Sylvester resultant, formally defined in Chapter 5.

The probabilistic analyses for our algorithms in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$, presented in Chapter 6, are complicated. In this section, we sketch the corresponding analyses for $\mathbb{Z}[x]$ and $\mathbb{Z}_p[x_2][x_1]$ to illustrate the general approach that will be used in Chapter 6 to compute the probability of encountering an unlucky prime or an unlucky evaluation point.

2.3.1 Unlucky primes

Let $f_1, f_2 \in \mathbb{Z}[x]$ and $g = \gcd(f_1, f_2)$. Write

$$f_1 = g\,h_1, \quad f_2 = g\,h_2,$$

so that h_1 and h_2 are the cofactors of f_1 and f_2, respectively. Let $\deg(f_1) = n$ and $\deg(f_2) = m$. For a prime p, let $\phi_p : \mathbb{Z}[x] \to \mathbb{Z}_p[x]$ denote the reduction modulo p homomorphism. According to Definition 2.2.10, a prime p is called unlucky if

$$\gcd(\phi_p(h_1), \phi_p(h_2)) \neq 1. \tag{2.20}$$

Let $\mathrm{res}(f_1, f_2, x) \in \mathbb{Z}$ denote the Sylvester resultant of f_1 and f_2 (see Definitions 5.2.1 and 5.2.2). By Theorem 5.2.3 in Chapter 5, we have

$$\gcd(f_1, f_2) \neq 1 \iff \mathrm{res}(f_1, f_2, x) = 0. \tag{2.21}$$

The modular GCD algorithms do not use lc-bad primes. That is, they compute g modulo a prime p, where $p \nmid \mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2)$. Since $\deg(\phi_p(f_1)) = \deg(f_1)$ and $\deg(\phi_p(f_2)) = \deg(f_2)$, from Remark 5.2 in Chapter 5, we have

$$\mathrm{res}(\phi_p(f_1), \phi_p(f_2), x) = \phi_p(\mathrm{res}(f_1, f_2, x)). \tag{2.22}$$

Since $p \nmid \mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2)$, we have $p \nmid \mathrm{lc}(h_1) \cdot \mathrm{lc}(h_2)$. Therefore,

$$\underbrace{\gcd(\phi_p(h_1), \phi_p(h_2))}_{\text{Equation (2.20)}} \neq 1 \underset{\text{From (2.21)}}{\iff} \mathrm{res}(\phi_p(h_1), \phi_p(h_2), x) = 0$$
$$\underset{\text{From (2.22)}}{\iff} \phi_p(\mathrm{res}(h_1, h_2, x)) = 0$$
$$\iff p \mid |\mathrm{res}(h_1, h_2, x)|.$$

Let $R = |\ \mathrm{res}(h_1, h_2, x)\ | \in \mathbb{Z}$. As established above, the unlucky primes are precisely those primes that divide R. Let $\mathbb{P}_b = \{\text{all } b - \text{bit primes}\}$. If $p \in \mathbb{P}_{31}$, then by definition $p \in (2^{30}, 2^{31})$. Since $|\mathbb{P}_{31}| = 50{,}697{,}537$, we will show that the probability that p is a divisor of R is small. Let $p_1 \ldots p_t$ be the 31-bit prime factors of R, where $2^{30} < p_i < 2^{31}$, for $1 \leq i \leq t$. To bound the number of 31−bit prime factors of R, t, we set

$$(2^{30})^t < \prod_{i=1}^{t} p_i \leq R.$$

Thus, $\log_{2^{30}}(2^{30})^t < R$ so

$$t < \log_{2^{30}} R \Rightarrow t \leq \lfloor \log_{2^{30}} R \rfloor.$$

Therefore, if $p \in \mathbb{P}_{31}$, then

$$\begin{aligned}\Pr[p \text{ is an unlucky prime}] = \Pr[p \mid R] = \frac{t}{|\ \mathbb{P}_{31}\ |} &\leq \frac{\lfloor \log_{2^{30}} R \rfloor}{|\ \mathbb{P}_{31}\ |} \\ &\leq \frac{\lfloor \frac{\log_2 R}{\log_2 2^{30}} \rfloor}{|\ \mathbb{P}_{31}\ |} \leq \frac{\frac{\lfloor \log_2 R \rfloor}{30}}{50,697,537} \\ &\leq \frac{\lfloor \log_2 R \rfloor}{30 \times 50,697,537}. \end{aligned} \tag{2.23}$$

Remark 2.6. *Maple represents algebraic numbers using recden, a library for the recursive dense representation of polynomials. The recden package benefits from fast routines implemented in C for primes* $< 2^{31.5}$. *In our implementation, we use* 31*-bit primes to avoid integer overflow during computations carried out by these C routines.*

Example 2.3.1. *Let*

$$f_1 = (x+1)(x^3 + 7x^2 + 5x - 6) \text{ and}$$
$$f_2 = (x+1)(2x^2 + 11x)$$

be two polynomials in $\mathbb{Z}[x]$. *By inspection,* $g = \gcd(f_1, f_2) = x + 1$, *and the cofactors of* f_1 *and* f_2 *are* $h_1 = x^3 + 7x^2 + 5x - 6$ *and* $h_2 = 2x^2 + 11x$, *respectively. Computing the resultant of the cofactors gives*

$$R = |\ \mathrm{res}(h_1, h_2, x)\ | = 570 = -2 \times 3 \times 5 \times 19.$$

Hence, the only unlucky primes are 2, 3, 5, *and* 19. *As we see,* R *has no prime factors with* 31 *bits.*

Remark 2.7. *The failure probability bound in (2.23) is pessimistic because even if R has hundreds of digits, it most likely will have no* 31*-bit prime divisors.*

In our analysis, we need to bound $R = |\ \mathrm{res}(h_1, h_2, x)\ |$. To do so, we apply Hadamard's bound (see Theorem 6.2.1 in Chapter 6). From Theorem 6.2.1, if A is an $n \times n$ integer matrix, then

$$|\ \det(A)\ | \leq \prod_{j=1}^{n} \sqrt{\sum_{i=1}^{n} A_{ij}^2}.$$

Example 2.3.2. *Applying Hadamard's bound to the Sylvester matrix of h_1 and h_2 in Example 2.3.1,*

$$S = \mathrm{sylv}(h_1, h_2, x) = \begin{bmatrix} 1 & 7 & 5 & -6 & 0 \\ 0 & 1 & 7 & 5 & -6 \\ 2 & 11 & 0 & 0 & 0 \\ 0 & 2 & 11 & 0 & 0 \\ 0 & 0 & 2 & 11 & 0 \end{bmatrix},$$

we obtain

$$R = |\ \det(S)\ | \leq \prod_{j=1}^{5} \sqrt{\sum_{i=1}^{5} S_{ij}^2} = 210\sqrt{25870} \approx 33776.72274.$$

Therefore, from (2.23), we have

$$\Pr[p \text{ is an unlucky prime}] = \Pr[p \mid R] \leq \frac{\lfloor \log_2 R \rfloor}{30 \times 50,697,537} = \frac{15}{30 \times 50,697,537} = 9.8 \times 10^{-9}. \quad (2.24)$$

To obtain a bound on R, we need a bound on the coefficients of h_1 and h_2. For this purpose, we use Definition 2.3.1.

Definition 2.3.1. *Let $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \in \mathbb{Z}[x]$. The height of f is*

$$\|f\|_\infty = \max_{i=0}^{n} |a_i|.$$

The challenge of using the bound in (2.23) is that the cofactors h_1 and h_2 are not known in advance. To bound R in (2.23) independently of h_1 and h_2, we need to bound $\|h_1\|_\infty$ and $\|h_2\|_\infty$ in terms of the heights of the inputs, $\|f_1\|_\infty$ and $\|f_2\|_\infty$. However, as the following example shows, $\|h_1\|_\infty$ and $\|h_2\|_\infty$ may be larger than $\|f_1\|_\infty$ and $\|f_2\|_\infty$.

Example 2.3.3. *Let $h = x^2 + 1000x + 1$ and $g = x - 1$ be polynomials in $\mathbb{Z}[x]$, and set*

$$f = g \cdot h = x^3 + 999x^2 - 999x - 1.$$

Then $\|f\|_\infty = 999 < \|h\|_\infty = 1000$.

We can bound $\|h_1\|_\infty$ and $\|h_2\|_\infty$ in terms of the heights of the inputs, using Mignotte's bound, stated in the theorem below.

Theorem 2.3.1. *[35, Corollary 6.33, Mignotte's bound] Let f and h be two polynomials in $\mathbb{Z}[x]$ with $\deg(f) = n \geq 1$ and $\deg(h) = k$. If $h \mid f$, then*

$$\|h\|_\infty \leq 2^k (n+1)^{\frac{1}{2}} \|f\|_\infty.$$

From Theorem 2.3.1, we have

$$\|h_1\|_\infty \leq 2^{\deg(h_1)} (n+1)^{\frac{1}{2}} \|f_1\|_\infty \text{ and}$$
$$\|h_2\|_\infty \leq 2^{\deg(h_2)} (m+1)^{\frac{1}{2}} \|f_2\|_\infty.$$

Since $n = \deg(f_1) = \deg(h_1) + \deg(g)$ and $m = \deg(f_2) = \deg(h_2) + \deg(g)$, we have $\deg(h_1) \leq n$ and $\deg(h_2) \leq m$. Therefore,

$$\|h_1\|_\infty \leq 2^n (n+1)^{\frac{1}{2}} \|f_1\|_\infty \text{ and} \tag{2.25}$$
$$\|h_2\|_\infty \leq 2^m (m+1)^{\frac{1}{2}} \|f_2\|_\infty. \tag{2.26}$$

The Sylvester matrix $S = \text{sylv}(h_1, h_2, x)$ is a $(\deg(h_1) + \deg(h_2)) \times (\deg(h_1) + \deg(h_2))$ matrix with $\deg(h_2)$ rows of coefficients of h_1, each of them bounded by $\|h_1\|_\infty$. Similarly, S has $\deg(h_1)$ rows of coefficients of h_2 each of them bounded by $\|h_2\|_\infty$. Applying Hadamard's bound to S, we have

$$\begin{aligned}
|\det(S)| &\leq \underbrace{\prod_{j=1}^{\deg(h_1)+\deg(h_2)} \sqrt{\sum_{i=1}^{\deg(h_2)} \|h_1\|_\infty^2 + \sum_{i=1}^{\deg(h_1)} \|h_2\|_\infty^2}}_{\text{Hadamard's bound (Theorem 6.2.1)}} \\
&= \left(\sqrt{\underbrace{\deg(h_2)}_{\deg(h_2)\leq m} \|h_1\|_\infty^2 + \underbrace{\deg(h_1)}_{\deg(h_1)\leq n} \|h_2\|_\infty^2} \right)^{\deg(h_1)+\deg(h_2)} \\
&\leq \left(\sqrt{m \underbrace{(2^{2n}(n+1)\|f_1\|_\infty^2)}_{\text{Mignotte's bound (2.25)}} + n \underbrace{(2^{2m}(m+1)\|f_2\|_\infty^2)}_{\text{Mignotte's bound (2.26)}}} \right)^{n+m}.
\end{aligned} \tag{2.27}$$

From (2.27) and (2.23), if $p \in \mathbb{P}_{31}$, we have

$$\begin{aligned}
\Pr[p \text{ is an unlucky prime}] = \Pr[p \mid R] &\leq \frac{\lfloor \log_{2^{30}} R \rfloor}{|\mathbb{P}_{31}|} \\
&= \frac{\lfloor \log_{2^{30}}(|\det(S)|) \rfloor}{|\mathbb{P}_{31}|} \\
&\leq \frac{\lfloor \log_2 \left((\sqrt{m(2^{2n}(n+1)\|f_1\|_\infty^2) + n(2^{2m}(m+1)\|f_2\|_\infty^2)})^{n+m} \right) \rfloor}{30 \times |\mathbb{P}_{31}|} \\
&\leq \frac{\lfloor \frac{n+m}{2} \log_2 \left(m(2^{2n}(n+1)\|f_1\|_\infty^2) + n(2^{2m}(m+1)\|f_2\|_\infty^2) \right) \rfloor}{30 \times |\mathbb{P}_{31}|}.
\end{aligned}$$

Example 2.3.4. *Consider the polynomials from Example 2.3.1:*

$$f_1 = (x^3 + 7x^2 + 5x - 6)(x + 1) = x^4 + 8x^3 + 12x^2 - x - 6 \text{ and}$$
$$f_2 = (2x^2 + 11x)(x + 1) = 2x^3 + 13x^2 + 11x.$$

Here $n = \deg(f_1) = 4$, $m = \deg(f_2) = 3$, $\|f_1\|_\infty = 12$, *and* $\|f_2\|_\infty = 13$. *Let* p *be a prime chosen at random from* $\mathbb{P}_{31}$. *Then*

$$\Pr[p \text{ is an unlucky prime}] \leq \frac{\lfloor \frac{n+m}{2} \log_2 \left(m(2^{2n}(n+1)\|f_1\|_\infty^2) + n(2^{2m}(m+1)\|f_2\|_\infty^2)\right) \rfloor}{30\times \mid \mathbb{P}_{31} \mid}$$
$$= \frac{38}{30 \times 50,697,537} \approx 2.5 \times 10^{-8}.$$

Although this bound, 2.5×10^{-8}, *is larger than the one obtained in Example 2.3.2 in (2.24), it has the advantage of not depending on the unknown cofactors* h_1 *and* h_2.

2.3.2 Unlucky evaluation points

Let p be a prime and $f_1, f_2 \in \mathbb{Z}_p[x_1][x_2]$, $g = \gcd(f_1, f_2)$, and h_1 and h_2 be the cofactors of f_1 and f_2, respectively. Brown's modular GCD algorithm computes $\gcd(f_1(x_1, \beta), f_2(x_1, \beta))$ for randomly chosen evaluation points $\beta \in [0, p)$. Recall that $\phi_{x_2=\beta} : \mathbb{Z}_p[x_1][x_2] \longrightarrow \mathbb{Z}_p[x_1]$ is a homomorphism s.t. $\phi_{x_2=\beta}(f) = f(x_1, \beta)$. Furthermore, from Definition 2.2.11, $\beta \in \mathbb{Z}_p$ is an unlucky evaluation point if

$$\gcd(\phi_{x_2=\beta}(h_1), \phi_{x_2=\beta}(h_2)) \neq 1.$$

Let $\text{res}(f_1, f_2, x_1) \in \mathbb{Z}_p[x_2]$ be the Sylvester resultant of f_1 and f_2 w.r.t. x_1. Similar to Equation (6.30), if f_1, f_2 are primitive, then

$$\gcd(f_1, f_2) \neq 1 \iff \text{res}(f_1, f_2, x_1) = 0. \tag{2.28}$$

Furthermore, if $\text{lc}(f_1)(\beta) \neq 0$ and $\text{lc}(f_2)(\beta) \neq 0$, similar to (2.22), we have

$$\text{res}(\phi_{x_2=\beta}(f_1), \phi_{x_2=\beta}(f_2), x_1) = \phi_{x_2=\beta}(\text{res}(f_1, f_2, x_1)). \tag{2.29}$$

Therefore, for the cofactors h_1 and h_2, we have

$$\begin{aligned}\gcd(\phi_{x_2=\beta}(h_1), \phi_{x_2=\beta}(h_2)) \neq 1 &\underset{\text{From (2.28)}}{\iff} \text{res}(\phi_{x_2=\beta}(h_1), \phi_{x_2=\beta}(h_2), x_1) = 0 \\ &\underset{\text{From (2.29)}}{\iff} \phi_{x_2=\beta}(\text{res}(h_1, h_2, x_1)) = 0 \\ &\iff \text{res}(h_1, h_2, x_1)(\beta) = 0.\end{aligned}$$

Accordingly, the unlucky evaluation points are precisely the roots of $R = \text{res}(h_1, h_2, x_1) \in \mathbb{Z}_p[x_2]$. Brown's algorithm restricts β so $\text{lc}(f_1, x_1)(\beta) \neq 0$ and $\text{lc}(f_2, x_1)(\beta) \neq 0$. Thus, there are at most

$$P = p - (\deg(\text{lc}(f_1, x_1)) + \deg(\text{lc}(f_2, x_1))) \tag{2.30}$$

valid choices for β. Since a polynomial of degree d over a field can have at most d roots, the probability that β is unlucky is given by

$$\Pr[\beta \in [0,p) \text{ is an unlucky point}] = \Pr[R(\beta)=0] \leq \frac{\deg(R)}{P}. \tag{2.31}$$

To bound (2.31), we need a bound for $\deg(R)$. We illustrate the process of bounding (2.31) through the following example.

Example 2.3.5. *Let $f_1 = g \cdot h_1$ and $f_2 = g \cdot h_2$ be two polynomials in $\mathbb{Z}_{101}[x_1, x_2]$, where*

$$\begin{aligned} g &= x_1 + 3x_2 + 1, \\ h_1 &= 7x_1^2 + 5x_1x_2 + 3x_2^2 + 2, \text{ and} \\ h_2 &= x_1^2 + 5x_2^2 + 7x_1x_2^2 + 3. \end{aligned}$$

The Sylvester matrix of h_1 and h_2 w.r.t. x_1 is

$$S = \operatorname{sylv}(h_1, h_2, x_1) = \begin{pmatrix} 7 & 5x_2 & 3x_2^2+2 & 0 \\ 0 & 7 & 5x_2 & 3x_2^2+2 \\ 1 & 7x_2^2 & 5x_2^2+3 & 0 \\ 0 & 1 & 7x_2^2 & 5x_2^2+3 \end{pmatrix}, \quad \text{with}$$

$$R = \operatorname{res}(h_1, h_2, x_1) = \det(S) = 19x_2^6 - 17x_2^5 + 17x_2^4 + 3x_2^3 - 22x_2^2 - 43. \tag{2.32}$$

Therefore,

$$\deg(R, x_2) = 6 \leq \sum_{j=1}^{6} \max_{i=1}^{6} \deg(S_{i,j}, x_2) = 6.$$

Since $\operatorname{lc}(f_1, x_1) = 7$ and $\operatorname{lc}(f_2, x_1) = 1$, we have $\deg(\operatorname{lc}(f_1, x_1)) = \deg(\operatorname{lc}(f_2, x_1)) = 0$. Therefore,

$$P = p - \Big(\deg(\operatorname{lc}(f_1, x_1)) + \deg(\operatorname{lc}(f_2, x_1))\Big) = 101$$

and the probability that a randomly chosen $\beta \in [0, p)$ is unlucky is bounded by

$$\Pr[R(\beta) = 0] \leq \frac{\deg(R)}{P} = \frac{6}{101}.$$

However, the actual resultant R has one root $x_2 = 30$ in $\mathbb{Z}_{101}$, since

$$R = 19\,(x_2^3 + 52x_2^2 + 30x_2 + 34)\,(x_2 + 71)\,(x_2^2 + 9x_2 + 89) \in \mathbb{Z}_{101}[x_2],$$

where $x_2 = 30$ is the root of the only linear factor $x_2 + 71$. Hence, the bound of 6 is pessimistic.

Similar to the previous section, the bound in (2.31) depends on the unknown cofactors h_1 and h_2. Let $n = \deg(f_1, x_1)$, $m = \deg(f_2, x_1)$, and $d_x = \max(\deg(f_1, x_2), \deg(f_2, x_2))$. Since $\deg(h_i, x_j) \leq \deg(f_i, x_j)$ for $1 \leq j, i \leq 2$, we obtain

$$\begin{aligned} \deg(R) = \deg(\operatorname{res}(h_1, h_2, x_1)) = \deg(\det(S)) &\leq (\deg(h_1, x_1) + \deg(h_2, x_1)) \max(\deg(h_1, x_2), \deg(h_2, x_2)) \\ &\leq (n+m)d_x. \end{aligned}$$

Therefore,

$$\Pr[\beta \in [0,p) \text{ is an unlucky point}] = \Pr[R(\beta)=0] \leq \frac{\deg(R)}{P} \leq \frac{d_x(n+m)}{P},$$

where P is defined as in (2.30). Let f_1 and f_2 be the polynomials in Example 2.3.5,

$$\begin{aligned} f_1 &= h_1 \cdot g = 7x_1^3 + 26x_1^2x_2 + 7x_1^2 + 18x_1x_2^2 + 5x_1x_2 + 2x_1 + 9x_2^3 + 3x_2^2 + 6x_2 + 2, \text{ and} \\ f_2 &= h_2 \cdot g = x_1^3 + 7x_1^2x_2^2 + 3x_1^2x_2 + x_1^2 + 21x_1x_2^3 + 12x_1x_2^2 + 3x_1 + 15x_2^3 + 5x_2^2 + 9x_2 + 3, \end{aligned}$$

with $n = \deg(f_1, x_1) = 3$, $m = \deg(f_2, x_1) = 3$, and $d_x = \max(\deg(f_1, x_2), \deg(f_2, x_2)) = 3$. Therefore,

$$\begin{aligned} \Pr[\beta \in [0,p) \text{ is an unlucky point}] = \Pr[R(\beta)=0] &\leq \frac{\deg(R)}{p - \Big(\deg(\mathrm{lc}(f_1, x_1)) + \deg(\mathrm{lc}(f_2, x_1))\Big)} \\ &\leq \frac{d_x(n+m)}{p - \Big(\deg(\mathrm{lc}(f_1, x_1)) + \deg(\mathrm{lc}(f_2, x_1))\Big)} = \frac{18}{93}. \end{aligned}$$

Chapter 3

Algebraic number fields

3.1 Summary of contributions

This chapter is based on our paper in the Proceedings of CASC '23 [2]. After introducing the necessary preliminaries in Section 3.2, we describe how to do arithmetic in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ in Section 3.3. Let $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1}\alpha_i$ where $C_1, \ldots, C_{n-1} \in \mathbb{Z}$. In Section 3.4, we prove Theorem 3.4.3, which provides a criterion for determining whether γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. We also present Algorithm 4, which can be executed over either $\mathbb{F} = \mathbb{Q}$ or $\mathbb{F} = \mathbb{Z}_p$, where p is a prime. When $\mathbb{F} = \mathbb{Q}$, the algorithm decides whether a given γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. If so, it returns the minimal polynomial $M(z) \in \mathbb{Q}[z]$ of γ, together with the coefficient (change-of-basis) matrix for the powers of γ and its inverse. To avoid coefficient swell in our modular algorithms MGCDNF and MRESNF, we run Algorithm 4 over $\mathbb{F} = \mathbb{Z}_p$. In this modular setting, it is likely that the generator polynomial $M(z)$ is reducible in $\mathbb{Z}_p[z]$. In this case, the quotient ring $\mathbb{Z}_p[z]/\langle M(z)\rangle$ is not a field and therefore contains zero divisors.

In Section 3.4.3, we introduce a ring isomorphism that maps elements of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ modulo a prime p to the elements in $\mathbb{Z}_p[z]/\langle M(z)\rangle$. For univariate polynomials, the computations of the monic GCD in MGCDNF and the resultant in MRESNF are carried out over the ring $\mathbb{Z}_p[z]/\langle M(z)\rangle$. Finally, in Section 3.5, we compute the number of zero divisors of $\mathbb{Z}_p[z]/\langle M(z)\rangle$.

3.2 Preliminaries

First, we explain relevant details and notations.

Definition 3.2.1. *A complex number $\alpha \in \mathbb{C}$ is called an **algebraic number** if there exists a non-zero polynomial $M(z) \in \mathbb{Q}[z]$ s.t. $M(\alpha) = 0$. In this case, we say that α is **algebraic** over $\mathbb{Q}$.*

Let $\alpha \in \mathbb{C}$ be an algebraic number. Define the set

$$I(\alpha) = \{f(z) \in \mathbb{Q}[z] \mid f(\alpha) = 0\},$$

the set of all polynomials in $\mathbb{Q}[z]$ that vanish at α. Since α is an algebraic number, there exists a non-zero polynomial $g(z) \in \mathbb{Q}[z]$ s.t. $g(\alpha) = 0$. Hence, $I(\alpha) \neq \langle 0\rangle$. It is straightforward to verify that $I(\alpha)$ is an ideal of the ring $\mathbb{Q}[z]$. Because $\mathbb{Q}$ is a field, the ring $\mathbb{Q}[z]$ is a principal ideal domain (PID).

Therefore, there exists $M(z) \in \mathbb{Q}[z]$ s.t.

$$I(\alpha) = \langle M(z) \rangle. \tag{3.1}$$

We claim that $M(z)$ is unique up to multiplication by a scalar. Suppose there is another polynomial $M_1(z)$ s.t. $I(\alpha) = \langle M_1(z) \rangle$. Then

$$\begin{aligned} &\langle M(z) \rangle = \langle M_1(z) \rangle \Rightarrow M(z) \mid M_1(z) \text{ and } M_1(z) \mid M(z) \Rightarrow \\ &M(z) = u \cdot M_1(z) \quad \text{s.t } u \text{ is a unit in } \mathbb{Q}[z] \Rightarrow \\ &M(z) = u \cdot M_1(z) \quad \text{s.t } u \in \mathbb{Q}. \end{aligned}$$

This implies that the generator $M(z)$ is unique up to multiplication by a non-zero scalar in $\mathbb{Q}$. Consequently, we may always choose $M(z)$ to be the unique monic polynomial that generates $I(\alpha)$ (See Section 5.1 of [1] for more details).

Definition 3.2.2. *Let α be an algebraic number. The **minimal polynomial** of α is the monic polynomial $M(z) \in \mathbb{Q}[z]$ s.t. $I(\alpha) = \langle M(z) \rangle$.*

We state the following Lemma without proof (See [1, Theorem 5.1.1]).

Lemma 3.2.1. *Let α be an algebraic number and $M(z) \in \mathbb{Q}[z]$ be a non-zero monic polynomial s.t. $M(\alpha) = 0$. Then $M(z)$ is irreducible over $\mathbb{Q}$ if and only if $I(\alpha) = \langle M(z) \rangle$.*

Example 3.2.1. *Let $\alpha = \sqrt{2} \in \mathbb{C}$. Then α is an algebraic number over $\mathbb{Q}$, since it is a root of the polynomial $M(z) = z^2 - 2 \in \mathbb{Q}[z]$. Furthermore, $M(z)$ is non-zero, monic, and irreducible over $\mathbb{Q}$. Thus, from Definition 3.2.2 and Lemma 3.2.1, $M(z)$ is the minimal polynomial of α.*

Theorem 3.2.2. *[1, Theorem 5.5.1] Let $\alpha \in \mathbb{C}$ be an algebraic number with minimal polynomial $M(z) \in \mathbb{Q}[z]$, and let $d = \deg(M)$. Then*

$$\mathbb{Q}(\alpha) = \{c_0 + c_1\alpha + \cdots + c_{d-1}\alpha^{d-1} \mid c_0, \ldots, c_{d-1} \in \mathbb{Q}\}.$$

Theorem 3.2.2 implies that $\mathbb{Q}(\alpha)$ is a d-dimensional vector space over $\mathbb{Q}$, with basis $\{1, \alpha, \ldots, \alpha^{d-1}\}$.

Definition 3.2.3. *Let $\alpha \in \mathbb{C}$ be an algebraic number with minimal polynomial $M(z) \in \mathbb{Q}[z]$. The **degree** of $\mathbb{Q}(\alpha)$ over $\mathbb{Q}$ is defined as*

$$[\mathbb{Q}(\alpha) : \mathbb{Q}] = \deg(M(z)).$$

Example 3.2.2. *Continuing from Example 3.2.1, since the minimal polynomial of $\sqrt{2}$ is $M(z) = z^2 - 2$, we have*

$$[\mathbb{Q}(\sqrt{2}) : \mathbb{Q}] = \deg(M) = 2.$$

Let α be an algebraic number with minimal polynomial $M(z)$. Then there exists a field isomorphism

$$\mathbb{Q}(\alpha) \cong \mathbb{Q}[z]/\langle M(z) \rangle.$$

This correspondence allows us to perform arithmetic over $\mathbb{Q}(\alpha)$ using the quotient ring representation (see Section 3.3). Now, we are well-equipped to define an algebraic number field obtained from the field of rational numbers $\mathbb{Q}$ by adjoining a finite number of algebraic numbers.

Definition 3.2.4. *Let $\alpha_1, \ldots, \alpha_n$ be algebraic numbers. The **algebraic number field** $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is the smallest subfield of $\mathbb{C}$ containing both $\mathbb{Q}$ and $\alpha_1, \ldots, \alpha_n$.*

The field $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is also called a multiple extension of $\mathbb{Q}$. Furthermore, $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ can be constructed via a succession of n single extensions, as follows

$$\begin{aligned}\mathbb{Q}(\alpha_1, \alpha_2) &= \mathbb{Q}(\alpha_1)(\alpha_2)\\ \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) &= \mathbb{Q}(\alpha_1, \alpha_2)(\alpha_3)\\ &\ldots\\ \mathbb{Q}(\alpha_1, \ldots, \alpha_n) &= \mathbb{Q}(\alpha_1, \ldots, \alpha_{n-1})(\alpha_n).\end{aligned}$$

Let

(i) $M_1 = M_1(z_1)$ be the minimal polynomial of α_1 over $\mathbb{Q}$,

(ii) $M_i = M_i(z_i)$ be the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \leq i \leq n$, and let

(iii) $d_i = \deg(M_i, z_i)$ for $1 \leq i \leq n$.

The algebraic number field $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is a $\mathbb{Q}$-vector space with a basis

$$B = \{\prod_{i=1}^{n} \alpha_i^{e_i} \;\; 0 \leq e_i < d_i\}.$$

The cardinality of B is $|B| = \prod_{i=1}^{n} d_i$, which is equal to the degree of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, $[\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$. In Section 3.3, we describe how to carry out arithmetic in the field $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$.

Definition 3.2.5. *Let α be an algebraic number with the minimal polynomial $M(z) \in \mathbb{Q}[z]$. The **conjugates** of α over $\mathbb{Q}$ are defined as the distinct roots of $M(z)$ in $\mathbb{C}$.*

Example 3.2.3. *Continuing from Example 3.2.1, the number $-\sqrt{2}$ is the conjugate of $\sqrt{2}$ over $\mathbb{Q}$ since both are roots of the minimal polynomial $M(z) = z^2 - 2 \in \mathbb{Q}[z]$.*

Theorem 3.2.3. *[1, Theorem 5.2.1] If $\alpha \in \mathbb{C}$ is an algebraic number with minimal polynomial $M(z) \in \mathbb{Q}[z]$, then its conjugates over $\mathbb{Q}$ are distinct.*

A fundamental result in algebraic number theory (see Theorem 3.2.5) shows that every multiple extension $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is, in fact, a single extension of the form $\mathbb{Q}(\gamma)$ for some appropriate algebraic number γ. We begin by proving this result in the case of two extensions, i.e. the field $\mathbb{Q}(\alpha, \beta)$.

Theorem 3.2.4. *[1, Theorem 5.6.1] Let $\alpha, \beta \in \mathbb{C}$ be algebraic over $\mathbb{Q}$. Then there exists an algebraic number $\gamma \in \mathbb{C}$ s.t.*

$$\mathbb{Q}(\alpha, \beta) = \mathbb{Q}(\gamma).$$

Proof. Let $p(x)$ be the minimal polynomial of α over $\mathbb{Q}$. Denote the roots of $p(x)$ by $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_m$. Let $q(x) = (x - \beta_1) \cdots (x - \beta_n) \in \mathbb{Q}[x]$, where $\beta_1 = \beta, \beta_2, \ldots, \beta_n$ are the conjugates of β over $\mathbb{Q}$. By Theorem 3.2.3, these conjugates are all distinct. Define the set

$$S = \{\frac{\alpha_r - \alpha_s}{\beta_t - \beta_u} \ : \ r, s = 1, \ldots, m \ ; \ t, u = 1, \ldots, n \ ; \ t \neq u\}.$$

Choose $C \in \mathbb{Q}$ s.t. $C \notin S$. We claim that $\gamma = \alpha + C\beta$ is a primitive element, i.e., $\mathbb{Q}(\gamma) = \mathbb{Q}(\alpha, \beta)$. Since $\mathbb{Q}(\alpha, \beta)$ is a field containing both α and β, it follows that $\gamma = \alpha + C\beta \in \mathbb{Q}(\alpha, \beta)$. On the other hand, $\mathbb{Q}(\gamma)$ is the smallest field containing γ and $\mathbb{Q}$. Thus,

$$\mathbb{Q}(\gamma) \subseteq \mathbb{Q}(\alpha, \beta).$$

To prove the reverse inclusion, note that the values $\alpha_i + C\beta_j$ are pairwise distinct for all $1 \leq i \leq m$, $1 \leq j \leq n$, due to the choice of $C \notin S$. Let $\gamma = \alpha + C\beta = \alpha_1 + C\beta_1$, and consider the polynomial

$$p_1(x) = p(\gamma - Cx) \in \mathbb{Q}(\gamma)[x].$$

Since $p(\alpha) = 0$, it follows

$$p_1(\beta) = p(\gamma - C\beta) = p(\alpha) = 0.$$

Moreover, by assumption $q(\beta) = 0$, so β is a common root of both $p_1(x)$ and $q(x)$. Suppose λ is another common root of $p_1(x)$ and $q(x)$. Then $\lambda = \beta_j$ for some $1 \leq j \leq n$, and

$$p_1(\lambda) = p_1(\beta_j) = p(\gamma - C\beta_j) = 0.$$

Thus, $\gamma - C\beta_j = \alpha_k$ for some $1 \leq k \leq m$, which implies that

$$\alpha_k + C\beta_j = \gamma = \alpha_1 + C\beta_1,$$

and hence

$$C = \frac{\alpha_1 - \alpha_k}{\beta_j - \beta_1},$$

contradicting the assumption that $C \notin S$. Therefore, β is the only common root of $p_1(x)$ and $q(x)$. Let $h(x)$ be the minimal polynomial of β over $\mathbb{Q}(\gamma)$. Then $h(x)|p_1(x)$ and $h(x)|q(x)$. Given that these polynomials share exactly one common root, we conclude $\deg(h) = 1$, so $h(x) = x + \sigma$ for some $\sigma \in \mathbb{Q}(\gamma)$. We have $h(\beta) = \beta + \sigma = 0$ so $\beta = -\sigma \in \mathbb{Q}(\gamma)$. Therefore, $\alpha = \gamma - C\beta \in \mathbb{Q}(\gamma)$ and we have

$$\mathbb{Q}(\alpha, \beta) \subseteq \mathbb{Q}(\gamma).$$

Combining both inclusions, we conclude $\mathbb{Q}(\alpha, \beta) = \mathbb{Q}(\gamma)$. □

We can generalize Theorem 3.2.4 for any finite set of algebraic numbers.

Theorem 3.2.5. *Let $\alpha_1, \ldots, \alpha_n$ be algebraic over $\mathbb{Q}$. Then there exists an algebraic number $\gamma \in \mathbb{C}$ s.t.*

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = \mathbb{Q}(\gamma).$$

Proof. The case $n = 1$ is trivial. Suppose $n \geq 2$. By Theorem 3.2.4, there exists an algebraic number $\beta_2 \in \mathbb{C}$ s.t.

$$\mathbb{Q}(\alpha_1, \alpha_2) = \mathbb{Q}(\beta_2).$$

Again, by applying Theorem 3.2.4, there exists an algebraic number $\beta_3 \in \mathbb{C}$ s.t.

$$\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) = \mathbb{Q}(\beta_2, \alpha_3) = \mathbb{Q}(\beta_3).$$

Continuing this process, we obtain a finite sequence $\beta_2, \ldots, \beta_n$ of algebraic numbers s.t.

$$\begin{aligned}
\mathbb{Q}(\alpha_1, \ldots, \alpha_n) &= \mathbb{Q}(\beta_2, \alpha_3, \ldots, \alpha_n) \\
&= \mathbb{Q}(\beta_3, \alpha_4, \ldots, \alpha_n) \\
&= \ldots \\
&= \mathbb{Q}(\beta_{n-1}, \alpha_n) \\
&= \mathbb{Q}(\beta_n)
\end{aligned}$$

□

Now we are prepared to define one of the most important tools used in our modular algorithms: a primitive element.

Definition 3.2.6. *Let $\alpha_1, \ldots, \alpha_n$ be algebraic numbers. An element $\gamma \in \mathbb{C}$ is called a* ***primitive element*** *for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ if*

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = \mathbb{Q}(\gamma).$$

Computation over the algebraic number field with multiple extensions $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ involves handling n algebraic numbers and their respective minimal polynomials. This results in complicated arithmetic and data representations, which slow down arithmetic in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. In contrast, working over a single extension $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, i.e., $\mathbb{Q}(\gamma) = \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, reduces the computation to univariate polynomial arithmetic modulo a single minimal polynomial (see Section 3.3 for details).

3.3 Computing over an algebraic number field

We construct the field L via a tower of field extensions as follows:

$$\begin{aligned}
L_0 &= \mathbb{Q} \\
L_i &= L_{i-1}[z_i]/\langle M_i \rangle \text{ for } i = 1, 2, \ldots, n,
\end{aligned}$$

where M_i is the minimal polynomial of α_i over L_{i-1} for $1 \leq i \leq n$. Let $L = L_n$ and $d_i = \deg(M_i, z_i)$. Then L is a $\mathbb{Q}$-vector space with a basis

$$B_L = \{\textstyle\prod_{i=1}^{n} (z_i)^{e_i} \mid 0 \leq e_i < d_i\} \tag{3.2}$$

of dimension $d = \prod_{i=1}^{n} d_i$, that is, $d = [L : \mathbb{Q}]$. This construction implies an isomorphism of fields

$$L \cong \mathbb{Q}(\alpha_1, \ldots, \alpha_n).$$

We perform computations over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ by replacing $\alpha_1, \ldots, \alpha_n$ with the corresponding variables $z_1, \ldots, z_n$ and carrying out the operations in L. In our modular algorithms, we assume that the minimal polynomials are given as follows:

- M_1 is the minimal polynomial of α_1 over $\mathbb{Q}$,
- M_2 is the minimal polynomial of α_2 over $\mathbb{Q}(\alpha_1)$,
  $\vdots$
- M_n is the minimal polynomial of α_n over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{n-1})$,

so that we can construct the field L.

Example 3.3.1. *Let*

$$\begin{aligned} &L_0 = \mathbb{Q} \\ &L_1 = L_0[z_1]/\langle M_1(z_1)\rangle, \quad \textit{where} \quad M_1 = z_1^2 - 2 \textit{ is the minimal polynomial of } \sqrt{2} \textit{ over } \mathbb{Q}, \textit{ and} \\ &L_2 = L_1[z_2]/\langle M_2(z_2)\rangle, \quad \textit{where} \quad M_2 = z_2^2 - 3 \quad \textit{is the minimal polynomial of } \sqrt{3} \textit{ over } L_1. \end{aligned}$$

A basis for L as a $\mathbb{Q}$-vector space is

$$B_L = \{1, z_1, z_2, z_1 z_2\}.$$

Furthermore, $[L : \mathbb{Q}] = \prod_{i=1}^{2} \deg(M_i, z_i) = 2 \times 2 = 4$.

Let $f \in L[x_1, \ldots, x_k]$, and let B_L be the basis for L defined in (3.2). Then each coefficient of f can be expressed as a unique linear combination of elements of B_L. In other words, if

$$f = \sum_{e_i \in \mathbb{Z}^k_{\geq 0}} a_{e_i} X^{e_i} \in L[x_1, \ldots, x_k],$$

then for each coefficient $a_{e_i} \in L$, there exist elements $b_j \in B_L$ and scalars $C_{ij} \in \mathbb{Q}$ s.t. $a_{e_i} = \sum_{j=1}^{d} C_{ij} b_j$. Therefore, the polynomial f can be written as

$$f = \sum_{e_i \in \mathbb{Z}^k_{\geq 0}} a_{e_i} X^{e_i} = \sum_{e_i \in \mathbb{Z}^k_{\geq 0}} \left(\sum_{j=1}^{d} C_{ij} b_j \right) X^{e_i}. \tag{3.3}$$

Definition 3.3.1. *Given Equation 3.3, we define the* ***coordinate vector*** *of a polynomial f w.r.t. the basis B_L as the vector $[f]_{B_L} = [v_1, \ldots, v_d]^T$ where $v_j = C_{1j} X^{e_1} + \ldots + C_{nj} X^{e_n}$.*

Example 3.3.2. *Let $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$ with a basis $B_L = \{1, z_2, z_1, z_1 z_2\}$. Let $f = 2z_1 + z_1 z_2 + 3 \in L$. Then,*

$$[f]_{B_L} = [3, 0, 2, 1]^T.$$

Definition 3.3.2. *Let $L_{\mathbb{Z}} = \mathbb{Z}[z_1, \ldots, z_n]$. For any polynomial $f \in L[x_1, \ldots, x_k]$, we have*

(i) The ***denominator*** *of f, denoted by $\operatorname{den}(f)$, is the smallest positive integer s.t.*

$$\operatorname{den}(f) \cdot f \in L_{\mathbb{Z}}[x_1, \ldots, x_k].$$

(ii) The ***associate*** *of f, denoted by $\tilde{f}$, is defined as*

$$\tilde{f} = \operatorname{den}(h) \cdot h, \quad \text{where } h = \operatorname{monic}(f).$$

(iii) The ***semi-associate*** *of f, denoted by $\check{f}$, is defined as*

$$\check{f} = s \cdot f,$$

where $s \in \mathbb{Q}_{>0}$ is the smallest positive rational number s.t. $\operatorname{den}(s \cdot f) = 1$.

Example 3.3.3. *Let $\alpha_1 = \sqrt{2}$, $\alpha_2 = \sqrt{3}$, and $f = \frac{3}{2}\alpha_1 x + \alpha_2$. Then $\operatorname{den}(f) = 2$. To compute the semi-associate of f, we determine the smallest positive rational s s.t. $\operatorname{den}(s \cdot f) = 1$. Here, $s = 2$, so $\check{f} = 3\alpha_1 x + 2\alpha_2$. To compute the associate $\tilde{f}$, we first compute $h = \operatorname{monic}(f) = x + \frac{1}{3}\alpha_1\alpha_2$. Multiplying h by $\operatorname{den}(h) = 3$ gives $\tilde{f} = 3x + \alpha_1\alpha_2$.*

To improve computational efficiency, in a preprocessing step, our modular GCD (MGCDNF) and resultant (MRESNF) algorithms clear fractions by replacing the input polynomials with their semi-associates. Since computing associates can be expensive when $\operatorname{lc}(f_1)$ and $\operatorname{lc}(f_2)$ are complicated algebraic numbers, we prefer using semi-associates to remove fractions.

For further efficiency, the MGCDNF and MRESNF algorithms compute $\gcd(f_1, f_2)$ and $\operatorname{res}(f_1, f_2)$ modulo a sequence of primes. This avoids coefficient swell in $\mathbb{Q}$ when applying the MEA. However, we must avoid using primes that divide $\operatorname{lc}(\check{f}_1)$, $\operatorname{lc}(\check{f}_2)$, or $\operatorname{lc}(\check{M}_i)$ for $1 \leq i \leq n$. Details regarding this choice of primes are discussed further in Sections 4.4.3 and 5.4.2.

Definition 3.3.3. *Let p be a prime s.t. $p \nmid \prod_{i=1}^{n} \operatorname{lc}(\check{M}_i, z_i)$. For $1 \leq i \leq n$, let $m_i = \check{M}_i \bmod p$. Define*

$$\begin{aligned} L_0 &= \mathbb{Z}_p \\ L_1 &= L_0[z_1]/\langle m_1 \rangle \quad \text{and} \\ L_i &= L_{i-1}[z_i]/\langle m_i \rangle. \end{aligned}$$

We write $L_n = L_p$.

Remark 3.1. *Equivalently, we may write*

$$L_p = \mathbb{Z}_p[z_1, \ldots, z_n]/\langle m_1, \ldots, m_n \rangle.$$

In practice, however, the recursive representation in Definition 3.3.3 is more efficient: at level i, elements are represented as univariate polynomials in z_i with coefficients in L_{i-1}, rather than as multivariate residue classes in $\mathbb{Z}_p[z_1, \ldots, z_n]/\langle m_1, \ldots, m_n \rangle$.

Let $d = \prod_{i=1}^{n} \deg(\check{M}_i, z_i)$, then L_p is a finite ring with p^d elements.

To perform computations L_p, Maple and Pari build the recursive tower of extensions, as presented in Definition 3.3.3. In Maple, the library used for this purpose is called `recden`. In this setting, fewer extensions lead to faster computations. In the following example, we illustrate how the number of extensions affects both the running time and memory allocation.

Example 3.3.4. *Let*

$$\begin{aligned} L_0 &= \mathbb{Z}_p \\ L_1 &= L_0[z_1]/\langle M_1(z_1)\rangle \quad s.t. \quad d_1 = \deg(M_1, z_1) = 2 \\ L_2 &= L_1[z_2]/\langle M_2(z_2)\rangle \quad s.t. \quad d_2 = \deg(M_2, z_2) = 4, \end{aligned}$$

where $M_1(z_1)$ is the minimal polynomial of α_1 over L_0 and $M_2(z_2)$ is the minimal polynomial of α_2 over L_1. In this example, $L_p = L_2$ and $d = d_1 \cdot d_2 = 8$. Let A and B be two elements in L_p. Then A and B can be written as

$$\begin{aligned} A &= \sum_{i=0}^{3} a_i(z_1) z_2^i, \quad \text{where} \quad a_i(z_1) \in L_1,\ \deg(a_i, z_1) \le d_1 - 1 = 1 \quad \text{and} \\ B &= \sum_{i=0}^{3} b_i(z_1) z_2^i, \quad \text{where} \quad b_i(z_1) \in L_1,\ \deg(b_i, z_1) \le d_1 - 1 = 1. \end{aligned}$$

Since $d_1 = 2$, each $a_i(z_1)$ and $b_i(z_1)$ has the form

$$a_i(z_1) = a_{i0} + a_{i1} z_1, \qquad b_i(z_1) = b_{i0} + b_{i1} z_1,$$

where $a_{ij}, b_{ij} \in \mathbb{Z}_p$ for $i = 1, 2, 3$ and $j = 0, 1$. In the recursive dense representation, using the `recden` library in Maple, A and B are stored as nested lists:

$$\begin{aligned} A &= [\underbrace{[a_{00}, a_{01}]}_{z_2^0}, \underbrace{[a_{10}, a_{11}]}_{z_2^1}, \underbrace{[a_{20}, a_{21}]}_{z_2^2}, \underbrace{[a_{30}, a_{31}]}_{z_2^3}] \\ B &= [\underbrace{[b_{00}, b_{01}]}_{z_2^0}, \underbrace{[b_{10}, b_{11}]}_{z_2^1}, \underbrace{[b_{20}, b_{21}]}_{z_2^2}, \underbrace{[b_{30}, b_{31}]}_{z_2^3}]. \end{aligned}$$

Now consider multiplying A and B. First, multiplication in L_1 requires multiplying two polynomials of degree at most $d_1 - 1 = 1$ in z_1. This requires $d_1^2 = 2^2 = 4$ multiplications in $\mathbb{Z}_p$ before reduction. The unreduced product has degree at most $2d_1 - 2$ in z_1. Reducing it modulo M_1 has at most

$$2d_1 - 2 + d_1 + 1 = d_1 - 1$$

division steps, and each step uses at most d_1 multiplications in $\mathbb{Z}_p$. Hence reduction modulo M_1 costs $d_1(d_1 - 1)$ additional multiplications in $\mathbb{Z}_p$. Therefore, one multiplication in L_1 requires at most

$$d_1^2 + d_1(d_1 - 1) = 2d_1^2 - d_1 = 6$$

multiplications in $\mathbb{Z}_p$. At the next level, multiplying A and B as polynomials in z_2 requires $d_2^2 = 4^2 = 16$ coefficient multiplications in L_1. The unreduced product has degree at most $2d_2 - 2$ in z_2. Reducing it

modulo M_2 requires at most $d_2 - 1$ division steps, and each step uses at most d_2 multiplications in L_1. Hence the reduction modulo M_2 costs

$$d_2(d_2 - 1)$$

multiplications in L_1. Thus multiplication and reduction at the z_2-level cost

$$d_2^2 + d_2(d_2 - 1) = 2d_2^2 - d_2 = 28$$

multiplications in L_1. Since each multiplication in L_1 costs at most $2d_1^2 - d_1 = 6$ multiplications in $\mathbb{Z}_p$, multiplying A and B in the tower representation costs at most

$$(2d_1^2 - d_1)(2d_2^2 - d_2) = 6 \cdot 28 = 168$$

multiplications in $\mathbb{Z}_p$. The tower representation also needs a large amount of temporary storage. A naive implementation creates intermediate products $a_i \times b_j$ in L_1 and then reduces each of them modulo M_1. There are $d_2^2 = 16$ such coefficient products. Thus, even before the final reduction modulo M_2, the computation may create $2d_2^2 = 2 \cdot 16 = 32$ intermediate L_1-objects. Measured in $\mathbb{Z}_p$-coefficients, the unreduced L_1-products require up to

$$16(2d_1 - 1) = 16 \cdot 3 = 48$$

coefficient slots, while their reduced remainders require

$$16d_1 = 16 \cdot 2 = 32$$

coefficient slots. Thus, these lower-level temporaries alone account for up to $48+32 = 80$ $\mathbb{Z}_p$-coefficient slots, not counting the additional temporary storage needed for the reduction modulo M_2, and the overhead from repeated recursive function calls.

Now compare this with computation in a single extension $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle \cong L_p$, where $\deg(M, z) = 8$. Let $A_\gamma, B_\gamma \in \bar{L}_p$, where

$$A_\gamma = \sum_{i=0}^{7} a_i z^i, \qquad B_\gamma = \sum_{i=0}^{7} b_i z^i,$$

with $a_i, b_i \in \mathbb{Z}_p$. The following denotes the data representation of A_γ and B_γ in recden:

$$A_\gamma = [\underbrace{a_0}_{z^0}, \underbrace{a_1}_{z^1}, \underbrace{a_2}_{z^2}, \underbrace{a_3}_{z^3}, \underbrace{a_4}_{z^4}, \underbrace{a_5}_{z^5}, \underbrace{a_6}_{z^6}, \underbrace{a_7}_{z^7}]$$
$$B_\gamma = [\underbrace{b_0}_{z^0}, \underbrace{b_1}_{z^1}, \underbrace{b_2}_{z^2}, \underbrace{b_3}_{z^3}, \underbrace{b_4}_{z^4}, \underbrace{b_5}_{z^5}, \underbrace{b_6}_{z^6}, \underbrace{b_7}_{z^7}].$$

Multiplying A_γ and B_γ costs $d^2 = 8^2 = 64$ multiplications in $\mathbb{Z}_p$. The unreduced product has degree at most $2d - 2$. Reducing it modulo M requires at most

$$2d - 2 - d + 1 = d - 1$$

division steps, and each step uses at most d multiplications in $\mathbb{Z}_p$. Hence the reduction costs $d(d-1) = 8 \cdot 7 = 56$ multiplications in $\mathbb{Z}_p$. Therefore, multiplying A_γ by B_γ requires at most

$$d^2 + d(d-1) = 2d^2 - d = 2 \cdot 8^2 - 8 = 120$$

multiplications in $\mathbb{Z}_p$ which is less than the cost of multiplying A and B in $\bar{L}_p$ which is 168. The memory usage is also smaller in the single-extension representation. The product $A_\gamma \times B_\gamma$ and its reduction modulo M are stored in two lists, compared with the 32 intermediate lists in L_p. The unreduced product requires at most $2d - 1 = 15$ slots to store coefficients in $\mathbb{Z}_p$. The reduced remainder requires $d = 8$ slots to store coefficients in $\mathbb{Z}_p$. Thus, storing both requires at most $15 + 8 = 23$ $\mathbb{Z}_p$-coefficient slots. Therefore, in this example, multiplication in L_p requires at most 168 base-field multiplications, whereas multiplication in $\bar{L}_p$ requires only 120. Moreover, multiplication in L_p creates many more intermediate objects. Already at the lower level, it may require up to 80 $\mathbb{Z}_p$-coefficient slots for temporary products and remainder modulo M_1, whereas multiplication in L_p requires only 23 $\mathbb{Z}_p$-coefficient slots for the corresponding arithmetic.

The above example shows that, if $L_p \cong \bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, then mapping elements of L_p to $\bar{L}_p$ can reduce both the number of base-field multiplications and the amount of temporary memory used. Consequently, GCD and resultant computations is faster over the single-extension representation $\bar{L}_p$. In Section 3.4.3, we present the isomorphism $\phi_\gamma : L_p \to \bar{L}_p$.

3.4 Converting $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to a single extension $\mathbb{Q}(\gamma)$

The primary goal of this section is to find a primitive element γ for the field $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and to compute its minimal polynomial $M(z)$. To avoid expression swell, we perform the computation modulo a prime p. This reduction allows us to construct the quotient ring

$$\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle,$$

where $M(z)$ is the characteristic polynomial of γ in $\mathbb{Z}_p[z]$. Note that $M(z)$ is often reducible over $\mathbb{Z}_p[z]$, which implies that the ring $\bar{L}_p$ may contain zero divisors. As part of our modular approach, univariate GCD and resultant computations are performed over the ring $\bar{L}_p$. Once $\bar{L}_p$ has been constructed, we define a ring isomorphism

$$\phi_\gamma : L_p \longrightarrow \bar{L}_p$$

which maps elements of L_p to their representations in the simple extension $\bar{L}_p$. The inverse map ϕ_γ^{-1} is then used to lift elements back from $\bar{L}_p$ to L_p.

3.4.1 Computing a primitive element and its minimal polynomial

By Theorem 3.2.5, there exist rational numbers $C_1, \ldots, C_{n-1}$ s.t. $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1} \cdot \alpha_i$ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, i.e., $\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = \mathbb{Q}(\gamma)$. It is worth mentioning that, in our proposed method (Algorithm 4), we aim to find integer values $C_1, \ldots, C_{n-1} \in \mathbb{Z}$ satisfying this condition, rather than rational ones. Thus, finding a primitive element is equivalent to finding appropriate integers $C_1, \ldots, C_{n-1}$. We describe two strategies for choosing these constants:

(i) Let S denote the set of all inappropriate constants, as described in the proof of Theorem 3.2.4. Since S is finite, any choice of $C_1, \ldots, C_{n-1}$ not in S yields a primitive element. However, constructing S is computationally expensive. For this reason, we do not adopt this approach in practice.

(ii) Our practical approach is to choose $C_1, \ldots, C_{n-1}$ at random and then test whether the resulting element

$$\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1} \cdot \alpha_i$$

is a primitive element. If it is not, we discard the current constants and repeat the procedure with new values.

To verify whether a randomly chosen γ is primitive, we use one of the following two methods: Gröbner Bases or Linear Algebra.

- **Using Gröbner Bases**
  Let $M_1(z_1), \ldots, M_n(z_n)$ be the minimal polynomials of $\alpha_1, \ldots, \alpha_n$. Define

$$\delta(z, z_1, \ldots, z_n) = z - z_1 - \sum_{i=2}^{n} C_{i-1} \cdot z_i,$$

  where $C_i \in \mathbb{Z}$ for $1 \leq i \leq n-1$ are chosen randomly. Let

$$G = \{\delta, M_1(z_1), \ldots, M_n(z_n)\}.$$

  By [11, Theorem 3, Chapter 2] and [11, proposition 4, Chapter 2], the set G is a Gröbner basis for the ideal $\langle M_1(z_1), \ldots, M_n(z_n), \delta \rangle$ w.r.t. the lexicographic order with $z > z_1 > \ldots > z_n$. We then apply the FGLM algorithm [16] to convert G to a Gröbner basis w.r.t. the lexicographic order with $z_1 > \ldots > z_n > z$. Denote the resulting basis by $G' = \{G'_1, \ldots, G'_{n+1}\}$, where $G'_1 = M(z) \in \mathbb{Q}[z]$, $G'_2 \in \mathbb{Q}[z, z_n], \ldots, G'_{n+1} \in \mathbb{Q}[z, z_1]$. If $M(z)$ is irreducible over $\mathbb{Q}$ and $M(\gamma) = 0$, then γ is a primitive element and $M(z)$ is its minimal polynomial.

