Gröbner Bases over Extension Fields of $\mathbb{Q}$

Dereje Kifle Boku

Vom Fachbereich Mathematik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte **Dissertation**

Gutachter:

1. Prof. Dr. Wolfram Decker

2. Prof. Dr. Anne Frühbis-Krüger

Datum der Disputation: 5. August 2016

D 386

Gedruckt mit Unterstützung des Deutschen Akademischen Austauschdienstes

To my
respected mother ***Birke***,
loving wife ***Tigist***
&
dear daughter ***Eliana***

Abstract

Gröbner bases are one of the most powerful tools in computer algebra and commutative algebra, with applications in algebraic geometry and singularity theory. From the theoretical point of view, these bases can be computed over any field using Buchberger's algorithm. In practice, however, the computational efficiency depends on the arithmetic of the coefficient field.

In this thesis, we consider Gröbner bases computations over two types of coefficient fields. First, consider a simple extension $K = \mathbb{Q}(\alpha)$ of $\mathbb{Q}$, where α is an algebraic number, and let $f \in \mathbb{Q}[t]$ be the minimal polynomial of α. Second, let K' be the algebraic function field over $\mathbb{Q}$ with transcendental parameters $t_1, \ldots, t_m$, that is, $K' = \mathbb{Q}(t_1, \ldots, t_m)$. In particular, we present efficient algorithms for computing Gröbner bases over K and K'. Moreover, we present an efficient method for computing syzygy modules over K.

To compute Gröbner bases over K, starting from the ideas of Noro [35], we proceed by joining f to the ideal to be considered, adding t as an extra variable. But instead of avoiding superfluous S-pair reductions by inverting algebraic numbers, we achieve the same goal by applying modular methods as in [2, 4, 27], that is, by inferring information in characteristic zero from information in characteristic $p > 0$. For suitable primes p, the minimal polynomial f is reducible over $\mathbb{F}_p$. This allows us to apply modular methods once again, on a second level, with respect to the modular factors of f. The algorithm thus resembles a divide and conquer strategy and is in particular easily parallelizable. Moreover, using a similar approach, we present an algorithm for computing syzygy modules over K.

On the other hand, to compute Gröbner bases over K', our new algorithm first specializes the parameters $t_1, \ldots, t_m$ to reduce the problem from $K'[x_1, \ldots, x_n]$ to $\mathbb{Q}[x_1, \ldots, x_n]$. The algorithm then computes a set of Gröbner bases of specialized ideals. From this set of Gröbner bases with coefficients in $\mathbb{Q}$, it obtains a Gröbner basis of the input ideal using sparse multivariate rational interpolation.

At current state, these algorithms are probabilistic in the sense that, as for other modular Gröbner basis computations, an effective final verification test is only known for homogeneous ideals or for local monomial orderings. The presented timings show that for most examples, our algorithms, which have been implemented in SINGULAR [17], are considerably faster than other known methods.

Zusammenfassung

Gröbnerbasen sind eines der leistungsfähigsten Werkzeuge in der Computeralgebra und der kommutativen Algebra, mit Anwendungen in algebraischer Geometrie und Singularitätentheorie. Aus theoretischer Sicht können solche Basen mit dem Algorithmus von Buchberger über beliebigen Körpern berechnet werden. In der Praxis hängt die Effizienz solcher Berechnungen aber von der Arithmetik im Koeffizientenkörper ab.

In dieser Dissertation betrachten wir Berechnungen von Gröbnerbasen über zwei verschiedenen Arten von Koeffizientenkörpern. Im ersten Fall betrachten wir einfache Erweiterungen $K = \mathbb{Q}(\alpha)$, wobei α eine algebraische Zahl sei und wir das Minimalpolynom von α mit $f \in \mathbb{Q}[t]$ bezeichnen. Im zweiten Fall sei K' der algebraische Funktionenkörper über $\mathbb{Q}$ mit transzendenten Parametern $t_1, \ldots, t_m$, also $K' = \mathbb{Q}(t_1, \ldots, t_m)$. Insbesondere stellen wir effiziente Algorithmen zur Berechnung von Gröbnerbasen über K und K' vor. Außerdem stellen wir eine effiziente Methode zur Berechnung von Syzygienmoduln über K vor.

Um Gröbnerbasen über K zu berechnen, fügen wir, ausgehend von den Ideen von Noro [35], das Polynom f zum betrachteten Ideal hinzu, mit t als zusätzlicher Variable. Anstatt aber überflüssige Reduktionen von S-Paaren durch das Invertieren algebraischer Zahlen zu vermeiden, erreichen wir dasselbe Ziel durch Anwendung modularer Methoden wie in [2, 4, 27], also durch Rückschlüsse von Daten in Charakteristik $p > 0$ auf Daten in Charakteristik Null. Für geeignete Primzahlen p ist das Minimalpolynom f über $\mathbb{F}_p$ reduzibel. Dies erlaubt uns,auf einer zweiten Ebeneein weiteres Mal modulare Methoden anzuwenden, bezüglich der modularen Faktoren von f. Der Algorithmus gleicht daher einer Teile-und-herrsche-Strategie und ist insbesondere leicht zu parallelisieren. Außerdem stellen wir einen Algorithmus zur Berechnung von Syzygienmoduln über K vor, der einen ähnlichen Ansatz verfolgt.

Um hingegen Gröbnerbasen über K' auszurechnen, werden in dem neuen Algorithmus zunächst verschiedene konkrete Werte für die Parameter $t_1, \ldots, t_m$ eingesetzt, um das Problem von $K'[x_1, \ldots, x_n]$ auf $\mathbb{Q}[x_1, \ldots, x_n]$ zurückzuführen. Der Algorithmus berechnet dann eine Menge von Gröbnerbasen der so erhaltenen Ideale. Aus dieser Menge von Gröbnerbasen mit Koeffizienten in $\mathbb{Q}$ wird dann durch Sparse Multivariate Rational Interpolation eine Gröbnerbasis des Inputideals berechnet.

Dem gegenwärtigen Stand nach sind diese Algorithmen probabilistisch in dem Sinne, dass wie bei anderen modularen Gröbnerbasisberechnungen ein effektiver finaler Verifikationstest nur für homogene Ideale oder für lokale Monomordnungen bekannt ist. Die vorgelegten Laufzeitmessungen zeigen, dass die neuen, in Singular [17] implementierten Algorithmen bei den meisten Beispielen wesentlich schneller sind als andere bekannte Methoden.

Acknowledgements

I am pleased to acknowledge my deepest gratitude to my supervisor Prof. Dr. Wolfram Decker whose guidance, creativity and keen interest enabled me to complete this thesis. Thank you for offering me the opportunity to work in AGAG research group at the University of Kaiserslautern. He has always been a source of inspiration for me.

I would like to express my heartiest gratitude to my co-supervisor Prof. Dr. Claus Fieker for his valuable supervision, insightful comments and suggestions. It would not have been possible for me to complete this task without his extra efforts.

My special thanks and appreciation go to Dr. Andreas Steenpaß for his constant support and friendly advices during this research. I am also very grateful to him for his patience, availability and prompt reply to my questions.

Moreover, I express my gratitude to Dr. Berhanu Bekele for his permanent encouragement and invaluable support during my studies.

Further on, I wish to thank all the members of AGAG group for their strong academic atmosphere and friendly environment. Here, I should mention Prof. Dr. Gerhard Pfister, Dr. Janko Böhm, Dr. Hans Schönemann, Shrawan Tiwari and Adrian Popescu for their kind help during my studies. In this occasion, I also thank the secretary of our group Petra Bäsel, she has been always there to help me.

I would like to acknowledge Deutscher Akademischer Austausch Dienst (DAAD) and the Department of Mathematics University of Kaiserslautern for the financial support. Many thanks to Dr. Falk Triebsch and his secretary for administration support.

Last but not least, I would like to express my deepest gratitude to my friends and family, especially to my mother, wife and daughter for their undivided love, understanding and continued encouragement.

Contents

List of Figures

List of Algorithms

List of Tables

Introduction

Gröbner bases were introduced by Bruno Buchberger in 1965. In his thesis, Buchberger used Gröbner bases to give an algorithmic way of computing in affine rings, see [11]. In particular, he designed an algorithm which computes Gröbner bases. Historically, this algorithm is the first for computing such bases, and it has been implemented in most computer algebra systems, see, for example, SINGULAR [17]. The basic idea of the method is the transformation of a given set of polynomials F into a certain standard form G, which Buchberger called Gröbner basis [11]. For a detailed information about Buchberger and Gröbner bases, we refer the reader to [1, 13, 18, 25, 34, 42].

Gröbner bases have found extensive applications in many areas of mathematics including algebraic geometry [13, 25, 18], invariant theory [13, 18, 19], and coding theory [14], to mention a few. Nowadays, these bases are one of the most powerful tools in computer algebra and commutative algebra, with applications in algebraic geometry and singularity theory, see [1, 13, 14, 18, 21, 25]. For example, they are used for solving systems of polynomial equations, the implicitization problem, intersecting ideals, the ideal and radical membership problem, and many more.

Gröbner bases can be computed over any field using Buchberger's algorithm. However, since the complexity of this algorithm is extremely high, computing these bases is in general time consuming. Howbeit, applications of Gröbner bases increase the demand for computationally efficient algorithms. In fact, extensive efforts in improving the method for computing these bases have been made, see [20, 22, 23, 24]. On the other hand, when Gröbner bases are computed over a field of characteristic zero using Buchberger's algorithm, one often suffers from intermediate coefficient swell which usually slows the computations down. That is, the number and size of the intermediate polynomials during the computation can become much larger than in the final result. For example, in order to tackle the coefficient swell problem over $\mathbb{Q}$, modular techniques have been employed, see [2, 27, 40, 36]. As defined in [44], *modular techniques* implies using certain projections for improving the efficiency of the total computation, whereas by *modular computation*, we mean corresponding computations applied to projected images.

The aim of this thesis is to investigate efficient methods for computing Gröbner bases over algebraic number fields and algebraic function fields. Furthermore, we investigate an efficient method for computing syzygy modules over algebraic number fields as one of the applications of Gröbner bases. In particular, we discuss the following problems:

(1) Given an ideal $I \subseteq K[x_1, \ldots, x_n]$ where $K = \mathbb{Q}(\alpha)$ is an algebraic number field, what is an efficient way to compute a Gröbner basis of I?

(2) If M a submodule of the free $K[x_1, \ldots, x_n]$-module

$$K[x_1, \ldots, x_n]^r = \bigoplus_{i=1}^{r} K[x_1, \ldots, x_n]e_i$$

where $K = \mathbb{Q}(\alpha)$ is an algebraic number field and $e_1, \ldots, e_r$ form the canonical basis of $K[x_1, \ldots, x_n]^r$, what is an efficient way to compute the syzygy module of M?

(3) What is an efficient way to compute a Gröbner basis of an ideal $I \subseteq K[x_1, \ldots, x_n]$ where K is a field of rational functions in the symbolic parameters $t_1, \ldots, t_m$ with coefficients in $\mathbb{Q}$?

Computing Gröbner bases or syzygy modules over the field $K = \mathbb{Q}(\alpha)$ or over $K = \mathbb{Q}(t_1, \ldots, t_m)$ are usually slow due to the arithmetic in K. To improve the efficiency of the computations, we study extended modular techniques over K, that is, we extend the modular algorithms described in [2, 27, 36]. An important question to address here is how to tackle the coefficient swell problem in K. The basic idea of the modular algorithms described in [2, 27, 36] is to perform the computational task over fields of prime characteristic for several primes and to lift the results back to the field of rational numbers via the Chinese remaindering algorithm and the Farey rational map.

Problem (1) above was studied by Noro [35], who designed a modified version of Buchberger's algorithm. In his algorithm, he computes the inverse of an algebraic number using modular techniques, which is in general computationally expensive. In this thesis, we study a new efficient method to solve this problem which uses

- modular methods over $\mathbb{Q}$ w.r.t. different prime numbers to avoid intermediate coefficient swell;

- factorization of the minimal polynomial of the algebraic number α in positive characteristics to considerably reduce the degree of the field extensions.

Furthermore, we extend the method adopted from Problem (1) to compute Gröbner bases of submodules of a free module with slight modifications. Based on this modified algorithm, we study Problem (2), where we also use results from [25].

In [10], Brickenstein introduced a variation of Buchberger's algorithm to investigate our third problem using so-called slim polynomials. The algorithm is designed to keep coefficients small and polynomials short in the intermediate computations. Although his method works well in many cases, it is however limited in terms of efficiency. The method we study in this thesis to address this challenge utilizes

- modular methods over $\mathbb{Q}$ as above w.r.t. different prime numbers;

- sparse interpolation of multivariate rational functions w.r.t. different evaluation points from $\mathbb{Q}^m$ to avoid the intermediate coefficient swell that occurs due to the arithmetic in K.

The basic idea of our new method for the third problem is as follows: First, we choose suitable evaluation points $b := (b_1, \ldots, b_m)$ for $(t_1, \ldots, t_m)$ from $\mathbb{Q}^m$ to map $I \subseteq K[x_1, \ldots, x_n]$ to $I_b \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ via the map sending t_i to $b_i \in \mathbb{Q}$. In the polynomial ring over $\mathbb{Q}$, we then compute Gröbner bases of I_b for different b's as many times as necessary. From this set of Gröbner bases with coefficients in $\mathbb{Q}$, we obtain a Gröbner basis corresponding to I using the sparse rational interpolation algorithm described in [16].

