FAST MULTIPLICATION OVER ALGEBRAIC NUMBER FIELDS

by

Cory Ahn

B.Sc. (Hons.), University of Western Ontario, 2008

THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE

DEPARTMENT OF MATHEMATICS

FACULTY OF SCIENCE

SIMON FRASER UNIVERSITY

Fall 2011

APPROVAL

Name: Cory Ahn

Degree: Master of Science

Title of thesis: Fast Multiplication Over Algebraic Number Fields

Examining Committee: Petr Lisonek (Chair)

Michael Monagan
Senior supervisor
Professor

Nils Bruin
Supervisor
Associate Professor

Jason Bell
Examiner
Associate Professor

Date Approved: December 1, 2011

Declaration of
Partial Copyright Licence

Abstract

Let $K = \mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_t)$ be an algebraic number field of degree D over $\mathbb{Q}$, and let f and g be polynomials in $K[x]$ of degrees at most n. Naïvely computing the product $f \cdot g$ requires $\mathcal{O}(n^2 D^2)$ arithmetic operations in $\mathbb{Q}$. We developed a more efficient algorithm that computes $f \cdot g$ modulo a series of primes to avoid working with rationals, and uses the fast Fourier Transform. For each prime p, it also avoids working over multiple extensions by computing a primitive element of K modulo p, which can be done using linear algebra or resultants and gcds. Our algorithm requires $\mathcal{O}(D^3 + D^2 n + Dn \log n)$ arithmetic operations in $\mathbb{F}_p$ for each prime p. We have implemented our algorithm in Maple and present some timings to demonstrate that there is good speed-up in practice.

To my mother, for providing me with
so much love, wisdom, support, and encouragement

Acknowledgments

First I wish to express my sincere gratitude to my supervisor Dr. Michael Monagan for introducing me to this problem and continually providing guidance, support, patience and insight. I am also grateful for the helpful comments and feedback on the thesis provided by Roman Pearce and Dr. Nils Bruin. I would also like to thank the Department of Mathematics at the Simon Fraser University for giving me the opportunity to do my Master's here. Lastly, I am grateful to all of those who provided emotional and moral support they provided during the completion of this thesis. You know who you are.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction and preliminaries

Let $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ be an algebraic number field of degree D where each α_i is algebraic over $\mathbb{Q}$. Furthermore let $f(x)$ and $g(x)$ be polynomials in $K[x]$ of degrees at most n. We would like to compute the product of f and g efficiently. If $m_1 \in \mathbb{Q}[u_1]$ is the minimal polynomial of α_1 over $\mathbb{Q}$ and $m_i \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ is the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \leq i \leq t$, a straightforward multiplication strategy is to view f and g as $(t+1)$-variate polynomials in $\mathbb{Q}[u_1, \ldots, u_t][x]$ and classically multiply them modulo the ideal $\langle m_1, \ldots, m_t \rangle$. Unfortunately, performing arithmetic operations in $\mathbb{Q}$ and divisions by the minimal polynomials $m_1, \ldots, m_t$ can be computationally expensive. In this thesis we speed up the multiplication using various strategies. Namely:

- We eliminate the need for computing over multiple extensions by performing the multiplication over a simple extension $\mathbb{Q}(\gamma) = K$, then converting the product back to $K[x]$ at the end. We discuss two methods for finding a primitive element γ in Chapter 5. One is a linear algebra approach and the other is a resultant approach.

- We eliminate the need for computing over $\mathbb{Q}$ by mapping the rational coefficients of f and g to $\mathbb{F}_{p_1}, \mathbb{F}_{p_2}, \ldots, \mathbb{F}_{p_k}$ where p_i's are primes that do not divide any denominator of $f, g, m_1, \ldots, m_t$. That is, we consider $f \mod p_i$ and $g \mod p_i$ as polynomials over $K_{p_i} = \mathbb{F}_{p_i}[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$ where $M_i = m_i \mod p_i$ for $1 \leq i \leq t$, then compute the product over these rings. We use the Chinese

Remainder Theorem and rational number reconstruction at the end to recover the rational coefficients in the product. These ideas are classical in computer algebra and are briefly discussed in Chapter 2.

- In Chapter 6, we modify the resultant approach for finding a primitive element in K to apply it to $K_{p_i} = \mathbb{F}_{p_i}[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$ for suitable primes p_i. Because K_{p_i} may not be a field (it is usually not a field), many modifications to the method are necessary. We show that computing a primitive element γ modulo p_i requires $\mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_{p_i}$, where $D = \deg(K)$.

- Once a primitive element $\overline{\gamma} = \gamma \mod p_i$ is found, one can convert $f, g \in K[x]$ to polynomials in $\mathbb{F}_{p_i}(\overline{\gamma})[x]$ and perform the multiplication over this ring instead. In Chapter 4, we discuss how to convert the representations of polynomials over multiple extensions to a simple extension, and vice versa. We also show that the cost of the conversions is $\mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_{p_i}$.

- We employ a fast multiplication method for the multiplication over $\mathbb{F}_{p_i}$, which is based on the Fast Fourier Transform (FFT). The idea of using the FFT for polynomial multiplication is classical in computer algebra and is discussed in Chapter 3. If we naïvely compute the product of two polynomials in $\mathbb{F}_{p_i}(\overline{\gamma})[x] \cong \mathbb{F}_{p_i}[z]/\langle M_\gamma(z) \rangle[x]$ of degrees n, where $m_\gamma(z) \in \mathbb{Q}[z]$ is the minimal polynomial of a primitive element γ of degree D and $M_\gamma(z) = m_\gamma(z) \mod p_i$, it would require $\mathcal{O}(D^2 n^2)$ arithmetic operations in $\mathbb{F}_{p_i}$. Using the FFT reduces the cost of multiplication to $\mathcal{O}(D^2 n + Dn \log n)$ arithmetic operations in $\mathbb{F}_{p_i}$.

- In Chapter 7 we discuss further efficiency improvements to the primitive element finding algorithm presented in Chapter 6, specifically faster gcd and resultant algorithms.

Our polynomial multiplication algorithm requires

$$\mathcal{O}(D^3 + D^2 n + Dn \log n) \text{ arithmetic operations in } \mathbb{F}_{p_i},$$

for each prime p_i. Since classical multiplication requires $\mathcal{O}(D^2 n^2)$ arithmetic operations in $\mathbb{Q}$, this is an improvement when D is less than n^2. We note that it is also

an improvement over performing the FFT multiplication over the multiple extensions when $D < n$.

Figure 1.1 gives an overview of various multiplication strategies. In this thesis, we use the strategy indicated by the double arrows, which we implemented in Maple. We present some timings in Chapter 8 and it shows a significant speed-up compared to the naïve multiplication method.

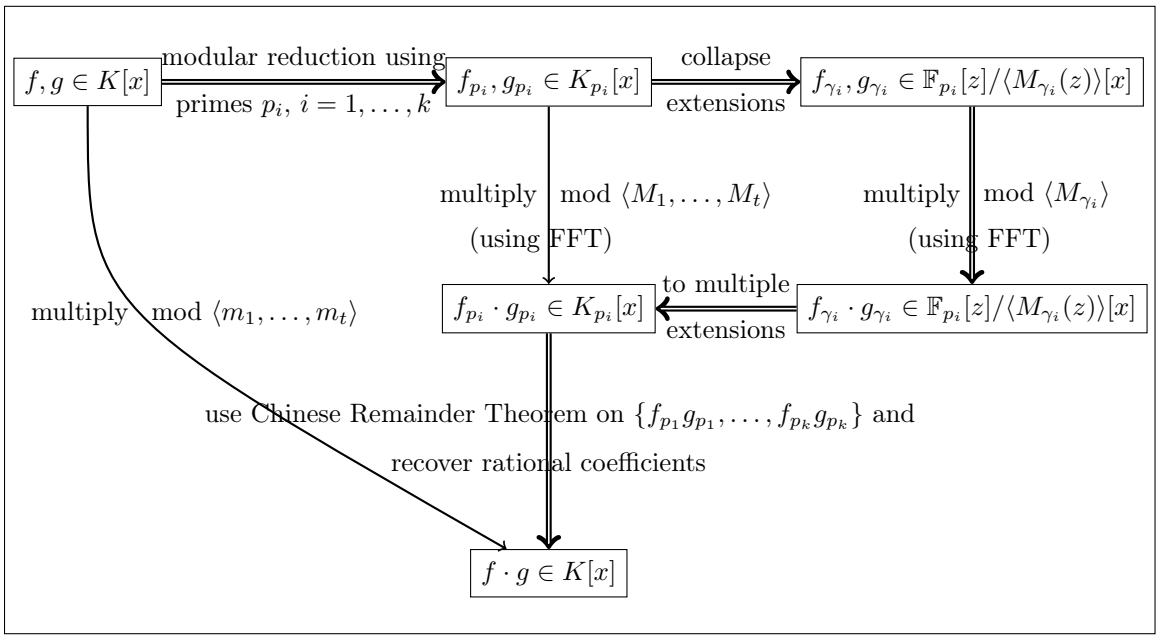

Figure 1.1: Overview of various multiplication strategies

1.1 Preliminaries

In this section, we give definitions, notations and basic results from algebraic number theory and computer algebra which will be used throughout this thesis.

In what follows, we will assume that K is a field and E is an extension of K.

Definition 1.1. Let K be a proper subfield of a field E. We say that E is an **extension field**, or simply an **extension**, of K. To indicate that E is an extension of K we write E/K.

Definition 1.2. Let $\alpha \in E$ and let $K(\alpha)$ be the smallest subfield of E containing both K and α. We say that $K(\alpha)$ is formed from K by **adjoining** a single element α, and we call it a **simple extension of** K. More generally for $t \geq 2$, let $K(\alpha_1, \ldots, \alpha_t)$ be the smallest field that contains K and $\alpha_1, \ldots, \alpha_t$. $K(\alpha_1, \ldots, \alpha_t)$ is called a **multiple extension of** K. For $t \geq 1$, if $K(\alpha_1, \ldots, \alpha_t) \cong K(\alpha)$ we say that α is a **primitive element** for $K(\alpha_1, \ldots, \alpha_t)$ over K.

Definition 1.2 implies that $K(\alpha_1, \ldots, \alpha_t)$ can be regarded as a field obtained by successively adjoining a single element to K as follows:

$$\begin{aligned} K(\alpha_1, \alpha_2) &= K(\alpha_1)(\alpha_2), \\ K(\alpha_1, \alpha_2, \alpha_3) &= K(\alpha_1, \alpha_2)(\alpha_3), \\ &\vdots \\ K(\alpha_1, \ldots, \alpha_{t-1}, \alpha_t) &= K(\alpha_1, \ldots, \alpha_{t-1})(\alpha_t). \end{aligned}$$

Definition 1.3. An element α in some extension of K is said to be **algebraic over** K if there exists a nonzero polynomial over K with α as a root.

Example 1.4. Consider the field $\mathbb{Q}(\sqrt{2}, \sqrt{5})$. Since $\sqrt{2}$ is a root of $x^2 - 2 \in \mathbb{Q}[x]$ and $\sqrt{5}$ is a root of $x^2 - 5 \in \mathbb{Q}[x]$, we conclude that $\sqrt{2}$ and $\sqrt{5}$ are algebraic over $\mathbb{Q}$.

Definition 1.5. Let α be algebraic over K. A *monic* polynomial of least degree that has α as a root is called a **minimal polynomial of** α **over** K and is denoted by $m_\alpha(x)$. We say that α and β are **conjugates over** K if they have the same minimal polynomial over K. In such a case β (respectively α) is a **conjugate of** α (respectively β) over K.

Lemma 1.6. *Let α be algebraic over K. Then a minimal polynomial $m_\alpha(x)$ is unique and irreducible over K. Moreover, if $f(x) \in K[x]$ and $f(\alpha) = 0$ then $m_\alpha(x)$ divides $f(x)$.*

Proof. See, for example, Alaca and Williams [1, Theorem 5.1.1, p. 89]. □

Theorem 1.7. *Let K be a subfield of E and $\alpha \in E$ be algebraic over K. If $n = \deg(m_\alpha)$ then*

$$K(\alpha) = \{c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1} | c_0, \ldots, c_{n-1} \in K\}.$$

Proof. Let

$$L = \left\{ \frac{f(\alpha)}{g(\alpha)} : f(x) = \sum_{i=0}^{k} a_i x^i, g(x) = \sum_{i=0}^{h} b_i x^i, k, h \in \mathbb{Z}_{\geq 0},\ a_i, b_i \in K,\ g(\alpha) \neq 0 \right\}.$$

Then L is a subfield of E that contains both α and K. Furthermore it is the smallest subfield containing α and K, since any subfield of E containing both α and K must contain all the elements of L. Therefore $L = K(\alpha)$. Now pick an element $\beta \in L = K(\alpha)$. Then $\beta = \dfrac{f(\alpha)}{g(\alpha)}$ where

$$\begin{aligned} &f(x) = a_0 + a_1 x + \cdots + a_k x^k \in K[x], \\ &g(x) = b_0 + b_1 x + \cdots + b_h x^h \in K[x], \text{ and } g(\alpha) \neq 0 \text{ for some } a_i, b_i \in K. \end{aligned}$$

Since $g(\alpha) \neq 0$, we must have $m_\alpha(x) \nmid g(x)$. Moreover, since $m_\alpha(x)$ is irreducible over K, $\gcd(m_\alpha(x), g(x)) = 1$. By the Extended Euclidean algorithm there exist $r(x)$ and $s(x)$ in $K[x]$ such that

$$r(x) \cdot m_\alpha(x) + s(x) \cdot g(x) = 1. \tag{1.1}$$

Substituting α for x in (1.1), we get $r(\alpha) \cdot 0 + s(\alpha) \cdot g(\alpha) = 1$, so

$$\beta = \frac{f(\alpha)}{g(\alpha)} = f(\alpha) \cdot s(\alpha).$$

That is, every $\beta \in L = K(\alpha)$ can be expressed as

$$d_0 + d_1\alpha + \cdots + d_l\alpha^l \text{ where } l \in \mathbb{Z}_{\geq 0} \text{ and } d_0, \ldots, d_l \in K.$$

Let $h(x) = d_0 + d_1 x + \cdots + d_l x^l \in K[x]$. Since K is a field, we have

$$h(x) = q(x)\ m_\alpha(x) + r(x), \text{ where } r(x) = 0 \text{ or } \deg(v(x)) < \deg(m_\alpha(x)) = n.$$

Thus we have $h(\alpha) = q(\alpha) \cdot 0 + r(\alpha) = r(\alpha)$. That is, every element of $K(\alpha)$ can be written as $c_0 + c_1\alpha + \cdots + c_{n-1}\alpha^{n-1}$, where $c_0, \ldots, c_{n-1} \in K$ and $n = \deg(m_\alpha(x))$. □

Theorem 1.7 implies that $K(\alpha)$ can be viewed as an n-dimensional vector space over K with basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$. Furthermore, it is easy to see that there is an isomorphism between $K(\alpha)$ and a quotient ring as follows:

$$K(\alpha) \cong K[x]/\langle m_\alpha(x) \rangle.$$

In general, if $\{\alpha_1, \ldots, \alpha_t\} \subseteq E$, each α_i is algebraic over K, $\alpha_1 \notin \mathbb{Q}$ and $\alpha_i \notin \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \leq i \leq t$, then

$$K(\alpha_1, \ldots, \alpha_t) \cong K[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle, \tag{1.2}$$

where each $m_i := m_i(u_i)$ represents the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \leq i \leq t$ and $m_1 := m_1(u_1)$ is the minimal polynomial of α_1 over $\mathbb{Q}$.

In this thesis, we will assume that $\{m_1, m_2, \ldots, m_t\}$ is a *triangular* set of minimal polynomials. By this we mean that if $K(\alpha_1, \ldots, \alpha_t) \cong K[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$ then:

- α_1 is a root of $m_1(u_1) \in K[u_1]$,
- α_2 is a root of $m_2(\alpha_1, u_2) \in K(\alpha_1)[u_2]$,
  $\vdots$
- α_t is a root of $m_t(\alpha_1, \ldots, \alpha_{t-1}, u_t) \in K(\alpha_1, \ldots, \alpha_{t-1})[u_t]$.

Example 1.8. The field $K(\alpha_1, \alpha_2) \cong K[u_1, u_2]/\langle u_1^2 - 2, u_2^2 - u_2 u_1 + 3 \rangle$ involves a triangular set of minimal polynomials $\{u_1^2 - 2, u_2^2 - u_2 u_1 + 3\} = \{m_1(u_1), m_2(u_1, u_2)\}$.

Definition 1.9. Let α be algebraic over K. The **degree of α over K**, $\mathbf{deg_K(\alpha)}$, is equal to $\deg_x(m_\alpha)$, where $m_\alpha(x) \in K[x]$ is the minimal polynomial of α over K.

When the field K is clear from context, we will simply write $\deg(\alpha)$.

Let α be algebraic over K and let $m_\alpha(x) \in K[x]$ be the minimal polynomial of α over K so that $K(\alpha) \cong K[x]/\langle m_\alpha \rangle$. If $d = \deg_K(\alpha)$ then every polynomial $f(x) \in K[x]$ can be reduced modulo $m_\alpha(x)$ to some $r(x)$ with $\deg(r) < d$. Note that two different polynomials $r(x)$ and $s(x)$ with $\deg(r), \deg(s) < d$ cannot be congruent modulo $f(x)$, as that would imply that $r(x) - s(x)$ (a non-zero polynomial) is a multiple of $m_\alpha(x)$. Thus every element $A \in K(\alpha)$ has a unique representation

$$A(x) = \sum_{i=0}^{d-1} a_i x^i + \langle m_\alpha(x) \rangle, \ a_i \in K.$$

Hence we arrive at the following definition.

Definition 1.10. Let α be algebraic over K with $d = \deg_K(\alpha)$ and let $m_\alpha(x) \in K[x]$ be the minimal polynomial of α over K. We say that the unique representation of any element $A \in K(\alpha)$,

$$A(x) = \sum_{i=0}^{d-1} a_i x^i + \langle m_\alpha(x) \rangle, \ a_i \in K$$

is the **normal representation** of A and sometimes write it as $A(\alpha)$.

Definition 1.11. Let K be a subfield of E and let $\alpha \in E$ be algebraic over K of degree n. The **degree of** $K(\alpha)$ over K, denoted $[K(\alpha) : K]$, is defined as $[K(\alpha) : K] = n$. More generally, if E is an extension of K, we denote by $[E : K]$ the dimension of E viewed as a vector space over K and call it the **degree of** E **over** K.

Lemma 1.12. *Let F be a subfield of a field K and let K be a subfield of a field E. Then*

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

Proof. See, for example, Gaal [9, Theorem, p. 34]. □

Example 1.13. Let us find $[\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}]$. Applying Lemma 1.12, we have

$$[\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}] = [\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}(\sqrt{5})] \cdot [\mathbb{Q}(\sqrt{5}) : \mathbb{Q}].$$

Since $[\mathbb{Q}(\sqrt{5}) : \mathbb{Q}] = \deg_{\mathbb{Q}}(\sqrt{5}) = \deg_x(x^2 - 5) = 2$ and noting that $x^2 - 2$ is irreducible over $\mathbb{Q}(\sqrt{5})$,

$$[\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}(\sqrt{5})] = [\mathbb{Q}(\sqrt{2})(\sqrt{5}) : \mathbb{Q}(\sqrt{5})] = \deg_x(x^2 - 2) = 2,$$

we conclude that $[\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}] = 2 \cdot 2 = 4$. One can also argue that, since $\{1, \sqrt{2}, \sqrt{5}, \sqrt{2}\sqrt{5}\}$ is a basis for the vector space $\mathbb{Q}(\sqrt{2}, \sqrt{5})$ and it has dimension 4, $[\mathbb{Q}(\sqrt{2}, \sqrt{5}) : \mathbb{Q}] = 4$.

Throughout this thesis, we will denote by m_i the minimal polynomial of α_i and D the degree of $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ over $\mathbb{Q}$. Since m_i is irreducible over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$, it follows from Lemma 1.12 that $D = \prod_{i=1}^{t} \deg(m_i)$. The cost of arithmetic in K will depend on D.

Definition 1.14. Let $f(x)$ be a polynomial over K. A field extension E over K in which $f(x)$ factors into linear factors is called a **splitting field** of $f(x)$ over K.

Definition 1.15. Let R be a commutative ring. A polynomial $f \in R[x_1, \ldots, x_n]$ is **square-free** if and only if no square polynomial of non-zero degree divides f. That is, if f is square-free then no polynomial $g \in R[x_1, \ldots, x_n]$ with $g \notin R$ exists such that $g^2 \mid f$.

Theorem 1.16. *Let $f(x)$ be a non-constant polynomial in $K[x]$ where K is a field of characteristic 0. Then $f(x)$ is square-free if and only if $\gcd(f(x), f'(x)) = 1$, where $f'(x)$ is the derivative of $f(x)$ with respect to x.*

Proof. Suppose that $f(x)$ is not square-free. Then it can be written as $f(x) = g(x)^2 \cdot h(x)$ where g is a polynomial of degree greater than zero. Differentiating f with respect to x, we have

$$f'(x) = 2g(x)g'(x)h(x) + g(x)^2h'(x) = g(x)\left(2g'(x)h(x) + g(x)h'(x)\right).$$

Thus $f(x)$ and $f'(x)$ have a non-trivial gcd since $g(x)$ divides them both. Thus if $\gcd(f, f') = 1$ then f is square-free.

On the other hand, if f is square-free we can write it as $f(x) = \prod_{i=1}^{n} f_i(x)$ where the $f_i(x)$'s are non-constant pairwise relatively prime irreducible polynomials. Again, differentiating $f(x)$ with respect to x, we get

$$f'(x) = \sum_{i=1}^{n} \left(f_i'(x) \prod_{j=1, j\neq i}^{n} f_j(x) \right).$$

Observe that f_i divides all the summands of $f'(x)$ except the i-th. Because K has characteristic 0, $f_i'(x) \neq 0$ since $f_i(x)$ is a non-constant polynomial. Hence $\gcd(f(x), f'(x)) = 1$. □

We remark that Theorem 1.16 does not generalize to polynomials over commutative rings. For example, consider the polynomial $f(x) = x^3 + t \in F_3(t)[x]$. Since $f'(x) = 0$, we have $\gcd(f(x), f'(x)) = x^3 + t$. However, $f(x)$ is an irreducible (hence square-free) polynomial. In our algorithm, we discard any polynomial f for which

$\gcd(f, f') \in \mathbb{F}_p[u_1, \cdots, u_t]/\langle M_1, \cdots, M_t\rangle[x]$ does not equal 1 (see Algorithm 6.1 Line 12) even if f is a square-free polynomial. This scheme does not cause an increase in the overall complexity of the algorithm.

One way to compute a primitive element of a multiple extension is by computing resultants, which we define below.

Definition 1.17. Let R be a commutative ring and let $f(x), g(x) \in R[x]\backslash\{0\}$ with $f(x) = \sum_{i=0}^{m} a_i x^i$ and $g(x) = \sum_{i=0}^{n} b_i x^i$. The **Sylvester matrix** of f and g, denoted $\mathrm{Syl}_x(f, g)$, is the $(m + n)$ by $(m + n)$ matrix

$$\mathrm{Syl}_x(f,g) = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_1 & a_0 & & \\ & a_m & a_{m-1} & \cdots & a_1 & a_0 & \\ & & & \vdots & & & \\ & & & a_m & \cdots & \cdots & a_0 \\ b_n & b_{n-1} & \cdots & b_1 & b_0 & & \\ & b_n & b_{n-1} & \cdots & b_1 & b_0 & \\ & & & \vdots & & & \\ & & & b_n & \cdots & \cdots & b_0 \end{bmatrix},$$

where the first n rows consist of the coefficients of $f(x)$, the remaining m rows consist of the coefficients of $g(x)$, and the entries not shown are zero.

Definition 1.18. Let R be a commutative ring and let $f, g \in R[x]$. The **resultant** of f and g with respect to x, written $\mathrm{res}_x(f, g)$, is the determinant of $\mathrm{Syl}_x(f, g)$.

We now list some properties of resultants as theorems and lemmas.

Theorem 1.19. *Let R be a commutative ring and let $f, g \in R[x]$ with nonzero degrees m and n respectively. Then*

(i) $\mathrm{res}(f, g) = (-1)^{mn}\,\mathrm{res}(g, f)$.

(ii) If R is an integral domain and $g(x) = b_n \prod_{i=1}^{n}(x - \beta_i)$ then

$$\mathrm{res}(f, g) = b_n^m \prod_{i=1}^{n} f(\beta_i).$$

Proof. See Geddes et al. [11, Theorem 9.2, p. 408] and Zippel [22, p. 142]. □

Lemma 1.20. *Let R be a commutative ring, $f(x) \in R[x]$, and $g(x,y) \in R[x][y]$. If $\deg_y(g) = n > 0$ and $\deg_x(g) = 0$ then $\operatorname{res}_y(f(x), g(y)) = (f(x))^n$.*

Proof. The Sylvester matrix formed by f and g (in y) is a diagonal matrix of the form:

$$\operatorname{Syl}_y(f(x), g(y)) = \begin{bmatrix} f(x) & 0 & \cdots & 0 \\ 0 & f(x) & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & f(x) \end{bmatrix} = (f(x))^n .$$

□

Theorem 1.21. *Let J be an integral domain and let $f(x), g(x) \in J[x]$ have degrees m and n respectively with $m+n \geq 1$. Then there exist $s(x), t(x) \in J[x]$ with $\deg(s) < n$ and $\deg(t) < m$ such that $f(x)s(x) + g(x)t(x) = \operatorname{res}(f,g)$.*

Proof. See Geddes et al. [11, Theorem 7.1, p. 287]. □

Corollary 1.22. *Let U be a UFD and let $f(x), g(x) \in U[x]$, not both zero. Then f and g have a non-trivial common factor if and only if $\operatorname{res}_x(f,g) = 0$.*

Proof. If $\operatorname{res}(f,g) \neq 0$ then, by a consequence of Theorem 1.21, any common factor of f and g must divide $\operatorname{res}(f,g)$. But since $\operatorname{res}(f,g)$ belongs to U, a common factor of f and g must have degree 0. So there cannot be any non-trivial factors. Conversely, if $\operatorname{res}(f,g) = 0$ then, again, Theorem 1.21 tells us that $f(x)s(x) = -g(x)t(x)$ with $\deg(s) < \deg(g)$ and $\deg(t) < \deg(f)$. Suppose for contradiction that $f(x)$ and $g(x)$ do not have a non-trivial common factor. Then $g(x)$ must divide $s(x)$. However $\deg(s) < \deg(g)$, so this is impossible. □

Remark 1.23. *Let K be a field of characteristic 0. Corollary 1.22 and Theorem 1.16 imply that $f \in K[x]$ is square-free if and only if $\operatorname{res}_x(f(x), f'(x)) \neq 0$.*

One can use resultants to compute *norms*, which we define below.

Definition 1.24. Let $h(x,\alpha)$ belong to $K(\alpha)[x]$ where K is a field of characteristic 0, and let $m_\alpha(x) \in K[x]$ be the minimal polynomial of α over K. The **norm of** h, denoted by $\text{norm}_{[K(\alpha)[x]/K[x]]}(h(x,\alpha))$, is

$$\text{norm}_{[K(\alpha)[x]/K[x]]}(h(x,\alpha)) := \prod_{i=1}^{n} h(x,\alpha_i)$$

where $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_n$ are conjugates of α over K in an algebraic closure of K.

When the minimal polynomial of α is clear from context, we simply write $\text{norm}(\cdot)$. The following theorem describes a relationship between resultants and norms.

Theorem 1.25. *Let K be a field of characteristic 0. Further let $h(x,\alpha)$ be a monic polynomial over $K(\alpha)$ and let $m_\alpha(y) \in K[y]$ be the minimal polynomial of α over K. Then*

$$\text{norm}(h(x,\alpha)) = \text{res}_y(h(x,y), m_\alpha(y)) \in K[x].$$

Proof. Let the roots of $m_\alpha(y)$ be $\alpha_1 = \alpha, \ldots, \alpha_n$. By Theorem 1.19, $\text{res}_y(h(x,y), m_\alpha(y)) = \prod_{i=1}^{n} h(x,\alpha_i)$, which, by definition, is equal to $\text{norm}(h(x,\alpha))$. Clearly, $\text{norm}(h(x,\alpha))$ belongs to $K[x]$, since taking the resultant of $h(x,y) \in K[x,y]$ and $m_\alpha(y) \in K[y]$ with respect to y eliminates the variable y. □

Example 1.26. Let $K(\alpha) = \mathbb{Q}(\sqrt{2})$. Then $m_\alpha(y) = y^2 - 2$ and the conjugates of $\alpha = \sqrt{2}$ are $\sqrt{2}, -\sqrt{2}$. If $h(x,\alpha) = x^4 - \alpha x^2 + 5$, then

$$\begin{aligned}
\text{norm}(h(x,\alpha)) &= h(x,\sqrt{2}) \cdot h(x,-\sqrt{2}) \\
&= x^8 + 10x^4 - \alpha^2 x^4 + 25 \\
&= x^8 + 8x^4 + 25 \\
&= \text{norm}(h(x,-\alpha)) \in \mathbb{Q}[x].
\end{aligned}$$

Furthermore,

$$\text{res}_y(h(x,y), m_\alpha(y)) = \det\left(\begin{bmatrix} -x^2 & x^4+5 & 0 \\ 0 & -x^2 & x^4+5 \\ 1 & 0 & -2 \end{bmatrix}\right) = x^8 + 8x^4 + 25 = \text{norm}(h(x,\alpha)),$$

as expected.

1.2 The Polynomial Representation

In what follows, let $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t) \cong \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$ where $m_1(u_1) \in \mathbb{Q}[u_1]$ is the minimal polynomial for α_1 over $\mathbb{Q}$, $m_i(u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ is the minimal polynomial for α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$, and $f, g \in K[x]$. In this chapter, we introduce the data structure we will use to represent f and g.

1.2.1 The `recden` representation

We will represent each α_i by the variable u_i. Hence f and g will be stored as t-variate polynomials in $\mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle[x]$. Since f and g can be dense polynomials, we will use a recursive dense (`recden` package in Maple) data structure to represent them. The `recden` data structure takes as input a polynomial together with information about the polynomial ring and outputs a recursive list that represents the polynomial.

Example 1.27. Let $f(x) = 8\sqrt[3]{5}x^2 - 4(\sqrt[3]{5})^2\sqrt{3} + 13 \in \mathbb{F}_7(\sqrt[3]{5}, \sqrt{3})[x]$. Observe that f can be equivalently expressed as $f(x, y, z) = 8x^2y - 4y^2z + 13 \in \mathbb{F}_7[y, z]/\langle y^3 - 5, z^2 - 3 \rangle[x]$. We can represent f using the `recden` data structure in Maple as follows:

```
> f:= 8*x^2*y - 4*z*y^2 +13:
> F:=rpoly(f,[x,y,z],[z=RootOf(a^2-3), y=RootOf(a^3-5)],7);
```

$$F := (x^2y \ + \ 6 + 3zy^2) \mod \langle \ y^3 + 2, \ z^2 + 4, 7 \rangle$$

```
> lprint(F);
POLYNOMIAL([7, [x, y, z], [[[2], 0, 0, [1]], [4, 0, 1]]],
[[[6], 0, [0, 3]], 0, [0, [1]]])
```

The first list `[7, [x, y, z], [[[2], 0, 0, [1]], [4, 0, 1]]]` provides information about the polynomial ring, namely the characteristic, the list of variables, and a list of the minimal polynomials in `recden` representation. The second list `[[[6], 0, [0, 3]], 0, [0, [1]]]` is the `recden` representation of f, as depicted in Figure 1.2.

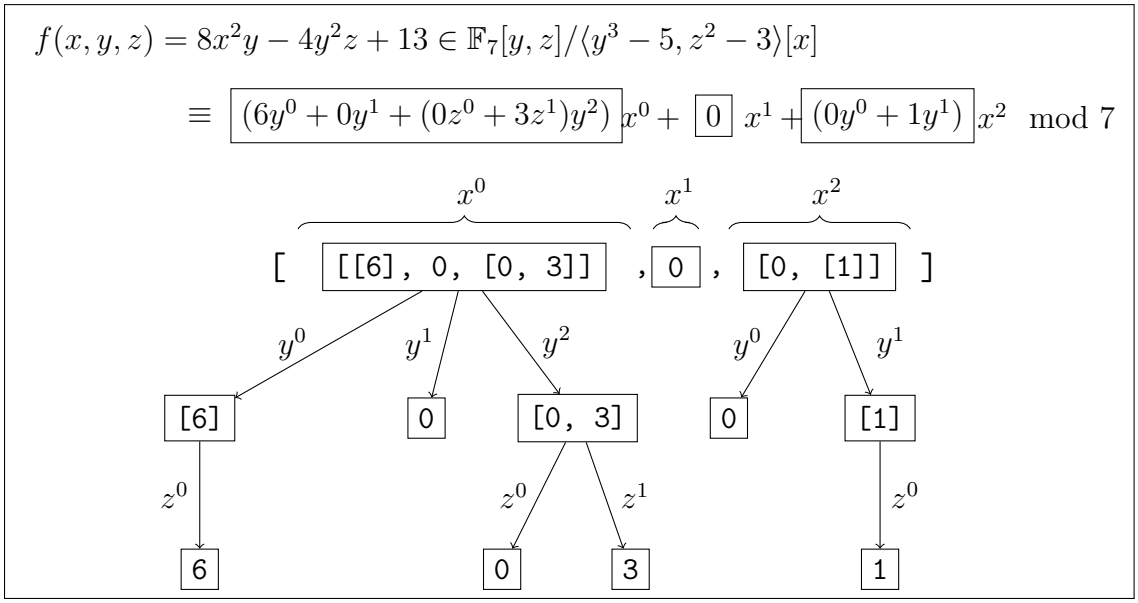

Figure 1.2: The `recden` representation of f in Example 1.27

Observe the recursive nature of this data structure and the importance in the ordering of the variables. It is a "dense" representation because *some* zero coefficients of $x^i y^j z^k$ (for $i = 0, 1, 2$, $j = 0, 1, 2$, and $k = 0, 1$) are stored. For example, the coefficient of x^1 is 0 so it does not recurse on this coefficient. We also remark that this polynomial representation is not unique to Maple. A recursive dense data structure in Magma can be constructed using the command `quo`:

```
Q := RationalField();
K<z> := PolynomialRing(Q); # build the polynomial ring Q[z]
m := z^2-2;
L<a> := quo<K|m>;  # build the field Q[z]/<m>
K2<y> := PolynomialRing(L); # build the polynomial ring Q[z,y]/<m>
m2 := y^2+a*y+1;
L2<b> := quo<K2|m2>;  # build the field Q[z,y]/<m,m2>
f := a+b;
g := 2*a+b+1;
f*g;
(2*a + 1)*b + a + 3
P<x> := PolynomialRing(L2); # build the poly. ring Q[z,y]/<m,m2>[x]
```

```
f := x+a+b+1;
g := a*x+b+2;
f*g;
a*x^2 + ((a + 1)*b + (a + 4))*x + 3*b + 2*a + 1
```

Chapter 2

Working modulo p

Let $m_1(u_1) \in \mathbb{Q}[u_1]$ be the minimal polynomial for α_1 over $\mathbb{Q}$ let $m_i(u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ be the minimal polynomials for α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$, $2 \leq i \leq t$. Multiplying polynomials in $K[x] = \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle[x]$ is relatively expensive partly due to expression swell in the size of the rationals. To control the swell, we shall make use of *modular homomorphisms*.

A basic tool in elementary number theory and computer algebra is the ring homomorphism $\Phi_p : \mathbb{Z} \mapsto \mathbb{F}_p$, where p is a prime. This morphism naturally extends to a homomorphism $\mathbb{Z}[u_1, \ldots, u_t] \mapsto \mathbb{F}_p[u_1, \ldots, u_t]$ via coefficient-wise application, which we also denote by Φ_p. It follows that for any ideal $\langle m_1, \ldots, m_t \rangle \subset \mathbb{Z}[u_1, \ldots, u_t]$ where the m_i's form a triangular set so that $m_i \in \mathbb{Z}[u_1, \cdots, u_{i-1}][u_i]$, and $\mathrm{lcoeff}_{u_i}(m_i) = 1$, we get a ring $\mathbb{Z}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$ and a corresponding homomorphism

$$\begin{aligned} \mathbb{Z}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle \mapsto \quad & \mathbb{F}_p[u_1, \ldots, u_t]/\langle \Phi_p(m_1), \ldots, \Phi_p(m_t) \rangle \\ & \cong \mathbb{Z}[u_1, \ldots, u_t]/\langle p, m_1, \ldots, m_t \rangle, \end{aligned}$$

which is denoted yet again by Φ_p. The context will clarify which Φ_p we mean. Many computational tasks in $\mathbb{Z}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$ can be solved by performing the tasks on the images under various Φ_p. The original answer can later be constructed by applying the Chinese Remainder Theorem. In Section 2.1, we will investigate to what extent the homomorphism Φ_p can be extended to $\mathbb{Q}$. In Section 2.2 we discuss the Chinese Remainder Theorem and rational number reconstruction, which is used for reconstructing the rational answers. In Section 2.3 we show by example how we

will use this method for the purpose of polynomial multiplication.