Example 3.4.1. *Refer to Example 3.3.1, where* $M_1(z_1) = z_1^2 - 2$ *and* $M_2(z_2) = z_2^2 - 3$. *Take* $C_1 = 1$, *so that*

$$\gamma = \sqrt{2} + \sqrt{3}, \quad \textit{and} \quad G = \{\, z - (z_1 + z_2),\ z_1^2 - 2,\ z_2^2 - 3 \,\}.$$

The leading monomials of the polynomials in G *w.r.t. the lexicographic order, with* $z > z_1 > z_2$, *namely* $\{z, z_1^2, z_2^2\}$, *are relatively prime, and hence* G *is a Gröbner basis. Applying the FGLM algorithm, we obtain*

$$G' = \{\, z^4 - 10z^2 + 1,\ z^3 + 2z_2 - 11z,\ -z^3 + 2z_1 + 9z \,\}.$$

Then

$$G' \cap \mathbb{Q}[z] = \{\, z^4 - 10z^2 + 1 \,\}.$$

Since this polynomial is irreducible over $\mathbb{Q}$ *and vanishes at* γ*, we conclude that* $\gamma = \sqrt{2} + \sqrt{3}$ *is a primitive element, and its minimal polynomial is* $M(z) = z^4 - 10z^2 + 1$*. Therefore, we have the isomprphism* $\phi_\gamma : \mathbb{Q}[z_1, z_2]/\langle M_1(z_1), M_2(z_2)\rangle \to \mathbb{Q}[z]/\langle M(z)\rangle$ *s.t.*

$$\phi_\gamma(z_1) = \frac{z^3}{2} - \frac{9z}{2} \text{ and}$$
$$\phi_\gamma(z_2) = \frac{-z^3}{2} + \frac{11z}{2}.$$

In this approach, although the Gröbner basis G can be constructed at no cost, transforming G into G' using the FGLM algorithm requires $O(nd^3)$ arithmetic operations in $\mathbb{Q}$ [16].

- **Using Resultants** Using resultants is another way to verify whether a randomly chosen γ is primitive. We explain this method through the following example.

  Example 3.4.2. *Let* $L = \mathbb{Q}[z_1, z_2]/\langle M_1(z_1), M_2(z_2)\rangle$*, where* $M_1(z_1) = z_1^2 - 2$ *and* $M_2(z_2) = z_2^2 - 3$*. Define*

  $$\delta(z, z_1, z_2) = z - z_1 - C_1 z_2,$$

  where $C_1 \in \mathbb{Z}$ *is chosen randomly. Then a candidate for the minimal polynomial of* γ *can be obtained by eliminating* z_2 *and then* z_1 *using resultants. If the resulting polynomial has degree equal to* $d = [L : \mathbb{Q}]$*, then* $\gamma = z_1 + C_1 z_2$ *is a primitive element of* L*, and*

  $$M(z) = \text{res}(\text{res}(\delta, M_2(z_2), z_2), M_1(z_1), z_1) \in \mathbb{Q}[z]$$

  is the minimal polynomial of it. For example, let $C_1 = 1$ *so* $\gamma = z_1 + z_2$*. Then*

  $$M(z) = \text{res}(\text{res}(z - z_1 - z_2, M_2(z_2), z_2), M_1(z_1), z_1) = z^4 - 10z^2 + 1 \in \mathbb{Q}[z]$$

  is the minimal polynomial of γ*. Although resultants can be used to test whether an element such as* $\gamma = z_1 + C_1 z_2$ *is a primitive element of* L*, this method does not directly provide the homomorphism* $\phi_\gamma : \mathbb{Q}[z_1, z_2]/\langle M_1(z_1), M_2(z_2)\rangle \to \mathbb{Q}[z]/\langle M(z)\rangle$*, unlike the method in Example 3.4.1.*

- **Using Linear Algebra**
  After choosing random integers $C_1, \ldots, C_{n-1}$, define $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1} \cdot \alpha_i$. Let
  - $B_\gamma = \{1, \gamma, \gamma^2, \ldots, \gamma^{d-1}\}$ be a basis of $\mathbb{Q}(\gamma)$,
  - $B_{\bar{L}} = \{1, z, z^2, \ldots, z^{d-1}\}$ be a basis of $\bar{L} = \mathbb{Q}[z]/\langle M(z)\rangle$ where $\mathbb{Q}(\gamma) \cong \mathbb{Q}[z]/\langle M(z)\rangle$, and
  - $B_L = \{\prod_{i=1}^{n}(z_i)^{e_i} |\ 0 \le e_i < d_i\}$ be a basis of L.

  Construct the $d \times d$ change-of-basis matrix A from B_γ to B_L, where the ith column of A is the coordinate vector $[\gamma^{i-1}]_{B_L}$. If $\det(A) = 0$, then the current constants $C_1, \ldots, C_{n-1}$ are unsuitable and must be discarded. We then choose a new set of constants and repeat the process. Otherwise, γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ (see Theorem 3.4.3), and the next step is to determine its minimal polynomial $M(z) \in \mathbb{Q}[z]$.

Let $M(z) = z^d + \sum_{i=1}^{d} q_i z^{i-1}$, where q_i are unknown coefficients. Since $M(\gamma) = 0$, we obtain

$$M(\gamma) = 0 \Rightarrow \gamma^d + \sum_{i=1}^{d} q_i \gamma^{i-1} = 0 \Rightarrow$$

$$\sum_{i=1}^{d} q_i \gamma^{i-1} = -\gamma^d \Rightarrow$$

$$\underbrace{\begin{bmatrix} \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \gamma & \gamma^2 & \dots & \gamma^{d-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{d-1} \end{bmatrix}}_{q} = \underbrace{\begin{bmatrix} \vdots \\ -\gamma^d \\ \vdots \end{bmatrix}}_{b}.$$

Therefore, in order to find $M(z)$, we need to solve the system $A \cdot q = b$. According to Theorem 3.4.3 and Corollary 3.1, $M(z)$ is the minimal polynomial of γ if and only if $\det(A) \neq 0$.

Applying the linear algebra method requires $\mathcal{O}(d^3)$ arithmetic operations to construct the matrix A, an additional $\mathcal{O}(d^3)$ operations to compute A^{-1}, and $\mathcal{O}(d^2)$ operations to solve the linear system $A \cdot q = b$ in order to obtain $M(z)$. In total, the linear algebra method has a computational cost of $\mathcal{O}(d^3)$ arithmetic operations, which is more efficient than the Gröbner basis method with a cost of $\mathcal{O}(nd^3)$. In the following example, we illustrate how the linear algebra method operates.

Example 3.4.3. *Let* $L = \mathbb{Q}[z_1, z_2, z_3]/\langle M_1, M_2, M_3\rangle$, *where*

$$\begin{aligned} M_1 &= z_1^2 - 2 \in \mathbb{Q}[z_1], \\ M_2 &= z_2^2 - 3 \in \mathbb{Q}[z_1]/\langle M_1\rangle[z_2], \text{ and} \\ M_3 &= z_3^2 - 5z_3 - 1 \in \mathbb{Q}[z_1, z_2]/\langle M_1, M_2\rangle[z_3]. \end{aligned}$$

Our goal is to find a primitive element γ *of* $\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) \cong L$. *Let*

$$B_L = \{1, z_3, z_2, z_2z_3, z_1, z_1z_3, z_1z_2, z_1z_2z_3\}$$

be a basis of L *and* $B_\gamma = \{1, \gamma, \gamma^2, \gamma^3, \gamma^4, \gamma^5, \gamma^6, \gamma^7\}$ *be a basis of* $\mathbb{Q}(\gamma)$. *Substituting* α_1, α_2, α_3 *with* z_1, z_2, *and* z_3, *respectively, we set* $\gamma = z_1 + C_1z_2 + C_2z_3$. *Take* $C_1 = C_2 = 1$ *so*

$$\begin{aligned} \gamma^0 &= 1 \\ \gamma^1 &= z_1 + z_2 + z_3 \\ \gamma^2 &= 2z_1z_2 + 2z_1z_3 + 2z_2z_3 + 5z_3 + 6 \\ \gamma^3 &= 6z_1z_2z_3 + 15z_1z_3 + 15z_2z_3 + 14z_1 + 12z_2 + 41z_3 + 5 \\ \gamma^4 &= 60z_1z_2z_3 + 32z_1z_2 + 148z_1z_3 + 140z_2z_3 + 20z_1 + 20z_2 + 285z_3 + 105 \\ \gamma^5 &= 620z_1z_2z_3 + 100z_1z_2 + 1225z_1z_3 + 1125z_2z_3 + 349z_1 + 309z_2 + 2246z_3 + 385 \\ \gamma^6 &= 5550z_1z_2z_3 + 1278z_1z_2 + 10580z_1z_3 + 9420z_2z_3 + 1910z_1 + 1710z_2 + 17440z_3 + 3871 \\ \gamma^7 &= 49028z_1z_2z_3 + 9170z_1z_2 + 88900z_1z_3 + 77350z_2z_3 + 18285z_1 + 15847z_2 + 140491z_3 + 26390. \end{aligned}$$

The coordinate vectors of $\gamma^0, \ldots, \gamma^7$ w.r.t. the basis B_L are:

$$
\begin{aligned}
[\gamma^0]_{B_L} &= [1,0,0,0,0,0,0,0]^T \\
[\gamma^1]_{B_L} &= \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T \\
[\gamma^2]_{B_L} &= \begin{bmatrix} 6 & 5 & 0 & 2 & 0 & 2 & 2 & 0 \end{bmatrix}^T \\
[\gamma^3]_{B_L} &= \begin{bmatrix} 5 & 41 & 12 & 15 & 14 & 15 & 0 & 6 \end{bmatrix}^T \\
[\gamma^4]_{B_L} &= \begin{bmatrix} 105 & 285 & 20 & 140 & 20 & 148 & 32 & 60 \end{bmatrix}^T \\
[\gamma^5]_{B_L} &= \begin{bmatrix} 385 & 2246 & 309 & 1125 & 349 & 1225 & 100 & 620 \end{bmatrix}^T \\
[\gamma^6]_{B_L} &= \begin{bmatrix} 3871 & 17440 & 1710 & 9420 & 1910 & 10580 & 1278 & 5550 \end{bmatrix}^T \\
[\gamma^7]_{B_L} &= \begin{bmatrix} 26390 & 140491 & 15847 & 77350 & 18285 & 88900 & 9170 & 49028 \end{bmatrix}^T .
\end{aligned}
$$

The change-of-basis-Matrix from B_γ to B_L is

$$
A = \begin{bmatrix}
1 & 0 & 6 & 5 & 105 & 385 & 3871 & 26390 \\
0 & 1 & 5 & 41 & 285 & 2246 & 17440 & 140491 \\
0 & 1 & 0 & 12 & 20 & 309 & 1710 & 15847 \\
0 & 0 & 2 & 15 & 140 & 1125 & 9420 & 77350 \\
0 & 1 & 0 & 14 & 20 & 349 & 1910 & 18285 \\
0 & 0 & 2 & 15 & 148 & 1225 & 10580 & 88900 \\
0 & 0 & 2 & 0 & 32 & 100 & 1278 & 9170 \\
0 & 0 & 0 & 6 & 60 & 620 & 5550 & 49028
\end{bmatrix} .
$$

Since $\det(A) = \mid -617872752 \mid = 2^4 \times 3^3 \times 7^2 \times 17^2 \times 101 \neq 0$, *$A$ is invertible. Therefore, $\gamma = \alpha_1 + \alpha_2 + \alpha_3$ is a primitive element of $\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3)$. To compute the minimal polynomial of γ, we first compute*

$$\gamma^8 = \gamma^7 \cdot \gamma = 420560 z_1 z_2 z_3 + 83160 z_1 z_2 + 750360 z_1 z_3 + 641144 z_2 z_3 + 142800 z_1 + 122080 z_2 + 1138695 z_3 + 224602$$

with coordinate vector $[\gamma^8]_{B_L} = \begin{bmatrix} 224602 & 1138695 & 122080 & 641144 & 142800 & 750360 & 83160 & 420560 \end{bmatrix}^T$. *We compute the coefficients of $M(z)$ by solving the linear system*

$$
\begin{aligned}
q &= A^{-1} \cdot -[\gamma^8]_{B_L} \\
&= \begin{bmatrix}
1 & -\frac{4915}{6363} & \frac{520}{303} & \frac{3757}{4242} & -\frac{6005}{6363} & \frac{8153}{12726} & -\frac{109}{42} & -\frac{425}{707} \\
0 & \frac{413968}{108171} & \frac{11653}{10302} & -\frac{99795}{12019} & -\frac{50371}{12726} & -\frac{235240}{108171} & \frac{325}{357} & \frac{253006}{36057} \\
0 & \frac{762950}{108171} & -\frac{88735}{10302} & -\frac{153358}{12019} & \frac{19855}{12726} & -\frac{699560}{108171} & \frac{1495}{714} & \frac{970405}{72114} \\
0 & -\frac{563060}{108171} & \frac{49123}{10302} & \frac{259365}{24038} & \frac{5561}{12726} & \frac{602215}{216342} & -\frac{200}{357} & -\frac{315632}{36057} \\
0 & \frac{3225}{12019} & -\frac{375}{3434} & -\frac{32594}{36057} & -\frac{225}{1414} & \frac{249}{707} & -\frac{5}{42} & \frac{575}{4242} \\
0 & \frac{41428}{108171} & -\frac{3935}{10302} & -\frac{55985}{72114} & -\frac{13}{12726} & -\frac{48275}{216342} & \frac{5}{119} & \frac{23363}{36057} \\
0 & -\frac{1250}{15453} & \frac{805}{10302} & \frac{13229}{72114} & \frac{5}{1818} & \frac{667}{30906} & -\frac{1}{357} & -\frac{2945}{24038} \\
0 & \frac{500}{108171} & -\frac{23}{5151} & -\frac{115}{10302} & -\frac{1}{6363} & -\frac{5}{12726} & 0 & \frac{2}{303}
\end{bmatrix} \cdot \begin{bmatrix} -224602 \\ -1138695 \\ -122080 \\ -641144 \\ -142800 \\ -750360 \\ -83160 \\ -420560 \end{bmatrix} = \begin{bmatrix} 289 \\ -2040 \\ -2642 \\ 4220 \\ -1297 \\ -140 \\ 126 \\ -20 \end{bmatrix} .
\end{aligned}
$$

Therefore, the minimal polynomial of $\gamma = \alpha_1 + \alpha_2 + \alpha_3$ *is*

$$M(z) = z^8 - 20z^7 + 126z^6 - 140z^5 - 1297z^4 + 4220z^3 - 2642z^2 - 2040z + 289.$$

Note $\mathbb{Q}(\gamma) \cong \mathbb{Q}[z]/\langle M(z)\rangle$.

Remark 3.2. *From Example 3.4.3, we observe the following:*

- *Computations over* $\mathbb{Q}$ *often lead to significant intermediate expression swell, especially when computing* A^{-1}*. To control this growth, our modular algorithms perform computations modulo carefully selected prime numbers. We return to this topic in Section 3.4.2.*
- *Since* $\det(A) = -617872752 = -101 \cdot 2^4 \cdot 3^3 \cdot 7^2 \cdot 17^2$*, any prime divisor of this integer, that is* $p \in \{101, 2, 3, 7, 17\}$ *makes the determinant zero modulo* p*. Consequently, these primes are excluded from the MGCDNF and MRESNF algorithms. In Definition 4.4.4, we formally define these problematic primes and integers* C_1 *and* C_2*. Moreover, an upper bound for the probability that our algorithms hit them is computed in Section 6.3.2.*

Theorem 3.4.1. *[17] Let* $F \subseteq K \subseteq L$ *be fields s.t.* $[L : K]$ *and* $[K : F]$ *are finite. Then,*

$$[L : F] = [L : K] \cdot [K : F].$$

Theorem 3.4.2. *Let* $\alpha_1, \ldots, \alpha_n$ *be algebraic numbers and* $C_i \in \mathbb{Q}$ *for* $1 \leq i \leq n$*. Define* $\gamma = \sum_{i=1}^{n} C_i\alpha_i$*. Then* γ *is a primitive element of* $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ *if and only if*

$$[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}].$$

Proof. If γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, then by definition,

$$[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}].$$

Conversely, suppose $[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$. Since $\gamma \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and $\mathbb{Q}(\gamma)$ is the smallest field containing γ and $\mathbb{Q}$, we have

$$\mathbb{Q} \subseteq \mathbb{Q}(\gamma) \subseteq \mathbb{Q}(\alpha_1, \ldots, \alpha_n).$$

Applying Theorem 3.4.1, we have

$$[\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}(\gamma)][\mathbb{Q}(\gamma) : \mathbb{Q}].$$

Since we assumed that $[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$, we can simplify the above equation to obtain $[\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}(\gamma)] = 1$. Thus, $\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = \mathbb{Q}(\gamma)$, and so γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. □

Let γ be an algebraic number with the minimal polynomial $M(z)$. By Theorem 3.4.2, γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ if and only if $\deg(M(z)) = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$. We use this characterization to prove Theorem 3.4.3.

Theorem 3.4.3. *Let $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ be an algebraic number field with degree d, and let $C_1, \ldots, C_{n-1} \in \mathbb{Z}$ be chosen randomly. Define $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1}\alpha_i$, and let $B = \{b_1, \ldots, b_d\}$ be a basis for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ as a $\mathbb{Q}$-vector space. Let A be the $d \times d$ matrix whose ith column is $[\gamma^{i-1}]_B$ for $1 \leq i \leq d$. Then,*

$$\gamma \text{ is a primitive element for } \mathbb{Q}(\alpha_1, \ldots, \alpha_n) \iff \det(A) \neq 0.$$

Proof. ($\Longrightarrow$) Suppose that γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. Then

$$[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}] = d.$$

Let $B_\gamma = \{1, \gamma, \ldots, \gamma^{d-1}\}$ be a basis of $\mathbb{Q}(\gamma)$ as a $\mathbb{Q}$-vector space. Since $\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = \mathbb{Q}(\gamma)$, each element of B_γ can be expressed as a linear combination of elements of B. Hence, the $d \times d$ linear system

$$\begin{aligned} 1 &= c_{11}b_1 + c_{12}b_2 + \cdots + c_{1d}b_d \\ \gamma &= c_{21}b_1 + c_{22}b_2 + \cdots + c_{2d}b_d \\ &\vdots \\ \gamma^{d-1} &= c_{d1}b_1 + c_{d2}b_2 + \cdots + c_{dd}b_d \end{aligned}$$

has a unique solution. Define the $d \times d$ matrix D whose i-th row is $[\gamma^{i-1}]_B^T$ for $1 \leq i \leq d$. Since the system above has a unique solution, the matrix D is invertible and therefore $\det(D) \neq 0$. Because $D = A^T$, it follows that

$$\det(A) = \det(A^T) = \det(D) \neq 0.$$

($\Longleftarrow$) Conversely, suppose that $\det(A) \neq 0$. Then A is invertible, and the linear system $A \cdot q = -[\gamma^d]_B$ has a unique solution $q = [q_1, \ldots, q_d]^T$ over $\mathbb{Q}$. Define the polynomial

$$M(z) = z^d + \sum_{i=1}^{d} q_i z^{i-1}.$$

We claim that $M(z)$ is the minimal polynomial of γ. By construction, $M(z)$ is monic, has degree d, and satisfies $M(\gamma) = 0$. Thus, it remains to prove that $M(z)$ is irreducible over $\mathbb{Q}$. Assume, for contradiction, that $M(z)$ is reducible. Since $\mathbb{Q}[z]$ is a UFD, we can factor $M(z)$ as a product of monic irreducible polynomials over $\mathbb{Q}$, that is,

$$M(z) = p_1(z) \cdots p_k(z),$$

where each $p_i(z) \in \mathbb{Q}[z]$ is irreducible for $1 \leq i \leq k$. Because $M(\gamma) = 0$, there exists some $1 \leq i \leq k$ s.t. $p_i(\gamma) = 0$, which implies that $p_i(z)$ is the minimal polynomial of γ. Let $\deg(p_i(z)) = h < d$. Then $\{1, \gamma, \ldots, \gamma^{h-1}\}$ is a basis of $\mathbb{Q}(\gamma)$. Consequently,

$$\{1, \gamma, \ldots, \gamma^{d-1}\} \subseteq \text{Span}\{1, \gamma, \ldots, \gamma^{h-1}\},$$

so the set $\{1, \gamma, \ldots, \gamma^{d-1}\}$ is linearly dependent. This contradicts the assumption that $\det(A) \neq 0$. Hence, $M(z)$ must be irreducible over $\mathbb{Q}$, and therefore $M(z)$ is the minimal polynomial of γ. It follows

that

$$[\mathbb{Q}(\alpha_1,\ldots,\alpha_n):\mathbb{Q}] = [\mathbb{Q}(\gamma):\mathbb{Q}] = \deg(M(z)) = d.$$

Hence, by Theorem 3.4.2, γ is a primitive element of $\mathbb{Q}(\alpha_1,\ldots,\alpha_n)$. □

We can apply Theorem 3.4.3 to determine the minimal polynomial of the primitive element γ.

Corollary 3.1. *Under the assumptions of Theorem 3.4.3, suppose* $\det(A) \neq 0$ *and let* $q = [q_1,\ldots,q_d]^T$ *be the solution of the linear system* $A \cdot q = -[\gamma^d]_B$. *Then the polynomial*

$$M(z) = z^d + \sum_{i=1}^{d} q_i z^{i-1}$$

is the minimal polynomial of γ.

Proof. This result follows directly from the proof of Theorem 3.4.3. □

3.4.2 LAminpoly algorithm

As illustrated in Example 3.4.3, computations over $\mathbb{Q}$ lead to rapid growth in the size of the coefficients. In particular, the matrix A^{-1} contains entries given by large rational fractions. To avoid this blow-up, our modular algorithms MGCDNF and MRESNF compute a primitive element modulo a prime p. In this modular setting, the minimal polynomial $M(z)$ is often reducible over $\mathbb{Z}_p$, which leads to the presence of zero divisors in the ring $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$. In this section, our objective is to compute $M(z)$ s.t. $L_p \cong \bar{L}_p$. To this end, we apply Algorithm LAminpoly (Algorithm 4).

The LAminpoly algorithm takes as input a list of minimal polynomials $[\check{M}_1(z_1),\ldots,\check{M}_n(z_n)]$, a ground field $\mathbb{F}$ over which the computations are performed, and a candidate primitive element

$$\gamma = z_1 + C_1 z_2 + \cdots + C_{n-1} z_n,$$

where $0 \neq C_i \in \mathbb{Z}$ for $1 \leq i \leq n-1$. If γ is not a primitive element, the algorithm returns the message **FAIL**. Otherwise, it returns the generator polynomial $M(z)$ s.t. $\langle I(\alpha)\rangle = \langle M(z)\rangle$, together with the change-of-basis matrix A and its inverse A^{-1}. The LAMinpoly algorithm is a Las Vegas algorithm and can be executed over two different ground fields: $\mathbb{F} = \mathbb{Q}$ or $\mathbb{F} = \mathbb{Z}_p$.

Remark 3.3. *By Theorem 3.2.4, there are only finitely many bad choices for the nonzero integers* $C_1,\ldots,C_{n-1}$. *In Section 6.3, we analyze the probability of choosing such bad integers.*

- **LAminpoly over $\mathbb{F} = \mathbb{Q}$:** When executed over $\mathbb{F} = \mathbb{Q}$, the algorithm first constructs the change-of-basis matrix A. If $\det(A) = 0$, the algorithm returns **FAIL**. Otherwise, by Theorem 3.4.3 and Corollary 3.1, the element γ is a primitive element of $\mathbb{Q}(\alpha_1,\ldots,\alpha_n)$, so the algorithm returns the minimal polynomial $M(z)$ of γ, the matrix A, and its inverse A^{-1} (see Example 3.4.3). The correctness of the algorithm is ensured by Theorem 3.4.3, and we have the isomorphism

$$L \cong \mathbb{Q}[z]/\langle M(z)\rangle.$$

- **LAminpoly over $\mathbb{F} = \mathbb{Z}_p$:** Let us choose a prime p s.t. $p \nmid \mathrm{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i)$. Running Algorithm LAminpoly over $\mathbb{F} = \mathbb{Z}_p$, if $\det(A) = 0$, the algorithm returns FAIL. Otherwise,

the resulting characteristic polynomial $M(z)$ and matrix A can be used to construct $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ s.t.

$$L_p \cong \bar{L}_p.$$

We will describe this isomorphism in detail in Section 3.4.3.

Algorithm 4: LAminpoly

Input: A list of the minimal polynomials $[\check{M}_1(z_1), \ldots, \check{M}_n(z_n)]$, the ground field $\mathbb{F}$ over which the computation is performed, and a candidate primitive element $\gamma = z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$, where $0 \neq C_i \in \mathbb{Z}$ for $1 \leq i \leq n-1$

Output: Either a message FAIL or a polynomial $M(z) \in \mathbb{F}[z]$ of degree d s.t. $M(\gamma) = 0$, and $d \times d$ matrices A and A^{-1} over $\mathbb{F}$.

1 $B = \{ \prod_{i=1}^{n} (z_i)^{e_{ij}}\ 0 \leq e_{ij} < d_i \}$ s.t $d_i = \deg(\check{M}_i(z_i))$ // A basis for L
2 $d = \prod_{i=1}^{n} d_i$
3 Initialize A to be a $d \times d$ zero matrix over $\mathbb{F}$.
4 $g_0 = 1$
5 **for** $i = 1$ *to* d **do**
6 Set column i of A to be $[g_{i-1}]_B$
7 $g_i = \gamma \cdot g_{i-1}$
8 **if** $\det(A) = 0$ **then**
9 **return**(FAIL)
10 Compute A^{-1} and set $q = A^{-1} \cdot (-[g_d]_B)$
11 Construct the polynomial $M(z) = z^d + q_d z^{d-1} + \ldots + q_2 z + q_1$ where $q_i \in \mathbb{F}$
12 **return**($M(z)$, A, A^{-1})

In the following example, we apply the LAminpoly algorithm over the field $\mathbb{F} = \mathbb{Z}_p$ for a prime p.

Example 3.4.4. *Let L, B_L, and B_γ be as defined in Example 3.4.3. Let us choose $p = 103$, so the LAminpoly algorithm performs computations over the ground field $\mathbb{F} = \mathbb{Z}_{103}$. After reducing minimal polynomials modulo p, we have $L_p = \mathbb{Z}_{103}[z_1, z_2, z_3]/\langle m_1, m_2, m_3\rangle$, where*

$$\begin{aligned} m_1 &= \check{M}_1 \mod p = z_1^2 + 101 \\ m_2 &= \check{M}_2 \mod p = z_2^2 + 100, \text{ and} \\ m_3 &= \check{M}_3 \mod p = z_3^2 + 98 z_3 + 102. \end{aligned}$$

Let us try $C_1 = C_2 = 1$, *so* $\gamma = z_1 + C_1 z_2 + C_2 z_3 = z_1 + z_2 + z_3$. *Then we have*

$$\begin{aligned}
\gamma^0 &= 1,\\
\gamma^1 &= z_1 + z_2 + z_3,\\
\gamma^2 &= (2z_2 + 2z_1 + 5)z_3 + 2z_2 z_1 + 6,\\
\gamma^3 &= ((6z_1 + 15)z_2 + 15z_1 + 41)z_3 + 12z_2 + 14z_1 + 5,\\
\gamma^4 &= ((60z_1 + 37)z_2 + 45z_1 + 79)z_3 + (32z_1 + 20)z_2 + 20z_1 + 2,\\
\gamma^5 &= ((2z_1 + 95)z_2 + 92z_1 + 83)z_3 + 100z_2 z_1 + 40z_1 + 76,\\
\gamma^6 &= ((91z_1 + 47)z_2 + 74z_1 + 33)z_3 + (42z_1 + 62)z_2 + 56z_1 + 60,\\
\gamma^7 &= (100z_2 + 11z_1 + 102)z_3 + (3z_1 + 88)z_2 + 54z_1 + 22,\\
\gamma^8 &= ((11z_1 + 72)z_2 + 5z_1 + 30)z_3 + (39z_1 + 25)z_2 + 42z_1 + 62.
\end{aligned}$$

We construct the change-of-basis matrix similar to Example 3.4.3, so

$$A = \begin{bmatrix}
1 & 0 & 6 & 5 & 2 & 76 & 60 & 22\\
0 & 1 & 0 & 14 & 20 & 40 & 56 & 54\\
0 & 1 & 0 & 12 & 20 & 0 & 62 & 88\\
0 & 0 & 2 & 0 & 32 & 100 & 42 & 3\\
0 & 1 & 5 & 41 & 79 & 83 & 33 & 102\\
0 & 0 & 2 & 15 & 45 & 92 & 74 & 11\\
0 & 0 & 2 & 15 & 37 & 95 & 47 & 100\\
0 & 0 & 0 & 6 & 60 & 2 & 91 & 0
\end{bmatrix}$$

Since $\det(A) \bmod p = 43 \neq 0$, *the matrix* A *is invertible, and we can construct the characteristic polynomial* $M(z)$. *To this end, we need to solve the linear system* $A \cdot q = -[\gamma^8]_{B_L}$ *for* q *over* $\mathbb{Z}_p$

$$q = A^{-1} \cdot (-[\gamma^8]_{B_L}) = \begin{bmatrix}
1 & 73 & 85 & 44 & 48 & 31 & 8 & 23\\
0 & 18 & 7 & 69 & 79 & 30 & 64 & 79\\
0 & 5 & 77 & 66 & 21 & 40 & 48 & 3\\
0 & 56 & 99 & 13 & 51 & 19 & 98 & 9\\
0 & 100 & 4 & 71 & 102 & 19 & 67 & 14\\
0 & 7 & 41 & 84 & 55 & 40 & 99 & 71\\
0 & 100 & 42 & 15 & 64 & 94 & 40 & 9\\
0 & 9 & 80 & 0 & 14 & 74 & 97 & 34
\end{bmatrix} \cdot \begin{bmatrix} -62\\ -42\\ -25\\ -39\\ -30\\ -5\\ -72\\ -11 \end{bmatrix} = \begin{bmatrix} 83\\ 20\\ 36\\ 100\\ 42\\ 66\\ 23\\ 83 \end{bmatrix}.$$

Thus,

$$M(z) = z^8 + 83z^7 + 23z^6 + 66z^5 + 42z^4 + 100z^3 + 36z^2 + 20z + 83,$$

and $L_p = \mathbb{Z}_p[z_1, z_2, z_3]/\langle m_1, m_2, m_3\rangle \cong \bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$.

Remark 3.4. *From Example 3.4.4, we observe the following:*

- *By performing computations over* $\mathbb{Z}_p$ *instead of* $\mathbb{Q}$, *we effectively control expression swell. This is evident when comparing the change-of-basis matrix* A *and its inverse* A^{-1} *in Examples 3.4.3 and 3.4.4.*

- *When Algorithm 4 is executed over $\mathbb{F} = \mathbb{Z}_p$, it is likely that one or more of the polynomials m_i will be reducible over $\mathbb{Z}_p[z_1, \ldots, z_{i-1}]/\langle m_1, \ldots, m_{i-1}\rangle$; in which case, the resulting generator polynomial $M(z)$ is reducible over $\mathbb{Z}_p$. In Example 3.4.4, m_1 and m_3 are reducible over $\mathbb{Z}_{103}$:*

$$m_1 = z_1^2 - 2 = (z_1 + 65)(z_1 + 38) \text{ and}$$
$$m_3 = z_3^2 - 5z_3 - 1 = (z_3 + 8)(z_3 + 90).$$

- *In Example 3.4.4, the characteristic polynomial $M(z)$ is reducible over $\mathbb{Z}_{103}$:*

$$\begin{aligned} M(z) &= z^8 + 83z^7 + 23z^6 + 66z^5 + 42z^4 + 100z^3 + 36z^2 + 20z + 83 \\ &= (z^2 + 92z + 53)(z^2 + z + 23)(z^2 + 43z + 73)(z^2 + 50z + 4). \end{aligned}$$

  Therefore, $L_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ is not a field and, in particular, L_p contains zero divisors.

- *In MGCDNF and MRESNF, we choose a prime p and integers $C_1, \ldots, C_{n-1} \in [1, p)$ at random. We then invoke the algorithm LAminpoly with $\mathbb{F} = \mathbb{Z}_p$ and $\gamma = z_1 + C_1 z_2 + \cdots + C_{n-1} z_n \in L_p$. If LAminpoly returns FAIL (which means that $\det(A) = 0$), this failure may be caused either by the choice of p or by the values of $C_1, \ldots, C_{n-1}$. In that case, MGCDNF and MRESNF are restarted with a new prime p and a newly chosen random set of integers $C_1, \ldots, C_{n-1} \in [1, p)$. This process terminates because the number of unsuitable constants $C_1, \ldots, C_{n-1}$ and primes dividing $\det(A)$ is finite (see the proof of Theorem 3.2.4).*

3.4.3 The primitive element isomorphism ϕ_γ

We are now well-equipped to introduce the **primitive element** isomorphism $\phi_\gamma : L_p \longrightarrow \bar{L}_p$. Let

$$B_{L_p} = \{\prod_{i=1}^{n} (z_i)^{e_i} \; s.t \; 0 \leq e_i < d_i\}$$

be bases for L_p and let

$$B_{\bar{L}_p} = \{1, z, z^2, \ldots, z^{d-1}\}$$

be a basis for $\bar{L}_p$. Let $C : L_p \longrightarrow \mathbb{Z}_p^d$ be a bijection s.t. $C(a) = [a]_{B_{L_p}}$ and $D : \bar{L}_p \longrightarrow \mathbb{Z}_p^d$ be another bijection s.t. $D(b) = [b]_{B_{\bar{L}_p}}$. Define $\phi_\gamma : L_p \longrightarrow \bar{L}_p$ s.t.

$$\phi_\gamma(a) = D^{-1}(A^{-1} \cdot C(a)),$$

where A is the matrix obtained from the LAminpoly algorithm over $F = \mathbb{Z}_p$. The inverse of ϕ_γ is $\phi_\gamma^{-1} : \bar{L}_p \longrightarrow L_p$ s.t.

$$\phi_\gamma^{-1}(b) = C^{-1}(A \cdot D(b)).$$

Lemma 3.4.4. *If $\det(A) \neq 0$, then the mapping ϕ_γ defined above is a ring isomorphism.*

Proof. Since A^{-1} exists and both C and D are bijections, we can conclude that ϕ_γ is well-defined and bijective. Additionally, if the LAminpoly algorithm does not fail for $\gamma = z_1 + C_1 z_2 + \cdots + C_{n-1} z_n$, then ϕ_γ^{-1} can be expressed as an evaluation homomorphism that substitutes z for $z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$. The fact that ϕ_γ^{-1} is a homomorphism implies that ϕ_γ is also a ring homomorphism. □

Isomorphism ϕ_γ induces the natural isomorphism $\phi_\gamma : L_p[x_1, \ldots, x_k] \longrightarrow \bar{L}_p[x_1, \ldots, x_k]$. The following example illustrates how we can compute $\phi_\gamma(f)$ for $f \in L_p[x_1, \ldots, x_k]$.

Example 3.4.5. *Let* $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$. *Let* $p = 7$ *so*

$$L_p = \mathbb{Z}_7[z_1, z_2]/\langle z_1^2 + 5, z_2^2 + 4\rangle,$$

with basis $B_{L_p} = \{1, z_2, z_1, z_1 z_2\}$. *By Algorithm LAminpoly, we obtain*

$$\bar{L}_p = \mathbb{Z}_7[z]/\langle z^4 + 4z^2 + 1\rangle,$$

with basis $B_{\bar{L}_p} = \{1, z, z^2, z^3\}$. *Let*

$$f = 2x_1 z_1 + x_2 + z_1 z_2 \in L_p[x_1, x_2].$$

We aim to compute $\phi_\gamma(f)$. *Let* A *be the change-of-basis matrix obtained from Algorithm 4. We have* $[f]_{B_{L_p}} = [x_2, 0, 2x_1, 1]^T$ *and*

$$b = A^{-1} \cdot [f]_{B_{L_p}} = \begin{bmatrix} x_2 + 1 \\ 5x_1 \\ 4 \\ x_1 \end{bmatrix}$$

as the coordinate vector of $\phi_\gamma(f)$ *relative to* $B_{\bar{L}_p}$. *Consequently,*

$$\phi_\gamma(f) = x_1 z^3 + 5x_1 z + 4z^2 + x_2 + 1 \in \bar{L}_p[x_1, x_2].$$

3.5 The number of zero divisors in $\bar{L}_p$

Let $M(z) \in \mathbb{Z}_p[z]$ be the generator polynomial obtained from running Algorithm LAminpoly over $\mathbb{F} = \mathbb{Z}_p$ and let $d = \deg(M)$. Recall that $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ is a finite ring with p^d elements. As noted in Section 3.4.2, $M(z)$ is often reducible over $\mathbb{Z}_p$, which implies the presence of zero divisors in $\bar{L}_p$. In this section, we determine the number of zero divisors in $\bar{L}_p$ using the Chinese Remainder Theorem for ideals (Theorem 3.5.3). We begin by introducing some preliminary concepts.

Let R and S be commutative rings. The direct product $R \times S$ is a ring with componentwise addition and multiplication. More generally, let $\prod_{i \in I} R_i$ be a finite direct product of rings. Then every ideal of $\prod_{i \in I} R_i$ is of the form $\prod_{i \in I} A_i$, where each A_i is an ideal of R_i.

Definition 3.5.1. *Let* I *be a proper ideal of* R. *The ideal* I *is called* ***maximal*** *if any of the following equivalent conditions hold:*

- *There is no proper ideal* J *of* R *s.t.* $I \subset J$.
- *For any ideal* J *of* R *with* $I \subseteq J$, *either* $J = I$ *or* $J = R$.

Theorem 3.5.1. *[13, Theorem 7.6 in Section 7.1] Let* R *be a commutative ring with identity, and let* I *denote a proper ideal of* R. *Then the quotient ring* R/I *is a field if and only if* I *is a maximal ideal of* R.

Definition 3.5.2. *Let I_1 and I_2 be two ideals of a ring R. We define I_1 and I_2 as* ***comaximal*** *ideals if $I_1 + I_2 = R$.*

Example 3.5.1. *Let $a, b \in \mathbb{Z}$. The sets $a \cdot \mathbb{Z}$ and $b \cdot \mathbb{Z}$ are ideals of $\mathbb{Z}$. These ideals are comaximal if and only if $\gcd(a, b) = 1$. For instance, $3\mathbb{Z}$ and $5\mathbb{Z}$ are comaximal because $3\mathbb{Z} + 5\mathbb{Z} = \mathbb{Z}$.*

Theorem 3.5.2. *Any two distinct maximal ideals I_1 and I_2 of R are comaximal.*

Proof. Since $I_1 \subset I_1 + I_2 \subseteq R$ and I_1 is a maximal ideal of R, it follows that either $I_1 + I_2 = I_1$ or $I_1 + I_2 = R$. However, given that $I_1 \neq I_2$, we can conclude that $I_1 + I_2 = R$. Thus, I_1 and I_2 are comaximal. □

Now we are well-equipped to state the Chinese Remainder Theorem for ideals.

Theorem 3.5.3. *(Chinese remainder theorem for ideals)[13, Section 7.6]*
Let $I_1, \ldots, I_k$ be ideals in R s.t. for each $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j$, the ideals I_i and I_j are comaximal. Then

- $\cap_{i=1}^{k} I_i = \prod_{i=1}^{k} I_i$.
- $R/(\prod_{i=1}^{k} I_i) \cong R/I_1 \times \cdots \times R/I_k$.

Definition 3.5.3. *A commutative ring R is called local if it has a unique maximal ideal.*

Let $\#Z$ denote the number of non-zero zero divisors in $\bar{L}_p$. By applying Theorems 3.5.2 and 3.5.3, we obtain the value of $\#Z$ as stated in Theorem 3.5.4.

Theorem 3.5.4. *Let $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, and write the factorization of $M(z)$ over $\mathbb{Z}_p$ as*

$$M(z) = \prod_{i=1}^{r} P_i(z)^{e_i},$$

where $P_1(z), \ldots, P_r(z)$ are pairwise distinct irreducible polynomials, $e_i \geq 1$ for $1 \leq i \leq r$, and

$$d_i = \deg(P_i), \qquad d = \deg(M) = \sum_{i=1}^{r} e_i d_i.$$

Let $\#Z$ denote the number of non-zero zero divisors in $\bar{L}_p$. Then

$$\#Z = p^d - p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right) - 1.$$

Proof. Set $R_i = \mathbb{Z}_p[z]/\langle P_i(z)^{e_i}\rangle$ for $1 \leq i \leq r$. Since the polynomials $P_i(z)$ are pairwise distinct irreducibles, the ideals $\langle P_i(z)^{e_i}\rangle$ and $\langle P_j(z)^{e_j}\rangle$ are comaximal for $i \neq j$. Hence, by the Chinese Remainder Theorem,

$$\bar{L}_p \cong \prod_{i=1}^{r} R_i.$$

Now fix i. The ring R_i is finite and local with maximal ideal

$$\mathfrak{m}_i = \langle P_i(z)\rangle.$$

Since $\deg(P_i^{e_i}) = e_i d_i$, we have

$$|R_i| = p^{e_i d_i}.$$

Also,

$$|R_i/\mathfrak{m}_i| = p^{d_i},$$

so

$$|\mathfrak{m}_i| = p^{(e_i-1)d_i}.$$

Thus, the number of units in R_i is

$$|R_i^*| = p^{e_i d_i} - p^{(e_i-1)d_i}.$$

An element of $\prod_{i=1}^r R_i$ is a unit if and only if each component is a unit. Therefore,

$$\#U = \prod_{i=1}^{r} \left(p^{e_i d_i} - p^{(e_i-1)d_i}\right) = \prod_{i=1}^{r} p^{e_i d_i} \left(1 - \frac{1}{p^{d_i}}\right).$$

Since $d = \sum_{i=1}^r e_i d_i$, we obtain

$$\#U = p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right).$$

Finally, $|\bar{L}_p| = p^d$, and every nonzero element is either a unit or a zero divisor. Hence

$$p^d = 1 + \#U + \#Z.$$

Substituting the value of $\#U$ gives

$$\#Z = p^d - p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right) - 1.$$

□

Example 3.5.2. *Let $\bar{L}_5 = \mathbb{Z}_5[z]/\langle M(z)\rangle$ where $M(z) = (z+2)(z^2+4z+1)(z^2+z+1)$. Here, $d = 5$, $d_1 = 1$, $d_2 = 2$, $d_3 = 2$, $e_1 = e_2 = e_3 = 1$, $r = 3$, and $p = 5$. From Theorem 3.5.4, the number of non-zero zero divisors in $\bar{L}_5$ is*

$$\#Z = p^d - p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right) - 1 = 820.$$

Corollary 3.2. *Let $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, where $M(z) = \prod_{i=1}^r P_i(z)^{e_i}$, with each $d_i = \deg(P_i(z)) = 1$ for $1 \le i \le r$. The count of non-zero zero divisors in $\bar{L}_p$ is given by*

$$\#Z = p^d - p^d(1 - \frac{1}{p})^r - 1.$$

Proof. This follows directly from Theorem 3.5.4.

□

Example 3.5.3. *Let $\bar{L}_5 = \mathbb{Z}_5[z]/\langle M(z)\rangle$, where $M(z) = (z-4)(z-3)(z-2)(z-1)z$. Similar to Example 3.5.2, we have $p = 5$ and $d = 5$. In this example, $M(z)$ has 5 linear factors, so $d_i = 1$ for*

$1 \leq i \leq 5$ *and* $r = d$. *Since* $M(z)$ *is square-free, we have* $e_i = 1$ *for* $1 \leq i \leq 4$, *so*

$$M(z) = P_1(z) \cdots P_d(z).$$

By Corollary 3.2, the number of non-zero zero divisors of $\bar{L}_p$ *is*

$$\#Z = p^d - p^d(1 - \frac{1}{p})^r - 1 = 2100.$$

Comparing Examples 3.5.2 and 3.5.3, we observe that the number of zero divisors is larger when $M(z)$ is expressed as a product of linear factors. This suggests a general principle: the maximum possible number of zero divisors occurs when $M(z)$ factors linearly. This is established in the following theorem.

Theorem 3.5.5. *Let* $\#Z$, p, r *and* d *be as in Theorem 3.5.4. Then*

$$\#Z \leq p^d - p^d(1 - \frac{1}{p})^r - 1.$$

Proof. Since $d_i \geq 1$ for $1 \leq i \leq r$, we have $p^{-d_i} \leq p^{-1}$. Hence

$$1 - p^{-d_i} \geq 1 - p^{-1}.$$

Multiplying these inequalities for $i = 1, \ldots, r$ gives

$$\prod_{i=1}^{r} \left(1 - p^{-d_i}\right) \geq (1 - p^{-1})^r = (1 - 1/p)^r.$$

Since $p^d > 0$, it follows that

$$p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right) \geq p^d(1 - 1/p)^r.$$

Subtracting both sides from $p^d - 1$ reverses the inequality. Therefore,

$$\#Z = p^d - p^d \prod_{i=1}^{r} \left(1 - p^{-d_i}\right) - 1 \leq p^d - p^d(1 - \frac{1}{p})^r - 1.$$

Equality holds if and only if $d_i = 1$ for all $1 \leq i \leq r$. □

Corollary 3.3. *Let* $\#Z$, p, r, *and* d *be as in Theorem 3.5.4. Then*

$$\#Z \leq p^d - (p-1)^d - 1.$$

Proof. By Theorem 3.5.5,

$$\#Z \leq p^d - p^d(1 - \frac{1}{p})^r - 1.$$

Since

$$p^d \left(1 - \frac{1}{p}\right)^r = p^d \left(\frac{p-1}{p}\right)^r = p^{d-r}(p-1)^r,$$

we need to compare $p^{d-r}(p-1)^r$ with $(p-1)^d$. If $r < d$, then $d - r > 0$. Since $p > p - 1 > 0$, we have

$$p^{d-r} > (p-1)^{d-r}.$$

Multiplying both sides by $(p-1)^r > 0$, we obtain

$$p^{d-r}(p-1)^r > (p-1)^d.$$

Thus,

$$p^d \left(1 - \frac{1}{p}\right)^r > (p-1)^d.$$

It follows that

$$p^d - p^d \left(1 - \frac{1}{p}\right)^r - 1 < p^d - (p-1)^d - 1.$$

Therefore, in this case,

$$\#Z < p^d - (p-1)^d - 1.$$

If $r = d$, then

$$p^d \left(1 - \frac{1}{p}\right)^r = p^d \left(1 - \frac{1}{p}\right)^d = (p-1)^d.$$

Hence

$$p^d - p^d \left(1 - \frac{1}{p}\right)^r - 1 = p^d - (p-1)^d - 1.$$

Therefore, in this case,

$$\#Z = p^d - (p-1)^d - 1.$$

Combining the two cases, we conclude that

$$\#Z \leq p^d - (p-1)^d - 1.$$

□

Example 3.5.4. *Let $p = 2^{30} + 3$ be the smallest 31-bit prime, $M(z) = (z+2)(z^2+4z+1)(z^2+z+4)$, and $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$. Then*

$$\begin{aligned}\text{Prob}[a \in \bar{L}_p \textit{ is a zero divisor}] &= \frac{\#Z}{p^d - 1} \leq \frac{p^5 - ((p-1)(p^2-1)(p^2-1)) - 1}{p^5 - 1} \\ &= \frac{1329228013116076503603636391334903922}{1427247712644379929246627909979632460223217906} \\ &= 9.3 \times 10^{-10}.\end{aligned}$$

As illustrated in the example above, when p is sufficiently large, the proportion of zero divisors in $\bar{L}_p$ becomes negligible compared with the total number of non-zero elements in $\bar{L}_p$. Consequently, when p is large, the probability of encountering a zero divisor while computing over $\bar{L}_p$ is small. Before proving this in Lemma 3.5.7, we recall Bernoulli's inequality, stated as follows.

Theorem 3.5.6. *[24, Section 0.2, Bernoulli's Inequality, p. 15] If $x > -1$ and $n \in \mathbb{N}$, then*

$$(1+x)^n \geq 1 + nx.$$

Lemma 3.5.7. *Let p be a prime and $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ with $d = \deg(M(z))$. Then*

$$\text{Prob}[a \in \bar{L}_p \text{ is a zero divisor}] = \frac{\#Z}{p^d - 1} < \frac{d}{p}.$$

Proof. From Theorem 3.5.5, it follows that

$$\frac{\#Z}{p^d - 1} \leq \frac{p^d - (p-1)^d - 1}{p^d - 1}.$$

Let $A = p^d - (p-1)^d$. Then

$$\begin{aligned}\frac{A}{p^d} - \frac{A-1}{p^d - 1} &= \frac{Ap^d - A - Ap^d + p^d}{p^d(p^d - 1)} \\ &= \frac{p^d - A}{p^d(p^d - 1)} = \frac{(p-1)^d}{p^d(p^d - 1)} > 0.\end{aligned}$$

Hence, $\frac{A-1}{p^d-1} < \frac{A}{p^d}$ and

$$\begin{aligned}\frac{\#Z}{p^d - 1} = \frac{A-1}{p^d - 1} &< \frac{A}{p^d} \\ &= \frac{p^d - (p-1)^d}{p^d} \\ &= 1 - (1 - \frac{1}{p})^d.\end{aligned}$$

Applying Bernoulli's inequality for $(1 - \frac{1}{p})^d$, we have $(1 - \frac{1}{p})^d \geq 1 - \frac{d}{p}$. Hence, from the above equation,

$$\begin{aligned}\frac{\#Z}{p^d - 1} &< 1 - (1 - \frac{1}{p})^d \\ &< 1 - 1 + \frac{d}{p} = \frac{d}{p}.\end{aligned}$$

□

From Lemma 3.5.7, if p is large and d is small, then $\frac{d}{p}$ would be small. In our modular algorithms, we use a significantly large p (31-bit prime) to construct the quotient ring $\bar{L}_p$. Consequently, in our algorithms, the value of $\frac{\#Z}{p^d-1}$ is small.

Chapter 4

GCDs in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$

4.1 Summary of contributions

Except for Sections 4.2 and 4.3, all material in this chapter is original and is based on our paper published in the Proceedings of CASC '23 [2]. The main contribution of this chapter is a new modular GCD algorithm, referred to as MGCDNF (Algorithm 8), which computes the monic GCD g of two non-zero polynomials $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ with $n > 1$ and $k \geq 1$ with high probability. In this chapter, we fix the lexicographic order over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ with $x_1 > x_2 > \ldots > x_k$. MGCDNF is the first modular GCD algorithm to achieve a speed-up by mapping input polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to corresponding polynomials over $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. This mapping is performed modulo a prime to avoid expression swell. In Theorem 4.4.3, we prove the correctness of MGCDNF.

In Section 4.3, we review Encarnacion's algorithm [14] for computing the monic GCD of univariate polynomials over $\mathbb{Q}(\alpha)$. Section 4.4 introduces a key subalgorithm, PGCDNF (Algorithm 7), which computes the monic GCD of two polynomials over $\bar{L}_p$, and then presents the modular GCD algorithm MGCDNF. In Section 4.5, we analyze the expected time complexity of MGCDNF and summarize two benchmark experiments. Moreover, a failure probability analysis of the MGCDNF algorithm is presented in Chapter 6. We have implemented MGCDNF and its subalgorithms in Maple, employing a recursive dense data structure for representing polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k, y]$ (see Chapter 7 for more details). Implementation details and benchmark data are available at `https://github.com/MahsAnsari/Mahsa_Ansari_Thesis`.

4.2 A brief history

In 1989, Langemyr and McCallum [22] modified Brown's modular GCD algorithm (see Section 2.2.8) to handle univariate polynomials over an algebraic number field $\mathbb{Q}(\alpha)$. Let $f_1, f_2 \in \mathbb{Q}(\alpha)[x]$ and $g = \gcd(f_1, f_2)$. In a preprocessing step, they replace α with an algebraic integer β that generates the same field and has the monic minimal polynomial $M(z) \in \mathbb{Z}[z]$. Consequently, computations are carried out in the algebraic integer extension

$$\mathbb{Z}[z]/\langle M(z)\rangle \cong \mathbb{Z}[\alpha] = \{a_0 + a_1 z + \cdots + a_{d-1} z^{d-1} \; s.t. \; a_i \in \mathbb{Z}\},$$

where $d = \deg(M)$. They also clear denominators in the inputs by replacing f_1 and f_2 with their associates $\tilde{f}_1$ and $\tilde{f}_2$. Thus, the algorithm takes as input $M(z) \in \mathbb{Z}[z]$ and $f_1, f_2 \in \mathbb{Z}[z]/\langle M(z)\rangle[x]$ with integer coefficients. As in Brown's method, the algorithm reduces the inputs modulo primes p and computes modular images $g_p = \gcd(\phi_p(\tilde{f}_1), \phi_p(\tilde{f}_2)) \in \mathbb{Z}_p[z]/\langle M_p\rangle[x]$, where $M_p = M(z) \mod p$. However, not all primes result in a successful reconstruction of the GCD. To state the required conditions on the primes, we define the following:

(i) The discriminant of $M(z)$, denoted by $\mathrm{disc}(M)$, is given by

$$\mathrm{disc}(M) = (-1)^{\frac{d(d-1)}{2}} \mathrm{lc}(M)^{-1} \mathrm{res}(M, M'),$$

where M' is the derivative of M.

(ii) Let $F_i = \mathrm{res}(M, \mathrm{lc}(f_i))$ denote the Sylvester resultant of M and $\mathrm{lc}(f_i)$ w.r.t. the variable z, for $i = 1, 2$ (see Definitions 5.2.1 and 5.2.2). Define $h = \gcd(F_1, F_2) \cdot \mathrm{disc}(M)$.

Langemyr and McCallum restrict to primes p satisfying $p \nmid \mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2) \cdot h$. For sufficiently many such primes, they apply the MEA modulo p and then reconstruct the associate $h \cdot g \in \mathbb{Z}[\alpha][x]$ from its homomorphic images using the CRT. To prove the correctness of their algorithm, they assumed that α is an algebraic integer in order to bound the denominators of g. However, later in 1995, Encarnacion [14] showed that this assumption can be removed, provided $p \nmid \mathrm{lc}(\tilde{M})$.

Even when the input polynomials have coefficients in $\mathbb{Z}[\alpha]$, their GCD often lies over $\mathbb{Q}(\alpha)$. To recover the rational coefficients of the monic g, Encarnacion [14] employed rational number reconstruction(RNR) [38], which makes his modular algorithm over $\mathbb{Q}(\alpha)$ output-sensitive (see Section 2.2.6 for RNR). In particular, his method reconstructs the rational coefficients of the monic GCD without requiring a multiple of the denominator of the GCD. His algorithm restricts to primes p s.t. $p \nmid \mathrm{lc}(f_1) \cdot \mathrm{lc}(f_2) \cdot \mathrm{lc}(\tilde{M}) \cdot \mathrm{disc}(M)$. In this setting, there is no need to test whether $p \mid \gcd(F_1, F_2)$. If $p \mid \gcd(F_1, F_2)$, then $p \mid F_2$, which implies that $\mathrm{lc}(f_2)$ is a zero divisor in $\mathbb{Z}_p[z]/\langle M_p\rangle$. Therefore, the MEA returns FAIL for the inputs $\phi_p(f_1)$ and $\phi_p(f_2)$, and the algorithm then chooses a new prime for the GCD computation. Hence, the check is implicit in the algorithm, and omitting it has the advantage of avoiding the computation of $\gcd(F_1, F_2)$, which may require an appreciable amount of time.