These new algorithms are implemented in SINGULAR [17], which is an open source general computer algebra system, and we demonstrate the efficiency of our methods via several examples. In addition, we present tables containing timings which compare our methods to other known methods.

This thesis is structured as follows: The first chapter introduces the most essential and important definitions, notation, results, and remarks regarding Gröbner bases and sparse interpolation of rational functions used in this thesis. As part of our investigation, we have implemented the algorithms in this chapter such as univariate polynomial interpolation, univariate rational function reconstruction, and sparse multivariate polynomial interpolation in SINGULAR [17]. Chapter 2 presents the aforementioned method that addresses our first problem. In this chapter, we explain how this method solves the coefficient swell problem which occurs due to the arithmetic in $K = \mathbb{Q}(\alpha)$. The key ingredient for this new method is factorizing the minimal polynomial of the algebraic number α modulo a suitable prime. In Chapter 3, an algorithm for computing syzygy modules which solves the second problem is presented. This algorithm uses the results from Chapter 2. Moreover, we discuss a further optimization of this algorithm.

The fourth chapter introduces the notion of sparse interpolation of multivariate rational functions. In this chapter, following [16], we discuss how to reconstruct a multivariate rational function in a black box from given numerical data. For this chapter, we use the last three sections from Chapter 1. As an application of the results from Chapter 4, we present, in Chapter 5, an efficient method that solves our third problem. Here we explain how the new method avoids the extreme growth of intermediate coefficients in $K = \mathbb{Q}(t_1, \ldots, t_m)$. We also apply further optimizations to our algorithm. Finally, in Chapter 6, we close our presentation by drawing conclusions and suggesting possible directions for future work.

A more detailed synopsis is given at the beginning of each chapter. In this thesis, we assume all rings to be commutative rings with identity. Chapter 2 is the result of a joint work with W. Decker, C. Fieker, and A. Steenpaß, see [8]. The benchmark problems which we use for the timings are listed in the appendix.

Chapter 1

Preliminaries

In this chapter we state the most essential and important definitions and results with respect to Gröbner bases [25], algebraic number fields [15], dense univariate rational interpolation [42], and sparse multivariate polynomial interpolation [3] (see also [31, 32]) that are used in this thesis. We restrict ourselves to a brief description, for example, we will not discuss the Berlekamp/Massey algorithm for computing minimal polynomials [3] (see also [31, algorithm in Section 2.1]). For a more detailed description, we refer to [1, 3, 13, 15, 21, 25, 31, 32, 34]. Throughout this chapter, let K be any field and let $X = \{x_1, \ldots, x_n\}$ be a set of variables.

In Section 1.1, we give a short but brief description of the theory of Gröbner bases. Since any method for computing Gröbner bases relies on the chosen monomial ordering, we also discuss the most important monomial orderings which are particularly relevant for this thesis. The notion of Gröbner bases for modules is presented in Section 1.2. In Section 1.3, we present some results on algebraic number fields. In Section 1.4, we discuss how univariate rational functions can be recovered from given numerical data. The notion of sparse interpolation of multivariate polynomials is discussed in Section 1.5. In particular, we discuss an algorithm by Ben-Or/Tiwari which recovers multivariate polynomials from given numerical data. In the last section, we consider the early termination version of the algorithms presented in Sections 1.4 and 1.5. As part of this thesis, all algorithms discussed in this chapter including the Berlekamp/Massey algorithm are implemented in the SINGULAR library `ffmodstd.lib` [6].

1.1 Rings, Ideals, and Gröbner Bases

Most of the definitions, theorems, and remarks which we give in this section can be found, for example, in any of the textbooks [1, 13, 25, 34]. We closely follow [25].

Definition 1.1.1. Let R be a ring.

1. (a) A *monomial* in n variables $X = \{x_1, \ldots, x_n\}$ is a power product

$$X^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}, \quad \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n.$$

(b) A non-zero constant multiple of a monomial is called a *term*.

(c) The *degree* $\deg X^\alpha$ of X^α is defined by

$$\deg X^\alpha = |\,\alpha\,| := \alpha_1 + \ldots + \alpha_n\,.$$

(d) The *set of monomials* in n variables is denoted by

$$\mathrm{Mon}(X) := \mathrm{Mon}_n := \{X^\alpha \mid \alpha \in \mathbb{N}^n\}\,.$$

Note that $\mathrm{Mon}(X)$ is a monoid under multiplication, with neutral element $1 = X^0$.

2. A *polynomial* f in X with coefficients in R is a finite sum of terms. We write f in the form

$$f = \sum_\alpha a_\alpha X^\alpha,\ a_\alpha \in R,$$

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \ldots, \alpha_n)$. For $\alpha \in \mathbb{N}^n$, the integer

$$\deg(f) := \begin{cases} \max\{|\,\alpha\,| \mid a_\alpha \neq 0\} & \text{if } f \neq 0 \\ -\infty & \text{otherwise} \end{cases}$$

is called the *degree* of f. If $a_\alpha \neq 0$, then we call $a_\alpha X^\alpha$ a *term* of f.

3. The *polynomial ring* in X over R, denoted by $R[X] = R[x_1, \ldots, x_n]$, is the set of all polynomials with coefficients in R together with the usual addition and multiplication:

$$\sum_\alpha a_\alpha X^\alpha + \sum_\alpha b_\alpha X^\alpha := \sum_\alpha (a_\alpha + b_\alpha) X^\alpha,$$

$$\left(\sum_\alpha a_\alpha X^\alpha\right) \cdot \left(\sum_\alpha b_\alpha X^\alpha\right) := \sum_\gamma \left(\sum_{\alpha+\beta=\gamma} a_\alpha b_\beta\right) X^\gamma.$$

The presentation of a polynomial as a finite sum of non-zero terms is unique up to the order of the summands, due to the commutativity of the addition. We make this order unique by choosing a total ordering on the set of monomials.

Definition 1.1.2. A *monomial ordering* is a total (or linear) ordering $>$ on the set of monomials $\mathrm{Mon}(X)$ in n variables satisfying

$$X^\alpha > X^\beta \Longrightarrow X^\gamma X^\alpha > X^\gamma X^\beta$$

for all $\alpha, \beta, \gamma \in \mathbb{N}^n$. Given a ring R, we then also say that $>$ is a monomial ordering on $R[X]$.

In the following, choosing a monomial ordering $>$ allows us to write a given polynomial in a uniquely ordered way.

Definition 1.1.3. Let $>$ be a fixed monomial ordering on $\mathrm{Mon}(X)$. Write $f \in R[X]\backslash\{0\}$ in a unique way as a finite sum of non-zero terms (sparse representation of f)

$$f = a_\alpha X^\alpha + a_\beta X^\beta + \ldots,$$

where $X^\alpha > X^\beta > \ldots$ and $a_\alpha, a_\beta, \ldots$ are in R. We define:

a) $\mathrm{lm}(f) = X^\alpha$, the *leading monomial* of f,

b) $\mathrm{lc}(f) = a_\alpha$, the *leading coefficient* of f,

c) $\mathrm{lt}(f) = a_\alpha X^\alpha$, the *leading term* or *head* of f,

d) $\mathrm{tail}(f) = f - \mathrm{lt}(f)$, the *tail* of f.

Monomial orderings are very important for computing Gröbner bases. They are classified as global, local and mixed.

Definition 1.1.4. Let $>$ be a monomial ordering on $\mathrm{Mon}(X)$.

a) $>$ is called a *global ordering* if $X^\alpha > 1$ for all $\alpha \neq (0, \ldots, 0)$.

b) $>$ is called a *local ordering* if $1 > X^\alpha$ for all $\alpha \neq (0, \ldots, 0)$.

c) $>$ is called a *mixed ordering* if it is neither global nor local.

In this thesis we only consider global orderings.

The most important monomial orderings which are particularly relevant for this thesis are introduced in the following definition.

Definition 1.1.5. The following are monomial orderings on $\mathrm{Mon}(X)$.

(i) *Lexicographical ordering* (denoted in SINGULAR by lp):

$$X^\alpha >_{\mathrm{lp}} X^\beta :\Longleftrightarrow \text{there exists } 1 \leq i \leq n : \;\; \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}, \;\; \alpha_i > \beta_i \,.$$

(ii) *Degree reverse lexicographical ordering* (denoted in SINGULAR by dp):

$$\begin{aligned} X^\alpha >_{\mathrm{dp}} X^\beta :\Longleftrightarrow & \deg X^\alpha > \deg X^\beta \\ & \text{or } (\deg X^\alpha = \deg X^\beta \,\text{and there exists } 1 \leq i \leq n : \\ & \alpha_n = \beta_n, \ldots, \alpha_{i+1} = \beta_{i+1}, \alpha_i < \beta_i) \,. \end{aligned}$$

(iii) *Product or Block ordering :* Let $Y = \{y_1, \ldots, y_m\}$ be a set of variables, and let $>_X$ and $>_Y$ be monomial orderings on $\mathrm{Mon}(X)$ and $\mathrm{Mon}(Y)$, respectively. Then the product ordering or block ordering $> = (>_X, >_Y)$ on $\mathrm{Mon}(X, Y)$ is defined by

$$X^{\alpha_X} Y^{\alpha_Y} > X^{\beta_X} Y^{\beta_Y} : \Longleftrightarrow X^{\alpha_X} >_X X^{\beta_X} \text{ or } (X^{\alpha_X} = X^{\beta_X} \text{ and } Y^{\alpha_Y} >_Y Y^{\beta_Y}),$$

for all $\alpha_X, \beta_X \in \mathbb{N}^n, \alpha_Y, \beta_Y \in \mathbb{N}^m$. The product ordering $>$ is global (resp. local) if both orderings $>_X$ and $>_Y$ are global (resp. local), otherwise it is a mixed ordering.

Note that $>_{\mathrm{lp}}$ and $>_{\mathrm{dp}}$ are global orderings.

Ideals are in the centre of commutative algebra and algebraic geometry. We introduce only basic notions which are relevant for this thesis. Let R be a ring.

Definition 1.1.6. A subset $I \subseteq R$ is called an *ideal* if it is an additive subgroup which is closed under scalar multiplication, that is,

$$\begin{aligned} I &\neq \emptyset, \\ f, g \in I &\Longrightarrow f + g \in I, \\ f \in I, r \in R &\Longrightarrow rf \in I. \end{aligned}$$

Definition 1.1.7. Let $I \subseteq R$ be an ideal.

a) A family $(f_\lambda)_{\lambda \in \Lambda}$ of elements $f_\lambda \in I$ is called a *system of generators* of I if every element $f \in I$ can be expressed as a finite linear combination $f = \sum_{\lambda \in \Lambda} r_\lambda f_\lambda$ for suitable $r_\lambda \in R$. We then write

$$I = \langle f_\lambda \mid \lambda \in \Lambda \rangle = \langle f_\lambda \mid \lambda \in \Lambda \rangle_R = \sum_\lambda f_\lambda R.$$

If $\Lambda = \{1, \ldots, k\}$ is finite, we write

$$I = \langle f_1, \ldots, f_k \rangle = \langle f_1, \ldots, f_k \rangle_R.$$

b) I is called *finitely generated* if it has a finite system of generators; it is *principal* if it can be generated by one element.

Definition 1.1.8. Let I be any ideal in the ring R. We define the *quotient ring or factor ring* R/I to be the set of co-sets $\{[r] = r + I \mid r \in R\}$ with addition and multiplication defined via representatives:

$$\begin{aligned} [r] + [s] &= [r + s], \\ [r] \cdot [s] &= [r \cdot s]. \end{aligned}$$

Definition 1.1.9. Let $f \in K[x]$ be a non-constant polynomial in one variable over the field K. The polynomial f is called *irreducible* if it is not the product of two non-constant polynomials of strictly smaller degree.

Theorem 1.1.10. *Let $f \in K[x]$ be a non-constant polynomial. Then the following are equivalent:*

1. *$K[x]/\langle f \rangle$ is a field.*
2. *$K[x]/\langle f \rangle$ is an integral domain.*
3. *f is irreducible.*

Proof. See [25, Exercise 1.1.5]. □

For a given set of polynomials we consider a special ideal, the so-called *leading ideal*, which is the central notion when defining a Gröbner basis of an ideal.

Definition 1.1.11. Let $>$ be a global monomial ordering on $\mathrm{Mon}(X)$ and $G \subseteq K[X]$ be a set of polynomials. Then we define the *set of leading monomials* of G, denoted by $\mathrm{Lm}(G)$, as

$$\mathrm{Lm}(G) = \{\mathrm{lm}(g) \mid g \in G \setminus \{0\}\},$$

and the *leading ideal* of G, denoted by $\mathrm{L}(G)$, as

$$\mathrm{L}(G) = \langle \mathrm{Lm}(G) \rangle \subseteq K[X].$$

In the following we introduce the notion of *Gröbner bases* of an ideal $I \subseteq K[X]$ as a finite set of polynomials of I such that their leading monomials generate the leading ideal $\mathrm{L}(I)$.

Definition 1.1.12. Let $>$ be a global monomial ordering on $\mathrm{Mon}(X)$ and $I \subseteq K[X]$ be an ideal.

(a) A finite set $G \subseteq K[X]$ of polynomials is called a *Gröbner basis* of the ideal I if

$$G \subseteq I \text{ and } \mathrm{L}(G) = \mathrm{L}(I).$$

(b) By saying that G is a Gröbner basis, we mean that G is Gröbner basis of the ideal $\langle G \rangle$ it generates.