2.1 Modular homomorphism

Not all elements in $\mathbb{Q}$ can be mapped to $\mathbb{F}_p$ for prime p. In particular, any element in $\mathbb{Q}$ whose denominator is divisible by p cannot be mapped. For example, if $m_1(x) = x^2 - \frac{3}{55}$ then $p = 5$ and $p = 11$ cannot be used since $\Phi_{11}(m_1)$ and $\Phi_5(m_1)$ are not defined. On the other hand, if the denominator $r \in \mathbb{Z}$ is not divisible by p then $\Phi_p(r)$ is invertible in $\mathbb{F}_p$, so we can extend Φ_p to $\mathbb{Z}[1/r]$ by prescribing

$$\Phi_p(1/r) := (\Phi_p(r))^{-1}.$$

The subset of $\mathbb{Q}$ consisting of 0 and elements whose denominators are not divisible by p is a subring of $\mathbb{Q}$, and it is called the *localization* of $\mathbb{Z}$ at the prime ideal $\langle p \rangle$. It is denoted by

$$\mathbb{Z}_{\langle p \rangle} := \mathbb{Z}[1/r : r \text{ prime}, r \neq p] = \mathbb{Q} \backslash \{n/d : p \mid d,\ p \nmid n,\ d \neq 0\}.$$

The ring $\mathbb{Z}_{\langle p \rangle}$ has a unique maximal ideal $p\mathbb{Z}_{\langle p \rangle}$ and by construction $\mathbb{Z}_{\langle p \rangle}/p\mathbb{Z}_{\langle p \rangle} \cong \mathbb{Z}/p\mathbb{Z} = \mathbb{F}_p$ so the homomorphism $\Phi_p : \mathbb{Z} \mapsto \mathbb{F}_p$ naturally extends to $\Phi_p : \mathbb{Z}_{\langle p \rangle} \mapsto \mathbb{F}_p$. It is straightforward to check that any element $a \in \mathbb{Q} \backslash \mathbb{Z}_{\langle p \rangle}$ is the inverse of an element $b \in \mathbb{Z}_{\langle p \rangle}$ with $\Phi_p(b) = 0$. This shows that Φ_p cannot be extended to any subring of $\mathbb{Q}$ bigger than $\mathbb{Z}_{\langle p \rangle}$.

In our setting, we are given $f, g, m_1, \ldots, m_t \in \mathbb{Q}[u_1, \ldots, u_t]$ and we are interested in computing a representative of the residue class of $f \cdot g$ in $\mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$. Since there are only finitely many coefficients in $f, g, m_1, \ldots, m_t$, there are only finitely many primes that occur in any of their denominators. Hence we have $f, g, m_1, \ldots, m_t \in \mathbb{Z}_{\langle p \rangle}[u_1, \ldots, u_t]$ for all but finitely many p. If p is a prime such that $f, g, m_1, \ldots, m_t \notin \mathbb{Z}_{\langle p \rangle}[u_1, \ldots, u_t]$, then we say that it is a **bad prime**. For any non-bad prime p, we can study the class of $\Phi_p(f \cdot g)$ in $\mathbb{F}_p[u_1, \ldots, u_t]/\langle \Phi_p(m_1), \ldots, \Phi_p(m_t) \rangle$ and use fast multiplication techniques for polynomials over finite fields (Chapters 3 and 6). We repeat this procedure for several primes p, then use the Chinese remainder Theorem and rational number reconstruction to lift the answer back to our original quotient ring $\mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$.

As it turns out, we need to impose additional conditions on our primes p in order for efficient multiplication techniques to apply. We explain these conditions below.

In our original description, we consider a triangular system $m_1, \ldots, m_t$ and write α_i for the residue class of u_i in $\mathbb{Q}[u_1, \ldots, u_i]/\langle m_1, \ldots, m_i \rangle$. Furthermore, we insist that each such quotient ring is a field, so each polynomial $m_i(\alpha_1, \ldots, \alpha_{i-1}, u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ must be irreducible and hence certainly square-free. For ease of notation, suppose that p is a non-bad prime and consider the induced reduction homomorphism (which we call *modular homomorphism*)

$$\Phi_p : \mathbb{Z}_{\langle p \rangle}[u_1, \ldots, u_t]/\langle m_1, \ldots m_t \rangle \mapsto \mathbb{F}_p[u_1, \ldots, u_t]/\langle \Phi_p(m_1), \ldots, \Phi_p(m_t) \rangle$$

and write $\overline{\alpha}_i := \Phi_p(\alpha_i)$. Insisting that the monic $m_i(\overline{\alpha}_1, \ldots, \overline{\alpha}_{i-1}, u_i)$ is irreducible over
$\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{i-1}]$turns out to be too restrictive and computationally expensive to check, but we do impose that it is square-free. The reason for this restriction is explained in Chapter 6. Primes that do not satisfy the square-free condition for some m_i are called **fail primes**. A non-bad, non-fail prime is called a **good prime**.

If $m_1, \ldots, m_t$ are monic and have integral coefficients then algebraic number theory associates to a ring $\mathbb{Z}[\alpha_1, \ldots, \alpha_t] \cong \mathbb{Z}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$ an integer called a *discriminant* whose definition can be found in, for example, Lang [12, pp. 64-66]. A prime p is a fail prime if and only if it divides the discriminant of the ring. One can extend this notion to more general m_i. The important observation for us is that primes are fail primes only if they divide an integer that depends only on $m_1, \ldots, m_t$ and hence there are only finitely many fail primes.

2.2 Chinese Remainder Theorem & rational number reconstruction

Let $f, g \in K[x] = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)[x]$. Furthermore let $p_1, \ldots, p_k$ be good primes and suppose that we computed $h_i = \Phi_{p_i}(f) \cdot \Phi_{p_i}(g) \in \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$ for $1 \leq i \leq k$. To recover the product $f \cdot g$ as a polynomial in $K[x]$, one can apply Chinese Remainder Theorem and rational reconstruction to the h_i's.

The **integer Chinese remainder problem** can be stated as follows:

Given moduli $m_0, \ldots, m_k \in \mathbb{Z}$ *and the corresponding* residues $u_i \in \mathbb{F}_{m_i}$ *for* $1 \leq i \leq k$, *find an integer* $u \in \mathbb{Z}$ *satisfying* $u \equiv u_i \mod m_i$.

An algorithm for solving the Chinese remainder problem is the "inverse" of the modular homomorphism, since it reconstructs the integer u from the residues $u_i \equiv u \mod m_i$.

The following theorem provides the sufficient condition for which the Chinese remainder problem can be solved with a unique solution.

Theorem 2.1. *Let* $m_0, m_1, \ldots, m_k \in \mathbb{Z}$ *be pairwise relatively prime. Further, let* $u_i \in \mathbb{F}_{m_i}$ *for* $1 \leq i \leq k$ *be the* $k+1$ *residues. Then there exists a unique integer* $u \in \mathbb{Z}$ *satisfying*

$$u \equiv u_i \mod m_i \quad \text{for } 0 \leq i \leq k \text{ and}$$
$$0 \leq u < m_0 \cdot m_1 \ldots m_k.$$

Proof. See Geddes et al. [11, Theorem 5.7, p. 175]. □

The Chinese remainder theorem generalizes to any finitely generated torsion-free $\mathbb{Z}$-module. We will apply it to the coefficients of polynomials in $\mathbb{F}_{p_i}[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$ for primes $p_1, \ldots, p_k$. Since the moduli $p_1, \ldots, p_k$ are relatively prime, Theorem 2.1 implies that given $h_i(x) = \Phi_{p_i}(f) \cdot \Phi_{p_i}(g) \in \mathbb{F}_{p_i}[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$ for $1 \leq i \leq k$, we can find a unique $h(x)$ satisfying $h(x) \equiv h_i(x) \mod p_i$ for $1 \leq i \leq k$, where the coefficients of $h(x)$ lie between 0 and $m-1$ inclusive, where $m = p_1 p_2 \cdots p_k$.

Once we obtain $h(x)$, we must find the *rational* coefficients of $f \cdot g$, which currently belong to $\mathbb{F}_m$. We accomplish this using *rational number reconstruction* on the coefficients of $h(x)$.

The **rational number reconstruction problem** can be stated as follows:

Let $\frac{n}{d} \in \mathbb{Q}$ *with* $\gcd(n, d) = 1$ *and* $m \in \mathbb{Z}$ *be the* modulus *with* $\gcd(m, d) = 1$. *If* $u \equiv \frac{n}{d} \mod m$, *recover* $\frac{n}{d}$ *from* u *and* m.

We use the Euclidean algorithm to recover n and d given u and m. Recall that on input of m and u, the extended Euclidean algorithm computes a sequence of integers r_i, s_i, and t_i satisfying

$$r_0 = m,\ s_0 = 1,\ t_0 = 0,$$
$$r_1 = u,\ s_1 = 0,\ t_1 = 1, \text{ and}$$
$$s_i \cdot m + t_i \cdot u = r_i \text{ for } i > 1.$$

Thus the rationals $\frac{r_i}{t_i}$ with $\gcd(t_i, m) = 1$ satisfy $\frac{r_i}{t_i} \equiv u \mod m$. The following theorem states that if the modulus m is "large enough", then one of the $\frac{r_i}{t_i}$'s is equal to $\frac{n}{d}$.

Theorem 2.2. (Wang et al. [20]) *If $m > 2|n \cdot d|$ then there exists $i \in \mathbb{N}$ such that $\frac{r_i}{t_i} = \frac{n}{d}$.*

The question then is, how do we select $\frac{n}{d}$ from the $\frac{r_i}{t_i}$'s? The approach taken by Wang et al. [20] requires input bounds $D > |d|$ and $N > |n|$. However, we will use the algorithm of Monagan [15] which does not require bounds and succeeds with high probability when m is a few (≈ 30) bits longer than $2|n \cdot d|$. Since both algorithms use the Extended Euclidean algorithm, the cost of rational number reconstruction of $\frac{n}{d}$ from $u \in \mathbb{F}_m$ is $\mathcal{O}((\log m)^2)$. Both algorithms are implemented in Maple's `iratrecon` command and `irrrpoly` command (for the `recden` data structure).

2.3 New multiplication strategy

We now illustrate our multiplication strategy using an example.

Example 2.3. Let $K = \mathbb{Q}[u_1, u_2]/\langle u_1^2 - 2, u_2^2 - 5\rangle \cong \mathbb{Q}(\sqrt{2}, \sqrt{5})[x]$. Furthermore let

$$f(x) = \frac{2}{7}x^2 - 2u_1x + \frac{19}{11}u_2 \in K[x], \text{ and } g(x) = \frac{6}{121}x^2 + \frac{1}{3}u_2u_1x - \frac{51}{2} + u_2 \in K[x].$$

We first note that the answer we seek is

$$h(x) = f(x) \cdot g(x) = \frac{12}{847}\,x^4 + \left(\frac{2}{21}\,u_2 + \frac{12}{121}\right)u_1\,x^3 + \left(\frac{47648}{27951}\,u_2 - \frac{51}{7}\right)x^2$$
$$+ \left(2\,u_2 - \frac{1588}{33}\right)u_1\,x - \frac{969}{22}\,u_2 + \frac{95}{11}.$$

To find the product of f and g, we apply a modular homomorphism on f and g. Let $|n|$ and $|d|$ denote the maximum coefficients (in magnitude) in the numerator and the denominator of the product, respectively. We remark that one can compute a bound on $|n|$ and $|d|$ by examining the coefficients appearing in $f(x)$ and $g(x)$. Recall that we must choose the modulus $m = \prod_{i=1}^{k} p_i$ to be a few bits larger than $2|n \cdot d|$ in order for the rational reconstruction to succeed (Monagan [15]). In this example, $2|n \cdot d| =$

$2|47648 \cdot 27951| = 2663618496$. Let $p_1 = 101, p_2 = 103, p_3 = 107, p_4 = 113, p_5 = 131$, and $p_6 = 137$. Then $m = \prod_{i=1}^{6} p_i = 2257421632331 > 2|n \cdot d|$. We compute the product of f and g modulo these primes in Maple as follows.

```
> f:=2/7*x^2 + 2*u_1*x + 19/11*u_2:
> g:= 6/121*x^2 + 1/3 *u_2*u_1*x - 51/2 + u_2:
> p:=[101,103,107,113,131,137]:
> FGp:=Array(1..nops(p)):
> vars:=[x,u_1,u_2]:
> r:= [u_1=RootOf(x^2-2), u_2=RootOf(x^2-5)]:
> for i from 1 to nops(p) do
>    Fp:= rpoly(f,vars,r,p[i]):# modular homomorphism (f mod p[i])
>    Gp:= rpoly(g,vars,r,p[i]):# modular homomorphism (g mod p[i])
>    FGp[i]:= mulrpoly(Fp,Gp); # Fp x Gp (over K mod p[i])
>    FGp[i]:= retextsrpoly(FGp[i]); # Make FGp[i] an element of
                            # Zp[x,u_1,u_2] (i.e. drop extensions)
> od:
```

Now we use Chinese remaindering to find the product mod m, where m is the product of primes in the list `p`.

```
> fg:= ichremrpoly(convert(FGp,list));
```

$$fg := 538369739942x^4 + (859970145650u_2 + 1511166547263)\,u_1x^3$$

$$+\ (225249505657u_2 + 1934932827705)\,x^2 +\ (2u_2 + 1162914174183)\,u_1x$$

$$+\ 718270519334u_2 + 1231320890371 \mod 2257421632331$$

Finally we apply rational number reconstruction on `fg`.

```
> FGrat:= irrrpoly(fg);
```

$$FGrat := \frac{12}{847}x^4 + \left(\frac{2}{21}u_2 + \frac{12}{121}\right)u_1x^3 + \left(\frac{47648}{27951}u_2 - \frac{51}{7}\right)x^2 +\ \left(2u_2 - \frac{1588}{33}\right)u_1x$$

$$-\frac{969}{22}u_2 + \frac{95}{11}$$

Chapter 3

Fast multiplication using the FFT

We will use the Fast Fourier Transform (FFT) to efficiently multiply polynomials in $K_p[x] = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$. In Sections 3.1 and 3.2 we describe the FFT and the inverse FFT. In Section 3.4 we explain how they can be used to speed up the multiplication process.

3.1 The Fast Fourier Transform (FFT)

Let $a(x) = \sum_{i=0}^{n-1} a_i x^i \in R[x]$ where R is a ring and let N be the smallest power of 2 greater than or equal to n. The idea of the FFT is to evaluate $a(x)$ at N points using $\mathcal{O}(N \log N)$ arithmetic operations in R. As a comparison, the cost of these N evaluations using the classical Horner's method requires $\mathcal{O}(N^2)$ multiplications and $\mathcal{O}(N^2)$ additions in R. To explain the FFT algorithm, we first need some definitions and lemmas.

Definition 3.1. Let F be a field, $N \in \mathbb{N}$, and $\omega \in F$. We say that ω is an **N-th root of unity** if $\omega^N = 1$. Moreover, we say that ω is a **primitive N-th root of unity** if $\omega^N = 1$ and $\omega^k \neq 1$ for $0 < k < N$. The set of N points $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ are called **Fourier points**.

We remark that if $\omega \in F$ is a primitive N-th root of unity, then $\omega^{-1} = \omega^{N-1} \in F$, since $\omega^{-1} = \omega^{N-1}$.

Lemma 3.2. *Let ω be a primitive N-th root of unity in F where N is even. Then the N Fourier points $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ satisfy the symmetry condition $\omega^{N/2+i} = -\omega^i$. Furthermore, ω^2 is a primitive $(N/2)^{th}$ root of unity.*

Proof. See Geddes et al. [11, Lemmas 4.2 & 4.3, pp. 125-126]. □

Lemma 3.2 implies that if $N = 2^k$ for some positive integer k, then the Fourier points recursively satisfy the symmetry condition; that is, the symmetry condition holds for the set of N points $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$, the set of $N/2$ points $\{1, \omega^2, \ldots, \omega^{N-2}\}$, and so on.

Definition 3.3. We say that a commutative ring R over a field F **supports the FFT for** $N = 2^k$ if F has a primitive N-th root of unity.

Definition 3.4. Let $a(x) \in R[x]$ where R is a ring (over a field F) that supports the FFT for N where $N = 2^k \geq \deg(a)$ for some integer k. Furthermore let ω be a primitive N-th root of unity in F. If a_i denotes the coefficient of x^i in $a(x)$, then the mapping $T_\omega^N : R^N \mapsto R^N$ given by

$$T_\omega^N(a_0, a_1, \ldots, a_{N-1}) = (a(1), a(\omega), \ldots, a(\omega^{N-1}))$$

is called the **discrete Fourier transform** (DFT).

The Fast Fourier Transform is an efficient algorithm that computes the DFT by utilizing the fact that the primitive N-th root of unity satisfies the recursive symmetry condition as follows. Let R be a ring over a field F and let $a(x) \in R[x]$ have degree less than N where $N = 2^k$. Moreover, let $\omega \in F$ be a primitive N-th root of unity$\in F$. Observe that we can express $a(x)$ as

$$a(x) = b(x^2) + x\, c(x^2), \tag{3.1}$$

where $b(y) = \sum_{i=0}^{N/2-1} a_{2i}\, y^i$ and $c(y) = \sum_{i=0}^{N/2-1} a_{2i+1}\, y^i$. Since $\omega_{N/2+i} = -\omega_i$, we have

$$\omega_{N/2+i}^2 = \omega_i^2 \text{ for } 0 \leq i \leq N/2 - 1.$$

Hence by evaluating $b(y)$ and $c(y)$ at $N/2$ points $\omega_0^2, \omega_1^2, \ldots, \omega_{N/2-1}^2$, we can evaluate $a(x)$ at the N points $\omega_0, \omega_1, \ldots, \omega_N$, saving approximately half the work.

Algorithm 3.1 returns the DFT of $a(x) \in R[x]$ using the FFT, where R is a commutative ring over a field F that supports the FFT for $N = 2^k > \deg_x(a)$.

Algorithm 3.1 : **FFT**$(a(x), R, N, \omega)$

Input: $a(x) = \sum_{i=0}^{n} a_i x^i \in R[x]$ where R is a commutative ring over a field F,
N: integer satisfying $n < N = 2^k$, ω: a primitive N-th root of unity in F.

Output: the Fourier transform of $a(x)$: $[a(\omega^0), a(\omega^1), \ldots, a(\omega^{N-1})] \in R^N$.

1: **if** $N = 1$ **then return** $[a_0]$; **end if**
2: $A \leftarrow$ array of length N; $W \leftarrow$ array of length $N/2$;
3: $b(x) \leftarrow \sum_{i=0}^{N/2-1} a_{2i} \cdot x^i$; $c(x) \leftarrow \sum_{i=0}^{N/2-1} a_{2i+1} \cdot x^i$;
4: $W[0] \leftarrow 1$;
5: **for** $i = 1$ to $N/2 - 1$ **do** $W[i] \leftarrow W[i-1] \cdot \omega$; **end for** { note: $W[i] = \omega^i$ }
6: $B \leftarrow$ **FFT**$(b(x), N/2, W[2])$;
7: $C \leftarrow$ **FFT**$(c(x), N/2, W[2])$;
8: **for** $i = 0$ to $N/2 - 1$ **do**
9: $T \leftarrow W[i] \cdot C[i]$;
10: $A[i] \leftarrow B[i] + T$; {note: $B[i], C[i] \in R$}
11: $A[N/2 + i] \leftarrow B[i] - T$;
12: **end for**
13: **return** A;

Lemma 3.5. *Algorithm **FFT** returns the Fourier transform of $a(x)$.*

Proof. Lines 6 and 7 of Algorithm **FFT** outputs

$$B = [b(1), b(\omega^2), b(\omega^4), \ldots, b(\omega^{N-2})] \text{ and } C = [c(1), c(\omega^2), c(\omega^4), \ldots, c(\omega^{N-2})].$$

Further, by Eq. (3.1) and Lemma 3.2,

$$A[i] = B[i] + W[i] \cdot C[i] = b(\omega^{2i}) + \omega^i \cdot c(\omega^{2i}) = a(\omega^i) \text{ for } 0 \le i < N/2 \text{ and}$$

$$\begin{aligned} A[N/2+i] &= B[i] - W[i] \cdot C[i] \\ &= b(\omega^{2i}) + (-\omega^i) \cdot c(\omega^{2i}) \\ &= b(\omega^{N+2i}) + \omega^{N/2+i} \cdot c(\omega^{N+2i}) \\ &= b(\omega^{2(N/2+i)}) + \omega^{N/2+i} \cdot c(\omega^{2(N/2+i)}) \\ &= a(\omega^{N/2+i}) \quad \text{for } 0 \le i < N/2. \end{aligned}$$

In other words, $A[i] = a(\omega^i)$ for $i = 0, \ldots, N-1$, as required. □

In our application, the ring R is equal to $\mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$, and $D = \prod_{i=1}^{t} \deg(M_i)$. Thus we analyze the complexity of Algorithm **FFT** for $R = \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$.

- Line 5: Computing all the required powers of ω^i uses $\dfrac{N}{2} - 1$ multiplications in $\mathbb{F}_p$.
- Lines 8 - 12: $C[i] \in R$, so it has at most $D = \prod_{i=1}^{t} \deg(M_i)$ terms. Hence multiplying $W[i] \in \mathbb{F}_p$ by $C[i]$ requires at most D multiplications in $\mathbb{F}_p$. Thus this for-loop requires at most

$$\sum_{i=0}^{N/2-1} D = \frac{ND}{2} \text{ multiplications in } \mathbb{F}_p.$$

Hence if $T(N)$ denotes the number of multiplications (in $\mathbb{F}_p$) in executing Algorithm **FFT** using the primitive N-th root of unity ω, then $T(N)$ satisfies the recursion

$$\begin{cases} T(1) = 0, \\ T(N) = \left(\frac{N}{2} - 1\right) + 2T(\frac{N}{2}) + \frac{ND}{2}, \ \ N \ge 1. \end{cases} \tag{3.2}$$

Solving (3.2), we obtain

$$T(N) = \frac{N}{2}(1+D)(\log_2 N) + 1 \in \mathcal{O}(ND \log N).$$

3.2 Inverse FFT

Definition 3.6. The **inverse discrete Fourier transform (IDFT)** for the Fourier points $\{1, \omega, \omega^2, \ldots, \omega^{N-1}\}$ is the mapping

$$S_\omega^N(q_0, q_1, \ldots, q_{N-1}) \mapsto (\tilde{q}_0, \tilde{q}_1, \ldots, \tilde{q}_{N-1}) \text{ where } \tilde{q}_i = N^{-1} \sum_{k=0}^{N-1} q_k \cdot (\omega^{-j})^k.$$

One can show that DFT and IDFT are inverses of each other, and that IDFT is also a Fourier transform (Geddes et al. [11, Theorem 4.2, pp. 130-132]). In fact, if $T_\omega^N(a_0, a_1, \ldots, a_{N-1}) = (\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{N-1})$ is a DFT then

$$S_\omega^N(\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{N-1}) \quad = N^{-1} T_\omega^N(\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{N-1}) = (a_0, a_1, \ldots, a_{N-1}).$$

3.3 Choosing the right prime

In our application, we are interested in finding a primitive N-th root of unity in $\mathbb{F}_p$, where p is a prime. As such, we can only use p in which there exists a primitive N-th root of unity in $\mathbb{F}_p$. We call such primes **Fourier primes**. The following lemma tells us how to find a Fourier prime.

Lemma 3.7. *Let p be a prime. $\mathbb{F}_p$ has a primitive N-th root of unity if and only if N divides $p-1$.*

Proof. See Geddes et al. [11, Theorem 4.3, p. 133]. □

In our Maple implementation, we choose p to be between 2^{30} and $2^{31.5}$ so that p is large but all multiplications can be performed using signed integers on a 64-bit machine without causing overflow. To find a Fourier prime of this magnitude, we will first find the largest integer $M < 2^{31.5}$ for which N divides $M-1$. If M is a prime, we have found a Fourier prime by Lemma 3.7. Otherwise, we subtract N from M as many times as necessary, until M is prime.

3.4 Fast multiplication using the FFT

The following lemma explains how to utilize the FFT to speed up the multiplication of polynomials.

Lemma 3.8. *Let f and g be polynomials in $R[x]$ where $\deg(f)+\deg(g) < N = 2^k$ and R is a ring over a field F that supports the FFT for N. Let us write f and g as $f = \sum_{i=0}^{N-1} f_i x^i$ and $g = \sum_{i=0}^{N-1} g_i x^i$ and define $\boldsymbol{f} = (f_0, \ldots, f_{N-1})$ and $\boldsymbol{g} = (g_0, \ldots, g_{N-1})$. If ω is the primitive N-th root of unity in F, then*

$$T_\omega^N(\boldsymbol{f} \cdot \boldsymbol{g}) = T_\omega^N(\boldsymbol{f}) \cdot T_\omega^N(\boldsymbol{g}) \in R^N,$$

where $\cdot$ denotes component-wise multiplication.

Proof. See von zur Gathen and Gerhard [18, Lemma 8.11, p. 228-229]. □

In light of Lemma 3.8 we can use the FFT for multiplying f and g over $K_p = \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$ as follows. First compute the Fourier transforms $T_\omega^N(\mathbf{f})$ and $T_\omega^N(\mathbf{g})$ using the FFT, then apply component-wise multiplication of the Fourier transforms (that is, perform N multiplications in K_p), and finally apply the inverse Fourier transform. Algorithm **FFTMult** uses the FFT to multiply polynomials in $K_p[x] = \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle[x]$.

We now present the fast FFT polynomial multiplication algorithm.

Algorithm 3.2 : **FFTMult**$(f(x), g(x), K_p)$

Input: $f(x), g(x) \in K_p[x] = \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle[x]$, p a good Fourier prime.
Output: $h(x) = f(x) \cdot g(x) \in K_p[x]$ via the FFT.

1: $N \leftarrow$ the smallest power of 2 greater than $(\deg_x(f) + \deg_x(g))$;
2: $\omega \leftarrow$ primitive N-th root of unity in $\mathbb{F}_p$;
3: $F \leftarrow \mathbf{FFT}(f(x), K_p, N, \omega)$; $\{F \in K_p^N\}$
4: $G \leftarrow \mathbf{FFT}(g(x), K_p, N, \omega)$; $\{G \in K_p^N\}$
5: **for** $i = 0$ to $N - 1$ **do**
6: $\quad H[i] = F[i] \cdot G[i]$; {component-wise multiplication where $F[i], G[i] \in K_p$}
7: **end for**
8: $h \leftarrow N^{-1} \cdot \mathbf{FFT}(\sum_{i=0}^{N-1} H[i] \cdot x^i, K_p, N, \omega^{-1})$; {inverse FFT}
9: $h(x) \leftarrow \sum_{i=0}^{N-1} h[i] \cdot x^i$;
10: **return** $h(x)$;

We analyze the cost of Algorithm **FFTMult** where we assume $D = \prod_{i=1}^{t} \deg(M_i)$.

- Algorithm **FFT** is called three times in Algorithm **FFTMult** and one execution of Algorithm **FFT** requires $\frac{N}{2}(1 + D)(\log N) + 1$ multiplications in $\mathbb{F}_p$ (Section 3.1). Thus the number of multiplications in $\mathbb{F}_p$ required in performing

three FFTs is

$$3\left(\frac{N}{2}(1+D)(\log_2 N)+1\right) \sim \frac{3}{2}ND\log_2 N \in \mathcal{O}(ND\log N).$$

- Lines 5 to 7: Since $F[i], G[i] \in K_p$, this for-loop requires N multiplications in K_p. An arithmetic operation in K_p can be done using $\mathcal{O}(D^2)$ arithmetic operations in $\mathbb{F}_p$ (von zur Gathen and Gerhard [18, Corollary 4.6, p. 72]), so this for-loop requires $\mathcal{O}(ND^2)$ multiplications in $\mathbb{F}_p$.
- Line 8: since $N^{-1} \in \mathbb{F}_p$, we perform at most ND multiplications in $\mathbb{F}_p$.

In total, Algorithm **FFTMult** uses

$$\begin{aligned} &O(ND\log N)+\mathcal{O}(ND^2)+\mathcal{O}(ND)\\ &\subseteq \mathcal{O}(ND\log N+ND^2) \text{ multiplications in } \mathbb{F}_p. \end{aligned}$$

If $n=\max\{\deg(f),\deg(g)\}$, then $N<4n\in\mathcal{O}(n)$. So Algorithm **FFTMult** requires

$$\mathcal{O}(nD\log n+nD^2) \text{ multiplications in } \mathbb{F}_p.$$

In comparison, classical multiplication of f and g requires $\mathcal{O}(n^2D^2)$ multiplications in $\mathbb{F}_p$.

Chapter 4

Polynomial Representation Simplification

Multiplying polynomials over $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$ is inefficient even with the use of the FFT because the `recden` data structure becomes more "complicated" with increasing number of variables in the polynomial ring. Consider the following example.

Example 4.1. Let $f = x+a+b+c+d \in \mathbb{F}_{101}[a,b,c,d]/\langle a^2-2, b^2-3, c^2-5, d^2-7\rangle[x]$. Let us build the `recden` data structure for f in Maple.

```
> f:=rpoly(a+b+c+d+x, [x,a,b,c,d], [a^2-2,b^2-3,c^2-5,d^2-7],101);
```

$$f := (a+b+c+d+x) \mod \langle\ a^2+99,\ b^2+98,\ c^2+96,\ d^2+94,\ 101\ \rangle$$

```
> lprint(f);
 POLYNOMIAL([101, [x, a, b, c, d], [[[[[99]]], 0, [[[1]]]],
[[[98]], 0, [[1]]], [[96], 0, [1]], [94, 0, 1]]],[[[[[0, 1], [1]],
[[1]]], [[[1]]], [[[[1]]]]])
```

Note the high levels of lists in the `recden` representation of f. In general, a polynomial with t variables has at most t levels of lists. Thus the overhead cost of computing over a multiple extension is high. In particular, if the degree of the first minimal polynomial M_1 is relatively low (in particular if $\deg(M_1) = 2$) then the cost of performing arithmetic in $\mathbb{F}_p$ is overwhelmed by the cost of data structure operations (see Table 8.3).

We can avoid this problem by computing $\overline{\gamma} := \gamma \mod p$ satisfying $\mathbb{Q}(\alpha_1, \ldots, \alpha_t) \cong \mathbb{Q}(\gamma)$ (see Chapters 5 and 6). Once $\overline{\gamma}$ is found, we can represent f and g as bivariate polynomials in $\mathbb{F}_p[\overline{\gamma}] \cong \mathbb{F}_p[z]/\langle \Phi_p(m_\gamma(z))\rangle[x]$, multiply them over this ring, then convert the product back to a polynomial in K_p. This method reduces the overhead and allows for fast arithmetic in $\mathbb{F}_p[\overline{\gamma}]$, but introduces extra costs associated with the conversions between the rings, which we show in Chapter 5 to be $\mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_p$, where $D = [\mathbb{Q}(\alpha_1, \ldots, \alpha_t) : \mathbb{Q}]$.

In Section 4.1, we present a linear algebra method for converting a polynomial represented over a multiple extension field of characteristic 0 to the equivalent polynomial over a simple extension field, and vice versa. This method relies on the use of change-of-basis (COB) matrices. In Section 4.2 we modify this method to apply to convert polynomials from $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$ to $\mathbb{F}_p[\overline{\gamma}][x]$, and vice versa.

4.1 Change-of-basis matrix (in characteristic 0)

In what follows, let $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ and let γ be a primitive element for K. Let us first find the bases for K and $\mathbb{Q}(\gamma)$. For $i = 1, \ldots, t$, let $d_i = [\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i) : \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})]$ and $D = [\mathbb{Q}(\gamma) : \mathbb{Q}]$. By application of Lemma 1.12,

$$D = [\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_{t-1}, \alpha_t) : \mathbb{Q}] = \prod_{i=1}^{t} d_i.$$

Moreover, Theorem 1.7 implies that a basis for $\mathbb{Q}(\gamma)$ is

$$B_\gamma = \{\gamma^0, \gamma^1, \ldots, \gamma^{D-1}\},$$

and a basis for $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ is

$$B_\alpha = \{\alpha_1^{j_1}\alpha_2^{j_2}\ldots\alpha_t^{j_t},\ 0 \le j_i \le d_i - 1,\ 1 \le i \le t\}.$$

Using the bases B_γ and B_α, we can build a change-of-basis matrix C that converts any element in $\mathbb{Q}(\gamma)$ to an element in $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ (and naturally C^{-1} would convert an element in $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ to an element in $\mathbb{Q}(\gamma)$).

Unfortunately, the `recden` data structure is not suitable for constructing the change-of-basis matrix, as the following example illustrates.

Example 4.2. Let the field be $\mathbb{Q}(\alpha_1, \alpha_2)$ with $\alpha_1 = \sqrt{2}$ and $\alpha_2 = \sqrt[3]{3}$. Then one can show that $\mathbb{Q}(\alpha_1, \alpha_2) \cong \mathbb{Q}(\gamma)$ where $\gamma = \alpha_1 + \alpha_2 = \sqrt{2} + \sqrt[3]{3}$. In `recden` with $\alpha_1 > \alpha_2$,

$$\gamma^0 = (\alpha_1 + \alpha_2)^0 = \texttt{[[1]]},$$
$$\gamma^1 = (\alpha_1 + \alpha_2)^1 = \texttt{[[0,1],[1]]},$$
$$\gamma^2 = (\alpha_1 + \alpha_2)^2 = 2 + \alpha_2^2 + 2\alpha_1\alpha_2 = \texttt{[[2, 0, 1], [0, 2]]}, \text{ and}$$
$$\gamma^3 = (\alpha_1 + \alpha_2)^3 = 3 + 6\alpha_2 + (3\alpha_2^2 + 2)\alpha_1 = \texttt{[[3, 6], [2, 0, 3]]}.$$

Because not all zero coefficients are stored in the `recden` data structure, the lists of γ^i are of different lengths. For building the change-of-basis matrix, it would be convenient to use a data structure in which each element in the polynomial ring will be of equal length. To that end, we introduce the following data structure, which is simply a dense one-dimensional array of size D.

Definition 4.3. Let R be a ring and let $f \in R[u_1, \ldots, u_t]/\langle g_1(u_1), \ldots, g_t(u_t)\rangle$. The **completely dense representation (CDR)** of f is the list of coefficients of *all* monomial basis elements in $R[u_1, \ldots, u_t]/\langle g_1, \ldots, g_t\rangle$ in lexicographical ordering with $u_t > \ldots > u_1$.

Example 4.4. Let $R = \mathbb{Q}[u, v]/\langle u^3 - 2, v^2 + 7\rangle$. Then every element in R can be written as

$$c_0 + c_1 u + c_2 u^2 + c_3 v + c_4 uv + c_5 u^2 v.$$

Moreover, the CDR with $v > u$ has the form: $[c_0, c_1, c_2, c_3, c_4, c_5]$.

Unlike the `recden` data structure, the completely dense data structure is an array of depth 1 and every polynomial in a ring in this representation will be of equal length.

In what follows, we let ϕ denote the isomorphism from $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ to $\mathbb{Q}(\gamma)$, and naturally ϕ^{-1} denotes the isomorphism from $\mathbb{Q}(\gamma)$ to $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$.

Let us use the CDR to construct a change-of-basis matrix from $B_\gamma = \{\gamma^0, \gamma^1, \ldots, \gamma^{D-1}\}$ to $B_\alpha = \{\alpha_1^{j_1}\alpha_2^{j_2}\cdots\alpha_t^{j_t},\ j_i = 0, 1, \ldots, d_i - 1, i = 1, \ldots, t\}$, which is the change-of-basis matrix for ϕ^{-1}. We show in Chapter 5 that we can find a primitive element γ of K of the form $\gamma = c_1\alpha_1 + c_2\alpha_2 + \cdots + c_{t-1}\alpha_{t-1} + \alpha_t$ where $c_i \in \mathbb{Z}$. Consider the following $D \times D$ matrix

$$C = \Big[\mathbf{U}_0 \mid \ \mathbf{U}_1 \mid \ \ldots \mid \ \mathbf{U}_{D-1}\Big]$$

where each $\mathbf{U}_i$ is the CDR (as a column vector) of

$$\gamma^i = (c_1\alpha_1 + \ldots + c_{t-1}\alpha_{t-1} + \alpha_t)^i \in \mathbb{Q}(\alpha_1, \ldots, \alpha_t) \cong \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t\rangle.$$

We claim that C is a change-of-basis matrix (COB) for ϕ^{-1} (i.e. from B_γ to B_α). To verify this, we first prove the following lemma.

Lemma 4.5. *The columns of C are linearly independent over K.*

Proof. Suppose towards a contradiction that the columns of C are linearly dependent. Then

$$\mathbf{U}_i = \sum_{j=0, j\neq i}^{D-1} k_j \cdot \mathbf{U}_j \quad \text{where each } k_j \in \mathbb{Q}. \tag{4.1}$$

But since $\mathbf{U}_i$ is the CDR of γ^i for $1 \leq i \leq D-1$, (4.1) is equivalent to stating that

$$\gamma^i = \sum_{j=0, j\neq i}^{D-1} k_j \cdot \gamma^j,$$

which is impossible since $\{\gamma^0, \gamma^1, \ldots, \gamma^{D-1}\}$ is a basis of $K(\gamma)$. Thus the columns of C must be linearly independent over $\mathbb{Q}$. □

Now define $\mathbf{G}_i$ to be the CDR (as a column vector) of γ^i for $1 \leq i \leq D-1$ written as a linear combination of the elements in $B_\gamma = \{1, \gamma, \gamma^2, \ldots, \gamma^{D-1}\}$. That is,

$$\mathbf{G}_i = \begin{bmatrix} 0 & \cdots & 0 & 1 & \cdots & 0 \end{bmatrix}^T, \tag{4.2}$$

where the only 1 is in the $(i+1)$-th row of $\mathbf{G}_i$. Note that

$$C \cdot \mathbf{G}_i = (i+1)\text{-th column of } C = \mathbf{U}_i.$$

Since $\mathbf{U}_i$ by definition is the CDR of γ^i expressed as a linear combination of the elements in B_α, C must be the COB matrix for ϕ. Moreover by Lemma 4.5 C is invertible, so C^{-1} must be the COB matrix for ϕ^{-1}.