4.3 Encarnacion's algorithm

In this section, we review Encarnacion's algorithm. Let $M(z) \in \mathbb{Q}[z]$ denote the minimal polynomial of α over $\mathbb{Q}$, so $\mathbb{Q}(\alpha) \cong \mathbb{Q}[z]/\langle M(z)\rangle$. Let $f_1, f_2 \in \mathbb{Q}[z]/\langle M(z)\rangle[x]$ with $\deg(f_1) \geq \deg(f_2) \geq 0$. Let g be the monic $\gcd(f_1, f_2)$. First, Encarnacion's algorithm reduces computations over $\mathbb{Q}[z]/\langle M(z)\rangle[x]$ to a simpler domain $\mathbb{Z}[z]/\langle \tilde{M}(z)\rangle[x]$ by replacing f_1 and f_2 with their associates $\tilde{f}_1$ and $\tilde{f}_2$. Let $L_p = \mathbb{Z}_p[z]/\langle M_p\rangle$, where p is a prime. Figure 4.1 outlines the overall structure of Encarnacion's algorithm.

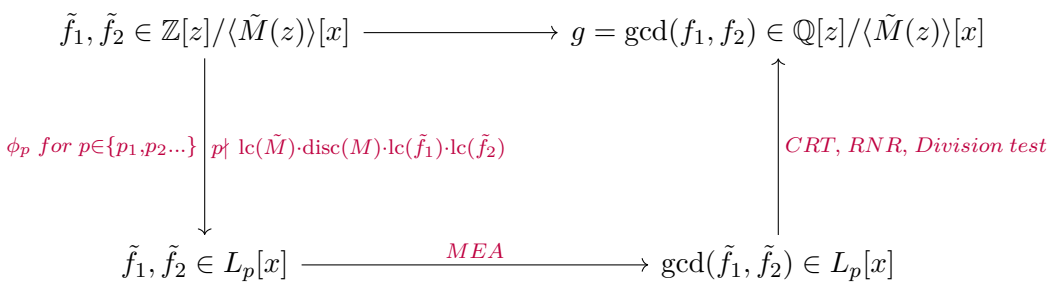

Figure 4.1: Homomorphism diagram of Encarnacion's algorithm

As in Brown's GCD algorithm, not all primes are suitable for modular homomorphisms (see Section 2.2.5). In particular, the chosen prime p must not divide $\mathrm{lc}(f_1)$, $\mathrm{lc}(f_2)$, $\mathrm{lc}(\tilde{M})$, or $\mathrm{disc}(M)$. If $p \mid \mathrm{lc}(f_1)$ (or $p \mid \mathrm{lc}(f_2)$) , then we cannot employ Lemma 2.2.9 to detect whether p is an unlucky prime, and we cannot recover g from its image $g_{\phi_p} = \gcd(\phi_p(\tilde{f}_1), \phi_p(\tilde{f}_2))$ (See Example 2.2.13 and the explanations above it). Moreover, if $p \mid \mathrm{lc}(\tilde{M})$, then the degree of the ground algebraic number field changes modulo p. Furthermore, to reconstruct the rational coefficients of g using Wang's method (see Section 2.2.6), it is necessary that no prime p divides the denominator of any rational coefficient appearing in g. Encarnacion's algorithm, presented in Algorithm 5, is illustrated in Example 4.3.1.

Example 4.3.1. *Let $f_1 = a \cdot g$ and $f_2 = b \cdot g$ be two polynomials in $\mathbb{Q}[z]/\langle z^2 - 5\rangle[x]$, where*

$$\begin{aligned} a &= zx + 1 \\ b &= 2x + 6, \text{ and} \\ g &= x^2 + \frac{1}{2}xz + \frac{9}{68}. \end{aligned}$$

Encarnacion's algorithm begins by replacing the input polynomials f_1 and f_2 with their associates:

$$\begin{aligned} \tilde{f}_1 &= 340x^3 + 238zx^2 + 215x + 9z \\ \tilde{f}_2 &= 68x^3 + (34z + 204)x^2 + (102z + 9)x + 27 \in \mathbb{Z}[z][x]. \end{aligned}$$

Then it chooses a prime, say $p_1 = 107$. Since $p_1 \nmid \mathrm{lc}(f_1) = z$, $p_1 \nmid \mathrm{lc}(f_2) = 2$, $p_1 \nmid \mathrm{lc}(M) = 1$, and $p \nmid \mathrm{disc}(M) = 20$, we can use prime p_1 for the modular reduction (see Line 4). Hence,

$$\begin{aligned} \phi_{p_1}(\tilde{f}_1) &= 19x^3 + 24zx^2 + x + 9z \\ \phi_{p_1}(\tilde{f}_2) &= 68x^3 + (34z + 97)x^2 + (102z + 9)x + 27 \in L_{p_1}[x] = \mathbb{Z}_{107}[z]/\langle z^2 - 5\rangle[x]. \end{aligned}$$

Applying the MEA to compute the GCD modulo p_1, we obtain

$$g_1 = \gcd(\phi_{p_1}(\tilde{f}_1), \phi_{p_1}(\tilde{f}_2)) = x^2 + 54zx + 8 \in L_{p_1}[x].$$

Algorithm 5: Encarnacion

Input: $f_1, f_2 \in \mathbb{Q}(\alpha)[x]$, $M(z)$ the minimal polynomial for α over $\mathbb{Q}$.
Output: monic $\gcd(f_1, f_2)$.

1 $P = 1$
2 $presult = 0$ // We compare the two consecutive results obtained from applying CRT and RNR. If they match, we then perform the division test.
3 **while** *true* **do**
4 Choose a new prime p s.t $p \nmid \mathrm{lc}(f_1)$, $p \nmid \mathrm{lc}(f_2)$, $p \nmid \mathrm{lc}(\tilde{M})$, $p \nmid \mathrm{disc}(M)$.
5 Let $f_{1_p} = \tilde{f_1} \mod p$, $f_{2_p} = \tilde{f_2} \mod p$, and $M_p = M \mod p$.
6 Run Algorithm 2 to compute $g_p = \gcd(f_{1_p}, f_{2_p})$ in $\mathbb{Z}_p[z]/\langle M_p \rangle[x]$.
7 **if** *Algorithm 2 hits a zero divisor* **then**
8 go to Line 4
9 **if** $g_p = 1$ **then**
// According to Lemma 2.2.9, GCD is 1.
10 **return**(1)
11 **if** $P = 1$ **then**
12 $G = g_p$
13 $P = p$
14 **else**
15 **if** $\deg(g_p, x) < \deg(G, x)$ **then**
// All the previous primes were unlucky (see Lemma 2.2.9).
16 $G = g_p$
17 $P = p$
18 **else if** $\deg(g_p, x) > \deg(G, x)$ **then**
// p is an unlucky prime (see Lemma 2.2.9).
19 Go to Line 4.
20 **else if** $\deg(G, x) = \deg(g_p, x)$ **then**
21 Find G' s.t $G' \equiv G \mod P$ and $G' \equiv g_p \mod p$.
22 Set $G = G'$ and $P = P \cdot p$.
23 $g = RNR(G, P)$ // Apply RNR to the coefficients of $G(x) \mod P$ to get $g(x) \in \mathbb{Q}(\alpha)[x]$.
24 **if** $g \neq FAIL$ **then**
25 **if** $g = presult$ *and* $g \mid f_1$ *and* $g \mid f_2$ **then**
26 **return**(g)
27 **else**
28 $presult = g$

We set $g = g_1$ and $P = p_1$. To reconstruct the rational coefficients of the GCD, we apply rational number reconstruction (RNR) to the integer coefficients of g with modulus P and bounds

$$N = D = \left\lfloor \sqrt{\frac{P}{2}} \right\rfloor = 7,$$

so that $2ND = 98 < P = 107$ (see Theorem 2.2.10). The RNR procedure fails in this case because one of the coefficients of g is $\frac{9}{68}$, which cannot be reconstructed modulo P since $D = 7 < 68$. Therefore,

we need a new prime satisfying Line 4 of Algorithm 5. Let $p_2 = 109$. *Next, we compute*

$$\begin{aligned}\phi_{p_2}(\tilde{f}_1) &= 13x^3 + 20zx^2 + 106x + 9z\\ \phi_{p_2}(\tilde{f}_2) &= 68x^3 + (34z + 95)x^2 + (102z + 9)x + 27\\ g_2 &= \gcd(\phi_{p_2}(\tilde{f}_1), \phi_{p_2}(\tilde{f}_2)) = x^2 + 55xz + 37 \in L_{p_2}[x] = \mathbb{Z}_{109}[z]/\langle z^2 - 5\rangle[x].\end{aligned}$$

Applying the CRT (Line 21 of Algorithm 5) to the coefficients of g_1 *and* g_2 *modulo* $P = p_1 \times p_2 = 11663$, *we obtain*

$$G = x^2 + 5832xz + 4288 \in \mathbb{Z}_P[z]/\langle z^2 - 5\rangle[x].$$

We then perform RNR on the integer coefficients of G *with modulus* $P = 11663$ *and bounds* $N = D = \left\lfloor \sqrt{\frac{P}{2}} \right\rfloor = 76$. *This time, RNR succeeds and yields*

$$g = x^2 + \frac{1}{2}zx + \frac{9}{68}. \tag{4.1}$$

We set $presult = g$ *and repeat the above process for another prime* $p_3 = 113$. *Thus,*

$$g_3 = \gcd(\phi_{p_3}(\tilde{f}_1), \phi_{p_3}(\tilde{f}_2)) = x^2 + 57zx + 45 \in \bar{L}_p[x].$$

By applying CRT followed by RNR, we obtain the same result for g *as in (4.1). Because the two consecutive RNR outputs are identical, we proceed with the division test. Since* $g \mid f_1$ *and* $g \mid f_2$, *we conclude that* g *is the monic GCD of* f_1 *and* f_2.

One of the main difficulties in modular GCD algorithms over algebraic number fields is the presence of zero divisors in the finite ring L_p. In Line 6 of Algorithm 5, the algorithm calls the MEA to compute the monic GCD over L_p. This requires inverting the leading coefficients of intermediate remainders in L_p (see Algorithm 2). If a leading coefficient of the remainders is a zero divisor, this inversion fails. For instance, let $\alpha = \sqrt{2}$ with minimal polynomial $M(z) = z^2 - 2$. Suppose Algorithm 5 chooses the prime $p = 17$ in Line 4, and the leading coefficient of an intermediate remainder in the MEA is $z + 6$. Then $z + 6$ is a zero divisor in $L_p = \mathbb{Z}_{17}[z]/\langle z^2 - 2\rangle$, since

$$z^2 - 2 \equiv (z + 6)(z + 11) \mod 17.$$

In this situation, MEA fails when attempting to invert $z+6$. We will return to the issue of zero divisors and their impact on our modular algorithm in Section 4.4.2.

The argument for the correctness of Encarnacion's algorithm parallels that of the MGCDNF, which is stated in Theorem 4.4.3. To avoid repetition, we therefore omit a separate proof of correctness for Encarnacion's algorithm.

Remark 4.1. *It is useful to highlight the following distinctions between our MGCDNF algorithm and those of Langemyr–McCallum and Encarnacion.*

- *Computing the associates of the input polynomials* f_1 *and* f_2 *is useful for removing denominators. However, if* $\mathrm{lc}(f_1)$ *and* $\mathrm{lc}(f_2) \in \mathbb{Q}(\alpha)$ *are complicated algebraic numbers, computing the associate would be expensive. This is especially true when working over algebraic number fields with multiple extensions, so that* $\mathrm{lc}(f_1)$ *and* $\mathrm{lc}(f_2) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. *For this reason, instead*

of using associates, we replace the input polynomials with their semi-associates (see Definition 3.3.2).

- *In both the Langemyr–McCallum and Encarnacion algorithms, primes p with $p \mid \operatorname{disc}(M)$ are avoided. However, in 2002, Monagan and Van Hoeij [34] showed that it is not necessary to check whether $p \mid \operatorname{disc}(M)$. In [34, Theorem 1], they proved that the modular GCD algorithm works, provided that $p \nmid \operatorname{lc}(\check{M}) \cdot \operatorname{lc}(\check{f}_1) \cdot \operatorname{lc}(\check{f}_2)$. Algorithm MGCDNF also does not check whether $p \mid \operatorname{disc}(M)$.*

Remark 4.2. *One may ask why Encarnacion's algorithm uses a sequence of primes rather than a single large prime to reconstruct the coefficients of the GCD, especially since using several primes may force the algorithm to restart when an unlucky prime is encountered. The reason is that using one large prime would make the algorithm no longer output-sensitive. In particular, when the size of the monic GCD is much smaller than the size of the input polynomials, a prime chosen from an a priori bound based on the input size may be unnecessarily large. For instance, let $f_1, f_2 \in \mathbb{Q}(\alpha)[x]$ with the monic $g = \gcd(f_1, f_2)$. Let $\|\tilde{f}_1\|_\infty$ and $\|\tilde{f}_2\|_\infty$ each have 1000 digits, then an input-based bound may require choosing a prime with about 2000 digits. However, if $\|\tilde{g}\|_\infty$ has only 10 digits, then using such a large prime is inefficient since arithmetic modulo a 2000-digit prime is much slower, and finding such a prime is itself expensive. Alternatively, using a few 31-bit primes is sufficient to reconstruct the coefficients of the monic GCD. For instance, four 31-bit primes give a modulus of about 124 bits, which is more than enough to reconstruct rational coefficients of the GCD. Therefore, using a sequence of small primes preserves the output-sensitive nature of the algorithm.*

4.4 A new modular algorithm for computing GCDs over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$

In this section, we fix the lexicographic order with $x_1 > x_2 > \ldots > x_k$. Let p be a prime number s.t. $p \nmid \prod_{i-1}^{n} \operatorname{lc}(\check{M}_i, z_i) \cdot \operatorname{lc}(\check{f}_1) \cdot \operatorname{lc}(\check{f}_2)$. From Section 3, recall that:

- $\bar{L} = \mathbb{Z}[z]/\langle M(z)\rangle$, where $M(z)$ is obtained from Algorithm 4 over $\mathbb{F} = \mathbb{Q}$.
- $L_{\mathbb{Z}} = \mathbb{Z}[z_1, \ldots, z_n]$,
- $L_p = \mathbb{Z}_p[z_1, \ldots, z_n]/\langle m_1, \ldots, m_n\rangle$ where $m_i \equiv \check{M}_i \mod p$,
- $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, where $M(z)$ is obtained from Algorithm 4 over $\mathbb{F} = \mathbb{Z}_p$.

As mentioned in Section 2.2, homomorphic reduction allows modular algorithms to transform the GCD problem over $\mathbb{Q}$ into a simpler domain $\mathbb{Z}_p$, where the coefficients do not grow. As shown in Section 4.3, Encarnacion employed RNR to recover the rational coefficients of the target GCD. Following the same strategy, our algorithm, **MGCDNF**, is a development of Brown's and Encarnacion's GCD algorithm for computing GCDs over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. The key distinction in our approach is the introduction of a new homomorphism ϕ_γ, which accelerates computation by reducing the input polynomials over L_p to their corresponding images over $\bar{L}_p$ (see Section 3.4.3). Subsequently, using the evaluation homomorphism, the MEA is applied to compute the monic GCD in $\bar{L}_p[x_1]$. The homomorphism diagram of the MGCDNF algorithm is presented in Figure 4.2. We begin with a motivating example (Example 4.4.1), which illustrates the key ideas underlying the MGCDNF algorithm. In Section 4.4.2, we present the

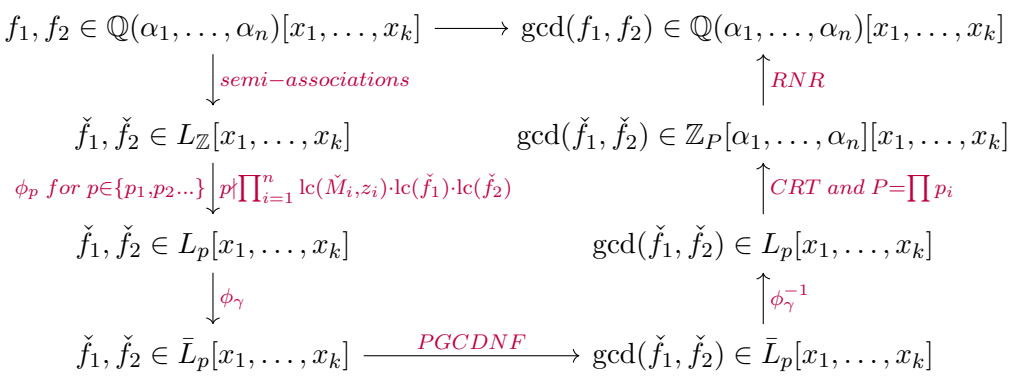

Figure 4.2: Homomorphism diagram of the MGCDNF algorithm.

subalgorithm PGCDNF, which computes the monic GCD of polynomials in $\bar{L}_p[x_1, \ldots, x_k]$. Finally, in Section 4.4.1, we describe the full MGCDNF algorithm.

Example 4.4.1. *Let*

$$f_1 = (z_2x_1 + z_1x_2)(z_1x_1 + x_2) \quad \textit{and} \quad f_2 = (z_2x_1 + z_1x_2)(x_1 - z_2x_2) \in L[x_1, x_2],$$

where $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$. *We illustrate how the MGCDNF computes the monic*

$$g = \gcd(f_1, f_2) = \mathrm{monic}(z_2x_1 + z_1x_2) = x_1 + \frac{1}{3}x_2z_2z_1.$$

In this example, $\check{f}_1 = f_1$ *and* $\check{f}_2 = f_2$. *MGCDNF first chooses a prime* p *s.t.* $p \nmid \mathrm{lc}(\check{M}_1) \cdot \mathrm{lc}(\check{M}_2) \cdot \mathrm{lc}(\check{f}_1) \cdot \mathrm{lc}(\check{f}_2)$. *Take* $p = 5$. *Reducing the minimal polynomials modulo* p *gives* $m_1 = \phi_p(z_1^2 - 2) = z_1^2 + 3$ *and* $m_2 = \phi_p(z_2^2 - 3) = z_2^2 + 2$ *and therefore*

$$L_5 = \mathbb{Z}_5[z_1, z_2]/\langle\, z_1^2 + 3, z_2^2 + 2\,\rangle.$$

Reducing the inputs modulo p *yields*

$$\begin{aligned}\phi_p(f_1) &= z_1z_2x_1^2 + (z_2 + 2)x_1x_2 + x_2^2z_1,\\ \phi_p(f_2) &= z_2x_1^2 + (z_1 + 2)x_1x_2 + 4z_1z_2x_2^2 \in L_5[x_1, x_2].\end{aligned}$$

To convert the ring L_5 *into a quotient ring with single extension, MGCDNF calls LAminpoly (Algorithm 4) with the inputs* $[m_1, m_2]$, $\mathbb{F} = \mathbb{Z}_5$, *and* $\gamma = z_1 + C_1z_2$, *where* $C_1 = 4 \in [0, 5)$ *is chosen randomly. The LAMinpoly algorithm computes the generator polynomial* $M(z) = z^4 + 1$ *s.t.*

$$\bar{L}_5 = \mathbb{Z}_5[z]/\langle z^4 + 1\rangle \cong L_5.$$

After mapping $\phi_p(f_1)$ *and* $\phi_p(f_2)$ *to their corresponding polynomials in* $\bar{L}_5[x_1, x_2]$,

$$\begin{aligned}\phi_\gamma(\phi_p(f_1)) &= 3x_1^2z^2 + (2z^3 + 3z + 2)x_2x_1 + (3z^3 + 3z)x_2^2\\ \phi_\gamma(\phi_p(f_2)) &= (2z^3 + 3z)x_1^2 + (3z^3 + 3z + 2)x_2x_1 + 2x_2^2z^2,\end{aligned}$$

MGCDNF calls PGCDNF (Algorithm 7) to compute $\gcd(\phi_\gamma(\phi_p(f_1)), \phi_\gamma(\phi_p(f_2))) \in \bar{L}_5[x_1, x_2]$. *Algorithm PGCDNF picks a random evaluation point* $x_2 = 2$, *and computes*

$$r_1 = \phi_\gamma(\phi_p(f_1))(x_2 = 2) = 3z^2x_1^2 + (4z^3 + z + 4)x_1 + 2z^3 + 2z$$
$$r_2 = \phi_\gamma(\phi_p(f_2))(x_2 = 2) = (2z^3 + 3z)x_1^2 + (z^3 + z + 4)x_1 + 3z^2 \in \bar{L}_5[x_1].$$

Then it applies MEA (Algorithm 2) to compute the monic GCD $g_{5,2} = \gcd(r_1, r_2)$. *In the first iteration of the MEA, we have*

$$r_3 = \mathrm{rem}(r_1, \mathrm{monic}(r_2)) = (2z^3 + 4z)x_1 + 3z^3 + z,$$

Since

$$\gcd(z^4 + 1, \mathrm{lc}(r_3)) = \gcd(z^4 + 1, 2z^3 + 4z) = z^2 + 2 \neq 1,$$

$\mathrm{lc}(r_3)$ *is not invertible over* $\bar{L}_5$. *Hence MEA returns FAIL. We call* $(p, \beta) = (5, 2)$ *a zero divisor pair (see Definition 4.4.2). Since MGCDNF cannot determine whether this failure is due to the choice of the prime* $p = 5$ *or the evaluation point* $x_2 = 2$, *it stops the computation modulo* $p = 5$ *and proceeds with a different prime.*

Take $p = 7$. *Applying LAminpoly over* $\mathbb{F} = \mathbb{Z}_7$ *for* $\gamma = z_1 + 2z_2$, *we obtain* $M(z) = z^4 + 2 \in \mathbb{Z}_7[z]$ *and*

$$\bar{L}_7 = \mathbb{Z}_7[z]/\langle z^4 + 2\rangle \cong L_7 = \mathbb{Z}_7[z]/\langle z_1^2 + 5, z_2^2 + 4\rangle.$$

After evaluating $\phi_\gamma(\phi_7(f_1))$ *and* $\phi_\gamma(\phi_7(f_2)) \in \bar{L}_7[x_1, x_2]$ *at* $x_2 = 2$, *the MEA algorithm returns*

$$g_{7,2} = \gcd(\phi_\gamma(\phi_p(f_1))(x_2 = 2), \phi_\gamma(\phi_p(f_2))(x_2 = 2)) = x_1 + 6z^2 \in \bar{L}_7[x_1].$$

Notice that $\mathrm{lm}(g_{7,1}) = x_1$. *A second evaluation at* $x_2 = 0$ *yields*

$$g_{7,0} = \gcd(\phi_\gamma(\phi_p(f_1))(x_2 = 0), \phi_\gamma(\phi_p(f_2))(x_2 = 0)) = x_1^2 \in \bar{L}_7[x_1],$$

which must be discarded because $\mathrm{lm}(g_{7,0}) = x_1^2 > \mathrm{lm}(g_{7,1}) = x_1$, *making* $\phi_{x_2=0}$ *an unlucky homomorphism (see Lemma 2.2.9 and Definition 4.4.3). Further evaluations at* $x_2 = 4$ *and* $x_2 = 5$ *allows interpolation of the monic GCD at* x_2 *modulo* 7 *so PGCDNF returns*

$$g_7 = \gcd(\phi_\gamma(\phi_7(f_1)), \phi_\gamma(\phi_7(f_2))) = x_1 + 3z^2x_2 \in \bar{L}_7[x_1, x_2].$$

Applying ϕ_γ^{-1}, *the MGCDNF computes*

$$\phi_\gamma^{-1}(g_7) = x_1 + 5z_2z_1x_2 \in L_7[x_1, x_2].$$

The MGCDNF algorithm then attempts to reconstruct the rational coefficients of g *using CRT and RNR. Since one modular image is insufficient, the above process is repeated for another prime such as* $p = 11$. *Calling LAMinpoly over* $\mathbb{F} = \mathbb{Z}_{11}$ *for* $\gamma = z_1 + 4z_2$, *we have*

$$\bar{L}_{11} = \mathbb{Z}_{11}/\langle z^4 + 10z^2 + 4\rangle \cong L_{11} = \mathbb{Z}_p[z_1, z_2]/\langle z_1^2 + 9, z_2^2 + 8\rangle.$$

Computing the GCD over $\bar{L}_{11}$ using the PGCDNF and then converting its output to a polynomial over L_{11}, we obtain

$$\phi_\gamma^{-1}(g_{11}) = x_1 + 4z_2z_1x_2 \in L_{11}[x_1, x_2].$$

We then apply the CRT to reconstruct the integer coefficients of the GCD, i.e. solve for G s.t.

$$\{G \equiv x_1 + 5z_2z_1x_2 \mod 7, \quad G \equiv x_1 + 4z_2z_1x_2 \mod 11\}.$$

We obtain, $G = x_1 + 26z_1z_2x_2$. To recover the rational coefficients of the GCD, we use RNR for G modulo $7 \times 11 = 77$. The RNR procedure returns

$$H = x_1 + \frac{1}{3}x_2z_2z_1 \in L[x_1, x_2].$$

Performing the same computation for an additional prime, namely $p = 13$, yields the same polynomial H after the RNR step. Since the result of RNR has stabilized, we perform the division test. As $H \mid f_1$ and $H \mid f_2$, from Lemma 2.2.9, we conclude that $H = g = \gcd(f_1, f_2) \in L[x_1, x_2]$.

Direct divisibility testing over L is computationally expensive. Instead, the algorithm reduces the candidate H obtained from RNR, together with the inputs f_1 and f_2, modulo a new prime q, and then evaluates them at $x_2 = \beta \in [0, q)$. This produces univariate polynomials $f_1(x_1, \beta)$, $f_2(x_1, \beta)$, $H(x_1, \beta) \in L_q[x_1]$. MGCDNF then verifies divisibility by performing trial division in $L_q[x_1]$.

As illustrated in the above example, not every choice of prime and evaluation point leads to a successful reconstruction of the monic GCD in MGCDNF. Sections 4.4.2 and 4.4.3 describe the PGCDNF and MGCDNF algorithms in detail and characterize the primes and evaluation points for which the reconstruction fails.

4.4.1 Division Test

In the last step of MGCDNF and PGCDNF, we run a divisibility test to verify whether the result is the monic GCD of the inputs. To do so, we use Algorithm DT (Algorithm 6). Let R be a commutative ring with characteristic p. Algorithm DT takes polynomials $f_1, f_2, H \in R[x_1, \ldots, x_k]$ and the characteristic of R, p. In Line 31 of PGCDNF, we employ DT (Algorithm 6) over $R = \bar{L}_p$ while in Line 30 of MGCDNF, DT is run over $R = L_q$ where q is a randomly chosen prime.

For each $1 \leq i \leq k$, DT reduces f_1, f_2, and H to univariate polynomials in $R[x_i]$ by evaluating all variables except x_i at a randomly chosen point

$$\beta = (\beta_1, \ldots, \beta_{k-1}) \in [0, p)^{k-1}.$$

The point β is chosen so that $\mathrm{lc}(H, x_i)(\beta)$ is invertible in R and no leading coefficient vanishes under this evaluation; that is,

$$\mathrm{lc}(f_1, x_i)(\beta) \neq 0, \ \mathrm{lc}(f_2, x_i)(\beta) \neq 0, \ \mathrm{lc}(H, x_i)(\beta) \neq 0.$$

After evaluation, we obtain

$$A = f_1(x_i, \beta), \qquad B = f_2(x_i, \beta), \qquad C = H(x_i, \beta)$$

as polynomials in $R[x_i]$. The algorithm then checks whether C divides both A and B in $R[x_i]$. If this divisibility test succeeds, DT chooses the next variable and does the same process. If the test fails for some variable x_i, then DT returns **False**, and control returns to the main algorithm either PGCDNF or MGCDNF. If DT returns **True**, it means all k divisibility tests have succeeded, and H is accepted as the monic GCD of f_1 and f_2 with high probability. Because this verification step is randomized and may accept an incorrect candidate, PGCDNF and MGCDNF are **Monte Carlo** algorithms. We state the following lemma without proof.

Lemma 4.4.1. *Algorithm DT costs*

$$\mathcal{O}(k(\underbrace{d^2D^2}_{\text{division in } \bar{L}_p[x_i]} + \underbrace{dD^k}_{\text{evaluation}}))$$

arithmetic operations in $\mathbb{Z}_p$.

Alternatively, one may perform the divisibility test for PGCDNF (MGCDNF) directly in $\bar{L}_p[x_1, \ldots, x_k]$ ($L_p[x_1, \ldots, x_k]$) rather than after specialization to $\bar{L}_p[x_i]$ ($L_p[x_i]$). This makes PGCDNF and MGCDNF **Las Vegas algorithm**, since correctness is then guaranteed. However, this guarantee comes at a significantly higher computational cost, because multivariate division may dominate the overall complexity. In this case, the division test costs

$$\mathcal{O}(d^2(D+1)^{2k})$$

arithmetic operations in $\mathbb{Z}_p$.

Algorithm 6: DT (Divisibility Test)

Input: Polynomials $f_1, f_2, H \in R[x_1, \ldots, x_k]$, and p s.t. R has characteristic p.
Output: Either False or True

for $i = 1, \ldots, k$ **do**
  Choose $\beta = (\beta_1, \ldots, \beta_{k-1})$, where $\beta_i \in \mathbb{Z}_p$, at random until $\mathrm{lc}(H, x_i)(\beta), \mathrm{lc}(f_1, x_i)(\beta), \mathrm{lc}(f_2, x_i)(\beta) \neq 0$ and $\mathrm{lc}(H, x_i)(\beta)$ is invertible over R.
  $A = f_1(x_i, \beta_1, \ldots, \beta_{k-1}) \in R[x_i]$
  $B = f_2(x_i, \beta_1, \ldots, \beta_{k-1}) \in R[x_i]$
  $C = H(x_i, \beta_1, \ldots, \beta_{k-1}) \in R[x_i]$
  if $C \nmid A$ *or* $C \nmid B$ **then**
    return(False)
return(True)

4.4.2 PGCDNF

Algorithm **PGCDNF** (Algorithm 7) computes the monic GCD of two polynomials $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ with high probability, where $k \geq 1$. It follows Brown's approach, using evaluation and dense interpolation. PGCDNF is a recursive algorithm. When $k = 1$, it calls the MEA to compute $\gcd(f_1, f_2) \in \bar{L}_p[x_1]$. Otherwise, it reduces f_1 and f_2 to polynomials in $\bar{L}_p[x_1, \ldots, x_{k-1}]$ by evaluating $x_k = \beta_k$, where β_k is chosen randomly from $[0, p)$. It then recursively computes

$$\gcd(f_1(x_1, x_2, \ldots, x_{k-1}, \beta_k), f_2(x_1, x_2, \ldots, x_{k-1}, \beta_k)).$$

Subsequently, PGCDNF interpolates x_k in g. The interpolation is performed incrementally until the interpolated polynomial H does not change. This termination condition is enforced in Line 29. The PGCDNF algorithm does not guarantee the correct reconstruction of the monic GCD for all choices of evaluation points and primes. Within the PGCDNF algorithm, we identify four types of evaluation points and primes: **lc-bad pairs**, **zero divisor pairs**, **unlucky evaluation points**, and **good pairs**. The definition of good pairs is deferred to Section 4.4.3.

Definition 4.4.1. *(**Lc-bad pair**) Let $f_1, f_2 \in L[x_1, \ldots, x_k]$ be two polynomials with $\deg(f_1) \geq \deg(f_2) \geq 0$. Let p be a prime, and $\beta \in [0, p)^{k-1}$ be an evaluation point. We call the ordered pair (p, β) **lc-bad** if $p \mid \prod_{i=1}^{n} \operatorname{lc}(\check{M}_i, z_i)$ or $p \mid \operatorname{lc}(\check{f}_1, x_1)(\beta) \cdot \operatorname{lc}(\check{f}_2, x_1)(\beta)$.*

Example 4.4.2. *Let $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$ and*

$$f_1 = (x_2 + z_1)x_1^3 + x_1 z_2 \quad \text{and} \quad f_2 = (x_2 + 1)x_1^2 + z_1 z_2 x_1$$

be two polynomials in $L[x_1, x_2]$ in lexicographic order with $x_1 > x_2$. In this example, $f_1 = \check{f}_1$ and $f_2 = \check{f}_2$. The ordered pair $(p, \beta) = (7, 6)$ is lc-bad since $7 \mid \operatorname{lc}(\check{f}_2, x_1)(6)$.

Remark 4.3. *For the correctness of our modular algorithm, it is sufficient to choose (p, β) s.t.*

$$p \nmid \prod_{i=1}^{n} \operatorname{lc}(\check{M}_i, z_i) \quad \text{and} \quad (p \nmid \operatorname{lc}(\check{f}_1, x_1)(\beta) \text{ or } p \nmid \operatorname{lc}(\check{f}_2, x_1)(\beta)).$$

In other words, it is enough that p does not divide at least one of the $\operatorname{lc}(\check{f}_1, x_1)(\beta)$ or $\operatorname{lc}(\check{f}_2, x_1)(\beta)$. Under this weaker assumption, Lemma 2.2.9 still holds, and all the unlucky homomorphisms can be detected. However, to simplify the failure probability analysis in Chapter 6, we impose the stronger condition as in Definition 4.4.1.

Definition 4.4.2. *(**zero divisor pair**) Let p be a prime and let $\beta \in [0, p)^{k-1}$ be an evaluation point s.t. (p, β) is not an lc-bad pair. We call the ordered pair (p, β) a **zero divisor pair** if the MEA (Algorithm 2) returns FAIL when it is invoked by PGCDNF over $\bar{L}_p[x_1, \ldots, x_k]$.*

In PGCDNF, MEA is invoked in Lines 2, 5, 6, 7, 9, and 30. Therefore, a zero divisor pair can cause PGCDNF to fail at any of these steps when MEA tries to invert a leading coefficient.

Example 4.4.3. *Let*

$$f_2 = (x_2 + x_3 + 8z + 4)x_1 + z x_2 x_3 \in \mathbb{Q}[z]/\langle z^2 - 2\rangle[x_1][x_2, x_3].$$

Take $p = 7$ and $\beta = (0, 0)$ for (x_2, x_3). Then

$$\operatorname{lc}(f_2, x_1)(\beta) \mod p = z + 4$$

is not invertible over $\mathbb{Z}_7[z]/\langle z^2 - 2\rangle$ because $\gcd(z^2 - 2, z + 4) = z + 4 \neq 1$ in $\mathbb{Z}_7[z]$. Thus, in Line 2 of PGCDNF, the MEA cannot make f_2 monic and returns FAIL, so $(7, (0, 0))$ is a zero divisor pair.

Definition 4.4.3. *(**Unlucky evaluation point**) Let $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ with $\deg(f_1, x_k) \geq \deg(f_2, x_k) \geq 0$.*

Algorithm 7: PGCDNF

Input: non-zero polynomials $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ in lexicographic order with $x_1 > \ldots > x_k$
Output: Either the monic $\gcd(f_1, f_2) \in \bar{L}_p[x_1, \ldots, x_k]$ or FAIL

1 **if** $k = 1$ **then**
2 $H = \gcd(f_1, f_2) \in \bar{L}_p[x_1]$ s.t. $\mathrm{lc}(H) = 1$// By using the MEA.
3 **return**(H)
4 $X_k = [x_1, \ldots, x_{k-1}]$
5 $c_1 = \mathrm{cont}(f_1, X_k) \in \bar{L}_p[x_k]$. **if** $c_1 = FAIL$ **then return**(FAIL)
6 $c_2 = \mathrm{cont}(f_2, X_k) \in \bar{L}_p[x_k]$. **if** $c_2 = FAIL$ **then return**(FAIL)
7 $c = \gcd(c_1, c_2) \in \bar{L}_p[x_k]$. **if** $c = FAIL$ **return**(FAIL)
8 $f_1, f_2 = f_1/c_1, f_2/c_2$
9 $\Gamma = \gcd(lc(f_1, X_k), lc(f_2, X_k)) \in \bar{L}_p[x_k]$. **if** $\Gamma = FAIL$ **return**(FAIL)
10 $prod = 1$
11 **while** *true* **do**
12 Take a new evaluation point $\beta_k \in [0, p)$ at random s.t. $\mathrm{lc}(f_2, X_k)(x_k = \beta_k) \neq 0$ and $\mathrm{lc}(f_1, X_k)(x_k = \beta_k) \neq 0$.// $\mathrm{lc}(f_1, X_k), \mathrm{lc}(f_2, X_k) \in \bar{L}_p[x_k]$ and (p, β) is not lc-bad
13 $F_{1_k} = f_1(x_1, \ldots, x_{k-1}, x_k = \beta_k)$ and $F_{2_k} = f_2(x_1, \ldots, x_{k-1}, x_k = \beta_k)$
14 $G_k = PGCDNF(F_{1_k}, F_{2_k}, p) \in \bar{L}_p[x_1, \ldots, x_{k-1}]$
  // $\mathrm{lc}(G_k) = 1$ in lex order with $x_1 > x_2 > \ldots > x_{k-1}$.
15 **if** $G_k = FAIL$ **then**
16 **return**(FAIL)
17 $lm = \mathrm{lm}(G_k, X_k)$ // In lex order with $x_1 > x_2 > \ldots > x_{k-1}$.
18 $\Gamma_k = \Gamma(\beta_k) \in \mathbb{Z}_p$
19 $g_k = \Gamma_k \cdot G_k$ // Solve the leading coefficient problem.
20 **if** $prod = 1$ *or* $lm < least$ **then**
  // First iteration or all previous evaluation points were unlucky.
21 $least,\ H,\ prod = lm,\ g_k,\ x_k - \beta_k$
22 **else**
23 **if** $lm > least$ **then**
  // β_k is an unlucky evaluation point.
24 Go back to Line 12.
25 **else**
  // $lm = least$, so interpolate x_k in the GCD H incrementally.
26 $V_k = prod(x_k = \beta_k)^{-1} \cdot (g_k - H(x_k = \beta_k))$
27 $H = H + V_k \cdot prod$
28 $prod = prod \cdot (x_k - \beta_k)$
29 **if** $\deg(prod, x_k) > \deg(H, x_k) + 1$ **then**
  // Make H primitive in $\bar{L}_p[x_k][x_1, \ldots, x_{k-1}]$.
30 $c_3 = \mathrm{cont}(H, X_k)$. **if** $c_3 = FAIL$ **then return**(FAIL) **else** $H = H/c_3$.
  // Employing Algorithm 6 for $f_1, f_2, H \in \bar{L}_p[x_1, \ldots, x_k]$, we test if $H \mid f_1$ and $H \mid f_2$.
31 **if** $DT(f_1, f_2, H, p) =$true **then return**$(c \cdot H)$

- *Let the monic* $g = \gcd(f_1, f_2)$ *exist, and* h_1 *and* h_2 *denote the cofactors of* f_1 *and* f_2*, respectively.*
- *Set* $X_k = [x_1, \ldots, x_{k-1}]$*, so* $\mathrm{lc}(f_1, X_k), \mathrm{lc}(f_2, X_k) \in \bar{L}_p[x_k]$.
- *Let* $\beta_k \in [0, p)$ *s.t.* $\mathrm{lc}(f_1, X_k)(\beta_k) \neq 0$ *and* $\mathrm{lc}(f_2, X_k)(\beta_k) \neq 0$.

- *Let* $g_{\beta_k} = \gcd(f_1(x_k = \beta_k), f_2(x_k = \beta_k))$ *exist.*

We call $x_k = \beta_k$ *an* ***unlucky evaluation point*** *if*

$$\deg(\gcd(h_1(x_k = \beta_k), h_2(x_k = \beta_k))) > 0. \tag{4.2}$$

Example 4.4.4. *Let* $\bar{L}_{11} = \mathbb{Z}_{11}[z]/\langle z^2 + 8\rangle$ *and let* $g = (x_2 + 2z)x_1$,

$$f_1 = \underbrace{(x_1 + z + 4x_2 + 8)}_{h_1} \cdot g, \text{ and}$$
$$f_2 = \underbrace{(x_1 + 2x_2 + z + 10)}_{h_2} \cdot g$$

be polynomials in $\bar{L}_{11}[x_1, x_2]$. *We use lexicographic order with* $x_1 > x_2$. *By inspection,*

$$h_1 = x_1 + z + 4x_2 + 8 \text{ and}$$
$$h_2 = x_1 + 2x_2 + z + 10$$

are the cofactors of f_1 *and* f_2, *respectively. Evaluating* h_1 *and* h_2 *at* $x_2 = \beta_2 = 1$, *we obtain*

$$g_{\beta_2} = \gcd(h_1(x_2 = 1), h_2(x_2 = 1)) = x_1 + z + 1.$$

Since $\deg(g_h) > 0$, *we conclude that* $x_2 = 1$ *is an unlucky evaluation point.*

Remark 4.4. *The following observations highlight key considerations regarding the structure of the PGCDNF algorithm:*

(i) If prime p *is chosen to be sufficiently large, the probability of PGCDNF returning FAIL is low.*

(ii) In Algorithm MGCDNF, if PGCDNF tries to invert a zero divisor in $\bar{L}_p$, *PGCDNF is aborted, and control is returned to MGCDNF. Then MGCDNF selects a new prime and calls PGCDNF again for the updated prime.*

(iii) Since $\mathrm{lm}(g)$ *is not known in advance, identifying unlucky evaluation points is non-trivial. To address this, similar to Brown's PGCD, we keep only those images* g_{β_i} *with the least leading monomial and discard the others (see Lines 20, 23, and 25 of Algorithm 7 (PGCDNF)).*

Let $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ and $X_k = [x_1, \ldots, x_{k-1}]$. The PGCDNF algorithm uses the property

$$\gcd(f_1, f_2) = \gcd(\mathrm{cont}(f_1, X_k), \mathrm{cont}(f_2, X_k)) \cdot \gcd(\mathrm{pp}(f_1, X_k), \mathrm{pp}(f_2, X_k)).$$

For $k > 1$, the PGCDNF algorithm recursively computes monic images of the GCD in $\bar{L}_p[x_1, \ldots, x_{k-1}]$. Let $\beta_2, \ldots, \beta_k \in [0, p)$ denote the evaluation points selected by PGCDNF. To recover the leading coefficient of g in x_k, we follow Brown's algorithm [6] and scale by

$$\Gamma(x_k = \beta_k) = \gcd(\mathrm{lc}(f_1, X_k), \mathrm{lc}(f_2, X_k))(x_k = \beta_k),$$

evaluated at the current evaluation point $x_k = \beta_k$. Thus, after interpolating the GCD H, we have $\mathrm{lc}(H, X_k) = \Gamma(x_k)$. The interpolation of x_k in PGCDNF (lines 26–28) is based on the Newton form

for H, namely,

$$H = V_1 + V_2(x_k - \beta_1) + \cdots + V_j \prod_{i=1}^{j-1}(x_k - \beta_i),$$

where $V_i \in \bar{L}_p[x_1, \ldots, x_{k-1}]$ for $1 \leq i \leq k$. To compute the new H from the previous one, it suffices to compute the Newton coefficient V_k. In the final phase of PGCDNF, we verify whether the primitive part of H is the GCD of $\text{pp}(f_1, X_k)$ and $\text{pp}(f_2, X_k)$. To do so, we employ the division test for k univariate polynomials in $\bar{L}_p[x_i]$ where $1 \leq i \leq k$, as presented in Algorithm 6. This strategy makes PGCDNF a **Monte Carlo algorithm**. Alternatively, if the division test is performed directly in $\bar{L}_p[x_1, \ldots, x_k]$ instead of in $\bar{L}_p[x_1]$, the algorithm becomes a **Las Vegas algorithm**, guaranteeing correctness, but at the cost of significantly higher computational complexity, since multivariate division in $\bar{L}_p[x_1, \ldots, x_k]$ may dominate the overall complexity.

4.4.3 MGCDNF

The MGCDNF algorithm, as presented in Algorithm 8, is a Monte Carlo algorithm for computing the monic $g = \gcd(f_1, f_2)$, where $f_1, f_2 \in L[x_1, \ldots, x_k]$. MGCDNF begins with a preprocessing step where the input polynomials, f_1, f_2, and the minimal polynomials $M_1, \ldots, M_n$ are replaced with their semi-associates. By canceling any rational scalar from the inputs, the MGCDNF algorithm converts the polynomials from $L[x_1, \ldots, x_k]$ to polynomials in a simpler domain $L_{\mathbb{Z}}[x_1, \ldots, x_k]$. This strategy makes the modular homomorphism ϕ_p faster.

The algorithm proceeds by selecting a prime p s.t. $p \nmid \text{lc}(\check{f}_1) \cdot \text{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \text{lc}(\check{M}_i, z_i)$ and applying ϕ_p to reduce the coefficients of the input polynomials from L to L_p. Subsequently, it employs the isomorphism ϕ_γ to convert the polynomials $\check{f}_1$ and $\check{f}_2$ over L_p to their corresponding polynomials over $\bar{L}_p$. Then MGCDNF calls PGCDNF to compute the monic GCD in $\bar{L}_p[x_1, \ldots, x_k]$.
Let G_p denote the output of PGCDNF. If $G_p = FAIL$, then PGCDNF has encountered a zero divisor pair in one of the lines 2, 5, 6, 7, 9, or 30. In that case, MGCDNF returns to Line 5 to choose a new prime. In Line 15, $G_p \in \bar{L}_p[x_1, \ldots, x_k]$ is converted to its corresponding polynomial over L_p. Applying Lemma 2.2.9, MGCDNF retains only those images G_p with the least leading monomial for Chinese remaindering. For instance, let G_{p_i} be the output of PGCDNF at the ith iteration, and $\text{lm}(G_{p_i}) > \text{lm}(G_{p_{i-1}})$; then ϕ_{p_i} is an unlucky homomorphism, and we simply ignore its result G_{p_i} and choose another prime, similar to what we did in Example 2.2.19 in Section 2.2.8 (also see Lemma 2.2.9).

After Chinese remaindering, MGCDNF employs rational number reconstruction (RNR)[38, 25] to recover the rational coefficients of g in L. If the RNR step fails, it indicates that the product of the primes used so far is not large enough to reconstruct the rational coefficients. In this case, additional modular images are required to complete the reconstruction of the GCD. Otherwise, if RNR succeeds, MGCDNF follows the same verification strategy as PGCDNF to confirm whether the reconstructed polynomial is indeed the GCD of f_1 and f_2. As mentioned MGCDNF is a Monte Carlo algorithm, which means it can return a wrong answer with low probability. Example 4.4.5 illustrates when MGCDNF returns a wrong result.

Algorithm 8: MGCDNF

Input: non-zero $f_1, f_2 \in L[x_1, \ldots, x_k]$ where $L = \mathbb{Q}[z_1, \ldots, z_n]/\langle M_1(z_1), \ldots, M_n(z_n)\rangle$, $\mathrm{lc}(f_1, x_1) \geq \mathrm{lc}(f_2, x_1)$ w.r.t. lexicographic order with $x_1 > x_2 \ldots > x_k$, and the set $\mathbb{P}_b = \{\text{all } \ b - \text{bit primes}\}$.

Output: The monic $\gcd(f_1, f_2)$.

1 $presult = 0$

2 $P = 1$

3 $f_1 = \check{f}_1$ and $f_2 = \check{f}_2$ // Clear fractions from the input polynomials.

4 **while** *true* **do**

5 Choose a new random prime $p \in \mathbb{P}_b$ s.t. $p \nmid \mathrm{lc}(\check{f}_1) \cdot \mathrm{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i)$.

6 Choose $C_1, \ldots, C_{n-1} \in [1, p)$ at random and set $\gamma = z_1 + \sum_{i=2}^{n} C_{i-1} z_i$

7 Call Algorithm 4 with inputs $[\phi_p(\check{M}_1), \ldots, \phi_p(\check{M}_n)]$, $\mathbb{Z}_p$ and $\phi_p(\gamma)$ to compute $M(z)$, A, and A^{-1}

8 **if** *Algorithm 4 fails* **then**

9 Go back to Line 5

// Applying Algorithm 7, we compute the monic GCD over $\bar{L}_p$.

10 $G_p = PGCDNF(\phi_\gamma(\phi_p(f_1)), \phi_\gamma(\phi_p(f_2))) \in \bar{L}_p[x_1, \ldots, x_k]$

11 **if** G_p= *FAIL* **then**

// PGCDNF has encountered a zero divisor pair, so we must choose a new prime.

12 Go back to Line 5.

13 **if** $\deg(G_p) = 0$ **then**

// Applying Lemma 2.2.9.

14 **return**(1)

// Convert $G_p \in \bar{L}_p$ to its corresponding polynomial over L_p before using the CRT and RNR.

15 $G_p = \phi_\gamma^{-1}(G_p)$

16 $lm = \mathrm{lm}(G_p)$

17 **if** $P = 1$ *or* $lm < least$ // First iteration or all the previous primes were unlucky.

18 **then**

19 G, $least$, $P = G_p$, lm, p

20 **else**

21 **if** $lm = least$ **then**

22 Using CRT, compute $G' \equiv G \mod P$ and $G' \equiv G_p \mod p$

23 Set $G = G'$ and $P = P \cdot p$

24 **else if** $lm > least$ **then**

// p is an unlucky prime or all β's used in PGCDNF were unlucky.

25 Go back to Line 5

26 H = Rational Number Reconstruction of G mod P// See Theorem 2.2.10.

27 **if** $H \neq$ *FAIL* **then**

28 **if** $H = presult$ **then**

// Two successive RNR results are the same, so we proceed with the division test.

29 Choose a new prime $q \in \mathbb{P}_b$ s.t $q \nmid \mathrm{lc}(\check{f}_1) \cdot \mathrm{lc}(\check{f}_2)$ and $q \nmid \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i)$ and compute $\phi_q(f_1)$, $\phi_q(f_2)$, $\phi_q(H) \in L_q[x_1, \ldots, x_k]$ // call Algorithm 6 for $f_1, f_2, H \in L_q[x_1, \ldots, x_k]$

30 **if** $DT(\phi_q(f_1), \phi_q(f_2), \phi_q(H), q) =$ true **then return**(H)

31 **else**

32 $presult = H$

Example 4.4.5. *Let p_1 and p_2 be two distinct primes. Let*

$$\begin{aligned} g &= x_1^2 + x_2 + (p_1 \cdot p_2)x_1x_2 + z_1z_2 \\ f_1 &= (x_1 + z_2) \cdot g \\ f_2 &= (x_2 + z_1) \cdot g \end{aligned}$$

be polynomials in $\mathbb{Q}[z_1, z_2]/\langle M_1, M_2\rangle[x_1, x_2]$ where $M_1(z_1) = z_1^2 - 2$ and $M_2(z_2) = z_2^2 - 3$. By inspection $g = \gcd(f_1, f_2)$. In Line 5 of MGCDNF, if $p = p_1$ is chosen, then RNR may return

$$H = x_1^2 + x_2 + z_1z_2.$$

In the next iteration, if MGCDNF chooses $p = p_2$, again RNR may return

$$H = x_1^2 + x_2 + z_1z_2.$$

Since the two reconstructed results are identical, MGCDNF runs Algorithm 6 to verify whether $H \mid f_1$ and $H \mid f_2$. For the pass with main variable x_2, suppose Algorithm 6 evaluates x_1 at 0. Thus

$$\begin{aligned} f_1(x_1 = 0) &= z_2(x_2 + z_1z_2) \\ f_2(x_1 = 0) &= (x_2 + z_1)(x_2 + z_1z_2) \\ H(x_1 = 0) &= (x_2 + z_1z_2). \end{aligned}$$

Therefore, $H(x_1 = 0) \mid f_1(x_1 = 0)$ and $H(x_1 = 0) \mid f_2(x_1 = 0)$. For the pass with main variable x_1, suppose Algorithm 6 evaluates x_2 at 0 so

$$\begin{aligned} f_1(x_2 = 0) &= (x_1 + z_2)(x_1^2 + z_1z_2) \\ f_2(x_2 = 0) &= z_1(x_1^2 + z_1z_2) \\ H(x_2 = 0) &= (x_1^2 + z_1z_2). \end{aligned}$$

Therefore, $H(x_2 = 0) \mid f_1(x_2 = 0)$ and $H(x_2 = 0) \mid f_2(x_2 = 0)$. In this case, Algorithm 6 returns true and MGCDNF returns $H = x_1^2 + x_2 + z_1z_2$ as the monic $\gcd(f_1, f_2)$ which is not the correct GCD g.

Remark 4.5. *For the efficiency of the MGCDNF algorithm, it is essential to apply the homomorphism ϕ_p prior to ϕ_γ. This ordering prevents expression swell over $\mathbb{Q}$. See Section 4.4.4 for further details. Moreover, it is essential to apply ϕ_γ^{-1} before CRT and RNR to avoid the reconstruction of large coefficients.*

As mentioned, in Line 5 of MGCDNF, a prime p is selected to reduce the input polynomials modulo p. However, not all primes lead to a successful reconstruction of g in MGCDNF. We classify the primes used in MGCDNF into two categories: **det-bad pairs** and **unlucky primes**.

Let A be the change-of-basis matrix obtained from the LAminpoly (Algorithm 4) over $\mathbb{F} = \mathbb{Q}$. Then

$$\det(A) = \frac{N}{D} \in \mathbb{Q},$$

where $D \neq 0$ and $\gcd(N, D) = 1$. We define $\text{num}(det(A)) = N$ and $\text{den}(det(A)) = D$. In Chapter 6 (Corollary 3.1), we prove that

$$D \mid \prod_{i=1}^{n} \text{lc}(\check{M}_i, z_i).$$

Hence, if $p \nmid \prod_{i=1}^{n} \text{lc}(\check{M}_i, z_i)$, then $p \nmid D$. If a prime p divides $\text{num}(\det(A))$, then the LAminpoly fails over $\mathbb{F} = \mathbb{Z}_p$. In this case, we cannot construct $\bar{L}_p \cong L_p$. This motivates the following definition.

Definition 4.4.4. *(**Det-bad pair**) Let p be a prime s.t. $p \nmid \prod_{i=1}^{n} \text{lc}(\check{M}_i, z_i)$. Let $C = (C_1, \ldots, C_{n-1}) \in [1, p)^{n-1}$ and $\gamma = z_1 + \sum_{i=2}^{n} C_{i-1} z_i$. Let A be the change-of-basis matrix obtained from the LAminpoly (Algorithm 4) over $\mathbb{F} = \mathbb{Q}$. The ordered pair (p, C) is called a **det-bad** pair if* $\det(A) = 0$ *or* $p \mid \text{num}(\det(A))$.

If Algorithm MGCDNF encounters a det-bad pair (p, C), it goes back to Line 5 and selects a new prime. It then chooses a new C in Line 6.

Example 4.4.6. *Let $L = \mathbb{Z}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$ with a basis $B_L = \{1, z_1, z_2, z_1 z_2\}$. Let $C = (2)$ so $\gamma = z_1 + 2z_2$. Algorithm 4 computes the change-of-basis matrix*

$$A = \begin{bmatrix} 1 & 0 & 14 & 0 \\ 0 & 1 & 0 & 38 \\ 0 & 2 & 0 & 36 \\ 0 & 0 & 4 & 0 \end{bmatrix}$$

over $\mathbb{F} = \mathbb{Q}$ with $\det(A) = 160 = 2^5 \times 5$. Thus, the pairs $(p, C) = (5, (2))$ and $(p, C) = (2, (2))$ are det-bad since $p \mid \det(A)$ when $\gamma = z_1 + 2z_2$.

Definition 4.4.5. *(**Unlucky prime**)*

Let $f_1, f_2 \in L[x_1, \ldots, x_k]$ be non-zero polynomials, and let h_1 and h_2 denote the cofactors of f_1 and f_2, respectively. Let p be a prime number s.t.