(c) A Gröbner basis $G \subseteq K[X]$ is called *reduced* if

(1) $0 \notin G$,

(2) $\mathrm{lc}(g) = 1$ for all $g \in G$ and

(3) for each $g \in G$, $\mathrm{lm}(g)$ does not divide any monomial of any element of $G \setminus \{g\}$.

The following basic proposition deals with existence and uniqueness of Gröbner bases.

Proposition 1.1.13. *Let $>$ be a global monomial ordering on $\mathrm{Mon}(X)$, and let $I \subseteq K[X]$ be a non-zero ideal. Then the following hold:*

(a) *There exists a Gröbner basis of I.*

(b) *If $G \subseteq K[X]$ is any Gröbner basis of I, then G generates the ideal I in $K[X]$.*

(c) *There exists a uniquely determined reduced Gröbner basis of I.*

Remark 1.1.14. A reduced Gröbner basis can always be computed from any given Gröbner basis (in a finite number of steps).

The definition of *normal form* is essential for the computation of Gröbner bases.

Definition 1.1.15. Let $\mathcal{G}$ denote the set of all finite ordered sets $G \subseteq K[X]$. A map

$$\begin{aligned} \mathrm{NF} \ : K[X] \times \mathcal{G} &\longrightarrow K[X] \\ (f, G) &\longmapsto \mathrm{NF}(f, G) \end{aligned}$$

is called a *normal form* on $K[X]$ if, for all $f \in K[X]$ and all $G \in \mathcal{G}$, the following conditions hold:

(1) $\mathrm{NF}(0, G) = 0$.

(2) $\mathrm{NF}(f, G) \neq 0$ implies $\mathrm{lm}(\mathrm{NF}(f, G)) \notin \mathrm{L}(G)$.

(3) Either $f - \mathrm{NF}(f, G) = 0$, or there exists a representation

$$f - \mathrm{NF}(f, G) = \sum_{g \in G} c_g g, \ c_g \in K[X]$$

such that

$$\mathrm{lm}\left(f - \mathrm{NF}(f, G)\right) \geq \max\{\mathrm{lm}(c_g g) \mid g \in G, c_g g \neq 0\}.$$

Such a representation is called a *standard representation* of $f - \mathrm{NF}(f, G)$ w.r.t. G.

Moreover, NF is called a *reduced normal form* if $\mathrm{lc}(\mathrm{NF}(f, G)) = 1$ and no monomial of $\mathrm{tail}(\mathrm{NF}(f, G))$ is contained in $\mathrm{L}(G)$ for all $f \in K[X]$ and all $G \in \mathcal{G}$.

Remark 1.1.16. For a global ordering, there exists a normal form (see [25, Algorithm 1.6.10]), respectively reduced normal form (see [25, Algorithm 1.6.11]).

The notion of *S-polynomials* and *Buchberger's criterion* are essential for the computation of Gröbner bases.

Definition 1.1.17. Let $f, g \in K[X]$ be non-zero polynomials with $\mathrm{lm}(f) = X^\alpha$ and $\mathrm{lm}(g) = X^\beta$. Set $\gamma := \mathrm{lcm}(\alpha, \beta) = (\max(\alpha_1, \beta_1), \ldots, \max(\alpha_n, \beta_n)) \in \mathbb{N}^n$, then X^γ is the *least common multiple* of X^α and X^β. We define the *S-polynomial* of f and g by

$$\mathrm{spoly}(f, g) := X^{\gamma-\alpha} \cdot \frac{f}{\mathrm{lc}(f)} - X^{\gamma-\beta} \cdot \frac{g}{\mathrm{lc}(g)}.$$

Theorem 1.1.18 (*Buchberger criterion*)**.** *Let $>$ be a global monomial ordering on $\mathrm{Mon}(X)$, let $I \subseteq K[X]$ be an ideal, and let G be a finite subset of I. Moreover, let $\mathrm{NF}(-, G)$ be a normal form on $K[X]$ with respect to G. Then the following are equivalent:*

a) *G is a Gröbner basis of I.*

b) *$\mathrm{NF}(f, G) = 0$ for all $f \in I$.*

c) *$I = \langle G \rangle$ and $\mathrm{NF}(\mathrm{spoly}(g, g'), G) = 0$ for all $g, g' \in G$.*

From this theorem, we obtain an algorithm due to Buchberger for computing Gröbner bases. This algorithm is called Buchberger algorithm [11], see also [1, Algorithm 1.7.1]. Note that the procedure `std` is implemented in SINGULAR [17] since 1990, and that it computes a Gröbner basis of the input ideal for any monomial ordering.

1.2 Modules and Gröbner Bases

The theory of Gröbner bases for ideals carries over to modules almost without any changes. In this section, we formulate, closely following [25], the relevant definitions and remarks. For a more detailed description, we refer to [25].

Definition 1.2.1. Let A be a ring. An *A-module* $(M, +, \cdot)$ is a set M with maps $+ : M \times M \longrightarrow M$ and $\cdot : A \times M \longrightarrow M$ such that

(1) $(M, +)$ is an abelian group,

(2) the scalar multiplication $\cdot$ is distributive over the addition $+$, that is, $(a+b) \cdot m = a \cdot m + b \cdot m$ and $a \cdot (m+n) = a \cdot m + a \cdot n$ for all $a, b \in A$ and $m, n \in M$, and

(3) for all $a, b \in A$ and $m \in M$, $(a \cdot b) \cdot m = a \cdot (b \cdot m)$ and $1 \cdot m = m$.

Example 1.2.2.

(a) Every abelian group $(G, +)$ is a $\mathbb{Z}$-module with scalar multiplication

$$\begin{aligned} \cdot : \mathbb{Z} \times G &\longrightarrow G, \\ (n, x) &\longmapsto n \cdot x, \end{aligned}$$

where $n \cdot x$ is defined by

$$n \cdot x := \begin{cases} \underbrace{x + \ldots + x}_{n \text{ times}} & \text{if } n > 0 \\ 0 & \text{if } n = 0 \\ \underbrace{-x + \ldots + -x}_{n \text{ times}} & \text{if } n < 0. \end{cases}$$

(b) Let A be a ring. Consider the Cartesian product

$$A^r = \left\{ \begin{bmatrix} a_1 \\ \vdots \\ a_r \end{bmatrix} \;\middle|\; a_i \in A,\ i = 1, \ldots, r \right\}.$$

Then A^r is an A-module (with the component-wise addition and scalar multiplication).

In this section, in the interest of saving space, we usually write elements of A^r as row vectors enclosed in parentheses, that is,

$$(a_1, \ldots, a_r) \text{ instead of } \begin{bmatrix} a_1 \\ \vdots \\ a_r \end{bmatrix}.$$

Definition 1.2.3. Let A be a ring, and let M, N be A-modules. A map $\varphi : M \longrightarrow N$ is called an *A-module homomorphism* (or *A-linear*) if, for all $a \in A$ and $m, n \in M$,

$$\varphi(m + n) = \varphi(m) + \varphi(n), \text{ and } \varphi(am) = a\varphi(m).$$

Definition 1.2.4. Let M be an A-module. A non-empty subset N of M is called a *submodule* of M if, for all $m, n \in N$ and $a \in A$, it holds

$$m + n \in N, \text{ and } a \cdot m \in N.$$

Remark and Example 1.2.5 ([25, Lemma 2.1.12])**.** Let $\varphi : M \to N$ be an A-module homomorphism. The kernel of φ, $\mathrm{Ker}(\varphi) = \{m \in M \mid \varphi(m) = 0\}$, is a submodule of M.

Definition 1.2.6. Let M be an A-module.

(a) M is called *finitely generated* if there is a surjective A-module homomorphism

$$\varphi : A^r \to M.$$

In this case, the images $m_i = \varphi(e_i) \in M$ of the canonical basis vectors e_i, $i = 1, \ldots, r$, are called *generators*. Since φ is surjective, every element of M can be written as an A-linear combination of $m_1, \ldots, m_r$, that is, for all $m \in M$ there exist $a_1, \ldots, a_r \in A$ with $m = a_1 m_1 + \ldots + a_r m_r$. We then write $M = \langle m_1, \ldots, m_r \rangle$. Moreover, we call M a *cyclic* module if it is generated by one element.

(b) M is called *free* of rank r, if there is an isomorphism $\varphi : A^r \to M$, that is, if $A^r \cong M$. In this case, a representation $m = a_1 m_1 + \ldots + a_r m_r$ as above is unique, and we call $m_1, \ldots, m_r$ a basis of M.

Let K be a field and consider the polynomial ring $A = K[X] = K[x_1, \ldots, x_n]$. To study Gröbner bases for modules, we need to extend the notion of monomial orderings to the free module $A^r = \bigoplus_{i=1}^r Ae_i$.

Definition 1.2.7. An element of the form $X^\alpha e_i = (0, \ldots, X^\alpha, \ldots, 0) \in A^r$ is called a *monomial* (involving the component i), and a non-zero constant multiple of a monomial is called a *term*.

Definition 1.2.8. A *module monomial ordering*, or simply a *module ordering*, on A^r is a total ordering $\succ$ on the set of monomials in A^r such that if $X^\alpha e_i$ and $X^\beta e_j$ are monomials in A^r, and X^γ is a monomial in A, then

$$X^\alpha e_i \succ X^\beta e_j \Rightarrow X^\gamma X^\alpha e_i \succ X^\gamma X^\beta e_j.$$

Remark 1.2.9. In this thesis, we always suppose

$$X^\alpha e_i \succ X^\beta e_i \Longleftrightarrow X^\alpha e_j \succ X^\beta e_j \text{ for all } i, j.$$

In this case, $\succ$ induces a unique monomial ordering on A, and we say that $\succ$ is *global*, *local*, *mixed* if the induced ordering on A is global, local, mixed, respectively.

If we are given a monomial ordering on A, there are two canonical ways of obtaining a module ordering on A^r.

Definition 1.2.10. Let $>$ be a monomial ordering on A. In a canonical way we obtain the following module orderings on A^r:

(1) Priority to the monomials, that is, term over position (TOP):

$$X^\alpha e_i >_{\mathrm{TOP}} X^\beta e_j :\Longleftrightarrow \begin{cases} X^\alpha > X^\beta \\ \text{or} \\ X^\alpha = X^\beta \text{ and } i < j, \end{cases}$$

for all monomials $X^\alpha e_i$ and $X^\beta e_j$ of A^r. In SINGULAR, $>_{\mathrm{TOP}}$ is denoted by $(>, \mathrm{c})$.

(2) Priority to the components, that is, position over term (POT):

$$X^\alpha e_i >_{\mathrm{POT}} X^\beta e_j :\Longleftrightarrow \begin{cases} i < j \\ \text{or} \\ i = j \text{ and } X^\alpha > X^\beta, \end{cases}$$

for all monomials $X^\alpha e_i$ and $X^\beta e_j$ of A^r. In SINGULAR, $>_{\mathrm{POT}}$ is denoted by $(\mathrm{c}, >)$.

Definition 1.2.11. With respect to a given module ordering $\succ$ on A^r, any element $f \in A^r \setminus \{0\}$ can be written uniquely as

$$f = c_\alpha X^\alpha e_i + f'$$

with $c_\alpha \in K \setminus \{0\}$ and $X^\alpha e_i \succ X^{\alpha'} e_j$ for any non-zero term $c' X^{\alpha'} e_j$ of f'. Then, as in Definition 1.1.3, we define:

a) $\mathrm{lm}(f) = X^\alpha e_i$, the *leading monomial* of f,

b) $\mathrm{lc}(f) = c_\alpha$, the *leading coefficient* of f,

c) $\mathrm{lt}(f) = c_\alpha X^\alpha e_i$, the *leading term* or *head* of f.

Definition 1.2.12. Let $\succ$ be a global module ordering on A^r and let $G \subseteq A^r$ be a set of vectors. Then we define

(a) the *set of leading monomials* of G, denoted by $\mathrm{Lm}(G)$, as

$$\mathrm{Lm}(G) = \{\mathrm{lm}(g) \mid g \in G \setminus \{0\}\}, \text{ and}$$

(b) the *leading module* of G, denoted by $\mathrm{L}(G)$, as

$$\mathrm{L}(G) = \langle \mathrm{lm}(g) \mid g \in G \setminus \{0\} \rangle \subseteq A^r,$$

the monomial submodule generated by the leading monomials.

The notion of *Gröbner bases* of a submodule M of A^r as a finite set of vectors of M such that their leading monomials generate the leading module $\mathrm{L}(M)$ is defined as follows:

Definition 1.2.13. Let $M \subseteq A^r$ be a submodule and let $\succ$ be a global module ordering on A^r.

(a) A finite set $G \subseteq M$ is called a *Gröbner basis* of M w.r.t. $\succ$ if

$$\mathrm{L}(G) = \mathrm{L}(M).$$

(b) A Gröbner basis $G \subseteq A^r$ of M is called *reduced* if

(1) $0 \notin G$,

(2) $\mathrm{lc}(g) = 1$ for all $g \in G$, and

(3) for each $g \in G$, $\mathrm{lm}(g)$ does not divide any monomial of any element of $G \setminus \{g\}$.

As for ideals, the definition of *normal forms* is essential for the computation of Gröbner bases.

Definition 1.2.14. Let $\mathcal{G}$ denote the set of all finite ordered sets $G \subseteq A^r$. A map

$$\begin{aligned} \mathrm{NF}(-, G) : A^r \times \mathcal{G} &\longrightarrow A^r \\ (f, G) &\longmapsto \mathrm{NF}(f, G) \end{aligned}$$

is called a *normal form* on A^r if, for all $f \in A^r$ and all $G \in \mathcal{G}$, the following conditions hold:

(1) $\mathrm{NF}(0, G) = 0$.