Example 4.6. Let $K = \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) \cong \mathbb{Q}[u, v, w]/\langle u^2 - 21, v^2 - 13, w^2 - 5\rangle$. One can show that $\gamma = \alpha_1 + \alpha_2 + \alpha_3$ is a primitive element for K (Chapter 5).

One can show that $K \cong \mathbb{Q}(\gamma)$ and $D = [\mathbb{Q}(\gamma) : \mathbb{Q}] = 8$. Since

$$\begin{aligned}
\gamma^0 &= 1,\\
\gamma^1 &= \alpha_1 + \alpha_2 + \alpha_3,\\
\gamma^2 &= (\alpha_1 + \alpha_2 + \alpha_3)^2 \equiv (2\alpha_2 + 2\alpha_3)\alpha_1 + 2\alpha_3\alpha_2 + 39,\\
\gamma^3 &= (6\alpha_3\alpha_2 + 75)\alpha_1 + 91\alpha_2 + 107\alpha_3,\\
\gamma^4 &= (196\alpha_2 + 260\alpha_3)\alpha_1 + 324\alpha_3\alpha_2 + 3293,\\
\gamma^5 &= (780\alpha_3\alpha_2 + 7141)\alpha_1 + 9029\alpha_2 + 12965\alpha_3,\\
\gamma^6 &= (20070\alpha_2 + 30246\alpha_3)\alpha_1 + 38374\alpha_3\alpha_2 + 332163,\\
\gamma^7 &= (88690\alpha_3\alpha_2 + 744303)\alpha_1 + 945503\alpha_2 + 1466191\alpha_3,
\end{aligned}$$

the basis of $\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3)$ in lexicographical order with $\alpha_3 < \alpha_2 < \alpha_1$ is $\{1,\ \alpha_3, \alpha_2, \alpha_2\alpha_3,\ \alpha_1,\ \alpha_1\alpha_3, \alpha_1\alpha_2,\ \alpha_1\alpha_2\alpha_3\}$. Hence the CDR's of γ^i (denoted by $\mathbf{U}_i$) for $0 \le i \le 7$ are:

$$\begin{aligned}
&\mathbf{U}_0 = [1,0,0,0,0,0,0,0,0]^T, && \mathbf{U}_1 = [0,1,1,0,1,0,0,0]^T,\\
&\mathbf{U}_2 = [39,0,0,2,0,2,2,0]^T, && \mathbf{U}_3 = [0,107,91,0,75,0,0,6]^T,\\
&\mathbf{U}_4 = [3293,0,0,324,0,260,196,0]^T, && \mathbf{U}_5 = [0,12965,9029,0,7141,0,0,780]^T,\\
&\mathbf{U}_6 = [332163,0,0,38374,0,30246,20070,0]^T,\\
&\mathbf{U}_7 = [0,1466191,945503,0,744303,0,0,88690]^T,
\end{aligned}$$

so

$$C = \left[\ \mathbf{U}_0 \mid \mathbf{U}_1 \mid \mathbf{U}_2 \mid \mathbf{U}_3\ \right] = \begin{bmatrix}
1 & 0 & 39 & 0 & 3293 & 0 & 332163 & 0\\
0 & 1 & 0 & 107 & 0 & 12965 & 0 & 1466191\\
0 & 1 & 0 & 91 & 0 & 9029 & 0 & 945503\\
0 & 0 & 2 & 0 & 324 & 0 & 38374 & 0\\
0 & 1 & 0 & 75 & 0 & 7141 & 0 & 744303\\
0 & 0 & 2 & 0 & 260 & 0 & 30246 & 0\\
0 & 0 & 2 & 0 & 196 & 0 & 20070 & 0\\
0 & 0 & 0 & 6 & 0 & 780 & 0 & 88690
\end{bmatrix} \tag{4.3}$$

is the change-of-basis matrix for ϕ. It follows that the change-of-basis matrix for ϕ^{-1}

is

$$C^{-1} = \begin{bmatrix} 1 & 0 & 0 & \frac{24909}{2048} & 0 & -\frac{16445}{1024} & -\frac{31955}{2048} & 0 \\ 0 & \frac{1038997}{514048} & \frac{1009739}{257024} & 0 & -\frac{2544427}{514048} & 0 & 0 & -\frac{33891}{1004} \\ 0 & 0 & 0 & -\frac{5547}{2048} & 0 & \frac{3979}{1024} & -\frac{1387}{2048} & 0 \\ 0 & -\frac{82115}{514048} & -\frac{181805}{257024} & 0 & \frac{445725}{514048} & 0 & 0 & \frac{5833}{2008} \\ 0 & 0 & 0 & \frac{159}{2048} & 0 & -\frac{143}{1024} & \frac{127}{2048} & 0 \\ 0 & \frac{1655}{514048} & \frac{5833}{257024} & 0 & -\frac{13321}{514048} & 0 & 0 & -\frac{39}{502} \\ 0 & 0 & 0 & -\frac{1}{2048} & 0 & \frac{1}{1024} & -\frac{1}{2048} & 0 \\ 0 & -\frac{9}{514048} & -\frac{39}{257024} & 0 & \frac{87}{514048} & 0 & 0 & \frac{1}{2008} \end{bmatrix} \tag{4.4}$$

Note the expression swell that occurs when computing with rationals. We control this swell by working modulo a series of primes $p_1, \ldots, p_k$.

4.2 Change-of-basis matrix modulo p

We now discuss the modifications necessary for applying the change-of-basis matrix method for converting from the finite ring $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$ to $\mathbb{F}_p[\overline{\gamma}]$, and vice versa. Let C be the change-of-basis matrix from $\mathbb{Q}(\alpha, \beta)$ to $\mathbb{Q}(\gamma)$ and let p be a good prime (recall that if p is a good prime then all the minimal polynomials of the extension exist mod p), so that C exists. Observe that C is non-invertible in $\mathbb{F}_p$ if and only if $\det(C) \equiv 0 \mod p$. Let us call good primes p that satisfy $\det(C) \equiv 0 \mod p$ **unlucky**.

Example 4.7. The determinant of C over $\mathbb{Q}$ in Example 4.6 is -12. Since the prime divisors of -12 are 2 and 3, the only unlucky primes in this case are 2 and 3.

As mentioned in Chapter 3, we choose p to be a random Fourier prime between 2^{30} and $2^{31.5}$. We would like to bound the probability of choosing an unlucky prime in this range. To that end, we first state some lemmas.

Lemma 4.8. *(Hadamard's inequality) Let $A \in GL_n(\mathbb{R})$ and let a_{ij} denote the entry of A in i-th row and j-th column. Then*

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

Proof. See Garling [10, Theorem 14.1.1, pp. 233-234]. □

Definition 4.9. Let R be a ring and $f(x) \in R[x]$ where $f(x) = \sum_{i=0}^{n} f_i x^i$. The **max norm** of f, denoted by $\|f\|$ is

$$\|f\| = \max\{|f_0|, |f_1|, \ldots, |f_n|\}.$$

Lemma 4.10. *Let $f(x), m(x) \in \mathbb{Z}[x]$, where $m(x)$ is monic and $\deg_x(m) = d$. Further let $\deg(f) \geq d$ and $\delta = \deg(f) - d + 1$. If $r(x) = f(x) \mod \langle m(x) \rangle$, then*

$$\|r\| \leq (1 + \|m\|)^{\delta} \cdot \|f\|.$$

Proof. See Chen and Monagan [6]. □

Applying Hadamard's inequality and Lemma 4.10, we provide a bound on the number of digits (in base B) of the determinant of change-of-basis matrix when the quotient ring is $\mathbb{Z}[x, y]/\langle m_1, m_2 \rangle$ where $m_1(x) \in \mathbb{Z}[x]$ and $m_2(y) \in \mathbb{Z}[x]/\langle m_1 \rangle[y]$.

Lemma 4.11. *Let $m_1(x) \in \mathbb{Z}[x]$ be monic and $m_2(y) \in \mathbb{Z}[x]/\langle m_1 \rangle[y]$ with $\deg_x(m_1) = d_1$ and $\deg_y(m_2) = d_2$. Furthermore, let $\|m_1\|, \|m_2\| < M$, $D = d_1 d_2$ and*

$$r_i \equiv (cx + y)^i \mod \langle m_1, m_2 \rangle \quad \text{for } c \in \mathbb{Z} \text{ and } 2 \leq i \leq D - 1.$$

If C is a $D \times D$ matrix whose i-th column consists of the coefficients of $(cx+y)^{i-1} \mod \langle m_1, m_2 \rangle$ and $\tilde{c} = \max\{|c|, 1\}$, then the number of digits (in base B) of $\det(C)$ is

$$\begin{aligned} \log_B(|\det(C)|) &\leq D \log_B(D) + \log_B(D \cdot \tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)(\log_B(\tilde{c}+1) + d_1 \log_B(M)) \\ &\in \mathcal{O}\left(D^2(\log_B(\tilde{c}) + d_1 \log_B(M))\right). \end{aligned} \tag{4.5}$$

Proof. See Appendix A. □

One can also prove a more general form of Lemma 4.11 for extensions of more than two steps.

Lemma 4.12. *Let* $m_1(x_1) \in \mathbb{Z}[x_1]$ *be a monic polynomial of degree* d_1*, and*

$$m_k(x_k) \in \mathbb{Z}[x_1, \ldots, x_{k-1}]/\langle m_1(x_1), \ldots, m_{k-1}(x_{k-1})\rangle[x_k]$$

be a monic polynomial of degree d_k *for* $k = 2, \ldots, t$ *and* $\{\|m_1\|, \|m_2\|, \ldots, \|m_t\|\} < M$*. Further, let* $D = \prod_{i=1}^{t} d_i$ *and let* C *be a* $D \times D$ *matrix whose* i*-th column consists of the coefficients from*

$$(c_1x_1 + c_2x_2 + \cdots + c_{t-1}x^{t-1} + x_t)^{i-1} \in \mathbb{Z}[x_1, \ldots, x_t]/\langle m_1, \ldots, m_t\rangle,\ c_i \in \mathbb{Z}.$$

If

$$\begin{aligned}
&\tilde{c} = \max\{|c_1|, |c_2|, \ldots, |c_{t-1}|, 1\},\\
&\tilde{s} = \max\{|c_1|, 1\} + \max\{|c_2|, 1\} + \cdots + \max\{|c_{t-1}|, 1\},\ \textit{and}\\
&\tilde{d} = d_1 + d_2 + \cdots + d_{t-1} + (t-2),
\end{aligned}$$

then the number of digits (in base B*) of* $\det(C)$ *is*

$$\begin{aligned}
\log_B(\det(C)) &\leq D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left(\log_B(\tilde{s}+1) + \tilde{d}\log_B(M)\right)\\
&\in \mathcal{O}\left(D^2\left[\log_B(\tilde{s}) + \tilde{d}\log_B(M)\right]\right).
\end{aligned}$$

Proof. See Appendix A. □

Example 4.13. Let

$$\begin{aligned}
&m_1(x_1) = x_1^2 - 2 \in \mathbb{Q}[x_1],\\
&m_2(x_2) = x_2^2 + 3 \in \mathbb{Q}[x_1]/\langle m_1\rangle[x_2],\\
&m_3(x_3) = x_3^2 - x_1 - 4 \in \mathbb{Q}[x_1, x_2]/\langle m_1, m_2\rangle[x_3],\ \text{and}\\
&m_4(x_4) = x_4^2 + x_1x_2 - 2 \in \mathbb{Q}[x_1, x_2, x_3]/\langle m_1, m_2, m_3\rangle[x_4].
\end{aligned}$$

Since all coefficients of m_i's are integers, we can apply Lemma 4.12. One can show

$$c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \cdot x_3 + c_4 \cdot x + 4 = 26 \cdot x_1 + 330 \cdot x_2 + 905 \cdot x_3 + x_4$$

is a primitive element (modulo p) for $\mathbb{Q}[x_1, x_2, x_3, x_4]/\langle m_1, m_2, m_3, m_4\rangle$. Thus we have

$$\begin{aligned}
&D = \textstyle\prod_{i=1}^{4} \deg_{x_i}(m_i) = 2^4 = 16,\\
&\tilde{c} = \max\{|c_1|, |c_2|, |c_3|, |c_4|, 1\} = \max\{26, 330, 905, 1, 1\} = 905,\\
&\tilde{s} = \max\{|c_1|, 1\} + \max\{|c_2|, 1\} + \max\{|c_3|, 1\} = 26 + 330 + 905 = 1261,\\
&\tilde{d} = \textstyle\sum_{i=1}^{3} \deg_{x_i}(m_i) + (t-2) = 2 + 2 + 2 + (4-2) = 8,\\
&M = \max\{\|m_1\|, \|m_2\|, \|m_3\|, \|m_4\|\} + 1 = 5.
\end{aligned}$$

Hence the length of digits of $\det(C)$ in base $B = 2^{31}$ must be at most

$$D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left(\log_B(\tilde{s}+1) + \tilde{d}\log_B(M)\right) < 105.$$

In fact, if we compute $\det(C)$ we find that $\det(C)$ is between 38 digits and 39 digits long in base 2^{31}.

To execute the FFT, we pick p to be a Fourier prime. We would like to determine the probability that a randomly chosen Fourier prime between 2^{30} and $2^{31.5}$ is unlucky.

Let p be a Fourier prime of the form $k \cdot 2^r + 1$, where $p-1$ is divisible by 2^R for some given $R \in \mathbb{Z}^+$. Further suppose that $2^{30} < p < 2^{31.5}$ and C is a $D \times D$ change-of-basis matrix with integer entries. Table 4.1 lists the number of primes between 2^{30} and $2^{31.5}$ for which $p-1$ is divisible by 2^R, $1 \leq R \leq 28$.

R	1	2	3	4	5	6	7
$n(R)$	91744290	45872521	22936042	11468644	5734170	2867571	1433414
$k(R)$	0	0.999988	2.000002	2.999924	3.999962	4.999717	6.000091
R	8	9	10	11	12	13	14
$n(R)$	716387	358119	178951	89409	44749	22377	11181
$k(R)$	7.00074	8.00104	9.00191	10.00298	11.00155	12.00139	13.00235
R	15	16	17	18	19	20	21
$n(R)$	5581	2773	1377	698	363	178	88
$k(R)$	14.0048	15.0139	16.0238	17.0040	17.9473	18.9754	19.9917
R	22	23	24	25	26	27	28
$n(R)$	45	21	14	9	5	2	0
$k(R)$	20.9593	22.0588	22.6438	23.2812	24.1292	25.4511	-

Table 4.1: $n(R)$ denotes the number of primes between 2^{30} and $2^{31.5}$ of the form $c \cdot 2^R + 1$, and $k(R)$ satisfies the equation $91744290/(2^{k(R)}) = n(R)$, where 91744290 is the number of Fourier primes between 2^{30} and $2^{31.5}$.

One can see by inspection that $k(R) < R - 0.98$ for $1 \leq R \leq 27$. Hence

$$\frac{91744290}{2^{R-0.98}} < \frac{91744290}{2^{k(R)}} = \#\text{ of Fourier primes in } (2^{30}, 2^{31.5}) \text{ of form } k \cdot 2^r + 1, r \geq R.$$

Let $\tilde{c}, \tilde{s}, \tilde{d}$, and M be as defined in Lemma 4.12. Then by that lemma, there are at most $D\log_{2^{31}}(D\tilde{c}) + \left(\frac{(D-1)(D-2)}{2}\right)\left(\log_{2^{31}}(\tilde{s}+1) + \tilde{d}\log_{2^{31}}(M)\right)$ unlucky primes between 2^{30} and $2^{31.5}$. It may be the case that all of these are Fourier primes. Hence we arrive at the following remark.

Remark 4.14. *The probability that a randomly chosen prime between 2^{30} and $2^{31.5}$ of the form $k \cdot 2^r - 1$ for $r \geq R$ is unlucky is at most*

$$\frac{D\log_{2^{31}}(D\tilde{c}) + \left(\frac{(D-1)(D-2)}{2}\right)\left[\log_{2^{31}}(\tilde{s}+1) + \tilde{d}\log_{2^{31}}(M)\right]}{91744290/(2^{R-0.98})},$$

where $\tilde{c}, \tilde{s}, \tilde{d}$, and M are as defined in Lemma 4.11.

The probability given in Remark 4.14 is a very conservative since it is derived from the assumption that $|\det(C)|$ can be factorized into a product of primes which are each between 2^{30} and $2^{31.5}$. This probability equals 1 if $|\det(C)|$ factors into all the Fourier primes between 2^{30} and $2^{31.5}$ of the form $k \cdot 2^r + 1$ for all $r \geq R$, which is highly unlikely. However, in the unlikely event that not enough lucky Fourier primes can be found, it may be necessary to use 63-bit primes on a 64-bit computer.

Example 4.15. Suppose we wish to multiply $f(x)$ and $g(x)$ in $\mathbb{Z}[x_1, \ldots, x_4]/\langle m_1, \ldots, m_4\rangle[x]$ where the m_i's are as in Example 4.13 and $\deg_x(f) + \deg_x(g) = 500$. Since $2^8 < 500 < 2^9$, $R = 9$. Furthermore, recall from Example 4.13 that at most 105 Fourier primes may be unlucky in this case. Therefore, if we randomly pick a random Fourier prime p between 2^{30} and $2^{31.5}$ such that $p - 1$ is divisible by 2^R, the probability that p is unlucky is at most

$$\frac{105}{91744290/2^{9-0.98}} < 0.00029708.$$

Chapter 5

Finding a primitive element (characteristic 0)

Let $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t) \cong \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle$. If the number of extensions t is large, it is expensive to perform multiplication over K due to the overhead of the `recden` representation and a large number of polynomial divisions required. One can avoid this problem by computing a primitive element γ for $K/\mathbb{Q}$. After γ is found, one can express a polynomial in $K[x]$ as a polynomial in $\mathbb{Q}(\gamma)[x] = \mathbb{Q}[z]/\langle m_\gamma(z) \rangle[x]$, a bivariate polynomial in x and z. The `recden` representation of this new polynomial is a nested list of depth 2, which requires less overhead and possibly offers faster arithmetic operations than the nested list of depth $(t + 1)$ required to represent the polynomial in $K[x] = \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t \rangle[x]$.

In this chapter, we present two approaches for finding a primitive element of a field of characteristic 0: a linear algebra approach and a resultant approach. We discuss these methods for fields given as a two-step extension, then generalize to fields given as a tower of more than two extensions.

5.1 Finding a primitive element of $K(\alpha, \beta)$

In what follows, let K be a field of characteristic 0. In order to explain how to compute a primitive element of $K(\alpha, \beta)$, we must state some lemmas.

Lemma 5.1. (Trager [17]) *Let α be algebraic over $\mathbb{Q}$ with minimal polynomial $m_\alpha(y) \in K[y]$ and let $f(x) \in K[x]$ be square-free. Then there exists $c \in K$ such that $\mathrm{norm}_{[K(\alpha)[x]/K[x]]}(f(x - c\alpha))$ is square-free.*

Proof. Let the roots of $f(x)$ over some splitting field be $\beta_1, \ldots, \beta_n$ and let the roots of $m_\alpha(y)$ over a splitting field be $\alpha = \alpha_1, \ldots, \alpha_d$. Since f is square-free, the β_i's are distinct and the α_i's are distinct as well since α is separable over K. Suppose that

$$N(x) := \mathrm{norm}(f(x - c\alpha)) = \prod_{j=1}^{d} f(x - c\alpha_j).$$

If $c = 0$ then $N(x)$ is clearly not square-free, since we can assume that $d > 1$. If $c \neq 0$, the roots of $f(x - c\alpha_j)$ for a fixed j are $\{\beta_i + c\alpha_j,\ i = 1, \ldots, n\}$. $N(x)$ has a multiple root if and only if

$$\beta_r + c\alpha_u = \beta_s + c\alpha_t, \text{ where } r, s \in \{1, \ldots, n\},\ t, u \in \{1, \ldots, d\}, \text{ and } t \neq u.$$

Thus $N(x)$ has a multiple root if and only if c belongs to

$$\begin{aligned} S &= \{0\} \cup \left\{ \frac{\beta_r - \beta_s}{\alpha_t - \alpha_u} : r, s \in \{1, \ldots, n\}, t, u \in \{1, \ldots, d\}, t \neq u \right\} \\ &= \left\{ \frac{\beta_r - \beta_s}{\alpha_t - \alpha_u} : r, s \in \{1, \ldots, n\}, t, u \in \{1, \ldots, d\}, t \neq u \right\}. \end{aligned} \tag{5.1}$$

Since $|S|$ is finite and K has characteristic 0, $K \backslash S$ is non-empty. So there exists $c \in K \backslash S$ for which $N(x)$ is square-free. □

Example 5.2. Let $m_\alpha(y) = y^2 - 2 \in \mathbb{Q}[y]$ and $f(x) = x^2 + 3 \in \mathbb{Q}[x]$. The roots of $f(x)$ over a splitting field are: $\{\beta_1, \beta_2\} = \{\sqrt{3}i, -\sqrt{3}i\}$, and the roots of $m_\alpha(y)$ are $\{\alpha_1, \alpha_2\} = \{\sqrt{2}, -\sqrt{2}\}$. The set S as defined by (5.1) is:

$$S = \left\{ 0, \frac{\sqrt{3}i - (-\sqrt{3}i)}{\sqrt{2} - (-\sqrt{2})}, \frac{-\sqrt{3}i - (\sqrt{3}i)}{\sqrt{2} - (-\sqrt{2})} \right\} = \{0, \frac{1}{2}\sqrt{6}i, -\frac{1}{2}\sqrt{6}i\}.$$

By the proof of Lemma 5.1, $g(x) = \mathrm{norm}(f(x - c\alpha))$ is square-free if and only if $c \notin S$. Since $S \cap \mathbb{Q} = \{0\}$, $\mathrm{norm}(f(x - c\alpha))$ is square-free for every $c \in \mathbb{Q} \backslash \{0\}$.

Alternatively, one can compute the elements in S as follows. By Theorem 1.25,

$$\begin{aligned} g(x) = \text{norm}(f(x - c\alpha)) &= \text{res}_y(f(x - cy), y^2 - 2) \\ &= x^4 - 4c^2x^2 + 6x^2 + 9 + 12c^2 + 4c^4. \end{aligned}$$

Since $g(x)$ is square-free if and only if $\text{res}_x(g(x), g'(x)) \neq 0$ by Remark 1.23, and the roots of

$$\text{res}_x(g(x), g'(x)) = 147456(3 + 2c^2)^2c^4$$

belong to $\hat{S} = \{0, \frac{1}{2}\sqrt{6}i, -\frac{1}{2}\sqrt{6}i\}$, any element $c \in \hat{S}$ yields a non-square-free $\text{norm}(f(x - c\alpha))$. As expected, $S = \hat{S}$.

The following lemma provides an upper bound on the number of $c \in K$ that yields a non-square-free $\text{norm}(f(x - c\alpha))$.

Lemma 5.3. *Let $m_\alpha(y)$ and $f(x)$ be defined as in Lemma 5.1. Furthermore let $d = \deg_y(m_\alpha)$ and $n = \deg_x(f)$. Then*

$$|S| \leq \frac{n(n-1) \cdot d(d-1)}{2} + 1,$$

where S is defined in (5.1).

Proof. There are at most $2\binom{n}{2}$ (non-zero) distinct possibilities for the numerator $\beta_r - \beta_s$, and at most $\binom{d}{2}$ distinct possibilities (up to sign) for the denominator, $\alpha_t - \alpha_u$. Also, $0 \in S$ since the two elements in the numerator can be the same. Hence

$$|S| \leq \left(2\binom{n}{2}\right) \cdot \binom{d}{2} + 1 \quad = \frac{n(n-1) \cdot d(d-1)}{2} + 1.$$

□

We now state a generalization of Lemma 5.1, which is the basis of Trager's algorithm for factoring polynomials in $\mathbb{Q}(\alpha)[x]$.

Theorem 5.4. (Trager [17]) *Let $m_\alpha(y) \in K[y]$ be the minimal polynomial for α and $f(x, \alpha) \in K(\alpha)[x]$ be square-free. Then there exists $c \in K$ for which* $\text{norm}(f(x - c\alpha))$ *is square-free.*

Proof. Let $\alpha = \alpha_1, \ldots, \alpha_d$ be the roots of $m_\alpha(y)$ over a splitting field and $\beta_1, \ldots, \beta_n$ be the (distinct) roots of $f(x, \alpha)$ over a splitting field. Let us write $\text{norm}(f(x, \alpha))$ as

$$\text{norm}(f(x, \alpha)) = \prod_i g_i(x)^{j_i} \in K[x],$$

where each $g_i(x)$ is square-free. Since $f(x, \alpha)$ is square-free and divides $\text{norm}(f(x, \alpha))$, it must divide the square-free polynomial

$$g(x) := \prod_i g_i(x) \in K[x].$$

By applying the proof of Lemma 5.1, there exists $c \in K$ for which $\text{norm}(g(x - c\alpha))$ is square-free. Since the roots of $\text{norm}(f(x - c\alpha)) = \prod_{i=1}^{d} f(x - c\alpha_i)$ belong to

$$R := \{\beta_j + c\alpha_i \,|\, i \in \{1, \ldots, d\}, j \in \{1, \ldots, n\}\},$$

and f divides g, $\beta_1, \ldots, \beta_n$ must be roots of g. That is, every element in R must be a root of $\text{norm}(g(x - c\alpha)) = \prod_{i=1}^{d} g(x - c\alpha_i)$. Hence $\text{norm}(f(x - c\alpha))$ divides $\text{norm}(g(x - c\alpha))$, so if $\text{norm}(g(x - c\alpha))$ is square-free then $\text{norm}(f(x - c\alpha))$ must be square-free as well. □

Lemma 5.5. *Let $m_\alpha(y) \in K[y]$ be the minimal polynomial for α and let $\alpha_1, \alpha_2, \ldots, \alpha_d$ be the roots of $m_\alpha(y) \in K[y]$ over a splitting field. Furthermore let $\beta_1(\alpha), \beta_2(\alpha), \ldots, \beta_n(\alpha)$ be the roots of $f(x, \alpha) \in K(\alpha)[x]$ over a splitting field. The number of elements $c \in K$ for which* $\text{norm}(f(x - c\alpha))$ *is* not *square-free is at most*

$$\frac{n^2 d(d-1)}{2}.$$

Proof. Let

$$N(x) = \text{norm}(f(x - c\alpha)) = \prod_{i=1}^{d} f(x - c\alpha_i, \alpha_i),$$

and let $\{\beta_i(\alpha_j), 1 \leq i \leq n\}$ be the roots of $f(x, \alpha_j)$ for $1 \leq j \leq d$. Since the roots of $f(x - c\alpha_j, \alpha_j)$ are $\{\beta_i(\alpha_j) + c\alpha_j, 1 \leq i \leq n\}$, $N(x)$ has a square-free norm if and only if $\beta_r(\alpha_u) + c\alpha_u = \beta_s(\alpha_t) + c\alpha_t$ for some $r, s \in \{1, \ldots, n\}$ and $t, u \in \{1, \ldots, d\}$ where $t \neq u$. That is, $N(x)$ has a multiple root if and only if c belongs to

$$S = \left\{ \frac{\beta_r(\alpha_u) - \beta_s(\alpha_t)}{\alpha_t - \alpha_u} : r, s \in \{1, \ldots, n\}, t, u \in \{1, \ldots, d\}, t \neq u \right\}.$$

There are $\begin{pmatrix} d \\ 2 \end{pmatrix}$ distinct choices for the denominator. For fixed α_t and α_u, there are n choices for $\beta_r(\alpha_u)$, as we must choose from $\{\beta_1(\alpha_u), \beta_1(\alpha_u), \ldots, \beta_n(\alpha_u)\}$. Moreover, there are n choices for $\beta_s(\alpha_t)$, since we must choose from $\{\beta_1(\alpha_t), \beta_2(\alpha_t), \ldots, \beta_n(\alpha_t)\}$. Thus there are at most n^2 distinct possibilities for the numerator $\beta_r(\alpha_u) - \beta_s(\alpha_t)$. In summary,

$$|S| \leq n^2 \cdot \begin{pmatrix} d \\ 2 \end{pmatrix} = \frac{n^2 d(d-1)}{2}.$$

□

Example 5.6. Let α be algebraic over $\mathbb{Q}$ with minimal polynomial $m_\alpha(y) = y^2 - 2 \in \mathbb{Q}[y]$ and let $f(x, \alpha) = x^2 - \alpha + 1 \in \mathbb{Q}(\alpha)[x]$. Then $n = \deg_y(m_\alpha) = 2$ and $d = \deg_x(f) = 2$. By Lemma 5.5, the number of $c \in K$ for which $\text{norm}(f(x - c\alpha))$ is not square-free is at most

$$\frac{n^2 d(d-1)}{2} = \frac{2^2 \cdot 2 \cdot 1}{2} = 4.$$

We verify this by explicitly computing the elements in $\mathbb{Q}$ for which $\text{norm}(f(x - c\alpha))$ is not square-free. The roots of $m_\alpha(y)$ are $\{\alpha = \alpha_1, \alpha_2\} = \{\sqrt{2}, -\sqrt{2}\}$, the roots of $f(x, \alpha) = f(x, \alpha_1)$ are

$$\{\beta_1(\alpha_1), \beta_2(\alpha_1)\} = \left\{\sqrt{-1+\sqrt{2}}, -\sqrt{-1+\sqrt{2}}\right\},$$

and the roots of $f(x, \alpha_2)$ are

$$\{\beta_1(\alpha_2), \beta_2(\alpha_2)\} = \left\{i\sqrt{1+\sqrt{2}}, -i\sqrt{1+\sqrt{2}}\right\}.$$

By the proof of Lemma 5.5, $f(x - c\alpha)$ has a multiple root if and only if c belongs to

$$\begin{aligned} S &= \left\{\frac{\beta_1(\alpha_1) - \beta_2(\alpha_2)}{\alpha_2 - \alpha_1}, \frac{\beta_2(\alpha_1) - \beta_1(\alpha_2)}{\alpha_2 - \alpha_1}, \frac{\beta_1(\alpha_1) - \beta_1(\alpha_2)}{\alpha_2 - \alpha_1}, \frac{\beta_2(\alpha_1) - \beta_2(\alpha_2)}{\alpha_2 - \alpha_1}\right\} \\ &= \left\{\pm\frac{\sqrt{-1+\sqrt{2}} + i\sqrt{1+\sqrt{2}}}{-2\sqrt{2}}, \quad \pm\frac{\sqrt{-1+\sqrt{2}} - i\sqrt{1+\sqrt{2}}}{-2\sqrt{2}}\right\}. \end{aligned}$$

Observe that $|S| = 4$ as expected, and $S \cap \mathbb{Q} = \varnothing$, so every $c \in \mathbb{Q}$ yields a square-free $\text{norm}(f(x - c\alpha))$.

Alternatively, by Theorem 1.25,

$$g(x) = \text{norm}(f(x-c\alpha)) = \text{res}_y(f(x-cy), y^2-2) = x^4+2\,x^2-4\,x^2c^2-1+4\,c^2+4\,c^4-8\,xc$$

is square-free if and only if $\text{res}_x(g(x), g'(x)) \neq 0$. Since

$$r(c) = \text{res}_x(g(x), g'(x)) = -1024(1 + 4c^2 + 8c^4)^2,$$

$r(c)$ has at most four distinct roots in K. With some algebra, one can show that the elements in S are exactly the roots of $r(c)$.

In practice, we will choose c from $\mathbb{Z}_{\geq 0}$ for simplicity. Rather than explicitly determining S and choosing c from $\mathbb{Z}_{\geq 0}\backslash S$ which can be computationally expensive, Trager's algorithm ([17]) counts up by one starting from $c = 0$ until he find a square-free $\text{norm}(f(x - c\alpha))$.

Let us return to the problem of finding γ, algebraic over K, such that $K(\gamma) \cong K(\alpha, \beta)$. To that end, the following lemma and theorem will prove useful.

Lemma 5.7. (Trager [17]) *Let $m_\alpha(y) \in K[y]$ be the minimal polynomial for α and let β be a root of square-free $g(x, \alpha) \in K(\alpha)[x]$. If $\text{norm}(g(x, \alpha))$ is square-free then*

$$\gcd(m_\alpha(y), g(\beta, y)) = y - \alpha \in K(\beta)[y].$$

Proof. We wish to show that α is the only common root of $g(\beta, y)$ and $m_\alpha(y)$. Let $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_d$ be the roots of $m_\alpha(y)$ over a splitting field and suppose that $g(\beta, \alpha_j) = 0$ for some $\alpha_j \neq \alpha_1$. But this implies that β is a multiple root of $\prod_{i=1}^{d} g(x, \alpha_i) = \text{norm}(g(x, \alpha))$, which contradicts the assumption that $\text{norm}(g(x, \alpha))$ is square-free. Thus the only common root of $g(\beta, y)$ and $m_\alpha(y)$ must be α. This proves that $\gcd(g(\beta, y), m_\alpha(y)) = y - \alpha$. □

Theorem 5.8. (Trager [17]) *Let $m_\alpha(y) \in K[y]$ and $m_\beta(x, \alpha) \in K(\alpha)[x]$ be the minimal polynomials for α over K and β over $K(\alpha)$, respectively. If $\text{norm}(m_\beta(x, \alpha))$ is square-free then*

$$K(\alpha, \beta) = K(\beta).$$

Proof. Note that $K(\beta) \subseteq K(\alpha, \beta)$. Therefore it is sufficient to show that $\alpha \in K(\beta)$. By Lemma 5.7,

$$\gcd(m_\alpha(y), m_\beta(\beta, y)) = y - \alpha. \tag{5.2}$$

Since $m_\alpha(x)$ and $m_\beta(\beta, y)$ are polynomials over $K(\beta)$, their gcd must also be over $K(\beta)$; that is, $\alpha \in K(\beta)$. That is, the solution to (5.2) is the normal representation of α in $K(\beta)$. □

In what follows, we let $m_\alpha(y) \in K[y]$ be the minimal polynomial for α and $m_\beta(x, \alpha) \in K(\alpha)[x]$ be the minimal polynomial for β. Further, we let $g(x, \alpha) = m_\beta(x - c\alpha, \alpha)$ where $c \in K$ is chosen so that $\text{norm}(g(x, \alpha))$ is square-free. Then we can show the following.

Theorem 5.9. *If $f(x, \alpha)$ is an irreducible polynomial over $K(\alpha)$ then*

$$\text{norm}(f(x, \alpha)) = h(x)^k$$

where $h(x)$ is an irreducible polynomial over K and $k \in \mathbb{Z}^+$.

Proof. See Trager [17]. □

Observe that since $m_\beta(x, \alpha)$ is irreducible over $K(\alpha)$, the polynomial $g(x, \alpha)$ must also be irreducible over $K(\alpha)$ by Theorem 5.9. Furthermore, $\gamma := \beta + c\alpha$ is a root of g, so we can apply Theorem 5.8 with $m_\alpha(y)$ and $g(x, \alpha)$ to conclude that $K(\alpha, \gamma) = K(\gamma)$. But since $K(\alpha, \gamma) = K(\alpha, \beta + c\alpha) = K(\alpha, \beta)$, we have

$$K(\alpha, \beta) = K(\gamma).$$

That is, γ is a primitive element of $K(\alpha, \beta)$.

Remark 5.10. *γ is a root of the irreducible monic polynomial $N(x) = \text{norm}\left(m_\beta(x - c\alpha, \alpha)\right) \in K[x]$, so $N(x)$ must be the minimal polynomial for γ over K.*

We now discuss two methods of finding a primitive element γ of $K(\alpha, \beta)$, namely a linear algebra approach, and a resultant approach.

5.2 Finding a primitive element using linear algebra

As previously mentioned, to find a primitive element $\gamma = \beta + c\alpha$ of $K(\alpha, \beta)$, we count up by one starting from $c = 0$ until we find a c which yields a square-free norm$(m_\beta(x - c\alpha))$. To test if a chosen $\beta + c\alpha$ yields a primitive element of $K(\alpha, \beta)$, one can find the change-of-basis matrix from $K(\alpha, \beta)$ to $K(\gamma)$ (or from $K(\gamma)$ to $K(\alpha, \beta)$) and test that it is invertible (Chapter 4). We illustrate this with an example.