- $p \nmid \text{lc}(\check{f}_1) \cdot \text{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \text{lc}(\check{M}_i)$,
- $p \nmid \det(A)$, *and*
- $\gcd(\phi_\gamma(\phi_p(\check{f}_1)), \phi_\gamma(\phi_p(\check{f}_2)))$ *exists.*

*Then p is called an **unlucky prime** if*

$$\deg(\gcd(\phi_p(\check{h}_1), \phi_p(\check{h}_2)) > 0.$$

Example 4.4.7. *Let $L = \mathbb{Q}[z]/\langle z^2 - 2\rangle$ and*

$$f_1 = (x + z) \underbrace{(5x + 2y + z)}_{h_1} \quad \text{and}$$

$$f_2 = (x + z) \underbrace{(5x + 9y + z)}_{h_2}$$

be two polynomials in $L[x, y]$. *By inspection,* $\gcd(f_1, f_2) = x + z$, $\check{f}_1 = f_1$, *and* $\check{f}_2 = f_2$. *Let* $p = 7$ *so*

$$g_p = \gcd(\phi_p(\check{h}_1), \phi_p(\check{h}_2)) = x + 6y + 3z \neq 1.$$

Since $\deg(g_p) > 0$, *the prime* $p = 7$ *is an unlucky prime.*

Definition 4.4.6. *Let* p *be a prime,* $C = (C_1, \ldots, C_{n-1}) \in (0, p)^{n-1}$, *and* $\beta = (\beta_1, \ldots, \beta_{k-1}) \in [0, p)^{k-1}$. *We call* (p, β) *a good pair if the following conditions are satisfied:*

- (p, β) *is neither an lc-bad pair nor a zero divisor pair,*
- β_i *is not unlucky for* $1 \leq i \leq k - 1$,
- p *is not unlucky.*

Moreover, (p, C) *is a good pair if it is not det-bad.*

Remark 4.6. *Although lc-bad can be efficiently ruled out in advance, we cannot detect either det-bad pairs, zero divisor pairs, unlucky evaluation points, or unlucky primes efficiently beforehand. Therefore, we end up calling the MEA in* $\bar{L}_p[x_1]$ *with zero divisor pairs, unlucky evaluation points, unlucky primes, and good pairs.*

Theorem 4.4.2. *Let* $\beta \in [0, p)^{k-1}$ *and* $\phi_\beta : \bar{L}_p[x_1][x_2, \ldots, x_k] \longrightarrow \bar{L}_p[x_1]$ *be an evaluation homomorphism where* p *is a prime. Let* $f_1, f_2 \in \bar{L}_p[x_1][x_2, \ldots, x_k]$. *Suppose that*

- $g = \mathit{monic}(\gcd(f_1, f_2))$,
- $g_\beta = \mathit{monic}(\gcd(\phi_\beta(f_1), \phi_\beta(f_2)))$, *and*
- $h = \mathit{monic}(\phi_\beta(g))$

all exist. If (p, β) *is a good evaluation pair, then* $h = g_\beta$.

Proof. If (p, β) is a good evaluation pair, then it is not lc-bad. Thus, we can infer that $\phi_\beta(\mathrm{lc}(f_2)) \neq 0$. By a similar argument as in the proof of Lemma 2.2.9, we can conclude that h is a common factor of $\phi_\beta(f_1)$ and $\phi_\beta(f_2)$ so $h \mid g_\beta$. In other words, there is a non-zero polynomial $t \in \bar{L}_p[x_1]$ s.t. $g_\beta = t \cdot h$. Since h is monic, the same justification in Lemma 2.2.9 leads us to conclude that $\mathrm{lm}(g_\beta) \geq \mathrm{lm}(h)$. On the other hand, by the definition of a good evaluation point, β is not unlucky. Thus, we can conclude that $\mathrm{lm}(g_\beta) = \mathrm{lm}(h)$. Finally, by part (ii) of Lemma 2.2.9, we have $h = g_\beta$. □

As mentioned, the MGCDNF algorithm is a Monte Carlo algorithm. In other words, if the MGCDNF algorithm terminates successfully and outputs H, then with high probability H is the monic GCD of the input polynomials. As in PGCDNF, Algorithm MGCDNF can be turned into a Las Vegas algorithm by performing the trial division in $L[x_1, \ldots, x_k]$ (rather than in $L_q[x_i]$) at Line 30. This modification guarantees correctness, but it can significantly increase the computational cost because multivariate division in $L[x_1, \ldots, x_k]$ may dominate the overall complexity. In the following theorem, we prove the correctness of MGCDNF if the trial division is performed in $L[x_1, \ldots, x_k]$.

Theorem 4.4.3. *Let* $f_1, f_2 \in L[x_1, \ldots, x_k]$, *and let* $g = \gcd(f_1, f_2)$ *be the monic GCD. Assume that in MGCDNF rational number reconstruction (RNR) succeeds and returns the polynomial* $H \in L[x_1, \ldots, x_k]$ *in Line 26. If* $H \mid f_1$ *and* $H \mid f_2$, *then* $H = g$.

Proof. Since $H \mid f_1$ and $H \mid f_2$, it follows that $H \mid g$. Hence, there exists $h \in L[x_1, \ldots, x_k]$ s.t. $g = H \cdot h$. Because both H and g are monic, h is monic as well. Moreover, for any monomial order,

$$\operatorname{lm}(g) = \operatorname{lm}(H) \operatorname{lm}(h). \tag{4.3}$$

In MGCDNF, lc-bad pairs, det-bad pairs, and zero divisor pairs are explicitly discarded in Lines 5, 8, and 11. Consequently, the CRT and RNR steps are applied only to primes $p_1, \ldots, p_N$ that pass these tests. Since MGCDNF employs CRT for the images with the same leading monomial (see Line 22), the primes used for CRT are either all lucky or all unlucky. For $1 \leq i \leq N$, define

$$G_i = \gcd(\phi_\gamma(\phi_{p_i}(f_1)), \phi_\gamma(\phi_{p_i}(f_2))) \in L_{p_i}[x_1, \ldots, x_k],$$

so that $\operatorname{lm}(G_i) = \operatorname{lm}(G_j)$ for $1 \leq i, j \leq N$. Suppose, for contradiction, that all primes used for the CRT are unlucky. Then Lemma 2.2.9 implies $\operatorname{lm}(G_i) > \operatorname{lm}(g)$ for each $1 \leq i \leq N$. Thus, $\operatorname{lm}(H) = \operatorname{lm}(G_i) > \operatorname{lm}(g)$ contradicts Equation (4.3). Therefore, the primes $p_1, \ldots, p_N$ must all be lucky, which implies that

$$\operatorname{lm}(H) = \operatorname{lm}(G_i) = \operatorname{lm}(g).$$

Since $g = H \cdot h$, the equality $\operatorname{lm}(H) = \operatorname{lm}(g)$ forces $\operatorname{lm}(h) = 1$. Since h is monic, this implies $h = 1$ and hence $H = g$ as required. □

4.4.4 Eliminating an expression swell

The order in which the homomorphisms ϕ_p and ϕ_γ are applied in the MGCDNF algorithm is critical for efficiency. Applying ϕ_γ before ϕ_p, in Line 10 of Algorithm MGCDNF, leads to significant expression swell. This increases both the computational time and memory usage of the algorithm. The following example illustrates how the choice of order directly impacts the implementation.

Example 4.4.8. *Let $f_1, f_2 \in \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3, \alpha_4)[x]$ with the monic GCD*

$$g = x^2 + \frac{3}{5}\alpha_1\alpha_2\alpha_3\alpha_4,$$

where the algebraic numbers are defined as:

- $\alpha_1 = \sqrt{2}$, *with minimal polynomial* $M_1 = z_1^2 - 2 \in \mathbb{Q}[z_1]$,
- $\alpha_2 = \sqrt{7} \in \mathbb{Q}(\alpha_1)$, *with minimal polynomial* $M_2 = z_2^2 - 7 \in \mathbb{Q}(\alpha_1)[z_2]$,
- $\alpha_3 = \sqrt{5} \in \mathbb{Q}(\alpha_1, \alpha_2)$, *with minimal polynomial* $M_3 = z_3^2 - 5 \in \mathbb{Q}(\alpha_1, \alpha_2)[z_3]$, *and*
- $\alpha_4 = \sqrt{3} \in \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3)$, *with minimal polynomial* $M_4 = z_4^2 - 3 \in \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3)[z_4]$.

Let $p = 307$, and define a primitive element

$$\gamma = s + 13w + 7t + 15k.$$

The minimal polynomial of γ over $\mathbb{Q}$ is

$$M(z) = z^{16} + 45z^{14} + 40z^{12} + 306z^{10} + 216z^8 + 78z^6 + 15z^4 + 86z^2 + 81.$$

- ***Efficient method:*** *We first reduce the coefficients modulo p, then apply the primitive element isomorphism ϕ_γ into g:*

$$\phi_p(g) = x^2 + 62\,\alpha_1\alpha_2\alpha_3\alpha_4 \in \mathbb{Z}_{307}(\alpha_1,\alpha_2,\alpha_3,\alpha_4)[x],$$

$$\phi_\gamma(\phi_p(g)) = x^2{+}140\,z^{14}{+}117\,z^{12}{+}256\,z^{10}{+}258\,z^8{+}294\,z^6{+}137\,z^4{+}235\,z^2{+}224 \in \mathbb{Z}_{307}/\langle M(z)\rangle[x]$$

- ***Inefficient method:*** *If we apply the primitive element isomorphism ϕ_γ first, then reduce the coefficients modulo p, we have:*

$$\begin{aligned}\phi_\gamma(g) = x^2 &- \frac{68314714248333}{379193446721615} - \frac{15175229454477\,z^2}{5308708254102610} + \frac{521955011043\,z^4}{530870825410261} \\ &+ \frac{6364297761009\,z^6}{2654354127051305} - \frac{3560922217107\,z^8}{2654354127051305} + \frac{127689153066\,z^{10}}{530870825410261} \\ &+ \frac{2846690115201\,z^{12}}{1061741650820522} - \frac{11384008608303\,z^{14}}{5308708254102610} \in \mathbb{Q}[z]/\langle M(z)\rangle[x],\end{aligned}$$

$$\phi_p(\phi_\gamma(g)) = x^2{+}140\,z^{14}{+}117\,z^{12}{+}256\,z^{10}{+}258\,z^8{+}294\,z^6{+}137\,z^4{+}235\,z^2{+}224 \in \mathbb{Z}_{307}[z]/\langle M(z)\rangle[x]$$

As shown, applying ϕ_γ prior to ϕ_p results in extremely large rational coefficients in $\phi_\gamma(g)$. This significantly increases both the time and memory required for the computation, making the implementation far less efficient.

Furthermore, we call $\phi_{\gamma^{-1}}$ before recovering the rational coefficients of the GCD using the CRT and RNR. By doing so, we avoid large coefficients for the CRT and RNR resulting in a decrease in the time and memory used for CRT and RNR.

4.5 Complexity

Similar to Definition 2.3.1 in Chapter 2, we define the height of a polynomial in $L_{\mathbb{Z}}[x_1,\ldots,x_k]$.

Definition 4.5.1. *Given $f \in L_{\mathbb{Z}}[x_1,\ldots,x_k]$, we can represent $f = \sum_a C_a X^a$ as a polynomial over $\mathbb{Z}[z_1,\ldots,z_n,x_1,\ldots,x_k]$ where $X^a = \prod_{i=1}^n z_i^{a_i} \prod_{j=1}^k x_j^{b_j}$ s.t. $a_i, b_j \in \mathbb{Z}_{\geq 0}$. If we use this representation, we denote the height of f by $\|f\|_\infty$ and define it as*

$$H(f) = \|f\|_\infty = \max_a(|\ C_a\ |)$$

Definition 4.5.2. *Let $f = \sum_{i=1}^t C_i X^{a_i}$ be a polynomial in $L[x_1,\ldots,x_k]$ s.t. $C_i \in \mathbb{Z}[z_1,\ldots,z_n]$ and $X^{a_i} = \prod_{j=1}^k x_j^{a_{ij}}$ where $a_{i_j} \in \mathbb{Z}_{\geq 0}$. The number of terms of f, denoted by $\#f$, is defined as the number of distinct monomials in $x_1,\ldots,x_k$ with non-zero coefficients in L, which in this case equals t.*

Example 4.5.1. *Let $f \in L[x_1,x_2]$ where $L = \mathbb{Q}[z_1,z_2]/\langle z_1^2-2, z_2^2-3\rangle$. Let*

$$\check{f} = (5z_1+8z_2)x_1^2x_2 + (6z_1+12)x_2 + 4z_1z_2 \in L_{\mathbb{Z}}[x_1,x_2].$$

Then $H(\check{f}) = \|\check{f}\|_\infty = 12$, and the terms of $\check{f}$ are $(5z_1+8z_2)x_1^2x_2$, $(6z_1+12)x_2$, and $4z_1z_2$, so $\#\check{f} = 3$.

Notation 4.1. *Let $f_1, f_2 \in L[x_1,\cdots,x_k]$ and g be the monic $\gcd(f_1,f_2)$. The quantities involved in the running time of the MGCDNF algorithm are as follows:*

- N *is the number of good primes needed to reconstruct the monic GCD* g,
- $d = [L : \mathbb{Q}]$ *and* $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$,
- $H_M = \log_2 \max_{i=1}^{n} H(\check{M}_i)$, *and* $C = \log_2 \max(H(\check{f}_1), H(\check{f}_2))$.

Remark 4.7. *We assume that multiplication and inverses in* $\bar{L}_p$ *cost* $\mathcal{O}(d^2)$, *as our implementation currently uses classical quadratic polynomial arithmetic. Moreover, addition and scalar multiplication in* $\bar{L}_p = \mathbb{Z}_p[z]/\langle M \rangle$ *cost* $\mathcal{O}(d)$ *operations in* $\mathbb{Z}_p$.

Theorem 4.5.1. *With Notation 4.1, the expected time complexity of the PGCDNF algorithm is* $\mathcal{O}(k^2 d^2 D^{k+1})$ *arithmetic operations in* $\mathbb{Z}_p$.

Proof. Since the PGCDNF is a recursive algorithm, to compute its complexity, we use a recurrence relation $T(k)$. In the base case, when $k = 1$, PGCDNF calls the MEA to compute the monic GCD of two univariate polynomials in $\bar{L}_p[x_1]$. Since the degrees are bounded by D, this costs $\mathcal{O}(D^2)$ operations in $\bar{L}_p$. Each arithmetic operation in $\bar{L}_p$ costs $\mathcal{O}(d^2)$ operations in $\mathbb{Z}_p$. Therefore, $T(1) = \mathcal{O}(d^2 D^2)$. Before introducing $T(k)$ for $k > 1$, we compute the time complexity of the most dominant operations in the PGCDNF. We have $\#f_1, \#f_2 \leq (D+1)^k$.

1. First, consider the cost of computing contents. Represent f_1 and f_2 as two polynomials in $\bar{L}_p[x_k][x_1, \ldots, x_{k-1}]$. In the Lines 5, 6, 7, and 30 of the PGCDNF, to compute the contents w.r.t. $X_k = [x_1, \ldots, x_{k-1}]$, we must compute the monic GCD of the coefficients of f_1 and f_2. There are at most $(D+1)^{k-1}$ coefficients in $\bar{L}_p[x_k]$. Thus, the time complexity of computing the content is
$$\mathcal{O}((D+1)^{k-1}) \cdot \mathcal{O}(d^2 D^2) = \mathcal{O}(d^2 D^{k+1}). \tag{4.4}$$
Moreover, computing the primitive part of the input polynomials in Line 8 costs
$$\mathcal{O}(d^2 D^{k+1})$$
arithmetic operations in $\mathbb{Z}_p$.

2. Second, consider the evaluation step. Represent f_1 and f_2 as two polynomials in $\bar{L}_p[x_k][x_1, \ldots, x_{k-1}]$. Using Horner's rule, a polynomial in $\bar{L}_p[x_k]$ of degree at most D can be evaluated at $x_k = \beta_k$ using $\mathcal{O}(D)$ additions and scalar multiplications in $\bar{L}_p$. Each such operation costs $\mathcal{O}(d)$ operations in $\mathbb{Z}_p$. Moreover, each polynomial f_1 and f_2 has at most $(1+D)^{k-1}$ coefficients in $\bar{L}_p[x_k]$. A successful interpolation of x_k at g requires evaluating f_1 and f_2 at most at $D+1$ points in Line 13. Therefore, evaluating both input polynomials for at most $D+1$ evaluation points for x_k costs
$$\mathcal{O}((D+1)(D+1)^{k-1} d(2D)) \in \mathcal{O}(dD^{k+1}) \tag{4.5}$$
arithmetic operations in $\mathbb{Z}_p$.

3. Third, consider the interpolation step. To interpolate x_k at g, in the worst case, we need $D+1$ points. This happens when $g = \text{monic}(f_1)$ or $g = \text{monic}(f_2)$. Let $\beta_1, \ldots, \beta_{D+1}$ be the evaluation points and $g_1, \ldots, g_{D+1} \in \bar{L}_p[x_1, \ldots, x_{k-1}]$ be the values in Line 19. Newton's interpolation over a field e.g. $\mathbb{Z}_p$ does $\mathcal{O}((D+1)^2) \in \mathcal{O}(D^2)$ field operations. To interpolate x_k, recden treats g_i as

vectors in $\mathbb{Z}_p$. Hence, the size of g_i is at most $d(D+1)^{k-1}$ with elements in $\mathbb{Z}_p$ for $1 \le i \le D+1$. Therefore, we have $\mathcal{O}(D^2)$ scalar operations in $\bar{L}_p[x_1, \ldots, x_{k-1}]$ which costs at most

$$\mathcal{O}(D^2)d(D+1)^{k-1} \in \mathcal{O}(dD^{k+1}) \tag{4.6}$$

field operations in $\mathbb{Z}_p$.

4. Fourth, consider the cost of the divisibility test. After enough evaluation points have been used to construct a GCD candidate H, PGCDNF makes H primitive and calls DT (Algorithm 6) to test whether $H \mid f_1$ and $H \mid f_2$. By Lemma 4.4.1, the cost of DT is

$$O(k(d^2D^2 + dD^k)) \tag{4.7}$$

arithmetic operations in $\mathbb{Z}_p$.

Simplifying (4.4), (4.5), and (4.6) and adding them up, we have the recurrence relation as follows

$$\begin{aligned}
T(1) &= \mathcal{O}(D^2d^2) \quad \text{for } k = 1 \\
T(k) &= \underbrace{\mathcal{O}(d^2D^{k+1})}_{\text{Contents + Primitive parts (4.4)}} + \underbrace{\mathcal{O}(dD^{k+1})}_{\text{Evaluation (4.5)}} + \underbrace{\mathcal{O}(dD^{k+1})}_{\text{Interpolation(4.6)}} \quad \text{for } k > 1 \\
&\quad + \underbrace{\mathcal{O}(k(d^2D^2 + dD^k))}_{\text{Division test (4.7)}} + \underbrace{(D+1)}_{\#\text{ recursion calls}} T(k-1) \\
&= \mathcal{O}(d^2D^{k+1}) + \mathcal{O}(kd^2D^2 + kdD^k) + (D+1)T(k-1) \text{ for } k > 1.
\end{aligned}$$

Solving for $T(k)$, we have

$$T(k) = \mathcal{O}(k^2d^2D^{k+1}).$$

Thus, the PGCDNF does $\mathcal{O}(k^2d^2D^{k+1})$ arithmetic operations in $\mathbb{Z}_p$. □

Remark 4.8. *The dominant cost in PGCDNF comes from division test, and computing the contents and primitive parts of the input polynomials at each recursive level.*

Theorem 4.5.2. *The expected time complexity of the MGCDNF algorithm is*

$$\mathcal{O}(Nd(nH_M + CD^k) + (Nd^3 + Nd^2D^k) + Nkd^2D^{k+1} + N^3dD^k).$$

Proof. Using the Notation 4.1, we have $\#f_1, \#f_2, \#g \le (D+1)^k$. In the MGCDNF algorithm, the most dominant operations are as follows:

1. **Modular homomorphism:** The MGCDNF algorithm reduces the input polynomials $\check{f}_1$ and $\check{f}_2$ and the minimal polynomials $\check{M}_1, \ldots, \check{M}_n$ modulo a prime. For N primes, this costs

$$\underbrace{\mathcal{O}(NdnH_M)}_{\text{Reducing } \check{M}_i s} + \underbrace{\mathcal{O}(NdC(D+1)^k)}_{\text{Reducing } \check{f}_1, \check{f}_2} = \mathcal{O}(Nd(nH_M + CD^k)). \tag{4.8}$$

2. ϕ_γ **isomorphism:** The time complexity of building the matrix A and calculating A^{-1} for each prime is $\mathcal{O}(d^3)$, and the running time complexity of applying ϕ_γ to the non-zero terms of $\check{f}_1$

and $\check{f}_2$ for each prime is $\mathcal{O}(d^2(D+1)^k)$. Additionally, let G_p be the output of the PGCDNF algorithm in Line 10. The time complexity of calling ϕ_γ^{-1} for G_p in Line 15 for each prime is $\mathcal{O}(d^2(D+1)^k)$. Therefore, the complexity of using ϕ_γ and ϕ_γ^{-1} for N primes is

$$\mathcal{O}(Nd^3) + \mathcal{O}(Nd^2(D+1)^k) + \mathcal{O}(Nd^2(D+1)^k) = \mathcal{O}(Nd^3 + Nd^2D^k). \tag{4.9}$$

3. **PGCDNF:** In Theorem 4.5.1, we computed the time complexity of the PGCDNF. For N primes, the cost of the PGCDNF is $\mathcal{O}(k^2d^2D^{k+1})$.

4. **CRT and RNR:** Reconstructing at most $d(D+1)^k$ rational coefficients in lines 22 and 26 for N primes costs

$$\mathcal{O}(d(D+1)^k N^3). \tag{4.10}$$

5. **Trial Division:** In Line 29, MGCDNF chooses a new prime q. Then it reduces three polynomials f_1, f_2, and the stabilized result of RNR H modulo q and calls the DT Algorithm (Algorithm 6) to verify whether $H \mid f_1$ and $H \mid f_2$ over . The cost of DT is provided in Lemma 4.4.1.

$$\underbrace{\mathcal{O}(dCD^k)}_{\phi_q} + \underbrace{\mathcal{O}(k(d^2D^2 + dD^k))}_{\text{Lemma } 4.4.1} \tag{4.11}$$

The theorem follows by adding the costs explained above:

$$\underbrace{\mathcal{O}(Nd(nH_M + CD^k))}_{(4.8)} + \underbrace{\mathcal{O}(Nd^3 + Nd^2D^k)}_{(4.9)} + \underbrace{\mathcal{O}(Nk^2d^2D^{k+1})}_{\text{Theorem}(4.5.1)} + \underbrace{\mathcal{O}(N^3dD^k)}_{(4.10)} + \underbrace{\mathcal{O}(dCD^k + k(d^2D^2 + dD^k))}_{(4.11)}$$
$$= \mathcal{O}(Nd(nH_M + CD^k) + (Nd^3 + Nd^2D^k) + kNd^2D^{k+1} + N^3dD^k).$$

□

Remark 4.9. *To improve the expected time complexity of MGCDNF, we can replace Brown's dense interpolation with a sparse interpolation method. Assume that* $\mathrm{lc}(g, x_1) = \gcd(\mathrm{lc}(f_1, x_1), \mathrm{lc}(f_2, x_1))$. *Under the dense interpolation approach, reconstructing* g *requires* $\mathcal{O}(D^{k-1})$ *calls to the MEA in* $\bar{L}_p[x_1]$. *If* g *is sparse with* $\#g$ *terms, sparse interpolation reduces the number of calls to the MEA. In particular, Zippel's algorithm [41] requires* $\mathcal{O}(kD\#g)$ *MEA calls, while the algorithm of Hu and Monagan [20] further reduces this to* $\mathcal{O}(\#g)$ *MEA calls. The latter is based on the work of Ben-Or and Tiwari [5].*

Remark 4.10. *Computing* $M(z)$, A, *and* A^{-1} *using LAminpoly costs* $\mathcal{O}(d^3)$ *operations. This cost is negligible compared with the cost of PGCDNF, which is* $\mathcal{O}(kd^2D^{k+1})$, *provided that*

$$d^3 \ll kd^2D^{k+1}, \text{ that is}$$
$$d \ll kD^{k+1}.$$

4.6 Benchmark

We implemented MGCDNF and its subroutines in Maple [28]. To represent elements of the algebraic number field $L = \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and polynomials in $L[x_1, \ldots, x_k]$, we employed the recursive dense

data structure (recden) from [34]. In Maple, arithmetic over L can be performed using the package $Algebraic : -RecursiveDensePolynomials$, which relies on efficient C routines. All modular computations in MGCDNF use primes from the set $\mathbb{P}_{31} = \{\text{all 31-bit primes}\}$. In particular, the smallest prime used in our implementation is $p = 2^{30} + 3$.

In the benchmarks below, the inputs are $f_1, f_2 \in L[x_1, x_2]$ with $D_f = \deg(f_1, x_1) = \deg(f_1, x_2) = \deg(f_2, x_1) = \deg(f_2, x_2)$. We denote the monic gcd by $g = \gcd(f_1, f_2)$ and set $D_g = \deg(g, x_1) = \deg(g, x_2)$. In Tables 4.1 and 4.2, we have the following columns:

- d is the degree of the algebraic number field L.
- n is the number of extensions of L.
- N is the number of primes needed by the algorithm.
- NMEA is the total number of calls to the MEA.

The timing columns report:

- **MGCDNF 1**: total time for our improved MGCDNF algorithm, which uses the primitive element isomorphism ϕ_γ and computes GCDs over $\bar{L}_p$,
- **MGCDNF 2**: total time for the original MGCDNF algorithm, without using the primitive element isomorphism ϕ_γ, which computes the GCDs over L_p,
- **LAMP**: total time spent in Algorithm LAminpoly,
- **PGCDNF**: total time spent in Algorithm PGCDNF for both algorithms.

The first benchmark (Table 4.1) fixes the polynomial degrees $D_f = 32$ and $D_g = 16$ and varies the field degree from $d = 4$ up to $d = 1024$.

Table 4.1: **Timings in CPU seconds for computing** $\gcd(f_1, f_2)$ **over an algebraic number field of degree** d **with** n **extensions (max number of extensions).**

d	n	N	NMEA	D_f	D_g	MGCDNF 1			MGCDNF 2	
						time	LAMP	PGCDNF	time	PGCDNF
4	2	4	128	32	16	2.594	0.000	2.422	11.391	11.297
8	3	4	136	32	16	3.578	0.000	3.359	40.531	40.423
16	4	4	136	32	16	3.813	0.111	3.437	124.375	124.281
24	3	4	136	32	16	4.843	0.251	4.329	209.313	209.219
32	5	4	136	32	16	5.360	0.734	4.281	271.984	271.860
64	6	5	165	32	16	14.687	4.734	9.360	1040.391	1040.234
128	7	5	170	32	16	37.141	23.907	12.421	>1000	(-)
256	8	5	170	32	16	136.406	115.250	19.874	>1000	(-)
512	9	5	170	32	16	621.968	571.922	47.093	>1000	(-)
1024	10	(-)	(-)	32	16	>1000	(-)	(-)	>1000	(-)

The speedup achieved by employing ϕ_γ can be observed by comparing the PGCDNF columns for MGCDNF 1 and MGCDNF 2. In particular, for $d = 4, 8, 16, 24, 32$, the PGCDNF in MGCDNF 1 is approximately 4.66×, 12.04×, 36.16×, 48.33×, and 63.50× faster than PGCDNF in MGCDNF 2, respectively. As d increases further, MGCDNF 2 exceeds the 1000-second time limit already at $d = 64$ (PGCDNF $\approx$ 1040 seconds), while MGCDNF 1 still completes for larger field degrees (e.g., up to $d = 512$ in this run).

Remark 4.11. *Notice that when $d \geq 128$ and $n \geq 7$, the cost of the LAminpoly algorithm is greater than the cost of the PGCDNF.*

Benchmark 2 (Table 4.2) fixes the number field

$$L = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13})$$

with degree $d = 64$ and $n = 6$ extensions, and varies D_f and D_g. The PGCDNF timings again show an improvement: for example, at $D_f = 16$, PGCDNF takes 1.500 seconds in MGCDNF 1 compared with 236.688 seconds in MGCDNF 2 (about 158× faster). Moreover, in this experiment MGCDNF 2 remains below the 1000-second time limit up to $D_f = 32$ (PGCDNF = 978.047 seconds), whereas MGCDNF 1 still completes comfortably for all reported degrees.

Table 4.2: **Timings in CPU seconds for computing** $\gcd(f_1, f_2)$ **over an algebraic number field of degree** $d = 64$ **with** 6 **extensions, using recden.**

d	n	N	$NMEA$	D_f	D_g	MGCDNF 1			MGCDNF 2	
						time	LAMP	PGCDNF	time	PGCDNF
64	6	4	24	4	2	4.297	3.843	0.281	6.094	6.032
64	6	4	40	8	4	4.750	3.922	0.563	24.187	24.094
64	6	4	56	12	6	5.500	4.062	1.064	78.485	78.406
64	6	4	72	16	8	5.750	3.937	1.500	236.891	236.688
64	6	4	88	20	10	7.485	4.031	3.140	512.859	512.734
64	6	4	104	24	12	6.672	3.922	2.438	258.657	258.531
64	6	4	120	28	14	8.688	4.109	4.157	840.078	839.969
64	6	4	136	32	16	9.812	4.235	5.109	978.188	978.047
64	6	5	190	36	18	15.187	4.985	9.703	>1000	(-)

All timings were obtained on an Intel Core i7-6700. For the details of the benchmark, see https://github.com/MahsAnsari/Mahsa_Ansari_Thesis.

Chapter 5

Resultants in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k, y]$

5.1 Summary of contributions

Let $L \cong \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ be as defined in Section 3.3. The main contribution of this chapter is a new Monte Carlo modular algorithm, MRESNF (Algorithm 11), for computing the resultant of two multivariate polynomials f_1 and f_2 in $L[x_1, \ldots, x_k][y]$ with high probability. Furthermore, by employing the MEA, we present Algorithm URES (Algorithm 9) to compute the resultant of univariate polynomials. This chapter also includes an analysis of the expected time complexity of the MRESNF algorithm, along with two benchmarks. We have implemented MRESNF in Maple, employing a recursive dense data structure for representing polynomials in $L[x_1, \ldots, x_k][y]$ (see Chapter 7 for more details). The implementation details and benchmarks are available at https://github.com/MahsAnsari/Mahsa_Ansari_Thesis. A failure probability analysis of the MRESNF algorithm is presented in Chapter 6.

The results presented in this chapter have been published in the Proceedings of CASC '24 [3], and the MRESNF algorithm and its subalgorithms were also presented at the Maple Conference '24.

5.2 Introduction

Computing the resultant of two polynomials plays a significant role in various areas of mathematics. Resultants appear as a subproblem in solving systems of multivariate polynomials, elimination theory [11], and the factorization of polynomials over algebraic fields [32].

In 1971, Collins [10] introduced a modular algorithm to compute the resultant of two multivariate polynomials over $\mathbb{Z}$. Collin's algorithm is implemented in Maple by Wittkopf. Given $f_1, f_2 \in L[x_1, \ldots, x_k][y]$, we build upon the approaches introduced by Collins [10] and Brown [7] to compute $r = \operatorname{res}(f_1, f_2, y) \in L[x_1, \ldots, x_k]$.

Similar to the MGCDNF algorithm, to improve efficiency, MRESNF (Algorithm 11) transforms $f_1, f_2 \in L[x_1, \ldots, x_k]$ into their corresponding polynomials over $\bar{L}_p$, where p is a prime number. Next, Algorithm MRESNF employs Algorithm 10 (PRESNF) to compute the resultant modulo the prime p by reducing the problem to that of computing the resultant of two univariate polynomials, where the MEA is applied. To perform this reduction, Algorithm PRESNF employs evaluation and dense interpolation. Finally, MRESNF applies the CRT and RNR to recover the rational coefficients of the target resultant. In this chapter, we fix the lexicographic order over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ with $x_1 > x_2 > \ldots > x_n$.

5.2.1 Resultants

In this section, we present the relevant background, definitions, and notation. Throughout this thesis, unless explicitly specified otherwise, R denotes a commutative ring with identity $1_R \neq 0$.

Definition 5.2.1. *Let $f_1 = \sum_{j=0}^{n_1} a_j y^j$ and $f_2 = \sum_{j=0}^{n_2} b_j y^j$ be two non-zero polynomials in $R[x_1, \ldots, x_k, y]$, where $n_1 = \deg(f_1, y)$, $n_2 = \deg(f_2, y)$, and $a_j, b_j \in R[x_1, \ldots, x_k]$. The Sylvester matrix of f_1 and f_2 w.r.t. the variable y is the $(n_1 + n_2) \times (n_1 + n_2)$ matrix*

$$\operatorname{sylv}(f_1, f_2, y) = \begin{bmatrix} a_m & \cdots & a_0 & & \\ & a_m & \cdots & a_0 & \\ & & \ddots & & \\ & & a_m & \cdots & a_0 \\ b_n & \cdots & b_0 & & \\ & & \ddots & & \\ & & b_n & \cdots & b_0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} n_2 \text{ rows of coefficients of } f_1 \\ \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} n_1 \text{ rows of coefficients of } f_2 \end{matrix},$$

where the upper part of the matrix consists of n_2 rows of coefficients of f_1, the lower part consists of n_1 rows of coefficients of f_2, and all entries not shown are zero.

Example 5.2.1. *Let $f_1 = y^3 + (2z)y + 1$ and $f_2 = 2y^2 + (z)y$ be two polynomials in $\mathbb{Z}_3[z]/\langle z^2 - 2\rangle[y]$. The Sylvester matrix of f_1 and f_2 w.r.t. y is given by*

$$\operatorname{sylv}(f_1, f_2, x) = \begin{bmatrix} 1 & 0 & 2z & 1 & 0 \\ 0 & 1 & 0 & 2z & 1 \\ 2 & z & 0 & 0 & 0 \\ 0 & 2 & z & 0 & 0 \\ 0 & 0 & 2 & z & 0 \end{bmatrix}.$$

Definition 5.2.2. *Let $f_1 = \sum_{j=0}^{n_1} a_j y^j$ and $f_2 = \sum_{j=0}^{n_2} b_j y^j$ be two non-zero polynomials in $R[x_1, \ldots, x_k, y]$, where $n_1 = \deg(f_1, y)$, $n_2 = \deg(f_2, y)$, and $a_j, b_j \in R[x_1, \ldots, x_k]$. Let $\operatorname{res}(f_1, f_2, y)$ denote the resultant of f_1 and f_2 w.r.t. the variable y.*

(i) If $n_1 = 0$, then $\operatorname{res}(f_1, f_2, y) = a_0^{n_2}$.

(ii) If $n_2 = 0$, then $\operatorname{res}(f_1, f_2, y) = b_0^{n_1}$.

(iii) If $n_1 > 0$ and $n_2 > 0$, then $\operatorname{res}(f_1, f_2, y) = \det(\operatorname{sylv}(f_1, f_2, y)) \in R[x_1, \ldots, x_k]$.

If f_1 and f_2 are univariate polynomials in $R[x]$, we write $\operatorname{res}(f_1, f_2)$ for $\operatorname{res}(f_1, f_2, x)$.

Remark 5.1. *In the above definition, if $n_1 = n_2 = 0$, then $\operatorname{res}(f_1, f_2, y) = 1$.*

Example 5.2.2. *Let f_1 and f_2 be the polynomials defined in Example 5.2.1. Then*

$$\operatorname{res}(f_1, f_2) = \det(\operatorname{sylv}(f_1, f_2)) = z + 1 \in \mathbb{Z}_3[z]/\langle z^2 - 2\rangle.$$

In the following theorem, we summarize the fundamental properties of the resultant that will be used throughout this chapter.

Theorem 5.2.1. *[18, Theorem 9.2] Let $f_1, f_2 \in R[x]$ with $\deg(f_1) = n_1 \geq 0$ and $\deg(f_2) = n_2 \geq 0$. Let $c \in R$ be a non-zero constant. Then*

(i) $\operatorname{res}(c, f_2) = c^{n_2}$.

(ii) $\operatorname{res}(f_1, f_1) = 0$.

(iii) $\operatorname{res}(f_1, f_2) = (-1)^{n_1 n_2}\operatorname{res}(f_2, f_1)$.

(iv) $\operatorname{res}(cf_1, f_2) = c^{n_2}\operatorname{res}(f_1, f_2)$.

In Lemma 2.2.9, we examined the relationship between $g_\phi = \gcd(\phi(f_1), \phi(f_2))$ and $\phi(g) = \phi(\gcd(f_1, f_2))$, where ϕ is a ring homomorphism. In the following theorem, we establish a similar correspondence for the $\operatorname{res}(f_1, f_2)$ under a ring homomorphism.

Theorem 5.2.2. *[18, Theorem 9.2, part (vi)] Let $f_1 = \sum_{j=0}^{n_1} a_j x_i^j$ and $f_2 = \sum_{j=0}^{n_2} b_j x_i^j$ be two non-zero polynomials, where $a_j, b_j \in R[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k]$. Assume that $\deg(f_1, x_i) = n_1 > 0$ and $\deg(f_2, x_i) = n_2 > 0$. Let $\phi : R \longrightarrow \tilde{R}$ be a ring homomorphism, and define $d_{1,i} = \deg(\phi(f_1), x_i)$, $d_{2,i} = \deg(\phi(f_2), x_i)$.*

(i) If $d_{1,i} < n_1$ and $d_{2,i} = n_2$, then

$$\phi(\operatorname{res}(f_1, f_2, x_i)) = (-1)^{n_2(n_1 - d_{1,i})}\phi(b_r)^{n_1 - d_{1,i}}\operatorname{res}(\phi(f_1), \phi(f_2), x_i).$$

(ii) If $d_{1,i} = n_1$ and $0 \leq d_{2,i} \leq n_2$, then

$$\phi(\operatorname{res}(f_1, f_2, x_i)) = \phi(a_t)^{n_2 - d_{2,i}}\operatorname{res}(\phi(f_1), \phi(f_2), x_i).$$

(iii) If $d_{1,i} < n_1$ and $d_{2,i} < n_2$, then $\phi(\operatorname{res}(f_1, f_2, x_i)) = 0$.

Proof. (i) From Definition 5.2.2, $\operatorname{res}(f_1, f_2, x_i) = \det(\operatorname{sylv}(f_1, f_2, x_i))$, where the Sylvester matrix is

$$\operatorname{sylv}(f_1, f_2, x_i) = \begin{bmatrix} a_t & \cdots & a_0 & 0 & \cdots & 0 \\ 0 & a_t & \cdots & a_0 & \cdots & 0 \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & a_t & \cdots & a_0 \\ b_r & \cdots & b_0 & 0 & \cdots & 0 \\ 0 & b_r & \cdots & b_0 & \cdots & 0 \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & b_r & \cdots & b_0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \} \, n_2 \text{ rows of coefficients of } f_1(x_i) \\ \\ \\ \\ \\ \} \, n_1 \text{ rows of coefficients of } f_2(x_i) \\ \\ \\ \end{matrix},$$

with n_2 rows of coefficients of f_1 and n_1 rows of coefficients of f_2. Given that $\deg(\phi(f_1), x_i) = d_{1,i} < n_1$ and $\deg(\phi(f_2), x_i) = d_{2,i} = n_2$, we can write

$$\phi(f_1) = \sum_{j=0}^{d_{1,i}} \phi(a_j)x_i^j, \quad \text{and} \quad \phi(f_2) = \sum_{j=0}^{n_2} \phi(b_j)x_i^j.$$

Thus,

$$\operatorname{sylv}(\phi(f_1),\phi(f_2),x_i) = \begin{bmatrix} \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) & 0 & \cdots & 0 \\ 0 & \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) & \cdots & 0 \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) \\ \phi(b_r) & \cdots & \phi(b_0) & 0 & \cdots & 0 \\ 0 & \phi(b_r) & \cdots & \phi(b_0) & \cdots & 0 \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & \cdots & 0 & \phi(b_r) & \cdots & \phi(b_0) \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} n_2 \text{ rows of } \phi(f_1) \\ \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} d_{1,i} \text{ rows of } \phi(f_2) \end{matrix}.$$

Let $S = \phi(\operatorname{sylv}(f_1, f_2, x_i))$. Since $\phi(a_j) = 0$ for $j > d_{1,i}$, the first $n_1 - d_{1,i}$ columns of the f_1 entries are entirely zero in the top n_2 rows of S. The matrix then takes the form:

$$S = \begin{bmatrix} 0 & \cdots & 0 & \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) & 0 & \cdots \\ 0 & \cdots & 0 & 0 & \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) & \cdots \\ \vdots & & \ddots & \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \phi(a_{d_{1,i}}) & \cdots & \phi(a_0) \\ \phi(b_r) & \phi(b_{n_2-1}) & \cdots & \phi(b_0) & 0 & \cdots & 0 & \cdots \\ 0 & \phi(b_r) & \phi(b_{n_2-1}) & \cdots & \phi(b_0) & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & & & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & \phi(b_r) & \phi(b_{n_2-1}) & \cdots & \phi(b_1) & \phi(b_0) \end{bmatrix}.$$

To compute $\det(S)$, expand the determinant down the first column. The only non-zero entry in the first column is $\phi(b_r)$, occurring in row $n_2 + 1$. Deleting that row and column produces a minor whose first column again has a unique nonzero entry $\phi(b_r)$ in row $n_2 + 1$. Repeating this process $n_1 - d_{1,i}$ times yields the factor $(-1)^{n_2(n_1-d_{1,i})}\phi(b_r)^{n_1-d_{1,i}}$, and the remaining minor is $\operatorname{sylv}(\phi(f_1), \phi(f_2), x_i)$. Therefore,

$$\det(S) = \phi(\operatorname{res}(f_1, f_2, x_i)) = (-1)^{n_2(n_1-d_{1,i})}\phi(b_r)^{n_1-d_{1,i}}\operatorname{res}(\phi(f_1), \phi(f_2), x_i).$$

(ii) By applying the same reasoning as in part (i), the desired result follows immediately.

(iii) Since ϕ is a ring homomorphism, and by the definition of the resultant,

$$\det(\phi(\operatorname{sylv}(f_1, f_2, x_i))) = \phi(\det(\operatorname{sylv}(f_1, f_2, x_i))) = \phi(\operatorname{res}(f_1, f_2, x_i)). \tag{5.1}$$

If both $d_{1,i} < n_1$ and $d_{2,i} < n_2$, then the first column of the matrix $\phi(\operatorname{sylv}(f_1, f_2, x_i))$ is zero. This implies that $\det(\phi(\operatorname{sylv}(f_1, f_2, x_i))) = 0$. Therefore, from Equation (5.1), $\phi(\operatorname{res}(f_1, f_2, x_i)) = 0$.

□

Remark 5.2. *From part (ii) of Theorem 5.2.2, if $d_{1,i} = n_1$ and $d_{2,i} = n_2$, then*

$$\phi(\operatorname{res}(f_1, f_2, x_i)) = \operatorname{res}(\phi(f_1), \phi(f_2), x_i).$$

Example 5.2.3. *Let*

$$f_1 = 5yx^2 + y^2x + 3, \text{ and}$$
$$f_2 = 7x^2y^2 + 2y + 1$$

be two polynomials in $\mathbb{Z}[x, y]$. *Let* $\phi_5 : \mathbb{Z} \to \mathbb{Z}_5$ *be the modular homomorphism reducing the coefficients of the polynomials modulo* 5. *Using Maple, we verify part (i) of Theorem 5.2.2 by computing*

$$\phi_5(\det(\text{sylv}(f_1, f_2, x))) = \phi_5(\text{res}(f_1, f_2, x)) \text{ and}$$

$$\det(\text{sylv}(\phi_5(f_1), \phi_5(f_2), x)) = \text{res}(\phi_5(f_1), \phi_5(f_2), x).$$

These Sylvester matrices and determinants are computed with Maple's `LinearAlgebra` *package.*

```
> with(LinearAlgebra):

> f1 := 5*y*x^2 + y^2*x + 3:
> f2 := 7*x^2*y^2 + 2*y + 1:
> S1 := SylvesterMatrix(f1,f2,x);  # Sylvester matrix of f1 and f2 w.r.t. x
                                   S1 := Matrix(4,4,[
                                      [5*y,    y^2,      3,       0],
                                      [0,      5*y,      y^2,     3],
                                      [7*y^2,  0,        2*y+1,   0],
                                      [0,      7*y^2,    0,       2*y+1]])
> da := Determinant(S1) mod 5;     # phi_5(res(f1,f2,x))
                                   da := 4*y^7 + 2*y^6 + y^4
# Reduce the polynomials modulo 5
> f1p := f1 mod 5;                 # phi_5(f1)
> f2p := f2 mod 5;                 # phi_5(f2)
                                   f1p := y^2*x + 3
                                   f2p := 2*x^2*y^2 + 2*y + 1
# Sylvester matrix of phi_5(f1) and phi_5(f2) w.r.t. x
> S2 := SylvesterMatrix(f1p,f2p,x);
                                   S2 := Matrix(3,3,[
                                      [y^2,    3,       0],
                                      [0,      y^2,     3],
                                      [2*y^2,  0,       2*y+1]])
> dp := Determinant(S2) mod 5;     # res(phi_5(f1),phi_5(f2),x)
                                   dp := 2*y^5 + y^4 + 3*y^2
> n1,n2 := degree(f1,x), degree(f2,x):          # deg(f1,x)=2, deg(f2,x)=2
> b,d := lcoeff(f2p,x), degree(f1p,x):          # b = 2*y^2, d = 1
# Verify part (i) of Theorem thm:phires
> E := Expand((-1)^((n1-d)*n2) * (b^(n1-d)) * dp) mod 5; #E=da
                                   E := 4*y^7 + 2*y^6 + y^4
>E-da;
                                    0
```

A connection between the GCD and the resultant of two univariate polynomials is established by Sylvester's Criterion [18], which characterizes when two polynomials share a non-trivial common factor.

Theorem 5.2.3. *[18, Sylvester's Criterion, Chapter 7] Let $f_1, f_2 \in R[x]$, and suppose that $g = \gcd(f_1, f_2)$ exists. Then $\deg(g, x) > 0$ if and only if $\operatorname{res}(f_1, f_2) = 0$.*

5.3 Computing resultants of univariate polynomials

Let $f_1, f_2 \in R[x]$. In this section, we present a new formula for computing $\operatorname{res}(f_1, f_2) \in R$ using the m.p.r.s. (see Definition 2.2.6). We then introduce Algorithm URES (Algorithm 9) for computing resultants of univariate polynomials over R. In Section 5.4.1, Algorithm PRESNF (Algorithm 10) applies URES with $R = \bar{L}_p$.

Theorem 5.3.1. *[11, Section 3.5, Exercise 16 part b] Let $f_1, f_2 \in R[x]$ s.t. $\operatorname{lc}(f_2)$ is a unit. Let $r = \operatorname{rem}(f_1, f_2) \in R[x]$, $n_1 = \deg(f_1)$, and $n_r = \deg(r)$. Then*

$$\operatorname{res}(f_2, f_1) = \operatorname{lc}(f_2)^{n_1 - n_r} \operatorname{res}(f_2, r).$$

Now, we are well-equipped to present a new formula for computing the resultant using m.p.r.s..

Theorem 5.3.2. *Let $f_1, f_2 \in R[x]$. Assume that the MEA does not fail for f_1 and f_2. Let $r_1, r_2, \ldots, r_l$ be the m.p.r.s. generated by f_1 and f_2, with $r_{l+1} = 0$. Let $n_i = \deg(r_i)$ for $1 \le i \le l$.*

(i) If $\deg(r_l) > 0$, then $\operatorname{res}(f_1, f_2) = 0$.

(ii) If $\deg(r_l) = 0$, then

$$\operatorname{res}(f_1, f_2) = (-1)^v \Big(\prod_{i=2}^{l-1} \operatorname{lc}(r_i)^{n_{i-1}}\Big) \operatorname{lc}(r_l)^{n_{l-1}},$$

where $v = \sum_{i=1}^{l-2} n_i n_{i+1}$.

Proof. (i) If $\deg(r_l) > 0$, then the monic $\gcd(f_1, f_2) \neq 1$. Applying Theorem 5.2.3, we have $\operatorname{res}(f_1, f_2) = 0$.

(ii) Now assume $\deg(r_l) = 0$. In the first step of the MEA, we have $M_1 = M_2 q_3 + r_3$ where $M_1 = f_1$, $M_2 = \operatorname{monic}(f_2)$, and $\deg(r_3) < \deg(M_2)$. Since the MEA does not fail, $\operatorname{lc}(f_2)$ is a unit. From Theorem 5.3.1, we have

$$\operatorname{res}(M_2, M_1) = \operatorname{lc}(M_2)^{n_1 - n_3} \operatorname{res}(M_2, r_3). \tag{5.2}$$

Therefore,

$$\begin{aligned}
\operatorname{res}(M_2, r_3) &= \underbrace{\operatorname{lc}(M_2)^{n_3 - n_1} \operatorname{res}(M_2, M_1)}_{\text{Equation (5.2)}} \\
&= \underbrace{\operatorname{res}(M_2, M_1)}_{\operatorname{lc}(M_2)=1} \\
&= \underbrace{\operatorname{res}(\operatorname{lc}(f_2)^{-1} f_2, f_1)}_{M_2 = \operatorname{monic}(f_2),\ M_1 = f_1} \\
&= \underbrace{(\operatorname{lc}(f_2)^{-1})^{n_1} \operatorname{res}(f_2, f_1)}_{\text{Theorem 5.2.1, part } (iv)} \\
&= \underbrace{(-1)^{n_1 n_2} \operatorname{lc}(f_2)^{-n_1} \operatorname{res}(f_1, f_2)}_{\text{Theorem 5.2.1, part } (iii)}.
\end{aligned} \tag{5.3}$$

From Equation (5.3), we obtain

$$\operatorname{res}(f_1, f_2) = (-1)^{n_1 n_2} \operatorname{lc}(f_2)^{n_1} \operatorname{res}(M_2, r_3).$$

Continuing this process, in the ith step of the MEA, where $M_{i-1} = M_i q_{i+1} + r_{i+1}$, we have

$$\begin{aligned} \operatorname{res}(M_i, r_{i+1}) &= \operatorname{res}(M_i, M_{i-1}) \\ &= \operatorname{lc}(r_i)^{-n_{i-1}} \operatorname{res}(r_i, M_{i-1}) \\ &= \operatorname{lc}(r_i)^{-n_{i-1}} (-1)^{n_i n_{i-1}} \operatorname{res}(M_{i-1}, r_i). \end{aligned}$$

Expanding $\operatorname{res}(M_{i-1}, r_i)$, we obtain

$$\operatorname{res}(M_i, r_{i+1}) = (-1)^v \Big(\prod_{j=2}^{i} \operatorname{lc}(r_j)^{-n_{j-1}}\Big) \operatorname{res}(f_1, f_2)$$

where $v = \sum_{j=2}^{i} n_{j-1} n_j$. Moreover, in the last step of the MEA, since $r_l \in R$ is a scalar, we have $\operatorname{res}(M_{l-1}, r_l) = r_l^{n_{l-1}} = \operatorname{lc}(r_l)^{n_{l-1}}$, which implies the result.

□

By applying Theorem 5.3.2, we modify the MEA to compute the resultant of two univariate polynomials $f_1, f_2 \in R[x]$, as described in Algorithm 9 (URES). This algorithm is employed at the base case in Algorithm 10 (PRESNF), Line 2.

Algorithm 9: URES

Input: non-zero $f_1, f_2 \in R[x]$ s.t. $0 \leq \deg(f_2) \leq \deg(f_1)$ where R is a commutative ring with identity $1 \neq 0$.

Output: Either $Res = \operatorname{res}(f_1, f_2)$ or a message FAIL.

1 $r_1 = f_1$, $r_2 = f_2$, $i = 2$
2 $M_1 = r_1$, $Res = 1$, $v = 0$
3 $n_1 = \deg(f_1)$,$n_2 = \deg(f_2)$
4 **while** $r_i \neq 0$ **do**
5 $M_i = \operatorname{monic}(r_i)$// $\operatorname{monic}(r_i) = \operatorname{lc}(r_i)^{-1} \cdot r_i$
6 **if** $M_i =$ failed **return** *(FAIL)* // The algorithm encounters a zero divisor.
7 Set r_{i+1} to be the remainder of M_{i-1} divided by M_i
8 **if** $r_{i+1} = 0$ and $n_i \neq 0$ **then return**(0) // If the monic $\gcd(f_1, f_2) \neq 1$, then $\operatorname{res}(f_1, f_2) = 0$.
9 Set $n_{i+1} = \deg(r_{i+1})$
10 Set $Res = Res \cdot \operatorname{lc}(r_i)^{n_{i-1}}$
11 Set $v = v + n_i n_{i-1}$
12 Set $i = i + 1$
13 $Res = (-1)^v Res$
14 **return**(Res)

Example 5.3.1. *Let $f_1, f_2 \in \bar{L}_3[x]$, where $\bar{L}_3 = \mathbb{Z}_3[z]/\langle z^2 - 2\rangle$. Given the inputs $f_1 = x^3 + (2z)x + 1$ and $f_2 = 2x^2 + xz$, Algorithm 9 computes $\operatorname{res}(f_1, f_2) = z + 1$. The intermediate values generated during the execution of Algorithm URES are summarized in Table 5.1.*

Table 5.1: Example 5.3.1 over $\mathbb{Z}_3[z]/\langle z^2-2\rangle$

Dividend	Divisor	Remainder	R
M_1	$M_2 = \text{monic}(f_2) = x^2 + 2xz$	$r_3 = (2z+2)x + 1$	$R = 2$
M_2	$M_3 = \text{monic}(r_3) = x + 2z + 1$	$r_4 = 2z + 1$	$R = z$
M_3	$M_4 = \text{monic}(r_4) = 1$	$r_5 = 0$	$R = z + 1$

Lemma 5.3.3. *Algorithm URES fails if and only if the MEA fails.*

Proof. The result follows directly from the construction of Algorithm URES. The URES algorithm fails if and only if, in Line 5, $\text{lc}(r_i)$ is not invertible in R, causing $\text{monic}(r_i)$ to fail. The MEA fails for exactly the same reason. Hence, URES fails if and only if the MEA fails. □

In the following theorem, we prove the correctness of the Algorithm URES.

Theorem 5.3.4. *Let $f_1, f_2 \in R[x]$. If the URES algorithm does not fail, then it outputs* $\text{res}(f_1, f_2)$.

Proof. If Algorithm URES does not fail for f_1 and f_2, then by Lemma 5.3.3, the MEA also does not fail for f_1 and f_2. Therefore, $r_1, r_2, \ldots, r_l$ generated in URES are the m.p.r.s. generated by f_1 and f_2, with $r_{l+1} = 0$. Let $n_i = \deg(r_i)$ for $1 \le i \le l$. If $n_l > 0$, then the monic $\gcd(f_1, f_2) \neq 1$, so URES returns 0 in Line 8. Otherwise, in Line 13, URES returns

$$Res = (-1)^v \Big(\prod_{i=2}^{l-1} \text{lc}(r_i)^{n_{i-1}}\Big)\text{lc}(r_l)^{n_{l-1}},$$

where $v = \sum_{i=1}^{l-2} n_i n_{i+1}$. From Part (ii) of Theorem 5.3.2, $Res = \text{res}(f_1, f_2)$. Therefore, if the URES algorithm does not fail, it returns $\text{res}(f_1, f_2)$. □

The cost of the algorithm URES is the same as that of the MEA.