(2) $\mathrm{NF}(f, G) \neq 0$ implies $\mathrm{lm}(\mathrm{NF}(f, G)) \notin \mathrm{L}(G)$.

(3) Either $f - \mathrm{NF}(f, G) = 0$, or there exists a representation

$$f - \mathrm{NF}(f, G) = \sum_{g \in G} c_g g, \ c_g \in A$$

such that

$$\mathrm{lm}\left(f - \mathrm{NF}(f, G)\right) \geq \max\{\mathrm{lm}(c_g g) \mid g \in G, c_g g \neq 0\}.$$

Such a representation is called a *standard representation* of $f - \mathrm{NF}(f, G)$ w.r.t. G.

Moreover, NF is called a *reduced normal form* if $\mathrm{lc}(\mathrm{NF}(f, G)) = 1$ and no monomial of $\mathrm{NF}(f, G) - \mathrm{lm}\left(\mathrm{NF}(f, G)\right)$ is contained in $\mathrm{L}(G)$ for all $f \in A^r$ and all $G \in \mathcal{G}$.

Remark 1.2.15 ([25, Remark 2.3.4])**.** For global orderings, a reduced normal form exists.

The notion of *S-vectors* for modules is defined as follows:

Definition 1.2.16. Consider $f, g \in A^r \setminus \{0\}$ with $\mathrm{lm}(f) = X^\alpha e_i$ and $\mathrm{lm}(g) = X^\beta e_j$. Set $\gamma := \mathrm{lcm}(\alpha, \beta) = (\max(\alpha_1, \beta_1), \ldots, \max(\alpha_n, \beta_n)) \in \mathbb{N}^n$, then X^γ is the *least common multiple* of X^α and X^β. We define the *S-vector* of f and g by

$$\text{S-vector}(f, g) := \begin{cases} X^{\gamma-\alpha} \cdot \dfrac{f}{\mathrm{lc}(f)} - X^{\gamma-\beta} \cdot \dfrac{g}{\mathrm{lc}(g)} & \text{if } i = j, \\ 0 & \text{if } i \neq j\ . \end{cases}$$

Using this definition, both the Buchberger criterion and Buchberger's algorithm can be extended from the case of ideals to submodules of A^r.

1.3 Algebraic Number Fields

In this section we study algebraic number fields. An algebraic number field is one of the coefficient fields over which we compute Gröbner bases (see Chapter 2). This section provides only basic notions and results that are relevant for this thesis. There are a couple of introductory textbooks about algebraic number fields. We refer to [15, 28], but we essentially follow [15].

We begin by introducing the notion of *integral* elements of ring extensions.

Definition 1.3.1. Let $R \subseteq S$ be a ring extension. An element $s \in S$ is said to be *integral* over R if it satisfies a polynomial equation

$$x^n + r_{n-1}x^{n-1} + \ldots + r_1 x + r_0 = 0$$

where $r_0, r_1, \ldots, r_{n-1} \in R$.

Note that every element $r \in R$ is integral over R as it is a root of the polynomial $x - r \in R[x]$.

Definition 1.3.2. A complex number which is integral over $\mathbb{Z}$ is called an *algebraic integer*.

Example 1.3.3.

a) $\sqrt{5}$ is an algebraic integer as it satisfies the equation $x^2 - 5 = 0$.

b) $\alpha = \dfrac{1}{\sqrt{2}}$ is not an algebraic integer. This is because the polynomial $f(x) = x^2 - 1/2$ is not in $\mathbb{Z}[x]$, though $f(\alpha) = 0$, and there is no monic polynomial in $\mathbb{Z}[x]$ with root α.

Definition 1.3.4. Let $R \subseteq S$ be a ring extension. Suppose that R is a field and $s \in S$ is integral over R; then s is said to be *algebraic* over R.

Definition 1.3.5. A complex number that is algebraic over $\mathbb{Q}$ is called an *algebraic number*.

Example 1.3.6. In Example 1.3.3, α is an algebraic number as it satisfies the equation $x^2 - \frac{1}{2} = 0$.

Theorem 1.3.7 ([15, Theorem 4.2.6])**.** *Every algebraic number is of the form a/b, where a is an algebraic integer and b is a non-zero ordinary integer.*

Example 1.3.8. The algebraic number α in Example 1.3.3 can be written as $\frac{\beta}{b}$ with $\beta = \sqrt{2}$ and $b = 2$. Clearly β is an algebraic integer.

Let K be a subfield of the field $\mathbb{C}$ of complex numbers. Let $\alpha \in \mathbb{C}$ be algebraic over K. We now study the notion of the *minimal polynomial* of α over K. As α is algebraic over K, there exists a non-zero polynomial $g(x) \in K[x]$ such that $g(\alpha) = 0$. We let $I_K(\alpha)$ denote the set of all polynomials in $K[x]$ having α as a root, that is,

$$I_K(\alpha) = \{f(x) \in K[x] \mid f(\alpha) = 0\}. \tag{1.1}$$

Clearly the set $I_K(\alpha)$ contains the zero polynomial. It is easy to check that $I_K(\alpha)$ is an ideal of $K[x]$. Moreover, $I_K(\alpha) \neq 0$ as $g(x) \in I_K(\alpha)$. As K is a field, we know that $K[x]$ is a Euclidean domain and thus a principal ideal domain. Hence there exists $p(x) \in K[x]$ such that

$$I_K(\alpha) = \langle p(x) \rangle. \tag{1.2}$$

Suppose $q(x) \in K[x]$ is another polynomial that generates $I_K(\alpha)$, that is,

$$I_K(\alpha) = \langle q(x) \rangle.$$

Then

$$\langle p(x) \rangle = \langle q(x) \rangle.$$

So, there exists a polynomial $u(x) \in K[x]$ such that

$$q(x) = u(x)p(x),$$

where $u(x)$ is a unit in $K[x]$. But $K[x]^* = K^*$, which implies $u(x) \in K^*$. This shows that we may assume the polynomial $p(x)$ to be monic, in which case $p(x)$ is uniquely determined by (1.2).

Definition 1.3.9. Let K be a subfield of $\mathbb{C}$. Let $\alpha \in \mathbb{C}$ be algebraic over K. Then the unique monic polynomial $p(x) \in K[x]$ such that

$$I_K(\alpha) = \langle p(x) \rangle$$

is called the *minimal polynomial* of α over K and is denoted by $\mathrm{irr}_K(\alpha)$.

Definition 1.3.10. Let K be a subfield of $\mathbb{C}$. Let $\alpha \in \mathbb{C}$ be algebraic over K. Then the *degree* of α over K, written $\deg_K(\alpha)$, is defined by

$$\deg_K(\alpha) = \deg(\mathrm{irr}_K(\alpha)).$$

When $K = \mathbb{Q}$, we write $\deg(\alpha)$ for $\deg_{\mathbb{Q}}(\alpha)$.

Theorem 1.3.11 ([15, Theorem 5.1.1])**.** *Let K be a subfield of $\mathbb{C}$. Let $\alpha \in \mathbb{C}$ be algebraic over K. Then $\mathrm{irr}_K(\alpha)$ is irreducible in $K[x]$.*

Definition 1.3.12. Let K be a subfield of $\mathbb{C}$.

(i) Let $\alpha \in \mathbb{C}$. Then we write

$$K(\alpha) = \bigcap_{\substack{K \subseteq F \subseteq \mathbb{C} \\ \alpha \in F}} F,$$

where the intersection is taken over all subfields F of $\mathbb{C}$ which contain both K and α.

(ii) A subfield L of $\mathbb{C}$ for which there exists $\alpha \in \mathbb{C}$ such that $L = K(\alpha)$ is called a *simple extension* of K.

Since the intersection of subfields of $\mathbb{C}$ is again a subfield of $\mathbb{C}$, $K(\alpha)$ is the smallest field containing both K and α. We say that $K(\alpha)$ is formed from K by adjoining a single element α. Clearly if $\alpha \in K$, then $K(\alpha) = K$.

Theorem 1.3.13 ([15, Theorem 5.5.1])**.** *Let K be a subfield of $\mathbb{C}$. Let $\alpha \in \mathbb{C}$ be algebraic over K of degree n. Then*

$$K(\alpha) = \left\{ a_0 + a_1\,\alpha + \ldots + a_{n-1}\,\alpha^{n-1} \mid a_0, a_1, \ldots, a_{n-1} \in K \right\}.$$

Theorem 1.3.13 shows that $K(\alpha)$ can be viewed as an $n-$dimensional vector space over K with basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$.

Definition 1.3.14. Let K be a subfield of $\mathbb{C}$, let $\alpha_1, \ldots, \alpha_k \in \mathbb{C}$ $(k \geq 2)$. Then the field $L = K(\alpha_1, \ldots, \alpha_k)$ defined inductively by

$$K(\alpha_1, \ldots, \alpha_k) = \ldots = K(\alpha_1, \ldots, \alpha_{k-1})(\alpha_k)$$

is called a *multiple extension* of K.

In this definition, L is the smallest subfield of $\mathbb{C}$ that contains both K and the α_i.

Theorem 1.3.15 ([15, Theorem 5.6.2]). *Let K be a subfield of $\mathbb{C}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{C}$ be algebraic over K. Then there exists $\alpha \in \mathbb{C}$ that is algebraic over K such that*

$$K(\alpha_1, \alpha_2, \ldots, \alpha_k) = K(\alpha).$$

The proof of Theorem 1.3.15 is constructive (see [15, Example 5.6.1]).

Definition 1.3.16. An *algebraic number field* is a subfield of $\mathbb{C}$ of the form

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_k),$$

where $\alpha_1, \ldots, \alpha_k$ are algebraic numbers.

Example 1.3.17. $\mathbb{Q}(\sqrt{2}, \sqrt{11})$ and $\mathbb{Q}(\sqrt{31})$ are examples of algebraic number fields.

By Theorem 1.3.15, each algebraic number field can be obtained by adjoining a single algebraic number α to $\mathbb{Q}$. Moreover, by Theorem 1.3.13 any element of an algebraic number field $K = \mathbb{Q}(\alpha)$ can be written as a polynomial in α with coefficients in $\mathbb{Q}$.

For a field K and an element α of an extension field E of K, we have a homomorphism

$$\begin{aligned} \varphi : K[x] &\longrightarrow E \\ f(x) &\longmapsto f(\alpha). \end{aligned}$$

If the kernel of this map is $\langle 0 \rangle$, then for $f \in K[x]$,

$$f(\alpha) = 0 \Longrightarrow f = 0 \text{ (in } K[x]).$$

This means there does not exist a non-zero polynomial in $K[x]$ satisfying $f(\alpha) = 0$, which shows that α is not algebraic, and that the homomorphism

$$K[x] \longrightarrow K[\alpha]$$

is an isomorphism, and that it extends to an isomorphism

$$K(x) \longrightarrow K(\alpha).$$

On the other hand, if the kernel of φ is not the zero ideal, then we have $f(\alpha) = 0$ for some non-zero $f \in K[x]$ which implies that α is algebraic over K by Definition 1.3.4. In this case, φ defines an isomorphism

$$K[x]/\ker\varphi \longrightarrow K[\alpha].$$

Moreover, if f is the minimal polynomial of α, then $\ker\varphi = \langle f \rangle$. Hence

$$K[\alpha] = K(\alpha) \cong K[x]/\langle f \rangle. \tag{1.3}$$

Example 1.3.18. Let $\alpha \in \mathbb{C}$ be such that $\alpha^3 - 3\alpha - 1 = 0$. Then the polynomial $x^3 - 3x - 1 \in \mathbb{Q}[x]$ is monic, irreducible, and has α as a root, and so it is the minimum polynomial of α over $\mathbb{Q}$. The set $\{1, \alpha, \alpha^2\}$ is a basis for $\mathbb{Q}[\alpha]$ over $\mathbb{Q}$ and

$$\mathbb{Q}[\alpha] = \mathbb{Q}(\alpha) \cong \mathbb{Q}[x]/\langle x^3 - 3x - 1 \rangle.$$

Note that an element t of E which is not algebraic over K is called *transcendental*.

1.4 Dense Univariate Rational Interpolation

In this section we discuss *polynomial interpolation* and, more generally, *rational function interpolation*. Throughout this section, we assume that F is a field.

The problem of (univariate) rational interpolation is, given distinct elements $x_0, \ldots, x_{n-1} \in F$ and arbitrary elements $y_0, \ldots, y_{n-1} \in F$, and a positive integer k with $0 \leq k \leq n$, to find a rational function $\Gamma(x) = r/t \in F(x)$, with $r, t \in F[x]$ satisfying $\deg r + \deg t < n, \deg r < k$, and $\deg t \leq n - k$ such that

$$t(x_i) \neq 0 \text{ and } \Gamma(x_i) = \frac{r(x_i)}{t(x_i)} = y_i \text{ for } 0 \leq i \leq n-1. \tag{1.4}$$

Note that the representation of the rational function $\Gamma(x)$ is not necessarily unique; however, if it is in canonical form, that is, $\mathrm{lc}(t) = 1$ and $\gcd(r, t) = 1$, then $\Gamma(x)$ is uniquely determined.

We will see later that this interpolation problem can be solved in two steps. For the first step, we need an algorithm that solves the polynomial interpolation problem, see equation (1.5) below.

1.4.1 Polynomial Interpolation

Polynomial interpolation is a simple but important algebraic tool to determine a polynomial formula from a sequence of numerical data.