Example 5.11. Let us find a primitive element γ of $\mathbb{Q}(\alpha, \beta) \cong \mathbb{Q}[y, x]/\langle m_\alpha(y), m_\beta(x)\rangle$ where $m_\alpha(y) = y^2 - 5 \in \mathbb{Q}[y]$ and $m_\beta(x) = x^2 - 2 \in \mathbb{Q}[x]$. Let $c = 0$. Then $\gamma = \beta + 0 \cdot \alpha = \beta$. A basis for $\mathbb{Q}(\gamma)$ is $B_\gamma = \{1, \gamma, \gamma^2, \gamma^3\}$ and a basis for $\mathbb{Q}(\alpha, \beta)$ is $B_\alpha = \{1, \alpha, \beta, \alpha\beta\}$. Note that

$$\gamma^0 = 1, \ \ \gamma^1 = \beta, \ \ \gamma^2 = \beta^2 = 2, \quad \text{and } \gamma^3 = \beta^3 = \beta \cdot \beta^2 = 2\beta.$$

Hence the change-of-basis matrix from B_γ to B_α is

$$C = \begin{array}{c} \\ 1 \\ \alpha \\ \beta \\ \alpha\beta \end{array} \begin{array}{c} \begin{array}{cccc} 1 & \gamma & \gamma^2 & \gamma^3 \end{array} \\ \begin{pmatrix} 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Clearly C is not invertible, so $\beta + 0 \cdot \alpha$ is *not* a primitive element for $\mathbb{Q}(\alpha, \beta)$. Thus we let $c = 1$. Then the change-of-basis matrix from B_γ to B_α is

$$C = \begin{array}{c} \\ 1 \\ \alpha \\ \beta \\ \alpha\beta \end{array} \begin{array}{c} \begin{array}{cccc} 1 & \gamma & \gamma^2 & \gamma^3 \end{array} \\ \begin{pmatrix} 1 & 0 & 7 & 0 \\ 0 & 1 & 0 & 11 \\ 0 & 0 & 0 & 17 \\ 0 & 1 & 2 & 0 \end{pmatrix} \end{array}. \tag{5.3}$$

One can verify that C is invertible. Thus $\gamma = \beta + 1 \cdot \alpha$ is a primitive element of $\mathbb{Q}(\alpha, \beta)$.

To express a polynomial in $\mathbb{Q}(\gamma)[x]$ as a `recden` polynomial, one must compute $m_\gamma(z) \in \mathbb{Q}[z]$, the minimal polynomial for γ. We explain an efficient method for this.

Lemma 5.12. *Let $K(\gamma)$ be an extension field and $D = [K(\gamma) : K]$. If $\gamma^D = a_0 + a_1\gamma + \cdots + a_{D-1}\gamma^{D-1} \in K(\gamma)$, then $m_\gamma(z)$, the minimal polynomial for γ, is*

$$m_\gamma(z) = -a_0 - a_1 z + \cdots - a_{D-1}z^{D-1} + z^D.$$

Proof. Let

$$m_\gamma(z) = m_0 + m_1 z + m_2 z^2 + \cdots + m_{D-1}z^{D-1} + z^D \in K[z].$$

By definition, $m_\gamma(\gamma) = m_0 + m_1\gamma + m_2\gamma^2 + \cdots + \gamma^D = 0$. Rearranging, we get

$$-\gamma^D = m_0 + m_1\gamma + m_2\gamma^2 + \cdots + m_{D-1}\gamma^{D-1}.$$

On the other hand, by Theorem 1.7 we have

$$-\gamma^D = a_0 + a_1\gamma + \cdots + a_{D-1}\gamma^{D-1} \in K(\gamma),\ a_0, \ldots, a_{d-1} \in K.$$

So

$$-(a_0 + a_1\gamma + \cdots + a_{D-1}\gamma^{D-1}) = m_0 + m_1\gamma + m_2\gamma^2 + \cdots + m_{D-1}\gamma^{D-1}.$$

Since $\{1, \gamma^1, \ldots, \gamma^{D-1}\}$ is a basis for $K(\gamma)$, we must have $m_i = -a_i$ for $0 \le i \le D-1$. That is, $m_\gamma(z) = -a_0 - a_1 z - \cdots - a_{D-1}z^{D-1} + z^D$, as required. □

Lemma 5.12 implies that one can compute $m_\gamma(z)$ by finding γ^D expressed as a linear combination of the γ^i's, $0 \le i \le D-1$. Observe that when we computed the change-of-basis matrix C, we have already computed γ^{D-1} in terms of the elements in B_α. Thus at this stage, $m_\gamma(z)$ can be found via a single multiplication $\gamma^{D-1} \cdot \gamma = \gamma^D$ and a single matrix-vector multiplication of C^{-1} by the CDR of γ^D to obtain $m_\gamma(z)$.

Example 5.13. Continuing from Example 5.11, let us find $m_\gamma(z) \in \mathbb{Q}[z]$. Since $D = [\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha, \beta) : \mathbb{Q}] = 4$ and $\gamma^3 = 11\alpha + 17\beta$, we have

$$\gamma^4 = \gamma \cdot \gamma^3 = (\alpha + \beta)(11\alpha + 17\beta) = 89 + 28\alpha\beta.$$

To represent $\gamma^4 = 89 + 28\alpha\beta$ as an element in $\mathbb{Q}(\gamma)$, we multiply C^{-1} (the change-of-basis matrix from B_α to B_γ) by the CDR of γ^4, which is $[89, 0, 0, 28]^T$, and C is given

by (5.3):

$$C^{-1} \cdot \begin{bmatrix} 89 \\ 0 \\ 0 \\ 28 \end{bmatrix} = \begin{bmatrix} 1 & \frac{7}{2} & -\frac{77}{34} & -\frac{7}{2} \\ 0 & 1 & -\frac{11}{17} & 0 \\ 0 & -\frac{1}{2} & \frac{11}{34} & \frac{1}{2} \\ 0 & 0 & \frac{1}{17} & 0 \end{bmatrix} \cdot \begin{bmatrix} 89 \\ 0 \\ 0 \\ 28 \end{bmatrix} = \begin{bmatrix} -9 \\ 0 \\ 14 \\ 0 \end{bmatrix}.$$

Hence $m_\gamma(z) = -9 + 14z^2 + z^4$.

5.3 Finding a primitive element using resultants

A second method for determining a primitive element and its minimal polynomial of $K(\alpha, \beta)$ is by using resultants and gcds. This method is based on the fact that norms can be computed using resultants (Theorem 1.25). We illustrate this method with an example.

Example 5.14. Let us find a primitive element $\gamma = \beta + c\alpha$ of $\mathbb{Q}(\alpha, \beta)$ where $m_\alpha(x) = x^2 - 5 \in \mathbb{Q}[x]$ and $m_\beta(x) = x^2 - 2 \in \mathbb{Q}[x]$. These polynomials are the same as in Example 5.11. When $c = 0$, we have

$$\text{norm}(m_\beta(x - c\alpha, \alpha)) = \text{norm}(m_\beta(x - 0 \cdot \alpha, \alpha)) = \text{res}_y(m_\beta(x, y), m_\alpha(y)) = (x^2 - 2)^2.$$

This polynomial is not square-free, so we recompute the norm of $m_\beta(x - c\alpha, \alpha)$ for $c = 1$:

$$N(x) = \text{norm}(m_\beta(x - \alpha, \alpha)) = \text{res}_y((x - y)^2 - 2,\ y^2 - 5) = x^4 - 14x^2 + 9.$$

$N(x)$ is square-free, so $\gamma = \beta + 1 \cdot \alpha$ is a primitive element of $\mathbb{Q}(\alpha, \beta)$ whose minimal polynomial over $\mathbb{Q}$ is $N(x)$. This result is consistent with that obtained using the linear algebra method in Example 5.11.

5.3.1 Finding $\alpha(\gamma)$ and $\beta(\gamma)$

Suppose that we have found a primitive element $\gamma = \beta + c\alpha$ satisfying $K(\alpha, \beta) = K(\gamma)$. To convert $f \in K(\alpha, \beta)[x]$ to a polynomial in $K(\gamma)[x]$, we need to substitute

the αs and βs in f with their normal representations $\alpha(\gamma), \beta(\gamma) \in K(\gamma)$, respectively. In the linear algebra approach, we can do this by inverting the change-of-basis matrix C and performing a series of matrix-vector multiplications.

In this section,we explain a different method for computing $\alpha(\gamma)$ and $\beta(\gamma)$, which requires computing a gcd, and avoids computing the change-of-basis matrix. By the proof of Theorem 5.8, the solution to the linear equation

$$\gcd(m_\beta(\gamma - cy, y), m_\alpha(y)) = 0$$

is equal to $\alpha(\gamma)$. Furthermore, since $\gamma = \beta + c\alpha = \beta + c\alpha(\gamma)$, $\beta(\gamma)$ can be found by a simple formula:

$$\beta(\gamma) = \gamma - c\alpha(\gamma).$$

We illustrate this with an example.

Example 5.15. Continuing with Example 5.14, let us find $\alpha(\gamma), \beta(\gamma) \in \mathbb{Q}(\gamma)$. We have

$$\begin{aligned} \gcd(m_\beta(\gamma - y, y), m_\alpha(y)) &= \gcd((\gamma - y)^4 - y^2(\gamma - y)^2 - 2, y^4 - 2) \\ &= y + \frac{16949}{4628}\gamma + \frac{5883}{4628}\gamma^5 + \frac{925}{9256}\gamma^9 - \frac{87}{37024}\gamma^{13}. \end{aligned} \tag{5.4}$$

Hence

$$\alpha(\gamma) = -\frac{16949}{4628}\gamma - \frac{5883}{4628}\gamma^5 - \frac{925}{9256}\gamma^9 + \frac{87}{37024}\gamma^{13} \quad \text{and}$$

$$\beta(\gamma) = \gamma - 1 \cdot \alpha(\gamma) = \frac{-87}{37024}\gamma^{13} + \frac{925}{9256}\gamma^9 + \frac{5883}{4628}\gamma^5 + \frac{21577}{4628}\gamma.$$

Notice the expression swell that occurs from computing with rationals. We will eliminate this swell by working modulo a prime (Chapter 6).

5.4 Algorithms

We present the algorithms for finding a primitive element of $K(\alpha, \beta)$ using resultants and gcds. Algorithm **sqfr_norm** finds $c \in \mathbb{Z}$ for which $\text{norm}(m_\alpha(x - c\alpha, \alpha))$ is square-free, and Algorithm **prim_elt** returns the minimal polynomial for the primitive element γ of $K(\alpha, \beta)$ and the normal representations $\alpha(\gamma), \beta(\gamma) \in K(\gamma)$.

Algorithm 5.1 : **sqfr_norm**$(m_\beta(x,\alpha), m_\alpha(y))$

Input: $m_\beta(x,\alpha) \in K(\alpha)[x]$, and $m_\alpha(y) \in K[y]$, the minimal polynomials for β over $K(\alpha)$ and α over K respectively, where K is a field of characteristic 0.

Output: $c \in \mathbb{Z}$, $g(x,\alpha) = m_\beta(x - c\alpha) \in K(\alpha)[x]$, and square-free $N(x) = \text{norm}(g(x,\alpha)) \in K[x]$.

1: $c \leftarrow 0$; $g(x,\alpha) \leftarrow m_\beta(x,\alpha)$;
2: **while** true **do**
3: $g(x,\alpha) \leftarrow g(x - c \cdot \alpha, \alpha)$;
4: $N(x) \leftarrow \text{res}_y(g(x,y), m_\alpha(y))$; $\{N(x) \in K[x]\}$
5: **if** $\deg(\gcd(N(x), N'(x))) = 0$ **then**
6: **return** $c, g(x,\alpha), N(x)$; $\{N(x)$ is the minimal polynomial for $\beta + c\alpha$ by Rem. 5.10$\}$
7: **else**
8: $c \leftarrow c + 1$;
9: **end if**
10: **end while**

Algorithm 5.2 : **prim_elt**$(m_\beta(x,\alpha), m_\alpha(y))$

Input: $m_\beta(x,\alpha) \in K(\alpha)[x]$, and $m_\alpha(y) \in K[y]$, minimal polynomials for β over $K(\alpha)$ and α over K respectively, where K is a field of characteristic 0.

Output: $c \in \mathbb{Z}$, $N(x) \in K[x]$, minimal polynomial for γ where $K(\alpha,\beta) = K(\gamma)$, and $A(\gamma) \in K(\gamma), B(\gamma) \in K(\gamma)$, the normal representations of α and β, respectively.

1: $c, g(x,\alpha), N(x) \leftarrow$ **sqfr_norm**$(m_\beta(x,\alpha), m_\alpha(y))$;
2: $h(\gamma, y) \leftarrow$ the monic $\gcd(g(\gamma,y), m_\alpha(y)) \in K(\gamma)[y]$ where $N(\gamma) = 0$;
$\{h(y) = y + a(\gamma)$ by Lemma 5.7$\}$
3: $A(\gamma) \leftarrow -a(\gamma)$; $B(\gamma) \leftarrow \gamma - c \cdot A(\gamma)$;
4: **return** c, $N(x)$, $A(\gamma)$, $B(\gamma)$;

5.5 Towers with more than two steps

We now generalize the resultant approach to apply to fields given as a tower of more than two extensions ($t > 2$). To find a primitive element γ of such fields, one can find a primitive element of two extensions at a time repeatedly as follows:

$$\begin{aligned} K(\alpha_1, \alpha_2, \ldots, \alpha_t) &= K(\alpha_1, \ldots, \alpha_{t-3}, \alpha_{t-2})(\alpha_{t-1}, \alpha_t) \\ &\cong K(\alpha_1, \ldots, \alpha_{t-3}, \alpha_{t-2})(\gamma_{t-1}) \\ &= K(\alpha_1, \ldots \alpha_{t-3})(\alpha_{t-2}, \gamma_{t-1}) \\ &\cong K(\alpha_1, \ldots, \alpha_{t-3})(\gamma_{t-2}) \\ &\quad \vdots \\ &\cong K(\alpha_1, \gamma_2) \\ &\cong K(\gamma). \end{aligned} \tag{5.5}$$

Remark 5.16. *One can alternatively collapse the extensions "bottom-up" as follows:*

$$K(\alpha_1, \alpha_2, \ldots, \alpha_t) \cong K(\gamma_1, \alpha_3, \ldots, \alpha_t) \cong K(\gamma_2, \alpha_4, \ldots, \alpha_t) \cdots \cong K(\gamma_{t-2}, \alpha_t) \cong K(\gamma).$$

The complexity of this method is comparable to the "top-down" method we proposed above.

To find the normal representations of all the α_i's in $K(\gamma)$, one needs to perform extra computations than in the two extension case. We illustrate this for the case $t = 3$.

Example 5.17. Let us find a primitive element of $\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) \cong \mathbb{Q}[z, y, x]/\langle m_1, m_2, m_3 \rangle$ where

$$\begin{aligned} m_1(z) &= z^2 - 2 \in \mathbb{Q}[z], \\ m_2(y) &= y^2 - \alpha_1 y + 11 \in \mathbb{Q}(\alpha_1)[y], \quad \text{and} \\ m_3(x) &= x^2 + 2\alpha_1\alpha_2 - 3 \in \mathbb{Q}(\alpha_1, \alpha_2)[x]. \end{aligned}$$

Let us find γ_1, algebraic over $K_1 := \mathbb{Q}(\alpha_1)$ such that $K_1(\alpha_2, \alpha_3) = K_1(\gamma_1)$. If $c = 0$, then $m_3(x - c\alpha_2, \alpha_2)$ is:

$$\begin{aligned} N_1(x, \alpha_1) := \operatorname{norm}(m_3(x, \alpha_2)) &= \operatorname{res}_y(m_3(x, y), m_2(y)) \\ &= \operatorname{res}_y(x^2 + 2\alpha_1 y - 3, y^2 - y\alpha_1 + 11) \\ &= x^4 - 2x^2 + 85 \in K_1[x]. \end{aligned} \tag{5.6}$$

Since $N_1(x, \alpha_1)$ is square-free, we conclude that $\gamma_1 = \alpha_2 + 0 \cdot \alpha_3$ satisfies $K_1(\alpha_2, \alpha_3) = K_1(\gamma_1)$ and $N_1(x, \alpha_1) \in K_1[x] = \mathbb{Q}(\alpha_1)[x]$ is the minimal polynomial for γ_1. Since

$$\gcd(m_3(\gamma_1, y), m_2(y)) = y - \frac{3}{4}\alpha_1 + \frac{1}{4}\alpha_1\gamma_1^2,$$

we conclude that the normal representations of α_2 and α_3 in $K_1(\gamma_1)$ are

$$\alpha_2(\gamma_1) = \frac{3}{4}\alpha_1 - \frac{1}{4}\alpha_1\gamma_1^2 \in K_1(\gamma_1), \text{ and } \alpha_3(\gamma_1) = \gamma_1 - 0 \cdot \alpha_2 = \gamma_1 \in K_1(\gamma_1).$$

Now we find γ, algebraic over $\mathbb{Q}$, such that $\mathbb{Q}(\gamma) = \mathbb{Q}(\alpha_1, \gamma_1)$. Note that

$$\text{norm}(N_1(x, \alpha_1)) = \text{res}_y(N_1(x), m_1(y)) = \text{res}_y(x^4 - 2x^2 + 85, y^2 - 2) = (x^4 - 2x^2 + 85)^2$$

is not square-free. Thus we try $c = 1$. The norm of $N(x - \alpha_1, \alpha_1)$ is

$$\begin{aligned} N(x) := \text{norm}(N_1(x - \alpha_1, \alpha_1)) &= \text{res}_y((x-y)^4 - 2(x-y)^2 + 85, y^2 - 2) \\ &= x^8 - 12x^6 + 206x^4 + 1668x^2 + 7225 \in \mathbb{Q}[x]. \end{aligned}$$

$N(x)$ is square-free. Hence

$$\mathbb{Q}(\gamma) \cong \mathbb{Q}(\alpha_1, \alpha_2, \alpha_3) \text{ where } \gamma = \gamma_1 + \alpha_1 = \alpha_1 + (\alpha_2 + 0 \cdot \alpha_3),$$

and the minimal polynomial for γ over $\mathbb{Q}$ is $N(x)$. Let us now find the normal representations of α_2 and α_1 in $\mathbb{Q}(\gamma)$. Since

$$\gcd(N(\gamma - y, y), m_1(y)) = y - \frac{1}{25840}\left(\gamma + 2959\gamma^3 - 193\gamma^5 + 9\gamma^7\right),$$

we have

$$\begin{aligned} \alpha_1(\gamma) &= \tfrac{1}{25840}\left(\gamma + 2959\gamma^3 - 193\gamma^5 + 9\gamma^7\right) \text{ and} \\ \gamma_1(\gamma) &= \gamma - 1 \cdot \alpha_1(\gamma) = \tfrac{1}{25840}\left(-9\gamma^7 + 22983\gamma - 2959\gamma^3 + 193\gamma^5\right). \end{aligned}$$

Finally, recall that $\alpha_3(\gamma_1) = \gamma_1 - 0 \cdot \alpha_2 = \gamma_1$. So

$$\alpha_3(\gamma) = \gamma_1(\gamma) = \tfrac{1}{25840}\left(-9\gamma^7 + 22983\gamma - 2959\gamma^3 + 193\gamma^5\right).$$

Moreover, recall that $\alpha_2(\gamma_1) = \frac{3}{4}\alpha_1 - \frac{1}{4}\alpha_1 \cdot \gamma_1^2$. Thus

$$\begin{aligned} \alpha_2(\gamma) &= \tfrac{3}{4}\alpha_1(\gamma) - \tfrac{1}{4}\alpha_1(\gamma) \cdot \gamma_1(\gamma)^2 \\ &= \tfrac{3}{4}\left(\tfrac{1}{25840}\left(\gamma + 2959\gamma^3 - 193\gamma^5 + 9\gamma^7\right)\right) - \tfrac{1}{4}\left(\tfrac{1}{25840}\left(\gamma + 2959\gamma^3 - 193\gamma^5 + 9\gamma^7\right)\right) \cdot \\ &\quad \left(\tfrac{1}{25840}\left(-9\gamma^7 + 22983\gamma - 2959\gamma^3 + 193\gamma^5\right)\right)^2 \mod \langle N(\gamma)\rangle \\ &= \tfrac{-1}{981920}\left(111909\gamma^3 - 9151\gamma^5 + 815\gamma^7 + 1736191\gamma\right). \end{aligned}$$

In general, if the field is a t-step extension where t is greater than 2, and we execute Algorithm **prim_elt** $(t-1)$ times to compute a primitive element γ of the field, we obtain the normal representations

$$\alpha_1(\gamma),\ \alpha_2(\gamma_1),\ \alpha_3(\gamma_1),\ \ldots,\ \alpha_{t-2}(\gamma_{t-2}),\ \alpha_{t-1}(\gamma_{t-1}),\ \alpha_t(\gamma_{t-1}) \quad \text{and}$$

$$\gamma,\ \gamma_1(\gamma),\ \gamma_2(\gamma_1),\ \ldots,\ \gamma_{t-1}(\gamma_{t-2}),$$

where γ_i's are as in (5.5). Thus we must convert $\alpha_2(\gamma_1), \ldots, \alpha_t(\gamma_{t-1})$ to $\alpha_2(\gamma), \ldots, \alpha_t(\gamma) \in K(\gamma)$. Note that for a fixed $j \in \{2, \ldots, t\}$,

$$\alpha_j(\gamma_{j-1}) \in K(\alpha_1, \ldots, \alpha_{j-2}, \gamma_{j-1}).$$

Hence for each $\alpha_j(\gamma_{j-1})$, we must make the substitutions

$$\alpha_i \leftarrow \alpha_i(\gamma), 1 \le i \le j-2 \ \text{ and } \ \gamma_{j-1} \leftarrow \gamma_{j-1}(\gamma).$$

Chapter 6

Finding a primitive element (characteristic p)

In what follows, we let $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_r) \cong \mathbb{Q}[u_1, \ldots, u_r]/\langle m_{\alpha_1}, \ldots, m_{\alpha_r} \rangle$ where $m_{\alpha_i}(u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ is the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \leq i \leq r$ and $m_{\alpha_1}(u_1) \in \mathbb{Q}[u_1]$ is the minimal polynomial for α_1 over $\mathbb{Q}$.

Consider

$$K(\alpha, \beta) \cong K[y, x]/\langle m_1, m_2 \rangle,$$

where $m_1 = m_1(y) \in K[y]$ and $m_2 = m_2(x) \in K(\alpha)[x]$ are the minimal polynomials of α over K and β over $K(\alpha)$ respectively. Furthermore, let $f, g \in K(\alpha, \beta)[x]$ be the polynomials we wish to multiply. As we have seen in Example 5.13, performing arithmetic in $\mathbb{Q}$ leads to a blow-up of coefficients. To control the growth, we work modulo primes.

Rather than working over $K(\alpha, \beta)$, we will work over the finite quotient ring

$$K_p[\overline{\alpha}, \overline{\beta}] \cong K_p[y, x]/\langle M_1, M_2 \rangle,$$

where p is a non-bad prime (so that $\Phi_p(f), \Phi_p(g), \Phi_p(\alpha_1), \ldots, \Phi_p(\alpha_r), \Phi_p(\alpha), \Phi_p(\beta)$ are all defined), $\overline{\alpha} := \Phi_p(\alpha)$, $\overline{\beta} := \Phi_p(\beta)$, $M_1 := M_1(y) = \Phi_p(m_1) \in K_p[y]$, and $M_2 := M_2(x, \overline{\alpha}) = \Phi_p(m_2) \in K_p[\overline{\alpha}][x]$. For many primes this ring is not a field, so the theorems mentioned in Chapter 5 may not apply. Nevertheless, we can implement modifications to Algorithm **prim_elt** (and Algorithm **sqfr_norm**) so that it takes

as input the ring $K_p[\overline{\alpha}, \overline{\beta}]$ and returns

$$\Phi_p(m_\gamma(x)),\ \ \overline{\alpha}(\overline{\gamma}) = \Phi_p(\alpha(\gamma)) \text{ and } \overline{\beta}(\overline{\gamma}) = \Phi_p(\beta(\gamma)),$$

where γ is a primitive element of $K(\alpha, \beta)$, $\overline{\gamma} = \Phi_p(\gamma)$, and $\alpha(\gamma), \beta(\gamma)$ are normal representations of α and β respectively in $K(\gamma)$. In this chapter, we examine the modifications that are necessary for the algorithms to work modulo p.

6.1 Modifications to Algorithm sqfr_norm

In what follows, let p be a non-bad prime and denote by K_p the ring

$$K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r] \cong \mathbb{F}_p[u_1, \ldots, u_r]/\langle \Phi_p(m_{\alpha_1}), \ldots, \Phi_p(m_{\alpha_r})\rangle.$$

We first describe modifications necessary for Algorithm **sqfr_norm** to run modulo p, specifically when the input ring is $K_p[\overline{\alpha}, \overline{\beta}] \cong K_p[y, x]/\langle M_1, M_2\rangle$. Recall that Algorithm **sqfr_norm** finds $c \in \mathbb{Z}$ such that $\text{res}_y(m_2(x - cy, y), m_1(y)) \in K[x]$ is square-free. In this section, we show the algorithm may fail over $K_p[\overline{\alpha}, \overline{\beta}]$ if:

(i) division by a zero divisor is encountered,

(ii) $M_1(y) = \Phi_p(m_1(y))$ or $M_2(x, \overline{\alpha}) = \Phi_p(m_2(x, \alpha))$ is not square-free over K_p or $K_p[\overline{\alpha}]$ respectively, or

(iii) p is not large enough for a suitable c to be found.

We now discuss how to handle each case.

6.1.1 Handling zero divisors

Executing Algorithm **sqfr_norm** modulo p may fail if a zero divisor is encountered while computing the resultant (Line 4) using evaluation and interpolation (Section 7.2) or the gcd (Line 5) using the Euclidean algorithm. Normally, division by a zero divisor is encountered only for certain values of c, in which case we can simply choose a different c and re-enter the while-loop. However, there may exist a prime p in which every $c \in \mathbb{F}_p$ results in a division by a zero divisor. In Section 7.1 we provide an example of such a case, and explain how to handle it.

6.1.2 Handling non-square-free M_1 or M_2

Certainly m_1 may be square-free over K, but not over K_p. For example, if $K = \mathbb{Q}$ and $K_p = \mathbb{F}_5$, then $m_1 = x^2 - 5 \in \mathbb{Q}[x]$ is square-free over K but $M_1 = \Phi_p(m_1) = x^2$ is not square-free over K_p. We insist that M_1 and M_2 must be square-free over K_p and $K_p[\overline{\alpha}]$ respectively. To see why this condition is required, we refer to the following lemma.

Lemma 6.1. *Let p be a non-bad prime, $\deg_y(m_1) > 1$, and $\deg_x(m_2) > 1$. If either $M_1(y) = \Phi_p(m_1) \in K_p[y]$ or $M_2(x) = \Phi_p(m_2) \in K_p[\overline{\alpha}][x]$ is not square-free over K_p and $K_p[\overline{\alpha}]$ respectively, then there does not exist $c \in \mathbb{F}_p$ for which $N_p(x) = \operatorname{res}_y(M_2(x - cy, y), M_1(y))$ is square-free over K_p.*

Proof. Let $d_1 = \deg_y(m_1)$ and $d_2 = \deg_x(m_2)$. Because m_1 and m_2 are monic in y and x respectively, it follows that $\deg_y(M_1) = d_1 > 1$ and $\deg_x(M_2) = d_2 > 1$.

Let $\alpha_1, \ldots, \alpha_{d_1}$ be the roots of $m_1(x) \in K[x]$ over a splitting field and suppose that $M_1(x)$ is not square-free. Then we must have

$$\alpha_k = \alpha_r \text{ where } 1 \leq k < r \leq d_1.$$

If $k = \deg_y(M_2(x - cy, y))$, then by Theorem 1.19 (i) and Theorem 6.2,

$$\begin{aligned}
N_p(x) &= \operatorname{res}_y(M_2(x - cy, y), M_1(y)) \\
&= (-1)^{d_2 d_1} \cdot \operatorname{res}_y(M_1(y), M_2(x - cy, y)) \\
&= (-1)^{d_2 d_1} \cdot \operatorname{res}_y(\Phi_p(m_1(y)), \Phi_p(m_2(x - cy, y))) \\
&= \frac{(-1)^{d_2 d_1}}{\operatorname{lcoeff}_y(m_1)^{d_2 - k}} \cdot \Phi_p\left(\operatorname{res}_y\left(m_1(y), m_2(x - cy, y)\right)\right) \\
&= \frac{(-1)^{d_2 d_1}}{1^{d_2 - k}} \cdot \Phi_p\left((-1)^{d_2 d_1} \operatorname{res}_y\left(m_2(x - cy, y), m_1(y)\right)\right) \\
&= \frac{(-1)^{2 d_2 d_1}}{1} \cdot \Phi_p(\operatorname{norm}(m_2(x - c\alpha))) \\
&= 1 \cdot \prod_{i=1}^{d_1} \left(\Phi_p(m_2(x - c\alpha_i))\right) \\
&= \left(\Phi_p(m_2(x - c\alpha_k))\right)^2 \cdot \Phi_p\left(\prod_{\substack{i=1 \\ i \notin \{k, r\}}}^{d_1} m_2(x - c\alpha_i)\right).
\end{aligned}$$

Since $\deg_x(\Phi_p(m_2(x-c\alpha_k))) = \deg_x(m_2(x)) > 1$, N_p cannot be square-free for $c \in \mathbb{F}_p$.

Now let $\beta_1, \ldots, \beta_{d_2}$ be the roots of m_2 over a splitting field and suppose that M_2 is not square-free over $K_p[\overline{\alpha}]$. Then $\Phi_p(\beta_k) = \Phi_p(\beta_r)$ for some $1 \leq k < r \leq d_2$. Using the result above,

$$N_p(x) = \operatorname{res}_y\left(M_2(x-cy, y), M_1(y)\right) \quad = \prod_{i=1}^{d_1} \left(\Phi_p(m_2(x - c\alpha_i))\right) = \prod_{i=1}^{d_1} \left(M_2(x - c \cdot \Phi_p(\alpha_i))\right).$$

Thus for each $i = 1, \ldots, d_1$, $\{\Phi_p(\beta_j) - c \cdot \Phi_p(\alpha_i),\ j = 1, \ldots, d_2\}$ are roots of $N_p(x)$. But since $\Phi_p(\beta_k) = \Phi_p(\beta_r)$, there are repeated roots in this set. Hence $N_p(x)$ cannot be square-free over K_p regardless of the value of $c \in \mathbb{F}_p$. □

Lemma 6.1 implies that if M_1 or M_2 is not square-free over K_p and $K_p[\overline{\alpha}]$ respectively, then Algorithm **sqfr_norm** will fail for all $c \in \mathbb{F}_p$. Recall that we call a prime in which M_2 or M_1 is not square-free a fail prime (Chapter 2).

To prevent the algorithm from running for many choices of c which yields a non-square-free N_p, or worse, running for all choices of $c \in \mathbb{F}_p$ before returning $FAIL$, we modify Algorithm **sqfr_norm** to return $FAIL$ if three choices of $c \in \mathbb{F}_p$ do not give a square-free $N_p(x)$.

6.1.3 Choosing a "large enough" p

Suppose now that p is a good, non-fail prime (so that M_1 and M_2 are square-free) and let

$$N_p(x) = \operatorname{res}_y(M_2(x - cy, y), M_1(y)).$$

Observe that, by Remark 1.23, N_p is not square-free if c is a root of

$$r(x) := \operatorname{res}_x(N_p, N_p') \in K_p[x].$$

If p is less than the number of distinct roots of r, then every choice of $c \in \mathbb{F}_p$ may yield a non-square-free N_p. Thus must choose p to be larger than the number of distinct roots of $r(x)$ that belong to $\mathbb{F}_p$. For this, we need to find an upper bound on the number of such roots of $r(x)$. The following theorem and lemma are helpful for this purpose.

Theorem 6.2. *Let R and $\tilde{R}$ be commutative rings and let $f, g \in R[x]$ with nonzero degrees m and n respectively. Furthermore let $\phi : R \to \tilde{R}$ be a homomorphism. If $\deg(\phi(f)) = m$ and $\deg(\phi(g)) = k, 0 \leq k \leq n$, then*

$$\phi(\operatorname{res}(f, g)) = \phi(\operatorname{lcoeff}(f))^{n-k} \operatorname{res}(\phi(f), \phi(g)).$$

Proof. See Geddes et al. [11, Theorem 9.2, p. 408]. □

Lemma 6.3. *Let $A(x) \in K_p[x]$ where $K_p = \mathbb{F}_p$ or $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r]$ and $M_i \in \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{i-1}][u_i]$ for $2 \leq i \leq r$ and $M_1 \in \mathbb{F}_p[u_1]$. If $\deg_x(A) = d \geq 0$, then the number of roots of A belonging to $\mathbb{F}_p$ is at most d.*

Proof. Suppose that $A(x) \in \mathbb{F}_p[x]$. We show that $A(x)$ has at most d roots in $\mathbb{F}_p$ using induction on $d = \deg_x(A)$. If $\deg_x(A) = 0$ then there are no roots and we are done. Suppose now that $\deg_x(A) = d+1$ and that the statement holds for A of degree d. If $\rho_1 \in \mathbb{F}_p$ is a root of A, then $A(x) = (x - \rho_1) \cdot B(x)$ where $B(x) \in \mathbb{F}_p[x]$. If $\deg_x(A) = 1$, then $\deg_x(B) = 0$ so we are done. Otherwise, let ρ_2 be a root of A, not equal to ρ_1. Then $f(\rho_2) = (\rho_2 - \rho_1) \cdot g(\rho_2) = 0$. Since $\rho_2 - \rho_1 \neq 0$ and $\mathbb{F}_p$ is a field, we have $B(\rho_2) = 0$. By induction hypothesis, B has at most d roots. Thus A has at most $d + 1$ roots. This proves that $A(x)$ of degree d can have at most d roots in $\mathbb{F}_p$. Now suppose that $A(x) \in K_p[x]$ where $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r] \cong \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r \rangle$ and $d_i = \deg_{u_i}(M_i), 1 \leq i \leq r$. Let

$$c_i(h_1, h_2, \ldots, h_r) = \operatorname{coeff}(x^i u_1^{h_1} u_2^{h_2} \cdots u_r^{h_r},\ A) \in \mathbb{F}_p,\ 0 \leq h_j \leq d_j, 1 \leq j \leq r.$$

If $A(\sigma) = 0$ for some $\sigma \in \mathbb{F}_p$, then σ must also be a root of *all* of the following $d_1 d_2 \cdots d_r$ polynomials

$$\begin{aligned}
&\sum_{i=0}^{d} c_i(0, 0, \ldots, 0) \cdot x^i \in \mathbb{F}_p[x], \\
&\sum_{i=0}^{d} c_i(0, 0, \ldots, 1) \cdot x^i \in \mathbb{F}_p[x], \\
&\qquad \vdots \\
&\sum_{i=0}^{d} c_i(d_1 - 1, d_2 - 1, \ldots, d_r - 1) \cdot x^i \in \mathbb{F}_p[x],
\end{aligned} \tag{6.1}$$

at least one of which is non-zero since $A(x)$ is non-zero. By the proof above, each polynomial in (6.1) can have at most d roots in $\mathbb{F}_p$. It may be the case that all the polynomials in (6.1) have the same d roots in $\mathbb{F}_p$. Thus there can be at most d roots of $A(x)$ in $\mathbb{F}_p$. □

By Lemma 6.3, the number of roots of $r(x) = \text{res}_x(N_p, N_p')$ belonging to $\mathbb{F}_p$ is at most the degree of $r(x)$. We now determine an upper bound on degree of $r(x)$. Let $M_2(x - cy, y) = \Phi_p(m_2(x - cy, y))$. By Theorem 6.2,

$$\Phi_p(\text{res}_y(m_2(x - cy, y), m_1(y))) = \Phi_p(N(x)) = N_p(x) = \text{res}_y(M_2(x - cy, y), M_1(y)). \tag{6.2}$$

Because $N(x)$ is a minimal polynomial by Remark 5.10, $N(x)$ must be monic. Hence we must have

$$\deg_x(N) = \deg_x(N_p). \tag{6.3}$$

Moreover, by Theorems 1.19 (i) and 6.2,

$$\begin{aligned} \Phi_p(\text{res}_x(N, N')) &= \Phi_p(\text{res}_x(N', N)) \\ &= \text{res}_x(\Phi_p(N'), \Phi_p(N)) \\ &= \text{res}_x(N_p', N_p) \\ &= (-1)^{mn}\,\text{res}_x(N_p, N_p') \\ &= (-1)^{mn} r(x), \end{aligned} \tag{6.4}$$

where $m = \deg_x(N_p)$ and $n = \deg_x(N_p')$. Let $d_p = \deg_x(r)$ and $d_r = \deg_x(\text{res}_x(N, N'))$. By Lemma 5.5 and (6.4), we must have $d_p \leq d_r$ and $d_r \leq d_2^2 d_1(d_1 - 1)/2$. That is, there can be at most $d_2^2 d_1(d_1 - 1)/2$ distinct roots of $r(x)$ that belong to $\mathbb{F}_p$. Thus we require $p \geq d_2^2 d_1(d_1 - 1)/2$ to guarantee that a square-free $N_p(x)$ exists. In fact, we will choose p so that

$$p \geq 2\left(\frac{d_2^2 d_1(d_1 - 1)}{2}\right) = d_2^2 d_1(d_1 - 1) \tag{6.5}$$

so that the probability of randomly choosing $c \in \mathbb{F}_p$ for which $N_p(x)$ is square-free will be at least $1/2$. However, in theory the first $\lfloor d_2^2 d_1(d_1 - 1)/2 \rfloor$ consecutive choices for $c \in \mathbb{F}_p$ may yield a non-square-free $N_p(x)$. If this happens, the while-loop in Algorithm **sqfr_norm** would have to run $\lfloor d_2^2 d_1(d_1 - 1)/2 \rfloor + 1$ times, which is inefficient. For this reason, we also modify the algorithm to choose c at random from $\mathbb{F}_p$, rather than starting from 0 then counting up by 1.