Lemma 5.3.5. *URES performs $\mathcal{O}(\deg(f_1) \cdot \deg(f_2))$ operations in the ring R.*

Corollary 5.1. *Assume that multiplication and inverses in $\bar{L}_p$ each cost $\mathcal{O}(d^2)$ operations in $\mathbb{Z}_p$. Therefore, when $R = \bar{L}_p$, the cost of URES is*

$$\mathcal{O}(d^2 \deg(f_1) \deg(f_2))$$

operations in $\mathbb{Z}_p$.

5.4 A new modular algorithm for computing resultants over $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$

Let $f_1, f_2 \in L[x_1, \ldots, x_k, y]$. In this section, we present our modular algorithm MRESNF for computing the resultant $r = \text{res}(f_1, f_2, y) \in L[x_1, \ldots, x_k]$ and its subalgorithm PRESNF, which computes the resultant of two polynomials in $\bar{L}_p[x_1, \ldots, x_k, y]$. Write

$$f_1 = \sum_{i=0}^{n_1} a_i y^i, \text{ and } f_2 = \sum_{j=0}^{n_2} b_j y^j$$

as two polynomials in $L[x_1,\ldots,x_k][y]$, with $n_1=\deg(f_1,y)$, $n_2=\deg(f_2,y)$, and $a_i,b_j\in L[x_1,\ldots,x_k]$. First, MRESNF computes the resultant modulo a sequence of primes. For each prime, it calls the PRESNF algorithm, which is a recursive algorithm to compute

$$r_p=\operatorname{res}(\phi_\gamma(\phi_p(\check{f}_1)),\phi_\gamma(\phi_p(\check{f}_2)),y)\in\bar{L}_p[x_1,\ldots,x_k],$$

using evaluation homomorphism and interpolation. Subsequently, MRESNF employs CRT and RNR to reconstruct the rational coefficients of the resultant. However, similar to the MGCDNF algorithm, the successful reconstruction of the resultant is not guaranteed for all primes and evaluation points. In Section 5.4.2, we identify problematic evaluation points and primes, respectively. The homomorphism diagram of the MRESNF algorithm is presented in Figure 5.1.

$$\begin{array}{ccc}
f_1,f_2\in\mathbb{Q}(\alpha_1,\ldots,\alpha_n)[x_1,\ldots,x_k][y] & \longrightarrow & \operatorname{res}(f_1,f_2,y)\in\mathbb{Q}(\alpha_1,\ldots,\alpha_n)[x_1,\ldots,x_k] \\
\downarrow \textit{semi-associations} & & \uparrow \textit{RNR} \\
\check{f}_1,\check{f}_2\in L_{\mathbb{Z}}[x_1,\ldots,x_k][y] & & \operatorname{res}(\check{f}_1,\check{f}_2,y)\in\mathbb{Z}_P[\alpha_1,\ldots,\alpha_n][x_1,\ldots,x_k] \\
\phi_p \text{ for } p\in\{p_1,p_2,\ldots\}\downarrow p\nmid\prod_{i=1}^{n}\operatorname{lc}(M_i,z_i)\cdot\operatorname{lc}(\check{f}_1)\cdot\operatorname{lc}(\check{f}_2) & & \uparrow \textit{CRT and } P=\prod p_i \\
\check{f}_1,\check{f}_2\in L_p[x_1,\ldots,x_k][y] & & \operatorname{res}(\check{f}_1,\check{f}_2,y)\in L_p[x_1,\ldots,x_k] \\
\downarrow \phi_\gamma & & \uparrow \phi_\gamma^{-1} \\
\check{f}_1,\check{f}_2\in \overline{L}_p[x_1,\ldots,x_k][y] & \xrightarrow{\textit{PRESNF}} & \operatorname{res}(\check{f}_1,\check{f}_2,y)\in \overline{L}_p[x_1,\ldots,x_k]
\end{array}$$

Figure 5.1: Homomorphism diagram of the MRESNF algorithm.

5.4.1 PRESNF

Let p be a large prime and $f_1,f_2\in\bar{L}_p[x_1,\ldots,x_k][y]$. To compute

$$r=\operatorname{res}(f_1,f_2,y)\in\bar{L}_p[x_1,\ldots,x_k],$$

Algorithm PRESNF (Algorithm 10) employs evaluation and dense interpolation following the strategy of Brown's algorithm. The PRESNF algorithm is recursive. If $f_1,f_2\in\bar{L}_p[y]$, the algorithm computes $\operatorname{res}(f_1,f_2)\in\bar{L}_p$ using URES (Algorithm 9). Otherwise, PRESNF chooses $\beta_k\in[0,p)$ randomly, and in Line 10, reduces f_1 and f_2 to polynomials in $\bar{L}_p[x_1,\ldots,x_{k-1}][y]$ by using the evaluation homomorphism (see Definition 2.2.7, part (ii)),

$$\phi_{x_k=\beta_k}(f_i)=f_i(x_1,\ldots,x_{k-1},\beta_k,y)\text{ for }i=1,2.$$

If the leading coefficients of f_1 and f_2 do not vanish at $x_k=\beta_k$, we can apply Remark 5.2 for $\phi_{x_k=\beta_k}$. That is,

$$\operatorname{res}(\phi_{x_k=\beta_k}(f_1),\phi_{x_k=\beta_k}(f_2),y)=\phi_{x_k=\beta_k}(\operatorname{res}(f_1,f_2,y)).$$

Algorithm PRESNF recursively computes

$$R_\beta=\operatorname{res}(\phi_{x_k=\beta_k}(f_1),\phi_{x_k=\beta_k}(f_2),y)\in\bar{L}_p[x_1,x_2,\ldots,x_{k-1}].$$

Next, it interpolates the variable x_k in r. Let $n_1 = \deg(f_1, y)$ and $n_2 = \deg(f_2, y)$. From the structure of Sylvester's matrix, we have

$$\deg(r, x_k) \leq n_2 \deg(f_1, x_k) + n_1 \deg(f_2, x_k).$$

Let $h = n_2 \deg(f_1, x_k) + n_1 \deg(f_2, x_k)$. To successfully interpolate the variable x_k at r, we require at most $B = h + 1$ evaluation points. In Line 8 of PRESNF, we use the bound B. The algorithm PRESNF fails if URES (in Line 2) attempts to invert a zero divisor while computing the resultant of two polynomials in $\bar{L}_p[y]$.

As with the PGCDNF algorithm, not all evaluation points used in the PRESNF algorithm guarantee the correct reconstruction of the resultant. This issue is discussed in detail in Section 5.4.2.

Algorithm 10: PRESNF

Input: non-zero $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k][y]$.
Output: $r = \text{res}(f_1, f_2, y) \in \bar{L}_p[x_1, \ldots, x_k]$ or FAIL.

1 **if** $k = 0$ **then**
2 **return**(URES(f_1, f_2)) // $f_1, f_2 \in \bar{L}_p[y]$.
3 $prod = 1$
4 $n_1, n_2 = \deg(f_1, y), \deg(f_2, y)$
5 $h = n_2 \deg(f_1, x_k) + n_1 \deg(f_2, x_k)$// $\deg(r, x_k) \leq h$
6 $B = h + 1$ // Since $\deg(r, x_k) \leq h$, to interpolate x_k, we need at most $h + 1$ evaluation points.
7 $j = 0$
8 **for** $i = 1$ *to* B **do**
9 Pick a new evaluation point β at random from $\mathbb{Z}_p$ s.t. $\text{lc}(f_1, y)(x_k = \beta) \neq 0$ and $\text{lc}(f_2, y)(x_k = \beta) \neq 0$
10 $F_{1,\beta} = f_1(x_k = \beta)$ and $F_{2,\beta} = f_2(x_k = \beta)$ // $F_{1,\beta}, F_{2,\beta} \in \bar{L}_p[x_1, \ldots, x_{k-1}][y]$
11 $R_\beta = PRESNF(F_{1,\beta}, F_{2,\beta}, y)$
12 **if** $R_\beta = FAIL$ **then**
13 **return**(FAIL)
14 **if** $j = 0$ **then**
  // First iteration.
15 $r = R_\beta$
16 $prod = x_k - \beta$
17 $j = j + 1$
18 **else**
  // Interpolate x_k in the resultant R incrementally
19 $V_\beta = prod(x_k = \beta)^{-1} \cdot (R_\beta - r(x_k = \beta))$
20 $r = r + V_\beta \cdot prod$
21 $prod = prod \cdot (x_k - \beta)$
22 **return**(r)

5.4.2 MRESNF

Let $f_1, f_2 \in L[x_1, \ldots, x_k, y]$. Algorithm MRESNF first computes the resultant of the semi-associates $\text{res}(\check{f}_1, \check{f}_2, y)$, and then recovers $\text{res}(f_1, f_2, y)$ using a known scaling factor. The next theorem gives the scaling rule needed to relate these two resultants.

Theorem 5.4.1. *Let f_1 and f_2 be two non-zero polynomials in $R[x_1, \ldots, x_k, y]$ with $n_1 = \deg(f_1, y)$ and $n_2 = \deg(f_2, y)$. Let $a, b \in R$. Then*

$$\operatorname{res}(af_1, bf_2, y) = a^{n_2} b^{n_1} \operatorname{res}(f_1, f_2, y).$$

Proof. From Parts (iii) and (iv) of Theorem 5.2.1, we have

$$\begin{aligned}
\operatorname{res}(af_1, bf_2, y) &= a^{n_2} \operatorname{res}(f_1, bf_2, y) \qquad \text{(part (iv))} \\
&= (-1)^{n_1 n_2} a^{n_2} \operatorname{res}(bf_2, f_1, y) \qquad \text{(part (iii))} \\
&= (-1)^{n_1 n_2} a^{n_2} b^{n_1} \operatorname{res}(f_2, f_1, y) \qquad \text{(part (iv))} \\
&= (-1)^{n_1 n_2} (-1)^{n_1 n_2} a^{n_2} b^{n_1} \operatorname{res}(f_1, f_2, y) \qquad \text{(part (iii))} \\
&= a^{n_2} b^{n_1} \operatorname{res}(f_1, f_2, y).
\end{aligned}$$

□

Using the above theorem, we prove that $\operatorname{res}(f_1, f_2, y) = a \cdot \operatorname{res}(\check{f}_1, \check{f}_2, y)$ for $a \in \mathbb{Q}_{>0}$.

Corollary 5.2. *Let f_1 and f_2 be non-zero polynomials in $L[x_1, \ldots, x_k, y]$ with $n_1 = \deg(f_1, y)$ and $n_2 = \deg(f_2, y)$. Consider the semi-associates*

$$\begin{aligned}
\check{f}_1 &= s_1 \cdot f_1 \\
\check{f}_2 &= s_2 \cdot f_2,
\end{aligned}$$

where $s_1, s_2 \in \mathbb{Q}_{>0}$ are the smallest positive rational numbers s.t. $\operatorname{den}(\check{f}_1) = \operatorname{den}(\check{f}_2) = 1$ (Definition 3.3.2). Then

$$\operatorname{res}(f_1, f_2, y) = \frac{\operatorname{res}(\check{f}_1, \check{f}_2, y)}{s_1^{n_2} \cdot s_2^{n_1}}.$$

Proof. Applying Theorem 5.4.1 w.r.t. y yields

$$\begin{aligned}
\operatorname{res}(\check{f}_1, \check{f}_2, y) &= \operatorname{res}(s_1 \cdot f_1, s_2 \cdot f_2, y) \\
&= s_1^{n_2} \cdot s_2^{n_1} \cdot \operatorname{res}(f_1, f_2, y),
\end{aligned}$$

and therefore

$$\operatorname{res}(f_1, f_2, y) = \frac{\operatorname{res}(\check{f}_1, \check{f}_2, y)}{s_1^{n_2} \cdot s_2^{n_1}}. \tag{5.4}$$

□

The Algorithm MRESNF begins by replacing the input polynomials f_1, f_2 and the minimal polynomials $M_1, \ldots, M_n$ with their semi-associates. It then chooses a prime p s.t. $p \nmid \operatorname{lc}(\check{f}_1) \cdot \operatorname{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \operatorname{lc}(\check{M}_i)$. Next, MRESNF applies the modular homomorphism ϕ_p to reduce coefficients modulo p, and map $\check{f}_1$ and $\check{f}_2$ to their corresponding polynomials in $L_p = \mathbb{Z}[z_1, \ldots, z_n]/\langle m_1, \ldots, m_n \rangle$, where $m_i = \phi_p(\check{M}_i)$, for $1 \leq i \leq n$. MRESNF then applies the primitive homomorphism ϕ_γ to convert the polynomials over L_p to their corresponding polynomials over $\bar{L}_p = \mathbb{Z}[z]/\langle M(z) \rangle$. Subsequently, it calls Algorithm

PRESNF to compute

$$r_p = \mathrm{res}(\phi_\gamma(\phi_p(\check{f}_1)), \phi_\gamma(\phi_p(\check{f}_2)), y) \in \bar{L}_p[x_1, \ldots, x_k].$$

If $r_p = FAIL$, this indicates that a zero divisor was encountered during a call to URES inside PRESNF (Line 2 of PRESNF). In this case, MRESNF discards the current prime p, goes back to Line 7, selects a different prime, and repeats the procedure for that new choice. Otherwise, if $r_p \neq FAIL$, MRESNF maps r_p to $\phi_\gamma^{-1}(r_p) \in L_p[x_1, \ldots, x_k]$ (Line 15). In this way, we avoid applying CRT and RNR to large coefficients in the single-extension representation.

Employing CRT (Line 20) together with RNR (Line 22), MRESNF reconstructs the rational coefficients of $\mathrm{res}(\check{f}_1, \check{f}_2, y)$. If RNR succeeds and the reconstructed polynomial $H \in L[x_1, \ldots, x_k]$ matches the previously computed resultant, then MRESNF terminates and returns

$$r = \frac{H}{s_1^{n_2} \cdot s_2^{n_1}},$$

as $\mathrm{res}(f_1, f_2, y)$, where s_1 and s_2 are defined in Corollary 5.2.

Example 5.4.1 demonstrates how MRESNF works.

Example 5.4.1. *Let* $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 7\rangle$ *and*

$$f_1 = \frac{2}{3}x^2 + \frac{4}{5}yz_2 - 8z_1, \text{ and } f_2 = \frac{1}{7}z_2x + 4yz_1$$

be two polynomials in $L[x, y]$ *with* $n_1 = \deg(f_1, y)$ *and* $n_2 = \deg(f_2, y)$. *We would like to compute* $r = \mathrm{res}(f_1, f_2, y)$ *using the MRESNF algorithm. In a preprocessing process, MRESNF clears fractions by computing*

$$\check{f}_1 = s_1 \cdot f_1 = \frac{15}{2} \cdot f_1 = 5x^2 + 6z_2y - 60z_1, \text{ and} \tag{5.5}$$

$$\check{f}_2 = s_2 \cdot f_2 = 7 \cdot f_2 = z_2x + 28z_1y. \tag{5.6}$$

Assume that MRESNF chooses $p_1 = 7$. *Reducing the polynomials modulo* p_1, *we have*

$$\phi_{p_1}(\check{f}_1) = 5x^2 + 6z_2y + 3z_1, \text{ and } \phi_{p_1}(\check{f}_2) = z_2x \in L_{p_1}[x, y],$$

where $L_{p_1} = \mathbb{Z}_7[z_1, z_2]/\langle z_1^2 - 2, z_2^2 \rangle$. *Next, MRESNF chooses* $C_1 = 6$ *randomly and calls LAminpoly for* $\gamma = z_1 + C_1z_2 = z_1 + 6z_2$ *over* $\mathbb{F} = \mathbb{Z}_7$ *to compute the generator polynomial*

$$M(z) = z^4 + 3z^2 + 4.$$

Hence, we can construct $\bar{L}_{p_1}$ *s.t.*

$$\bar{L}_{p_1} = \mathbb{Z}_7[z]/\langle z^4 + 3z^2 + 4\rangle \cong L_{p_1} = \mathbb{Z}_7[z_1, z_2]/\langle z_1^2 - 2, z_2^2 \rangle.$$

Algorithm 11: MRESNF

Input: non-zero polynomials $f_1, f_2 \in L[x_1, \ldots, x_k, y]$ and $\mathbb{P}_b$ a large set of b-bit primes.
Output: $r = \text{res}(f_1, f_2, y) \in L[x_1, \ldots, x_k]$.

1 $H_{\text{prev}} = 0$ // If the current reconstruction equals the previous one, MRESNF terminates.
2 $prod = 1$
3 Let $\check{f}_1 = s_1 \cdot f_1$ s.t. $s_1 \in \mathbb{Q}_{>0}$ // Clearing fractions, so $F_1, F_2 \in L_{\mathbb{Z}}[x_1, \ldots, x_k, y]$.
4 Let $\check{f}_2 = s_2 \cdot f_2$ s.t. $s_2 \in \mathbb{Q}_{>0}$
5 $n_1, n_2 = \deg(\check{f}_1, y), \deg(\check{f}_2, y)$
6 **while** *true* **do**
7 Choose a new prime $p \in \mathbb{P}_b$ at random s.t. $p \nmid \text{lc}(\check{f}_1) \cdot \text{lc}(\check{f}_2) \cdot \prod_{i=1}^{n} \text{lc}(\check{M}_i)$.
8 Choose $C = (C_1, \ldots, C_{n-1}) \in [1, p)^{n-1}$ at random and set $\gamma = z_1 + \sum_{i=2}^{n} C_{i-1} z_i$.
9 Call Algorithm 4 with inputs $[\phi_p(\check{M}_1), \ldots, \phi_p(\check{M}_n)]$, $\mathbb{F} = \mathbb{Z}_p$, and γ to compute $M(z)$ s.t. $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z) \rangle$.
10 **if** *Algorithm 4 returns FAIL* **then**
11 Go back to Line 7.
12 $r_p = PRESNF(\phi_\gamma(\phi_p(\check{f}_1)), \phi_\gamma(\phi_p(\check{f}_2)), y) \in \bar{L}_p[x_1, \ldots, x_k]$
13 **if** $r_p = FAIL$ **then**
// A zero divisor was encountered in URES.
14 Go back to Line 7.
15 $r_p = \phi_\gamma^{-1}(r_p) \in L_p[x_1, \ldots, x_k]$ // Convert back r_p to its corresponding polynomial over L_p.
16 **if** $prod = 1$ **then**
17 $R = r_p$
18 $prod = p$
19 **else**
20 Using CRT, solve $\{R' \equiv R \mod prod,\ R' \equiv r_p \mod p\}$ for R'
21 Set $R = R'$ and $prod = prod \cdot p$
22 $H =$ Rational Number Reconstruction of R mod $prod$
23 **if** $H \neq FAIL$ *and* $H = H_{prev}$// w.h.p. $H = \text{res}(\check{f}_1, \check{f}_2, y)$
24 **then**
25 $r = H/(s_1^{n_2} \cdot s_2^{n_1})$// Recover $\text{res}(f_1, f_2, y)$ from $\text{res}(F_1, F_2, y)$ (see Corollary 5.2)
26 **return**(r)
27 **else**
28 $presult = H$

Employing ϕ_γ, MRESNF maps $\phi_{p_1}(\check{f}_1)$ and $\phi_{p_1}(\check{f}_2)$ in $L_{p_1}[x, y]$ to their corresponding polynomials in the single extension $\bar{L}_{p_1}[x, y]$:

$$F_{1_p} = \phi_\gamma(\phi_{p_1}(\check{f}_1)) = 5x^2 + (2z^3 + 3z)y + z^3 + z, \text{ and}$$
$$F_{2_p} = \phi_\gamma(\phi_{p_1}(\check{f}_2)) = (5z^3 + 4z)x \in \bar{L}_{p_1}[x, y].$$

Subsequently, MRESNF calls the PRESNF algorithm to compute $\text{res}(F_{1_p}, F_{2_p}, y) \in \bar{L}_{p_1}[x]$. Algorithm PRESNF picks an evaluation point $x = 1 \in [0, 7)$ at random and attempts to compute

$$\text{res}(F_{1_p}(1, y), F_{2_p}(1, y), y) \in \bar{L}_{p_1}$$

using Algorithm URES (see Algorithm 9). In Line 5 of URES, the algorithm tries to compute $\operatorname{monic}(F_{2_p})$ *by inverting* $\operatorname{lc}(F_{2_p}(1,y)) = 5z^3+4z \in \bar{L}_{p_1}$. *However, since* $\gcd(\operatorname{lc}(F_{2_p}(1,y)), M(z)) = z^2+5 \neq 1$, $\operatorname{lc}(F_{2_p}(1,y))$ *is not invertible in* $\bar{L}_{p_1}$, *which leads to the failure of URES over* $\bar{L}_{p_1}$.

Since MRESNF cannot identify whether this failure is due to the choice of the prime $p_1 = 7$ *or the evaluation point* $x = 1$, *it aborts the computation of* $\operatorname{res}(\check{f}_1, \check{f}_2, y)$ *modulo* $p_1 = 7$ *and tries another prime, say* $p_2 = 11$. *After reducing* $\check{f}_1$, $\check{f}_2$ *and the minimal polynomials modulo* p_2, *MRESNF calls LAminpoly for* $\gamma = z_1 + 8z_2$ *to obtain* $M(z) = z^4 + 2z^2 + 3$ *and construct* $\bar{L}_{p_2} \cong L_{p_2}$. *After computing*

$$F_{1_p} = \phi_\gamma(\phi_{p_2}(\check{f}_1)) = 5x^2 + (2z^3+3z)y + 6z^3 + 4z, \text{ and}$$
$$F_{2_p} = \phi_\gamma(\phi_{p_2}(\check{f}_2)) = (4z^3+6z)x + (6z^3+4z)y \in \bar{L}_{p_2}[x,y],$$

MRESNF calls Algorithm PRESNF to compute the resultant over $\bar{L}_{p_2}$. *PRESNF picks* $x = \beta = 2 \in [0, 11)$ *randomly and computes* $F_{1_p}(2,y)$ *and* $F_{2_p}(2,y)$ *in* $\bar{L}_{p_2}[x]$. *This time, the URES succeeds and outputs* $z^3 + 8z + 1 \in \bar{L}_{p_2}$. *After evaluating at three more evaluation points* $x = 7, 10, 6$, *PRESNF interpolates* x *and outputs*

$$R_{p_2} = \operatorname{res}(F_{1_p}, F_{2_p}, y) = (3z^3+2z)x^2 + 9x + 5 \in \bar{L}_{p_2}[x].$$

The MRESNF iterates this procedure for three additional primes $p = 13, 17, 19, 29$ *and computes* $\phi_\gamma^{-1}(R_p) \in L_p[x]$ *for each prime. Then it employs the CRT and RNR to reconstruct the rational coefficients of* $\operatorname{res}(\check{f}_1, \check{f}_2, y) \in L[x_1, \ldots, x_k]$. *After employing the RNR, the reconstructed polynomial does not change for* $p = 19$ *and* $p = 29$ *with* $H = -140z_1x^2 + 42x + 3360$. *By Corollary 5.2, MRESNF computes* $\frac{H}{s_1^{n_2} s_2^{n_1}}$, *where* $s_1 = \frac{15}{2}$ *and* $s_2 = 7$ *(see Equations (5.5) and (5.6)), and returns*

$$\begin{aligned} H = \operatorname{res}(f_1, f_2, y) &= \frac{2}{105}(-140z_1x^2 + 42x + 3360) \\ &= -\frac{8}{3}z_1x^2 + \frac{4}{5}x + 64 \in L[x_1, \ldots, x_k]. \end{aligned}$$

As illustrated in Example 5.4.1, not every choice of a prime p, an evaluation point $\beta \in [0,p)^k$, and $C = (C_1, \ldots, C_{n-1}) \in (0,p)^{n-1}$ leads to a successful reconstruction of the resultant modulo p. We classify the ordered pairs (p, β) and (p, C) into four categories: **lc-bad pairs**, **zero divisor pairs**, **det-bad pairs**, and **good pairs**. The notions of lc-bad and det-bad are the same as in MGCDNF (Definitions 4.4.1 and 4.4.4). However, the notion of a zero divisor pair is slightly different in MRESNF. Similar to MGCDNF, to reduce the chance of hitting a zero divisor in L_p, our Maple implementation of MRESNF utilizes 31-bit primes.

Definition 5.4.1. *Let* $f_1, f_2 \in \bar{L}[x_1, \ldots, x_k][y]$. *Let* p *be a prime and* $\beta \in [0,p)^k$ *be an evaluation point s.t.* (p, β) *is not lc-bad. We call* (p, β) *a* ***zero divisor pair*** *if URES (Algorithm 9) returns FAIL for the inputs* $\phi_p(\check{f}_1)(\beta)$ *and* $\phi_p(\check{f}_2)(\beta) \in \bar{L}_p[y]$.

By Definition 4.4.2, in MGCDNF, a zero divisor pair is defined by the failure of PGCDNF while trying to compute the monic GCD in Lines 2, 5, 6, 7, 9, and 30. In MRESNF, a zero divisor pair is defined by the failure of URES inside PRESNF. Because URES fails if and only if MEA fails (Lemma 5.3.3), these two definitions describe the same obstruction: the algorithm attempts to invert a leading coefficient that is not a unit in the finite ring $\bar{L}_p$.

Example 5.4.2. *Let* $f_1 = (x+20)z_2y + z_1x$ *and* $f_2 = (z_2+5+x)y + z_1x$ *be two polynomials listed in lexicographic order with* $x > y$ *in* $L[x,y]$*, where* $L = \mathbb{Q}[z_1,z_2]/\langle z_1^2-2, z_2^2-3\rangle$*. Computing* $\operatorname{res}(f_1,f_2,y)$ *using MRESNF, the ordered pair* $(p,\beta) = (23,3)$ *is lc-bad since* $\operatorname{lc}(\check{f}_1)(x=3) = 0 \mod p$*. Moreover,* $(p,\beta) = (11,0)$ *is a zero divisor pair because* $\operatorname{lc}(\check{f}_2)(x=0) = z_2+5$ *is not invertible over* $\mathbb{Z}_{11}[z_1,z_2]/\langle z_1^2-2, z_2^2-3\rangle$*, since* $\gcd(z_2^2-3, z_2+5) = z_2+6 \neq 1$ *over* $\mathbb{Z}_{11}[z_2]$*.*

Definition 5.4.2. *Let* p *be a prime,* $\beta \in [0,p)^k$*, and* $C = (C_1,\ldots,C_{n-1}) \in (0,p)^{n-1}$*. We call* (p,β) *a good pair if it is neither a zero divisor nor an lc-bad pair. Also,* (p,C) *is a good pair if it is not a det-bad pair.*

As mentioned, MRESNF is a Monte Carlo algorithm, so it may return an incorrect result with low probability. The following example illustrates such a case.

Example 5.4.3. *Let* p_1 *and* p_2 *be two distinct primes. Let*

$$f_1 = z_2y - x_1 + z_1$$
$$f_2 = z_2y - p_1p_2z_1x_2$$

be two polynomials in $\mathbb{Q}[z_1,z_2]/\langle z_1^2-2, z_2^2-3\rangle[x_1,x_2][y]$ *with*

$$\operatorname{res}(f_1,f_2,y) = -(p_1p_2)z_1z_2x_2 + z_2x_1 - z_1z_2.$$

In Line 7 of MRESNF, in the first iteration, if $p = p_1$ *is chosen, then after RNR and scaling in Line 25, the algorithm may return*

$$r = z_2x_1 - z_1z_2.$$

In the second iteration, if $p = p_2$ *is chosen, then the algorithm may again obtain* $r = z_2x_1 - z_1z_2$ *as the resultant. Since two successive results are identical, MRESNF terminates and returns* $r = z_2x_1 - z_1z_2$*, which is not the correct resultant.*

5.5 Complexity

For simplicity, the following notation will remain unchanged throughout this section.

Notation 5.1. *Let* $f_1, f_2 \in R[x_1,x_2,\ldots,x_k,y]$*, where* $R = L$ *or* $R = \bar{L}_p$*.*

- $d = [L:\mathbb{Q}]$.
- $\#f$ *denotes the number of terms of* f *in the variables* $x_1,x_2,\ldots,x_k,y$.
- $n_1 = \deg(f_1,y)$, $n_2 = \deg(f_2,y)$.
- $D = \max\{n_1,\, n_2,\, \max_{i=1}^{k}(\deg(f_1,x_i), \deg(f_2,x_i))\}$.
- N *is the number of good primes needed to reconstruct the resultant in MRESNF.*
- $H_M = \log_2\left(\max_{i=1}^{n}(H(\check{M}_i))\right)$ *(see Definition 4.5.1).*
- $C = \log_2\left(\max(H(\check{f}_1), H(\check{f}_2))\right)$.

Remark 5.3. *Since our implementation currently uses classical quadratic polynomial arithmetic, we assume that multiplication and inverses in $\bar{L}_p$ cost $\mathcal{O}(d^2)$ in $\mathbb{Z}_p$. Moreover, the scalar multiplication and addition in $\bar{L}_p$ cost $\mathcal{O}(d)$ in $\mathbb{Z}_p$.*

Theorem 5.5.1. *Using Notation 5.1, the algorithm PRESNF performs*

$$\mathcal{O}(kd^2D^{2k+2})$$

arithmetic operations in $\mathbb{Z}_p$.

Proof. Let $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k, y]$ and $r = \operatorname{res}(f_1, f_2, y) \in \bar{L}_p[x_1, \ldots, x_k]$. Let $T(k)$ denote the number of arithmetic operations in $\mathbb{Z}_p$ performed by PRESNF. In the base case of PRESNF (Line 2), the algorithm calls URES to compute the resultant of univariate polynomials in $\bar{L}_p[y]$. URES performs $\mathcal{O}(D^2)$ operations in $\bar{L}_p$ (see Lemma 5.3.5). Accordingly, when $k = 0$ (the univariate case), the PRESNF costs $\mathcal{O}(d^2D^2)$ arithmetic operations in $\mathbb{Z}_p$, so $T(0) = d^2D^2$. To obtain a formula for $T(k)$ when $k \geq 1$, we first compute the time complexity of the most dominant operations at each iteration of PRESNF: (i) the evaluations at $x_k = \beta$ in Line 10 and (ii) the incremental interpolation steps in Lines 19–21.

(i) **Evaluation:** In Line 10, PRESNF evaluates f_1 and f_2 at $x_k = \beta_k \in [0, p)$. To speed this up, we precompute the powers β_k^j for $0 \leq j \leq D$, which has negligible cost. For $i = 1, 2$,

$$\#f_i \leq (n_i + 1) \cdot \prod_{j=1}^{k} (\deg(f_i, x_j) + 1) \leq (D + 1)^{k+1},$$

so $\#f_1 + \#f_2 \leq 2(D + 1)^{k+1}$. Therefore, evaluating f_1 and f_2 at a single evaluation point $x_k = \beta_k$ costs $\mathcal{O}(d(D + 1)^{k+1})$ arithmetic operations in $\mathbb{Z}_p$. To interpolate x_k in r, we need at most $\deg(r, x_k) + 1$ evaluation points. By the construction of the Sylvester matrix $\operatorname{sylv}(f_1, f_2, y)$, for each $1 \leq i \leq k$, we have

$$\begin{aligned} \deg(r, x_i) &\leq n_1 \deg(f_2, x_i) + n_2 \deg(f_1, x_i) \\ &\leq 2D^2. \end{aligned}$$

Accordingly, the total evaluation cost of evaluating both f_1 and f_2 for at most $2D^2 + 1$ evaluation points for x_k costs

$$\mathcal{O}(d\,(2D^2 + 1)(2(D + 1)^{k+1})) = \mathcal{O}(dD^{k+3}). \tag{5.7}$$

arithmetic operations in $\mathbb{Z}_p$.

(ii) **Interpolation:** Algorithm PRESNF is called once to interpolate x_k from $2D^2 + 1$ values of $r(z, x_1 \ldots, x_{k-1}, x_k = \beta_k)$. It does at most $(2D^2 + 1)^{k-1}$ univariate interpolations in x_k, each of which costs $\mathcal{O}((2D^2 + 1)^2)$. Therefore, interpolation in PRESNF costs

$$\mathcal{O}(d(2D^2 + 1)^2(2D^2 + 1)^{k-1}) = \mathcal{O}(dD^{2k+2}) \tag{5.8}$$

arithmetic operations in $\mathbb{Z}_p$.

For each variable x_i, we require at most $2D^2+1$ evaluation points. Hence, the recurrence relation is

$$\begin{aligned} T(0) &= d^2D^2 \\ T(k) &= (2D^2+1)T(k-1) + \underbrace{\mathcal{O}(dD^{k+3})}_{\text{Evaluation}} + \underbrace{\mathcal{O}(dD^{2k+2})}_{\text{Interpolation}} \quad \text{for } k \geq 1 \\ &= \mathcal{O}(D^2)T(k-1) + \mathcal{O}(dD^{2k+2}). \end{aligned}$$

Solving the above recurrence, we obtain,

$$T(k) = \mathcal{O}(d^2D^{2k+2}), \quad \text{for } k \geq 1.$$

□

Theorem 5.5.2. *Considering Notation 5.1, Algorithm MRESNF costs*

$$\mathcal{O}\left(Nd(H_M + CD^{k+1} + d^2 + dD^{2k} + dD^{2k+2} + N^2D^{2k})\right)$$

arithmetic operations in $\mathbb{Z}_p$.

Proof. Let $f_1, f_2 \in L[x_1, \ldots, x_k, y]$ and $r = \operatorname{res}(f_1, f_2, y) \in L[x_1, \ldots, x_k]$. Similar to Theorem 5.5.1, $\#f_1 + \#f_2 \leq 2(D+1)^{k+1}$. Moreover,

$$\#r \leq \prod_{j=1}^{k} (\deg(r, x_j) + 1) \leq (2D^2+1)^k.$$

The cost of MRESNF over $\mathbb{Z}_p$ is broken down into the following components:

- **Modular homomorphism** ϕ_p: Algorithm MRES reduces the minimal polynomials $\check{M}_1, \ldots, \check{M}_n$ and the input polynomials $\check{f}_1$ and $\check{f}_2$ modulo each of the N primes, which costs

$$\mathcal{O}(Nd(H_M + C(2(D+1)^{k+1}))) = \mathcal{O}(Nd(H_M + CD^{k+1})). \tag{5.9}$$

- **Primitive homomorphism** ϕ_γ: The time complexity of building the matrix A and A^{-1} in Algorithm 4 for N primes is $\mathcal{O}(Nd^3)$. The running time complexity of applying ϕ_γ to the non-zero terms of f_1 and f_2 for N primes is

$$\mathcal{O}(Nd^2 \underbrace{(2(D+1)^{k+1})}_{\#f_1+\#f_2}) = \mathcal{O}(Nd^2D^{k+1}). \tag{5.10}$$

- **Inverse of primitive homomorphism** ϕ_γ^{-1}: Let R_p be the output of Algorithm PRES in Line 12 of MRES. The time complexity of calling ϕ_γ^{-1} for R_p in Line 15 for N primes is

$$\mathcal{O}(Nd^2 \underbrace{(2D^2+1)^k}_{\#r}) = \mathcal{O}(Nd^2D^{2k}). \tag{5.11}$$

- **PRESNF**: According to Theorem 5.5.1, calling PRESNF in Line 12 of MRES for N primes costs

$$\mathcal{O}(Nd^2D^{2k+2}). \tag{5.12}$$

- **CRT and RNR**: Finally, Algorithm MRES reconstructs $\mathcal{O}(d\#r)$ rational coefficients in Lines 20 and 22 which costs $\mathcal{O}(N^3)$ each. Hence, the total cost of CRT and RNR is

$$\mathcal{O}(N^3 d \underbrace{(2D^2+1)^k}_{\#r}) = \mathcal{O}(dN^3D^{2k}). \tag{5.13}$$

Adding Equations (5.9) to (5.13), we obtain

$$\begin{aligned}&\mathcal{O}(\underbrace{Nd(H_M + CD^{k+1})}_{\phi_p \text{ 5.9}} + \underbrace{Nd^3}_{\text{Matrix A (5.10)}} + \underbrace{Nd^2D^{k+1}}_{\phi_\gamma \text{ (5.10)}} + \underbrace{Nd^2D^{2k}}_{\phi_\gamma^{-1} \text{ (5.11)}} + \underbrace{Nd^2D^{2k+2}}_{\text{PRESNF (5.12)}} + \underbrace{N^3dD^{2k}}_{\text{CRT,RNR (5.13)}})\\ &= \mathcal{O}\left(Nd(H_M + CD^{k+1} + d^2 + dD^{2k} + dD^{2k+2} + N^2D^{2k})\right).\end{aligned}$$

□

Remark 5.4. *The cost of the LAminpoly algorithm for computing A and calculating A^{-1} is $\mathcal{O}(d^3)$. This cost is negligible when*

$$\begin{aligned}Nd^3 &\ll \underbrace{Nkd^2D^{2k+2}}_{PRESNF}, \text{ that is,}\\ d &\ll kD^{2k+2}.\end{aligned}$$

Remark 5.5. *In the dense case, the resultant has size $\mathcal{O}(dD^{2k})$. Since*

$$\begin{aligned}\deg(\operatorname{res}(f_1, f_2, y), x_i) &\le \deg(f_1, y)\deg(f_2, x_i) + \deg(f_2, y)\deg(f_1, x_i)\\ &\le 2D^2,\end{aligned}$$

this cost is unavoidable.

5.6 Benchmark

We implemented the MRESNF algorithm and its subalgorithms in Maple [28]. Similar to MGCDNF, we used the recursive dense data structure from [34] to represent elements of $L = \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and polynomials in $L[x_1, \ldots, x_k]$. Moreover, for the modular reductions, we used the set of primes $\mathbb{P}_{31} = \{\text{all 31-bit primes}\}$.

This section presents two timing benchmarks. All timings were obtained on an Intel Core i7-6700. Let $f_1, f_2 \in L[x_1, x_2]$ with degree D_f in x_1 and x_2. The degree of $r = \operatorname{res}(f_1, f_2, x_1)$ is denoted by D_r. Furthermore, column N denotes the number of primes needed by the algorithm. In both Tables 4.1 and 4.2, the timing columns report the time spent in the following:

- Column MRESNF 1 presents the time for our algorithm, MRESNF, using ϕ_γ and computing over $\bar{L}_p$.

- Column MRESNF 2 is the time for MRESNF if we do not use ϕ_γ and compute over L_p.
- Column LAMP is the time spent in Algorithm 4.
- Column PRESNF is the time spent in Algorithm 10.

The speedup achieved by employing ϕ_γ can be observed by comparing the PRESNF columns for MRESNF 1 and MRESNF 2. The first benchmark, Table 5.2, presents timings of the resultant computations in $L[x_1, x_2]$ where the number field $L = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11})$ has degree $d = 32$ and $n = 5$ extensions. In Table 5.2, the input polynomials f_1 and f_2 have degree D_f in x_1 and x_2, and $\text{res}(f_1, f_2, x_1) \in L[x_2]$ has degree D_r in x_2. Observe that, in the MRESNF1 column, the cost of LAminpoly, which is $\mathcal{O}(Nd^3)$, is negligible.

Table 5.2: **Timings in CPU seconds for computing** $\text{res}(f_1, f_2, x_1)$ **over an algebraic number field of degree** $d = 32$**.**

d	n	N	D_f	D_r	MRESNF 1				MRESNF 2		
					time	LAMP	PRESNF	MEA	time	PRESNF	MEA
32	5	4	2	6	0.234	0.031	0.142	36	0.235	0.203	36
32	5	6	4	20	1.860	0.064	1.673	198	2.672	2.608	198
32	5	9	6	48	8.843	0.094	8.156	657	24.657	24.206	657
32	5	12	8	80	23.390	0.139	21.922	1548	79.563	78.734	1548
32	5	15	10	122	78.219	0.155	74.408	3015	242.813	240.126	3015
32	5	20	12	192	200.360	0.156	188.641	5780	726.547	718.436	5780
32	5	26	14	269	487.484	0.326	457.922	10218	>1000	(-)	(-)
32	5	30	16	272	517.516	0.328	502.435	15390	>1000	(-)	(-)
32	5	35	18	366	1052.219	0.734	1016.718	22715	>1000	(-)	(-)

The second benchmark (Table 5.3) shows timings for computing the resultant of two polynomials f_1 and f_2 in $L[x_1, x_2]$, where L is an algebraic number field of degree d with n extensions. In this table, the input polynomials have degrees $D_f = 8$ in x_1 and x_2. Observe that the cost of LAminpoly is significant when d is large.

Table 5.3: **Timings in CPU seconds for computing** $\mathrm{res}(f_1, f_2, x_1)$ **over an algebraic number field of degree** d**.**

d	n	N	D_f	MRESNF 1				MRESNF 2		
				time	LAMP	PRESNF	MEA	time	PRESNF	MEA
4	2	10	8	7.953	0.016	7.780	1290	12.281	12.188	1290
8	3	11	8	11.203	0.000	10.812	1419	25.047	24.797	1419
16	4	12	8	17.859	0.015	16.829	1548	50.422	49.716	1548
32	5	12	8	24.235	0.155	22.515	1548	81.406	80.390	1548
64	6	12	8	41.640	0.595	38.515	1548	159.782	158.156	1548
128	7	13	8	83.406	2.219	76.251	1677	323.547	320.611	1677
256	8	13	8	180.204	9.736	159.454	1677	654.687	649.342	1677
512	9	14	8	366.360	56.077	296.562	1806	>1000	>1000	>1000
1024	10	14	8	1184.891	355.629	729.766	1806	>1000	(-)	(-)

For the details of the benchmark, see https://github.com/MahsAnsari/Mahsa_Ansari_Thesis.

Chapter 6

Failure probability

6.1 Summary of contributions

This chapter is based on our paper published in the Proceedings of CASC '25 [4]. In Sections 4.4.3 and 4.4.2, we categorized the primes and evaluation points in MGCDNF (and its subalgorithm PGCDNF) into five types: **lc-bad pairs**, **zero divisor pairs**, **det-bad pairs**, **unlucky primes**, and **unlucky evaluation points**. Similarly, in Section 5.4.2, we classified the primes and evaluation points in MRESNF (and its subalgorithm PRESNF) as **lc-bad pairs**, **zero divisor pairs**, and **det-bad pairs**. When MGCDNF and PGCDNF (also MRESNF and PRESNF) encounter a failure; control is passed back to MGCDNF (MRESNF), which restarts with a new prime and a new choice for the integers $C_1, \ldots, C_{n-1} \in (0, p)$.

In this chapter, we analyze the probability of encountering each category. Since the definitions of lc-bad pairs and det-bad pairs are identical in both algorithms, we derive bounds for these events only in MGCDNF. The same bounds then apply to MRESNF. The notion of a zero divisor pair also overlaps in the two algorithms. Accordingly, we prove the required bounds for zero divisor pairs in the MGCDNF setting in Section 6.3 and reuse them for MRESNF in Section 6.4. Section 6.5 summarizes the resulting bounds and shows that they are polynomial in the sizes of the input and output, namely, d the degree of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, the size of the integer coefficients in $\check{M}_1, \check{M}_2, \ldots, \check{M}_n$, $\check{f}_1, \check{f}_2$, and the cofactors $\check{h}_1, \check{h}_2$, the degrees of f_1, f_2 in $x_1, x_2, \ldots, x_k$, the number of evaluation points and primes used, and the number of terms of f_1, f_2, and g. Let

$$D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i)).$$

Then the number of terms of each of f_1, f_2, and g is bounded by $(1 + D)^k$. We use this bound in our analysis.

6.2 Introduction

Notation 6.1. *For simplicity, the following notations will remain unchanged and will be used consistently throughout this chapter.*

- *Let $f \in R[x_1, \ldots, x_k]$. Then $\#f$ denotes the number of monomials of f in $x_1, \ldots, x_k$.*
- *$d_i = \deg(\check{M}_i, z_i)$ where $\check{M}_i$ is the semi-associate of the ith minimal polynomial.*

- $\mathbb{P}_b = \{$*all b-bit primes*$\}$*, that is, primes in* $(2^{b-1}, 2^b)$*. Let* $\mid \mathbb{P}_b \mid$ *denote the cardinality of* $\mathbb{P}_b$*. In our current implementation, we use* 31*-bit primes, for which* $\mid \mathbb{P}_{31} \mid = 50,697,537$*.*

We begin by reviewing two key tools used in this chapter: Hadamard's bound and the Schwartz–Zippel lemma.

To bound the probability of hitting a det-bad pair, we need to bound the absolute value of a determinant in $\mathbb{Z}$. Furthermore, to bound the probabilities of encountering a zero divisor pair and an unlucky prime, we will need to bound the height of a suitable resultant over $\mathbb{Z}$. By Definition 5.2.1, the resultant of two polynomials can be expressed as the determinant of their Sylvester matrix. Consequently, bounding the height of a resultant reduces to bounding the height of the determinant of the corresponding Sylvester matrix. We obtain such a determinant bound using Hadamard's Theorem 6.2.1.

Theorem 6.2.1. *[19, Hadamard's bound, Corollary 7.8.2] Let* A *be an* $n \times n$ *matrix with* $A_{i,j} \in \mathbb{Z}$*. Then*

$$\mid \det(A) \mid \leq \prod_{j=1}^{n} \sqrt{\sum_{i=1}^{n} A_{i,j}^2}.$$

Notice 6.1. *Let* A *be an* $n \times n$ *matrix with* $A_{i,j} \in \mathbb{Z}$*, and let* $Col(j, A)$ *denote the* j*th column of* A*. Hadamard's bound can be equivalently expressed as*

$$\mid \det(A) \mid \leq \prod_{j=1}^{n} \|Col(j, A)\|_2,$$

where $\|Col(j, A)\|_2$ *denotes the 2-norm of* $Col(j, A)$*.*

Example 6.2.1. *Let* $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$*. Using Hadamard's bound, we have*

$$\mid \det(A) \mid = 4 \leq \prod_{j=1}^{3} \|Col(j, A)\|_2 \leq 3\sqrt{3}.$$

To estimate the probability of encountering an lc-bad pair or an unlucky evaluation point, we need to bound the probability that a multivariate polynomial is zero after evaluating it at a point. In the univariate case, when $f \in \mathbb{Z}[x]$ is a non-zero polynomial and $\beta \in [0, p)$, we have

$$\text{Prob}[f(\beta) = 0] \leq \frac{\deg(f)}{p},$$

because a univariate polynomial of degree n over $\mathbb{Z}$ has at most n roots in $\mathbb{Z}$. Now, consider the multivariate case where $f \in \mathbb{Z}[x_1, \ldots, x_k]$ and $\beta \in [0, p)^k$. What can we say about $\text{Prob}[f(\beta) = 0]$? To answer this, we invoke the Schwartz–Zippel lemma (also known as the DeMillo–Lipton–Schwartz–Zippel lemma)[30, 40, 11].

Lemma 6.2.2. *[30, Schwartz-Zippel lemma] Let* R *be an integral domain, and let* $S \subseteq R$ *be finite. Let* $f \in R[x_1, x_2, \ldots, x_k]$ *be a non-zero polynomial with total degree* D*. Then the number of roots of* f *in* S^k *is at most* $D|S|^{k-1}$*. Hence, if* β *is chosen at random from* S^k*, then* $\text{Prob}[f(\beta) = 0] \leq \frac{D}{|S|}$ *.*

The Schwartz–Zippel lemma is widely used as a probabilistic method for testing whether a given multivariate polynomial is identically zero by evaluating the polynomial at a randomly chosen point.The following example illustrates the manner in which the lemma is applied in Section 6.3.1 to bound the probability of hitting an lc-bad pair.

Example 6.2.2. *Let* $L = \mathbb{Q}[z]/\langle z^2 - 2\rangle$ *and let* $f \in L[x_2, x_3][x_1]$ *s.t.*

$$\check{f} = (x_2^4 + 2x_3^2)zx_1^2 + 2x_2 + 9z.$$

Let $p = 1009$ *and* $\beta = (\beta_1, \beta_2) \in [0, p)^2$ *be randomly chosen. We would like to find the probability that* $p \mid \mathrm{lc}(f, x_1)(\beta)$. *Since* $\mathrm{lc}(f, x_1) = zx_2^4 + 2zx_3^2 \in \mathbb{Z}[z][x_2, x_3]$, *the Schwartz-Zippel lemma gives*

$$\mathrm{Prob}[\mathrm{lc}(f, x_1)(\beta) = 0] \leq \frac{4}{1009}.$$

6.3 Failure probability analysis of MGCDNF

6.3.1 Lc-bad pairs

Let $f_1, f_2 \in L_{\mathbb{Z}}[x_1, \ldots, x_k]$ be non-zero polynomials with $\deg(f_1, x_1) \geq \deg(f_2, x_1) \geq 0$. Let p be a prime and $\beta \in [0, p)^{k-1}$ be an evaluation point. By Definition 4.4.1, the ordered pair (p, β) is called **lc-bad** if either $p \mid \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i, z_i)$ or $p \mid \mathrm{lc}(f_1, x_1)(\beta) \cdot \mathrm{lc}(f_2, x_1)(\beta)$. The definition of lc-bad pairs is identical in both the MGCDNF and MRESNF algorithms. Consequently, the results presented in this section apply to both algorithms. In Theorem 6.3.1, we bound the probability that Algorithm MGCDNF (or MRESNF) encounters an lc-bad pair.

Theorem 6.3.1. *Let* $f_1, f_2 \in L_{\mathbb{Z}}[x_1, \ldots, x_k]$ *with* $\deg(f_1, x_1) \geq \deg(f_2, x_1) \geq 0$ *and* $k > 1$. *Assume*

- p *is chosen at random from* $\mathbb{P}_b$,
- β *is chosen at random from* $[0, p)^{k-1}$,
- $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$,
- $\|\check{M}_i\| \leq 2^m$ *for* $1 \leq i \leq n$, *and*
- $\|f_i\|_\infty \leq 2^h$ *for* $i = 1, 2$.

Then

$$\mathrm{Prob}[(p, \beta) \ \textit{is lc-bad}] \leq \frac{2h}{(b-1)\cdot \mid \mathbb{P}_b \mid} + \frac{2(k-1)D}{p} + \frac{nm}{(b-1) \mid \mathbb{P}_b \mid}.$$

Proof. Let $L_{f_i} = \mathrm{lc}(f_i, x_1) \in L_{\mathbb{Z}}[x_2, \ldots, x_k]$ for $i = 1, 2$. Write

$$L_{f_1} = \mathrm{lc}(f_1, x_1) = \sum_{i=1}^{T} b_{a_i}(X) Z^{a_i} \in \mathbb{Z}[x_2, \ldots, x_k][z_1, \ldots, z_n], \tag{6.1}$$

where $a_i = (a_{i_1}, \ldots, a_{i_n}) \in \mathbb{Z}^n_{\geq 0}$, $Z^{a_i} = z_1^{a_{i_1}} \cdots z_n^{a_{i_n}}$, and $b_{a_i}(X) \in \mathbb{Z}[x_2, \ldots, x_k]$. We can write L_{f_2} in the same form.

By definition,

$$\begin{aligned}\text{Prob}[(p,\beta) \text{ is lc-bad}] &\leq \text{Prob}[p \mid L_{f_1} \cdot L_{f_2} \ \vee L_{f_1}(\beta) \cdot L_{f_2}(\beta) = 0 \ \vee \ p \mid \prod_{i=1}^{n} \text{lc}(\check{M}_i, z_i)] \\ &\leq \text{Prob}[p \mid L_{f_1} \cdot L_{f_2}] + \text{Prob}[L_{f_1}(\beta) \cdot L_{f_2}(\beta) = 0] + \sum_{i=1}^{n} \text{Prob}[p \mid \text{lc}(\check{M}_i, z_i)].\end{aligned} \tag{6.2}$$

First, we bound

$$\text{Prob}[p \mid L_{f_1} \cdot L_{f_2}] \leq \text{Prob}[p \mid L_{f_1}] + \text{Prob}[p \mid L_{f_2}]. \tag{6.3}$$

if $p \mid L_{f_1}$, from Equation (6.1), $p \mid b_{a_i}(X)$ for each $1 \leq i \leq T$. That is,

$$\begin{aligned}\text{Prob}[p \mid L_{f_1}] &= \text{Prob}[p \mid b_{a_1}(X) \wedge \ldots \wedge p \mid b_{a_T}(X)] \\ &\leq \text{Prob}[p \mid b_{a_1}(X)].\end{aligned} \tag{6.4}$$

Since $\|b_{a_1}\|_\infty \leq 2^h$ and $2^{b-1} < p < 2^b$. The number of primes in $\mathbb{P}_b$ that divides $\|b_{a_1}\|_\infty$ is at most

$$\lfloor \log_{2^{b-1}} 2^h \rfloor = \lfloor \frac{h}{b-1} \rfloor.$$

Therefore,

$$\begin{aligned}\text{Prob}[p \mid L_{f_1}] &\leq \text{Prob}[p \mid b_{a_1}] \\ &\leq \frac{\lfloor \frac{h}{b-1} \rfloor}{\mid \mathbb{P}_b \mid} \\ &\leq \frac{h}{(b-1) \cdot \mid \mathbb{P}_b \mid}.\end{aligned} \tag{6.5}$$

The same argument gives

$$\text{Prob}[p \mid L_{f_2}] \leq \frac{h}{(b-1) \cdot \mid \mathbb{P}_b \mid}. \tag{6.6}$$

Substituting Equations 6.5 and 6.6 into Equation 6.3, we obtain

$$\text{Prob}[p \mid L_{f_1} \cdot L_{f_2}] \leq \text{Prob}[p \mid L_{f_1}] + \text{Prob}[p \mid L_{f_2}] \leq \frac{2h}{(b-1) \cdot \mid \mathbb{P}_b \mid}. \tag{6.7}$$

Now, suppose that $p \nmid L_{f_1} \cdot L_{f_2}$. Hence, for L_{f_1}, there exists $1 \leq i \leq T$ s.t. $b_{a_i}(X) \neq 0$. Without loss of generality, let $b_{a_1}(X) \neq 0$. We would like to bound

$$\text{Prob}[L_{f_1}(\beta) \cdot L_{f_2}(\beta) = 0] \leq \text{Prob}[L_{f_1}(\beta) = 0] + \text{Prob}[L_{f_2}(\beta) = 0]. \tag{6.8}$$

From Equation 6.1 and since $b_{a_1}(X) \neq 0$, we have

$$\begin{aligned}\text{Prob}[L_{f_1}(\beta) = 0] &= \text{Prob}[b_{a_1}(\beta) = 0 \wedge \ldots \wedge b_{a_T}(\beta) = 0] \\ &\leq \text{Prob}[b_{a_1}(\beta) = 0].\end{aligned} \tag{6.9}$$

Since $\deg(b_{a_1}(X)) \leq (k-1)D$, from Lemma 6.2.2, we obtain

$$\begin{aligned}\text{Prob}[L_{f_1}(\beta) = 0] &\leq \text{Prob}[b_{a_1}(\beta) = 0] \\ &\leq \frac{\deg(b_{a_1})}{p} \\ &\leq \frac{(k-1)D}{p}. \end{aligned} \tag{6.10}$$

The same argument applies for L_{f_2}. Therefore,

$$\text{Prob}[L_{f_1}(\beta) \cdot L_{f_2}(\beta) = 0] \leq \frac{2(k-1)D}{p}. \tag{6.11}$$

Next, write $\check{M}_i(z_i) = l_i z_i^{d_i} + \sum_{j=1}^{d_i-1} a_{i,j} z_i^j$, where $a_{i,j} \in \mathbb{Z}[z_1, \ldots, z_{i-1}]/\langle \check{M}_1(z_1), \ldots, \check{M}_{i-1}(z_{i-1})\rangle$ and $l_i \in \mathbb{Z}$. By assumption, $\mid l_i \mid \leq 2^m$ and $p > 2^{b-1}$. Thus,

$$\begin{aligned}\text{Prob}[p \mid \text{lc}(\check{M}_i, z_i)] = \text{Prob}[p \mid l_i] &\leq \frac{\lfloor \log_{2^{b-1}} \mid l_i \mid \rfloor}{\mid \mathbb{P}_b \mid} \leq \frac{\lfloor \log_{2^{b-1}} 2^m \rfloor}{\mid \mathbb{P}_b \mid} = \frac{\lfloor \frac{\log_2 2^m}{\log_2 2^{b-1}} \rfloor}{\mid \mathbb{P}_b \mid} = \frac{\lfloor \frac{m}{b-1} \rfloor}{\mid \mathbb{P}_b \mid} \\ &\leq \frac{m}{(b-1) \mid \mathbb{P}_b \mid} \quad \text{for} \quad 1 \leq i \leq n. \end{aligned}$$

By summing over $i = 1, \ldots, n$, we obtain

$$\sum_{i=1}^{n} \text{Prob}[p \mid \text{lc}(\check{M}_i, z_i)] \leq \frac{nm}{(b-1) \mid \mathbb{P}_b \mid}. \tag{6.12}$$

Substituting Equations (6.7), (6.11), and (6.12) into Equation (6.2) completes the proof.