The problem of (univariate) polynomial interpolation is, given n distinct elements x_0, x_1, …, $x_{n-1} \in F$, and $y_0, y_1, \ldots, y_{n-1} \in F$, to find a polynomial $P_{n-1}(x) \in F[x]$ of degree at most $n-1$ such that the following interpolation conditions are satisfied:

$$P_{n-1}(x_i) = y_i \quad \text{for } 0 \leq i \leq n-1. \tag{1.5}$$

The existence and uniqueness of an interpolating polynomial is given by the following theorem which can be found in any introductory book on numerical analysis, see, for example, [12, Theorem 3.2].

Theorem 1.4.1. *Let $x_0, x_1, \ldots, x_{n-1} \in F$ be distinct elements, and let $y_0, \ldots, y_{n-1} \in F$. Then a unique polynomial $P_{n-1}(x) \in F[x]$ of degree less than n exists satisfying the interpolation conditions (1.5). Moreover, this polynomial is inductively given by*

$$P_j(x) = v_0 + \sum_{i=1}^{j} v_i \prod_{k=0}^{i-1} (x - x_k) \tag{1.6}$$

for $0 \leq j \leq n-1$ where the v_i are given by

$$\begin{aligned} v_0 &= y_0, \\ v_i &= \frac{y_i - P_{i-1}(x_i)}{\prod_{k=0}^{i-1}(x_i - x_k)} \quad \text{for } 1 \leq i \leq n-1. \end{aligned}$$

Note that we do not assume any ordering between the points $x_0, x_1, \ldots, x_{n-1}$, as such an order will make no difference. However, we limit the degree of $P_{n-1}(x)$ to be at most $n-1$ because it singles out precisely one interpolating polynomial that will do the job.

The interpolation formula given in (1.6) is called the *Newton form of the interpolation formula*. The polynomial $P_{n-1}(x)$ given in the theorem above can easily be computed with the following straightforward algorithm which can be performed with $O(n^2)$ operations in F, see [42, Theorem 5.1]:

Algorithm 1.1 Polynomial Interpolation (`polyInterpolation`)

Input: $x_0, x_1, \ldots, x_{n-1} \in F$ pairwise distinct, and $y_0, y_1, \ldots, y_{n-1} \in F$.

Output: An interpolating polynomial $P_{n-1}(x) \in F[x]$ of degree at most $n-1$ satisfying the interpolation conditions (1.5).

1: $P_0(x) \longleftarrow y_0, \quad g_0(x) \longleftarrow 1$
2: **for** $j = 1, \ldots, n-1$ **do**
3: $\quad s_j \longleftarrow x_j - x_0, \quad t_j \longleftarrow y_j - P_{j-1}(x_j), \quad g_j(x) \longleftarrow g_{j-1}(x) \cdot (x - x_{j-1})$
4: $\quad$ **for** $k = 1, \ldots, j-1$ **do**
5: $\quad\quad s_j \longleftarrow s_j \cdot (x_j - x_k)$
6: $\quad v_j \longleftarrow t_j / s_j, \quad P_j(x) \longleftarrow P_{j-1}(x) + v_j \cdot g_j(x)$
7: **return** $P_{n-1}(x)$

Definition 1.4.2. ([25, Definition 8.26]) Let R be a ring. A function $M : \mathbb{N}_{>0} \longrightarrow \mathbb{R}_{>0}$ is called a *multiplication time* for $R[x]$ if polynomials in $R[x]$ of degree less than n can be multiplied using at most $M(n)$ operations in R.

For example, the classical algorithm for multiplying two polynomials in $R[x]$ of degree less than n uses at most $M(n) = 2n^2$ operations in R, see [25, Table 8.6].

Remark 1.4.3. Given an algorithm for multiplying two polynomials of degree less than n, let $M(n)$ be its complexity. Then based on this algorithm, there is a fast polynomial interpolation algorithm which takes at most $O(M(n)\log n)$ arithmetic operations in F, see [25, Section 10.2]. However, we do not consider it here.

Turning our attention to the rational interpolation problem (1.4), we now briefly explain how this problem can be solved in two steps. In the first step, we apply Algorithm 1.1 to compute a unique interpolating polynomial $g := P_{n-1}(x) \in F[x]$ of degree less than n such that $g(x_i) = y_i$ for $i = 0, \ldots, n-1$. Let $f = \prod_{i=0}^{n-1}(x - x_i)$. In the second step, we wish to find a rational function $r/t \in F(x)$ satisfying

$$\gcd(f, t) = 1 \text{ and } rt^{-1} \equiv g \bmod f, \ \deg r < k, \ \deg t \leq n - k, \tag{1.7}$$

where $t^{-1} \in F[x]$ is the inverse of t modulo f.

The algorithm recovering fractions in $F(x)$ modulo f that we consider in Subsection 1.4.2 relies on the extended Euclidean algorithm (EEA), see, for example, [42, Algorithm 3.14]. We thus fix, only for this section, the following notation. Let f and g be polynomials in $F[x]$ with $\deg f > \deg g$. Set $r_0 := f$ and $r_1 := g$. By division with remainder, we inductively define $r_{i+1} = r_{i-1} - q_i r_i$, where the q_i are the quotients, $\deg r_{i+1} < \deg r_i$. This process ends when $r_{l+1} = 0$ is reached for some $l \in \mathbb{N}$. For the extended Euclidean algorithm, we additionally consider coefficients s_i and t_i inductively defined by $s_0 = t_1 = 1$, $s_1 = t_0 = 0$, $s_{i+1} = s_{i-1} - q_i s_i$ and $t_{i+1} = t_{i-1} - q_i t_i$ for $1 \leq i \leq l$. Note that we obtain, thus, a representation

$$r_i = s_i f + t_i g$$

of the r_i as a linear combination of the input polynomials (see Lemma 1.4.4 below). The elements r_i, s_i and t_i form the i-th row in the EEA, for $1 \leq i \leq l$. We will see later that the intermediate results r_i, q_i and t_i computed by the extended Euclidean algorithm are useful for recovering fractions in $F(x)$.

The following lemma presents some properties of the EEA.

Lemma 1.4.4 ([42, Lemma 3.8 and 3.15])**.** *Let $r_0 = f$, $r_1 = g$ for $f, g \in F[x]$, and let r_i, s_i, t_i for $0 \leq i \leq l+1$ be the rows of the* EEA *for the pair (r_0, r_1), with $r_{l+1} = 0$. Let $n_i = \deg r_i$. Then for $0 \leq i \leq l$, we have*

(i) $\gcd(f, g) = \gcd(r_i, r_{i+1}) = r_l$;

(ii) $s_i f + t_i g = r_i$; *in particular,* $s_l f + t_l g = \gcd(f, g)$;

(iii) $\gcd(s_i, t_i) = 1$;

(iv) $\gcd(r_i, t_i) = \gcd(f, t_i)$;

(v) $\deg s_{i+1} = n_1 - n_i$, *and* $\deg t_{i+1} = n_0 - n_i$ *for* $i \neq 0$.

1.4.2 Univariate Rational Function Reconstruction

This subsection presents an algorithm for computing a rational function that solves (1.7) if such a function exists. The general problem of rational reconstruction consists of rational number reconstruction and rational function reconstruction problems. Rational number (resp. function) reconstruction is a method that allows one to recover a rational number (resp. function) from its image modulo an integer (resp. a polynomial). Rational number reconstruction, originally developed by Paul Wang in 1981 [43], has found many applications in computer algebra. Later, in 2004, Michael Monagan [33] presented a more efficient solution for the rational number reconstruction problem, which he called *Maximal Quotient Rational Reconstruction* (MQRR). His algorithm applies the EEA to inputs $m, u \in \mathbb{Z}$ with $\gcd(m, u) = 1$ and outputs a rational number r_i/t_i where i

represents the index of the maximal quotient q_i appearing in the EEA for the inputs $m, u \in \mathbb{Z}$. In the following we will see that the same strategy can be used to recover fractions in $F(x)$ from their image modulo a polynomial $f(x) \in F[x]$. For a more detailed description, we refer to [33, 42].

Definition 1.4.5. A rational function $r/t \in F(x)$ is said to be in *canonical form* if $\mathrm{lc}(t) = 1$ and $\gcd(r, t) = 1$.

In (1.7), since t is a unit modulo f, we may multiply the congruence by t to obtain an equivalent condition. When we drop the gcd requirement, we obtain

$$r \equiv tg \bmod f,\ \deg r < k,\ \deg t \leq n - k\,. \tag{1.8}$$

This condition is strictly weaker because for some fixed $k \in \{0, \ldots, n\}$, there are cases where (1.7) has no solution. Let us take a look at the following example:

Example 1.4.6. Consider $f = x^3 - x,\ g = x^2 + 1 \in \mathbb{Q}[x]$. Here, for $k = 3$, we have the trivial solution $r = g$ and $t = 1$ which satisfies both (1.7) and (1.8). However, if $k = 2$, then the solution $r = x$ and $t = \frac{1}{2}x$ satisfies (1.8), but not (1.7).

By the following theorem, there is an algorithm which decides whether (1.7) is solvable.

Theorem 1.4.7 ([42, Theorem 5.16])**.** *Let $f \in F[x]$ be of degree $n > 0$ and let $g \in F[x]$ be of degree less than n. Furthermore, let $r_j, s_j, t_j \in F[x]$ be the j-th row in the* EEA *for the pair (f, g), where j is minimal such that $\deg r_j < k$.*

(i) *There exist polynomials $r, t \in F[x]$ satisfying (1.8), namely $r = r_j$ and $t = t_j$. If in addition $\gcd(r_j, t_j) = 1$, then r and t also solve (1.7).*

(ii) *If $r/t \in F(x)$ is a canonical form solution to (1.7), then $r = \beta^{-1} r_j$ and $t = \beta^{-1} t_j$, where $\beta = \mathrm{lc}(t_j) \in F \setminus \{0\}$. In particular, (1.7) is solvable if and only if $\gcd(r_j, t_j) = 1$.*

The algorithm which we obtain from the above theorem finds a canonical form solution (if it exists) to (1.7) for any fixed k. Since for our application we need the sum of the degrees of r and t in Theorem 1.4.7 to be minimal, it is very important to know the value of k in advance for which the solution r/t satisfies this condition. Nevertheless, there is an algorithm, due to Monagan [33], which finds a rational function of small degree even without an integer k being supplied. This algorithm is called the *Maximal Quotient Rational Function Reconstruction* (MQRFR).

Maximal Quotient Rational Function Reconstruction

The MQRFR algorithm presented in [33] finds a correct solution for (1.7) satisfying $n > \deg r + \deg t + 1$. Like over the rational numbers, Monagan's algorithm runs the EEA on the input $f, g \in F[x]$ with $\gcd(g, f) = 1$ and outputs the rational function r_i/t_i, where i represents the index of the quotient q_i of maximal degree appearing in the EEA. Let us start with an example that illustrates how the MQRFR algorithm works:

Example 1.4.8 ([33, Example 2.1])**.** Consider $f = \prod_{i=5}^{12}(x - i)$ and $g = 10x^7 + x^6 + 2x^5 + 10x^4 + 12x^3 + 7x^2 + 12x + 8 \in \mathbb{Z}_{13}[x]$. The EEA with inputs f and g yields the following table.

i	$\deg r_i$	$\deg t_i$	$\deg r_i + \deg t_i$	$\deg q_i$
1	7	0	7	1
2	6	1	7	1
3	5	2	7	1
4	2	3	5	3
5	1	6	7	1
6	0	7	7	1

From the table one can simply choose a rational function r_i/t_i where $\deg r_i + \deg t_i$ is minimal. As illustrated in the table, $\deg r_i + \deg t_i$ reaches its minimum for $i = 4$ which also corresponds to the quotient q_4 of maximal degree 3. The reason for this is clear, as explained in the following lemma.

Lemma 1.4.9. *Let $f, g \in F[x]$. In the* EEA *for f and g we have, with notation as above,*

$$\deg r_i + \deg t_i + \deg q_i = \deg f$$

for $1 \leq i \leq l$, where l is the total number of division steps.

Proof. By Lemma 1.4.4 (v), we have $\deg t_i = \deg f - \deg r_{i-1}$. Now

$$\begin{aligned}\deg r_i + \deg t_i + \deg q_i &= \deg r_i + (\deg f - \deg r_{i-1}) + (\deg r_{i-1} - \deg r_i)\\ &= \deg f.\end{aligned}$$

□

The following lemma states that if the degree of f is large enough then there is only one pair of (r_j, t_j) such that $\deg r_j + \deg t_j$ is minimal.

Lemma 1.4.10 ([33, Lemma 2.3])**.** *Let $r, t \in F[x]$ be polynomials with $\mathrm{lc}(t) = 1$ and $\gcd(r, t) = 1$. Let f, g be two polynomials in $F[x]$ satisfying $\gcd(f, t) = 1$ and $g \equiv$*

$rt^{-1} \bmod f$, *where* t^{-1} *is the inverse of* t *modulo* f. *Let* j *denote the index of a quotient with maximal degree in the* EEA *with inputs* f *and* g. *If* $\deg f > 2 \cdot (\deg r + \deg t)$, *then* j *is unique, and we have* $r = r_j$ *and* $t = t_j$.

Definition 1.4.11. Let $f \in F[x]$. We define

a) the *normal form* of f, denoted by $\operatorname{normal}(f)$, as:

$$\operatorname{normal}(f) = \begin{cases} f/\operatorname{lc}(f) & \text{if } f \neq 0 \\ 0 & \text{otherwise;} \end{cases}$$

b) the *leading unit* of f, denoted by $\operatorname{lu}(f)$, as:

$$\operatorname{lu}(f) = \begin{cases} \operatorname{lc}(f) & \text{if } f \neq 0 \\ 1 & \text{otherwise.} \end{cases}$$

Taking the above results into account, we have Algorithm 1.2.