6.1.4 Proof of correctness

We now show that if Algorithm **sqfr_norm** is modified as discussed above and terminates successfully, then its output, a square-free $N_p(x)$, satisfies

$$N_p(x) = \Phi_p(N(x)) = \mathrm{res}_y(m_2(x - cy, y), m_1(y)) \mod p,$$

where $N(x)$ is square-free over K.

To show this, observe that by (6.2), if $N_p(x)$ is square-free over K_p for some $c \in \mathbb{F}_p$, then $N(x)$ must be square-free over K for this c as well. Hence a square-free $N_p(x)$ will always be equal to $N(x) \mod p$, where $N(x)$ is the minimal polynomial for some primitive element of $K(\alpha, \beta)$.

6.1.5 Modified algorithm of sqfr_norm and its complexity

We present the modified Algorithm **sqfr_norm** and its time complexity.

Suppose that $d_1 = \deg_y(M_1(y))$ and $d_2 = \deg_x(M_2(x))$. We analyze the cost of Algorithm **sqfr_norm_p** for $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r]$.

- Line 7: To find the expanded form of $M_2(x - c\overline{\alpha}, \alpha)$, we pre-compute $C = -c\overline{\alpha}$ then apply the substitution $x \leftarrow x + C$ from the rightmost x to the leftmost x in the Horner form of $M_2(x, \overline{\alpha})$:

  $$M_2(x, \overline{\alpha}) = a_0(\overline{\alpha}) + x(a_1(\overline{\alpha}) + x(a_2(\overline{\alpha}) + \cdots + x(a_{d_2-1}(\overline{\alpha}) + x)\cdots)).$$

  When substituting the k-th rightmost x for $x + C$ (for $1 \le k \le d_2$), we are required to compute the expanded form of

  $$a_{d_2-k}(\overline{\alpha}) + (x + C) \cdot r(x, \overline{\alpha}) = a_{d_2-k}(\overline{\alpha}) + x \cdot r(x, \overline{\alpha}) + C \cdot r(x, \overline{\alpha}),$$

  where $r(x, \overline{\alpha})$ is a polynomial in $K_p[\overline{\alpha}][x]$ whose degree in x is $k - 1$. Multiplying x by $r(x, \overline{\alpha})$ is equivalent to shifting the degrees of x in $r(x, \overline{\alpha})$ by $+1$, so no arithmetic operation is required for this step. On the other hand, multiplying $C = -c\overline{\alpha}$ by $r(x, \overline{\alpha})$ requires $\deg_x(r)$ multiplications in $K_p[\overline{\alpha}]$, or

Algorithm 6.1 : **sqfr_norm_p**$(M_2(x,\overline{\alpha}), M_1(y), K_p)$

Input: $M_2(x,\overline{\alpha}) = \Phi_p(m_2(x,\alpha)) \in K_p[\overline{\alpha}][x]$ and $M_1(y) = \Phi_p(m_1(y))) \in K_p[y]$ where $m_2(x,\alpha)$ and $m_1(y)$ are minimal polynomials for β over $K(\alpha)$ and α over K respectively where $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_r)$ and $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r]$ for a good prime p satisfying $p > (\deg_x(M_2)^2 \cdot \deg_y(M_1) \cdot (\deg_y(M_1) - 1))$.

Output: $c \in \mathbb{F}_p$, $g(x,\overline{\alpha}) = M_2(x - c\overline{\alpha}) \in K_p[\overline{\alpha}][x]$, and square-free $N_p(x) = \mathrm{res}_y(g(x,y), M_1(y)) \in K_p[x]$, or $FAIL$.

1: $c \leftarrow 0$; $S \leftarrow \{0\}$; $i \leftarrow 1$; $g(x,\overline{\alpha}) \leftarrow M_2(x,\overline{\alpha})$;
2: **if** $\deg_1(g) = 0$ **then**
3: {$\mathrm{norm}(g(x,\overline{\alpha})) = M_2(x,\overline{\alpha})$ is known to be not square-free by Lemma 1.20}
4: $c \leftarrow$ random integer in $\mathbb{F}_p \backslash S$; $S \leftarrow S \cup \{c\}$;
5: **end if**
6: **while** true **do**
7: $g(x,\overline{\alpha}) \leftarrow M_2(x - c\overline{\alpha}, \overline{\alpha})$;
8: $N_p(x) \leftarrow \mathrm{res}_y(g(x,y), M_1(y))$;
9: {compute $N_p(x)$ using evaluation homomorphism & interpolation (Section 7.1)}
10: {$N_p(x)$ returns $FAIL$ if a zero divisor is encountered}
11: **if** $N_p(x) \neq FAIL$ **then**
12: $T(x) \leftarrow \gcd(N_p(x), N_p'(x))$; {compute $T(x)$ using the Euclidean algorithm over K_p; if the Euclidean algorithm encounters a division by a zero divisor, it returns $FAIL$}
13: **if** {$T(x) \neq FAIL$ **and** $\deg_x(T) = 0$} **then**
14: {No division by a zero divisor is encountered during the computation of $T(x)$, and $N_p(x)$ is square-free over K_p}
15: **return** $c, g(x,\overline{\alpha}), N_p(x)$;
16: **end if**
17: **else if** $N_p(x)$ is not square-free for three values of c **then**
18: **return** $FAIL$;
19: **end if**
20: $c \leftarrow$ random element in $\mathbb{F}_p \backslash S$; $S \leftarrow S \cup \{c\}$; $i \leftarrow i + 1$;
21: **end while**

$(\deg_x(r) \cdot (\deg(\alpha)^2) = k \cdot d_1^2$ multiplications in K_p. Hence the total number of multiplications required (in K_p) for expanding out $M(x - c\overline{\alpha}, \overline{\alpha})$ is

$$\sum_{k=1}^{d_2} k \cdot d_1^2 = \frac{d_2(d_2+1)}{2} d_1^2 \subseteq \mathcal{O}(d_1^2 d_2^2).$$

- Line 8: In Section 7.1 we discuss an efficient method for computing multivariate resultants (based on evaluation and interpolation of the variable x), and show that it requires

$$\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) \text{ arithmetic operations in } K_p.$$

- Line 12: By (6.3), $\deg_x(N_p) = d_1 d_2$. One can attempt to find the gcd of $N_p(x)$ and $N_p'(x)$ via the Euclidean algorithm, which requires

$$\mathcal{O}((d_1 d_2)(d_1 d_2 - 1)) \subseteq \mathcal{O}(d_1^2 d_2^2) \text{ arithmetic operations over } K_p.$$

Since the while-loop in the algorithm will be executed at most three times, the number of arithmetic operations in K_p in executing Algorithm **sqfr_norm_p** is

$$\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) + \mathcal{O}(d_1^2 d_2^2) + O(d_1^2 d_2^2) \subseteq O(d_1^3 d_2 + d_1^2 d_2^2).$$

We remark that, because of the asymmetry in the cost, if $d_1 \leq d_2$ then this cost can be simplified to

$$\mathcal{O}(d_1^2 d_2^2 + d_1^2 d_2^2) \subseteq \mathcal{O}(d_1^2 d_2^2).$$

Thus in the special case where $m_2(x)$ has no dependence on α (i.e. $m_2(x) \in K_p[x]$) and $\deg_y(m_1) > \deg_y(m_2)$, switching the variable ordering allows the resultant to be computed in $\mathcal{O}(d_1^2 d_2^2)$, rather than in $\mathcal{O}(d_1^3 d_2)$.

6.2 Modifications to Algorithm prim_elt

We now discuss modifications we must implement to Algorithm **prim_elt** so that it takes inputs $M_1(y) \in K_p[y]$ and $M_2(x, \overline{\alpha}) \in K_p[\overline{\alpha}][x]$ and, if it does not return $FAIL$, then it successfully outputs

$$\Phi_p(m_\gamma(x)), \quad \overline{\alpha}(\overline{\gamma}), \quad \text{and} \quad \overline{\beta}(\overline{\gamma}),$$

where $\overline{\alpha}(\overline{\gamma}) = \Phi_p(\alpha(\gamma))$ and $\overline{\beta}(\overline{\gamma}) = \Phi_p(\beta(\gamma))$.

6.2.1 Handling zero divisors

In Section 7.2 we introduce a method for computing a gcd required in Line 5 of Algorithm **prim_elt** for which no inversion of zero divisors is encountered. However, the gcd found using this method may not be monic. Since Algorithm **prim_elt** requires the monic gcd, Algorithm **prim_elt** may still fail over K_p if a zero divisor is encountered while trying to make the gcd monic. If this is the case, we will choose a new prime p and restart the algorithm. Experimentally, we have found that such failure is rare.

6.2.2 Proof of correctness

Let $m_\gamma(x) \in K[x]$ is the minimal polynomial for a primitive element γ satisfying $K(\alpha, \beta) = K(\gamma)$. We have already shown that Algorithm **sqfr_norm_p** either returns $FAIL$ or correctly returns $\Phi_p(m_\gamma(x))$. Thus here we show that if Algorithm **prim_elt** over K_p successfully terminates for input polynomials M_1 and M_2, then it outputs $\overline{\alpha}(\overline{\gamma})$ and $\overline{\beta}(\overline{\gamma})$.

Suppose that Algorithm **prim_elt** successfully terminates and let

$$G(x, y) := M_2(x - cy, y) \quad \text{and} \quad g(x, y) := m_2(x - cy, y),$$

where $c \in \mathbb{F}_p$ is chosen so that $N_p(x) = \text{res}_y(G(x, y), M_1(y))$ is square-free over K_p. We know by Theorem 5.8 that $\deg(\gcd(g(\gamma, y), m_1(y)) = 1$. However, it could be that $\deg(\gcd(\Phi_p(g(\gamma, y)), \Phi_p(m_1(y))) > 1$. If $\deg(\gcd(\Phi_p(g(\gamma, y)), \Phi_p(m_1(y))) \leq \deg(\gcd(g(\gamma, y), m_1(y))$ and the gcds are monic, then one can show that $\Phi_p(\gcd(g(\gamma, y), m_1(y))) = \gcd(G(\gamma, y), M_1(y))$ (Geddes et al. [11, Lemma 7.3, p.300]). Hence after we compute $\gcd(\Phi_p(g(\gamma, y)), \Phi_p(m_1(y))$ in the modified algorithm, we check that its degree is one. If it is not, then it returns $FAIL$.

Moreover, by Theorem 5.8

$$\gcd(g(\gamma, y), m_1(y)) = y - \alpha(\gamma) \in K(\gamma)[y] \cong K[x]/\langle m_\gamma(x)\rangle[y], \tag{6.6}$$

where $\alpha(\gamma)$ is the normal representation of α in $K(\gamma)$. Thus the solution to

$$\gcd(G(\gamma, y), M_1(y)) = \Phi_p(y - \alpha(\gamma)) = y - \Phi_p(\alpha(\gamma)) = y - \overline{\alpha}(\overline{\gamma})$$

is $\overline{\alpha}(\overline{\gamma}) = \Phi_p(\alpha(\gamma))$, as required.

Furthermore, recall that $\beta(\gamma)$ can be found by the formula $\beta(\gamma) = \gamma - c \cdot \alpha(\gamma)$. Thus $\overline{\beta}(\overline{\gamma}) = \Phi_p(\beta(\gamma))$ can be computed in an analogous way:

$$\overline{\gamma} - c \cdot \overline{\alpha}(\overline{\gamma}) = \Phi_p(\gamma - c \cdot \alpha(\gamma)) = \overline{\beta}(\overline{\gamma}).$$

In summary, if Algorithm **prim_elt**, is executed over K_p successfully, then it returns $\Phi_p(m_\gamma(x)), \Phi_p(\alpha(\gamma)) = \overline{\alpha}(\overline{\gamma})$ and $\Phi_p(\beta(\gamma)) = \overline{\beta}(\overline{\gamma})$.

Example 6.4. Let $m_1(y)$ and $m_2(x, \alpha)$ be as in Examples 5.14 and 5.15:

$$m_1(y) = y^4 - 2 \in \mathbb{Q}[y] \text{ and } m_2(x, \alpha) = x^4 - \alpha^2 x^2 - 2 \in \mathbb{Q}(\alpha)[x].$$

We wish to find $c \in \mathbb{Z}$ for which $\mathbb{Q}(\beta + c\alpha) \cong \mathbb{Q}(\alpha, \beta)$. By (6.5), we choose the prime p to be greater than

$$\frac{\deg_x(m_2)^2 \cdot \deg_y(m_1) \cdot (\deg_y(m_1) - 1)}{2} = \frac{4^2 \cdot 4 \cdot (4-1)}{2} = 96$$

so that there exists $c \in \mathbb{F}_p$ for which $\overline{\alpha} + c\overline{\beta}$ is a primitive element modulo p of $\mathbb{Q}(\alpha, \beta)$ with probability greater than $1/2$. Let $p = 113$. Then

$$\begin{aligned} &M_1(y) := \Phi_p(m_1(y)) = (y - 47)\,(y + 27)\,(y + 47)\,(y - 27) \text{ and} \\ &M_2(x, \overline{\alpha}) := \Phi_p(m_2(x, \alpha)) = x^4 - \overline{\alpha}^2 x^2 - 2, \text{ where } \overline{\alpha} \text{ is a root of } M_1(y). \end{aligned}$$

One can check that $M_1(y)$ and $M_2(x, \overline{\alpha})$ are square-free over K_p and $K_p[\overline{\alpha}]$ respectively.

We let $c = 0$ and compute the resultant of $M_2(x - cy, y)$ and $M_1(y)$:

$$\begin{aligned} \operatorname{res}_y(M_2(x, y), M_1(y)) &= \operatorname{res}_y(x^4 - y^2x^2 - 2,\, y^4 - 2) \\ &= (x^8 - 6x^4 + 4)^2 \in \mathbb{F}_p[x]. \end{aligned} \tag{6.7}$$

Since (6.7) is not square-free, we choose a new, random c. We use $c = 1$ so that we can compare the output with Example 5.14. Using this new c, the resultant is:

$$\begin{aligned} N_p(x) = \operatorname{res}_y(M_2(x - y, y), M_1(y)) &= \operatorname{res}_y((x - y)^4 - y^2(x - y)^2 - 2,\, y^4 - 2) \\ &= x^{16} - 44x^{12} - 16x^8 + 13x^4 + 16 \in \mathbb{F}_p[x]. \end{aligned}$$

One can check that $N_p(x)$ is square-free over $\mathbb{F}_p$. Thus $\overline{\gamma} := \Phi_p(\gamma) = \overline{\beta} + c \cdot \overline{\alpha} = \overline{\beta} + \overline{\alpha}$ is a primitive element (modulo p) of $\mathbb{Q}(\alpha, \beta)$, and $N_p(x)$ is equal to $\Phi_p(m_\gamma(x))$, where

$m_\gamma(x) \in \mathbb{Q}[x]$ is the minimal polynomial for γ over $\mathbb{Q}$. Indeed, one can verify this by comparing $N_p(x)$ with $m_\gamma(x)$ found in Example 5.14.

Now let us find $\overline{\alpha}(\overline{\gamma})$ and $\overline{\beta}(\overline{\gamma})$. In this example, we will do this using the Euclidean algorithm over $\mathbb{F}_p[\overline{\gamma}] \cong \mathbb{F}_p[x]/\langle\langle N_p(x)\rangle\rangle$. No zero divisor is encountered during the computation and we obtain

$$\gcd(M_2(\overline{\gamma} - y, y), M_1(y)) = y - 45\overline{\gamma} - 24\overline{\gamma}^5 - 36\overline{\gamma}^9 + 5\overline{\gamma}^{13} \in \mathbb{F}_p[\overline{\gamma}][y].$$

Therefore,

$$\overline{\alpha}(\overline{\gamma}) = 45\overline{\gamma} + 24\overline{\gamma}^5 + 36\overline{\gamma}^9 - 5\overline{\gamma}^{13} \in \mathbb{F}_p[\overline{\gamma}] \quad \text{and}$$
$$\overline{\beta}(\overline{\gamma}) = \overline{\gamma} - A(\overline{\gamma}) = \overline{\gamma} - (45\overline{\gamma} + 24\overline{\gamma}^5 + 36\overline{\gamma}^9 - 5\overline{\gamma}^{13}) = -44\overline{\gamma} - 24\overline{\gamma}^5 - 36\overline{\gamma}^9 + 5\overline{\gamma}^{13} \in \mathbb{F}_p[\overline{\gamma}].$$

Again, one can verify that $\overline{\alpha}(\overline{\gamma}) = \Phi_p(\alpha(\gamma))$ and $\overline{\beta}(\overline{\gamma}) = \Phi_p(\beta(\gamma))$ where $\alpha(\gamma)$ and $\beta(\gamma)$ were computed in Example 5.15.

6.2.3 Modified algorithm of prim_elt and its complexity

We present the modified version of Algorithm **prim_elt** to work modulo p and its time complexity.

Let $d_1 = \deg_y(M_1)$ and $d_2 = \deg_x(M_2)$. We analyze the cost of Algorithm **prim_elt_p**.

- Line 1: we showed in Section 6.1.5 that Algorithm **sqfr_norm_p** requires

  $$\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) \text{ arithmetic operations in } K_p.$$

- Line 5: We will use evaluation & interpolation to find the gcd (Section 7.2). There, we show that this method requires

  $$\mathcal{O}((d_1 d_2)^2) \text{ arithmetic operations in } K_p.$$

- Line 13: Since $\deg_x(N_p) = d_1 d_2$, in computing $B(\overline{\gamma})$ we perform at most one scalar multiplication and one subtraction in $K_p[x]/\langle N_p(x)\rangle$. Since $\deg_x(N_p) = d_1 d_2$, the cost of finding $A(\overline{\gamma})$ and $B(\overline{\gamma})$ is

  $$\mathcal{O}(d_1 d_2) \text{ arithmetic operations in } K_p.$$

Algorithm 6.2 : **prim_elt_p**$(M_2(x,\overline{\alpha}), M_1(y), K_p)$

Input: $M_2(x,\overline{\alpha}) = \Phi_p(m_2(x,\alpha)) \in K_p[\overline{\alpha}][x]$ and $M_1(y) = \Phi_p(m_1(y))) \in K_p[y]$ where $m_2(x,\alpha)$ and $m_1(y)$ are minimal polynomials for β over $K(\alpha)$ and α over K respectively, where $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_r)$ and $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r]$ for good prime p satisfying $p > (\deg_x(M_2)^2 \cdot \deg_y(M_1) \cdot (\deg_y(M_1) - 1)$.

Output: $c \in \mathbb{F}_p$, square-free $N_p(x) \in K_p[x]$ satisfying $N(\overline{\gamma} = \overline{\beta} + c\overline{\alpha}) = 0$, and $A(\overline{\gamma}), B(\overline{\gamma})$, the normal representations of $\overline{\alpha}$ and $\overline{\beta}$ respectively in $K_p[\overline{\gamma}] \cong K_p[x]/\langle N_p(x)\rangle$, or $FAIL$.

1: $c, g(x,\overline{\alpha}), N_p(x) \leftarrow$ **sqfr_norm_p**$(M_2(x,\overline{\alpha}), M_1(y), K_p, p)$;
2: **if** Algorithm **sqfr_norm_p** outputs $FAIL$ **then**
3: **return** $FAIL$;
4: **end if**
5: $h(\overline{\gamma}, y) \leftarrow$ monic $\gcd(g(\overline{\gamma}, y), M_1(y))$; {$h$ is computed over $K_p[\overline{\gamma}] \cong K_p[x]/\langle N_p(x)\rangle$ via evaluation & interpolation (Section 7.2). Note: $h(\overline{\gamma}, y) = y + a(\overline{\gamma}) \in K_p[\overline{\gamma}][y]$}
6: **if** division by a zero divisor is encountered while computing $h(\overline{\gamma}, y)$ **then**
7: **return** $FAIL$;
8: **end if**
9: **if** $\deg_y(h) \neq 1$ **then**
10: **return** $FAIL$;
11: **end if**
12: $A(\overline{\gamma}) \leftarrow -a(\overline{\gamma})$;
13: $B(\overline{\gamma}) \leftarrow \overline{\gamma} - cA(\overline{\gamma})$;
14: **return** $c, N_p(x), A(\overline{\gamma}), B(\overline{\gamma})$;

Thus in total, the cost of running Algorithm **prim_elt_p** is

$$\mathcal{O}((d_1^3 d_2 + d_1^2 d_2^2) + (d_1 d_2)^2 + d_1 d_2) \subseteq \mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) \text{ arithmetic operations in } K_p.$$

As with Algorithm **sqfr_norm_p**, if $d_1 \leq d_2$ then this cost simplifies to

$$\mathcal{O}(d_1^2 d_2^2 + d_1^2 d_2^2) \subseteq \mathcal{O}(d_1^2 d_2^2) \text{ arithmetic operations in } K_p.$$

6.3 Towers with more than two steps

Recall that to find a primitive element of a field towers with more than two steps of the form $K(\alpha_1, \ldots, \alpha_t)$ where $m_i(u_i) \in K(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ for $i = 2, \ldots, t$, we find a primitive element of two successive towers at a time:

$$\begin{aligned} K(\alpha_1, \alpha_2, \ldots, \alpha_t) &\cong K(\alpha_1, \ldots, \alpha_{t-3}, \alpha_{t-2})(\gamma_{t-1}) \\ &\cong K(\alpha_1, \ldots, \alpha_{t-3})(\gamma_{t-2}) \\ &\quad \vdots \\ &\cong K(\alpha_1, \gamma_2) \\ &\cong K(\gamma). \end{aligned}$$

Let

$$\begin{aligned} &M_i = \Phi_p(m_i) \text{ for } i = 1, \ldots, t, \quad M_{\gamma_i} = \Phi_p(m_{\gamma_i}) \text{ for } i = 2, \ldots, t, \\ &d_i = \deg_x(m_i) \text{ for } i = 1, \ldots, t, \quad \text{and } D = \textstyle\prod_{i=1}^{t} d_i. \end{aligned}$$

For the algorithm to work, we need to choose p such that a set $\{c_2, \ldots, c_t\} \in \mathbb{F}_p^{t-1}$ exists for which we can find square-free

$$\begin{aligned} &M_{\gamma_t}(x) = \operatorname{res}_y(M_t(x - c_t y, y), M_{t-1}(y)) \text{ and} \\ &M_{\gamma_k}(x) = \operatorname{res}_y(M_{\gamma_{k+1}}(x - c_k y, y), M_{k-1}(y)) \text{ for } k = 2, \ldots, t-1. \end{aligned}$$

By Lemma 5.5 there are at most

$$\frac{d_t^2 \cdot d_{t-1}(d_{t-1} - 1)}{2} < \frac{d_t^2 \cdot d_{t-1}^2}{2}$$

distinct $c_t \in \mathbb{F}_p$ for which $M_{\gamma_t}(x) = \operatorname{res}_y(M_{t-1}(x - c_t y, y), M_t(y))$ is not square-free. Note that $\deg_x(M_{\gamma_t}) = d_{t-1} d_t$. Thus once we have found M_{γ_t}, there are at most

$$\frac{\deg(M_{\gamma_t})^2 \, d_{t-2}(d_{t-2} - 1)}{2} = \frac{(d_{t-1} d_t)^2 \, d_{t-2}(d_{t-2} - 1)}{2} < \frac{d_{t-2}^2 d_{t-1}^2 d_t^2}{2}$$

distinct $c_{t-1} \in \mathbb{F}_p$ for which $\operatorname{res}_y(M_{\gamma_t}(x - cy, y), M_{t-2}(y))$ is not square-free. In general, for $2 \leq i \leq t-1$, there are at most

$$\frac{d_{i-1}^2 d_i^2 \cdots d_t^2}{2}$$

distinct $c_i \in \mathbb{F}_p$ for which $\operatorname{res}_y(M_{\gamma_{i+1}}(x - c_i y, y), M_{i-1}(y))$ is not square-free. By application of Lemma 5.5, if we choose p so that it satisfies

$$p > \max\left\{ \frac{d_{t-1}^2 d_t^2}{2}, \frac{d_{t-2}^2 d_{t-1}^2 d_t^2}{2}, \ldots, \frac{d_1^2 d_2^2 \cdots d_t^2}{2} \right\} = \frac{d_1^2 d_2^2 \cdots d_t^2}{2}, \tag{6.8}$$

then there must exist $\{c_2, \ldots, c_t\} \in \mathbb{F}_p^{t-1}$ such that every $\text{res}_y(M_{\gamma_i}(x - c_i y, y), M_{i-1}(y))$ (for $2 \leq i \leq t-2$) and $\text{res}_y(M_t(x - c_t y, y), M_{t-1}(y))$ are square-free. In fact, if we choose p so that

$$p > 2\left(\frac{d_1^2 d_2^2 \cdots d_t^2}{2}\right) = d_1^2 d_2^2 \cdots d_t^2 = D^2,$$

the probability that a random c_i gives a square-free resultant at the $(i-1)$-th step is at least $1/2$.

6.3.1 Finding the normal representations

As mentioned in Section 5.5, if the number of extensions t is greater than two, then executing Algorithm **prim_elt_p** does not directly give us the normal representations $\overline{\alpha}_i(\overline{\gamma})$. For example, if $t = 3$ then running Algorithm **prim_elt_p** twice returns the normal representations

$$\overline{\alpha}_2(\overline{\gamma}_3),\ \overline{\alpha}_3(\overline{\gamma}_3),\ \overline{\alpha}_1(\overline{\gamma})\ \text{ and } \overline{\gamma}_3(\overline{\gamma}),$$

where $K_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3] = K_p[\overline{\alpha}_1, \overline{\gamma}_3]$ and $K_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3] = K_p[\overline{\gamma}]$. Thus at this point we must still determine $\overline{\alpha}_2(\overline{\gamma})$ and $\overline{\alpha}_3(\overline{\gamma})$. In this section, we explain in detail how to do so, and also analyze it complexity.

To find the normal representation $\overline{\alpha}_2(\overline{\gamma})$ from $\overline{\alpha}_2(\overline{\alpha}_1, \overline{\gamma}_3)$ we need to make the substitutions $\overline{\alpha}_1 \mapsto \overline{\alpha}_1(\overline{\gamma})$ and $\overline{\gamma}_3 \mapsto \overline{\gamma}_3(\overline{\gamma})$ to $\overline{\alpha}_2(\overline{\alpha}_1, \overline{\gamma}_3)$. Note that

$$\overline{\alpha}_2(\overline{\alpha}_1, \overline{\gamma}_3) = \sum_{j=0}^{d_1-1}\left(\sum_{i=0}^{d_2 d_3 - 1} k_{ij}\overline{\gamma}_3^i\right)\overline{\alpha}_1^j,\ \text{where } k_{ij} \in \mathbb{F}_p. \tag{6.9}$$

We first pre-compute $\overline{\alpha}_1(\overline{\gamma})^j \in \mathbb{F}_p[\overline{\gamma}]$ for $2 \leq j \leq d_1 - 1$ and $\overline{\gamma}_3(\overline{\gamma})^i \in \mathbb{F}_p[\overline{\gamma}]$ for $2 \leq i \leq d_2 d_3 - 1$. If $D = d_1 d_2 d_3$, then computing all of the required powers of $\overline{\alpha}_1(\overline{\gamma})$ and $\overline{\gamma}_3(\overline{\gamma})$ requires $\mathcal{O}(d_1 D^2)$ and $\mathcal{O}(d_2 d_3 D^2)$ arithmetic operations in $\mathbb{F}_p$ respectively.

Now we make the substitutions

$$\overline{\alpha}_1^i \mapsto \overline{\alpha}_1(\overline{\gamma})^i \text{ for } 1 \leq i \leq d_1 - 1\ \text{ and }\ \overline{\gamma}_3^j \mapsto \overline{\gamma}_3(\overline{\gamma})^j \text{ for } 1 \leq j \leq d_2 d_3 - 1$$

to (6.9). Making these substitutions in the inner summation of (6.9) for a fixed j requires $\mathcal{O}(d_2 d_3 D)$ multiplications in $\mathbb{F}_p$ since $k_{ij} \in \mathbb{F}_p$. Therefore, the total cost of

computing the inner summation for all i and j is

$$\mathcal{O}(d_1 \cdot (d_2 d_3 D)) = \mathcal{O}(D^2) \text{ arithmetic operations in } \mathbb{F}_p.$$

Now for each j, the inner summation $\sum_{i=0}^{d_2 d_3 - 1} k_{ij} \overline{\gamma}_3 (\overline{\gamma})^i$ is a polynomial in $\mathbb{F}_p[\overline{\gamma}]$ of degree less than D. So the cost of computing the outer summation is equal to the cost of multiplying $(d_1 - 1)$ pairs of polynomials in $\mathbb{F}_p[\overline{\gamma}]$, which is

$$\mathcal{O}(d_1 D^2) \text{ arithmetic operations in } \mathbb{F}_p.$$

Hence computing $\overline{\alpha}_2(\overline{\gamma})$ from $\overline{\alpha}_2(\overline{\alpha}_1, \overline{\gamma}_3)$ from (6.9) requires

$$\mathcal{O}(d_1 D^2) + \mathcal{O}(d_2 d_3 D^2) + \mathcal{O}(D^2) + \mathcal{O}(d_1 D^2) \subseteq \mathcal{O}(D^2(d_1 + d_2 d_3))$$

arithmetic operations in $\mathbb{F}_p$.

To find $\overline{\alpha}_3(\overline{\gamma})$, we can simply solve the equation $\overline{\gamma}_2(\overline{\gamma}) = \overline{\alpha}_2(\overline{\gamma}) + c_3 \overline{\alpha}_3(\overline{\gamma})$ for $\overline{\alpha}_3(\overline{\gamma})$ where $c_3 \in \mathbb{F}_p$ is returned by Algorithm **prim_elt_p**. This requires at most D arithmetic operations in $\mathbb{F}_p$. In summary, the cost in $\mathbb{F}_p$ of computing the normal representations $\overline{\alpha}_2(\overline{\gamma})$ and $\overline{\alpha}_3(\overline{\gamma})$ is

$$\mathcal{O}(D^2(d_1 + d_2 d_3) + \mathcal{O}(D) \subseteq \mathcal{O}(D^2(d_1 + d_2 d_3)).$$

In general, one can show that the total number of arithmetic operations in $\mathbb{F}_p$ required in finding $\overline{\alpha}_k(\overline{\gamma})$, $k = 2, \ldots, t$, where t is the number of extensions, is

$$\begin{aligned}
&\mathcal{O}(D^2(d_2 \cdots d_t + d_1)) + \mathcal{O}(D^2(d_3 \cdots d_t + d_1 d_2)) + \cdots \\
&+\mathcal{O}(D^2(d_{t-1} d_t + d_1 \cdots d_{t-2})) + \mathcal{O}(D) \\
&\subseteq \mathcal{O}\left(D^2(d_2 \cdots d_t + d_3 \cdots d_t + \cdots + d_{t-1} d_t) + D^2(d_1 + d_1 d_2 + \cdots + d_1 \cdots d_{t-2})\right).
\end{aligned}$$

Observe that

$$\begin{aligned}
d_2 \cdots d_t + d_3 \cdots d_t + \cdots + d_{t-1} d_t &= D\left(\frac{1}{d_1} + \frac{1}{d_1 d_2} + \cdots + \frac{1}{d_1 d_2 \cdots d_{t-2}}\right) \\
&< D\left(\frac{d_2 \cdots d_{t-2} + d_3 \cdots d_{t-2} + \cdots + d_{t-3} d_{t-2} + 2 d_{t-2}}{d_1 d_2 \cdots d_{t-2}}\right) \\
&< D\left(\frac{d_2 \cdots d_{t-2} + d_3 \cdots d_{t-2} + \cdots + 3 d_{t-3} d_{t-2}}{d_1 d_2 \cdots d_{t-2}}\right) \\
&\vdots \\
&< D\left(\frac{(t-3) d_2 \cdots d_{t-2}}{d_1 d_2 \cdots d_{t-2}}\right) = D\left(\frac{t-3}{d_1}\right).
\end{aligned}$$

Using a similar argument, it is easy to see that

$$d_1 + d_1 d_2 + \cdots + d_1 \cdots d_{t-2} < D\left(\frac{t-3}{d_{t-1}d_{t-2}}\right).$$

Therefore, the total number of arithmetic operations required in $\mathbb{F}_p$ in finding $\overline{\alpha}_k(\overline{\gamma}), k = 2, \ldots, t$ can be simplified to

$$\mathcal{O}\left(D^2 \cdot D\left(\frac{t-3}{d_1}\right) + D^2 \cdot D\left(\frac{t-3}{d_{t-1}d_{t-2}}\right)\right) \subseteq \mathcal{O}\left(D^3 t\left(\frac{1}{d_1} + \frac{1}{d_{t-1}d_{t-2}}\right)\right). \quad (6.10)$$

Note, in particular, that if $d_1 = d_2 = \cdots = d_t$, then $d_i = \sqrt[t]{D}$ for all i, so (6.10) becomes

$$\mathcal{O}\left(D^3 t\left(\frac{1}{\sqrt[t]{D}} + \frac{1}{\sqrt[t]{D}^2}\right)\right) \subseteq \mathcal{O}\left(\frac{D^3 t}{\sqrt[t]{D}}\right) = \mathcal{O}(D^{3-1/t} \cdot t).$$

We now present a generalization of Algorithm **prim_elt_p** which outputs a primitive element γ modulo p of $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_t) \cong \mathbb{Q}[u_1, \ldots, u_t]/\langle m_1, \ldots, m_t\rangle$ for arbitrary finite t, together with the minimal polynomial (modulo p) of γ, and $\overline{\alpha}_1(\overline{\gamma}), \ldots, \overline{\alpha}_t(\overline{\gamma})$, the normal representations (modulo p) of $\alpha_1(\gamma), \ldots, \alpha_t(\gamma)$.

Let us analyze the cost of Algorithm **prim_elt_multi**. As before, we denote by d_i the degree of $m_i(u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$ for $1 \le i \le t$, and let $D = \prod_{i=1}^{t} d_i$.

- Lines 3 to 11: When we execute this loop the first time, we perform $\mathcal{O}(d_t d_{t-1}^3 + d_t^2 d_{t-1}^2)$ arithmetic operations over $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{t-2}]$, which is equivalent to

$$\begin{aligned}
&\mathcal{O}\left((d_t^2 d_{t-1}^2 + d_t d_{t-1}^3) \cdot (d_1 \cdots d_{t-2})^2\right) \\
&\subseteq \mathcal{O}\left(\left[(d_t^2 d_{t-1}^2) \cdot (d_1 \cdots d_{t-2})^2\right] + \left[(d_t d_{t-1}^3) \cdot (d_1 \cdots d_{t-2})^2\right]\right) \\
&\subseteq \mathcal{O}([D^2] + [(d_t d_{t-1}^3) \cdot (d_1 \cdots d_{t-2})^2]) \\
&\subseteq \mathcal{O}(D^2 + D^2 d_{t-1}) \\
&\subseteq \mathcal{O}(D^2 d_{t-1}) \text{ arithmetic operations in } \mathbb{F}_p.
\end{aligned}$$

  In general, one can show that when executing the loop for the k-th time ($1 \le k \le t-1$) we perform $\mathcal{O}(D^2(d_{t-k}))$ arithmetic operations in $\mathbb{F}_p$. Thus the *total* cost of executing the while-loop is

$$O\left(\sum_{k=1}^{t-1}\left(D^2 d_{t-k}\right)\right) \subseteq \mathcal{O}\left(D^2\left(\sum_{i=1}^{t-1} d_i\right)\right) \text{ arithmetic operations in } \mathbb{F}_p. \quad (6.11)$$

- Lines 12 to 15: this for-loop computes $\overline{\alpha}_i(\overline{\gamma})$, $2 \leq i \leq t$ in $\mathbb{F}_p$. The analysis in Section 6.3.1 shows that this requires

$$\mathcal{O}\left(D^3 t\left(\frac{1}{d_1}+\frac{1}{d_{t-1}d_{t-2}}\right)\right) \text{ arithmetic operations in } \mathbb{F}_p.$$

The cost of Algorithm **prim_elt_p_multi** is dominated by the computations above. Thus the total cost of running this algorithm on a t-step extension for t greater than three is

$$\mathcal{O}\left(D^2\left(\sum_{i=1}^{t-1} d_i\right)\right)+\mathcal{O}\left(D^3 t\left(\frac{1}{d_1}+\frac{1}{d_{t-1}d_{t-2}}\right)\right) \subseteq \mathcal{O}\left(D^3 t\left(\frac{1}{d_1}+\frac{1}{d_{t-1}d_{t-2}}\right)\right)$$

arithmetic operations in $\mathbb{F}_p$. Observe that unfortunately the cost of the algorithm is dominated by the cost of finding the normal representations $\overline{\alpha}_i$'s.

Algorithm 6.3 : **prim_elt_multi**$(M_1(u_1), \ldots, M_t(u_t), K_p)$

Input: $M_i(u_i) = \Phi_p(m_i(u_i))$, $1 \le i \le t$, where $m_i(u_i)$ is the minimal poly. for α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ for $2 \le i \le t$, $m_1(u_1)$ is the minimal poly. for α_1 over $\mathbb{Q}$, and p is a good prime satisfying $p > \prod_{i=1}^{t} (\deg_{u_i}(m_i))^2$.