□

Remark 6.1. *In the above theorem, we can make the bound $\frac{2(k-1)D}{p}$ independent of p. Since $p \in \mathbb{P}_b$, we have $2^{b-1} < p < 2^b$. Thus,*

$$\text{Prob}[\text{lc}(f_1, x_1)(\beta) = 0 \ \vee \ \text{lc}(f_2, x_1)(\beta) = 0] \leq \frac{2(k-1)D}{p} \leq \frac{2(k-1)D}{2^{b-1}}.$$

Remark 6.2. *In Theorem 6.3.1, one could replace the assumption $\|\check{M}_i\|_\infty \leq 2^m$ by the tighter assumption $|\text{lc}(\check{M}_i, z_i)| \leq 2^m$. However, in order to make the final bounds for the failure probabilities, as stated in Section 6.5, depend on fewer parameters, we sacrifice this tightness.*

Example 6.3.1. *Let $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - \frac{2}{5}, z_2^2 - \frac{7}{3}\rangle$ and $f_1, f_2 \in L[x_1, x_2]$ s.t.*

$$\begin{aligned} f_1 &= (87x_1^5x_2^5 + 97z_2z_1x_1 - 56z_2)(z_2 + z_1x_1x_2) \\ f_2 &= (17z_2x_1^4x_2^3 + 4z_2z_1x_1x_2 - 82z_1)(z_2 + z_1x_1x_2). \end{aligned}$$

Thus,

$$\begin{aligned} \check{f}_1 &= 1305\, z_1\, x_2^6x_1^6 + 1305\, z_2\, x_2^5x_1^5 + 582\, z_2\, x_2\, x_1^2 + (-840\, z_2z_1\, x_2 + 3395\, z_1)\, x_1 - 1960, \\ \check{f}_2 &= 255\, z_2z_1\, x_2^4x_1^5 + 595\, x_2^3x_1^4 + 24\, z_2\, x_2^2x_1^2 + (140\, z_1 - 492)\, x_2x_1 - 1230\, z_2z_1. \end{aligned}$$

In this example,

- *the number of variables is* $k = 2$*, and the number of extensions is* $n = 2$*,*
- $p \in \mathbb{P}_{31}$*, so* $2^{30} < p < 2^{31}$*,*
- $\beta = (\beta_1, \ldots, \beta_{k-1}) \in [0, p)^{k-1}$*,*
- $\check{M}_1 = 5z_1^2 - 2$ *and* $\check{M}_2 = 3z_2^2 - 7$*,*
- $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i)) = 6$*,*
- $\|\mathrm{lc}(\check{M}_i, z_i)\|_\infty \leq 2^m = 2^3$ *for* $i = 1, 2$*, and*
- $\|f_i\|_\infty \leq 2^h = 2^{12}$ *for* $i = 1, 2$*.*

To bound the probability that (p, β) *is an lc-bad pair, we use Theorem 6.3.1. Therefore,*

$$\begin{aligned}\mathrm{Prob}[(p,\beta) \text{ is lc-bad}] &\leq \frac{2h}{(b-1)\cdot \mid \mathbb{P}_b \mid} + \frac{2(k-1)D}{p} + \frac{nm}{(b-1) \mid \mathbb{P}_b \mid} \\ &= \frac{24}{30 \times 50697537} + \frac{12}{2^{30}} + \frac{6}{30 \times 50697537} \\ &\approx 3.09 \times 10^{-8}.\end{aligned}$$

Now, instead of using the bounds, we use the actual values: $\deg(\mathrm{lc}(f_1, x_1)) = 6$*,* $\deg(\mathrm{lc}(f_2, x_1)) = 4$*,* $\mathrm{lc}(\check{M}_1, z_1)\|_\infty = 5 \leq 2^3$*,* $\|\mathrm{lc}(\check{M}_2, z_2)\|_\infty = 7 \leq 2^3$*, and* $\|f_1\|_\infty = 3395 \leq 2^{12}$*, and* $\|f_2\|_\infty = 1230 \leq 2^{12}$*. Since there is no 31-bit prime that divides either* $\|f_1\|_\infty$ *or* $\|f_2\|_\infty$*, we have*

$$\mathrm{Prob}[p \mid \mathrm{lc}(f_1, x_1) \ \vee \ p \mid \mathrm{lc}(f_2, x_1)] = 0.$$

Moreover, for $i = 1, 2$*,* $\mid \mathrm{lc}(\check{M}_i, z_i) \mid$ *does not have any 31-bit prime factor which implies that*

$$\mathrm{Prob}[p \mid \mathrm{lc}(M_1, z_1) \ \vee \ p \mid \mathrm{lc}(M_2, z_1)] = 0.$$

On the other hand, each $\mathrm{lc}(f_1, x_1) = 1305\, x_2^6$ *and* $\mathrm{lc}(f_2, x_1) = 255\, x_2^4$ *has only one root* $x_2 = 0$ *which implies that*

$$\mathrm{Prob}[\mathrm{lc}(f_1, x_1) = 0 \ \vee \mathrm{lc}(f_2, x_1) = 0] \leq \frac{1}{2^{30}}.$$

Therefore,

$$\mathrm{Prob}[(p,\beta) \text{ is lc-bad}] \leq \frac{1}{2^{30}} \approx 9.31 \times 10^{-10}.$$

6.3.2 Det-bad pairs

Notation 6.2. *For simplicity, we keep the following notation fixed throughout this section.*

- $d = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$*.*
- p *is chosen randomly from* $\mathbb{P}_b$ *s.t.* $p \nmid \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i, z_i)$*.*
- $C = (C_1, \ldots, C_{n-1}) \in (0, p)^{n-1}$ *is chosen randomly.*
- $\gamma = z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$*.*

- $B_L = \{\prod_{i=1}^{n} z_i^{e_i} \mid 0 \le e_i \le d_i - 1\}$ *is a basis for* L.
- $B_\gamma = \{1, \gamma, \ldots, \gamma^{d-1}\}$ *is a basis for* $\mathbb{Q}(\gamma)$.
- A *is the change-of-basis matrix from* B_γ *to* B_L *obtained from Algorithm 4 over* $\mathbb{F} = \mathbb{Q}$.

Recall that (p, C) is a det-bad pair if either $\det(A) = 0$ or $p \mid \text{num}(\det(A))$ (see Definition 4.4.4). This section establishes a bound on the probability that a randomly chosen pair (p, C) is det-bad. That is,

$$\begin{aligned}\text{Prob}[(p,C) \text{ is a det-bad pair}] &= \text{Prob}[\det(A) = 0 \ \vee (\det(A) \neq 0 \wedge \ p \mid \text{num}(\det(A)))] \\ &\le \text{Prob}[\det(A) = 0] + \text{Prob}[p \mid \text{num}(\det(A))]. \end{aligned} \tag{6.13}$$

The definition of det-bad pairs is the same in both the MRESNF and MGCDNF algorithms, so the findings presented in this section apply to both algorithms. First, we bound $\text{Prob}[\det(A) = 0]$ in the following theorem.

Theorem 6.3.2. *With Notation 6.2,*

$$\text{Prob}[\det(A) = 0] \le \frac{d(d-1)}{2(p-1)}.$$

Proof. Let $\Gamma = z_1 + \sum_{i=1}^{n-1} w_i z_{i+1} \in \mathbb{Z}[w_1, \ldots, w_{n-1}][z_1, \ldots, z_n]$. Define B as the $d \times d$ matrix whose jth column is $[\Gamma^{j-1}]_{B_L}$ for $1 \le j \le d$ obtained from running LAminpoly over $\mathbb{Q}$. Hence, each entry $B_{i,j}$ is a polynomial in $\mathbb{Q}[w_1, \ldots, w_{n-1}]$, and therefore $\det(B) \in \mathbb{Q}[w_1, \ldots, w_{n-1}]$. Now, Substitute $w_i = C_i$ for $1 \le i \le n-1$. This substitution maps Γ to $\gamma = z_1 + \sum_{i=1}^{n-1} C_i z_{i+1}$, and correspondingly maps the matrix B to the matrix A obtained from LAminpoly for γ over $\mathbb{F} = \mathbb{Q}$. Since evaluation is a ring homomorphism, we have $\det(A) = \det(B)(C)$, so

$$\text{Prob}[\det(B)(C) = 0] = \text{Prob}[\det(A) = 0].$$

By Theorem 3.2.5, there exists $a = (a_1, \ldots, a_{n-1}) \in (0, p)^{n-1}$ for which $\Gamma(a) = z_1 + \sum_{i=1}^{n-1} a_i z_{i+1}$ is a primitive element of L, so $\det(B)(a) \neq 0$, which implies that $\det(B)$ is a non-zero polynomial. Thus, $\det(B)$ satisfies the Schwartz-Zippel lemma. Let $\deg(B)$ denote the total degree of B in variables $w_1, \ldots, w_{n-1}$. In the jth column of B, we have $\deg(B_{i,j}) \le j - 1$ for $1 \le i \le d$. Hence,

$$\deg(\det(B)) \le \sum_{j=1}^{d} (j-1) = \frac{d(d-1)}{2}.$$

Since $C_i \in (0, p)$ for $1 \le i \le n-1$, by the Schwartz–Zippel lemma, we obtain

$$\text{Prob}[\det(A) = 0] = \text{Prob}[\det(B)(C) = 0] \le \frac{d(d-1)}{2\underbrace{(p-1)}_{C_i \in (0,p)}}.$$

□

Remark 6.3. *We can make the bound obtained from Theorem 6.3.2 independent of p. Since $p \in \mathbb{P}_b$, we have $p \geq 2^{b-1}$. Thus, $p - 1 \geq 2^{b-1} - 1$ and*

$$\text{Prob}[\det(A) = 0] \leq \frac{d(d-1)}{2(p-1)} \leq \frac{d(d-1)}{2^b - 2}.$$

Example 6.3.2. *Consider Example 6.3.1, where $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - \frac{2}{5}, z_2^2 - \frac{7}{3}\rangle$ with degree $d = 4$. Let $\Gamma = z_1 + wz_2$, so*

$$\begin{aligned}
\Gamma^0 &= 1 \\
\Gamma^1 &= z_1 + wz_2 \\
\Gamma^2 &= (\frac{7}{3}w^2 + \frac{2}{5}) + (2w)\, z_1 z_2, \\
\Gamma^3 &= (7w^2 + \frac{2}{5})z_1 + (\frac{7}{3}w^3 + \frac{6}{5}w)z_2.
\end{aligned}$$

Let $B_L = \{1, z_2, z_1, z_1 z_2\}$ be a basis for L. Running Algorithm LAminpoly on Γ over $\mathbb{F} = \mathbb{Q}$ gives the change-of-basis matrix B (from Theorem 6.3.2):

$$B = \begin{bmatrix} 1 & 0 & \frac{7}{3}w^2 + \frac{2}{5} & 0 \\ 0 & w & 0 & \frac{7}{3}w^3 + \frac{6}{5}w \\ 0 & 1 & 0 & 7w^2 + \frac{2}{5} \\ 0 & 0 & 2w & 0 \end{bmatrix}.$$

For a randomly chosen $C_1 \in (0, p)$, where $p \in \mathbb{P}_{31}$, from Theorem 6.3.2, we have

$$\text{Prob}[\det(A) = 0] = \text{Prob}[\det(B)(C_1) = 0] \leq \frac{d(d-1)}{2(p-1)} \leq \frac{d(d-1)}{2^b - 2} = \frac{12}{2^{31} - 2} \approx 5.588 \times 10^{-9},$$

where A is the matrix obtained from LAminpoly algorithm for $\gamma = z_1 + C_1 z_2$ over $\mathbb{F} = \mathbb{Q}$. However,

$$\det(B) = -\frac{28}{3}w^4 + \frac{8}{5}w^2 \in \mathbb{Q}[w],$$

does not have a non-zero integer root. Consequently, for every $C_1 \in (0, p)$ (and $\gamma = z_1 + C_1 z_2$), we have $det(A) \neq 0$. Thus, in practice,

$$\text{Prob}[\det(A) = 0] = 0.$$

Notice that if we run the LAminpoly over $\mathbb{F} = \mathbb{Z}_{11}$, then

$$\det(B) = 9w^2(w + 5)(w + 6) \in \mathbb{Z}_{11}[w],$$

so $w = 0$, $w = 5$, and $w = 6$ are the roots of $\det(B)$ in $\mathbb{Z}_{11}$. Therefore, if $C = (C_1)$, the ordered pairs $(p, C) = (11, (5))$ and $(p, C) = (11, (6))$ are det-bad.

To bound Prob$[p \mid \text{num}(\det(A))]$, when $\det(A) \neq 0$ in (6.13), we first need to compute an upper bound on $|\text{num}(\det(A))|$.

Definition 6.3.1. *Let $f \in R[z_1, \ldots, z_n]$ and $\check{M}_1, \ldots, \check{M}_n$ be the minimal polynomials as defined in Section 3.3. Then*

$$r = f \text{ rem } \langle \check{M}_n, \ldots, \check{M}_1 \rangle$$

denote the recursive remainder of f divided by $\check{M}_n, \ldots, \check{M}_1$, s.t. $r = 0$ or $\deg(r, z_i) < \deg(M_i, z_i)$ for all $1 \leq i \leq n$. Likewise, let

$$\tilde{r} = f \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1 \rangle$$

denote the recursive pseudo-remainder of f with respect to $\check{M}_n$ through $\check{M}_1$ s.t. $\tilde{r} = 0$ or $\deg(\tilde{r}, z_i) < \deg(M_i, z_i)$ for all $1 \leq i \leq n$.

The following example reviews the construction of matrix A over $\mathbb{F} = \mathbb{Q}$ (also see Example 3.4.3 from Chapter 3).

Example 6.3.3. *Let $M_1(z_1) = z_1^2 - \frac{7}{2}$ and $M_2(z_2) = z_2^2 - \frac{11}{3}$. Let $B_L = \{1, z_2, z_1, z_1 z_2\}$ be a basis for $\mathbb{Q}[z_1, z_2]/\langle M_1(z_1), M_2(z_2) \rangle$ of dimension $d = 4$. To construct A, first we compute the semi-associates $\check{M}_1 = 2z_1^2 - 7$ and $\check{M}_2 = 3z_2^2 - 11$. Take $C_1 = 3$, so $\gamma = z_1 + 3z_2$. Then,*

$$\begin{aligned}
\gamma^0 &= 1 \\
\gamma^1 &= (z_1 + 3z_2)^1 \text{ rem } \langle \check{M}_1, \check{M}_2 \rangle = z_1 + 3z_2 \\
\gamma^2 &= (z_1 + 3z_2)^2 \text{ rem } \langle \check{M}_1, \check{M}_2 \rangle = 6z_1 z_2 + \frac{73}{2} \\
\gamma^3 &= (z_1 + 3z_2)^3 \text{ rem } \langle \check{M}_1, \check{M}_2 \rangle = \frac{205}{2} z_1 + \frac{261}{2} z_2,
\end{aligned}$$

and the change-of-basis matrix is

$$A = \begin{bmatrix} 1 & 0 & \frac{73}{2} & 0 \\ 0 & 3 & 0 & \frac{261}{2} \\ 0 & 1 & 0 & \frac{205}{2} \\ 0 & 0 & 6 & 0 \end{bmatrix}.$$

To bound the determinant of a matrix over $\mathbb{Z}$, we can employ Hadamard's bound (Theorem 6.2.1). However, as illustrated in Example 6.3.3, when computing over $\mathbb{Q}$, the matrix A is a $d \times d$ matrix over $\mathbb{Q}$, so we cannot use Hadamard's bound for A. However, by replacing long division with pseudo-division in Algorithm 4, we can obtain the matrix $\tilde{A} \in \mathbb{Z}^{d \times d}$ for which Hadamard's bound can be applied. We construct Matrix $\tilde{A}$ s.t. its jth column represents the coordinate vector $[\gamma^{j-1} \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1 \rangle]_{B_L}$.

Example 6.3.4. *Considering Example 6.3.3, by employing pseudo-division, we obtain*

$$\begin{aligned}
\gamma^0 &= 1 \\
\gamma^1 &= (z_1 + 3z_2)^1 \text{ prem } \langle \check{M}_1(z_1), \check{M}_2(z_2) \rangle = z_1 + 3z_2 \\
\gamma^2 &= (z_1 + 3z_2)^2 \text{ prem } \langle \check{M}_1(z_1), \check{M}_2(z_2) \rangle = 36z_1 z_2 + 219 \\
\gamma^3 &= (z_1 + 3z_2)^3 \text{ prem } \langle \check{M}_1(z_1), \check{M}_2(z_2) \rangle = 3690z_1 + 4698z_2.
\end{aligned}$$

Since $B_L = \{1, z_2, z_1, z_1z_2\}$, *the change-of-basis matrix is*

$$\tilde{A} = \begin{bmatrix} 1 & 0 & 219 & 0 \\ 0 & 3 & 0 & 4698 \\ 0 & 1 & 0 & 3690 \\ 0 & 0 & 36 & 0 \end{bmatrix}$$

While pseudo-division enables the construction of an integer matrix $\tilde{A} \in \mathbb{Z}^{d\times d}$, natural division is significantly faster in practice. Consequently, Algorithm 4 employs natural division. As we will show in Corollary 6.2, the determinants of A and $\tilde{A}$ satisfy

$$\underbrace{\det(\tilde{A})}_{\in\mathbb{Z}} = \Big(\prod_{i=0}^{n-1} \underbrace{\mathrm{lc}(\check{M}_{n-i})^{\Delta_i}}_{\in\mathbb{Z}}\Big) \underbrace{\det(A)}_{\in\mathbb{Q}} \text{ for certain } \Delta_i \in \mathbb{Z}_{\geq 0}. \tag{6.14}$$

For $0 \leq i \leq n-1$, we know that $\mathrm{lc}(\check{M}_{n-i}) \in \mathbb{Z}$ and $\Delta_i \geq 0$. Therefore, $\prod_{i=0}^{n-1} \mathrm{lc}(\check{M}_{n-i})^{\Delta_i} \in \mathbb{Z}$. Since $\det(\tilde{A}) \in \mathbb{Z}$, Equation (6.14) shows that $\mathrm{den}(\det(A))$ must divide $\prod_{i=0}^{n-1} \mathrm{lc}(\check{M}_{n-i})^{\Delta_i}$, while $\mathrm{num}(\det(A))$ must divide $\det(\tilde{A})$. By the definition of det-bad pairs (Definition 6.3.3), we choose the prime p so that $p \nmid \prod_{i=0}^{n-1} \mathrm{lc}(\check{M}_{n-i})$. Therefore, $p \mid \det(\tilde{A})$ if and only if $p \mid \mathrm{num}(\det(A))$. Thus,

$$\mathrm{Prob}[p \mid \mathrm{num}(\det(A))] = \mathrm{Prob}[p \mid \det(\tilde{A})].$$

Moreover, since $\mathrm{num}(\det(A)) \mid \det(\tilde{A})$, any upper bound on $|\mathrm{num}(\det(A))|$ can be obtained by bounding $|\det(\tilde{A})|$ instead.

Example 6.3.5. *From Examples 6.3.3 and 6.3.4, we have* $|\det(A)| = 1062$ *and* $|\det(\tilde{A})| = 229392$. *Clearly,* $\mathrm{num}(\det(A)) = \det(A)$ *is a divisor of* $\det(\tilde{A})$. *In fact,*

$$\frac{|\det(\tilde{A})|}{|\mathrm{num}(\det(A))|} = 216 \in \mathbb{Z}.$$

To bound $|\det(\tilde{A})|$, we need to bound the entries of $\tilde{A}$. Among the vectors $[\gamma^j]_{B_L}$, the largest entries appear in $[\gamma^{d-1}]_{B_L}$. Moreover, we have $\|\gamma^{d-1} \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1\rangle\|_\infty < \|\gamma^d \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1\rangle\|_\infty$. Thus, bounding the quantity $\|\gamma^d \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1\rangle\|_\infty$ yields a valid upper bound for the entries $\tilde{A}_{i,j}$. Note that $\deg(\gamma^j, z_i) = j$ for $1 \leq i \leq n$.

Notation 6.3. *To avoid repeating, the following terminology will remain unchanged and will be used throughout this section.*

(i) $d_i = \deg(\check{M}_i, z_i)$ *and* $d = \prod_{i=1}^{n} d_i \neq 0$.

(ii) $D_i = \frac{d}{\prod_{j=1}^{i} d_{n-j+1}} = \prod_{j=1}^{n-i} d_j$.

(iii) $\tilde{A}$ *is a* $d \times d$ *matrix in* $\mathbb{Z}$ *whose* j*th column is the coordinate vector* $[\gamma^{j-1} \text{ prem } \langle \check{M}_n, \ldots, \check{M}_1\rangle]_{B_L}$.

(iv) A *is the* $d \times d$ *matrix obtained from Algorithm 4 over* $\mathbb{F} = \mathbb{Q}$, *whose* j*th column is the coordinate vector* $[\gamma^{j-1} \text{ rem } \langle \check{M}_n, \ldots, \check{M}_1\rangle]_{B_L}$.

We present Lemma 6.3.3 and 6.3.4 without proof.

Lemma 6.3.3. *Let $f, g \in \mathbb{Z}[z_1, \ldots, z_n]$. Then*

$$\|fg\|_\infty \leq \|f\|_\infty \|g\|_\infty \min(\#f, \#g).$$

Lemma 6.3.4. *Let $\check{M}_i = l_i z_i^{d_i} + \sum_{j=1}^{d_i-1} a_{i,j} z_i^j$ be the minimal polynomial of α_i, where $a_{i,j} \in \mathbb{Z}[z_1, \ldots, z_{i-1}]$. We have $\deg(a_{i,j}, z_k) \leq d_k - 1$ for $1 \leq k \leq i-1$. Thus,*

$$\#a_{i,j} \leq \prod_{k=1}^{i-1} d_k = \frac{d}{d_i d_{i+1} \cdots d_n} < d.$$

In Theorem 6.3.5, we bound $\|\mathrm{prem}(\gamma^d, \check{M}_n, z_n)\|_\infty$, the pseudo-remainder of γ^d divided by the polynomial $\check{M}_n$ w.r.t. z_n.

Theorem 6.3.5. *Let $f = \gamma^d$ and $\tilde{r} = \mathrm{prem}(f, \check{M}_n, z_n)$, where $\check{M}_n = l_n z_n^{d_n} + \sum_{j=0}^{d_n-1} a_j z_n^j$ with $l_n \in \mathbb{Z}$ and $a_j \in \mathbb{Z}[z_1, \ldots, z_{n-1}]$ for $0 \leq j \leq d_n - 1$. Let $\delta = d - d_n + 1$ denote the maximum number of division steps. Then,*

(i) Either $\tilde{r} = 0$ or $\deg(\tilde{r}, z_n) \leq d_n - 1$ and $\deg(\tilde{r}, z_i) \leq d + \delta(d_i - 1)$, for $1 \leq i \leq n-1$.

(ii) $\|\tilde{r}\|_\infty \leq \|f\|_\infty (l_n + d/d_n \|\check{M}_n\|_\infty)^\delta$.

Proof. Since $f = \gamma^d$, we have $\deg(f, z_i) = d$ for $1 \leq i \leq n$. Let $f = \sum_{i=0}^{d} f_i z_n^i$ s.t. $f_i \in \mathbb{Z}[z_1, \ldots, z_{n-1}]$ for $1 \leq i \leq d$.

(i) $\mathrm{pquo}(f, \check{M}_n, z_n)$ has a degree of $d - d_n$, so the pseudo-division of f by $\check{M}_n$ has up to $\delta = d - d_n + 1$ steps. In the first step of the pseudo division, we have $\tilde{r}_1 = l_n f - f_d z_n^{d-d_n} \check{M}_n$. If $\tilde{r}_1 = 0$, then $\tilde{r} = 0$ and we are done. Otherwise, $\deg(\tilde{r}_1, z_n) \leq d - 1$. Moreover, for $1 \leq i \leq n-1$, we have $\deg(f_d, z_i) \leq \deg(f, z_i) = d$ and $\deg(\check{M}_n, z_i) \leq d_i - 1$. Consequently,

$$\begin{aligned} \deg(\tilde{r}_1, z_i) &= \max\{\deg(f, z_i), \deg(f_d, z_i) + \deg(\check{M}_n, z_i)\} \\ &\leq \deg(f, z_i) + \deg(\check{M}_n, z_i) \leq d + 1(d_i - 1). \end{aligned}$$

If $\deg(\tilde{r}_1, z_n) \geq d_n$, we continue the division. Define $b_1 = \mathrm{lc}(\tilde{r}_1, z_n)$ and set $\deg(\tilde{r}_1, z_n) = d - 1$. During the second step of the division, we obtain $\tilde{r}_2 = l_n \tilde{r}_1 - b_1 z_n^{d-d_n-1} \check{M}_n$. If $\tilde{r}_2 = 0$, then $\tilde{r} = 0$ and we are done. Otherwise, we have

$$\begin{aligned} &\deg(\tilde{r}_2, z_n) \leq \deg(\tilde{r}_1, z_n) - 1 \leq d - 2 \quad \text{and} \\ &\deg(\tilde{r}_2, z_i) \leq \deg(\tilde{r}_1, z_i) + \deg(\check{M}_n, z_i) \leq d + 2(d_i - 1) \quad \text{for} \quad 1 \leq i \leq n-1. \end{aligned}$$

Since the division algorithm has at most δ steps, in the last step, we have

$$\begin{aligned} &\deg(\tilde{r}, z_n) \leq d - \delta = d_n - 1 \quad \text{and} \\ &\deg(\tilde{r}, z_i) \leq d + \delta(d_i - 1) \quad \text{for} \quad 1 \leq i \leq n-1. \end{aligned}$$

(ii) In the first step of pseudo-division of f by $\check{M}_n$, we obtain $\tilde{r}_1 = l_n f - f_d z^{d-d_n} \check{M}_n$. Thus,

$$\|\tilde{r}_1\|_\infty \leq \|l_n f\|_\infty + \|f_d \check{M}_n\|_\infty.$$

To compute an upper bound for $\|f_d\check{M}_n\|_\infty$, it is sufficient to compute an upper bound for $\|f_d a_j\|_\infty$ where $a_j \in \mathbb{Z}[z_1,\ldots,z_{n-1}]$. Using Lemma 6.3.4, we have $\#a_j < d/d_n$ which implies that

$$\|f_d a_j\|_\infty \leq \|f_d\|_\infty \|\check{M}_n\|_\infty \min(\#a_j, \#f_d) \leq d/d_n \|f_d\|_\infty \|\check{M}_n\|_\infty$$

for $1 \leq j \leq d_n - 1$. Moreover, since $\|f_d\|_\infty \leq \|f\|_\infty$, we obtain

$$\|\tilde{r}_1\|_\infty \leq l_n\|f\|_\infty + \|f_d\check{M}_n\|_\infty \leq \|f\|_\infty(l_n + d/d_n\|\check{M}_n\|_\infty).$$

Furthermore, $\deg(\tilde{r}_1, z_n) \leq d-1$. If $\deg(\tilde{r}_1, z_n) \geq d_n$, we continue the division. In the second division step, we have $\tilde{r}_2 = l_n\tilde{r}_1 - b_1 z_n^{d-d_n-1}\check{M}_n$, where $b_1 = \mathrm{lc}(\tilde{r}_1, z_n)$. Since $\|b_1\|_\infty \leq \|\tilde{r}_1\|_\infty$, using the same strategy as the first division step, we have

$$\begin{aligned}\|\tilde{r}_2\|_\infty &\leq l_n\|\tilde{r}_1\|_\infty + \|b_1\check{M}_n\|_\infty \leq l_n\|\tilde{r}_1\|_\infty + d/d_n\|\tilde{r}_1\|_\infty\|\check{M}_n\|_\infty \\ &\leq \|\tilde{r}_1\|_\infty(l_n + d/d_n\|\check{M}_n\|_\infty) \leq \|f\|_\infty(l_n + d/d_n\|\check{M}_n\|_\infty)^2.\end{aligned}$$

Continuing this argument, the result is obtained.

□

In Theorem 6.3.5, we bounded $\|\mathrm{prem}(\gamma^d, \check{M}_i, z_i)\|_\infty$. In the following theorem, we apply Theorem 6.3.5 to bound $\|\gamma^d \text{ prem } \langle\check{M}_n,\ldots,\check{M}_1\rangle\|_\infty$.

Theorem 6.3.6. *Let $f = \gamma^d \in \mathbb{Z}[z_1,\ldots,z_n]$ and $\check{M}_i = l_i z_i^{d_i} + \sum_{j=0}^{d_i-1} b_{i,j} z_i^j$ s.t. $l_i \in \mathbb{Z}$ and $b_{i,j} \in \mathbb{Z}[z_1,\ldots,z_{i-1}]$ for $1 \leq i \leq n$ and $0 \leq j \leq d_i - 1$. Let $d = \prod_{i=1}^n d_i$. Let $\tilde{r} = f$ prem $\langle\check{M}_n,\ldots,\check{M}_1\rangle$. Then*

$$\|\tilde{r}\|_\infty \leq \|f\|_\infty \prod_{i=1}^{n}(l_{n-i+1} + D_i\|\check{M}_{n-i+1}\|_\infty)^{\delta_i},$$

where $\delta_1 = d - d_n + 1$, and $\delta_i = d - d_{n-i+1} + 1 + (d_{n-i+1} - 1)\sum_{j=1}^{i-1}\delta_j$ for $2 \leq i \leq n$.

Proof. Since $f = \gamma^d$, we have $\deg(f, z_i) \leq d$ for $1 \leq i \leq n$. Let $\tilde{r}_1 = \mathrm{prem}(f, \check{M}_n, z_n)$ and $\delta_1 = d - d_n + 1$ be the maximum number of division steps. From Theorem 6.3.5, we have

$$\|\tilde{r}_1\|_\infty \leq \|f\|_\infty(l_n + D_1\|\check{M}_n\|_\infty)^{\delta_1}. \tag{6.15}$$

Let $\tilde{r}_2 = \mathrm{prem}(\tilde{r}_1, \check{M}_{n-1}, z_{n-1})$. From Theorem 6.3.5 part (i), we have $\deg(\tilde{r}_1, z_{n-1}) \leq d + \delta_1(d_{n-1} - 1)$ so

$$\deg(\tilde{r}_1, z_{n-1}) - d_{n-1} + 1 \leq d + \delta_1(d_{n-1} - 1) - d_{n-1} + 1.$$

Thus, $\delta_2 = d - d_{n-1} + 1 + \delta_1(d_{n-1} - 1)$ is the maximum number of division steps. Let $\check{M}_{n-1} = l_{n-1}z_{n-1}^{d_{n-1}} + \sum_{j=0}^{d_{n-1}-1} b_{n-1_j} z_{n-1}^j$ s.t. $l_{n-1} \in \mathbb{Z}$ and $b_{n-1,j} \in \mathbb{Z}[z_1,\ldots,z_{n-2}]$ for $0 \leq j \leq d_{n-1} - 1$. Hence, applying Theorem 6.3.4, we have $\#b_{n-1,j} \leq D_2 = \frac{d}{d_n d_{n-1}}$. Using the same strategy as the proof of part (ii) of Theorem 6.3.5, we have

$$\begin{aligned}\|\tilde{r}_2\|_\infty &\leq \|\tilde{r}_1\|_\infty(l_{n-1} + D_2\|\check{M}_{n-1}\|_\infty)^{\delta_2} \\ &\leq \underbrace{\|f\|_\infty(l_n + D_1\|\check{M}_n\|_\infty)^{\delta_1}}_{\text{According to 6.15}}(l_{n-1} + D_2\|\check{M}_{n-1}\|_\infty)^{\delta_2}\end{aligned}$$

The result is obtained by repeating this process for polynomials $\check{M}_{n-2}, \ldots, \check{M}_1$.

□

Using Theorem 6.3.6, we are well-equipped to compute an upper bound for the integer entries of $\tilde{A}$, namely on $\mid \tilde{A}_{i,j} \mid$.

Corollary 6.1. *Let $\tilde{r} = \gamma^d$ prem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$, $l_i = \mathrm{lc}(\check{M}_i, z_i)$, and δ_i be as defined in Theorem 6.3.6. If $\|\gamma^d\|_\infty \leq 2^C$, then*

$$\mid \tilde{A}_{i,j} \mid \leq \|\tilde{r}\|_\infty \leq 2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i \|\check{M}_{n-i+1}\|_\infty)^{\delta_i}.$$

Proof. This follows directly from Theorem 6.3.6. □

We determined that $\delta_n \leq d^2/d_n$ through computational experiments, but we can only prove this for $d_1 = d_2 = \cdots = d_n$. Thus, Corollary 6.1 implies that $\log\|r\|_\infty$ is polynomial in d, C and $\|\check{M}_i\|_\infty$.

Suppose that Algorithm MGCDNF (and MRESNF) chooses $p \in \mathbb{P}_b$ randomly. The following theorem bounds the probability that $p | \det(\tilde{A})$.

Theorem 6.3.7. *Let $p \in \mathbb{P}_b$ be chosen randomly, $\|\gamma^d\|_\infty \leq 2^C$, $\|\check{M}_i\|_\infty \leq 2^m$, and $l_i = \mathrm{lc}(\check{M}_i, z_i)$. Let δ_i be as defined in Theorem 6.3.6. Then*

(i) $\mid \det(\tilde{A}) \mid \leq d^{d/2} \left(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i m)^{\delta_i}\right)^d$, *and*

(ii) $\mathrm{Prob}[p \mid \det(\tilde{A})] \leq \frac{\lfloor d/2 \log_2 d + d\left(C + \sum_{i=1}^{n} \delta_i \log_2 (l_{n-i+1} + D_i m)\right) \rfloor}{(b-1)\mid \mathbb{P}_b \mid}$.

Proof. (i) From Hadamard's bound (Theorem 6.2.1) and Corollary 6.1, we obtain

$$\begin{aligned}
\mid \det(\tilde{A}) \mid &\leq \underbrace{\prod_{j=1}^{d} \sqrt{\sum_{i=1}^{d} \tilde{A}_{i,j}^2}}_{\text{Theorem (6.2.1)}} \\
&\leq \prod_{j=1}^{d} \sqrt{d \underbrace{(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i \|\check{M}_{n-i+1}\|_\infty)^{\delta_i})^2}_{\text{Corollary (6.1)}}} \\
&\leq d^{d/2} \left(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i m)^{\delta_i}\right)^d.
\end{aligned}$$

(ii) Since $p \in \mathbb{P}_b$, we have $p \in (2^{b-1}, 2^b)$, which implies that $\log_2 p > \log_2 2^{b-1} = b - 1$. Thus,

$$\begin{aligned}
\mathrm{Prob}[p \mid \det(\tilde{A})] &\leq \frac{\lfloor \log_{2^{b-1}} \mid \det(\tilde{A}) \mid \rfloor}{\mid \mathbb{P}_b \mid} = \frac{\lfloor \frac{\log_2 (\mid \det(\tilde{A}) \mid)}{b-1} \rfloor}{\mid \mathbb{P}_b \mid} \\
&\leq \frac{\lfloor d/2 \log_2 d + dC + d \sum_{i=1}^{n} \delta_i \log_2 (l_{n-i+1} + D_i m) \rfloor}{(b-1) \mid \mathbb{P}_b \mid}.
\end{aligned}$$

□

Now, in Theorem 6.3.8, we present the connection between $\tilde{r} = \gamma^d$ prem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$ and $r = \gamma^d$ rem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$.

Theorem 6.3.8. *Let* $\tilde{r} = \gamma^d$ prem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$ *and* $r = \gamma^d$ rem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$. *Let* $r_0 = \tilde{r}_0 = \gamma^d$, *and for* $1 \leq i \leq n$. *Define* $r_i = \text{rem}(r_{i-1}, \check{M}_{n-i+1}, z_{n-i+1})$ *and* $\tilde{r}_i = \text{prem}(\tilde{r}_{i-1}, \check{M}_{n-i+1}, z_{n-i+1})$. *Let* $d_{r_i} = \deg(r_i, z_{n-i}) = \deg(\tilde{r}_i, z_{n-i})$ *for* $0 \leq i \leq n$. *Then*

$$\tilde{r} = r \prod_{i=0}^{n} \text{lc}(\check{M}_{n-i})^{\Delta_i},$$

where $\Delta_i = d_{r_i} - d_{n-i} + 1$ *is the number of division steps.*

Proof. We construct $\tilde{r} = \gamma^d$ prem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$ and $r = \gamma^d$ rem $\langle \check{M}_n, \ldots, \check{M}_1 \rangle$ step by step in parallel. The first pseudo-division and natural division are

$$\begin{aligned} \tilde{r}_1 &= \text{prem}(\tilde{r}_0, \check{M}_n, z_n) \quad \text{and} \\ r_1 &= \text{rem } (r_0, \check{M}_n, z_n), \end{aligned}$$

respectively. By Lemma 2.2.7 part (ii), $\tilde{r}_1 = \text{lc}(\check{M}_n)^{\Delta_0} r_1$.
In the second pseudo-division and natural division, we have

$$\tilde{r}_2 = \text{prem}(\tilde{r}_1, \check{M}_{n-1}) = \text{lc}(\check{M}_{n-1})^{\Delta_1} \tilde{r}_1 - \check{M}_{n-1} \tilde{q}_2 \tag{6.16}$$

$$r_2 = \text{rem } (r_1, \check{M}_{n-1}) = r_1 - \check{M}_{n-1} q_2. \tag{6.17}$$

Multiplying Equation 6.17 by $\text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0}$ we have

$$\text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0} r_2 = \text{lc}(\check{M}_{n-1})^{\Delta_1} \underbrace{\text{lc}(\check{M}_n)^{\Delta_0} r_1}_{\tilde{r}_1} - \text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0} \check{M}_{n-1} q_2$$

$$= \text{lc}(\check{M}_{n-1})^{\Delta_1} \tilde{r}_1 - \text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0} \check{M}_{n-1} q_2.$$

Subtracting Equation 6.16 from above, we have

$$\begin{aligned} \text{lc}(\check{M}_n)^{\Delta_0} \text{lc}(\check{M}_{n-1})^{\Delta_1} r_2 - \tilde{r}_2 &= -\text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0} \check{M}_{n-1} q_2 + \check{M}_{n-1} \tilde{q}_2 \\ &= \check{M}_{n-1}(-\text{lc}(\check{M}_n)^{\Delta_0} \text{lc}(\check{M}_{n-1})^{\Delta_1} q_2 + \tilde{q}_2). \end{aligned}$$

Since $\deg(r_2), \deg(\tilde{r}_2) < \deg(\check{M}_{n-1})$, and $\check{M}_{n-1} \neq 0$, we must have

$$-\text{lc}(\check{M}_n)^{\Delta_0} \text{lc}(\check{M}_{n-1})^{\Delta_1} q_2 + \tilde{q}_2 = 0,$$

which implies that $\tilde{q}_2 = \text{lc}(\check{M}_n)^{\Delta_0} \text{lc}(\check{M}_{n-1})^{\Delta_1} q_2$ and $\tilde{r}_2 = \text{lc}(\check{M}_{n-1})^{\Delta_1} \text{lc}(\check{M}_n)^{\Delta_0} r_2$. Continuing this argument, in the last division, we have $\tilde{r}_n = \prod_{i=0}^{n} \text{lc}(\check{M}_{n-i})^{\Delta_i} r_n$. By construction, we have $\tilde{r} = \tilde{r}_n$ and $r = r_n$.

□

Corollary 6.2. $\det(\tilde{A}) = \prod_{i=0}^{n} \text{lc}(\check{M}_{n-i})^{\Delta_i} \det(A)$ *where* Δ_i *is defined in Theorem 6.3.8.*

Proof. This is an immediate consequence of Theorem 6.3.8. □

Theorem 6.3.9. *Let C, δ_i, d, m, and l_i be as defined in Theorem 6.3.7. Let p be chosen randomly from $\mathbb{P}_b$ s.t. $p \nmid \prod_{i=0}^{n} \mathrm{lc}(\check{M}_{n-i})$ for $1 \leq i \leq n$. Suppose that $\det(A) \neq 0$. Then*

$$\mathrm{Prob}[p \mid \mathrm{num}(\det(A))] = \mathrm{Prob}[p \mid \det(\tilde{A})] \leq \frac{\lfloor d/2 \log_2 d + d(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m)) \rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}.$$

Proof. From Corollary 6.2,

$$\det(\tilde{A}) = \prod_{i=0}^{n} \mathrm{lc}(\check{M}_{n-i})^{\Delta_i} \det(A),$$

where $\Delta_i \in \mathbb{Z}$ is defined in Theorem 6.3.8. Since $p \nmid \prod_{i=0}^{n} \mathrm{lc}(\check{M}_{n-i})$ for $1 \leq i \leq n$, we have $p \mid \det(\tilde{A})$ if and only if $p \mid \mathrm{num}(\det(A))$. Therefore,

$$\mathrm{Prob}[p \mid \mathrm{num}(\det(A))] = \mathrm{Prob}[p \mid \det(\tilde{A})],$$

which implies that

$$\mathrm{Prob}[p \mid \mathrm{num}(\det(A))] = \mathrm{Prob}[p \mid \det(\tilde{A})] \leq \frac{\lfloor d/2 \log_2 d + d\,(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m)) \rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}.$$

□

Theorem 6.3.10. *Let C, δ_i, d, m, and l_i be as defined in Theorem 6.3.7. Let p be chosen randomly from $\mathbb{P}_b$ s.t. $p \nmid \prod_{i=0}^{n} \mathrm{lc}(\check{M}_{n-i})$ for $1 \leq i \leq n$. Then*

$$\mathrm{Prob}[(p, C) \ \textit{ is a det-bad pair}] \leq \frac{d(d-1)}{2(p-1)} + \frac{\lfloor d/2 \log_2 d + d\,(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m)) \rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}$$

Proof. Employing the results from Theorems 6.3.2 and 6.3.9, we have

$$\begin{aligned}
&\mathrm{Prob}[(p, C) \ \text{ is a det-bad pair}] = \mathrm{Prob}[\det(A) = 0 \ \vee (\det(A) \neq 0 \wedge \ p \mid \mathrm{num}(\det(A)))] \\
&\leq \mathrm{Prob}[\det(A) = 0] + \mathrm{Prob}[p \mid \mathrm{num}(\det(A))] \\
&\leq \underbrace{\frac{d(d-1)}{2(p-1)}}_{\text{Theorem 6.3.2}} + \underbrace{\frac{\lfloor d/2 \log_2 d + d\,(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m)) \rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}}_{\text{Theorem 6.3.9}}.
\end{aligned}$$

□

Now we can also obtain a bound for $\|M(z)\|_\infty$, where $M(z)$ is the generator polynomial obtained from Algorithm 4.

Theorem 6.3.11. *Let $M(z)$ be the generator polynomial obtained from Algorithm 4. Define*

$$B_M = d^{d/2} \left(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i \|\check{M}_{n-i+1}\|_\infty)^{\delta_i} \right)^d,$$

where C and l_i are from Theorem 6.3.7. Then

$$\|\check{M}(z)\|_\infty \leq B_M.$$

Proof. To construct $\check{M}(z)$, we solve the linear system $A \cdot q = -[\gamma^d]_{B_L}$ for $q \in \mathbb{Q}^d$. By Cramer's rule, $q_k = \frac{\det(A^{(k)})}{\det(A)}$, where $A^{(k)}$ is the matrix formed by replacing the kth column of A with $[\gamma^d \text{ rem } \langle \check{M}_n, \ldots, \check{M}_1 \rangle]_{B_L}$ for $1 \leq k \leq d$. Thus, the largest entries of $A^{(k)}$ appear in the kth column. Applying the same justification as in Theorem 6.3.9 for the matrices $A^{(k)}$ and $\tilde{A}^{(k)}$, by Theorem 6.2.1, we obtain

$$| \det(A^{(k)}) | \leq \prod_{i=1}^{d} \sqrt{\sum_{j=1}^{d} \tilde{A}_{j,i}^{(k)2}} \leq d^{d/2} \underbrace{(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i \| \check{M}_{n-i+1} \|_\infty)^{\delta_i})^d}_{\text{Corollary 6.1}}.$$

Since $\check{M}_i \in \mathbb{Z}[z_1, \ldots, z_i]$ for $1 \leq i \leq n$ and $\check{M}(z) \in \mathbb{Z}[z]$, we have $q_k = \frac{\det(A^{(k)})}{\det(A)} \in \mathbb{Z}$, so $\det(A)$ must divide $\det(A^{(k)})$. Thus, $q_k \leq | \det(A^{(k)}) | \leq d^{d/2}(2^C \prod_{i=1}^{n} (l_{n-i+1} + D_i \| \check{M}_{n-i+1} \|_\infty)^{\delta_i})^d$, which completes the proof. □

Example 6.3.6. *Consider Example 6.3.2. Let $C = (c_1) = (3)$, and $\gamma = z_1 + c_1 z_2 = z_1 + 3z_2$ be a potential primitive element of $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - \frac{2}{5}, z_2^2 - \frac{7}{3} \rangle$. Running the LAminpoly over $\mathbb{F} = \mathbb{Q}$, we obtain the change-of-basis matrix from $B_\gamma = \{1, \gamma, \gamma^2, \gamma^3\}$ to $B_L = \{1, z, z^2, z^3\}$ as*

$$A = \begin{bmatrix} 1 & 0 & \frac{107}{5} & 0 \\ 0 & 3 & 0 & \frac{333}{5} \\ 0 & 1 & 0 & \frac{317}{5} \\ 0 & 0 & 6 & 0 \end{bmatrix},$$

with $\det(A) = \frac{-3708}{5}$. Since $| \text{num}(\det(A)) | = 3708 = 2^2 \times 3^2 \times 103$, the ordered pairs $(p, C) = (2, (3))$, $(3, (3))$, and $(103, (3))$ are det-bad. Let $p \in \mathbb{P}_{31}$ be chosen randomly. In this example, we have

- *$\check{M}_1 = 5z_1^2 - 2$ and $\check{M}_2 = 3z_2^2 - 7$,*
- *$d = 4$, $d_1 = \deg(\check{M}_1, z_1) = 2$, and $d_2 = \deg(\check{M}_2, z_2) = 2$,*
- *$D_1 = \frac{d}{d_2} = d_1$ and $D_2 = \frac{d}{d_1 d_2} = 1$,*
- *$\| \gamma^d \|_\infty = 108 \leq 2^7$, so $C = 7$,*
- *$l_1 = \text{lc}(\check{M}_1, z_1) = 5$ and $l_2 = \text{lc}(\check{M}_2, z_2) = 3$.*

From Theorem 6.3.6,

$$\delta_1 = d - d_2 + 1 = 3, \text{ and}$$
$$\delta_2 = d - d_1 + 1 + (d_1 - 1) \cdot \delta_1 = 6.$$

Therefore,

$$\begin{aligned} \text{Prob}[p \mid \text{num}(\det(A))] &\leq \frac{\lfloor d/2 \log_2 d + d \left(C + \sum_{i=1}^{n} \delta_i \log_2 (l_{n-i+1} + D_i \| \check{M}_{n-i+1} \|_\infty) \right) \rfloor}{(b-1) \times | \mathbb{P}_b |} \\ &= \frac{121}{30 \times | \mathbb{P}_{31} |} \approx 7.956 \times 10^{-8}. \end{aligned} \tag{6.18}$$

From Equation (6.18) and Example 6.3.2, we obtain

$$\begin{aligned}\text{Prob}[(p,C) \textit{ is det-bad}] &\leq \text{Prob}[\det(A)=0] + \text{Prob}[p \mid \text{num}(\det(A))] \\ &\leq \underbrace{\frac{12}{2^{31}-2}}_{\textit{Example 6.3.2}} + \frac{121}{30 \times |\mathbb{P}_{31}|} \approx 8.514 \times 10^{-8}.\end{aligned}$$

However, the actual size $|\text{num}(\det(A))| = 3708$ *does not have a* 31*-bit prime divisor. Therefore, if* $p \in \mathbb{P}_{31}$*, then* $\text{Prob}[p \mid \text{num}(\det(A))] = 0$*. Moreover, from Example 6.3.2, since* $\det(B)$ *does not have a non-zero root in* $\mathbb{Z}$*,* $\text{Prob}[\det(A)=0] = 0$*. Hence, for this example*

$$\text{Prob}[(p,C) \textit{ is det-bad}] = 0.$$

6.3.3 Zero divisor pairs

MGCDNF employs Algorithm PGCDNF (Algorithm 7) to compute the monic GCDs over $\bar{L}_p$. Moreover, MRESNF uses PRESNF (Algorithm 10) to compute the resultants over $\bar{L}_p$. Since $\bar{L}_p$ is generally not a field, PGCDNF and PRESNF may encounter zero divisors while computing the monic GCDs and resultants. In PGCDNF, this can happen in the GCD computations carried out in Lines 2, 5, 6, 7, 9, and 30. Therefore,

$$\begin{aligned}&\text{Prob}[\text{PGCDNF encounters a zero divisor pair}] = \\ &\text{Prob}[\text{A zero divisor pair appears in either Lines 2, 5, 6, 7, 9, or 30 of PGCDNF}].\end{aligned}$$

In each of these lines, PGCDNF applies the MEA to compute the monic GCD. Therefore, estimating the probability that the MEA fails over $\bar{L}_p[x_1]$ in Line 2, we can use this bound to estimate the failure probabilities of the remaining GCD computations in the other lines of PGCDNF. Let $f_1, f_2 \in L[x_1,\ldots,x_k]$ and let p be a prime, and $\beta \in [0,p)^{(k-1)}$ be an evaluation point s.t. (p,β) is not lc-bad. Let M be the generator polynomial obtained from LAminpoly over $\mathbb{F} = \mathbb{Q}$. The MEA fails if the leading coefficient of one of the intermediate remainders is not invertible over $\bar{L}_p = \mathbb{Z}_p[z]/\langle \phi_p(\check{M})\rangle$. In other words,

$$\text{Prob}[\text{MEA fails over } \bar{L}_p[x_1]] = \text{Prob}[\gcd(\text{lc}(r_i), \phi_p(\check{M})) \neq 1, \text{ for some } i \geq 2], \tag{6.19}$$

where $r_1,\ldots,r_l$ is the m.p.r.s. generated by $\phi_p(\check{f}_1)(\beta)$ and $\phi_p(\check{f}_2)(\beta) \in \bar{L}_p[x_1]$. To bound (6.19), at some point, we need to bound $\|\text{lc}(r_i)\|_\infty$. To do so, we use subresultants as defined in the following section.

Subresultant

We are interested in a Polynomial Remainder Sequence (p.r.s.) that avoids the appearance of fractions in the remainders. The Subresultant Polynomial Remainder Sequence (s.p.r.s.) is such a sequence. We present two ways of defining the subresultants. The first definition of subresultants (Definition 6.3.2) is based on pseudo-division for univariate polynomials. The second one (Definition 6.3.3) defines subresultants using determinants.

Definition 6.3.2. *Let $f_1, f_2 \in R[x]$ be two non-zero polynomials. The Polynomial Remainder Sequence $S_1, S_2, S_3, \ldots, S_l$ obtained from Algorithm 12 is called the Subresultant Polynomial Remainder Sequence (s.p.r.s.). Note that the last subresultant $S_l = \mathrm{res}(f_1, f_2, x)$.*

Theorem 6.3.12 presents a connection between the m.p.r.s. obtained from Algorithm 2 and the s.p.r.s. obtained from Algorithm 12 (see [8, 4.1 Subresultant Chain Algorithm]).