Algorithm 1.2 MQRFR (`fareypoly(g,f)`)

Input: $g, f \in F[x]$ and $n = \deg f > \deg g$.

Output: $r, t \in F[x]$ with $\deg r + \deg t + 1 < n$, $\mathrm{lc}(t) = 1$, $\gcd(r,t) = 1$, and $r/t \equiv g \bmod f$, or FAIL implying that no solution of (1.7) exists.

1: $\rho \longleftarrow \mathrm{lu}(g)$
2: $(r_1, r_2) \longleftarrow (\mathrm{normal}(f), \mathrm{normal}(g))$
3: **if** $r_2 = 0$ **then**
4: **return** $(0, 1)$
5: **if** $2 \cdot \deg g < n$ **then**
6: **return** $(g, 1)$
7: $(t_1, t_2) \longleftarrow (0, 1/\rho)$
8: $(r_3, r_m, q) \longleftarrow (0, 0, 0)$ (the index m refers to maximum)
9: $(q_m, t_m) \longleftarrow (1, 1)$
10: **while** $r_2 \neq 0$ **do**
11: $(q, r_3) \longleftarrow (r_1 \text{ quo } r_2, r_2)$ (q is the quotient in $F[x]$ on dividing r_1 by r_2)
12: $r_2 \longleftarrow r_1 \text{ rem } r_2$ (r_2 is the remainder in $F[x]$ on dividing r_1 by r_2)
13: $(\rho, r_2) \longleftarrow (\mathrm{lu}(r_2), r_2/\rho)$
14: $(r_1, r_3) \longleftarrow (r_3, t_2)$
15: $(t_1, t_2) \longleftarrow (r_3, (t_1 - qt_2)/\rho)$
16: **if** $\deg q > \deg q_m$ **then**
17: $(q_m, r_m, t_m) \longleftarrow (q, r_1, t_1)$
18: **if** $\deg q_m = 1$ **then**
19: **return** FAIL
20: **if** $\gcd(r_m, t_m) \neq 1$ **then**
21: **return** FAIL
22: **return** $(r_m/\mathrm{lc}(t_m), t_m/\mathrm{lc}(t_m))$

In Algorithm 1.2, we start by setting $\rho := \mathrm{lu}(g)$ and $(r_1, r_2) := (\mathrm{normal}(f), \mathrm{normal}(g))$, see Definition 1.4.11. If $g = 0$, then the only solution to (1.7) is $\frac{0}{1} = 0$, see lines 3-4.

If $2 \cdot \deg g < n$ (see line 5), then the unique solution to (1.7) which corresponds to the quotient of maximal degree is $(g, 1)$ (see line 6), see Lemma 1.4.10. This is because by division with remainder, we have $f = qg + s$ with $\deg s < \deg g$ and, hence, $\deg g = \deg f - \deg q$. This implies $\deg f < 2 \cdot \deg q$. Let r and t be polynomials which correspond to q in the EEA. Then $2 \cdot (\deg r + \deg t) < \deg f$, see Lemma 1.4.9. In this case, Lemma 1.4.10 implies that one division step yields $(r, t) = (r_2/\mathrm{lc}(t_2), t_2/\mathrm{lc}(t_2)) = (g, 1)$ which is a solution to (1.7) in canonical form, see line 6.

In all the other cases, a solution is computed by division with remainder. The while loop in lines 10-17 is a simple modification of the while loop in the extended Euclidean algorithm, see, for example, [42, Algorithm 3.14]. The algorithm returns FAIL if the degree of q_m is 1 since by Lemma 1.4.9 the algorithm requires the degree to be at least 2 (see line 18), or if $\gcd(r_m, t_m) \neq 1$ (see line 20) since in this case, $\frac{r_m}{t_m}$ is not a solution to (1.7), see Example 1.4.6. But if $\gcd(r_m, t_m) = 1$, then $\frac{r}{t}$ with $(r, t) = (r_m/\mathrm{lc}(t_m), t_m/\mathrm{lc}(t_m))$ is a solution to (1.7) in canonical form by Lemma 1.4.7, see line 22.

Note that Algorithm 1.2 computes a unique solution of (1.7) using $O(n^2)$ operations in F (see, for example, [42, Corollary 5.17]), where n is the degree of f.

Remark 1.4.12. There is a fast rational function reconstruction algorithm which takes $O(M(n)\log n)$ arithmetic operations in F, where $M(n)$ is as in Definition 1.4.2, see [42, Section 11.1] and [33]. This algorithm is based on a fast extended Euclidean algorithm (FEEA) which in turn is based on a fast algorithm to compute the quotients in the Euclidean algorithm for univariate polynomials over F, using $O(M(n)\log n)$ field operations for inputs of degree at most n. The fast algorithm for the quotients uses a fast polynomial multiplication algorithm, see, for example, [42, Chapter 8]. In [33], it is pointed out that in practice, the usual EEA performs better than the FEEA for polynomials of low degree. However, the Java implementation of the FEEA beats the EEA for $\deg f \geq 200$.

The fast algorithm highlighted in the remark above is not considered in this thesis. We thus close this section with an example illustrating how to reconstruct a rational function from given numerical data.

Example 1.4.13. Compute a rational function $\Gamma(x) \in \mathbb{Q}(x)$ for the following interpolation points:

i	0	1	2	3	4	5
x_i	1	2	3	4	5	6
$\Gamma(x_i)$	39/7	10	171/11	288/13	149/5	654/17

In the first step, using the information given in the table, Algorithm 1.1 computes the unique polynomial

$$g = \frac{-12}{85085}x^5 + \frac{282}{85085}x^4 - \frac{3}{91}x^3 + \frac{11675}{17017}x^2 + \frac{217457}{85085}x + \frac{200748}{85085} \in \mathbb{Q}[x]$$

of degree less than 6. Let $f = \prod_{i=0}^{5}(x - x_i)$. We then apply Algorithm 1.2 to the pair (g, f) to obtain

$$\Gamma(x) = \frac{1/2x^3 + 9/2x^2 + 17/2x + 6}{x + 5/2} \in \mathbb{Q}(x). \tag{1.9}$$

In this example, the rational function $\Gamma(x) = \frac{r(x)}{t(x)}$ with $(r, t) = (1/2x^3 + 9/2x^2 + 17/2x + 6, x + 5/2)$ satisfies the following conditions:

(1) The value of $\frac{r}{t}$ at $x = x_i$ coincides with the corresponding value $\Gamma(x_i)$ given in the table for each i,

(2) $\deg r + \deg t + 1 < \deg f = 6$, and

(3) $\gcd(r, t) = \gcd(t, f) = 1$. Here, the fact that $\gcd(r, t) = \gcd(t, f)$ follows from Lemma 1.4.4 (iv). Thus it suffices to show that either $\gcd(r, t) = 1$ or $\gcd(t, f) = 1$. Since the degrees of r and t are relatively small in comparison to the degree of f, we prefer to check whether r and t are coprime. Indeed, one can easily verify that $\gcd(r, t) = 1$.

1.5 Sparse Multivariate Polynomial Interpolation

Suppose we are given a black box, see Figure 1.1 below, which contains a multivariate polynomial $B = B(x_1, \ldots, x_n)$ represented as

$$B = \sum_{i=1}^{t} c_i B_i \in \mathbb{Q}[x_1, \ldots, x_n], \tag{1.10}$$

in the standard power basis where

$$B_i = x_1^{e_{i,1}} \cdots x_n^{e_{i,n}} \text{ and } c_i \in \mathbb{Q}.$$

The black box implicitly defines the unknown multivariate polynomial through substituting elements from a given field for the variables. Thus to determine the coefficients and terms of a black box polynomial is the problem of black box interpolation. There are various techniques for interpolating sparse polynomials such as Zippel's probabilistic algorithm [45] and the algorithm of Ben-Or and Tiwari [3]. In this section, for our purpose, only the algorithm by Ben-Or and Tiwari will be discussed for the following reasons:

(1) It is a deterministic algorithm for interpolating a multivariate polynomial in a polynomial ring with characteristic zero.

(2) It does not interpolate a multivariate polynomial one variable at a time. Instead, it interpolates all variables at once.

(3) Although it requires a bound τ on the number of terms, it does not require a bound on the partial degrees $d_i = \deg_{x_i} B,\ 1 \leq i \leq n$.

Due to the above reasons we prefer to use the interpolation algorithm by Ben-Or and Tiwari as a tool to recover polynomials in a polynomial ring with characteristic zero, see Chapter 4 and Chapter 5.

Figure 1.1: Black box for polynomial evaluation

1.5.1 The Sparse Interpolation Algorithm of Ben-Or and Tiwari

If the number of variables n and a bound τ on the number of terms t, that is, $\tau \geq t$ are given, then the algorithm uses the first n distinct primes $p_1, p_2, \ldots, p_n$ in the 2τ evaluation points $e_j = (p_1^j, p_2^j, \ldots, p_n^j),\ 0 \leq j \leq 2\tau - 1$. The algorithm can be partitioned into two phases. In the first phase, we determine the exponents $e_{i,k}$ using a linear generator, and then in the second phase we determine the coefficients c_i by solving a linear system of equations over $\mathbb{Q}$.

Suppose for simplicity $\tau = t$. Let a_j be the output from a probe to the black box with the input e_j for $0 \leq j \leq 2t-1$. Let $b_i = B_i(p_1, \ldots, p_n) = p_1^{e_{i,1}} \cdots p_n^{e_{i,n}}$. Then

$$a_j = B(e_j) = \sum_{i=1}^{t} c_i b_i^j.$$

In order to determine the integers b_i, let us define an auxiliary polynomial $\Lambda(z)$ as

$$\Lambda(z) = \prod_{i=1}^{t} (z - b_i) = z^t + \lambda_{t-1}\, z^{t-1} + \ldots + \lambda_0 \in \mathbb{Q}[z]. \tag{1.11}$$

In the following we show how to determine the coefficients of this polynomial by solving a linear system. This linear system tells us the relation between the coefficients of the auxiliary polynomial $\Lambda(z)$ and the sequence $\{a_j\}_{j \geq 0}$. The linear system for determining the coefficients λ_i is derived as follows: First observe that, since $\Lambda(b_i) = 0$ for any $1 \leq i \leq t$, we have

$$0 = c_i b_i^j\, \Lambda(b_i) = c_i \left(\lambda_0\, b_i^j + \lambda_1\, b_i^{j+1} + \ldots + b_i^{j+t}\right).$$

Summing over all i, we get

$$0 = \lambda_0 \sum_{i=0}^{t} c_i b_i^j + \lambda_1 \sum_{i=0}^{t} c_i b_i^{j+1} + \ldots + \sum_{i=0}^{t} c_i b_i^{j+t}. \tag{1.12}$$

But it also holds that

$$b_i^j = (p_1^{e_{i,1}} \cdots p_n^{e_{i,n}})^j = (p_1^j)^{e_{i,1}} \cdots (p_n^j)^{e_{i,n}} = B_i(p_1^j, \ldots, p_n^j) = B_i(e_j). \tag{1.13}$$

This implies

$$\sum_{i=1}^{t} c_i b_i^j = \sum_{i=1}^{t} c_i B_i(e_j) = B(e_j) = a_j. \tag{1.14}$$

Using the relation (1.14) for all $j \geq 0$, we can rewrite (1.12) as

$$-a_{j+t} = \lambda_0\, a_j + \lambda_1\, a_{j+1} + \ldots + \lambda_{t-1}\, a_{j+t-1}. \tag{1.15}$$

This last equation gives us the linear relation we want in order to determine the coefficients λ_i. Now to determine t coefficients $\lambda_0, \ldots, \lambda_{t-1}$, we need t such linear relations. Letting $j = 0, \ldots, t-1$, we need $2t$ elements $a_0, \ldots, a_{2t-1}$, so we make $2t$ probes to the black box. Once we have these evaluations, we can solve $A\,\lambda = a$, where

$$A = \begin{bmatrix} a_0 & a_1 & \ldots & a_{t-1} \\ a_1 & a_2 & \ldots & a_t \\ \vdots & \vdots & \ddots & \vdots \\ a_{t-1} & a_t & \ldots & a_{2t-2} \end{bmatrix}, \quad \lambda = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{t-1} \end{bmatrix}, \quad a = \begin{bmatrix} -a_t \\ -a_{t+1} \\ \vdots \\ -a_{2t-1} \end{bmatrix}. \tag{1.16}$$

This method of finding a linear generator $\Lambda(z)$ for B costs $O(\tau^3)$ arithmetic operations in $\mathbb{Q}$. Thus we use the *Berlekamp/Massey Algorithm* (BMA) (see, for example, [3, Section 2.1]), a technique from coding theory, to reduce the runtime to $O(\tau^2)$ arithmetic operations in $\mathbb{Q}$. The algorithm depends on the key equation (1.15). It works by processing a sequence of elements $a_0, a_1, \ldots \in F$ where F is a field. If the sequence has a linear generator $\Lambda(z)$ of degree t satisfying the key equation (1.15), the algorithm will compute it after processing $2t$ elements from the sequence. There is no a priori bound for the sequence and, hence, the algorithm can update the current guess for the linear generator appropriately whenever the next element a_i of the sequence does not fit the current linear recursion. In that case a non-zero discrepancy

$$\Delta_j = \lambda_s\, a_{j-1} + \lambda_{s-1}\, a_{j-2} + \lambda_{s-2}\, a_{j-3} + \ldots + \lambda_0\, a_{j-s-1} \tag{1.17}$$

is detected for $0 \leq s < t$ and $s+1 \leq j \leq 2s+1$. This relation can be obtained in a similar way as Equation (1.15). To see this, note that the algorithm computes the reverse of the generator polynomial. That is, in contrast to the polynomial Λ in (1.11), we define the same polynomial with reversed coefficients. In this case, we have

$$\Lambda(z) = \lambda_0\, z^t + \lambda_1\, z^{t-1} + \ldots + 1 = (-1)^t \prod_{i=1}^{t} (b_i z - 1). \tag{1.18}$$