Output: $\overline{\gamma}$: a primitive element (mod p) of $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$, $N_p(y) = \Phi_p(m_\gamma(y))$ where $m_\gamma(y)$ is the minimal polynomial for γ, and $A = [\overline{\alpha}_1(\overline{\gamma}), \ldots, \overline{\alpha}_t(\overline{\gamma})]$ where $\overline{\alpha}_i(\overline{\gamma}) = \Phi_p(\alpha_i(\gamma))$, or $FAIL$.

1: $A \leftarrow$ empty list of length t; $B \leftarrow$ empty list of length $t-2$;
2: $c \leftarrow$ empty list of length $t-1$; $k \leftarrow t-1$;
3: **while** $k \ge 1$ **do**
4: **if** $k = t-1$ **then**
5: $c[k], N_p(y), A[k], A[k+1] \leftarrow$**prim_elt_p**$(M_t(x, \overline{\alpha}_k), M_k(y), \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{k-1}], p)$;
6: **else**
7: $c[k], N_p(y), A[k], B[k] \leftarrow$ **prim_elt_p**$(N(x, \overline{\alpha}_k), M_k(y), \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{k-1}], p)$;
8: **end if**
9: **if** (Algorithm **prim_elt_p** outputs $FAIL$) **then return** $FAIL$; **end if**
10: $k \leftarrow k-1$;
11: **end while**
$\{A = [\overline{\alpha}_1(\overline{\gamma}), \overline{\alpha}_2(\overline{\gamma}_2), \ldots, \overline{\alpha}_{t-2}(\overline{\gamma}_{t-2}), \overline{\alpha}_{t-1}(\overline{\gamma}_{t-1}), \overline{\alpha}_t(\overline{\gamma}_{t-1})]$ and $B = [\overline{\gamma}_2(\overline{\gamma}), \ldots, \overline{\gamma}_{t-1}(\overline{\gamma}_{t-2})]$, where $\overline{\gamma}_k = \Phi_p(\gamma_k)$ and γ_k satisfies $\mathbb{Q}(\alpha_1, \ldots, \alpha_t) \cong \mathbb{Q}(\alpha_1, \ldots, \alpha_{k-2}, \gamma_{k-1})$ for $3 \le k \le t$. $\overline{\gamma} = \Phi_p(\gamma)$ and γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$. $\}$
12: **for** $i = 2$ to $t-2$ **do**
13: $A[i] \leftarrow$subs$(\{\overline{\gamma}_i = B[i-1], \overline{\alpha}_1 = A[1], \ldots, \overline{\alpha}_{i-1} = A[i-1]\}, A[i])$;
14: $B[i] \leftarrow B[i-1] - c[i+1] \cdot A[i+1]$; $\{\overline{\gamma}_{i+2} = \overline{\gamma}_{i+1} - c_i \overline{\alpha}_i$ $\}$
15: **end for**
16: **if** $t > 2$ **then**
17: $A[t-1] \leftarrow$subs$(\{\overline{\gamma}_t = B[t-2], \overline{\alpha}_1 = A[1], \ldots, \overline{\alpha}_{t-2} = A[t-2]\}, A[t-1])$;
18: $A[t] \leftarrow B[t-2] - c[t-1] \cdot A[t-1]$; $\{$ $\overline{\alpha}_t \leftarrow \overline{\gamma}_t - c_{t-1} \overline{\alpha}_{t-1}$ $\}$
19: **end if**
20: $\overline{\gamma} \leftarrow c[1] \cdot \overline{\alpha}_1 + c[2] \cdot \overline{\alpha}_2 + \cdots + c[t-1] \cdot \overline{\alpha}_{t-1} + \overline{\alpha}_t$;
21: **return** $\overline{\gamma}$, $N_p(y)$, A;

Chapter 7

Algorithmic improvements

To improve the time complexity of Algorithm **sqfr_norm_p** and Algorithm **prim_elt_p**, we developed efficient methods for computing the resultant and gcd.

In Algorithm **sqfr_norm_p** we must compute the resultant

$$N_p(x) = \mathrm{res}_y(M_2(x - cy, y), M_1(y)) \in K_p[x].$$

Although one can compute $N_p(x)$ by computing the determinant of a Sylvester's matrix, it is more efficient to use the subresultant algorithm of Brown ([4], [5]). The subresultant algorithm computes $N_p(x)$ by modifying the Euclidean algorithm to use pseudo-division instead of ordinary polynomial division and by scaling the polynomials appearing in the Euclidean remainder sequence by elements of $K_p[x]$.

Suppose that $d_1 = \deg_y(M_1), d_2 = \deg_x(g)$ and $d_y = \deg_y(g)$. Since $d_y < d_1$, the subresultant algorithm performs $\mathcal{O}(d_1 d_y) \subseteq \mathcal{O}(d_1^2)$ arithmetic operations in $K_p[x]$, which is the same as the cost of the Euclidean algorithm in $K_p[x]$. The polynomials appearing in the sequence belong to $K_p[x, y]$ and have degree $(d_1 d_2)/2$ in x on average. Hence if classical algorithms for multiplication and exact division in $K_p[x]$ are used, the algorithm performs $\mathcal{O}((d_1^2)(d_1 d_2/2)^2) \in \mathcal{O}(d_1^4 d_2^2)$ arithmetic operations in K_p.

To improve on this for multivariate polynomials, Collins [7] showed that one can compute their resultant using evaluation and interpolation. The idea (as applied to our application) is to evaluate x in $g(x, y)$ at points chosen from $\mathbb{F}_p$, use the subresultant algorithm on the univariate polynomials in $K_p[y]$, then interpolate in x to obtain the resultant $N_p(x) \in K_p[x]$. We show that this approach reduces the cost

to $O(d_1^3 d_2 + d_1^2 d_2^2)$ arithmetic operations in K_p.

Recall that $N_p(x)$ is the minimal polynomial (mod p) of degree $D = d_1 d_2$ for a primitive element γ of $K(\alpha, \beta) \cong K[y, x]/\langle m_1, m_2 \rangle$. As before, assume that $\overline{\gamma} = \Phi_p(\gamma)$. To determine the normal representation $A(\overline{\gamma}) = \Phi_p(\alpha(\gamma))$ by computing $\gcd(M_2(\overline{\gamma} - cy, y), M_1(y))$ using classical quadratic algorithms for polynomial arithmetic in $K_p[\overline{\gamma}]$, we require $\mathcal{O}(d_1^2 \cdot (d_1^2 d_2^2)) \subseteq \mathcal{O}(d_1^4 d_2^2)$ arithmetic operations in K_p. In Section 7.2, we show that we can modify the subresultant algorithm to use evaluation and interpolation (for x) to simultaneously compute $\overline{\alpha}(\overline{\gamma})$ as well as $N_p(x)$ in $\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2)$ arithmetic operations in K_p.

7.1 Resultant computation

We now explain how to use evaluation and interpolation to compute bivariate resultants.

7.1.1 Evaluation and interpolation

Let J be an integral domain, $R = J[x, y]$, and $\sigma \in J$. An *evaluation homomorphism* $\Phi_{x=\sigma} : J[x, y] \to J[y]$ is a homomorphism defined by

$$\Phi_{x=\sigma}(f(x, y)) = f(\sigma, y), \ \ f(x, y) \in R.$$

We say that σ is an **evaluation point**. We may alternatively write $f(x = \sigma, y)$ or $f(x, y)|_{x=\sigma}$ to mean $\Phi_{x=\sigma}(f(x, y))$.

The "inverse" of evaluation homomorphism is interpolation, defined as follows:
The Interpolation Problem (over a field):

> Let K be a field. Given distinct evaluation points $\{\sigma_0, ..., \sigma_n\} \in K$ and the values $\{y_0, y_1, ..., y_n\} \subset K$, find a polynomial $f(x) \in K[x]$ satisfying
>
> $$f(\sigma_i) = y_i, \quad i = 0, \ldots, n.$$

The following well-known theorem gives the condition for which there exists a (unique) $f(x)$:

Theorem 7.1. *If the evaluation points $\sigma_0, \sigma_1, ..., \sigma_n$ are distinct, there exists a unique polynomial $f(x) \in K[x]$ of degree at most n such that $f(\sigma_i) = y_i, \ \ i = 0, \ldots, n$.*

Proof. See Geddes et al. [11, Theorem 5.8, p. 185]. □

There are several algorithms for interpolating $f(x) \in K[x]$ of degree at most n. For example, the algorithms known as Newton interpolation and Lagrange interpolation both perform $O(n^2)$ arithmetic operations in K (von zur Gathen, Gerhard [18, p. 132]).

In our application, we wish to interpolate over $K_p = \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r\rangle$. We can state the interpolation problem over K_p as follows.

The Interpolation Problem (over the ring K_p):

> Let $K_p = \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r\rangle$, where $M_1 \in \mathbb{F}_p[u_1]$ and $M_i \in \mathbb{F}_p[u_1, \ldots, u_{i-1}]/\langle M_1, \ldots, M_{i-1}\rangle[u_i]$ for $2 \leq i \leq r$. Given evaluation points $\{\sigma_0, ..., \sigma_n\} \subset \mathbb{F}_p$ and the values $\{y_0, y_1, ..., y_n\} \subset K_p$, find a polynomial $f(x) \in K_p[x]$ satisfying
>
> $$f(\sigma_i) = y_i, \quad i = 0, \ldots, n.$$

When working over K_p and one runs the Newton or Lagrange interpolation, one may encounter divisions by zero divisors. However, the following lemma says that if the evaluation points are chosen from $\mathbb{F}_p$, no zero divisors are encountered.

Lemma 7.2. (existence) *If all the evaluation points belong to $\mathbb{F}_p$ and are distinct, then Newton interpolation can be performed over $K_p = \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r\rangle$, even if K_p has zero divisors.*

Proof. We wish to find $f(x) \in K_p[x]$ satisfying $f(\sigma_i) = y_i$ for $i = 1, \ldots, n$ where $\sigma_i \in \mathbb{F}_p$. Let us write $f(x)$ as

$$f(x) = a_0 + a_1(x-\sigma_1) + a_2(x-\sigma_1)(x-\sigma_2) + \cdots + a_n(x-\sigma_1)(x-\sigma_2)\cdots(x-\sigma_n) \in K_p[x],$$

where $a_0, a_1, \ldots, a_n \in K_p$ are unknown. Substituting σ_1 for x in $f(x)$, we obtain

$$f(\sigma_1) = a_0 = y_1 \in K_p.$$

Next, substituting σ_2 for x in in $f(x)$ we obtain

$$f(\sigma_2) = a_0 + a_1(\sigma_2 - \sigma_1) = y_2 \;\Rightarrow\; a_1 = \frac{y_2 - a_0}{\sigma_2 - \sigma_1} = \frac{y_2 - y_1}{\sigma_2 - \sigma_1} \in K_p.$$

Since $\sigma_2 - \sigma_1 \in \mathbb{F}_p \backslash \{0\}$, a_1 is well-defined. Similarly, substituting σ_3 for x in $f(x)$ gives

$$a_2 = \frac{1}{\sigma_3 - \sigma_1}\left(\frac{y_3 - y_2}{\sigma_3 - \sigma_2} - \frac{y_2 - y_1}{\sigma_2 - \sigma_1}\right) \in K_p. \tag{7.1}$$

Again, every denominator in (7.1) is invertible, since it is a non-zero integer. Continuing in this manner, one can show that the only inverses that need to be computed in determining $a_1, a_2, \ldots, a_n$ are of the form $\sigma_i - \sigma_j \in \mathbb{F}_p \backslash \{0\}, 1 \leq i \neq j \leq n$. Thus if the evaluation points $\sigma_1, \ldots, \sigma_n$ belong to $\mathbb{F}_p$ and are all distinct, then one can successfully find the polynomial $f(x) \in K_p[x]$ satisfying $f(\sigma_i) = y_i,\ i = 0, \ldots, n$. □

In fact, the interpolated $f(x) \in K_p[x]$ is unique, as the following lemma states.

Lemma 7.3. (uniqueness) *Let $K_p = \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r \rangle$. Given distinct points $\{\sigma_0, \ldots, \sigma_n\} \subset \mathbb{F}_p$ and corresponding values $\{y_0, \ldots, y_n\} \subset K_p$, there exists a* unique *polynomial $f(x) \in K_p[x]$ of degree $\leq n$ such that $f(\sigma_i) = y_i$, $i = 0, \ldots, n$.*

Proof. Let $\deg_{u_i}(M_i) = d_i$ for $1 \leq i \leq r$. Then $K_p = \mathbb{F}_p[u_1, \ldots, u_r]/\langle M_1, \ldots, M_r \rangle$ is a vector space of dimension $D = d_1 d_2 ... d_r$ over $\mathbb{F}_p$ whose monomial basis is

$$B = \{u_1^{h_1} u_2^{h_2} \cdots u_r^{h_r} :\ 0 \leq h_j \leq d_j - 1,\ \ 1 \leq j \leq r\}.$$

Since $\{y_0, \ldots, y_n\} \subset K_p$, each y_i can be written as a linear combination of elements in B. Let $c_i(h_1, h_2, \ldots, h_r) \in \mathbb{F}_p$ denote the coefficient of $u_1^{h_1} u_2^{h_2} \cdots u_r^{h_r}$ in y_i, $i = 0, \ldots, n$. Then interpolating the y_i's is equivalent to interpolating the values

$$c_0(h_1, h_2, \ldots, h_r), c_1(h_1, h_2, \ldots, h_r), \ldots, c_n(h_1, h_2, \ldots, h_r)$$

with the corresponding evaluation points $\sigma_0, \sigma_1, \ldots, \sigma_n$ for $0 \leq h_j \leq d_j - 1$, $1 \leq j \leq r$ *separately*, which is an interpolation over the field $\mathbb{F}_p$. Since each of these interpolated polynomials is unique by Theorem 7.1, $f(x) \in K_p[x]$ must be unique as well. □

7.1.2 Resultant computation using evaluation & interpolation

To compute $N_p(x) = \mathrm{res}_y(M_2(x - cy, y), M_1(y)) \in K_p[x, y]$ using evaluation and interpolation, we first evaluate $g(x, y)$ at $x = \sigma_i \in \mathbb{F}_p$ to obtain a univariate polynomial $g(\sigma_i, y) \in K_p[y]$ for sufficiently many evaluation points σ_i's. We then compute the resultants of the univariate $g(\sigma_i, y)$ and $M_1(y)$ and apply interpolation to recover the resultant of $g(x, y) := M_2(x - cy, y)$ and $M_1(y)$.

Let us determine an upper bound on the number of evaluation points needed. Let

$$g(x, y) = \sum_{i=0}^{d_y} a_i(x) y^i, \ M_1(y) = \sum_{i=0}^{d_1} b_i y^i, \text{ and } d_2 = \deg_x(g).$$

Observe that $b_{d_1} = 1$, $\deg_y(g) < d_1$ and $\deg_x(M_1) = 0$. For reasons that will become apparent in the next section, we will compute $\mathrm{res}_y(M_1, g)$, rather than $\mathrm{res}_y(g, M_1)$. Since the two are equal up to a multiplication by (-1) and $\mathrm{res}_y(g, M_1)$ is monic, we may need to multiply $\mathrm{res}_y(M_1, g)$ by (-1) if necessary. The Sylvester matrix formed by $g(x, y)$ and $M_1(y)$, $\mathrm{Syl}_y(M_1, g)$, is the $(d_y + d_1)$ by $(d_y + d_1)$ matrix of the following form

$$\mathrm{Syl}_y(M_1, g) = \begin{array}{l} \text{row } 1 \\ \text{row } 2 \\ \vdots \\ \text{row } d_y \\ \text{row } (d_y + 1) \\ \\ \text{row } (d_y + 2) \\ \vdots \\ \text{row } (d_y + d_1) \end{array} \left(\begin{array}{ccccccc} 1 & b_{d_1-1} & \cdots & b_1 & b_0 & & \\ & 1 & b_{d_1-1} & \cdots & b_1 & b_0 & \\ & & \vdots & & & & \\ & & & 1 & \cdots & \cdots & b_0 \\ a_{d_y}(x) & a_{d_y-1}(x) & \cdots & a_1(x) & a_0(x) & & \\ & a_{d_y}(x) & a_{d_y-1}(x) & \cdots & a_1(x) & a_0(x) & \\ & & \vdots & & & & \\ & & & a_{d_y}(x) & \cdots & \cdots & a_0(x) \end{array} \right).$$

Observe that $a_0(x)$ contains the term with the largest degree of x in $\mathrm{Syl}_y(M_1, g)$, namely x^{d_2}. Thus x^{d_2} appears in the last d_1 rows of $\mathrm{Syl}_y(M_1, g)$. Since every element in the first d_y rows of $\mathrm{Syl}_y(M_1, g)$ has degree 0 in x and the determinant of a matrix is a sum (up to sign) of the products obtained by multiplying one element from every row of the matrix, we conclude that

$$\deg_x(\mathrm{res}_y(g(x, y), M_1(y))) = \deg_x(\det(\mathrm{Syl}_y(M_1, g))) = d_1 d_2.$$

That is, $d_1 d_2 + 1$ distinct evaluation points must suffice to interpolate the resultant.

One may ask, are there any evaluation points from $\mathbb{F}_p$ that cannot be used? In particular, it may not be obvious whether or not an evaluation point σ for which $a_{d_y}(\sigma) = 0$ still satisfies $\det(\mathrm{Syl}_y(M_1, g))|_{x=\sigma} = \det(\mathrm{Syl}_y(M_1|_{x=\sigma}, g|_{x=\sigma}))$. Let $N_p(x) = \mathrm{res}_y(M_1(y), g(x,y))$. Since $\mathrm{lcoeff}_y(M_1) = 1$, Theorem 6.2 implies that

$$N_p(x = \sigma) = \mathrm{res}_y(M_1(y), g(x,y))|_{x=\sigma} = \mathrm{res}_y(M_1(y), g(x = \sigma, y)) \ \ \forall \sigma \in \mathbb{F}_p.$$

That is, every element from $\mathbb{F}_p$ can be used as an evaluation point.

We have now reduced the problem of solving a bivariate resultant to that of solving the resultant of univariate polynomials $M_1(y)$ and $g(\sigma, y)$ over K_p. For this, we use an efficient algorithm involving computation of a *polynomial remainder sequence*.

7.1.3 Polynomial remainder sequences

To understand the definition of a polynomial remainder sequence, we first need to give some preliminary definitions and lemmas.

Theorem 7.4. *Let J be an integral domain, $f(x), g(x) \in J[x]\backslash\{0\}$ and $m = \deg(f) \geq n = \deg(g)$. Then there exist polynomials $q(x), r(x) \in J[x]$ such that*

$$\mathrm{lcoeff}(g)^{m-n+1} \cdot f(x) = q(x) \cdot g(x) + r(x) \tag{7.2}$$

where either $r(x) = 0$ or $\deg(r) < \deg(g)$. We call $r(x)$ is the **pseudo remainder** *of f and g, which we denote as* $\mathbf{prem}(f, g)$.

Proof. See Winkler [21, Theorem 2.2.2, pp. 40-41]. □

Definition 7.5. Let U be a UFD. The polynomials $f(x), g(x) \in U[x]$ are said to be **similar**, written $f(x) \simeq g(x)$, if there exist $\alpha, \beta \in U\backslash\{0\}$ such that $\alpha \cdot f(x) = \beta \cdot g(x)$.

Lemma 7.6. *Let U be a* UFD *and $f, g \in U[x]\backslash\{0\}$. If $\deg(f) \geq \deg(g)$ and $r \simeq \mathrm{prem}(f, g)$, then $\gcd(f, g) \simeq \gcd(g, r)$.*

Proof. See Winkler [21, Lemma 4.1.2, p. 83]. □

Definition 7.7. Let J be an integral domain and $f_1, f_2, \ldots, f_{n+1}$ be polynomials in $J[x]$, $n \geq 2$. We say that $\{f_1, f_2, \ldots, f_{n+1}\}$ is a **polynomial remainder sequence starting from** f_1 **and** f_2, denoted $\mathrm{PRS}_x(f_1, f_2)$, if and only if:

- $\deg(f_1) \geq \deg(f_2)$,
- $f_i \neq 0$ for $i = 1, \ldots, k$ and $f_{n+1} = 0$, and
- $f_i \simeq \mathrm{prem}(f_{i-2}, f_{i-1})$ for $i = 3, \ldots, n+1$.

Remark 7.8. *If $f_1, f_2 \in U[x]$ and $\{f_1, f_2, \ldots, f_n, 0\}$ is a* PRS*, then Lemma 7.6 implies that*

$$\gcd(f_1, f_2) \simeq \gcd(f_2, f_3) \simeq \ldots \simeq \gcd(f_{n-1}, f_n) \simeq \gcd(f_n, 0) = f_n.$$

That is, the gcd *of two polynomials can be computed using their* PRS*.*

A *subresultant polynomial remainder sequence* (sPRS) starting with $f_1(x)$ and f_2, denoted by $\mathrm{sPRS}_x(f_1, f_2)$, is a PRS whose last non-zero polynomial is equal to $\mathrm{res}_x(f_1, f_2)$. In this thesis, we will not provide the rather cumbersome definition of the subresultant PRS (for this one can refer to Brown [4]), but we do present the sPRS algorithm and its time complexity. Even though PRS's are defined for integral domains, Algorithm **sr_prs** (Algorithm 7.1) takes as input polynomials over any commutative ring. It returns $FAIL$ if a division by a zero divisor occurs.

We analyze the cost of Algorithm **sr_prs** assuming that $R = K_p, d_1 = \deg_x(f_1)$, $d_2 = \deg_x(f_2)$ and $d_1 \geq d_2 \geq 0$.

- Line 5: The asymptotic cost of computing all the pseudo-remainders in this algorithm is the same as that of the Euclidean algorithm, which is $\mathcal{O}(d_1 d_2)$ arithmetic operations in K_p (von zur Gathen, Gerhard [18, Theorem 3.11, pp.50-51]).
- Line 7: Since $h, g \in K_p$, we perform δ multiplications and one inversion in K_p when computing $1/(g \cdot h^\delta)$. Moreover, we perform at most $\deg_x(f')$ multiplications in K_p in multiplying f' by $1/(g \cdot h^\delta)$. Since $\delta < d_1$ and $\deg_x(f') < d_1$, it requires

  $$\mathcal{O}(\delta + 1 + \deg_x(f')) \subseteq \mathcal{O}(d_1 + d_1) \subseteq \mathcal{O}(d_1) \text{ arithmetic operations in } K_p.$$

Algorithm 7.1 : **sr_prs**(f_1, f_2, R)

Input: $f_1, f_2 \in R[x]$, R a commutative ring.

Output: $r(x) = \text{res}_x(f_1, f_2)$ and subresultant PRS $[f_1, f_2, \ldots, f_n]$ where $f_{n+1} = 0$, or $FAIL$.

1: $g \leftarrow 1$; $h \leftarrow 1$; $f' \leftarrow f_2$; $i \leftarrow 3$; $s \leftarrow 1$;
2: **while** $f' \neq 0$ and $\deg(f') > 0$ **do**
3: $\quad \delta \leftarrow \deg(f_{i-2}) - \deg(f_{i-1})$;
4: $\quad s \leftarrow s \cdot (-1)^{\deg(f_{i-2}) \cdot \deg(f_{i-1})}$;
5: $\quad f' \leftarrow \text{prem}(f_{i-2}, f_{i-1})$;
6: $\quad$ **if** $f' \neq 0$ **then**
7: $\quad\quad f_i \leftarrow (-1)^{\delta+1} f'/(g \cdot h^{\delta})$;
8: $\quad\quad$ **if** $f_i = FAIL$ **then** $\{g \cdot h^{\delta}$ is a zero divisor in $R\}$ **return** $FAIL$; **end if**
9: $\quad\quad g \leftarrow \text{lcoeff}(f_{i-1})$;
10: $\quad\quad h \leftarrow h^{1-\delta} \cdot g^{\delta}$;
11: $\quad\quad$ **if** $h = FAIL$ **then** $\{\delta > 1$ and $h^{\delta-1}$ is a zero divisor in $R\}$ **return** $FAIL$; **end if**
12: $\quad\quad i \leftarrow i + 1$;
13: $\quad$ **end if**
14: **end while**
15: $r \leftarrow s \cdot (f_{i-1})^{\deg(f_{i-2})} \cdot h^{1-\deg(f_{i-2})}$;
16: **if** $r = FAIL$ **then**
17: $\quad \{\deg(f_{i-2}) > 1$ and $h^{\deg(f_{i-2})-1}$ is a zero divisor in $R\}$
18: $\quad$ **return** $FAIL$;
19: **end if**
20: **return** r, $[f_1, f_2, \ldots, f_{i-1}]$;

- Line 10: If $\delta = 1$, no operations are necessary. If $\delta \neq 1$, in computing $h^{1-\delta}$ we perform $|1 - \delta| - 1$ multiplications and ≤ 1 inversion (if $\delta > 1$), each in K_p. Moreover, we require δ additional multiplications in multiplying $h^{1-\delta}$ by g^{δ}. Hence this line requires

$$\mathcal{O}((|1 - \delta| - 1) + 1 + \delta) \subseteq \mathcal{O}(\delta) \subseteq \mathcal{O}(d_1) \text{ arithmetic operations in } K_p.$$

- Line 15: This line can be analyzed in a similar manner as in Line 10, and it requires

$$\mathcal{O}(d_1) \text{ arithmetic operations in } K_p.$$

Since the while-loop is executed at most d_2 times, we conclude that Algorithm **sr_prs** requires

$$\mathcal{O}(d_2(d_1 + d_1) + d_1) \subseteq \mathcal{O}(d_1 d_2) \text{ arithmetic operations in } K_p.$$

7.1.4 Failure cases of the algorithm

In our application, we wish to use Algorithm **sr_prs** to compute the resultant of $M_2(\sigma - cy, y)$ and $M_1(y)$ over K_p. However, in some cases the choice of the evaluation point $\sigma \in \mathbb{F}_p$ may lead to a division by a zero divisor. In other cases, the choice of p results in a division by a zero divisor for *every* $\sigma \in \mathbb{F}_p$. In either case, the algorithm will return $FAIL$. We illustrate these two types of failure by examples.

Example 7.9. Let $p = 101$,

$$\begin{aligned}
&M_1(z) = z^2 + 37 \in \mathbb{F}_p[z], \\
&M_2(y) = y^3 + (46\,z - 8)\,y^2 - 30\,y - 1 \in \mathbb{F}_p[z]/\langle M_1(z)\rangle[y], \text{ and} \\
&M_3(x) = x^2 + (3\,x + 65 + 46\,z)\,y^2 + x^2 y + 3 \in \mathbb{F}_p[z, y]/\langle M_1(z), M_2(y)\rangle[x].
\end{aligned}$$

We must find $\text{res}_y(M_3(x - cy, y), M_2(y))$. Suppose that we choose $c = 0$. Let us find $\text{res}_y(M_3(x - 0 \cdot y, y), M_2(y))$ using evaluation & interpolation. Consider the evaluation point $\sigma = 0$. Then $M_3(\sigma, y) = (65 + 46\,z)\,y^2 + 3 \in \mathbb{F}_p[z]/\langle M_1(z)\rangle[y]$. We use Algorithm **sr_prs** to find $\text{res}_y(M_3(\sigma, y), M_2(y))$. Since $\deg_y(M_2) > \deg_y(M_3)$, we start the sequence with $f_1 := M_2$ and $f_2 := M_3$. After executing the while-loop the first time, we obtain $f_3 = (17 + 40z)y + (29 - 9z)$. The second time in the while-loop, the algorithm computes

$$f_4 = (-1)^2\,\text{prem}(f_2, f_3)/(g \cdot h).$$

However, $g \cdot h \equiv 21z - 34 \mod \langle M_1(z)\rangle = 21(z + 8) \mod \langle M_1(z)\rangle$ is not invertible in $K_p = \mathbb{F}_p[z]/\langle M_1(z)\rangle$ since $(z + 8)$ divides $M_1(z) = z^2 + 37$. One can check that Algorithm **sr_prs** also fails for the evaluation point $\sigma = 24$ as it encounters a division by a zero divisor, but does not fail for all other evaluation points in $\mathbb{F}_p$.

Example 7.10. Let

$$\begin{aligned} m_1(z) &= z^2 - 2 \in \mathbb{Q}[z], \\ m_2(y) &= y^4 - 3 \in \mathbb{Q}[z]/\langle m_1 \rangle[y], \text{ and} \\ m_3(x) &= x^2 + (z-11)y^3 - 2 \in \mathbb{Q}[y,z]/\langle m_1, m_2 \rangle[x]. \end{aligned}$$

If $p = 17$, then

$$\begin{aligned} M_1(z) &= z^2 + 15 \in \mathbb{F}_p[z], \\ M_2(y) &= y^4 + 14 \in \mathbb{F}_p[z]/\langle M_1 \rangle[y], \text{ and} \\ M_3(x) &= x^2 + (z+6)y^3 + 15 \in \mathbb{F}_p[y,z]/\langle M_1, M_2 \rangle[x]. \end{aligned}$$

Consider the evaluation point $\sigma \in \mathbb{F}_p$. We first find the resultant of $M_3(\sigma - cy, y)$ and $M_2(y)$ using Algorithm **sr_prs**. After executing the while-loop the first time, we obtain

$$\begin{aligned} f_3 := &\left(c^4 + (2\,z + 12)\,\sigma c\right) y^2 + \left(15\,c^3\sigma + 12 + (16\,z + 11)\,\sigma^2 + 2\,z\right) y \\ &+ \left(\left(\sigma^2 + 15\right) c^2 + 5 + 15\,z\right). \end{aligned}$$

The second time in the loop the algorithm computes $f' = \text{prem}(M_3(\sigma, y), f_3(y))$. Then a division of f' by

$$A(x) := \text{lcoeff}_y(M_3(x - cy, y))^2 = z^2 + 12z + 2 \equiv 12z + 4 = 12(z+6)$$

is required in Line 7. However, $(z+6)$ divides $M_1(z) = z^2 + 15 = (z+6)(z+11)$, so it is a zero divisor in $\mathbb{F}_p[z]/\langle M_1 \rangle$. Hence the algorithm returns $FAIL$. Since $A(x)$ does not have any dependence on σ, a division by a zero divisor will always be encountered regardless of the value of σ.

In such cases, a new prime must be chosen. In light of the fact that such a situation can occur, if three (random) evaluation points cause Algorithm **sr_prs** to output $FAIL$, then we return $FAIL$. A new prime must be chosen and the algorithm must be re-executed. This strategy avoids the case in which every evaluation point is tried before returning $FAIL$.

7.1.5 Modified resultant algorithm

We now present Algorithm **res_modp** that takes as input $g(x,y) \in K_p[x,y]$, $M_1(y) \in K_p[y]$ and K_p, and returns $\text{res}_y(g, M_1)$ computed via evaluation & interpolation and subresultant PRS's, or $FAIL$. We analyze the cost of Algorithm

Algorithm 7.2 : **res_modp**$(g(x,y), M_1(y), K_p)$

Input: $g(x,y) \in K_p[x,y]$, $M_1(y) \in K_p[y]$, where $\deg_y(M_1) \geq \deg_y(g) > 0$ and $K_p = \mathbb{F}_p[u_1,\ldots,u_r]/\langle N_1,\ldots,N_r\rangle$ or $\mathbb{F}_p$ with $p > \prod_{i=1}^{r} \deg_{u_i}(N_i)$.

Output: $N_p(x) = \text{res}_y(g(x,y), M_1(y)) \in K_p[x]$, or $FAIL$.

1: $d_1 \leftarrow \deg_y(M_1)$; $d_2 \leftarrow \deg_x(g)$; $B \leftarrow d_1 d_2 + 1$;
2: $C, R \leftarrow$ empty lists of length B each; $S \leftarrow \{0\}$; $k \leftarrow 1$;
3: $\sigma \leftarrow 0$;
4: **while** $k \leq B$ **do**
5: $\quad \hat{g}(y) \leftarrow g(\sigma, y)$;
6: $\quad c_\sigma, C \leftarrow$ **sr_prs**$(M_1(y), \hat{g}(y), K_p)$; $\{c_\sigma = \text{res}_y(M_1, \hat{g}),\ \ C = \text{sPRS}(M_1, \hat{g})\}$
7: $\quad$ **if** Algorithm **sr_prs** does not return $FAIL$ **then**
8: $\quad\quad$ {Algorithm **sr_prs** returns $FAIL$ if a zero divisor is encountered}
9: $\quad\quad C[k] \leftarrow \sigma$; $\quad R[k] \leftarrow (-1)^{\deg_y(\hat{g})\cdot d_1} \cdot c_\sigma$; $\quad \{C[k] \in \mathbb{F}_p,\ R[k] = \text{res}_y(\hat{g}, M_1) \in K_p\}$
10: $\quad\quad k \leftarrow k + 1$;
11: $\quad$ **else if** Algorithm **sr_prs** returned $FAIL$ for three σ's **then**
12: $\quad\quad$ **return** $FAIL$;
13: $\quad$ **end if**
14: $\quad \sigma \leftarrow$ random element of $\mathbb{F}_p \backslash S$; $\quad S \leftarrow S \cup \{\sigma\}$;
15: **end while**
16: interpolate $N_p(x) \in K_p[x]$ from points $[C[1],\ldots,C[B]] \in \mathbb{F}_p^B$ and values $[R[1],\ldots,R[B]] \in K_p^B$;
17: **return** $N_p(x)$;

res_modp for input polynomials $g(x,y) \in K_p[x,y]$ and $M_1(y) \in K_p[y]$ where $d_1 = \deg_y(M_1), d_2 = \deg_x(g),$ and $\deg_y(g) < d_1$.

- Line 5: Evaluating $g(x,y)$ at $x = \sigma \in \mathbb{F}_p$ using Horner's algorithm requires

  $$\mathcal{O}(d_1 d_2) \text{ scalar multiplications and additions in } K_p$$

  (von zur Gathen, Gerhard [18, Theorem 5.1, pp.100-101]).

- Line 6: Executing Algorithm **sr_prs** with $g(\sigma, y)$ and $M_1(y)$ as inputs requires

  $$\mathcal{O}(\deg_y(g)\deg_y(M_1)) \subseteq \mathcal{O}(d_1^2) \text{ arithmetic operations in } K_p \text{ (Section 7.1.3).}$$

- Line 16: We interpolate B images in K_p corresponding to distinct evaluation points from $\mathbb{F}_p$ using Newton interpolation. Since $B = d_1d_2 + 1 \in \mathcal{O}(d_1d_2)$, at most $\mathcal{O}(d_1^2d_2^2)$ arithmetic operations in K_p are needed (von zur Gathen, Gerhard [18, p.132]). We remark here that because all the evaluation points are in $\mathbb{F}_p$, most of the arithmetic operations required in the interpolation are done in $\mathbb{F}_p$, not K_p. However, this does not change the overall complexity of Algorithm **res_modp**.

Since Algorithm **res_modp** will return $FAIL$ only if three failures were encountered while executing Algorithm **sr_prs**, the while-loop will be executed at most $(d_1d_2 + 1) + 3 \in \mathcal{O}(d_1d_2)$ times.

In summary, Algorithm **res_modp** requires:

$$\mathcal{O}([(d_1d_2)(d_1d_2 + d_1^2)] + d_1^2d_2^2) \in \mathcal{O}(d_1^3d_2 + d_1^2d_2^2) \text{ arithmetic operations in } K_p.$$

If $d_1 \leq d_2$ then this cost simplifies to $\mathcal{O}(d_1^2d_2^2)$ arithmetic operations in K_p. Compared with the cost of the linear algebra method ($\mathcal{O}(d_1^3d_2^3)$), this is an improvement.