Algorithm 12: s.p.r.s. Algorithm ([8, 4.1 Subresultant Chain Algorithm])

Input: $f_1, f_2 \in R[x]$, non-zero polynomials s.t. $\deg(f_1) \geq \deg(f_2) \geq 0$
Output: Either$S = S_1, S_2, \ldots, S_l \in R[x]$, the s.p.r.s. generated by f_1, f_2, or FAIL

$m, n = \deg(f_1), \deg(f_2)$
$S_1, S_2 = f_1, f_2$
if $\deg(f_1) = 0$ *and* $\deg(f_2) = 0$ **then**
    return$(S_1, S_2, 1)$
if $\deg(f_1) \neq 0$ *and* $\deg(f_2) = 0$ **then**
    return(S_1, S_2, f_2^m)
$c, r = 1, n$
$j = m - 1$
$S = S_1, S_2$
$i = 2$ `// i counts the number of subresultants.`
while $r \neq 0$ **do**
    $r = \deg(S_i)$
    if $r = 0$ **then**
        return(S)
    if *c is not invertible in R* **then**
        return*(FAIL)*
    $S_{i+1} = \mathrm{prem}(S_{i-1}, S_i, x)/(-c)^{j-r+2}$
    if $j \neq r$ **then**
        $S_i = (\mathrm{lc}(S_i)^{j-r} S_i)/c^{j-r}$
    $S = S, S_{i+1}$
    $j = r - 1$
    $c = \mathrm{lc}(S_i)$
    $i = i + 1$
return(S);

Theorem 6.3.12. *Let $f_1, f_2 \in R[x]$ be two polynomials s.t. $\deg(f_1) \geq \deg(f_2) \geq 0$. Suppose that Algorithms 2 and 12 do not fail for f_1 and f_2. Let $r_1, r_2, \ldots, r_l$ denote the m.p.r.s. from Definition*

2.2.6, and let $S_1, \ldots, S_l$ be the s.p.r.s. from Algorithm 12. For $1 \leq i \leq l$, set $d_i = \deg(S_i)$. Then

$$\begin{cases} S_1 & = r_1 \\ S_2 & = r_2 \\ S_3 & = (-\mathrm{lc}(S_2))^{d_1-d_2+1} r_3 \\ S_4 & = \frac{(-\mathrm{lc}(S_3))^{d_2-d_3+1}}{\mathrm{lc}(S_2)^{(d_1-d_2)(d_2-d_3)}} r_4 \quad \text{if } \deg(\mathrm{S_1}) - 1 \neq \deg(\mathrm{S_2}) \\ S_i & = \frac{(-\mathrm{lc}(S_{i-1}))^{d_{i-2}-d_{i-1}+1}}{\mathrm{lc}(S_{i-2})^{d_{i-2}-d_{i-1}}} r_i \quad \text{if } \deg(\mathrm{S_{i-3}}) - 1 = \deg(\mathrm{S_{i-2}}) \text{ for i} \geq 4 \\ S_i & = \frac{(-\mathrm{lc}(S_{i-1}))^{d_{i-2}-d_{i-1}+1} \mathrm{lc}(S_{i-3})^{(d_{i-3}-d_{i-2}-1)(d_{i-2}-d_{i-1})}}{\mathrm{lc}(S_{i-2})^{(d_{i-3}-d_{i-2})(d_{i-2}-d_{i-1})}} r_i \quad \text{if } \deg(\mathrm{S_{i-3}}) - 1 \neq \deg(\mathrm{S_{i-2}}) \text{ for i} > 4. \end{cases}$$

Proof. From Definition 2.2.6 and Algorithm 12, we have $r_1 = S_1 = f_1$ and $r_2 = S_2 = f_2$. Comparing the iterations of Algorithm 2 with the iterations of Algorithm 12, we prove the theorem. In the first iteration of Algorithm 2, we have

$$r_3 = M_1 - M_2\, q_3 = f_1 - \mathrm{lc}(f_2)^{-1} f_2\, q_3.$$

From Lemma 2.2.8, part (i), we have

$$\text{rem } (f_1, f_2, x) = r_3. \tag{6.20}$$

In the first iteration of Algorithm 12, we have $j = d_1 - 1$, $c = 1$, and $r = \deg(f_2) = d_2$. We set

$$S_3 = \frac{\text{prem}(S_1, S_2, x)}{(-c)^{j-r+2}} = \frac{\text{prem}(f_1, f_2, x)}{(-1)^{d_1-d_2+1}}.$$

From Lemma 2.2.7, part (ii), we have $\text{prem}(f_1, f_2, x) = \mathrm{lc}(f_2)^{d_1-d_2+1}\text{rem } (f_1, f_2, x)$. Substituting rem (f_1, f_2, x) from (6.20), we have

$$\text{prem}(f_1, f_2, x) = \mathrm{lc}(f_2)^{d_1-d_2+1} r_3,$$

which implies that

$$S_3 = (-\mathrm{lc}(S_2))^{d_1-d_2+1} r_3$$

If $j \neq r$, we set $S_2 = \mathrm{lc}(f_2)^{d_1-d_2-1} f_2$. In the second iteration of Algorithm 2, we have

$$r_4 = M_2 - M_3 q_4,$$

where $M_2 = \text{monic}(f_2)$ and $M_3 = \text{monic}(r_3)$. Applying part (i) of Lemma 2.2.7, we have

$$\text{rem } (f_2, r_3, x) = \mathrm{lc}(f_2) r_4.$$

In the second iteration of Algorithm 12, we have $i = 3$, $j = d_2 - 1$, $r = \deg(S_3)$, and $c = \mathrm{lc}(S_2)$. Two cases are possible for S_4. The first case occurs when $r = j$ in the first iteration. In this case,

$$S_4 = \frac{\mathrm{prem}(S_2, S_3, x)}{(-c)^{j-r+2}}.$$

Employing Lemma 2.2.7, part (iii), we have

$$\begin{aligned}\mathrm{prem}(S_2, S_3, x) &= \mathrm{lc}(S_2)\mathrm{lc}(S_3)^{d_2-d_3+1}\mathrm{mrem}(S_2, S_3, x)\\ &= \mathrm{lc}(S_2)\mathrm{lc}(S_3)^{d_2-d_3+1}r_4.\end{aligned}$$

Thus,

$$\begin{aligned}S_4 &= \frac{\mathrm{prem}(S_2, S_3, x)}{(-c)^{j-r+2}}\\ &= \frac{\mathrm{lc}(S_3)^{d_2-d_3+1}\mathrm{lc}(S_2)r_4}{(-\mathrm{lc}(S_2))^{d_2-d_3+1}}\\ &= \frac{(-\mathrm{lc}(S_3))^{d_2-d_3+1}}{(\mathrm{lc}(S_2))^{d_2-d_3}}r_4.\end{aligned}$$

The second case happens when $r \neq j$ in the first iteration. In this case, we replace S_2 by $\mathrm{lc}(S_2)^{d_1-d_2-1}S_2$. Hence, $c = \mathrm{lc}(S_2)^{d_1-d_2}$. Using Lemma 2.2.8, part (ii), we have

$$\mathrm{prem}(\mathrm{lc}(S_2)^{d_1-d_2-1}S_2, S_3, x) = \mathrm{lc}(S_2)^{d_1-d_2-1}\mathrm{prem}(S_2, S_3, x).$$

Moreover, using Lemma 2.2.7, part (iii), we have $\mathrm{prem}(S_2, S_3, x) = \mathrm{lc}(S_2)\mathrm{lc}(S_3)^{d_2-d_3+1}r_4$. Hence,

$$\begin{aligned}S_4 &= \frac{\mathrm{prem}(\mathrm{lc}(S_2)^{d_1-d_2-1}S_2, S_3, x)}{(-c)^{j-d_3+2}}\\ &= \frac{\mathrm{lc}(S_2)^{d_1-d_2-1}\mathrm{prem}(S_2, S_3, x)}{(-(\mathrm{lc}(S_2)^{d_1-d_2}))^{d_2-d_3+1}}\\ &= \frac{\mathrm{lc}(S_2)^{d_1-d_2-1}\mathrm{lc}(S_2)\mathrm{lc}(S_3)^{d_2-d_3+1}r_4}{(-(\mathrm{lc}(S_2)^{d_1-d_2}))^{d_2-d_3+1}}\\ &= \frac{(-\mathrm{lc}(S_3))^{d_2-d_3+1}}{(\mathrm{lc}(S_2)^{d_1-d_2})^{d_2-d_3}}r_4.\end{aligned}$$

Employing the same argument for $i > 4$, the result will be obtained. □

Example 6.3.7. *Let*

$$\begin{aligned}f_1 &= 24x^6 + 12zx^5 + 8x^4 + 2zx^3 + 8x^2 + zx + 4, \quad \textit{and}\\ f_2 &= 8x^6 + 8zx^5 + 4x^4 + 8zx^3 + 2zx + 4\end{aligned}$$

be two polynomials in $\mathbb{Z}[z]/\langle z^2 - 2\rangle[x]$. Table 6.1 compares the s.p.r.s. obtained from Algorithm 12 and the m.p.r.s. obtained from Algorithm 9 for the input polynomials f_1 and f_2. As seen, the coefficients of

the subresultants grow significantly. In particular, the last subresultant S_8 has the largest coefficients. Using 6.1 and Theorem 6.3.12, we can check the equalities in Theorem 6.3.12.

Table 6.1: Connection between s.p.r.s. and m.p.r.s.

s.p.r.s.	**m.p.r.s.**
$S_1 = 24x^6 + 12zx^5 + 8x^4 + 2zx^3 + 8x^2 + zx + 4$	$r_1 = 24x^6 + 12zx^5 + 8x^4 + 2zx^3 + 8x^2 + xz + 4$
$S_2 = 8x^6 + 8zx^5 + 4x^4 + 8zx^3 + 2zx + 4$	$r_2 = 8x^6 + 8zx^5 + 4x^4 + 8zx^3 + 2zx + 4$
$S_3 = 96zx^5 + 32x^4 + 176zx^3 - 64x^2 + 40zx + 64$	$r_3 = -12zx^5 - 4x^4 - 22zx^3 + 8x^2 - 5zx - 8$
$S_4 = -29696x^4 - 3584zx^3 + 2560x^2 - 7936zx - 1024$	$r_4 = -\frac{29}{18}x^4 - \frac{7}{36}zx^3 + \frac{5}{36}x^2 - \frac{31}{72}zx - \frac{1}{18}$
$S_5 = 8765440zx^3 - 5480448x^2 + 1642496zx + 3047424$	$r_5 = \frac{1605}{841}x^3 - \frac{2007}{3364}zx^2 + \frac{1203}{3364}x + \frac{279}{841}z$
$S_6 = 13828096x^2 - 63209472zx + 601096192$	$r_6 = -\frac{6119}{2289800}x^2 + \frac{55941}{4579600}zx - \frac{66497}{572450}$
$S_7 = -47448064zx - 4034396160$	$r_7 = -\frac{193670}{44521}x - \frac{8233650}{44521}z$
$S_8 = 1317760139264$	$r_8 = \frac{132583327}{32761}$

Corollary 6.3. *Let $f_1, f_2 \in \bar{L}_p[x_1]$ be two polynomials s.t. $\deg(f_1) \geq \deg(f_2) \geq 0$. Suppose that Algorithm 2 and Algorithm 12 do not fail for f_1 and f_2 over $R = \bar{L}_p$. Let $r_1, r_2, \ldots, r_l$ denote the m.p.r.s. defined in Definition 2.2.6 and $S_1, \ldots, S_h$ be the s.p.r.s. obtained from Algorithm 12. Then*

(i) $l = h$

(ii) $\mathrm{lc}(r_i, x_1) = u \cdot \mathrm{lc}(S_i, x_1)$ *for a unit* $u \in \bar{L}_p$.

(iii) $\deg(r_i, x_1) = \deg(S_i, x_1)$

Proof. (i) Since Algorithm 2 does not fail for f_1 and f_2, $\mathrm{lc}(r_i)$ is invertible for $1 \leq i \leq l$. Thus, we can alternate natural division with pseudo-division to compute S_i for $1 \leq i \leq l$. According to Theorem 6.3.12, since $\mathrm{lc}(r_i)$ is invertible, $\mathrm{lc}(S_i)$ is also invertible for $1 \leq i \leq l$. Hence, Algorithm 12 must have the same number of division steps as Algorithm 2, which implies that $l = h$.

(ii) From part (i), Algorithm 12 terminates after l iterations, so $\mathrm{lc}(S_i)$ is invertible for $1 \leq i \leq l$. Thus, in Theorem 6.3.12, fractions are units. Accordingly, $\mathrm{lc}(r_i, x_1) = u \cdot \mathrm{lc}(S_i, x_1)$ for some unit $u \in \bar{L}_p$.

(iii) Since Algorithm 12 terminates after l iterations, $\mathrm{lc}(S_i)$ is invertible for $1 \leq i \leq l$. Thus, from Theorem 6.3.12, we have $S_i = u_i r_i$ for a unit u_i. Multiplying by a unit does not change $\deg(r_i, x_1)$, so $\deg(r_i, x_1) = \deg(S_i, x_1)$.

□

The second way of defining s.p.r.s. is to use determinants (see [18, Definition 7.3]). In our failure probability analyses, we use determinants because they are more convenient for deriving degree and height bounds for the remainders (as determinants) in the s.p.r.s. and, consequently, for the remainders in the m.p.r.s.

Definition 6.3.3. *Let $f_1 = \sum_{i=1}^{m} a_i x_1^i$ and $f_2 = \sum_{i=1}^{n} b_i x_1^i \in R[x_2, \ldots, x_k][x_1]$ with $0 < n \leq m$. Let $M_{i,j}$ be the $(m + n - 2j) \times (m + n - 2j)$ matrix determined from $\mathrm{sylv}(f_1, f_2, x_1)$ by deleting*

- *rows $n - j + 1$ to n;*
- *rows $m + n - j + 1$ to $m + n$;*

- *columns $m+n-2j$ to $n+m$ except for column $m+n-i-j$.*

The jth subresultant of f_1 and f_2 w.r.t. x_1 is the polynomial of degree j defined by

$$S(j, f_1, f_2, x_1) = \det(M_{0j}) + \det(M_{1j})x_1 + \cdots + \det(M_{jj})x_1^j.$$

Since $\det(M_{ij}) = 0$ for $i > j$, we can express $S(j, f_1, f_2, x_1)$ as

$$S(j, f_1, f_2, x_1) = \det\left(\begin{bmatrix} a_m & a_{m-1} & \cdots & & a_1 & a_{2j-n} & x_1^{n-j-1}f_1 \\ & a_m & a_{m-1} & \cdots & & & \\ & & & \cdots & & & \\ & & & & a_m & a_{j+1} & f_1 \\ b_n & b_{n-1} & \cdots & & b_1 & b_{2j-m} & x_1^{m-j-1}f_2 \\ & b_n & b_{n-1} & \cdots & & & \\ & & & \cdots & & & \\ & & & & b_n & b_{j+1} & f_2 \end{bmatrix}\right).$$

It is clear that $S(0, f_1, f_2, y) = \det(M_{0,0}) = \operatorname{res}(f_1, f_2, y)$.

Example 6.3.8. *Let*

$$f_1(x) = 2x^4 + 3zx^3 + x^2 + zx + 3 \quad \textit{and}$$
$$f_2(x) = x^3 + zx^2 + 5x + 4$$

be polynomials in $\mathbb{Z}[z]/\langle z^2 - 2\rangle[x]$ with $\gcd(f_1, f_2) = 1$. Then

$$\operatorname{sylv}(f_1, f_2, x) = \begin{bmatrix} 2 & 3z & 1 & z & 3 & 0 & 0 \\ 0 & 2 & 3z & 1 & z & 3 & 0 \\ 0 & 0 & 2 & 3z & 1 & z & 3 \\ 1 & z & 5 & 4 & 0 & 0 & 0 \\ 0 & 1 & z & 5 & 4 & 0 & 0 \\ 0 & 0 & 1 & z & 5 & 4 & 0 \\ 0 & 0 & 0 & 1 & z & 5 & 4 \end{bmatrix}.$$

By Definition 6.3.3, we have

$$S(0, f_1, f_2, x) = \det(\operatorname{sylv}(f_1, f_2, x)) = \operatorname{res}(f_1, f_2, x) = -11336z + 18262.$$

To compute $S(1, f_1, f_2, x)$, we need $M_{0,1} = \begin{bmatrix} 2 & 3z & 1 & z & 0 \\ 0 & 2 & 3z & 1 & 3 \\ 1 & z & 5 & 4 & 0 \\ 0 & 1 & z & 5 & 0 \\ 0 & 0 & 1 & z & 4 \end{bmatrix}$, *and* $M_{1,1} = \begin{bmatrix} 2 & 3z & 1 & z & 3 \\ 0 & 2 & 3z & 1 & z \\ 1 & z & 5 & 4 & 0 \\ 0 & 1 & z & 5 & 4 \\ 0 & 0 & 1 & z & 5 \end{bmatrix}$ *so*

$$S(1, f_1, f_2, x) = \det(M_{0,1}) + x\det(M_{1,1}) = (-68z + 646)x + 53z + 404.$$

To compute $S(2, f_1, f_2, x)$, *we need* $M_{0,2} = \begin{bmatrix} 2 & 3z & 3 \\ 1 & z & 0 \\ 0 & 1 & 4 \end{bmatrix}$, $M_{1,2} = \begin{bmatrix} 2 & 3z & z \\ 1 & z & 4 \\ 0 & 1 & 5 \end{bmatrix}$, *and* $M_{2,2} = \begin{bmatrix} 2 & 3z & 1 \\ 1 & z & 5 \\ 0 & 1 & z \end{bmatrix}$

so

$$S(2, f_1, f_2, x) = \det(M_{0,2}) + \det(M_{1,2})x + \det(M_{2,2})x^2 = -11x^2 + (-4z - 8)x - 4z + 3.$$

Employing Algorithm 12 to compute the subresultants, we have 6.2

Table 6.2: s.p.r.s. by Algorithm 12

s.p.r.s.
$S_1 = 2x^4 + 3zx^3 + x^2 + zx + 3$
$S_2 = x^3 + zx^2 + 5x + 4$
$S_3 = -11x^2 + (-4z - 8)x - 4z + 3$
$S_4 = (-68z + 646)x + 53z + 404$
$S_5 = -11336z + 18262$

As illustrated in Example 6.3.8,

$$S_3 = S(1, f_1, f_2, x)$$
$$S_4 = S(2, f_1, f_2, x)$$
$$S_5 = S(0, f_1, f_2, x).$$

Theorem 6.3.13 confirms that the subresultant polynomial remainder sequences (s.p.r.s.) defined in Definitions 6.3.2 and 6.3.3 have the same polynomials; the only difference lies in the indexing of the subresultants.

Theorem 6.3.13. *[8, Subresultant Chain Algorithm, page 129] Let* $S_1, \ldots, S_l$ *be the s.p.r.s. obtained from Algorithm 12 for the input polynomials* $f_1, f_2 \in R'[x_1]$ *where* $R' = R[x_2, \ldots, x_k]$ *and* $\deg(f_1, x_1) \geq \deg(f_2, x_1) \geq 0$. *Let* $0 \leq j \leq \deg(f_2, x_1)$ *and* $S(j, f_1, f_2, x_1) \neq 0$. *Then there exists* $1 \leq i \leq l$ *s.t.* $S(j, f_1, f_2, x_1) = S_i$. *In particular,* $S(0, f_1, f_2, x_1) = S_l$.

Notation 6.4. *Let* $f_1, f_2 \in R[x_1, \ldots, x_k]$. *In this section, we use the following terminology:*

(i) $R_{x_1} = \operatorname{res}(f_1, f_2, x_1) \in R[X, z]$, *where* $X = x_2, \ldots, x_k$,

(ii) $D = \max_{j=1}^{k}(\deg(f_1, x_j), \deg(f_2, x_j))$, *and*

(iii) $H = \max(\|f_1\|_\infty, \|f_2\|_\infty)$.

We summarize some properties of R_{x_1} in Proposition 6.3.1 below, without proof. Recall that $\bar{L}_{\mathbb{Z}} = \mathbb{Z}[z]/\langle \check{M}(z) \rangle$ and $\bar{L}_p = \mathbb{Z}_p[z]/\langle \check{M}(z) \rangle$.

Proposition 6.3.1. *Let* $f_1 = \sum_{i=0}^{m} a_i(X, z)x_1^i$ *and* $f_2 = \sum_{i=0}^{n} b_i(X, z)x_1^i$ *be two non-zero polynomials in* $\bar{L}_{\mathbb{Z}}[X, z][x_1]$, *where* $X = x_2, \ldots, x_k$. *Let* $m = \deg(f_1, x_1)$ *and* $n = \deg(f_2, x_1)$ *s.t.* $m \geq n \geq 0$. *Let* $d = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$. *Then*

(i) $\deg(R_{x_1}, X) \leq 2D^2$, *where* $\deg(R_{x_1}, X)$ *is the total degree of* R_{x_1} *w.r.t. the variables* $x_2, \ldots, x_k$,

(ii) $\deg(R_{x_1}, z) \leq 2D(d - 1)$,

(iii) $\#R_{x_1} \leq (2D(D+1))^{k-1}$, *where* $\#R_{x_1}$ *is the number of monomials of* R_{x_1} *in* $X = x_2, \ldots, x_k$,

(iv) $\|R_{x_1}\|_\infty \leq (2D(D+1))^{k-1} H^{2D}$.

Example 6.3.9. *Let* $M(z) = z^2 - 2$ *and* $\bar{L}_\mathbb{Z} = \mathbb{Z}[z]/\langle \check{M} \rangle$. *Let*

$$f_1 = zx_1^3 + zx_2^2x_1 + 2x_2, \text{ and}$$
$$f_2 = x_1^2 + x_1x_2 + (x_2z + z + 8)$$

be two polynomials in $\bar{L}_\mathbb{Z}[x_1, x_2]$. *Then*

$$R_{x_1} = \operatorname{res}(f_1, f_2, x_1) = 2x_2^5z^3 + (-2z^4 + 2z^3 + 16z^2 - 4z)x_2^4 + (z^5 - 4z^4 - 32z^3 + 6z^2)x_2^3$$
$$+ (3z^5 + 22z^4 - 32z^3 - 122z^2 + 48z + 4)x_2^2 + (3z^5 + 48z^4 + 192z^3)x_2 + z^5 + 24z^4 + 192z^3 + 512z^2.$$

We have

- $m = \deg(f_1, x_1) = 3$ *and* $n = \deg(f_2, x_1) = 2$,
- $D = \max_{i=1}^{2}(\deg(f_1, x_i), \deg(f_2, x_i)) = 3$,
- $H = \max(\|f_1\|_\infty, \|f_2\|_\infty) = 8$.

Therefore,

(i) $\deg(R_{x_1}, x_2) = 5 \leq 2D^2 = 18$.

(ii) $\deg(R_{x_1}, z) = 5 \leq 2D(d-1) = 6$.

(iii) $\#R_{x_1} = 6 \leq (2D(D+1))^{k-1} = 24$.

(iv) $\|R_{x_1}\|_\infty = 512 \leq (2D(D+1))^{k-1} H^{(2D)} = 24 \times 8^6 = 6,291,456$.

The probability that PGCDNF encounters a zero divisor pair

As mentioned at the beginning of Section 6.3.3, to bound the probability that PGCDNF encounters a zero divisor pair, we first need to bound the probability that MEA fails over $\bar{L}_p[x_1]$. To do so, we use the following theorem.

Theorem 6.3.14. *Let* $f_1, f_2 \in \bar{L}_\mathbb{Z}[x_2, \ldots, x_k][x_1]$. *Assume the following.*

(i) Let A *be the change-of-basis matrix and* $\check{M}$ *be the generator polynomial obtained from the LAminpoly (Algorithm 4) over* $\mathbb{F} = \mathbb{Q}$.

(ii) Let p *be a prime and let* $\beta \in [0, p)^{k-1}$ *s.t.* (p, β) *is not lc-bad and* $p \nmid \det(A)$.

(iii) For $i \geq 1$, *let* S_i *be the* i*th subresultant generated by* f_1 *and* f_2 *w.r.t.* x_1, *computed over* $R = \bar{L}_\mathbb{Z}[x_2, \ldots, x_k][x_1]$.

(iv) For $i \geq 1$, *define* $l_i = \operatorname{lc}(S_i, x_1) \in \mathbb{Z}[x_2, \ldots, x_k][z]$.

If the MEA fails for the input polynomials $\phi_p(f_1)(\beta), \phi_p(f_2)(\beta) \in \bar{L}_p[x_1]$, *then there exists* $i \geq 2$ *s.t.* $\phi_p(l_i(\beta)) \neq 0$ *and* $p \mid \operatorname{res}(l_i(\beta), \check{M}, z)$.

Proof. By assumption, the MEA fails for the input polynomials $\phi_p(f_1)(\beta)$ and $\phi_p(f_2)(\beta) \in \bar{L}_p[x_1]$. Hence, there exists a remainder $r_i \neq 0$, with $i \geq 2$, s.t. $\mathrm{lc}(r_i)$ is not invertible in $\bar{L}_p = \mathbb{Z}_p[z]/\langle \phi_p(\check{M})(z) \rangle$. By definition of $\bar{L}_p$,

$$\mathrm{lc}(r_i) \text{ is not invertible in } \bar{L}_p \iff \gcd(\mathrm{lc}(r_i), \phi_p(\check{M})) \neq 1. \tag{6.21}$$

The relation between the s.p.r.s. and the m.p.r.s. (as stated in Theorem 6.3.12 and Corollary 6.3), implies that there exists a unit $u_i \in \bar{L}_p$ s.t. $\phi_p(S_i(\beta)) = u_i \cdot r_i$, so

$$\phi_p(l_i(\beta)) = u_i \cdot \mathrm{lc}(r_i). \tag{6.22}$$

Because $r_i \neq 0$, we have $\mathrm{lc}(r_i) \neq 0$, and since u_i is a unit, it follows that $\phi_p(l_i(\beta)) \neq 0$ in $\bar{L}_p$. Therefore, from Equations (6.22) and (6.21), we have

$$\gcd(\mathrm{lc}(r_i), \phi_p(\check{M})) \neq 1 \iff \gcd(\phi_p(l_i(\beta)), \phi_p(\check{M})) \neq 1. \tag{6.23}$$

From Theorem 5.2.3,

$$\gcd(\phi_p(l_i(\beta)), \phi_p(\check{M})) \neq 1 \iff \mathrm{res}(\phi_p(l_i(\beta)), \phi_p(\check{M}), z) = 0. \tag{6.24}$$

Since $p \nmid \det(A)$, LAminpoly does not fail over $\mathbb{F} = \mathbb{Z}_p$, and returns $\phi_p(\check{M}) \neq 0$. Therefore, as $\phi_p(\check{M}) \neq 0$ and $\phi_p(l_i(\beta)) \neq 0$, Remark 5.2 yields

$$\mathrm{res}(\phi_p(l_i(\beta)), \phi_p(\check{M}), z) = \phi_p(\mathrm{res}(l_i(\beta), \check{M}, z)), \tag{6.25}$$

which implies that,

$$\begin{aligned} \mathrm{res}(\phi_p(l_i(\beta)), \phi_p(\check{M}), z) = 0 &\iff \phi_p(\mathrm{res}(l_i(\beta), \check{M}, z)) = 0 \\ &\iff p \mid \mathrm{res}(l_i(\beta), \check{M}, z). \end{aligned}$$

Therefore, if the MEA fails for polynomials in $\bar{L}_p[x_1]$, then $p \mid \mathrm{res}(l_i(\beta), \check{M}, z)$ for some $i \geq 2$. □

Corollary 6.4. *The number of zero divisor pairs is finite.*

Proof. A direct consequence of Theorem 6.3.14. □

Theorem 6.3.15. *Under assumptions (i)-(iv) of Theorem 6.3.14. If the MEA fails for $\phi_p(f_1)(\beta)$ and $\phi_p(f_2)(\beta) \in \bar{L}_p[x_1]$, then for each $i \geq 2$*

$$\mathrm{res}(l_i(\beta), \check{M}, z) \neq 0.$$

Proof. Assume, for contradiction, that there exists $i \geq 2$ s.t. $\mathrm{res}(l_i(\beta), \check{M}, z) = 0$, so $\gcd(l_i(\beta), \check{M}) \neq 1$ (see Theorem 5.2.3). From assumption (i) of Theorem 6.3.14, $\check{M}$ is produced by LAminpoly over $\mathbb{F} = \mathbb{Q}$. Hence, by Theorem 3.1, $\check{M}$ is irreducible. Therefore, $\gcd(l_i(\beta), \check{M}) \neq 1$ implies that $\gcd(l_i(\beta), \check{M}) = \check{M}$. Hence, there exists $h \in \mathbb{Z}[z]$ s.t. $l_i(\beta) = \check{M} \cdot h$. Reducing modulo p gives

$$\phi_p(l_i(\beta)) = \phi_p(\check{M}) \cdot \phi_p(h). \tag{6.26}$$

The right hand side of the above equation is zero in $\bar{L}_p = \mathbb{Z}_p[z]/\langle \phi_p(\check{M}) \rangle$, which means $\phi_p(l_i(\beta)) = 0$ in $\bar{L}_p$. On the other hand, because the MEA fails, all remainders computed up to the failure are non-zero. In particular, the remainder $r_i \in \bar{L}_p[x_1]$ corresponding to $\phi_p(S_i)$ is not zero, so $\mathrm{lc}(r_i) \neq 0$ in $\bar{L}_p$. From Corollary 6.3, there exists a unit $u_i \in \bar{L}_p$ s.t.

$$\phi_p(l_i(\beta)) = u_i \cdot \mathrm{lc}(r_i). \tag{6.27}$$

Substituting $\phi_p(l_i(\beta))$ from Equation (6.27) into Equation (6.26) yields

$$u_i \cdot \mathrm{lc}(r_i) = \phi_p(\check{M}) \cdot \phi_p(h).$$

Since $u_i \in \bar{L}_p$ is a unit, it follows that $\mathrm{lc}(r_i) = 0$ in $\bar{L}_p$, which contradicts the assumption that $\mathrm{lc}(r_i) \neq 0$. Therefore, no such index i can exist, and we conclude that for every $i \geq 2$, $\mathrm{res}(l_i(\beta), \check{M}, z) \neq 0$. □

Let $\deg(f_1, x_1) \geq \deg(f_2, x_1) \geq 0$. Then the number of remainders in s.p.r.s. is bounded by $l \leq \deg(f_2, x_1) + 2$. By Theorem 6.3.15, if the MEA fails over $\bar{L}_p[x_1]$, then $\mathrm{res}(l_i(\beta), \check{M}, z) \neq 0$ for each $i \geq 2$. Thus, from Theorem 6.3.14, we have

$$\begin{aligned} \text{Prob[MEA fails over } \bar{L}_p[x_1]] &\leq \text{Prob}[p \mid \prod_{i=2}^{l} \mathrm{res}(l_i(\beta), \check{M}, z)] \\ &\leq \sum_{i=2}^{l} \text{Prob}[p \mid \mathrm{res}(l_i(\beta), \check{M}, z)]. \end{aligned} \tag{6.28}$$

To bound $\text{Prob}[p \mid \mathrm{res}(l_i(\beta), \check{M}, z)]$ in (6.28), we first need an upper bound on $\mathrm{res}(l_i(\beta), \check{M}, z) \in \mathbb{Z}$. By Hadamard's bound (Theorem 6.2.1), this requires upper bounds on $\|\check{M}(z)\|_\infty$ and $\|l_i(\beta)\|_\infty$. Theorem 6.3.11 already yields $\|\check{M}(z)\|_\infty \leq B_M$, so it remains to derive a bound on $\|l_i(\beta)\|_\infty$. For $0 \leq i \leq l$, suppose that $\|S_i\|_\infty \leq B_i$ and $\|R_{x_1}\|_\infty \leq B_l$. Due to coefficient growth in Algorithm 12, we have $B_i \leq B_l$. Moreover, $\|l_i\|_\infty \leq \|S_i\|_\infty$, which implies that $\|l_i\|_\infty \leq B_i \leq B_l$. Example 6.3.10 illustrates this.

Example 6.3.10. *Let f_1 and f_2 be the polynomials in Example 6.3.7. Note that f_1 and f_2 are two univariate polynomials so we do not need to be worried about substituting S_i at β. In this case,*

Table 6.3: s.p.r.s.

s.p.r.s.	$l_i = \mathrm{lc}(S_i, x)$	$\|l_i\|_\infty$	$\|S_i\|_\infty$
$S_1 = 24x^6 + 12zx^5 + 8x^4 + 2zx^3 + 8x^2 + xz + 4$	$l_1 = 24$	24	24
$S_2 = 8x^6 + 8zx^5 + 4x^4 + 8zx^3 + 2zx + 4$	$l_2 = 8$	8	8
$S_3 = 96zx^5 + 32x^4 + 176zx^3 - 64x^2 + 40zx + 64$	$l_3 = 96z$	96	176
$S_4 = -29696x^4 - 3584zx^3 + 2560x^2 - 7936zx - 1024$	$l_4 = -29696$	29696	29696
$S_5 = 8765440zx^3 - 5480448x^2 + 1642496zx + 3047424$	$l_5 = 8765440z$	8765440	8765440
$S_6 = 13828096x^2 - 63209472zx + 601096192$	$l_6 = 13828096$	13828096	601096192
$S_7 = -47448064zx - 4034396160$	$l_7 = -47448064z$	47448064	4034396160
$S_8 = 1317760139264$	$l_8 = 1317760139264$	1317760139264	1317760139264

$\mathrm{res}(f_1, f_2) \neq 0$. *Thus, we can use a bound for $\|\mathrm{res}(f_1, f_2)\|_\infty$ as a bound for $\|S_i\|_\infty$ for $1 \leq i \leq 7$. For $1 \leq i \leq 8$, we have $\|l_i\|_\infty \leq \|S_i\|_\infty \leq 131776013926$.*

Theorem 6.3.16. *Let f_1 and f_2 be two non-zero polynomials in $L_{\mathbb{Z}}[x_1, \ldots, x_k]$. Consider Notation 6.4. Let p be a prime and let $\beta \in [0, p)^{k-1}$. Let*

$$B_l = p^{2D^2}(2D(D+1))^{2k-2}H^{2D}.$$

Then $\|l_i(\beta)\|_\infty \leq B_l$.

Proof. Due to the coefficient growth in the s.p.r.s. algorithm, to bound $\|l_i(\beta)\|_\infty$, it is sufficient to bound $\|R_{x_1}(\beta)\|_\infty$. From Proposition 6.3.1, we have

- $\deg(R_{x_1}, X) \leq (\deg(f_1, x_1) + \deg(f_2, x_1))D \leq 2D^2$ (part (i)),
- $\#R_{x_1} \leq \Big((\deg(f_1, x_1) + \deg(f_2, x_1))(D+1)\Big)^{k-1} \leq (2D(D+1))^{k-1}$ (part(iii)), and
- $\|R_{x_1}\|_\infty \leq (2D(D+1))^{k-1}H^{(\deg(f_1,x_1)+\deg(f_2,x_1))} \leq (2D(D+1))^{k-1}H^{2D}$ (part (iv)).

Therefore,

$$\begin{aligned}\|l_i(\beta)\|_\infty &\leq \|R_{x_1}(\beta)\|_\infty \\ &\leq p^{\deg(R_{x_1},X)}\#R_{x_1}\|R_{x_1}\|_\infty \\ &\leq p^{2D^2}(2D(D+1))^{2k-2}H^{2D}.\end{aligned}$$

□

Lemma 6.3.17. *Let $\|\check{M}(z)\|_\infty \leq B_M$ (Theorem 6.3.11) and $\|l_i(\beta)\|_\infty \leq B_l$ (Theorem 6.3.16). Let $p \in \mathbb{P}_b$ and let $\beta \in [0, p)^{k-1}$. Then,*

$$\text{Prob}[p \mid \text{res}(l_i(\beta), \check{M}, z)] \leq \frac{\lfloor (d(2D+1))/2 \log_2(dB_l^2 + 2dDB_M^2) \rfloor}{(b-1) \mid \mathbb{P}_b \mid}.$$

Proof. Let $S = \text{sylv}(l_i(\beta), \check{M}, z)$ and let $m = \deg(f_1, x_1)$ and $n = \deg(f_2, x_1)$. Then S contains $d = \deg(\check{M})$ rows of coefficients of $l_i(\beta)$ and $\deg(l_i(\beta), z)$ rows of coefficients of $\check{M}$. To bound $\deg(l_i(\beta), z)$, recall Definition 6.3.3, so S_i can be defined as the determinant of a matrix of size $(m+n-2i) \times (m+n-2i)$. Hence,

$$\deg(l_i(\beta), z) \leq \deg(S_i, z) \leq (m+n-2i)d \leq (m+n)d \leq 2dD.$$

Therefore, S has at most $2dD$ rows of coefficients of $\check{M}(z)$. Since $\|l_i(\beta)\|_\infty \leq B_l$ and $\|M(z)\|_\infty \leq B_M$, by Hadamard's bound, we have

$$\begin{aligned}\mid \text{res}(l_i(\beta), \check{M}, z) \mid = \mid \det(S) \mid &\leq \prod_{i=1}^{d(2D+1)} \sqrt{\sum_{j=1}^{d(2D+1)} S_{i,j}^2} \leq \prod_{i=1}^{d(2D+1)} \sqrt{\sum_{j=1}^{d} \underbrace{B_l^2}_{\text{Theorem 6.3.16}} + \sum_{j=1}^{d(2D)} \underbrace{B_M^2}_{\text{Theorem 6.3.11}}} \\ &\leq (dB_l^2 + 2dDB_M^2)^{d(2D+1)/2}.\end{aligned}$$

Hence, $\text{Prob}[p \mid \text{res}(l_i(\beta), \check{M}, z)] \leq \frac{\lfloor (d(2D+1))/2 \log_2(dB_l^2+2dDB_M^2) \rfloor}{(b-1)\mid \mathbb{P}_b \mid}$. □

We now estimate the probability that MEA fails in $\bar{L}_p[x_1]$.

Corollary 6.5. *Let $f_1, f_2 \in \bar{L}_{\mathbb{Z}}[x_2, \ldots, x_k][x_1]$ with $\deg(f_1, x_1) \geq \deg(f_2, x_1) \geq 0$. Assume the following.*

(i) Let $S_1, \ldots, S_l \in \bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k][z]$ be the s.p.r.s. generated by f_1 and f_2.

(ii) Let l be the number of the s.p.r.s., then $l \leq \deg(f_2, x_1) + 2 \leq D + 2$.

(iii) Let $l_i = \mathrm{lc}(S_i, x_1) \in \mathbb{Z}[x_2, \ldots, x_k][z]$ be as defined in Theorem 6.3.14, and let $\beta \in [0, p)^{k-1}$. Then $l_i(\beta) \in \mathbb{Z}[z]$.

Substituting the bound obtained from Theorem 6.3.17 into Equation (6.28), we have

$$\begin{aligned} \mathrm{Prob}[\textit{MEA fails over } \bar{L}_p[x_1]] &\leq \sum_{i=2}^{l} \mathrm{Prob}[p \mid \mathrm{res}(l_i(\beta), \check{M}, z)] \\ &\leq \underbrace{(D+1)\frac{\lfloor (d(2D+1)/2)\,(\log_2(dB_l^2 + 2dDB_M^2))\rfloor}{(b-1) \mid \mathbb{P}_b \mid}}_{B}. \end{aligned} \tag{6.29}$$

We define B as the right-hand side of Equation (6.29).

Theorem 6.3.18. *Let $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$. Let B be as defined in Corollary 6.5. Then*

$$\mathrm{Prob}[\textit{PGCDNF encounters a zero divisor pair}] \leq 3k(D+1)^{k-1}B.$$

Proof. In PGCDNF, the probability that any GCD computation in Lines 2, 5, 6, 7, 9, or 30 fails is bounded by B. Let $\#\beta$ be the number of evaluation points required to interpolate $x_2, \ldots, x_k$ in the GCD. Then, $\#\beta \leq (D+1)^{k-1}$.

Line 2 performs at most $(D+1)^{k-1}$ GCD computations in $\bar{L}_p[x_1]$.

Next, we count the number of GCD computations required when computing the content of a polynomial $f_1 \in \bar{L}_p[x_k][x_1, \ldots, x_{k-1}]$, in Line 5. Let $C(k)$ denote this number. For $k = 1$, no content computation is required, so $C(1) = 0$. For $k = 2$, the polynomial is viewed as an element of $\bar{L}_p[x_2][x_1]$. Computing its content requires taking the GCD of at most $D+1$ coefficients in $\bar{L}_p[x_2]$, and hence requires at most D GCD computations. Now suppose $k > 2$. At the top level, computing the content w.r.t. $X_k = x_1, \ldots, x_{k-1}$ requires $(D+1)^{k-1} - 1$ GCD computations. This process is repeated for at most $D+1$ evaluation points of x_k. Therefore,

$$\begin{aligned} C(1) &= 0 \\ C(2) &= D \\ C(k) &= (D+1)^{k-1} - 1 + (D+1)C(k-1). \end{aligned}$$

Solving for $C(k)$, in Line 5, there is at most $(k-1)(D+1)^{k-1}$ GCD computations. The same applies to Lines 6, and 30. Lines 7 and 9 each involve one GCD computation, which we can include in the count for Lines 5 and 6. Therefore, we have the following result.

$$\mathrm{Prob}[\text{PGCDNF encounters a zero divisor}] \leq \underbrace{(D+1)^{k-1}B}_{\text{Line 2}} + \underbrace{3(k-1)(D+1)^{k-1}B}_{\text{Lines 5, 6, and 30}} \leq 3k(D+1)^{k-1}B.$$

□

Example 6.3.11. *Let* $f_1, f_2 \in L[x_1, x_2]$ *be the polynomials from Example 6.3.1 over* $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - \frac{2}{5}, z_2^2 - \frac{7}{3}\rangle$ *with*

$$\check{f}_1 = 1305\, z_1\, x_2^6 x_1^6 + 1305\, z_2\, x_2^5 x_1^5 + 582\, z_2\, x_2\, x_1^2 + (-840\, z_2 z_1\, x_2 + 3395\, z_1)\, x_1 - 1960,$$
$$\check{f}_2 = 255\, z_2 z_1\, x_2^4 x_1^5 + 595\, x_2^3 x_1^4 + 24\, z_2\, x_2^2 x_1^2 + (140\, z_1 - 492)\, x_2 x_1 - 1230\, z_2 z_1.$$

We use the primes $p \in \mathbb{P}_{31}$*, so* $2^{30} < p < 2^{31}$*. In this example, we use the following parameters:*

- *The number of variables is* $k = 2$*.*
- *The number of extensions is* $n = 2$*.*
- *As in Notation 6.3,* $D_i = \frac{d}{\prod_{j=1}^{i} d_{n-j+1}}$*, so* $D_1 = 2$ *and* $D_2 = 1$*.*
- $\check{M}_1 = 5z_1^2 - 2$*, so* $\|\check{M}_1\|_\infty \leq 2^3$ *and* $\mathrm{lc}(\check{M}_1) = 5$*.*
- $\check{M}_2 = 3z_2^2 - 7$*, so* $\|\check{M}_2\|_\infty \leq 2^3$ *and* $\mathrm{lc}(\check{M}_2) = 3$*.*
- $\delta_1 = 3$ *and* $\delta_2 = 6$*.*
- $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i)) = 6$*.*
- $H = \max(\|\check{f}_1\|_\infty, \|\check{f}_2\|_\infty) = 3395$*.*

Let $\gamma = z_1 + 3z_2$*. Then* $\log_2 \|\gamma^d\|_\infty = \log_2 108 \leq C = 7$*, so*

$$B_M = d^{d/2}\left(2^C \prod_{i=1}^{n}(l_{n-i+1} + D_i\|\check{M}_{n-i+1}\|_\infty)^{\delta_i}\right)^d \approx 2.5 \times 10^{48} \text{ and}$$
$$B_l = p^{2D^2}(2D(D+1))^{2k-2}H^{2D} \approx 1.31 \times 10^{718}.$$

By Theorem 6.3.18, the probability that PGCDNF encounters a zero divisor pair is bounded by

$$3k(D+1)^{k-1}\frac{\lfloor (d(2D+1)/2)\,(\log_2(dB_l^2 + 2dDB_M^2))\rfloor}{(b-1)\mid \mathbb{P}_b \mid} \approx 3.43 \times 10^{-3}.$$

6.3.4 Unlucky primes

Let p be a prime chosen randomly from $\mathbb{P}_b$. In this section, we bound the probability that p is an unlucky prime. To do so, assume the following.

- Let $f_1, f_2 \in \bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k]$ be two non-zero polynomials, and let $g = \gcd(f_1, f_2)$ be their monic GCD.
- Let h_1 and h_2 denote the cofactors of f_1 and f_2, respectively, so that $f_1 = h_1 \cdot g$ and $f_2 = h_2 \cdot g$.
- Let $p \nmid \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i) \cdot \mathrm{lc}(f_1, x_1) \cdot \mathrm{lc}(f_2, x_1)$, and $p \nmid \det(A)$.
- Let $\gcd(\phi_p(f_1), \phi_p(f_2))$ exist and let $g_p = \gcd(\phi_p(\check{h}_1), \phi_p(\check{h}_2))$.

By Definition 4.4.5, p is an unlucky prime if $\deg(g_p) > 0$, or equivalently, $g_p \neq 1$. Therefore, we have

$$\begin{aligned} g_p \neq 1 &\iff \deg(g_p, x_i) > 0, \text{ for some } 1 \leq i \leq k \\ &\iff \underbrace{\operatorname{res}(\phi_p(\check{h}_1), \phi_p(\check{h}_2), x_i) = 0}_{\text{Theorem 5.2.3}}, \text{ for some } 1 \leq i \leq k \\ &\iff \underbrace{\phi_p(\operatorname{res}(\check{h}_1, \check{h}_2, x_i)) = 0}_{\text{Remark 5.2}}, \text{ for some } 1 \leq i \leq k, \\ &\iff p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i), \text{ for some } 1 \leq i \leq k, \\ &\iff p \mid \prod_{i=1}^{k} \operatorname{res}(\check{h}_1, \check{h}_2, x_i). \end{aligned} \tag{6.30}$$

Example 6.3.12 illustrates the above argument.

Example 6.3.12. *Let $\bar{L}_{\mathbb{Z}} = \mathbb{Z}[z]/\langle z^2 - 2\rangle$. Consider the polynomials $f_1, f_2 \in \bar{L}_{\mathbb{Z}}[x_1, x_2]$ given by*

$$\begin{aligned} f_1 &= (x_1 + z)\underbrace{(zx_1^2 + 2x_2x_1 + 7z(x_2 - 3))}_{\check{h}_1}, \\ f_2 &= (x_1 + z)\underbrace{(zx_1^2 + 14x_2x_1)}_{\check{h}_2}. \end{aligned}$$

Let $p = 7$. Reducing the cofactors, $\check{h}_1$ and $\check{h}_2$, modulo p, we have

$$\begin{aligned} \phi_p(\check{h}_1) &= zx_1^2 + 2x_2x_1, \\ \phi_p(\check{h}_2) &= zx_1^2, \text{ and} \\ g_p &= \gcd(\phi_p(\check{h}_1), \phi_p(\check{h}_2)) = x_1 \neq 0. \end{aligned}$$

Since $\deg(g_p, x_1) > 0$, the prime $p = 7$ is unlucky. In particular, from Theorem 5.2.3, we have $\operatorname{res}(\phi_p(\check{h}_1), \phi_p(\check{h}_2), x_1) = 0 \in \mathbb{Z}_p[x_2]$. Moreover, a direct computation gives

$$\operatorname{res}(\check{h}_1, \check{h}_2, x_1) = (2^2)(7^2)(x_2 - 3)(12x_2^2 + x_2 - 3) \in \mathbb{Z}[x_2][z],$$

hence $\phi_7(\operatorname{res}(\check{h}_1, \check{h}_2, x_1)) = 0$. Equivalently, $p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_1)$. Moreover,

$$\operatorname{res}(\check{h}_1, \check{h}_2, x_2) = -2x_1(6x_1^2z - 7x_1 - 147z) \in \mathbb{Z}[x_1].$$

If we choose $p = 2$, then $\phi_2(\operatorname{res}(\check{h}_1, \check{h}_2, x_2)) = 0$. Therefore, $p = 7$ and $p = 2$ are unlucky primes.

With (6.30) and the above example, we state the following theorem.

Theorem 6.3.19. *Let $f_1, f_2 \in \bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k]$ be two non-zero polynomials with cofactors h_1 and h_2, respectively. Then*

$$\text{Prob}[p \text{ is an unlucky prime}] \leq \sum_{i=1}^{k} \text{Prob}[p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i)].$$

Proof. Let $g_p = \gcd(\phi_p(\check{h}_1), \phi_p(\check{h}_2))$. By Definition 4.4.5, if p is an unlucky prime, then $\deg(g_p) > 0$, i.e., $g_p \neq 1$. This implies there exists some variable x_i s.t. $\deg(g_p, x_i) > 0$. Treat $\phi_p(\check{h}_1)$ and $\phi_p(\check{h}_2)$ as univariate polynomials in x_i over the ring $\bar{L}_{\mathbb{Z}}[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k]$. Then, by Theorem 5.2.3,

$$g_p \neq 1 \iff \operatorname{res}(\phi_p(\check{h}_1), \phi_p(\check{h}_2), x_i) = 0.$$

By definition of an unlucky prime, $p \nmid \operatorname{lc}(f_1, x_1) \cdot \operatorname{lc}(f_2, x_1)$, so $\phi_p(\check{h}_1) \neq 0$, and $\phi_p(\check{h}_2) \neq 0$. Thus, Corollary 5.2 implies that for some $1 \leq i \leq k$,

$$\begin{aligned} \operatorname{res}(\phi_p(\check{h}_1), \phi_p(\check{h}_2), x_i) = 0 &\iff \phi_p(\operatorname{res}(\check{h}_1, \check{h}_2, x_i)) = 0 \\ &\iff p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i). \end{aligned} \tag{6.31}$$

Since (6.31) holds for some x_i, we conclude that if p is an unlucky prime, then $p \mid \prod_{j=1}^{k} \operatorname{res}(\check{h}_1, \check{h}_2, x_j)$. Therefore,

$$\begin{aligned} \text{Prob}[p \text{ is an unlucky prime}] &\leq \text{Prob}[p \mid \prod_{i=1}^{k} \operatorname{res}(\check{h}_1, \check{h}_2, x_i)] \\ &\leq \sum_{i=1}^{k} \text{Prob}[p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i)]. \end{aligned} \tag{6.32}$$

□

Corollary 6.6. *The number of unlucky primes is finite.*

Proof. Direct consequence of Theorem 6.3.19. □

Similar to the previous section, to bound $\text{Prob}[p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i)]$ (from Theorem 6.3.19), we first use Hadamard's bound to bound $\|\operatorname{res}(\check{h}_1, \check{h}_2, x_i)\|_\infty$. However, to do so, we need a bound on $\|\check{h}_1\|_\infty$ and $\|\check{h}_2\|_\infty$. It is possible that $\|\check{h}_1\|_\infty > \|f_1\|_\infty$ and $\|\check{h}_2\|_\infty > \|f_2\|_\infty$ (see Example 2.3.3). Hence, to bound the probability of encountering an unlucky prime using Theorem 6.3.19, we use $\|\check{h}_1\|_\infty$ and $\|\check{h}_2\|_\infty$ instead of $\|f_1\|_\infty$ and $\|f_2\|_\infty$.

Theorem 6.3.20. *Let f_1 and $f_2 \in \bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k]$ be two non-zero polynomials with cofactors $\check{h}_1$ and $\check{h}_2$, respectively. Define*

- $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$ *and*
- $H = \max(\|\check{h}_1\|_\infty, \|\check{h}_2\|_\infty)$.

Let $p \in \mathbb{P}_b$. Then

$$\text{Prob}[p \textit{ is an unlucky prime}] \leq k \frac{\lfloor (k-1)(1 + \log_2(D) + \log_2(D+1) + 2D \log_2 H) \rfloor}{(b-1) \mid \mathbb{P}_b \mid}.$$

Proof. From Theorem 6.3.19,

$$\text{Prob}[p \text{ is an unlucky prime}] \leq \sum_{i=1}^{k} \text{Prob}[p \mid \operatorname{res}(\check{h}_1, \check{h}_2, x_i)].$$

We first bound Prob[$p \mid \text{res}(\check{h}_1, \check{h}_2, x_i)$]. For each $1 \le i \le k$, we have $\deg(\check{h}_1, x_i) \le \deg(f_1, x_i)$ and $\deg(\check{h}_2, x_i) \le \deg(f_2, x_i)$. Thus, from Proposition 6.3.1 part (iv),

$$\|\text{res}(\check{h}_1, \check{h}_2, x_i)\|_\infty \le (2D(D+1))^{k-1} H^{2D}$$

for each $1 \le i \le k$. Since $p \in \mathbb{P}_b$, we have $\log_2 p > b-1$. Therefore, for $1 \le i \le k$,

$$\begin{aligned}
\text{Prob}[p \mid \text{res}(\check{h}_1, \check{h}_2, x_i)] &\le \text{Prob}[p \mid \|\text{res}(\check{h}_1, \check{h}_2, x_i)\|_\infty] \\
&\le \frac{\lfloor \frac{\log_2 (2D(D+1))^{k-1} H^{2D}}{\log_2 p} \rfloor}{\mid \mathbb{P}_b \mid} \\
&\le \frac{\lfloor (k-1)(1 + \log_2(D) + \log_2(D+1) + 2D \log_2(H)) \rfloor}{(b-1) \mid \mathbb{P}_b \mid}.
\end{aligned}$$

Summing over all k variables gives the final bound:

$$\text{Prob}[p \text{ is unlucky}] \le k \frac{\lfloor (k-1)(1 + \log_2(D) + \log_2(D+1) + 2D \log_2(H)) \rfloor}{(b-1) \mid \mathbb{P}_b \mid}.$$

□

Example 6.3.13. *Let* $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - \frac{2}{5}, z_2^2 - \frac{7}{3} \rangle$ *and* $f_1, f_2 \in L[x_1, x_2]$ *s.t.*

$$\begin{aligned}
f_1 &= \underbrace{(87 x_1^5 x_2^5 + 97 z_2 z_1 x_1 - 56 z_2)}_{\check{h}_1} (z_2 + z_1 x_1 x_2), \\
f_2 &= \underbrace{(17 z_2 x_1^4 x_2^3 + 4 z_2 z_1 x_1 x_2 - 82 z_1)}_{\check{h}_2} (z_2 + z_1 x_1 x_2)
\end{aligned}$$

be as in Example 6.3.1. Let $\check{h}_1$ *and* $\check{h}_2$ *be the cofactors of* $\check{f}_1$ *and* $\check{f}_2$*, respectively. Then*

$$H = \max(\|\check{h}_1\|_\infty, \|\check{h}_2\|_\infty) = 97.$$

From Theorem 6.3.20, the probability of encountering an unlucky prime in MGCDNF is bounded by

$$\frac{\lfloor (k-1)(1 + \log_2(D) + \log_2(D+1) + 2D \log_2 H) \rfloor}{(b-1) \mid \mathbb{P}_b \mid} \approx 5.6 \times 10^{-8}.$$

Now, let us use the actual values instead of the bounds. Computing $r_{x_1} = \text{res}(h_1, h_2, x_1)$ *and* $r_{x_2} = \text{res}(h_1, h_2, x_2)$*, we have*

$$\begin{aligned}
\|r_{x_1}\|_\infty &= 2^4 \times 3^{10} \times 5 \times 7^{16} \times 17 \times 41^4 \times 97^4 \\
\|r_{x_2}\|_\infty &= 3^{10} \times 7^{12} \times 17^5 \times 97^3.
\end{aligned}$$

As you can see, neither $\|r_{x_1}\|_\infty$ *nor* $\|r_{x_2}\|_\infty$ *has a* $31-$*bit divisor. Thus, if we choose primes from* $\mathbb{P}_{31}$*, the probability of encountering an unlucky prime is zero.*

6.3.5 Unlucky evaluation points

Let β_k be chosen randomly from $[0, p)$. In this section, we bound the probability that $x_k = \beta_k$ is an unlucky evaluation point. Then we use it to bound the probability that PGCDNF encounters an unlucky evaluation point. In this section, we assume the following:

- $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ with $\deg(f_1, x_k) \geq \deg(f_2, x_k) \geq 0$.
- The monic $g = \gcd(f_1, f_2)$ exists.
- Set $X_k = [x_1, \ldots, x_{k-1}]$, so $\operatorname{lc}(f_1, X_k), \operatorname{lc}(f_2, X_k) \in \bar{L}_p[x_k]$.
- $\operatorname{lc}(f_1, X_k)(\beta_k) \neq 0$ and $\operatorname{lc}(f_1, X_k)(\beta_k) \neq 0$, and $\gcd(f_1(x_k = \beta_k), f_2(x_k = \beta_k))$ exists.
- h_1 and h_2 denote the cofactors of f_1 and f_2.
- $g_{\beta_k} = \gcd(h_1(x_k = \beta_k), h_2(x_k = \beta_k))$.

From Definition 4.4.3, $x_k = \beta_k$ is an unlucky evaluation point if $\deg(g_{\beta_k}) > 0$. Similar to unlucky primes in the previous section, we have

$$\begin{aligned} g_{\beta_k} \neq 1 &\iff \deg(g_{\beta_k}, x_i) > 0 \text{ for some } \ 1 \leq i \leq k-1 \\ &\iff \underbrace{\operatorname{res}(h_1(x_k = \beta_k), h_2(x_k = \beta_k), x_i) = 0}_{\text{Theorem 5.2.3}} \text{ for some } \ 1 \leq i \leq k-1. \end{aligned} \tag{6.33}$$

Example 6.3.14 illustrates the above argument.

Example 6.3.14. *Let $\bar{L}_p = \mathbb{Z}_5[z]/\langle z^2 - 2\rangle$ and let f_1 and f_2 be two polynomials in $\bar{L}_p[x_1, x_2]$ s.t.*

$$\begin{aligned} f_1 &= (x_1 + z)\underbrace{(zx_1^2 + 3x_1x_2 + 2z(x_2 - 3))}_{h_1}, \\ f_2 &= (x_1 + z)\underbrace{(zx_1^2 + 3x_1x_2)}_{h_2}. \end{aligned}$$

Evaluating the cofactors h_1 and h_2 at $x_2 = 3$ yields

$$\begin{aligned} \phi_{x_2=3}(h_1) &= zx_1^2 + 4x_1, \\ \phi_{x_2=3}(h_2) &= zx_1^2 + 4x_1, \ \textit{and} \\ g_\beta = \gcd(\phi_{x_2=3}(h_1), \phi_{x_2=3}(h_2)) &= zx_1^2 + 4x_1 \neq 1 \end{aligned}$$

in $\bar{L}_p[x_1]$. Since $\deg(g_\beta, x_1) > 0$, the evaluation point $x_2 = 3$ is unlucky. From Theorem 5.2.3, since $g_\beta \neq 1$, we have

$$\operatorname{res}(\phi_{x_2=3}(h_1), \phi_{x_2=3}(h_2), x_1) = 0.$$

Furthermore, $\beta_2 = 3$ is a root of the polynomial

$$\operatorname{res}(h_1, h_2, x_1) = x_2^2 + 4x_2 + 4 \in \bar{L}_p[x_2].$$

The following theorem generalizes this: if $x_2 = \beta_2$ is an unlucky evaluation point, then β_2 is a root of $\operatorname{res}(h_1, h_2, x_1)$.