Within the Berlekamp/Massey algorithm, the polynomial Λ in (1.18) is internally computed by using the relation in (1.17) but only if $\Delta_j = 0$ it is linearly factored. In this case, the algorithm reverses the coefficients λ_i and returns the polynomial Λ as in (1.11). Coming back to Equation (1.18), since $\Lambda(b_i^{-1}) = 0$ for any $1 \leq i \leq t$, we have

$$0 = c_i b_i^{j-1} \Lambda(b_i^{-1}) = c_i \left(\lambda_0 b_i^{j-t-1} + \lambda_1 b_i^{j-t} + \ldots + b_i^{j-1}\right).$$

From (1.12) to (1.15), we obtain the following relation by a similar argument :

$$0 = \lambda_t a_{j-1} + \lambda_{t-1} a_{j-2} + \lambda_{t-2} a_{j-3} + \ldots + \lambda_0 a_{j-t-1}.$$

If, in this relation, t is less than the number of terms of B (or equivalently, if t is less than the degree of the generating polynomial Λ), then the equality does not hold, that is,

$$\Delta_j := \lambda_s a_{j-1} + \lambda_{s-1} a_{j-2} + \lambda_{s-2} a_{j-3} + \ldots + \lambda_0 a_{j-s-1} \neq 0$$

for $0 \leq s < t$ and $s+1 \leq j \leq 2s+1$. Thus we have obtained the relation we wanted. For a detailed description, we refer to [31].

Now, once we have found λ_i, we compute all integer roots of $\Lambda(z)$ to obtain b_i, see, for example, [3, integer root finding algorithm in Section 5]. Then each term $B_i = x_1^{e_{i,1}} \cdots x_n^{e_{i,n}}$ is recovered through repeatedly dividing b_i by $p_1, p_2, \ldots, p_n$. Finally, the coefficients c_i are computed via solving the linear system

$$a_j = \sum_{i=1}^{t} c_i b_i^j \quad \text{with } 0 \leq j \leq t-1,$$

which turns out to be a $t \times t$ transposed Vandermonde system:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ b_1 & b_2 & \cdots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{t-1} & b_2^{t-1} & \cdots & b_t^{t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} \tag{1.19}$$

Before presenting the algorithm, let us first take a look at the following example:

Example 1.5.1. If $B = x^2y + 2$, then $t = 2$. Let $e_j = (2^j, 3^j)$ for $0 \leq j \leq 2t-1 = 3$. Then $a_0 = B(e_0) = 3, a_1 = 14, a_2 = 146, a_3 = 1730$. From the system of equations

$$\begin{bmatrix} a_0 & a_1 \\ a_1 & a_2 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} -a_2 \\ -a_3 \end{bmatrix}$$

for $0 \leq j \leq 3$, as in (1.16), we obtain

$$\begin{aligned} &\lambda_1 = -13,\ \lambda_0 = 12 \\ &\quad \Longrightarrow \Lambda(z) = z^2 - 13z + 12 = (z-1)(z-12) \\ &\quad \Longrightarrow b_1 = 1 = 2^0 \cdot 3^0,\ b_2 = 12 = 2^2 \cdot 3^1 \\ &\quad \Longrightarrow B_1 = 1,\ B_2 = x^2y \end{aligned}$$

From the 2×2 transposed Vandermonde system

$$\begin{bmatrix} 1 & 1 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix},$$

we obtain $c_1 = 2$, and $c_2 = 1$.

The so-called *sparse multivariate polynomial interpolation* algorithm, see Algorithm 1.3 below, recovers multivariate polynomials given in a black box.

Algorithm 1.3 Sparse Multivariate Polynomial Interpolation (`sparseInterpolation`)

Input: $B(x_1, \ldots, x_n)$: a multivariate black box polynomial.
$\tau : \tau \geq t$, t is the number of non-zero terms in B.

Output: c_i and B_i such that $B(x_1, \ldots, x_n) = \sum_{i=1}^{t} c_i B_i$ and $c_i \neq 0$.

[The BMA]

1: $a_j = B(p_1^j, \ldots, p_n^j)$, $0 \leq j \leq 2\tau - 1$, where p_i is the i-th prime
2: compute $\Lambda(z)$ from $\{a_j\}_{2\tau-1 \geq j \geq 0}$

[Determine B_i]

3: find all t distinct roots b_i of $\Lambda(z)$
4: determine each B_i through repeatedly dividing every b_i by $p_1, \ldots, p_n$

[Compute the coefficients c_i]

5: solve the transposed Vandermonde system (1.19)
6: **return** c_i and B_i

In Algorithm 1.3, a severe coefficient swell occurs when the BMA is run over $\mathbb{Q}$, see line 2, which makes the algorithm very slow. This problem was addressed by Kaltofen et al. [30] using modular methods. Later, different approaches have been described to tackle the above problem. These approaches are highlighted in a recent paper by Javadi et al. [29]. In this paper, the drawbacks of the above approaches and the modular methods by Kaltofen et al. are given. A probabilistic algorithm described in [29] interpolates a sparse multivariate polynomial $f \in \mathbb{F}_p[x_1, \ldots, x_n]$ over a finite field, given in a black box. This algorithm modifies Algorithm 1.3 to interpolate f in a positive characteristic by doing extra probes to determine the degrees of the variables in each monomial in f. To interpolate the target polynomial, the algorithm needs to know a bound on the degree of f and on the number of terms in f. However, for the application that we are interested in, we prefer to consider the early termination strategy described in [31, 32], see the next section. This is because the method described in these papers is a useful tool for controlling intermediate coefficient swell in computer algebra and, hence, saves computation time. Moreover, it performs well for sparse examples, even without knowing any bounds on the degree or the number of terms. A more detailed description is given in the following section.

1.6 Early Termination Strategy

To interpolate a polynomial of degree at most n, we need $n+1$ evaluation points, see Subsection 1.4.1. Let F be a field. Suppose we are given, for a black box univariate polynomial f of unknown degree, $n+1$ evaluation points $(x_0, f(x_0)), \ldots, (x_n, f(x_n)) \in F^2$. To find the polynomial f, we need to know a degree bound. Recall that Algorithm 1.3 described in Subsection 1.5.1 requires a bound on the number of terms. Some important questions, when no such bound is given, are now the following:

(a) Given evaluation points for a black box univariate polynomial of unknown degree, is there a way to find a degree bound so that we can recover the black box polynomial?

(b) Given evaluation points for a black box multivariate polynomial with an unknown number of terms, is there a way to find a bound on the number of terms so that we can recover the black box polynomial?

(c) Given evaluation points for the polynomial in (a) or (b), is there any other approach to find the desired polynomial without knowing the bounds, the degree bound required in (a) and the bound on the number of terms required in (b), in advance?

Unfortunately, we do not know the answer for the questions (a) and (b). But for the last question, an efficient probabilistic approach, *early termination*, presented in [31, 32] finds the targeted polynomial when no such bound is supplied. The early termination algorithms are randomized in the Monte Carlo sense, that is, their results are correct with high probability, see [31, 32].

In this section, we shortly explain, closely following [32], how the early termination version of the sparse interpolation algorithm described in Subsection 1.5.1 works. Moreover, we also extend the early termination version of the polynomial interpolation method described in [32, Theorem 1] (see Theorem 1.6.1) to rational interpolations in a straightforward way. The basic idea behind the early termination approaches is based on the fact that an already interpolated rational function does not change as more interpolation points are added.

1.6.1 Early Termination with Threshold in Dense Rational Interpolations

We start by the following theorem:

Theorem 1.6.1 ([32, Theorem 1])**.** *Let F be a field. Let $f(x)$ be a black box univariate polynomial with coefficients in F, and let $\eta \in \mathbb{Z}$ be a positive integer. Suppose $p_0, p_1, \ldots$ are distinct points that are chosen randomly and uniformly from a subset S of F, and that the polynomial $f^{[i]}(x)$ is the i-th interpolant that interpolates $f(p_0), f(p_1), \ldots, f(p_i)$. If d is the smallest non-negative integer such that*

$$f^{[d]}(x) = f^{[d+1]}(x) = \ldots = f^{[d+\eta]}(x),$$

then $f^{[d]}(x)$ *correctly interpolates* $f(x)$ *with probability no less than*

$$1 - \eta \cdot \deg(f(x)) \cdot \left(\frac{\deg f(x)}{\#(S)}\right)^{\eta}.$$

In this theorem, we assume the points $p_0, p_1, \ldots$ to be distinct although they are not necessarily distinct in [32, Theorem 1]. However, it is pointed out that the assumption that $p_0, p_1, \ldots$ are distinct avoids false early termination due to interpolating at repeated points, see the paragraph below Lemma 2 in [32]. Indeed, it is very significant for the application that we are interested in that the above points are distinct.

The early termination strategy in the theorem above does not always work if the target function is rational, see, for example, the i-th interpolating polynomial $g_i = L[1]$ in Example 1.6.2. In order to work for a black box univariate rational function Γ, we do the same as in the above theorem except that for every $i \geq 0$, the polynomials $f^{[i]}(x)$ are considered as rational functions over the field F. The idea is as follows: to recover a fraction $\Gamma(x) \in F(x)$ from its evaluations at distinct points $x_0, x_1, \ldots$, Algorithm 1.1 first updates an i-th interpolant $g^{[i]}(x)$ for every $i \geq 0$ and Algorithm 1.2 is then applied to the pair $(g^{[i]}(x), f^{[i]}(x))$ to obtain the i-th rational function $\Gamma^{[i]}(x)$ where $g^{[i]}(x)$ is a polynomial interpolating $\Gamma(x_0), \ldots, \Gamma(x_i)$ of degree at most i and $f^{[i]}(x) = \prod_{j=0}^{i}(x - x_j)$. As we pointed out earlier, the polynomials $g^{[i]}(x)$ and $g^{[i+1]}(x)$ do not always coincide. Now, since at least one i-th order term is constructed in every $g^{[i]}(x)$, the target rational function $\Gamma(x)$ is recovered as a possible dense rational interpolation up to the degree bound. Note that for successful interpolation, the size of S needs to cover enough distinct points required by the early termination, that is, no less than $n_1 + d_1 + 1 + \eta$ points where n_1 and d_1 are the degrees of the numerator and the denominator of Γ, respectively.

An important fact in the rational interpolation algorithm is that once the target rational function is interpolated, the interpolant does not change even if we keep interpolating $\Gamma(x)$ at more distinct points. Based on this observation, the early termination with thresholds is applied as follows: An integer $\eta > 0$ is given as a threshold, the sequence $x_0, x_1, \ldots$ are random distinct values, $g^{[i]}(x)$ is updated for every $i \geq 0$ and, hence, a new rational function $\Gamma^{[i]}(x)$ is obtained. Whenever $\Gamma^{[i]}(x)$ stops changing η times in a row, we have $\Gamma(x) = \Gamma^{[i]}(x)$ with high probability.

The following example illustrates the early termination version of the univariate rational interpolation described in Section 1.4:

Example 1.6.2.

i	0	1	2	3	4	5	6
x_i	1	2	3	4	5	6	7
$\Gamma(x_i)$	39/7	10	171/11	288/13	149/5	654/17	915/19

We use the SINGULAR commands `polyInterpolation` and `fareypoly` from the SINGULAR library `ffmodstd.lib` [6] to compute the rational function Γ. In the following,

$i = \text{size}(d) - 1$, $L[1] = g_i := g^{[i]}(x)$ is the i-th interpolant, $L[2] = \prod_{j=0}^{i}(x - x_j) =: f_i$, and $L[3] = x_0, x_1, \ldots, x_i$. If there are not enough interpolation points, then the list L is required for adding more interpolation points. Moreover, $\Gamma^{[i]} := R_i[1]/R_i[2]$ is the i-th rational function interpolating Γ. For simplicity, let us start from $i = 4$:

```
> ring r = 0,x,dp;
> list d = 1,2,3,4,5;
> list e = 39/7,10,171/11,288/13,149/5;
> list L = polyInterpolation(d,e,1); // 1 refers to the output w.r.t.
                                     // the 1st variable
> L;
  L[1] = 6/5005*x^4-3/143*x^3+655/1001*x^2+371/143*x+11724/5005
  L[2] = x^5-15*x^4+85*x^3-225*x^2+274*x-120
  L[3] = 1,2,3,4,5
> poly g_4 = L[1];
> poly f_4 = L[2];
> list R4 = fareypoly(g_4,f_4); // returns numerator vs denominator
> R4;
  R4[1] = 6/5005*x^4-3/143*x^3+655/1001*x^2+371/143*x+11724/5005
  R4[2] = 1
> L = polyInterpolation(list(6),list(654/17),1,L);
> L;
  L[1] = -12/85085*x^5+282/85085*x^4-3/91*x^3+11675/17017*x^2
         +217457/85085*x+200748/85085
  L[2] = x^6-21*x^5+175*x^4-735*x^3+1624*x^2-1764*x+720
  L[3] = 1,2,3,4,5,6
> poly g_5 = L[1];
> poly f_5 = L[2];
> list R5 = fareypoly(g_5,f_5);
> R5;
  R5[1] = 1/2*x^3+9/2*x^2+17/2*x+6
  R5[2] = x+5/2
```

We see that the rational functions $R_4[1]/R_4[2]$ and $R_5[1]/R_5[2]$ do not coincide. We still need to continue updating L by adding more points until $R_{i-1}[1]/R_{i-1}[2]$ and $R_i[1]/R_i[2]$ coincide. Now for $i = 6$, we have

```
> L = polyInterpolation(list(7),list(915/19),1,L);
```

```
> L;
  L[1] = 24/1616615*x^6-732/1616615*x^5+9558/1616615*x^4-14187/323323*x^3
         +1148101/1616615*x^2+4089347/1616615*x+547356/230945
  L[2] = x^7-28*x^6+322*x^5-1960*x^4+6769*x^3-13132*x^2+13068*x-5040
  L[3] = 1,2,3,4,5,6,7
> poly g_6 = L[1];
> poly f_6 = L[2];
> list R6 = fareypoly(g_6,f_6);
> R6;
  R6[1] = 1/2*x^3+9/2*x^2+17/2*x+6
  R6[2] = x+5/2
```

The rational function stops changing at $i = 6$. In fact, the functions $R_5[1]/R_5[2]$ and $R_6[1]/R_6[2]$ are the same. Hence

$$\Gamma = \frac{1/2x^3 + 9/2x^2 + 17/2x + 6}{x + 5/2}$$

with high probability. Note that the interpolating polynomials g_5 and g_6 are not equal although the functions $R_5[1]/R_5[2]$ and $R_6[1]/R_6[2]$ coincide.