7.2 gcd computation

Let $N_p(x) = \text{res}_y(g(x, y), M_1(y)) \in K_p[x]$ be the minimal polynomial (modulo p) for $\overline{\gamma}$ and $g(x, y) = M_2(x - cy, y)$ be square-free. Lines 5 and 12 of Algorithm **prim_elt_p** (Algorithm 6.2) compute the monic gcd

$$G = \gcd(g(\overline{\gamma}, y), M_1(y)) = y + c(\overline{\gamma}) \in K_p[\overline{\gamma}][y] \cong K_p[x]/\langle N_p(x)\rangle[y].$$

Let $d_1 = \deg_y(M_1)$, $d_2 = \deg_x(g)$ and $d_y = \deg_y(g)$. Since $\deg_x(N_p) = d_1d_2$ and $d_y < d_1$, computing G using the Euclidean algorithm requires $\mathcal{O}(d_1d_y)$ arithmetic operation in $K_p[\overline{\gamma}]$, or equivalently,

$$\mathcal{O}\left((d_1d_y) \cdot (\deg_x(N_p))^2\right) \in \mathcal{O}(d_1^2 \cdot (d_1d_2)^2) = \mathcal{O}(d_1^4d_2^2) \text{ arithmetic operations in } K_p.$$

In this section, we show that this cost can be reduced by using the following idea. Let

$$\begin{aligned}\mathrm{PRS}_x &= \mathrm{sPRS}_y(M_1(y), g(x,y)) \\ &= \{f_1(x,y) = M_1(y),\ f_2(x,y) = g(x,y),\ f_3(y), \ldots, f_{n-1}(x,y),\ f_n(x,y), 0\} \text{ and} \\ \mathrm{PRS}_{\overline{\gamma}} &= \mathrm{sPRS}_y(M_1(y), g(\overline{\gamma},y)) \\ &= \{h_1(\overline{\gamma},y) = M_1(y),\ h_2(\overline{\gamma},y) = g(\overline{\gamma},y),\ h_3(y), \ldots,\ h_{m-1}(\overline{\gamma},y),\ h_m(\overline{\gamma},y), 0\}.\end{aligned}$$

By Remark 7.8 and Lemma 5.7,

$$h_m(\overline{\gamma},y) = a(\overline{\gamma})y + b(\overline{\gamma}) \simeq G = \gcd(g(\overline{\gamma},y), M_1(y)).$$

Moreover, we shall see in this section that $f_{n-1}(x = \overline{\gamma}, y) = h_m(\overline{\gamma}, y)$. Thus if we can determine $f_{n-1}(x,y)$, there is no need to compute $\mathrm{sPRS}_{\overline{\gamma}}$ (or any variant of the Euclidean algorithm) to determine G. Observe that the sPRS's of $g(\sigma_i, y)$ and $M_1(y)$, $1 \le i \le \deg_x(N_p)$, are known from having computed $\mathrm{res}_y(g(x,y), M_1(y))$ using Algorithm **res_modp**. Thus we can compute $f_{n-1}(x,y)$ by interpolating the $f_{n-1}(\sigma_i, y)$'s appearing in these sequences. Making the substitution $x \mapsto \overline{\gamma}$ to the interpolated polynomial, then making it monic, (provided that the leading coefficient is not a zero divisor in K_p) we obtain G. We illustrate this idea with an example.

Example 7.11. Let $p = 17$,

$$g(x,y) = x^2 + \left(8 + 15\,y^2 + 3\,y\right)x + (5 + 11\,y^2 + 4\,y), \quad \text{and } M_1(y) = y^3 + 9.$$

Algorithm **res_modp** computes $\deg_x(g)\deg_y(M_1) + 1 = 7$ evaluation homomorphisms:

σ	$f_2(y) = g(\sigma, y)$	$f_3(y) = f_{n-1}(\sigma, y)$	$f_4(y) = f_n(\sigma, y)$	$f_5(y) = f_{n+1}(\sigma, y)$
0	$11\,y^2 + 4\,y + 5$	$12\,y + 4$	1	0
1	$9y^2 + 7y + 14$	$8\,y + 11$	2	0
2	$7\,y^2 + 10\,y + 8$	$10\,y + 11$	7	0
3	$5\,y^2 + 13\,y + 4$	$13\,y + 5$	3	0
4	$3\,y^2 + 16\,y + 2$	$12\,y + 11$	2	0
5	$y^2 + 2\,y + 2$	$2\,y + 13$	6	0
6	$16\,y^2 + 5\,y + 4$	$12\,y + 12$	1	0

To obtain $\mathrm{res}_y(g(x,y), M_1(y))$, recall that we interpolate $f_n(\sigma, y)$ for $0 \le \sigma \le 6$:

$$N_p(x) := \mathrm{res}_y(g(x,y), M_1(y)) = x^6 + 7\,x^5 + 11\,x^4 + 5\,x^3 + 9\,x^2 + 8\,x + 12.$$

Interpolating the next-to-last non-zero polynomials $f_{n-1}(\sigma, y)$ for $0 \le \sigma \le 6$ gives

$$\left(2\,x^3 - 3\,x^2 - 3\,x - 5\right) y + (3\,x^3 - 4\,x^2 + 8\,x + 4),$$

which, upon division by its leading coefficient of y in $\mathbb{F}_p[x]/\langle N_p(x)\rangle$ yields

$$A(x,y) = y + (16\,x^5 + 13\,x^4 + 9\,x^3 + 13\,x^2 + x + 15).$$

Indeed, one can verify that

$$G = \gcd(g(\overline{\gamma}, y), M_1(y)) = A(x = \overline{\gamma}, y) = y + (16\,\overline{\gamma}^5 + 13\,\overline{\gamma}^4 + 9\,\overline{\gamma}^3 + 13\,\overline{\gamma}^2 + \overline{\gamma} + 15).$$

To prove the correctness of the method described above in computing G, we define subresultants and discuss properties of subresultants and subresultant PRS's.

7.2.1 Subresultants and properties of subresultant PRS's

Definition 7.12. Let R be a commutative ring and let $f_1(y), f_2(y) \in R[y]/\{0\}$ with $d_1 = \deg(f_1)$ and $d_2 = \deg(f_2)$. For $j = 0, \ldots, \min\{d_1, d_2\} - 1$, the **j-th subresultant** of f_1 and f_2, denoted by $\mathrm{sRes}_j(f_1, f_2)$, is

$$\mathrm{sRes}_j(f_1, f_2) = \sum_{i=0}^{j} \det(\mathrm{Syl}_y(f_1, f_2)_{i,j}) y^i,$$

where $\mathrm{Syl}_y(f_1, f_2)_{i,j}$ is the matrix derived from the Sylvester matrix of f_1 and f_2 by deleting

- the last j rows of coefficients of f_1,
- the last j rows of coefficients of f_2, and
- the last $2j + 1$ columns except the $(d_1 + d_2 - i - j)$th.

Theorem 7.13. *Let U be a* UFD *and $f_1, f_2 \in U[y]$ with $\deg_y(f_1) > \deg_y(f_2)$. Further let $\mathrm{sPRS}_y(f_1, f_2) = \{f_1, f_2, \ldots, f_n, 0\}$ and $n_i = \deg_y(f_i)$ for $1 \leq i \leq n$. Then for $0 \leq j \leq \deg_y(f_2) - 1$,*

$$\mathrm{sRes}_j(f_1, f_2) = \begin{cases} f_i & \text{if } j = n_{i-1} - 1, \\ \tau_i f_i & \text{if } j = n_i \text{ where } \tau_i \in U, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. See Brown and Traub [5]. □

Recall that we defined PRS_x and $\mathrm{PRS}_{\overline{\gamma}}$ as follows:

$$\mathrm{PRS}_x = \mathrm{sPRS}_y(M_1(y), g(x,y)) = \{M_1(y),\ g(x,y),\ f_3(x,y), \ldots, f_{n-1}(x,y),\ f_n(x,y), 0\} \text{ and}$$

$$\mathrm{PRS}_{\overline{\gamma}} = \mathrm{sPRS}_y(M_1(y), g(\overline{\gamma}, y)) = \{M_1(y),\ g(\overline{\gamma}, y),\ h_3(y), \ldots,\ h_{m-1}(\overline{\gamma}, y),\ h_m(\overline{\gamma}, y), 0\}.$$

Since h_m is linear in y (Lemma 5.7), Theorem 7.13 implies that

$$\mathrm{sRes}_1(M_1(y), g(\overline{\gamma}, y)) = \begin{cases} h_m, & \text{if } \deg(h_{m-1}) = 2 \\ \tau \cdot h_m \text{ for some } \tau \in K_p, & \text{if } \deg(h_{m-1}) > 2. \end{cases} \tag{7.3}$$

Theorem 7.14. *Let $\Phi : R \to R'$ be a ring homomorphism and let Φ also denote the induced homomorphism $R[y] \to R'[y]$. Further let $f, g \in R[y] \backslash \{0\}$ and $\deg(f) > \deg(g)$. If $\deg(\Phi(f)) = \deg(f)$ and $\delta = \deg(g) - \deg(\Phi(g))$, then for $0 \leq j < \deg(f)$,*

$$\Phi(\mathrm{sRes}_j(f, g)) = \Phi(\mathrm{lcoeff}(f))^{\delta} \cdot \mathrm{sRes}_j(\Phi(f), \Phi(g)).$$

Proof. See Mishra [14, Lemma 7.8.1, pp.263-265]. □

By Theorem 7.13, each f_i in PRS_x is similar to a subresultant of $M_1(y)$ and $g(x,y)$. Moreover, Theorem 7.14 implies that each f_i must be similar to some h_j. In particular, we know that

$$f_n(x) = \mathrm{res}_y(M_1(y), g(x,y)) = \mathrm{sRes}_0(M_1(y), g(x,y)) = \Phi_p(m_\gamma(x)),$$

a polynomial of degree 0 in y. Since $(\Phi_p(m_\gamma(x)))|_{x=\overline{\gamma}} = 0$, we have $f_n(x = \overline{\gamma}, y) = h_{m+1}(\overline{\gamma}, y) = 0$. Furthermore, there must exist some f_i that is similar to

$\mathrm{sRes}_1(M_1(y), g(x,y))$, a degree 1 polynomial. Since every PRS is a sequence of polynomials of decreasing degree and $\deg_y(f_n) = 0$, the next-to-last non-zero polynomial f_{n-1} in PRS_x must be similar to $\mathrm{sRes}_1(M_1(y), g(x,y))$. Combining this observation with (7.3) and Theorem 7.14, we obtain

$$\begin{aligned} f_{n-1}|_{x=\overline{\gamma}} &\simeq \mathrm{sRes}_1(M_1(y), g(x,y))|_{x=\overline{\gamma}} \\ &= \mathrm{lcoeff}(M_1)^{\delta}|_{x=\overline{\gamma}} \cdot \mathrm{sRes}_1(M_1|_{x=\overline{\gamma}}, g|_{x=\overline{\gamma}}) \\ &\simeq h_m, \end{aligned}$$

where $\delta = \deg_x(g) - \deg_x(\Phi_{\overline{\gamma}}(g))$. That is, the next-to-last polynomial $f_{n-1}(x,y)$ in $\mathrm{PRS}_x = \mathrm{sPRS}_y\left(g(x,y), M_1(y)\right)$ can be used to find $\gcd(g(\overline{\gamma}, y), M_1(y))$ via the substitution $x \mapsto \overline{\gamma}$.

Example 7.15. Suppose that $K = \mathbb{Q}$ and $K_p = \mathbb{F}_{4133}$. Let

$g(x, \overline{\alpha}) = M_2(x - \overline{\alpha}, \overline{\alpha}) = m_2(x, \overline{\alpha}) \mod 4133 = (x - \overline{\alpha})^4 - \overline{\alpha}^2(x - \overline{\alpha})^2 - 2$ and
$M_1(y) = m_1(y) \mod 4133 = y^4 - 2$.

We determined in Example 6.4 that a primitive element of $K(\alpha, \beta)$ is $\gamma = \alpha + \beta$ with

$$N_p(x) = m_{\gamma}(x) \mod p = x^{16} - 44\,x^{12} - 468\,x^8 - 1456\,x^4 + 16, \text{ and}$$

$$\gcd(g(\overline{\gamma}, y), M_1(y)) = y - 951\,\overline{\gamma} + 588\,\overline{\gamma}^5 + 982\,\overline{\gamma}^9 + 980\,\overline{\gamma}^{13}.$$

The $\mathrm{sPRS}_y(M_1(y), g(x,y))$ over $K_p[x]$ is

$$\begin{aligned} & & f_1(x,y) &= M_1(y) \equiv y^4 + 4131, \\ & & f_2(x,y) &= g(x,y) = (x-y)^4 - y^2(x-y)^2 - 2 \\ & & &\equiv 4131\,xy^3 + 5\,x^2y^2 + 4129\,x^3y + x^4 + 4131, \\ & & f_3(x,y) &= 17\,x^4y^2 + \left(4115\,x^5 + 4129\,x\right)y + 5\,x^6 + 4115\,x^2, \\ f_{n-1}(x,y) &= & f_4(x,y) &= \left(4107\,x^9 + 3993\,x^5 + 4125\,x\right)y + 11\,x^{10} + 86\,x^6 + 4097\,x^2, \\ f_n(x,y) &= & f_5(x,y) &= x^{16} + 16 + 4089\,x^{12} + 3665\,x^8 + 2677\,x^4, \\ & & f_6(x,y) &= 0. \end{aligned}$$

As expected, $f_n = N_p(x)$. The $\mathrm{sPRS}_y(M_1(y), g(\overline{\gamma}, y))$ over $K_p[\overline{\gamma}] \cong K_p[x]/\langle N_p(x)\rangle$ is:

$$\begin{aligned}
& h_1(\overline{\gamma}, y) = y^4 - 2, \\
& h_2(\overline{\gamma}, y) = (\overline{\gamma} - y)^4 - y^2(\overline{\gamma} - y)^2 - 2 \\
& \qquad \equiv 4131\,\overline{\gamma} y^3 + 5\,\overline{\gamma}^2 y^2 + 4129\,\overline{\gamma}^3 y + \overline{\gamma}^4 + 4131, \\
h_{m-1}(\overline{\gamma}, y) \;=\; & h_3(\overline{\gamma}, y) = 17\,\overline{\gamma}^4 y^2 + \left(4115\,\overline{\gamma}^5 + 4129\,\overline{\gamma}\right) y + 5\,\overline{\gamma}^6 + 4115\,\overline{\gamma}^2, \\
h_m(\overline{\gamma}, y) \;=\; & h_4(\overline{\gamma}, y) = \left(4107\,\overline{\gamma}^9 + 3993\,\overline{\gamma}^5 + 4125\,\overline{\gamma}\right) y + 11\,\overline{\gamma}^{10} + 86\,\overline{\gamma}^6 + 4097\,\overline{\gamma}^2, \\
& h_5(\overline{\gamma}, y) = 0.
\end{aligned} \tag{7.4}$$

One can verify that $f_i(x = \overline{\gamma}, y) = h_i(\overline{\gamma}, y)$ for $i = 1, \ldots, 5$. In particular,

$$f_{n-1}(x = \overline{\gamma}, y) = h_m(\overline{\gamma}, y) \simeq \gcd(g(\overline{\gamma}, y), M_1(y)).$$

To obtain the monic gcd, we invert $\mathrm{lcoeff}_y(f_{n-1}(\overline{\gamma}, y)) = -26\,\overline{\gamma}^9 - 140\,\overline{\gamma}^5 - 8\,\overline{\gamma} \in K_p[\overline{\gamma}]$ using the Extended Euclidean algorithm and multiplying it by $f_{n-1}(\overline{\gamma}, y)$.

7.2.2 Unlucky evaluation points

We must determine how many evaluation points are sufficient to interpolate the next-to-last linear polynomials $f_{n-1}(\sigma_i, y)$'s in the subresultant PRS's obtained from Algorithm **res_modp** to determine $f_{n-1}(x, y)$. Since

$$\deg_x(f_{n-1}) < \deg(N_p) = \deg_y(M_1) \cdot \deg_x(g),$$

$\deg_y(M_1) \cdot \deg_x(g)$ images are sufficient to interpolate x in $f_{n-1}(x, y)$. Note that this is less than the number of images we computed in Algorithm **res_modp** for computing the resultant, which is $\deg_y(M_1) \cdot \deg_x(g) + 1$. However, not every image can be used, as the following examples illustrate.

Example 7.16. Let $p = 17$, $M_1(y) = y^3 - 2y^2 - 1$ and $g(x, y) = x^2 - 5xy^2 - x + 4$. Then $\mathrm{sPRS}_y(M_1(y), g(x, y))$ is:

$$\begin{aligned}
& f_1(x, y) = M_1(y) = y^3 - 2y^2 - 1, \\
& f_2(x, y) = g(x, y) = x^2 - 5xy^2 - x + 4, \\
f_{n-1}(x, y) \;=\; & f_3(x, y) = (5x^3 + 12x^2 + 3x)y + 7x^3 + 2x^2 + 11x, \\
f_n(x, y) \;=\; & f_4(x, y) = x^6 + 11\,x^5 + 6\,x^4 + 8\,x^3 + 7\,x^2 + 6\,x + 13, \\
& f_5(x, y) = 0.
\end{aligned}$$

It is $f_{n-1}(x,y)$ that we wish to interpolate. However, $\text{sPRS}_y(M_1(y), g(x=6,y))$ is:

$$\begin{array}{rcll} & & \hat{f}_1(y) = M_1(y) = y^3 + 15\,y^2 + 10 & = f_1(x=6,y), \\ \hat{f}_{m-1}(y) & = & \hat{f}_2(y) = 4y^2 & = f_2(x=6,y), \\ \hat{f}_m(y) & = & \hat{f}_3(y) = 13 & = f_3(x=6,y), \\ & = & \hat{f}_4(y) = 0 & = f_4(x=6,y). \end{array}$$

Since $\text{lcoeff}_y(f_3)|_{x=6} = 0$, the next-to-last non-zero polynomial $\hat{f}_{m-1}$ is not equal to the desired $f_{n-1}(x=6,y) = 1$, and it is not even linear in y. This evaluation point cannot be used.

Example 7.17. Let $p = 17$, $M_1(y) = y^4 + 11\,y^2 + 15$, and $g(x,y) = x^3 + 8\,yx + 15\,y^3$. The $\text{sPRS}_y(M_1(y), g(x,y))$ computed over $\mathbb{F}_{17}[x]$ is:

$$\begin{array}{rcl} & & f_1(x,y) = M_1(y) = y^4 + 11\,y^2 + 15, \\ & & f_2(x,y) = g(x,y) = x^3 + 8\,yx + 15\,y^3, \\ & & f_3(x,y) = (10+16x)y^2 + 2x^3y + 9, \\ f_{n-1}(x,y) & = & f_4(x,y) = (15x^6 + 11 + 2x^3 + 11x^2)y + 13x^5 + 12x^4 + 16x^3, \\ f_n(x,y) & = & f_5(x,y) = x^{12} + 8 + 7\,x^8 + 5\,x^7 + 12\,x^6 + 2\,x^4 + 11\,x^3 + 4\,x^2 + 5\,x, \\ & & f_6(x,y) = 0. \end{array}$$

It is $f_{n-1}(x,y)$ that we wish to interpolate. However, $\text{sPRS}_y(M_1(y), g(x=10,y))$ is:

$$\begin{array}{rcll} & & \hat{f}_1(y) = M_1(y) = y^4 + 15 + 11\,y^2 & = f_1(x=10,y), \\ & & \hat{f}_2(y) = 15y^3 + 12y + 14 & = f_2(x=10,y), \\ \hat{f}_{m-1}(y) & = & \hat{f}_3(y) = 11\,y + 9 & = f_3(x=10,y), \\ \hat{f}_m(y) & = & \hat{f}_4(y) = 11 & \neq f_4(x=10,y), \\ & & \hat{f}_5(y) = 0 & = f_5(x=10,y). \end{array}$$

The next-to-last non-zero polynomial $\hat{f}_{m-1}$ is linear, but it corresponds to f_3 of degree 2, since $\text{lcoeff}_y(f_3)|_{x=10} = 0$. As such, we cannot use 10 as an evaluation point.

We say that an evaluation point σ is **unlucky** if a leading coefficient (in y) of any polynomial in $\text{sPRS}_y(M_1(y), g(x,y))$ vanishes with the substitution $x \mapsto \sigma$. By definition, an unlucky evaluation point causes an abnormal degree drop. Thus either the number of polynomials in $S_\sigma = \text{sPRS}_y(M_1(y), g(\sigma,y))$ will be smaller or will have a different degree sequence than that of $S_x = \text{sPRS}_y(M_1(y), g(x,y))$. For example, the

degree sequence of the non-zero polynomials of S_x may be $\{7, 6, 4, 2, 1, 0\}$ and that of S_σ may be $\{7, 6, 4, 1, 0\}$ or $\{7, 6, 3, 2, 1, 0\}$. The polynomials after the abnormal degree drop will not correspond to the polynomials in S_x, since each polynomial depends on the leading coefficient of the previous polynomial.

There is no way of determining the number of polynomials in $\text{sPRS}_y(M_1(y), g(x, y))$ without computing it. However, one can detect unlucky evaluation point by comparing the degree sequence of this sPRS with the degree sequences of the sPRS's computed so far. To explain, we need the following definition.

Definition 7.18. Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$, where each a_i and b_i belong to $\mathbb{Z}$. We write $A \succ B$ if and only if

- $n > m$, or
- $a_i = b_i$ for $1 \leq i \leq k < n$ and $a_{k+1} > b_{k+1}$.

Based on the above definition, we can devise an unlucky evaluation point detection scheme as follows. A similar scheme has been used by Collins [8]. Let Δ_{prev} be the degree sequence of the non-zero polynomials in the sPRS obtained using the first evaluation point and let $\Delta_{current}$ be the degree sequence of the non-zero polynomials obtained using the second evaluation point. If

- $\Delta_{current} \prec \Delta_{prev}$, discard second sPRS (current evaluation point is unlucky).
- $\Delta_{current} \succ \Delta_{prev}$, discard first sPRS's (all previous evaluation points unlucky). Set $\Delta_{prev} \leftarrow \Delta_{current}$.
- $\Delta_{current} = \Delta_{prev}$, keep both sPRS's (current and previous evaluation points likely not unlucky).

We repeat this process with other evaluation points until we have the desired number of linear polynomials.

Example 7.19. Suppose that the degree sequence of $\text{sPRS}_y(M_1(y), g(\sigma_1, y))$ is

$$\Delta_1 = \{a_1, a_2, a_3, a_4, a_5, a_6\} = \{7, 6, 4, 2, 1, 0\}$$

and that of $\mathrm{sPRS}_y(M_1(y), g(\sigma_2, y))$ is

$$\Delta_2 = \{b_1, b_2, b_3, b_4, b_5, b_6\} = \{7, 6, 3, 2, 1, 0\}.$$

We have $\Delta_1 \succ \Delta_2$ since $a_1 = b_1$, $a_2 = b_2$ and $a_3 > b_3$. Hence σ_1 is an unlucky evaluation point, so we discard the sequence $\mathrm{sPRS}_y(M_1(y), g(\sigma_2, y))$.

Now suppose that the degree sequence of $\mathrm{sPRS}_y(M_1(y), g(\sigma_3, y))$ is

$$\Delta_3 = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\} = \{7, 6, 5, 4, 2, 1, 0\}.$$

Since $|\Delta_3| > |\Delta_1|$, we conclude that $\Delta_3 \succ \Delta_1$. Hence σ_1 is an unlucky evaluation point. As such, we discard $\mathrm{sPRS}_y(M_1(y), g(\sigma_1, y))$.

We remark that if the degrees of the non-zero polynomials in an sPRS decreases by one each time, then the number of polynomials in the sPRS is maximal and any abnormal degree drop will produce a shorter degree sequence, so we simply need to check the number of polynomials in the future sequences in such cases.

Unfortunately, all the $B = \deg_y(M_1) \cdot \deg_x(g)$ linear polynomials we found could be unlucky; for example, all of $\mathrm{sPRS}_y(g(\sigma_i, y), M_1(y))$ for $i = 1, \ldots, B$ found could have the degree sequence $\{7, 6, 3, 2, 1, 0\}$, whereas that of $\mathrm{sPRS}_y(g(x, y), M_1(y))$ is $\{7, 6, 4, 2, 1, 0\}$. However, the following lemma shows that this happens rarely.

Lemma 7.20. *Let $f_1 \in K_p[y]$, $f_2 \in K_p[x, y]$ where $d_1 = \deg_y(f_1) > \deg_y(f_2)$ and $d_2 = \deg_x(f_2)$. If $\mathrm{sPRS}_y(f_1, f_2) = \{f_1, f_2, \ldots, f_n, 0\}$, then the number of unlucky evaluation points in $\mathbb{F}_p$ is at most*

$$\frac{d_1(d_1 - 1)d_2}{2}.$$

Proof. Let $f_1(y) = \sum_{i=0}^{d_1} b_i y^i$, $f_2(x, y) = \sum_{i=0}^{d_2} a_i(x) y^i$ and $d_y = \deg_y(f_2)$. Then

$n \leq d_y$ and

$$\mathrm{Syl}_y(f_1, f_2) = \begin{array}{l} \text{row } 1 \\ \text{row } 2 \\ \quad\vdots \\ \text{row } d_y \\ \text{row } (d_y+1) \\ \\ \text{row } (d_y+2) \\ \quad\vdots \\ \text{row } (d_y+d_1) \end{array} \left(\begin{array}{ccccccc} b_{d_1} & b_{d_1-1} & \cdots & b_1 & b_0 & & \\ & b_{d_1} & b_{d_1-1} & \cdots & b_1 & b_0 & \\ & & \vdots & & & & \\ & & & b_{d_1} & \cdots & \cdots & b_0 \\ \hline a_{d_y}(x) & a_{d_y-1}(x) & \cdots & a_1(x) & a_0(x) & & \\ & a_{d_y}(x) & a_{d_y-1}(x) & \cdots & a_1(x) & a_0(x) & \\ & & \vdots & & & & \\ & & & a_{d_y}(x) & \cdots & \cdots & a_0(x) \end{array}\right).$$

For $k = 3, \ldots, n$, f_k is similar to a subresultant of f_1 and f_2 by Theorem 7.13, and

$$\mathrm{sRes}_j(f_1, f_2) = \sum_{i=0}^{j} \det(M(f_1, f_2)_{i,j}) \cdot y^i, \tag{7.5}$$

where $M(f_1, f_2)_{i,j}$ is obtained from $\mathrm{Syl}_y(f_1, f_2)$ by deleting the last j rows of coefficients of f_1, the last j rows of coefficients of f_2, and the last $2j+1$ columns except the $(d_1 + d_y - i - j)$-th, for $0 \leq i \leq j \leq d_y - 1$. Hence for a fixed j and $0 \leq i \leq j$, $M(f_1, f_2)_{i,j}$ is a square matrix of dimension $d_1 + d_y - 2j$. Observe from (7.5) that

$$\mathrm{lcoeff}_y(\mathrm{sRes}_j(f_1, f_2)) = \det(M(f_1, f_2)_{j,j}) \in K_p[x].$$

Thus an evaluation point σ is unlucky if $\Phi_{x=\sigma}(\det(M(f_1, f_2)_{j,j})) = 0$ for some $j = 1, \ldots, d_y - 1$. Since $f_2 \in K_p[x, y]$ and $f_1 \in K_p[y]$, only the elements in the last $d_y - j$ rows of $M(f_1, f_2)_{j,j}$ have terms in them whose degree in x is non-zero (and at most d_2). Thus

$$\deg_x \left(\det \left(M(f_1, f_2)_{j,j}\right)\right) \leq d_2(d_y - j) < d_2(d_1 - j).$$

Hence by Lemma 6.3, the number of roots in $\mathbb{F}_p$ of $\det(M(f_1, f_2)_{j,j})$ is at most $d_2(d_1 - 1)$. In total then, the number of unlucky evaluation points must be at most

$$\sum_{j=1}^{d_y-1} \deg_x \left(M(f_1, f_2)_{j,j}\right) < \sum_{j=1}^{d_1-1} d_2(d_1 - j) = \frac{d_1(d_1-1)d_2}{2}.$$

□

Lemma 7.20 implies that since evaluation points are chosen from $\mathbb{F}_p$ where p is large, the probability of choosing $B = d_1 d_2$ evaluation points that are all unlucky that have the same degree sequence will be highly unlikely.

Another problem one may encounter in computing the gcd via evaluation and interpolation and sPRS's is that there may exist a prime p in which no linear polynomial appears in S_σ for *any* $\sigma \in \mathbb{F}_p$, as the following example illustrates.

Example 7.21. Let $m_1(y) = y^3 - 2 \in \mathbb{Q}[y]$ and $m_2(x, y) = x^2 - y + 101y^2 + 103 \in \mathbb{Q}[y]/\langle m_1 \rangle[x]$. If $p = 101$ then $M_1(y) = \Phi_p(m_1(y)) = y^3 + 99$ and $M_2(x, y) = \Phi_p(m_2) = x^2 + 2$. Regardless of the value of the evaluation point $\sigma \in \mathbb{F}_p$ we have

$$S_\sigma = \text{sPRS}_y(M_1(y), g(\sigma, y)) = \{M_1(y) = y^3 + 99, \sigma^2 + 2, 0\}.$$

Rather than trying all elements in $\mathbb{F}_p$ as evaluation points before returning $FAIL$, we proceed as follows: if three evaluation points produce subresultant PRS's without a linear polynomial in y, then we return $FAIL$. In such cases, one must re-run the algorithm using a different prime. This strategy also prevents the rare case in which the first $d_1(d_1 - 1)d_2/2$ evaluation points tried are unlucky, which would make algorithm computationally expensive. It also detects with high probability the case in which $N_p(x) = \text{res}_y(g(x, y), M_1(y))$ or $g(x, y)$ is not square-free, since a linear polynomial may not exist in $\text{sPRS}_y(g(x, y), M_1(y))$ if $N_p(x)$ is not square-free.

7.2.3 Modified resultant algorithm and complexity analysis

Algorithm **res_modp2** is a modified version of Algorithm **res_modp** that takes as input $g(x, y) \in K_p[x, y]$ and $M_1(y) \in K_p[y]$ and returns $N_p(x) = \text{res}_y(g(x, y), M_1(y)) \in K_p[x]$ and $G(x, y) \in K_p[x, y]$ such that $G(\overline{\gamma}, y) = \gcd(g(\overline{\gamma}, y), M_1(y))$ where $\overline{\gamma}$ is the root of $N_p(x)$, or $FAIL$.

Algorithm 7.3 : **res_modp2**$(g(x,y), M_1(y),\ K_p)$

Input: $g(x,y) \in K_p[x,y]$, $M_1(y) \in K_p[y]$, where $\deg_y(M_1) > \deg_y(g) > 0$ and $K_p = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_r] \cong \mathbb{F}_p[u_1, \ldots, u_r]/\langle N_1, \ldots, N_r\rangle$, $p > \prod_{i=1}^{r} \deg_{u_i}(N_i)$.

Output: $N_p(x) = \text{res}_y(g(x,y), M_1(y)) \in K_p[x]$ and $G(x,y) \in K_p[x,y]$ where $G(\overline{\gamma}, y) = \gcd(g(\overline{\gamma}, y), M_1(y))$ and $\overline{\gamma}$ is a root of N_p, or $FAIL$.

$d_1 \leftarrow \deg_y(M_1)$; $d_2 \leftarrow \deg_x(g)$; $B \leftarrow d_1 d_2 + 1$;
$C, R, G \leftarrow$ empty lists of length B each; $S \leftarrow \{0\}$; $k \leftarrow 1$; $\sigma \leftarrow 0$; $\Delta \leftarrow \{\}$; $c \leftarrow 0$;
while $k \leq B$ **do**
  $\hat{g}(y) \leftarrow g(\sigma, y)$; $c_\sigma, C \leftarrow$ **sr_prs**$(M_1(y), \hat{g}(y), K_p)$;
  if Algorithm **sr_prs** does not return $FAIL$ **then**
    $\Delta_{current} \leftarrow$ degree sequence of C;
    if $(\Delta_{current} \succ \Delta_{prev})$ **then**
      $\Delta_{prev} \leftarrow \Delta_{current}$; $k \leftarrow 0$; {discard all previous evaluation points}
    else if $(\Delta_{current} = \Delta_{prev})$ **then**
      if (degree of the next-to-last element in C) $= 1$ **then**
        $k \leftarrow k + 1$; $G[k] \leftarrow$ next-to-last element of C; $\{G[k] \in K_p[y]\}$
        $C[k] \leftarrow \sigma$; $R[k] \leftarrow (-1)^{\deg_y(\hat{g})\cdot d_1} \cdot c_\sigma$; $\{C[k] \in \mathbb{F}_p,\ R[k] = \text{res}_y(\hat{g}, M_1) \in K_p\}$
      else
        if (no linear polynomial found for three σ's) **then**
          return $FAIL$; {$\text{res}_y(g, M_1)$ highly likely not square-free}
        end if
      end if
    else
      skip; {$\Delta_{current} \prec \Delta_{prev}$, so σ is unlucky}
    end if
  else if Algorithm **sr_prs** returned $FAIL$ for three σ's **then**
    return $FAIL$;
  end if
  $\sigma \leftarrow$ random integer in $\mathbb{F}_p \backslash S$; $S \leftarrow S \cup \{\sigma\}$;
end while
Interpolate $N_p(x) \in K_p[x]$ from $[C, R]$; Interpolate $f_{n-1}(x,y) \in K_p[x,y]$ from $[C, G]$;
$G(x,y) \leftarrow \text{lcoeff}_y(f_{n-1})^{-1} \cdot f_{n-1}(x,y)$; {make $f_{n-1}(x,y)$ monic in y}
if $G = FAIL$ **then return** $FAIL$; {division by zero divisor encountered} **end if**
return $N_p(x)$, $G(x,y)$;

Let us analyze the time complexity of Algorithm **res_modp2**, where $d_1 = \deg_y(M_1)$ and $d_2 = \deg_x(g)$. The only difference between Algorithm **res_modp** and Algorithm **res_modp2** is that the latter also outputs $G(x, y)$. Thus the only extra cost associated with Algorithm **res_modp2** is an additional interpolation of x in $f_{n-1}(x, y)$ from $f_{n-1}(\sigma_i, y) \in K_p[y]$, $i = 1, \ldots, \deg_x(N_p)$ (Line 26), which requires

$$\mathcal{O}((\deg(N_p) + 1)^2) \subseteq \mathcal{O}(d_1^2 d_2^2) \text{ arithmetic operations in } K_p.$$

This is an improvement from computing $\gcd(g(\overline{\gamma}, y), M_1(y)) \in K_p[\overline{\gamma}]$ using, for example, the Euclidean algorithm, which would cost $\mathcal{O}(d_1^4 d_2^2)$ arithmetic operations in K_p. The while-loop in Algorithm **res_modp2** may be executed more number of times than the while-loop in Algorithm **res_modp** because some evaluation points may be unlucky. However, the algorithm returns $FAIL$ if three unlucky evaluation points are encountered. Hence this extra cost is asymptotically negligible. Since we have shown that Algorithm **res_modp** requires $\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2)$ arithmetic operations in K_p (Section 7.1.5), Algorithm **res_modp2** requires

$$\mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) + \mathcal{O}(d_1^2 d_2^2) \subseteq \mathcal{O}(d_1^3 d_2 + d_1^2 d_2^2) \text{ arithmetic operations in } K_p,$$

which is equivalent to $\mathcal{O}(d_1^2 d_2^2)$ arithmetic operations in K_p if $d_1 \leq d_2$. Attempting to compute a primitive element of $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t] \cong \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t \rangle$ for $t > 2$ requires executing Algorithm **res_modp2** $(t-1)$ times. If $D = \prod_{i=1}^{t} \deg(M_i)$, then by the analysis done on page 69, we conclude that the total number of arithmetic operations required in $\mathbb{F}_p$ is

$$\begin{cases} \mathcal{O}(tD^2) & \text{if } d_i \leq d_j, 1 \leq i < j \leq t, \\ \mathcal{O}((\sum_{i=1}^{t-1} d_i) D^2) & \text{otherwise.} \end{cases}$$

Remark 7.22. *Theorem 7.14 and inspecting the structure of the Sylvester matrices* $\mathrm{Syl}_y(M_1, g)_{i,1}$ *for* $0 \leq i \leq 1$ *imply that*

$$\deg_x(\mathrm{sRes}_1(M_1, g)) \leq (\deg_x(g) - 1)(\deg_x(M_1)).$$

That is, we only need $(\deg_x(g)-1)(\deg_x(M_1))+1$ *evaluation points to interpolate the* gcd, *rather than* $(\deg_x(g))(\deg_x(M_1))+1$ *evaluation points that we used in Algorithm*

res_modp2*. However, reducing the number of evaluation points does not change the overall complexity of Algorithm* ***res_modp2****. We also remark that we used* $(\deg_x(g)-1)(\deg_x(M_1))+1$ *evaluation points in our Maple implementation of this algorithm.*

7.3 Complete algorithm and complexity

Algorithm **AlgFFTMult** uses the resultant-based primitive element finding algorithm (Algorithm **prim_elt_multi**), and FFT polynomial multiplication (Algorithm **FFTMult**) to compute the product of polynomials in $K_p[x] = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$.

Let $\deg(f) = m \leq \deg(g) = n$ and $D = \prod_{i=1}^{t} \deg_{u_i}(M_i)$. We analyze the complexity of Algorithm **AlgFFTMult**.

- Line 2: The cost in $\mathbb{F}_p$ of executing Algorithm **prim_elt_multi** is $\mathcal{O}\left(D^3 t\left(\frac{1}{d_1} + \frac{1}{d_{t-2}d_{t-1}}\right)\right)$ (see Chapter 6).

- Line 5: To build the change-of-basis matrix C, we need to compute $\overline{\alpha}_1^{j_1} \cdots \overline{\alpha}_t^{j_t}$ for $1 \leq j_i \leq \deg_{u_i}(M_i)$ and $1 \leq i \leq t$. Because the normal representations of each $\overline{\alpha}_i \in \mathbb{F}_p[\overline{\gamma}] \cong \mathbb{F}_p[z]/\langle M_\gamma(z)\rangle$ are known from executing Algorithm **prim_elt_multi**, *all* the ${\overline{\alpha}_1}^{j_1} \cdots {\overline{\alpha}_t}^{j_t} \in \mathbb{F}_p[\overline{\gamma}]$ can be computed in at most D multiplication in $\mathbb{F}_p[\overline{\gamma}]$. Converting each of these elements from its `recden` representation to CDR does not require any arithmetic operations, as we merely need to "pad" the vector with zeros. Since the cost of one arithmetic operation in $\mathbb{F}_p[\overline{\gamma}]$ is equivalent to $\mathcal{O}(D^2)$ arithmetic operations in $\mathbb{F}_p$, the cost of computing C in $\mathbb{F}_p$ is $\mathcal{O}(D \cdot D^2) \subseteq \mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_p$.