Theorem 6.3.21. *Let f_1 and $f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ be two non-zero polynomials and $g = \gcd(f_1, f_2)$. Let h_1 and h_2 be the cofactors of f_1 and f_2, respectively. If $x_k = \beta_k \in [0, p)$ is an unlucky evaluation point, then it is a root of $r = \prod_{i=1}^{k-1} \operatorname{res}(h_1, h_2, x_i)$.*

Proof. Let

$$g_{\beta_k} = \gcd(h_1(x_k = \beta_k), h_2(x_k = \beta_k)).$$

If $\beta_k \in [0, p)$ is an unlucky evaluation point, then $\deg(g_{\beta_k}) > 0$, i.e., $g_{\beta_k} \neq 1$. Hence, there exists some $1 \leq i \leq k-1$ s.t. $\deg(g_{\beta_k}, x_i) > 0$. Treat $h_1(x_k = \beta_k)$ and $h_2(x_k = \beta_k)$ as univariate polynomials in x_i with coefficients in $\bar{L}_p[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{k-1}]$. Since $g_{\beta_k} \neq 1$, Theorem 5.2.3 implies

$$\operatorname{res}(h_1(x_k = \beta_k), h_2(x_k = \beta_k), x_i) = 0.$$

Let $X_k = [x_1, \ldots, x_{k-1}]$; then by the definition of unlucky evaluation points, $\operatorname{lc}(f_1, X_k)(\beta_k) \neq 0$ and $\operatorname{lc}(f_2, X_k)(\beta_k) \neq 0$ imply that $\operatorname{lc}(h_1, X_k)(x_k = \beta_k) \neq 0$ and $\operatorname{lc}(h_2, X_k)(x_k = \beta_k) \neq 0$. Thus, Theorem 5.2.2 implies

$$\operatorname{res}(h_1, h_2, x_i)(x_k = \beta_k) = 0.$$

Therefore, $x_k = \beta_k$ is a root of $r = \prod_{i=1}^{k-1} \operatorname{res}(h_1, h_2, x_i)$. □

Let $r = \prod_{i=1}^{k-1} \operatorname{res}(h_1, h_2, x_i)$. From Theorem 6.3.21,

$$\text{Prob}[x_k = \beta_k \text{ is an unlucky evaluation point}] \leq \text{Prob}[r(\beta_k) = 0].$$

To bound this probability, we use the Schwartz-Zippel Lemma. To do so, we first need a bound for $\deg(r)$. In the following theorem, we bound the above equation.

Theorem 6.3.22. *Let f_1 and $f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ with $\deg(f_1, x_k) \geq \deg(f_2, x_k) \geq 0$. Assume the following.*

(i) Let $\beta_k \in [0, p)$.

(ii) Let the monic $g = \gcd(f_1, f_2)$ exist, and let h_1 and h_2 be the cofactors of f_1 and f_2, respectively.

(iii) Define $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$.

Then, at the first iteration of the PGCDNF, we have

$$\text{Prob}[x_k = \beta_k \text{ is an unlucky evaluation point}] \leq \frac{2(k-1)^2 D^2}{p - 2D}$$

Proof. Let $R_i = \operatorname{res}(h_1, h_2, x_i) \in \mathbb{Z}_p[z][x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{k-1}][x_k]$ and $r = \prod_{i=1}^{k-1} R_i$. By Theorem 6.3.21, if $x_k = \beta_k$ is an unlucky evaluation point, then $r(x_k = \beta_k) = 0$. To compute

$$\text{Prob}[r(x_k = \beta_k) = 0],$$

we use the Schwartz-Zippel lemma. Let $F_1 = \operatorname{lc}(f_1, \{x_1, \ldots, x_{k-1}\})$ and $F_2 = \operatorname{lc}(f_2, \{x_1, \ldots, x_{k-1}\})$ be two polynomials in $\mathbb{Z}_p[z][x_k]$. By the definition of unlucky evaluation points (Definition 4.4.3), $F_1(x_k = \beta_k) \neq 0$ and $F_2(x_k = \beta_k) \neq 0$. Each F_1 and F_2 has at most D roots in $\mathbb{Z}_p$. Excluding the roots

of F_1 and F_2 from the integers in $[0, p)$, there are at least $p - 2D$ possible choices for β. Therefore, by the Schwartz-Zippel lemma, we obtain

$$\begin{aligned}\text{Prob}[x_k = \beta_k \text{ is an unlucky evaluation point}] &\leq \text{Prob}[r(\beta_k) = 0] \\ &\leq \frac{\deg(r)}{p - 2D}.\end{aligned}$$

Now, we bound $\deg(r)$. For $1 \leq i \leq k - 1$, we have

$$\begin{aligned}\deg(h_1, x_i) &\leq \deg(f_1, x_i) \leq D \text{ and} \\ \deg(h_2, x_i) &\leq \deg(f_2, x_i) \leq D.\end{aligned}$$

Thus, by the construction of the Sylvester matrix, for $j \neq i$

$$\deg(R_i, x_j) \leq (\deg(h_1, x_i) + \deg(h_2, x_i))D \leq D(D + D) = 2D^2,$$

so $\deg(R_i) \leq 2(k - 1)D^2$, which implies that $\deg(r) \leq 2(k - 1)^2 D^2$, and therefore

$$\text{Prob}[x_k = \beta_k \text{ is an unlucky evaluation point}] \leq \frac{2(k - 1)^2 D^2}{p - 2D}.$$

□

Now, we can bound the probability that the PGCDNF encounters an unlucky evaluation point in all its recursive calls.

Theorem 6.3.23. *Let $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ and $D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$. Then*

$$\text{Prob}[\textit{PGCDNF encounters an unlucky evaluation point}] \leq \frac{2(D + 1)^{k+1}}{p - 2D}.$$

Proof. Let h_1 and h_2 be the semi-associates of the cofactors of f_1 and f_2, respectively. For the evaluation process, the worst case occurs when $h_1 = f_1$ and $h_2 = f_2$. In this case, to interpolate x_k at $g = \gcd(f_1, f_2)$, we need at most $D + 1$ evaluation points. For each of the $D + 1$ evaluation points of x_k, we need $D + 1$ evaluation points to interpolate x_{k-1} in g, and so forth. In the recursive calls of the PGCDNF, $\deg(R_i, x_j) \leq 2D^2$ does not change for $1 \leq i, j \leq k$. However, the number of variables, and thus the number of R_i's at each level, changes. Table 6.4 summarizes the number of required evaluation points and the number of possible R_i's at each level of the PGCDNF. In Table 6.4, the polynomials F_{1_k} and F_{2_k} denote the evaluated polynomials in Line 13 of PGCDNF. From the number

Level	F_{1_k}, F_{2_k}	Evaluate	# Required evaluation points to interpolate x_i in g	# R_i
1	$\bar{L}_p[x_1, \ldots, x_{k-1}]$	x_k	$D+1$	$k-1$
2	$\bar{L}_p[x_1, \ldots, x_{k-2}]$	x_{k-1}	$(D+1)^2 = \underbrace{(D+1)}_{\text{for } x_k} \cdot \underbrace{(D+1)}_{\text{for } x_{k-1}}$	$k-2$
3	$\bar{L}_p[x_1, \ldots, x_{k-3}]$	x_{k-2}	$(D+1)^3 = \underbrace{(D+1)}_{\text{for } x_k} \cdot \underbrace{(D+1)}_{\text{for } x_{k-1}} \cdot \underbrace{(D+1)}_{\text{for } x_{k-2}}$	$k-3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k-1$	$\bar{L}_p[x_1, x_2]$	x_2	$(D+1)^{k-1}$	1
k	$\bar{L}_p[x_1]$	–	0	0

Table 6.4: Recursive calls of the PGCDNF.

of resultants in the fourth column of Table 6.4, similar to Theorem 6.3.22, we can prove that

$$\begin{aligned}
\text{Prob}[x_k = \beta_k \text{ is an unlucky evaluation point}] &\leq \frac{2D^2(k-1)^2}{p-2D}, \text{ we need } \leq D+1 \ \beta_k\text{'s} \\
\text{Prob}[x_{k-1} = \beta_{k-1} \text{ is an unlucky evaluation point}] &\leq \frac{2D^2(k-2)^2}{p-2D}, \text{ we need } \leq (D+1)^2 \ \beta_{k-1}\text{'s} \\
\text{Prob}[x_{k-2} = \beta_{k-2} \text{ is an unlucky evaluation point}] &\leq \frac{2D^2(k-3)^2}{p-2D}, \text{ we need } \leq (D+1)^3 \ \beta_{k-2}\text{'s} \\
&\vdots \\
\text{Prob}[x_2 = \beta_2 \text{ is an unlucky evaluation point}] &\leq \frac{2D^2}{p-2D}, \text{ we need } \leq (D+1)^{k-1} \ \beta_2\text{'s}.
\end{aligned}$$

From the above argument and Table 6.4, we have

$$\begin{aligned}
&\text{Prob[PGCDNF encounters an unlucky evaluation point]} \\
&\leq \text{Prob}[x_i = \beta_i \text{ is an unlucky evaluation point for some } 2 \leq i \leq k] \\
&\leq \sum_{i=2}^{k} \underbrace{(D+1)^{k-i+1}}_{\#\text{ Evaluation points}} \text{Prob}[x_i = \beta_i \text{ is an unlucky evaluation point}] \\
&\leq \sum_{i=2}^{k} (D+1)^{k-i+1} \frac{2D^2(i-1)^2}{p-2D} \\
&\leq \frac{2(D+1)\Big((D+2)((D+1)^k - 1) - Dk(Dk+2)\Big)}{D\,(p-2D)}.
\end{aligned}$$

□

Example 6.3.15. *Consider the polynomials f_1 and f_2 defined in 6.3.1. From Theorem 6.3.21, the probability of encountering an unlucky evaluation point in PGCDNF is bounded by*

$$\frac{2(D+1)\Big((D+2)((D+1)^k - 1) - Dk(Dk+2)\Big)}{D\,(p-2D)} \approx 4.7 \times 10^{-7}.$$

Now, we use the actual values instead of bounds. Computing $r = \text{res}(h_1, h_2, x_1)$*, we have* $\deg(r, y) = 20$*. Thus,*

$$\text{Prob}[x_k = \beta_k \textit{ is an unlucky evaluation point}] \leq \frac{2 \times 20}{p - (\deg(f_1, y) + \deg(f_2, y))} = 3.72 \times 10^{-8}.$$

We evaluate y *at most for* $D + 1$ *evaluation points, so*

$$\text{Prob}[\textit{PGCDNF encounters an unlucky evaluation point}] \leq (D + 1) \times 3.72 \times 10^{-8} \approx 2.6 \times 10^{-7}.$$

6.4 Failure probability analysis of MRESNF

Since lc-bad pairs and det-bad pairs are defined identically for MGCDNF and MRESNF, the probability bounds proved for MGCDNF apply to MRESNF without change. In MRESNF, we do not introduce unlucky primes or unlucky evaluation points because we do not use Lemma 2.2.9 in MRESNF. Consequently, the only additional category to analyze is the occurrence of a zero divisor in the finite ring $\bar{L}_p[y]$ during the univariate resultant computation.

Let $f_1, f_2 \in \bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k][y]$. Let p be a prime and let $\beta \in [0, p)^k$ be an evaluation point s.t. (p, β) is not lc-bad. By Definition 5.4.1, (p, β) is a zero divisor pair if URES (Algorithm 9) fails when applied to the inputs $\phi_p(f_1)(\beta)$ and $\phi_p(f_2)(\beta) \in \bar{L}_p[y]$ in Line 2 of PRESNF. To compute the resultant of univariate polynomials in $\bar{L}_p[y]$, URES employs the MEA. Thus, URES fails if and only if the MEA fails (Theorem 5.3.3). Consequently, the results obtained in Section 6.3.3 also apply to URES, and the bound obtained in Corollary 6.5 for the probability that the MEA fails over $\bar{L}_p[y]$ also yields a bound on the probability that URES encounters a zero divisor pair. Notice that in PRESNF, we assumed that the input polynomials are in $\bar{L}_{\mathbb{Z}}[x_1, \ldots, x_k][y]$ with $k + 1$ variables, so Theorem 6.3.16 holds for PRESNF if we replace $2k - 2$ with $2k$, to obtain

$$B_l = p^{2D^2}(2D(D + 1))^{2k} H^{2D}.$$

We bound the probability that URES fails over $\bar{L}_p[y]$ using the same justification as in Theorem 6.3.17:

$$\begin{aligned} \text{Prob}[\text{URES fails in } \bar{L}_p[y]] &= \text{Prob}[\text{MEA fails in } \bar{L}_p[y]] \\ &\leq \underbrace{(D + 1)\frac{\lfloor (d(2D + 1)/2)\,(\log(dB_l^2 + 2dDB_M^2)) \rfloor}{(b - 1) \mid \mathbb{P}_b \mid}}_{B_{res}}. \end{aligned}$$

We call the above bound B_{res}. Moreover, to interpolate the resultant, we need at most $(2D^2 + 1)$ evaluation points for k variables. Therefore,

$$\text{Prob}[\text{PRESNF encounters a zero divisor pair}] \leq (2D^2 + 1)^k B_{res}. \tag{6.34}$$

6.5 Failure probability summary

We analyzed the probability that MGCDNF, PGCDNF, MRESNF, and PRESNF encounter a bad prime or evaluation point. In the following theorems, we summerizes this analyzes.

Theorem 6.5.1. *Let $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ with monic $g = \gcd(f_1, f_2)$. Let N denote the number of primes used, and assume that the number of evaluation points used in PGCDNF is at most $(D+1)^{k-1}$ (Theorem 4.5.1). Let*

- *p be chosen at random from $\mathbb{P}_b$ and β be chosen at random from $[0, p)^{k-1}$,*
- *$D = \max_{i=1}^{k}(\deg(f_1, x_i), \deg(f_2, x_i))$,*
- *$D_i = \frac{d}{\prod_{j=1}^{i} d_{n-j+1}} = \prod_{j=1}^{n-i} d_j$, where $d_i = \deg(M_i)$,*
- *$\|\check{M}_i\| \leq 2^m$ and $l_i = \mathrm{lc}(\check{M}_i, z_i)$ for $1 \leq i \leq n$,*
- *$\|\gamma^d\|_\infty \leq 2^C$, where $\gamma = z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$ s.t. $C_1, \ldots, C_{n-1} \in (0, p)$ are chosen randomly*
- *$\|f_i\|_\infty \leq 2^h$ for $i = 1, 2$,*
- *$H = \max(\|\check{h}_1\|_\infty, \|\check{h}_2\|_\infty)$, where $h_1 = f_1/g$ and $h_2 = f_2/g$,*
- *$\delta_1 = d - d_n + 1$, and $\delta_i = d - d_{n-i+1} + 1 + (d_{n-i+1} - 1)\sum_{j=1}^{i-1} \delta_j$ for $2 \leq i \leq n$, and*
- *B be as defined in Theorem (6.3.18).*

The probability that MGCDNF encounters either an lc-bad pair, a det-bad pair, or an unlucky prime, or that PGCDNF encounters an unlucky evaluation point or a zero divisor pair, is bounded by

$$N\Big((D+1)^{k-1}\Big(\underbrace{\frac{2h}{(b-1)\cdot \mid \mathbb{P}_b \mid} + \frac{2(k-1)D}{p} + \frac{nm}{(b-1) \mid \mathbb{P}_b \mid}}_{\text{Prob}[(p,\beta) \text{ is lc-bad}] \text{ (Theorem 6.3.1)}} \tag{6.35}$$

$$+ \underbrace{3k(D+1)^{k-1}B}_{\text{Prob}[PGCDNF \text{ encounters a zero divisor pair}] \text{ (Theorem 6.3.18)}} \tag{6.36}$$

$$+ \underbrace{\frac{2(D+1)\Big((D+2)\big((D+1)^k - 1\big) - Dk(Dk+2)\Big)}{D\,(p-2D)}}_{\text{Prob}[PGCDNF \text{ encounters an unlucky evaluation point}] \text{ (Theorem 6.3.22)}} \tag{6.37}$$

$$+ \underbrace{\frac{d(d-1)}{2(p-1)} + \frac{\lfloor d/2 \log_2 d + d\,(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m))\rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}}_{\text{Prob}[(p,c) \text{ is det-bad}] \text{ (Theorem 6.3.10)}} \tag{6.38}$$

$$+ \underbrace{k\frac{\lfloor (k-1)\big(1 + \log_2(D) + \log_2(D+1) + 2D\log_2 H\big)\rfloor}{(b-1) \mid \mathbb{P}_b \mid}}_{\text{Prob}[p \text{ is an unlucky prime}] \text{ (Theorem 6.3.20)}}\Big). \tag{6.39}$$

Since MGCDNF is output-sensitive, its failure probability depends on both the inputs and the output. Therefore, the algorithm benefits from a speedup when the output is much smaller than the inputs.

Theorem 6.5.2. *Let $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k][y]$ with $r = \mathrm{res}(f_1, f_2, y)$. let N be the number of primes used in MRESNF, and let the number of evaluation points used in PRESNF be at most $(2D^2+1)^k$ (Theorem 5.5.1). Let*

- *p be chosen at random from $\mathbb{P}_b$ and β be chosen at random from $[0, p)^k$,*
- *$D = \max_{i=1}^{k}(\deg(f_1, y), \deg(f_2, y), \deg(f_1, x_i), \deg(f_2, x_i))$,*

- $D_i = \frac{d}{\prod_{j=1}^{i} d_{n-j+1}} = \prod_{j=1}^{n-i} d_j$, *where* $d_i = \deg(M_i)$,
- $\|\check{M}_i\| \leq 2^m$ *and* $l_i = \mathrm{lc}(\check{M}_i, z_i)$ *for* $1 \leq i \leq n$,
- $\|\gamma^d\|_\infty \leq 2^C$, *where* $\gamma = z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$ *s.t.* $C_1, \ldots, C_{n-1} \in (0, p)$ *are chosen randomly*
- $\|f_i\|_\infty \leq 2^h$ *for* $i = 1, 2$,
- $\delta_1 = d - d_n + 1$, *and* $\delta_i = d - d_{n-i+1} + 1 + (d_{n-i+1} - 1) \sum_{j=1}^{i-1} \delta_j$ *for* $2 \leq i \leq n$, *and*
- B_{res} *be as defined in Equation (6.34).*

Then the probability that MRESNF encounters either an lc-bad pair or a det-bad pair, or that PRESNF encounters a zero divisor pair is bounded by

$$N\Big((2D^2+1)^k\Big(\underbrace{\frac{2h}{(b-1)\cdot \mid \mathbb{P}_b \mid} + \frac{2(k-1)D}{p} + \frac{nm}{(b-1) \mid \mathbb{P}_b \mid}}_{\text{Prob}[(p,\beta) \text{ is lc-bad}] \text{ (Theorem 6.3.1)}} \tag{6.40}$$

$$+ \underbrace{(D+1)B_{res}}_{\text{Prob}[\text{PRESNF encounters a zero divisor pair}] \text{ (Equation 6.34)}} \Big) \tag{6.41}$$

$$+ \underbrace{\frac{d(d-1)}{2(p-1)} + \frac{\lfloor d/2 \log_2 d + d\,(C + \sum_{i=1}^{n} \delta_i \log_2(l_{n-i+1} + D_i m)) \rfloor}{(b-1)\times \mid \mathbb{P}_b \mid}}_{\text{Prob}[p \text{ is det-bad}] \text{ (Theorem 6.3.10)}} \Big) \tag{6.42}$$

The numerators in the equations (6.35)-(6.42) are polynomial in the sizes of the input and output, namely, d the degree of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, the size of the integer coefficients in $\check{M}_1, \check{M}_2, \ldots, \check{M}_n$, $\check{f}_1, \check{f}_2$, and the cofactors $\check{h}_1, \check{h}_2$, the degrees of f_1, f_2 in $x_1, x_2, \ldots, x_k$, the number of evaluation points used, the number of primes used, and the number of terms of f_1, f_2, g each of which is bounded by $(D+1)^k$.

Remark 6.4. *Since we assume that* $\#f_1, \#f_2, \#g \leq (1+D)^k$, *the bound above is polynomial in number of terms of the inputs and the output.*

Remark 6.5. *In practice, using large* $p \in \mathbb{P}_{31}$, $C_i \in (0, p)$ *and* $\beta_i \in [0, p)$, *we do not see any failures.*

Chapter 7

Implementation and demo of software

7.1 Summary of contributions

Section 7.2 describes the data structures employed to represent polynomials over algebraic number fields $L = \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. Section 7.3 presents the Maple implementation of the LAMinpoly algorithm (Algorithm 4) together with the primitive isomorphism, as outlined in Section 3.4.3. In Section 7.4, we implement Algorithm MGCD (Algorithm 8) on a range of examples to illustrate the different failure cases. The implementation of Algorithm URES (Algorithm 9) is presented in Section 7.5.

7.2 A data structure for $L[x_1, \ldots, x_k]$

We employ a recursive dense data structure, `POLYNOMIAL`, to represent polynomials in $L[x_1, \ldots, x_k]$. This structure follows the same design as the one introduced in [34]. We specify the Maple data type `<poly>` using **Backus–Naur Form (BNF)**[1] as follows:

```
<poly> ::= POLYNOMIAL(<ring>, <data>)
<ring> ::= [<char>,<vars>,<exts>]
<char> ::= <nonnegative integer>
<vars> ::= vector(<variables>)
<data> ::= <immediate integer>| <rational number> | vector(<data>)
<exts> ::= vector(<data>)
```

In the examples, we indicate vectors by square brackets corresponding to lists in Maple. They are essentially read-only lists. Through the following examples, we discuss the details of the `POLYNOMIAL` data structure.

Example 7.2.1. *We use the* `rpoly` *command to convert Maple's representation of a polynomial into the POLYNOMIAL data structure. For instance, the polynomial* $f = 2x^2 + 3xy^2 \in \mathbb{Q}[x, y]$ *is expressed as follows:*

[1] BNF is a formal notation for describing the syntax of a programming language. It is composed of a collection of production rules that define the valid syntax of a language. Each rule consists of a non-terminal symbol, which can be expanded into other symbols, along with one or more alternatives that specify the possible expansions of that symbol. The vertical bar | denotes an alternative, and ::= denotes a definition.

```
> f:=rpoly(2*x^2+3*x*y^2,[x,y]);
   f:=2x^2+3xy^2
```

We use the **lprint** *command to print the* POLYNOMIAL *data structure.*

```
> lprint(f);
          POLYNOMIAL([0, [x, y], [ ]],[0, [0, 0, 3], [2]])
```

The POLYNOMIAL data structure contains two primary components:

1. *The* ***first vector****,* `[0, [x, y], [ ]]`*, contains the ring's information. We decode the first vector as follows.*
   - *The first entry* `0` *indicates the characteristic of the ring.*
   - *The second entry* `[x,y]` *specifies the list of variables.*
   - *The third entry, an empty list* `[ ]`*, indicates that there are no algebraic extensions.*
2. *The* ***second vector****,* `[0, [0, 0, 3], [2]]`*, encodes the polynomial recursively. To do so, it uses the isomorphism*
$$\mathbb{Q}[x_1, \ldots, x_k] \cong \mathbb{Q}[x_k][x_{k-1}] \ldots [x_1].$$
   In this example, we treat x as the main variable and map $f \in \mathbb{Q}[x, y]$ to $f := 2x^2 + (3y^2)x \in \mathbb{Q}[y][x]$. Consequently, the entries `0, [0,0,3], [2]` *are the coefficients of $x^0, x,^1$, and x^2, respectively. Thus,*
   - *0 is the coefficient of x^0,*
   - *$0 + 0 \cdot y + 3y^2$ is the coefficient of x^1, and*
   - *2 is the coefficient of x^2.*

To retrieve the ring associated with a given polynomial, we use the **getring** command.

```
> getring(f);
        [0, [x, y], [ ]]
```

Example 7.2.2. *Let $L = \mathbb{Q}[z, w]/\langle M_1(z), M_2(w)\rangle$, where $M_1(z) = z^2 - 2$, and $M_2(w) = w^2 - 3$. Let*
$$f = 3xy^2 + w^2 + wz + 2x^2 \in L[x, y].$$

We use the **phirpoly** *command to reduce a polynomial modulo either an integer p or a list of minimal polynomials $m_1, \ldots, m_n$ or both. This allows us to construct the field $L \cong \mathbb{Q}(\sqrt{2}, \sqrt{3})$ and the polynomial f as follows. To obtain the ring information associated with a polynomial, we use the* **getring** *command. Additionally, to extract the internal representation of the polynomial itself, we use the* **getpoly** *command.*

```
> F:=rpoly(2*x^2+3*x*y^2+z*w+w^2,[x,y,w,z]);
   F := 3xy^2 + w^2 + wz + 2x^2
>M1:=rpoly(z^2-2,z);
   M1 := z^2 - 2
> M2:=phirpoly(rpoly(w^2-3,[w,z]),M1);
   M2 := (w^2 - 3)  mod <z^2 - 2>
> f:=phirpoly(F,M1,M2); #Create the polynomial f over Q[z]/<z^2-2,w^2-3>[x,y]
   f := 3xy^2 + wz + 2x^2 + 3 mod <w^2 - 3, z^2 - 2>
> getring(f); #Get the information of the ring L[x,y]
   [0, [x, y, w, z], [[[-3], 0, [1]], [-2, 0, 1]]]
> getpoly(f);  #Get the information of the polynomial F
    [[[[3], [0, 1]]], [0, 0, [[3]]], [[[2]]]]
```

Alternatively, we can construct the field L and the polynomial ring $L[x, y]$ more directly using the **rring** command as follows.

```
> L:=rring([z,w],[z^2-2,w^2-3]); #L=Q[z,w]/<z^2-2,w^2-3>
   L := [0, [z, w], [[[-2], 0, [1]], [-3, 0, 1]]]
> Lxy:=rring(L,[x,y]);  # Construct L[x,y] from L
   Lxy := [0, [x, y, z, w], [[[-2], 0, [1]], [-3, 0, 1]]]
> f:=rpoly(2*x^2+3*x*y^2+z*w+w^2,Lxy);
   f := 3xy^2 +  wz + 2x^2 +3 mod < z^2 - 2, w^2 - 3>
```

7.3 Implementation of Algorithm LAminpoly in Maple

In the following, we present our Maple code for Algorithm LAminpoly (Algorithm algorithm 4), along with several illustrative examples. The file `recden` contains all the necessary commands for computations involving the `POLYNOMIAL` data structure. Accordingly, it is called at the beginning of the process. Additionally, the file `MonOp` includes routines for computing the coordinate vector of γ^i with respect to a basis B for L_p.

Maple code 7.1: `LAminpoly.mw`

```
>restart;
with(PolynomialTools):
currentdir();
read "recden": read "MonOp":
>LAminpoly:=proc(m,p,gamma,X,Z)
     local n,deg,d,monombasis,g,A,B,det,q,i,b,M;
     n:=nops(m); #nops(m)=nops(X)
     deg:=[seq(degree(m[i],X[i]),i=1..n)];
     d:=mul(deg[i],i=1..n);
     monombasis:=Basemaker(X,deg,m);
     #monombasis is a basis for K=Z_p[z1,z2]/<z1^2-2,z2^2-3>i.e. {1,z1,z2,z1z2}
     g[0]:=rpoly(1,getring(gamma));
     for i to d do
        g[i]:=mulrpoly(gamma,g[i-1]);
     od;
     A:=Matrix(d);
     #Creating the coeffs matrix
     for i from 1 to d do
         g[i-1] := expand(rpoly(g[i-1]));
         A[1..d,i]:=Monomlist(g[i-1],monombasis,X);
     od;
     g[d]:= expand(rpoly(g[d]));
     b:=Monomlist(g[d],monombasis,X);
     # Construct the augmented matrix B = [A|I|b] and row reduce it
     B:=Matrix(d,2*d+1,datatype=integer[8]);
     B[1..d,1..d] := A;
     for i to d do B[i,d+i] := 1; od;
     for i to d do B[i,2*d+1] := b[i] od;
     #Check for appropriate C here: If det(A)<>0, then A is invertible and we are
         good.
     #Otherwise, MGCD goes back and chooses another C
     Modular:-RowReduce(p,B,d,2*d+1,2*d+1,'det',0,0,0,0,true);
     if det=0 then return FAIL fi;
     q := -B[1..d,2*d+1]; # Solve of Aq=-b for q
     B := B[1..d,d+1..2*d]; # A^(-1)
     #Constructing the characteristic polynomial
     M:=add(q[i]*Z^(i-1),i=1...d)+Z^d mod p;
    return M,A,B;
end:
```

Example 7.3.1. *Let $L = \mathbb{Z}/\langle z^2 - 2, w^2 - 3\rangle$. The procedure LAminpoly, as presented above, takes the following inputs*

- *a list of minimal polynomials* $m = [z^2 - 2, w^2 - 3]$,
- *a prime* $p = 1009$,
- *a candidate primitive element* $\gamma = 2z + w$,
- *a list of variables associated with the minimal polynomials* $X = [z, w]$, *and*
- *a variable* Z , *with respect to which the characteristic polynomial is to be constructed.*

It then returns the following outputs:

- *characteristic polynomial* M,
- *the matrix* A, *whose columns contain the coordinate vectors of* γ^i *for* $0 \leq i \leq 3$,

```
>currentdir():
>read "recden"; read "MonOp"; read "LAMinpoly";read "Phigamma";
>p:=nextprime(1008);
M1:=z^2-2:
M2:=w^2-3:
m:=[M1,M2]:
local gamma:
gamma:=rpoly(2*z+w,[w,z],[M1,M2]):
X:=[z,w]:
M,A,Ainv:=LAMinpoly(m,p,gamma,X,Z);
                                1009

M,A,Ainv :=  Z^4 + 987*Z^2 + 25,
    Matrix(4, 4, [[1, 0, 11, 0], [0, 1, 0, 27], [0, 2, 0, 34], [0, 0, 4, 0]]),
    Matrix(4, 4, [[1, 0, 0, 754], [0, 301, 859, 0], [0, 0, 0, 757],[0, 101, 454, 0]])
```

Example 7.3.2. *Let*

$$f = (14w + 70)x^2 + ((70w + 42)y + (z + 10)w + 5z + 6)x + ((5z + 6)w + 3z + 30)y \in L_p[x, y],$$

where $L_p = \mathbb{Z}_p/\langle z^2 + 1007, w^2 + 1006\rangle$. *The procedure* **Phi**, *described in Maple Code 7.2, converts* f *to its corresponding polynomial in* $\bar{L}_p = \mathbb{Z}_p/\langle Z^4 + 987Z^2 + 25\rangle$ *where* $M(Z) = Z^4 + 987Z^2 + 25$ *is obtained from LAminpoly algorithm. The file* **Phigamma** *contains the Maple implementation of the primitive isomorphism* ϕ_γ, *as described in Section 3.4.3.*

Maple code 7.2: `phigamma.mw`

```
>read "Phigamma";
>f:=rpoly((w+5)*(x+w*y)*(14*x+2*w+z),[x,y,w,z],[m1,m2],p):
Xmin:=[z,w]:
Xpoly:=[x,y]:
flag:=1:
Phi(f,M,Xmin,Xpoly,flag,L,A,Ainv);
((405*Z^3 + 178*Z + 70)*x^2 + ((7*Z^3 + 890*Z + 42)*y + 253*Z^3 + 757*Z^2 + 242*Z +
    760)*x + (959*Z^3 + 758*Z^2 + 347*Z + 773)*y) mod <Z^4 + 987*Z^2 + 25, 1009>
```

7.4 Implementation of Algorithm MGCD in Maple

Using the `POLYNOMIAL` data structure introduced in Section 7.2, we present the Maple implementation of our MGCD algorithm. As shown in the code, the MGCD algorithm prints the primes used. Furthermore, if a prime p is identified as unlucky, det-bad, or is involved as the first component in a zero divisor or lc-bad pair, (p, β), then MGCD prints p along with an explanation about it. In Examples 7.4.1, **??**, 7.4.2, and 7.4.3, the input polynomials f_1, f_2 belong to $L[x, y]$, where $L = \mathbb{Q}[z, w]/\langle z^2 - 2, w^2 - 3\rangle$.

Example 7.4.1. *In this example, MGCD hits one unlucky prime.*

```
>currentdir():
>read "recden"; read "MonOp"; read "LAMinpoly"; read "Phigamma"; read "PGCD"; read "
    MGCD";
>M1:=rpoly(z^2-2,z):
>M2:=phirpoly(rpoly(w^2-3,[w,z]),M1):
>f1:=phirpoly(rpoly((x+w+13*y)*(x*y+z+4)*(x+1),[x,y,w,z]),M1,M2);
f2:=phirpoly(rpoly((x+w)*(x+z)*(x+1),[x,y,w,z]),M1,M2);
output:=MGCD(f1,f2,9);
   f1 := y*x^3 + (13*y^2 + (w + 1)*y + z + 4)*x^2 + (13*y^2 + (13*z + 52 + w)*y + w*(
       z + 4) + z + 4)*x + (13*z + 52)*y + w*(z + 4)mod <w^2-3,z^2-2>
    f2 := x^3 + (w + z + 1)*x^2 + ((z + 1)*w + z)*x + z*w mod <w^2-3,z^2-2>
MGCD:prime=11
MGCD:prime=13
13 is an unlucky prime
MGCD:prime=17
MGCD:prime=19
output := (x + 1) mod <w^2 - 3, z^2 - 2>
```

The output of PGCD for $p_1 = 11$ is $C_{p_1} = x + 1$ while for $p_2 = 13$ PGCD returns $C_{p_2} = x^2 + (w + 1)x + w$. As $\mathrm{lm}(C_{p_2}) = x^2 > \mathrm{lm}(C_{p_1}) = x$, we conclude that $p_2 = 13$ is an unlucky evaluation point.

Example 7.4.2. *In this example, MGCD hits two unlucky primes.*

```
>currentdir():
>read "recden"; read "MonOp"; read "LAMinpoly"; read "Phigamma"; read "PGCD"; read "
    MGCD";
> M1:=rpoly(z^2-2,z):
> M2:=phirpoly(rpoly(w^2-3,[w,z]),M1):
> f1:=phirpoly(rpoly(254*(x+3)*(x+z+127*w)*(x+y)*x,[x,y,w,z]),M1,M2):
>f2:=phirpoly(rpoly((x+1)*(x+z)*(x+113*z+y)*x,[x,y,w,z]),M1,M2):
>output:=MGCD(f1,f2,112 );
MGCD:prime=113
MGCD:prime=127
MGCD:prime=131
p=131 and all the previous primes were unlucky
MGCD:prime=137
output := x mod <w^2 - 3, z^2 - 2>
```

The output of calling PGCD for $p_1 = nextprime(112) = 113$ is equal to $C_{p_1} = x^2 + xy$. When we call the second prime $p_2 = 127$, the output of PGCD is $C_{p_2} = x^2 + xz$. For the third prime, $p_3 = 131$, however, we have $C_{p_3} = x$. Since $\mathrm{lm}(C_{p_3}) < \mathrm{lm}(C_{p_2})$, we conclude that all the primes used before $p = 131$ were unlucky.

Example 7.4.3. *In the following example, MGCD hits one zero divisor and one lc-bad prime.*

```
>currentdir():
>read "recden"; read "MonOp"; read "LAMinpoly"; read "Phigamma"; read "PGCD"; read "
    MGCD";
> M1:=rpoly(z^2-2,z):
> M2:=phirpoly(rpoly(w^2-3,[w,z]),M1):
>f1:=phirpoly(rpoly(38*(w+9)*x^2+2*x+3*z+w,[x,y,w,z]),M1,M2):
> f2:=phirpoly(rpoly((x+7)*z*x^3*y+(z-2)*x+2*y*w,[x,y,w,z]),M1,M2):
> g:=phirpoly(rpoly(w*x+y+z,[x,y,w,z]),M1,M2):
> f1:=mulrpoly(f1,g);
> f2:=mulrpoly(f2,g);
> output:=MGCD(f1,f2,12);
   f1 := (342 w +114) x^3+((38 w +342) y +(38 z +2) w +342 z) x^2+(3 z w +2 y +2 z
       +3) x+(3 z +w) y +z w +6 mod <w^2-3,z^2-2>
   f_2 := zwyx^5+(zy^2+(7zw +2)y)x^4+(7zy^2+14 y)x^3+(z-2)wx^2+((z +4)y-2z+2)x +2wy
       ^2+2zwy mod <w^2-3,z^2-2>
MGCD:prime=13
p=13 is a ZD prime ZD=POLYNOMIAL([13, [z], [[4, 0, 1, 0, 1]]],[2, 4, 1])
MGCD:prime=17
MGCD:prime=19
p=19 is an lc-bad prime
MGCD:prime=23
p=23 is a ZD prime ZD=POLYNOMIAL([23, [z], [[3, 0, 6, 0, 1]]],[20, 1])
MGCD:prime=29
MGCD:prime=31
   output := (x + 1/3yw + 1/3zw) mod <w^2 - 3, z^2 - 2>
```

Since $w^2-3=(w+9)(w+4) \mod 13$*, the monic Euclidean Algorithm hits a zero divisor while trying to find the inverse of* $\mathrm{lc}(\check{f}_1) \mod 13 = 4w+10$.

Example 7.4.4. *In this example, we compute the* gcd *of two random polynomials over a random algebraic number field. We create a random algebraic number field as follows.*

```
> #We construct the field of L=Q[z,w]/ <M1(z),M2(w)>, where M1 and M2 are the minimal
> #polynomials of two algebraic numbers of degree d so [L,Q]=d^2.
> d:=3:
> do;
    m1:=z^d+randpoly(z,degree=d-1,dense): # Monic random polynomial s.t deg(M1)=3
    m2:=w^d+randpoly(w,degree=d-1,dense): #Monic random polynomial s.t deg(M2)=3
    alias(sigma=RootOf(M1)):
 until irreduc(M1) and irreduc(M2,sigma):
> #M1 and M2 must be irreducible over Q and Q[z]/<M_1(z)> respectively
> M1:=rpoly(m1,z);
> M2:=phirpoly(rpoly(m2, [w,z]),M1);
   M1:=z^3-21z^2+63z-78
   M2:=w^3-91w^2-46w +86 mod <z^3-21z^2+63z-78>
```

Now we build two random polynomials $f_1, f_2 \in L[x,y]$ *where* $L=\mathbb{Q}[z,w]/ < M_1(z), M_2(w) >$*, and* $\gcd(f_1,f_2) \neq 1$*. Then, we call MGCD to compute* $\gcd(f_1,f_2)$.

```
> g:=phirpoly(rpoly(randpoly([x,y,w,z],degree=3,terms=4),[x,y,w,z]),M1,M2):
> f1:=phirpoly(rpoly(randpoly([x,y,w,z],degree=2,terms=6),[x,y,w,z]),M1,M2):
```

```
> f2:=phirpoly(rpoly(randpoly([x,y,w,z],degree=2,terms=6),[x,y,w,z]),M1,M2):
> f1:=mulrpoly(f1,g); #f1*g
> f2:=mulrpoly(f2,g); #f2*g
> output:=MGCD(f1,f2,1009);
   f1 := ((-1350z + 2025)w - 2550z^2 + 3825z)yx^2 + ((702w + 1326z)y^3 + (1026w^2 +
       1938wz + 300z - 450)y^2 + ((-1350z^2 - 1674)w-53550z^2 + 157488z - 198900)y+
       (-57525z^2 + 185850z - 230100)w)x - 156y^4 - 228wy^3 + (1534wz^2 + 300z^2 +
       372)y^2 + 2242z^2w^2y  + (-1118758z^2 + 3672750z - 4832100)w mod <w^3 - 91w^2
       - 46w + 86, z^3 - 21z^2 + 63z - 78>

   f2 := ((-3300 z + 4950) y + 3850 z^2 - 5775 z) x^2 + (1716 y^3 + (2508 w - 4152 z
       + 3225) y^2 + (-2926 w z - 1350 z^2 - 2925 z - 4092) y + (3900 z - 5850) w +
       114975 z^2 - 348026 z + 436800) x  + 1118 y^4  + (1634 w - 1014 z) y^3 +
       ((-1482 z - 2028) w - 3060 z^2 - 2666) y^2 + (-1330 w z^2 - 2964 w^2 + 40950 z
       ^2 - 120432 z + 152100) y + (3900 z^2 + 4836) w + 663670 z^2 - 2178750 z +
       2866500 mod <w^3 - 91 w^2 - 46 w + 86, z^3 - 21 z^2 + 63 z - 78>
MGCD:prime=1013
gamma := z + 784 w;
M := z^9 + 661 z^8 + 303 z^7 + 269 z^6 + 462 z^5 + 699 z^4 + 638 z^3 + 985 z^2 + 430
    z + 144;
MGCD: prime = 1019;
gamma := z + 969 w;
M := z^9 + 340 z^8 + 234 z^7 + 553 z^6 + 919 z^5 + 421 z^4 + 301 z^3 + 391 z^2 + 685
    z + 779;
MGCD: prime = 1021;
gamma := z + 782 w;
M := z^9 + 861 z^8 + 994 z^7 + 853 z^6 + 234 z^5 + 405 z^4 + 759 z^3 + 650 z^2 + 442
    z + 318;
MGCD: prime = 1031;
gamma := z + 905 w;
M := z^9 + 312 z^8 + 568 z^7 + 851 z^6 + 800 z^5 + 877 z^4 + 705 z^3 + 890 z^2 + 531
    z + 400;
 output := x + (-104/5475 z^2 + 676/1825 z - 234/365) y^2+ (-152/5475 z^2 + 988/1825
     z - 342/365) w y+ 698/5475 z^2 - 2712/1825 z + 2118/365 mod <w^3 - 91 w^2 - 46 w
      + 86, z^3 - 21 z^2 + 63 z - 78>;
```

7.5 Implementation of Algorithm URES in Maple

Maple code 7.3: URES.mw

```
>currentdir():
>with(PolynomialTools):
>read(recden):
>monic:=proc(a) #This procedure makes polynomials monic
 if getpoly(a)=0 then a else mulrpoly(invrpoly(lcrpoly(a)),a); fi;
end proc:
#ures computes the resultant of two univariate polynomials using m.p.r.s.
ures:=proc(a, b)
local q,r,s,t,k,n,m,h,R,c,mprs,Ring,v,prod,i,l;
r[0]:=a, r[1]:=b;
m[0],mprs:=r[0], [a];
```

```
k:=1;
Ring:=getring(a);
while getpoly(r[k])=0 do
   m[k]:=monic(r[k]);
   mprs:=[op(mprs),r[k]];
   r[k+1]:=remrpoly(m[k-1],m[k]);
   k:=k+1;
od; #Now, we have m.p.r.s. and can compute the resultant
 R,v:=getring(mprs[1]),0;
 prod,n:=rpoly(1,R),nops(mprs);
 c:=[seq(0,i=1..n)];
 for i to  n-2 do
     v:=v+degrpoly(mprs[i])*degrpoly(mprs[i+1]);
 od;
 for i to n do
      c[i]:=lcrpoly(mprs[i]);
 od;
for i from 2 to n-1 do
        prod:=scarpoly(powrpoly(c[i],degrpoly(mprs[i-1])),prod);
od;
prod:=scarpoly(powrpoly(c[n],degrpoly(mprs[n-1])),scarpoly((-1)^v,prod));
return(prod);
end:
```

The code 7.3 presents the Maple code for the URES algorithm (Algorithm 9). In the following example, we compute the resultant of two univariate polynomials using the code 7.3.

Example 7.5.1. *In the following, we compute the resultant of two polynomials in* $\mathbb{Q}[z_1, z_2, z_3, z_4]/\langle z_1^2 - 2, z_2^2 - 3, z_3^2 - 5, z_4^2 - 7\rangle$ *using the URES algorithm.*

```
>M1:=z1^2-2:
 M2:=z2^2-3:
 M3:=z3^2-5:
 M4:=z4^2-7:
 a:=rpoly(x^3+z1*x+z2*z3+z4+1,[x,z4,z3,z2,z1],[M1,M2,M3,M4]);
 b:=rpoly(x^2+z1*z3*x-1,[x,z4,z3,z2,z1],[M1,M2,M3,M4]);
 r:=ures(a,b);
    a := (x^3+x*z1+z2*z3+z4+1) mod <z4^2-7, z3^2-5, z2^2-3, z1^2- 2>
    b := (x*z1*z3+x^2-1) mod <z4^2-7, z3^2-5, z2^2-3, z1^2-2>
    r:=(((2*z2-13*z1-2)*z3+2)*z4+(2*z2-13*z1-2)*z3+(-65*z1-10)*z2-12*z1+20)
    mod <z4^2-7, z3^2-5, z2^2-3, z1^2-2>
>R:=resrpoly(a,b,x): #check correctness by comparing r with R
>subrpoly(r,R);
0 mod <z4^2 - 7, z3^2 - 5, z2^2 - 3, z1^2 - 2>
```

Chapter 8

Conclusion and future work

8.1 Conclusion

We designed and implemented two new modular algorithms for computing the monic GCD (Algorithm MGCDNF) and the resultant (Algorithm MRESNF) of two polynomials in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$, where $n > 1, k \geq 1$ with high probability. Furthermore, we derived a formula for computing the resultant of univariate polynomials. MGCDNF and MRESNF both achieve speedups through the following strategies:

- As a preprocessing step, both algorithms replace the input polynomials f_1 and f_2 and the minimal polynomials $M_1, \ldots, M_n$ with their semi-associates $\check{f}_1$, $\check{f}_2$, and $\check{M}_1, \ldots, \check{M}_n$. This strategy improves the efficiency of modular reduction.
- To avoid expression swell, the coefficients of $\check{f}_1$ and $\check{f}_2$ are mapped to their images in $\mathbb{Z}_{p_i}$ for a sequence of primes p_i, using modular homomorphisms.
- To avoid computing over multiple field extensions $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, we converted the problem to a simple extension $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. This conversion is performed modulo a prime to avoid coefficient growth. Hence, the GCD and resultant computations are performed over the ring $\bar{L}_{p_i} = \mathbb{Z}_{p_i}[z]/\langle M(z) \rangle$ for a sequence of primes p_i.
- To reduce the multivariate problem to the univariate case, we applied the classical strategy of evaluation followed by dense interpolation. This enables the use of the monic Euclidean algorithm to compute the monic GCD and resultant in the univariate setting.

The MGCDNF and MRESNF algorithms apply Chinese remaindering and rational number reconstruction to recover the rational coefficients of the target GCD and resultant. However, not all primes and evaluation points lead to a successful reconstruction. In particular, most computations are performed over $\bar{L}_{p_i}$. If $M(z)$ is reducible modulo p_i, then $\bar{L}_{p_i} = \mathbb{Z}_{p_i}[z]/\langle M(z) \rangle$ is not a field. Consequently, our modular algorithms may encounter zero divisors. We classified all potential failure cases and provided a detailed analysis of their probabilities. We also presented an analysis of the expected time complexity of our algorithms. Both algorithms and their subalgorithms have been implemented in Maple, and we presented benchmark results demonstrating their performance.

8.2 Future work

Building on the results of this thesis, several applications and computational enhancements remain to be investigated.

- Let $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, where p is a prime and $M(z) \in \mathbb{Z}_p[z]$ has $\deg(M(z)) = d$. Performing arithmetic over $\bar{L}_p$, multiplication and division will be time-consuming, especially when $\deg(M(z)) = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}]$ is large (over 100). To speed up our algorithms, we can apply Fast Multiplication and Fast Division algorithms over $\bar{L}_{p_i}$.

- To improve the expected time complexity, we aim to replace the dense interpolation with a sparse interpolation method using Zippel's algorithm [41] or Hu and Monagan's algorithm [20].

- Both MRESNF and MGCDNF assume that the input polynomials are defined over an algebraic number field

$$L = \mathbb{Q}[z_1, \ldots, z_n]/\langle M_1(z_1), \ldots, M_n(z_n)\rangle,$$

  where each minimal polynomial $M_i(z_i)$ is irreducible over

$$\mathbb{Q}[z_1, \ldots, z_{i-1}]/\langle M_1(z_1), \ldots, M_{i-1}(z_{i-1})\rangle.$$

  The algorithms do not verify these irreducibility assumptions since doing so would require polynomial factorization, which is computationally expensive. We would like to extend MRESNF and MGCDNF to the case where some $M_i(z_i)$ are reducible; equivalently, the coefficient domain is no longer a field and may contain zero divisors.

  For instance, let $f_1 = x^2 + 1$ and $f_2 = (z-1)(x+1)$ be two polynomials over $\mathbb{Q}[z]/\langle z^2 - 1\rangle$, where $z^2 - 1 = (z-1)(z+1)$ is reducible over $\mathbb{Q}$. In this quotient ring, $z - 1$ is a zero divisor and hence not invertible. Since $\mathrm{lc}(f_2) = z - 1$, the Monic Euclidean Algorithm fails while trying to compute $\mathrm{monic}(f_2)$ modulo a prime p_1. Consequently, the MGCDNF (or MRESNF) algorithm changes p_1 and calls the Monic Euclidean Algorithm for a new prime, but $\mathrm{lc}(f_2)$ remains a zero divisor modulo any prime. We would like to extend our algorithms so that they also apply to polynomials defined over such quotient rings.

Bibliography

[1] Saban Alaca and Kenneth S. Williams. *Introductory Algebraic Number Theory*. Cambridge University Press, 2003.

[2] Mahsa Ansari and Michael Monagan. Computing GCDs of multivariate polynomials over algebraic number fields presented with multiple extensions. In *Computer Algebra in Scientific Computing*, volume 14139 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2023.

[3] Mahsa Ansari and Michael Monagan. A modular algorithm to compute the resultant of multivariate polynomials over algebraic number fields presented with multiple extensions. In *Computer Algebra in Scientific Computing*, volume 14938 of *Lecture Notes in Computer Science*, pages 27–46. Springer, 2024.

[4] Mahsa Ansari and Michael Monagan. A failure probability analysis of a modular algorithm to compute the monic GCD of multivariate polynomials over algebraic number fields. In *Computer Algebra in Scientific Computing*, volume 16235 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2025.

[5] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of STOC 1988*, pages 301–309. ACM, 1988.

[6] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18:478–504, 1971.

[7] W. S. Brown and Joseph F. Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM*, 18:505–514, 1971.

[8] Bruno Buchberger, George E. Collins, Rüdiger Loos, and Rudolf Albrecht, editors. *Computer Algebra: Symbolic and Algebraic Computation*. Springer, 2nd edition, 1983.

[9] Bruce W. Char, Keith O. Geddes, and Gaston H. Gonnet. GCDHEU: Heuristic polynomial GCD algorithm based on integer GCD computation. *J. Symb. Comput.*, 7(1):31–48, 1989.

[10] George E. Collins. The calculation of multivariate polynomial resultants. *J. ACM*, 18(4):515–532, 1971.

[11] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 3rd edition, 2007.

[12] Jennifer de Kleine, Michael Monagan, and Allan Wittkopf. Algorithms for the non-monic case of the sparse modular GCD algorithm. In *Proceedings of ISSAC 2005*, pages 124–131. ACM, 2005.

[13] David S. Dummit and Richard M. Foote. *Abstract Algebra*. Wiley, 3rd edition, 2004.

[14] Mark Encarnación. Computing GCDs of polynomials over algebraic number fields. *J. Symb. Comput.*, 20:299–313, 1995.

[15] Euclid. *The Thirteen Books of Euclid's Elements. Volume 2: Books III–IX*. Dover Publications, New York, 1956. Translated by Thomas L. Heath; 2nd revised edition; Book VII included.

[16] J.-C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[17] Lisl Gaal. *Classical Galois Theory: With Examples*. AMS Chelsea Publishing, American Mathematical Society, Providence, RI, 2nd edition, 1998.

[18] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Springer, 1992.

[19] Roger A. Horn and Charles R. Johnson. *Positive definite matrices*, pages 391–486. Cambridge University Press, Cambridge, 1985.

[20] Jiaxiong Hu and Michael Monagan. A fast parallel sparse polynomial GCD algorithm. *J. Symb. Comput.*, 105(1):28–63, 2021.

[21] L. Kronecker. Grundzüge einer arithmetischen theorie der algebraischen grössen. *Journal für die reine und angewandte Mathematik*, 92:1–122, 1882.

[22] Lars Langemyr and Scott McCallum. The computation of polynomial greatest common divisors over an algebraic number field. *J. Symb. Comput.*, 8(5):429–448, 1989.

[23] Xin Lin, Marc Moreno Maza, and Éric Schost. Fast arithmetic for triangular sets: from theory to practice. *J. Symb. Comput.*, 44(7):891–907, 2009.

[24] Dragoslav S. Mitrinović. *Elementary Inequalities*. P. Noordhoff Ltd., Groningen, 1964. See §0.2 "Bernoulli's Inequality", p. 15, formula (A).

[25] Michael Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In *Proceedings of ISSAC 2004*, pages 243–249. ACM, 2004.

[26] Michael Monagan. Probabilistic algorithms for computing resultants. In *Proceedings of ISSAC 2005*, pages 245–252. ACM, 2005.

[27] Michael Monagan. Speeding up polynomial GCD, a crucial operation in maple. *Maple Trans.*, 2(1), 2022. Article (August 2022), 18 pages.

[28] Michael Monagan, Keith Geddes, K. Heal, G. Labahn, S. Vorkoetter, J. McCarron, and P. DeMarco. *Maple 8 Introductory Programming Guide*, 2003.

[29] Joel Moses and David Y. Y. Yun. The EZ GCD algorithm. In *Proceedings of the ACM Annual Conference*, pages 159–166. ACM, 1973.

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

[31] Trevor Smedley. A new modular algorithm for computation of algebraic number polynomial gcds. In *Proceedings of ISSAC 1989*, pages 91–94. ACM, 1989.

[32] Barry M. Trager. Algebraic factoring and rational function integration. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '76)*, pages 219–226. ACM, 1976.

[33] B. L. van der Waerden. *Modern Algebra*, volume 1. Frederick Ungar Publishing Co., New York, 1953. Translated by Fred Blum.

[34] Mark van Hoeij and Michael Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *Proceedings of ISSAC 2002*, pages 109–116. ACM, 2002.

[35] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013.

[36] Paul S. Wang. The EEZ-GCD algorithm. *SIGSAM Bull.*, 14(2):50–60, 1980.

[37] Paul S. Wang. A p-adic algorithm for univariate partial fractions. In *Proceedings of the Fourth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '81)*, pages 212–217. ACM, 1981.

[38] Paul S. Wang, M. J. T. Guy, and J. H. Davenport. P-adic reconstruction of rational numbers. *SIGSAM Bull.*, 16(2):2–3, 1982.

[39] Franz Winkler. *Polynomial Algorithms in Computer Algebra*. Texts and Monographs in Symbolic Computation. Springer-Verlag, Wien, 1996.

[40] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM '79*, pages 216–226. Springer-Verlag, 1979.

[41] Richard Zippel. Interpolating polynomials from their values. *J. Symb. Comput.*, 9(3):375–403, 1990.

[42] Richard Zippel. *Effective Polynomial Computation*. The Computer Algebra Handbook. Kluwer Academic Publishers, Boston, 1993.