- Lines 7-10 and 11-14: Each of these for-loops requires multiplying the matrix $C \in \mathbb{F}_p^D \times \mathbb{F}_p^D$ by a column vector ($\in \mathbb{F}_p^D$) each time in the loop. Thus both for-loops require $\mathcal{O}(D^2 n + D^2 m) \subseteq \mathcal{O}(D^2 n)$ multiplications in $\mathbb{F}_p$.

- Line 16: Algorithm **FFTMult** requires $\mathcal{O}(Dn \log n + D^2 n)$ arithmetic operations in $\mathbb{F}_p$ (see Chapter 3).

- Lines 18 - 21: For each i, we must substitute $\overline{\gamma}^i$ for $(c_1\overline{\alpha}_1 + \cdots + \overline{\alpha}_t)^i \in K_p$. There are at most $m+n+1$ terms that contain $\overline{\gamma}^i$ in h. After each substitution,

one may need to perform a scalar multiplication with $(c_1\overline{\alpha}_1 + \cdots + \overline{\alpha}_t)^i \in K_p$, which has at most D terms. Hence Line 19 requires $\mathcal{O}((m+n+1)D) \subseteq \mathcal{O}(Dn)$ multiplications in $\mathbb{F}_p$ per loop, and Line 20 requires of one multiplication in K_p, or $\mathcal{O}(D^2)$ arithmetic operations in $\mathbb{F}_p$ per loop. In total, completing this for-loop requires $\mathcal{O}((Dn + D^2)D) = \mathcal{O}(D^2n + D^3)$ arithmetic operations in $\mathbb{F}_p$.

Thus the total number of arithmetic operations in $\mathbb{F}_p$ required by Algorithm **AlgFFTMult** is

$$\mathcal{O}\left(D^3 t\left(\tfrac{1}{d_1} + \tfrac{1}{d_{t-2}d_{t-1}}\right) + D^3 + D^2n + (D\,n\log n + D^2n) + (D^2n + D^3)\right)$$
$$\subseteq \mathcal{O}(D^3 + D^2n + Dn\log n).$$

Algorithm 7.4 : **AlgFFTMult**$(f(x), g(x), K_p)$

Input: $f(x), g(x) \in K_p[x] = \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x] \cong \mathbb{F}_p[u_1, \ldots, u_t]/\langle M_1, \ldots, M_t\rangle[x]$ where p is a good Fourier prime greater than $(\prod_{i=1}^{t} \deg_{u_i}(M_i))^2$.

Output: $h = f \cdot g \in K_p[x]$, or $FAIL$.

1: $m \leftarrow \deg_x(f)$; $n \leftarrow \deg_x(g)$;
2: $\overline{\gamma}$, $M_\gamma(z)$, $A \leftarrow$ **prim_elt_multi**$(M_1, \ldots, M_t)$;
3: $\{\overline{\gamma} = c_1\overline{\alpha}_1 + c_2\overline{\alpha}_2 + \cdots + c_{t-1}\overline{\alpha}_{t-1} + \overline{\alpha}_t$;
$M_\gamma(z)$: minimal polynomial for γ (modulo p) where γ is a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_t)$;
$A : [\overline{\alpha}_1(\overline{\gamma}), \overline{\alpha}_2(\overline{\gamma}), \ldots, \overline{\alpha}_t(\overline{\gamma})]\}$
4: **if** Algorithm **prim_elt_multi** outputs $FAIL$ **then return** $FAIL$; **end if**
5: Compute change-of-basis matrix C from
$B_\alpha = \{u_1^{d_1} \cdots u_t^{d_t}, 0 \le d_i \le \deg(M_i) - 1, 1 \le i \le t\}$ to $B_\gamma = \{z^i, 0 \le i \le D-1\}$ where $D = \prod_{i=1}^{t} \deg_{u_i}(M_i)$;
6: $F, G \leftarrow$ empty lists of length $(m+1)$ and $(n+1)$ respectively;
7: **for** $i = 0 \ldots m$ **do**
8: $R \leftarrow$ CDR of coeff(f, x^i) as a column vector;
9: $F[i] \leftarrow$ `recden` rep. of $C \cdot r$; $\{F[i] = \text{coeff}(f, x^i) \in \mathbb{F}_p[z]/\langle M_\gamma(z)\rangle\}$
10: **end for**
11: **for** $i = 0 \ldots n$ **do**
12: $R \leftarrow$ CDR of coeff(g, x^i) as a column vector;
13: $G[i] \leftarrow$ `recden` rep. of $C \cdot r$; $\{G[i] = \text{coeff}(g, x^i) \in \mathbb{F}_p[z]/\langle M_\gamma(z)\rangle\}$
14: **end for**
15: $\{F$ and G are `recden` reps of $f, g \in \mathbb{F}_p(\overline{\gamma}) \cong \mathbb{F}_p[z]/\langle M_\gamma(z)\rangle[x]$ respectively$\}$
16: $h \leftarrow$ **FFTMult**$(F, G, \mathbb{F}_p(\overline{\gamma}))$; $\{\ h = f \cdot g \in \mathbb{Z}_p[z]/\langle M_\gamma(z)\rangle[x]\}$
17: $k \leftarrow c_1\overline{\alpha}_1 + c_2\overline{\alpha}_2 + \cdots + c_{t-1}\overline{\alpha}_{t-1} + \overline{\alpha}_t\ (= \overline{\gamma})$;
18: **for** $i = 1 \ldots D-1$ **do**
19: substitute $\overline{\gamma}^{\,i}$ for k in h;
20: $k \leftarrow k \cdot \overline{\gamma} \in \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$;
21: **end for**
22: **return** $h(x) \in \mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$;

Chapter 8

Benchmarks and conclusion

We have implemented the resultant-based polynomial multiplication algorithm as described by Algorithm **AlgFFTMult**. This routine and all of its sub-routines were implemented in Maple, except for the resultant algorithm (Algorithm **res_modp2**) for the 2-step extensions case, which we implemented in C by modifying the existing Maple's kernel resultant routine to return not only the resultant but also the next-to-last element in the subresultant PRS and the degree sequence of the PRS. We gratefully acknowledge Roman Pearce for helping us with this implementation.

We compare the performance (in seconds) of our algorithm and the naïve multiplication algorithm over 3- and 4-step extensions of varying degrees. All timings were obtained using Maple 15 on a 64-bit Intel Core i7 2.67 GHz running Linux. The polynomials to be multiplied (f and g) and all the minimal polynomials were generated at random. In each table, the column labelled 'n' indicates the degrees of polynomials being multiplied, which were chosen to be the median of two consecutive powers of two, so that FFT multiplication requires the most number of "unnecessary" calculations. The 'd_i's in the second column denote the degree (in u_i) of $m_i(u_i) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[u_i]$, the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$. Under the multi-columns '$\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$' and '$\mathbb{F}_p[\overline{\gamma}]$', we list the timings for computing the product of f and g over the respective rings using naïve multiplication (column labelled 'mult') and FFT (column labelled 'FFT'). Under the multi-column '$\mathbb{F}_p[\overline{\gamma}]$', we further have the following columns:

- 'prim': timings for *computing* the mapping from $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$ to $\mathbb{F}_p[\overline{\gamma}]$, which includes computing $\overline{\gamma} = \Phi_p(\gamma)$, the minimal polynomial of γ (mod p), and the normal representations (mod p) of each α_i (line 2 in Algorithm **AlgFFTMult**).

- 'COB': timings for computing the change-of-basis matrix (Line 5 in Algorithm **AlgFFTMult**).

- 'ϕ': timings for *applying* the mapping from $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t]$ to $\mathbb{F}_p[\overline{\gamma}]$ by representing f and g as polynomials over $\mathbb{F}_p[\overline{\gamma}]$ using a series of matrix-vector multiplications (Lines 7-14 in Algorithm **AlgFFTMult**).

- 'ϕ^{-1}': timings of mapping the product $f \cdot g$ back to $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_t][x]$ (Lines 18-21 in Algorithm **AlgFFTMult**).

- 'total': total timings for computing the polynomial product using the resultant approach (i.e. the sum of the columns 'prim', 'COB', 'ϕ', 'FFT' and 'ϕ^{-1}').

Table 8.1 lists the timings for polynomial multiplication over a field with a tower of three extensions. It shows that it is more efficient to use FFT and to perform multiplications in $\mathbb{F}_p[\overline{\gamma}]$ in all cases. In particular, when $n = 384$ and $d_i = 10$ (last row in the table), using FFT multiplication over $\mathbb{F}_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3]$ provides a gain by a factor of $\frac{19584}{409.01} \approx 50$ over naïve multiplication performed over $\mathbb{F}_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3]$. Finding the product using FFT multiplication over $\mathbb{F}_p[\overline{\gamma}]$ shows a greater gain by a factor of $\frac{19584}{55.561} \approx 350$ over naïve multiplication performed over $\mathbb{F}_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3]$. The table also shows that when the degree of the field D is held constant and the degree of the polynomials n becomes large, the speed gain obtained by converting to a simple extension becomes more pronounced. For example, when $n = 24$ and $d_i = 10$ (fifth row from the bottom of the table), $\frac{11.896+1.568+0.738+9.102}{24.161} \approx 96\%$ of the total time is taken in mapping and applying the conversions (i.e., sum of columns 'prim', 'COB', 'ϕ', and 'ϕ^{-1}'). However, when $n = 384$ and $d_i = 10$ (last row in the table), it comprises only $\frac{12.944+1.651+6.168+13.630}{55.561} \approx 62\%$ of the total time.

		$\mathbb{F}_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3]$		$\mathbb{F}_p[\overline{\gamma}]$						
n	d_i	mult	FFT	mult	prim	COB	ϕ	FFT	ϕ^{-1}	total
24	2	0.450	0.116	0.003	0.018	0.000	0.004	0.018	0.003	0.043
48	2	1.456	0.292	0.012	0.020	0.000	0.008	0.039	0.005	0.072
96	2	5.736	0.634	0.052	0.018	0.000	0.015	0.087	0.008	0.128
192	2	22.722	1.343	0.203	0.018	0.000	0.030	0.193	0.016	0.258
384	2	90.865	2.960	0.799	0.018	0.000	0.060	0.446	0.040	0.564
24	4	3.010	0.772	0.017	0.160	0.003	0.016	0.036	0.013	0.228
48	4	11.146	1.609	0.067	0.159	0.004	0.029	0.080	0.020	0.293
96	4	43.094	3.445	0.248	0.163	0.003	0.056	0.185	0.037	0.445
192	4	171.06	7.310	0.984	0.157	0.010	0.118	0.421	0.076	0.782
384	4	692.71	15.941	4.030	0.171	0.003	0.238	0.994	0.146	1.552
24	6	11.894	2.859	0.252	0.846	0.037	0.063	0.109	0.132	1.187
48	6	43.224	5.868	0.934	0.823	0.036	0.112	0.249	0.165	1.385
96	6	167.42	12.286	3.719	0.830	0.036	0.204	0.566	0.229	1.864
192	6	669.19	26.325	15.293	0.853	0.037	0.388	1.291	0.355	2.923
384	6	2755.4	58.134	64.923	0.890	0.050	0.757	2.991	0.625	5.313
24	8	34.920	8.306	1.134	3.519	0.237	0.241	0.324	1.323	5.644
48	8	124.36	16.652	4.155	3.467	0.236	0.375	0.719	1.427	6.225
96	8	484.05	35.078	16.728	3.456	0.235	0.753	1.596	1.634	7.675
192	8	1959.7	76.118	70.455	3.596	0.236	1.259	3.638	2.024	10.753
384	8	8280.6	173.57	312.56	3.857	0.237	2.377	8.579	2.893	17.943
24	10	84.206	20.112	3.757	11.896	1.568	0.738	0.857	9.102	24.161
48	10	287.65	38.758	13.412	11.527	1.455	1.196	1.823	9.434	25.434
96	10	1121.9	81.923	54.589	11.761	1.479	1.914	3.985	10.041	29.180
192	10	4576.6	179.50	234.14	12.164	1.529	3.411	9.005	11.240	37.348
384	10	19584	409.01	1053.3	12.944	1.651	6.168	21.167	13.630	55.561

Table 8.1: Polynomial multiplication over a field given as 3-step extensions using naïve multiplication and FFT, with and without converting to a simple extension.

On the other hand, when the degree of the polynomials n stays constant and $D = d_1 d_2 d_3$ becomes larger, mapping and applying the conversions become the bottleneck

of the algorithm, which is as expected. For example, when $n = 384$ and $d_i = 2$ (fifth row in the table) so that $D = \prod_{i=1}^{3} d_i = (2)(2)(2) = 8$, the time spent on mapping and applying the conversions is $\frac{0.018+0.000+0.060+0.040}{0.564} \approx 21\%$, whereas when $n = 384$ and $d_i = 10$ (last row in the table) so that $D = \prod_{i=1}^{3} d_i = (10)(10)(10) = 1000$, the fraction of the time spent on mapping and applying the conversions as we have computed above is higher at $\approx 62\%$. However, the gain in the speed of FFT multiplication over a simple extension offsets this bottleneck.

		$\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_4]$		$\mathbb{F}_p[\overline{\gamma}]$						
n	d_i	mult	FFT	mult	prim	COB	ϕ	FFT	ϕ^{-1}	total
24	2	2.446	0.492	0.005	0.064	0.001	0.009	0.019	0.005	0.098
48	2	9.113	1.035	0.016	0.093	0.000	0.018	0.043	0.010	0.164
96	2	35.313	2.202	0.061	0.093	0.000	0.035	0.096	0.018	0.243
192	2	139.92	4.672	0.298	0.064	0.000	0.069	0.243	0.036	0.412
384	2	563.64	10.166	1.076	0.065	0.001	0.172	0.544	0.071	0.852
24	4	70.976	13.540	0.321	2.051	0.065	0.126	0.139	0.208	2.590
48	4	262.37	27.549	1.149	2.050	0.094	0.197	0.308	0.255	2.903
96	4	1029.0	57.716	4.668	2.054	0.065	0.407	0.670	0.372	3.568
192	4	4188.6	124.38	19.580	2.172	0.107	0.623	1.622	0.533	5.058
384	4	17657	281.46	85.102	2.339	0.066	1.379	3.805	1.011	8.600
24	6	631.72	132.23	7.785	36.767	4.881	1.286	1.531	19.541	64.006
48	6	2236.1	256.89	29.268	36.174	4.839	1.925	3.143	20.111	66.191
96	6	8773.4	537.74	120.10	36.936	5.026	3.392	6.922	21.202	73.479
192	6	36390	1218.3	493.65	38.313	5.210	5.758	14.989	23.045	87.315
384	6	154166	2664.4	2318.2	41.428	5.738	10.618	35.881	26.733	120.40

Table 8.2: Polynomial multiplication over fields given as a 4-step extension using naïve multiplication and FFT, with and without converting to a simple extension.

Table 8.2 lists the timings for polynomial multiplication over a field with a tower of four extensions. It indicates, there is even more significant speed-up in using our algorithm over fields given as four-step extensions. However, because the degree of the field $D = \prod_{i=1}^{4} d_i$ is large in all cases, the bottleneck is computing and applying

the conversions, as expected. For example, for the case $n = 384$ and $d_i = 6$ (last row in the table) so that $D = 6^4 = 1296$, approximately $\frac{41.428+5.738+10.618+26.733}{120.40} \approx 70\%$ of the time is spent on computing and applying the conversions (sum of columns 'prim', 'COB', 'ϕ', and 'ϕ^{-1}'). Nevertheless, as with the 3-step extension case, performing the FFT multiplication over a simple extension provides a significant speed advantage over performing the multiplication over multiple extension, so this offsets the high conversion costs. For example, in the case $n = 384$ and $d_i = 6$ (last row in table), the speed-up in performing the FFT multiplication over the simple extension versus multiple extensions is approximately a factor of $\frac{2664.4}{35.881} \approx 74$.

Table 8.3 lists timings for computing the product of two polynomials of degree 96 each (that is, $n = 96$) over a field given as a three-step extension of degree 256, where the degrees of each extension vary. As mentioned in Chapter 4, the table shows that when d_1 is small relative to D, naïve multiplication is very slow; in particular, when $[d_1, d_2, d_3] = [2, 2, 64]$ (that is, when multiplying over $\mathbb{F}_p[u_1, u_2, u_3]\langle u_1^2 + \ldots, u_2^2 + \ldots, u_3^{64} + \cdots\rangle$), the speed gain in computing over a simple extension is a factor of $\frac{4485.2}{3.462} \approx 1300$. However, when d_1 is large, naïve multiplication is relatively efficient; for example, when $[d_1, d_2, d_3] = [64, 2, 2]$ (that is, when multiplying over $\mathbb{F}_p[u_1, u_2, u_3]\langle u_1^{64} + \ldots, u_2^2 + \ldots, u_3^2 + \cdots\rangle$), the gain is only a factor of $\frac{15.870}{2.133} \approx 7.5$. In fact, in this case it is more efficient to compute the product using FFT multiplication over *multiple* extensions than using the FFT multiplication after converting to a simple extension, by a factor of $\frac{2.133}{1.393} \approx 1.5$. There are two underlying reasons for such drastically different timings observed in Table 8.3 when d_1 is small and d_1 is large. The first reason arises from the structure of the `recden` representation. When $[d_1, d_2, d_3] = [2, 2, 64]$ the `recden` data structure is illustrated in Figure 8.1.

			$\mathbb{F}_p[\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3]$		$\mathbb{F}_p[\overline{\gamma}]$						
d_1	d_2	d_3	mult	FFT	mult	prim	COB	ϕ	FFT	ϕ^{-1}	total
2	2	64	4485.2	329.69	5.203	1.637	0.035	0.604	0.714	0.472	3.462
2	4	32	2347.9	172.81	4.995	3.379	0.034	0.487	0.722	0.448	5.069
2	8	16	1529.5	114.54	4.842	4.942	0.036	0.676	0.708	0.429	6.792
2	16	8	1246.8	90.899	4.768	8.338	0.046	0.661	0.700	0.421	10.166
2	32	4	1284.7	83.377	5.013	19.276	0.066	0.358	0.742	0.409	20.852
2	64	2	1569.8	78.457	5.016	55.292	0.108	0.353	0.744	0.408	56.906
4	2	32	1086.5	81.055	4.723	0.761	0.034	0.391	0.692	0.367	2.245
4	4	16	616.42	44.876	4.758	1.210	0.054	0.327	0.676	0.364	2.631
4	8	8	429.45	31.060	4.710	1.936	0.046	0.287	0.686	0.344	3.299
4	16	4	401.33	26.480	4.955	3.721	0.066	0.331	0.729	0.347	5.194
4	32	2	439.63	23.065	4.969	9.262	0.108	0.308	0.717	0.346	10.741
8	2	16	302.85	22.256	4.637	0.596	0.051	0.256	0.678	0.320	1.901
8	4	8	178.78	13.064	4.594	0.790	0.046	0.249	0.674	0.314	2.072
8	8	4	142.42	9.921	4.806	1.136	0.090	0.230	0.710	0.306	2.473
8	16	2	140.06	8.222	4.801	2.111	0.107	0.249	0.713	0.305	3.485
16	2	8	89.932	6.910	4.362	0.570	0.046	0.232	0.647	0.293	1.787
16	4	4	62.592	4.770	4.746	0.769	0.066	0.184	0.695	0.293	2.007
16	8	2	49.766	3.462	4.357	0.861	0.127	0.184	0.643	0.286	2.101
32	2	4	34.439	2.806	4.714	0.734	0.077	0.181	0.691	0.285	1.969
32	4	2	24.753	1.934	4.411	0.767	0.109	0.172	0.642	0.279	1.968
64	2	2	15.870	1.393	4.564	0.902	0.135	0.157	0.664	0.275	2.133

Table 8.3: Multiplication of two polynomials of degree 96 each over a field given as a 3-step extension of degree $D = \prod_{i=1}^{3} d_i = 256$ is held constant.

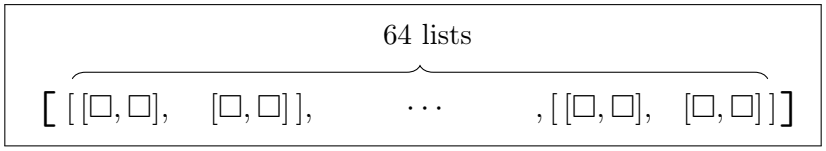

Figure 8.1: The `recden` representation of an element in $\mathbb{F}_p[u_1, u_2, u_3]/\langle M_1, M_2, M_3\rangle$ where $\deg_{u_1}(M_1) = 2$, $\deg_{u_2}(M_2) = 2$, and $\deg_{u_3}(M_3) = 64$.

On the other hand, when $[d_1, d_2, d_3] = [64, 2, 2]$, the `recden` data structure is illustrated by Figure 8.2.

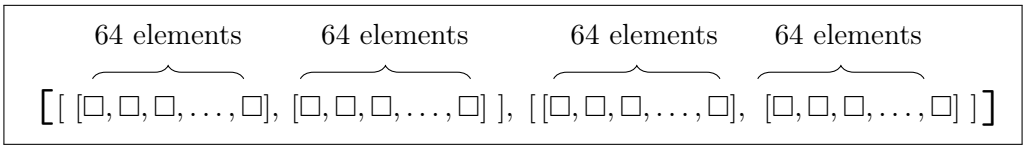

Figure 8.2: The `recden` representation of an element in $\mathbb{F}_p[u_1, u_2, u_3]/\langle M_1, M_2, M_3\rangle$ where $\deg_{u_1}(M_1) = 64$, $\deg_{u_2}(M_2) = 2$, and $\deg_{u_3}(M_3) = 2$.

Because of the nature of the data structure, there relatively large overhead costs associated with doing polynomial arithmetic (for example, allocating storage, copying objects, et cetera) when d_1 is small compared with the case when d_1 is large.

The second reason depends on the number of polynomial multiplications and divisions performed in each case. When $[d_1, d_2, d_3] = [2, 2, 64]$, multiplying a coefficient of f by a coefficient of g requires a multiplication of two polynomials of three variables, followed by a division by a degree 64 polynomial (M_3), then a division by M_2 up to 64 times (one for each coefficient of M_3), and subsequently by M_1 another 64 times (once for each coefficient of M_3). On the other hand, when $[d_1, d_2, d_3] = [64, 2, 2]$, after the multiplication one must divide by a degree two polynomial M_3, followed by a degree two polynomial M_2 at most two times, then finally a division by a degree 64 polynomial M_1 at most twice. Clearly there are many more multiplications one must perform in the case when d_1 is small compared to other d_i's.

Observe further from the table that our algorithm is quite inefficient at computing the mapping from multiple extensions to a simple extension when $[d_1, d_2, d_3] =$

$[2, 64, 2]$ (sixth row and column 'prim_elt' of the table). We now explain why this is so. Recall that to convert from a fields given as a 3-step extension to a field with a simple extension, we first collapse the last two extensions by finding a $\overline{\gamma}_1$ satisfying $\mathbb{F}_p[\overline{\alpha}_1][\overline{\alpha}_2, \overline{\alpha}_3] = \mathbb{F}_p[\overline{\alpha}_1, \overline{\gamma}_1]$, after which we collapse $\mathbb{F}_p[\overline{\alpha}_1, \overline{\gamma}_1]$ to $\mathbb{F}_p[\overline{\gamma}_2]$. Since the cost of mapping from three extensions to two is $\mathcal{O}(d_2^3 d_3 + d_2^2 d_3^2)$ (Chapter 6), the cost of mapping is high if $d_2 \gg d_3$, as is the case when $[d_1, d_2, d_3] = [2, 64, 2]$ and $[d_1, d_2, d_3] = [2, 32, 4]$. The next step of mapping from two extensions to one costs $\mathcal{O}(d_1^3(d_2 d_3) + d_1^2(d_2 d_3)^2)$ arithmetic operations. Hence if $d_1 \gg d_2 d_3$, the cost of the algorithm must become expensive. However, this is not reflected in Table 8.3 (for example, when $[d_1, d_2, d_3] = [64, 2, 2]$ the algorithm is relatively efficient) because the resultant algorithm used in the two extensions case (used in finding $\overline{\gamma}_2$) is implemented in C, which is much more efficient than the Maple resultant algorithm used in finding $\overline{\gamma}_1$.

Our algorithm is most efficient when $d_2 \approx d_3$ and $d_1 \approx d_2 d_3$, which is expected, since the cost of mapping to a simple extension for these cases is $\mathcal{O}(d_1^2 d_2^2)$.

8.1 Conclusion and future work

In this thesis we presented two efficient algorithms for computing the product of univariate polynomials of degrees at most n over a multiple extension field $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ of degree D. The main ideas used were: mapping the rational coefficients to integers modulo a prime p, collapsing the multiple extension to a simple extension, computing the product over the simple extension using the FFT for bivariate polynomials, and using efficient methods of computing the resultant and the gcd. We have shown that the time complexity of applying the resultant & gcd method and the FFT to compute the product is $\mathcal{O}(D^3 + D^2 n + Dn \log n)$ arithmetic operations in $\mathbb{F}_p$ for each prime p. This is an improvement (for the case $D \leq n^2$) over the naïve multiplication method,which requires $\mathcal{O}(D^2 n^2)$ arithmetic operations in $\mathbb{Q}$.

To speed up the computation, we mapped $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_t)$ to $K_p = \mathbb{F}_p[u_1, \ldots, u_t]/\langle \Phi_p(m_1), \ldots, \Phi_p(m_t) \rangle$, where p was chosen to be a good Fourier prime between 2^{30} and $2^{31.5}$. However, because K_p may not be a ring depending on p, many complications arose. In particular, there were problems arising from: attempting to

divide by zero divisors, $\Phi_p(m_i)$ not being square-free over $\mathbb{F}_p[\overline{\alpha}_1, \ldots, \overline{\alpha}_{i-1}]$, p being a fail prime, and encountering unlucky evaluation points while computing the gcd. However, we showed that these complications rarely occur; in fact they are polynomially bounded by D. Thus, provided p is sufficiently large the algorithm will not fail with high probability.

To collapse the multiple extension to a simple extension, we discussed two methods: a linear algebra approach (which was previously known) and a resultants & gcd approach (whose modular method is new). Of the two approaches, computing the minimal polynomial (modulo p) for a primitive element of the multiple extension field using the resultant method is more efficient ($\mathcal{O}((\sum_{i=1}^{t} d_i)D^2)$ arithmetic operations in $\mathbb{F}_p$ are required in executing Algorithm **res_modp2**) than the linear algebra method ($\mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_p$ required in computing the change-of-basis matrix). We do note however that we can lower the complexity of making the substitutions by viewing the series of matrix-vector multiplication problem as a single matrix-matrix multiplication problem, then employing the Schönhage-Strassen method (Strassen [16]) for efficiency.

However, computing the normal representations (modulo p) and making the substitutions using the normal representations to the polynomials causes a bottleneck of the resultant method ($\mathcal{O}(D^3)$ arithmetic operations in $\mathbb{F}_p$), making it only as efficient as the linear algebra method. In the future, we will investigate a faster method for this step in order to prevent the bottleneck.

Appendix A

Proof of Lemma 4.11

Proof. For $i > 1$, we have

$$\begin{aligned}
&(cx+y)^i \mod \langle m_1, m_2\rangle \\
&\quad = ((cx+y)\cdot r_{i-1}) \mod \langle m_1, m_2\rangle \\
&\quad = (cx\cdot r_{i-1} \mod \langle m_1\rangle) + ((y\cdot r_{i-1} \mod \langle m_2\rangle) \mod \langle m_1\rangle) \qquad (1)\\
&\quad = (c\cdot A_{(d_1,d_2-1)}\cdot x^{d_1}y^{d_2-1} + \cdots + A_{(0,0)}) \mod \langle m_1\rangle + \\
&\qquad ((B_{(d_1-1,d_2)}\cdot x^{d_1-1}y^{d_2} + \cdots + B_{(0,0)}) \mod \langle m_2\rangle) \mod \langle m_1\rangle
\end{aligned}$$

where $A_{(i,j)}, B_{(k,r)} \in \mathbb{Z}$. Let

$$g_1 = c\cdot A_{(d_1,d_2-1)}\cdot x^{d_1}y^{d_2-1} + \cdots + A_{(0,0)}, \text{ and } g_2 = B_{(d_1-1,d_2)}\cdot y^{d_2}x^{d_1-1} + \cdots + B_{(0,0)}.$$

Only the first term of g_1 requires division by m_1. If $\tilde{c} = \max\{|c|, 1\}$, then by Lemma 4.10,

$$\|\hat{g}_1\| := \|g_1 \mod \langle m_1\rangle\| \le (1+\|m_1\|)^{d_1-d_1+1}\cdot\tilde{c}\cdot\|r_{i-1}\| \le M\cdot\tilde{c}\cdot\|r_{i-1}\|.$$

Similarly,

$$\|\hat{g}_2\| := \|g_2 \mod \langle m_2\rangle\| \le (1+\|m_2\|)^{d_2-d_2+1}\cdot\|r_{i-1}\| \le M\cdot\|r_{i-1}\|.$$

At this point, $\deg_x(\hat{g}_2) \le (d_1-1)+(d_1-1) = 2d_1-2$. Thus after division by m_1, we get

$$\begin{aligned}
\|\hat{g}_2 \mod \langle m_1\rangle\| &\le (1+\|m_1\|)^{(2d_1-2)-\deg(m_1)+1}\cdot(M\cdot\|r_{i-1}\|)\\
&\le M^{(2d_1-2)-d_1+1}\cdot M\cdot\|r_{i-1}\|\\
&= M^{d_1}\cdot\|r_{i-1}\|.
\end{aligned}$$

Hence,

$$\begin{aligned}\|r_i\| = \|(cx+y)^i \mod \langle m_1, m_2\rangle\| &= \|\hat{g}_1 + \hat{g}_2\| \\ &\le \|\hat{g}_1\| + \|\hat{g}_2\| \\ &\le M \cdot \tilde{c} \cdot \|r_{i-1}\| + M^{d_1} \cdot \|r_{i-1}\| \\ &= \|r_{i-1}\| \cdot (\tilde{c}M + M^{d_1}) \\ &= \|r_{i-1}\| \cdot (\tilde{c}+1) \cdot M^{d_1} \quad \text{for } 2 \le i \le D-1.\end{aligned} \tag{2}$$

Since $\|r_1\| = \|cx+y\| = \tilde{c} = \max\{|c|, 1\}$, we have

$$\|(cx+y)^i\| = \|r_i\| = \|r_{i-1}\|(\tilde{c}+1)M^{d_1} = \tilde{c}\left[(\tilde{c}+1)M^{d_1}\right]^{i-1} \text{ for } i = 2, \cdots, D.$$

Let C be the change-of-basis matrix whose i-th column consists of the coefficients of $(cx+y)^{i-1}$ for $i = 1, \cdots, D$. If a_{ij} denotes the entry in the i-th row and j-th column of C, then by Hadamard's inequality we have

$$\begin{aligned}|\det(C)| = |\det(C)^T| &\le \prod_{i=1}^{D} \sqrt{\sum_{j=1}^{D} a_{ij}^2} \\ &\le \prod_{i=0}^{D-1} \sqrt{D(\|r_i\|^2)} \\ &= (\sqrt{D}\|r_0\|) \cdot \prod_{i=1}^{D-1} (\sqrt{D}\|r_i\|) \\ &\le \sqrt{D} \cdot \prod_{i=1}^{D-1} \left(\sqrt{D}\left[\tilde{c}\left[(\tilde{c}+1)M^{d_1}\right]^{i-1}\right]\right) \\ &= D^{D/2}\tilde{c}^D \left(\left[(\tilde{c}+1)M^{d_1}\right]^{\frac{(D-1)(D-2)}{2}}\right).\end{aligned}$$

Taking the log of both sides with base B, we obtain

$$\begin{aligned}\log_B(|\det(C)|) &\le \log_B(D^{D/2}\tilde{c}^D) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left(\log_B(\tilde{c}+1) + \log_B(M^{d_1})\right) \\ &< D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left(\log_B(\tilde{c}+1) + d_1\log_B(M)\right).\end{aligned}$$

That is, the number of digits of $\det(C)$ in base B is at most

$$\begin{aligned}&D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left(\log_B(\tilde{c}+1) + d_1\log_B(M)\right) \\ &\in \mathcal{O}\left(D^2(\log_B(\tilde{c}) + d_1\log_B(M))\right).\end{aligned}$$

□

Proof of Lemma 4.12

Proof. Let $\|r_i\| \equiv (c_1x_1 + \cdots c_{t-1}x_{t-1} + x_t)^i \mod \langle m_1, \cdots, m_t\rangle$ for $i = 0, \cdots, D-1$. Using a similar reasoning as in the proof of Lemma 4.11, one can show that, for $i \geq 0$,

$$\|(c_1x_1 + c_2x_2 + \cdots + c_{t-1}x_{t-1} + x_t)^i\| = \|r_i\| \leq \tilde{c}\left((\tilde{s}+1)M^{\tilde{d}}\right)^{i-1}.$$

Hence

$$\begin{aligned} \|\det(C)\| &\leq \prod_{i=0}^{D-1}\sqrt{D(\|c_1x_1 + \cdots + c_{t-1}x^{t-1} + x_t)^i\|)^2} \\ &\leq \sqrt{D}\cdot\prod_{i=1}^{D-1}\left(\sqrt{D}\tilde{c}\left[(\tilde{s}+1)M^{\tilde{d}}\right]^{i-1}\right) \\ &= D^{D/2}\tilde{c}^D\left[(\tilde{s}+1)M^{\tilde{d}}\right]^{(D-1)(D-2)/2} \\ &< (D\tilde{c})^D\left[(\tilde{s}+1)M^{\tilde{d}}\right]^{(D-1)(D-2)/2}, \end{aligned} \tag{3}$$

so

$$\begin{aligned} \log_B(\|\det(C)\|) &< \log_B\left((D\tilde{c})^D\left[(\tilde{s}+1)M^{\tilde{d}}\right]^{(D-1)(D-2)/2}\right) \\ &= D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\log_B((\tilde{s}+1)M^{\tilde{d}}) \\ &= D\log_B(D\tilde{c}) + \left(\tfrac{(D-1)(D-2)}{2}\right)\left[\log_B(\tilde{s}+1) + \tilde{d}\log_B(M)\right] \\ &\in \mathcal{O}\left(D^2\left[\log_B(\tilde{s}) + \tilde{d}\log_B(M)\right]\right). \end{aligned} \tag{4}$$

□

Bibliography

[1] Alaca, S., Williams, K. S. *Introductory Algebraic Number Theory*. Cambridge University Press, 1st edition, 2004.

[2] Basu, S., Pollack, R., Roy, M. *Algorithms in Real Algebraic Geometry*. Springer, 2nd edition, 2006, p. 316-317.

[3] Bronstein, M. *Symbolic Integration I: Transcendental Functions*. Springer, 2nd edition, 2005.

[4] Brown, W. S. *The subresultant PRS algorithm*. ACM Trans. Math. Software, 4 (1978), no. 3, p. 237-249.

[5] Brown, W. S., Traub, J.F. *On Euclid's algorithm and the theory of subresultants*. J. Assoc. Comput. Mach. 18 (1971), p. 505-514.

[6] Chen, L., Monagan, M. *Algorithms for solving linear systems over cyclotomic fields*. J. Symbolic Comput. 45 (2010), p. 902-917.

[7] Collins, G.E. *The calculation of multivariate polynomial resultants*. J. Assoc. Comput. Mach. 18 (1971), p. 515-532.

[8] Collins, G.E. *Subresultants and Reduced Polynomial Remainder Sequences*. J. ACM, **14**(1) (1967) p. 128-142.

[9] Gaal, L. *Classical Galois Theory, with Examples*. American Mathematical Society, 5th edition, 1998.

[10] Garling, D.J.H. *Inequalities: a journey into linear analysis*. Cambridge University Press, 1st edition, 2007.

[11] Geddes, K.O., Czapor, S.R., & Labahn, G. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1st edition, 1992.

[12] Lang, S. *Algebraic Number Theory.* Springer-Verlag, 2nd edition, 1994.

[13] McCarthy, P. *Algebraic Extensions of Fields.* Chelsea Publishing Company, 2nd edition, 1976.

[14] Mishra, B. *Algorithmic Algebra.* Texts and Monographs in Computer Science, Springer-Verlag, New York, 1993.

[15] Monagan, M. *Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction.* Proceedings of ISSAC '04, ACM Press, pp. 243-249, 2004.

[16] Strassen, V. *Gaussian elimination is not optimal.* Numerische Mathematik, Springer Berlin, 1969.

[17] Trager, B. *Algebraic Factoring and Rational Function Integration.* Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, 1976.

[18] von zur Gathen, J., Gerhard, J. *Modern Computer Algebra.* Cambridge University Press, 2nd edition, 2003.

[19] Wang, P. S., Guy, M. J. T., Davenport, J. H. *p-adic reconstruction of rational numbers.* ACM SIGSAM Bulletin, **16**, No 2, 1982.

[20] Wang, P. S. *A p-adic Algorithm for Univariate Partial Fractions.* Proceedings of SYMSAC '81, ACM Press, p.212-217, 1981.

[21] Winkler, F. *Polynomial Algorithms in Computer Algebra.* Springer-Verlag/Wien, 1st edition, 1996.

[22] Zippel, R. *Effective polynomial computation.* Kluwer Academic Publishers Group, 1st edition, 1993.