1.6.2 Early Termination in the Ben-Or/Tiwari Interpolation Algorithm

The algorithm by Ben-Or/Tiwari, Algorithm 1.3, needs to know the number of terms t, or an upper bound $\tau \geq t$. From the practical point of view, since there are, depending on the output in our application, usually two or more polynomials need to be interpolated at once, knowing this bound in advance would save a lot of computation time. For example, we would factorize then only need a single factorization step, see line 13 of Algorithm 1.4. Even if the bound τ on the number of terms is not known in advance, the early termination version of the algorithm by Ben-Or/Tiwari described in [32], which requires a single interpolation run, tackles this problem with high probability, see Algorithm 1.4. We describe, following [32], the basic idea of this strategy:

Pick a random point $p = (p_1, \ldots, p_n)$, where the p_i are distinct primes, for the evaluations $a_j = B(p_1^j, \ldots, p_n^j)$ in the Ben-Or/Tiwari algorithm, and show that with high probability the Berlekamp/Massey algorithm (see Algorithm in [31, Section 2.1]) does not encounter a singular $j \times j$ principal sub-matrix

$$A_j = \begin{pmatrix} a_0 & \cdots & a_{j-1} \\ \vdots & \ddots & \vdots \\ a_{j-1} & \cdots & a_{2(j-1)} \end{pmatrix} \tag{1.20}$$

until $j = t + 1$. This is not generally true since for any $B(x_1, \ldots, x_n) = B(X) = \sum_{k=1}^{t} c_k X^{e_k}$ that satisfies $B(p^0) = a_0 = c_1 + \ldots + c_t = 0$, the first discrepancy Δ_1 is zero. Let us now show that the first discrepancy is zero: As in the Berlekamp/Massey algorithm, we define Λ_{j-1} by

$$\Lambda_{j-1} = \lambda_0 z^t + \lambda_1 z^{t-1} + \ldots + \lambda_t$$

where $\lambda_0 \neq 0$, $t = \deg(\Lambda_{j-1})$ for $j \geq 1$, $\Lambda_0 = 1$ and $\lambda_t = 1$ for all $t \geq 0$. Since $t = \deg(\Lambda_0) = 0$, we have $\lambda_0 = 1$ for $j = 1$. Thus from (1.17), we get

$$\Delta_1 = \lambda_0 \cdot a_0 = 1 \cdot 0 = 0.$$

The above problem, nevertheless, can be fixed either by shifting the sequence by one element or by increasing the threshold for the early termination.

In the following theorem, each embedded principal sub-matrix A_j has non-zero determinant for $1 \leq j \leq t$.

Theorem 1.6.3 ([32, Theorem 4]). *The determinant of A_j is non-zero for $j = 1, \ldots, t$.*

Theorem 1.6.4 ([32, Theorem 5]). *Let S be a subset of an integral domain. If $p_1, \ldots, p_n$ are chosen randomly and uniformly from S, then for the sequence $\{a_j\}_{j \geq 0}$, where $a_j = B(p_1^j, \ldots, p_n^j)$, the Berlekamp/Massey algorithm encounters a singular A_j matrix (and the corresponding zero discrepancy) the first time at $j = t+1$ (or when the length of the sequence is equal to $2t + 1$) with probability no less than*

$$1 - \frac{t(t+1)(2t+1)\deg(B)}{6 \cdot \#(S)}$$

where $\#(S)$ is the number of elements in S.

Based on this theorem, Algorithm 1.4, the early termination version of the sparse interpolation algorithm, recovers a multivariate polynomial given in a black box with high probability.

In line 11 of Algorithm 1.4, the polynomial $\Lambda(z)$ is updated whenever the next element a_j of the sequence does not fit the current linear recursion. In line 12, instead of checking whether the matrix A_j is singular, we check whether a zero discrepancy Δ_j is detected for some j, as in (1.17).

The following example illustrates the early termination version of the sparse multivariate polynomial interpolation:

Example 1.6.5. Compute the multivariate polynomial $B(x_1, x_2, x_3) \in \mathbb{Q}[x_1, x_2, x_3]$ for the following evaluation points $a_j = B(2^j, 3^j, 5^j)$

j	1	2	3	4	5	6	7
a_{j-1}	1	8	64	512	4096	32768	262144

Algorithm 1.4 Early Termination Sparse Interpolation (`sparseInterpolation`)

Input: $B(x_1, \ldots, x_n)$: a multivariate black box polynomial.

η: a positive integer (default 1), the threshold for early termination.

Output: $c_i \neq 0$ and B_i such that $B(x_1, \ldots, x_n) = \sum_{i=1}^{t} c_i B_i$ with high probability, or an error message if the procedure fails to complete.

[The early termination within the BMA]

1: pick random distinct prime numbers $p_1, \ldots, p_n$

2: $j \longleftarrow 1$

3: $a_0 \longleftarrow B(1, \ldots, 1)$

4: **while** (TRUE) **do**

5: $\quad j \longleftarrow j + 1$

6: $\quad t \longleftarrow j - 1$

7: $\quad i \longleftarrow 2 \cdot j - 2$

8: $\quad$ **for** $k = i - 1, \ldots, i$ **do**

9: $\quad\quad a_k \longleftarrow B(p_1^k, \ldots, p_n^k)$

10: $\quad A_j \longleftarrow \begin{pmatrix} a_0 & \cdots & a_{j-1} \\ \vdots & \ddots & \vdots \\ a_{j-1} & \cdots & a_{2(j-1)} \end{pmatrix}$

11: $\quad$ let $\Lambda(z)$ be the polynomial obtained by performing the BMA on $\{a_k\}_{0 \leq k \leq i}$

12: $\quad$ **if** the matrix A_j is singular η times in a row **then**

13: $\quad\quad$ compute all the roots b_i of $\Lambda(z)$ in the domain of $p_1, \ldots, p_n$

14: $\quad\quad$ **if** the number of the roots b_i of $\Lambda(z)$ is equal to $\deg(\Lambda)$ **then**

[the early termination was correct]

15: $\quad\quad\quad$ break out of the loop

[Determine B_i]

16: repeatedly divide the b_i's by $p_1, \ldots, p_n$ to obtain the exponents $\alpha_{i1}, \ldots, \alpha_{in} \in \mathbb{Z}_{\geq 0}$ such that $B_i = x_1^{\alpha_{i1}} \cdots x_n^{\alpha_{in}}$

[Compute the coefficients c_i]

17: solve the linear system $a_j = \sum_{i=1}^{t} c_i b_i^j \quad$ with $0 \leq j \leq t - 1$

18: **return** c_i and B_i

Consider the embedded principal sub-matrices

$$A_1 = \begin{pmatrix} 1 \end{pmatrix}, \; A_2 = \begin{pmatrix} 1 & 8 \\ 8 & 64 \end{pmatrix}, \; A_3 = \begin{pmatrix} 1 & 8 & 64 \\ 8 & 64 & 512 \\ 64 & 512 & 4096 \end{pmatrix}, \ldots.$$

Since $\det(A_2) = 0$, the number of terms t of the target polynomial is equal to 1 (see Theorem 1.6.3) and, hence, the length of the sequence required to recover this polynomial is equal to $2t + 1 = 3$, see Theorem 1.6.4. Thus, to recover the multivariate polynomial B in $\mathbb{Q}[x_1, x_2, x_3]$, it suffices to take the sequence $1, 8, 64$. The auxiliary polynomial Λ is of degree 1 since $t = 1$. In fact, it is easy to see that $\Lambda(z) = z - 8$ and its integer root is equal to 8. Using the SINGULAR command `sparseInterpolation` from the SINGULAR library `ffmodstd.lib` [6], we compute the polynomial B as follows:

```
> ring r = 0, (x1,x2,x3),dp;
> list L = 1, 8, 64, 512, 4096, 32768, 262144;
> list T = BerlekampMassey(L,1); // 1 refers to the position
                                 // of the variable in Lambda
> poly Lambda = T[1]; // only if size(T)=2, otherwise we use T
                    // for adding more evaluation points to determine Lambda
> Lambda; // the minimal polynomial generating the sequence L = 1,8,64
  x1-8
> T[2]; // the max. length of L required to recover the polynomial B
  2
> list pr = 2,3,5; // list of primes where pr[i] -> x_i
> poly B = sparseInterpolation(Lambda,L,pr,0,0); // The fourth
                    // argument refers to the position of the auxiliary
                    // variable z. In this case, it is 0 since z is not
                    // defined here. The fifth argument refers to the
                    // exponent of the primes that the evaluation
                    // started with, that is, to j-1. In this case,
                    // we have j-1=0 and a_0 = B(1,1,1).
> B; // the desired polynomial
  x1^3
```

Chapter 2

Gröbner Bases over Algebraic Number Fields

From the theoretical point of view, Gröbner bases computations can be done over any field by using Buchberger's algorithm (see, for example, [1, 13, 25]). In particular, they can be performed over an algebraic number field, but the computation is often inefficient if the arithmetic operations in this field are used directly. Consider a simple extension $K = \mathbb{Q}(\alpha)$ of $\mathbb{Q}$. Let $f \in \mathbb{Q}[t]$ be the minimal polynomial of α. The algebraic number field K can be represented as the residue class ring $\mathbb{Q}[t]/\langle f \rangle$, and a Gröbner basis computation over K can then be reduced to one over $\mathbb{Q}$ by joining f to the ideal to be considered. Unfortunately, this method is not satisfactory in view of efficiency. One of the reasons for this is that over the field of rational numbers, we often suffer from coefficient swell. Various methods to avoid this have been investigated; the trace algorithm [40] and modular algorithms [2, 27] are successful in this direction. But using these approaches, we still have to deal with the complicated arithmetic in algebraic number fields, in particular with the computation of inverses.

In this chapter we present a new efficient method to compute Gröbner bases over an algebraic number field. Starting from a polynomial ring over $\mathbb{Q}$ as explained above, we apply the modular methods for computing Gröbner bases discussed in [2, 4, 27] to pass to positive characteristic p. Choosing a set $\mathcal{P}$ of *suitable* prime numbers, see Definition 2.4.2, the image f_p of f in $\mathbb{F}_p[t]$ is, for $p \in \mathcal{P}$, reducible and squarefree. We can thus again apply modular methods, with respect to the factors $f_{1,p}, \ldots, f_{r_p,p}$ of f_p, passing to the rings $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$. As above, we avoid computing in quotient rings by joining $f_{i,p}$ to the ideal to be considered. Having computed the corresponding reduced Gröbner basis for each of these factors, we first recombine the results to a set of polynomials G_p over $\mathbb{F}_p[t]/\langle f_p \rangle$ using Chinese remaindering for polynomials. In a second lifting step, the sets G_p, $p \in \mathcal{P}$, are then used to reconstruct a set of polynomials G over $\mathbb{Q}$, via Chinese remaindering for integers and rational reconstruction. Finally, we verify whether the input ideal is contained in $\langle G \rangle$ and G is the reduced Gröbner bases of the ideal it generates. If not, we enlarge $\mathcal{P}$ and repeat the process.

In Section 2.1, we introduce some notation which is used throughout this chapter. The structure of the new method is outlined in Section 2.2. Since this method relies on the Chinese remainder algorithm applied to different domains, we shortly recall the

relevant theoretical background in Section 2.3. The core part of the proposed algorithm is discussed in Section 2.4. Here we explain how modular methods are applied on different levels and why our approach is considerably faster than other known methods. The application of modular methods follows a well-known scheme, see [4]. For reference, we recall the relevant parts of this scheme in Section 2.5. An illustrating example is given in Section 2.6. Finally, Section 2.7 contains remarks on the implementation of the new method in SINGULAR [17] and timings comparing it to other approaches. The benchmark problems which we used for the timings are listed in the appendix, see Section A.1.

2.1 Notation

Let $K = \mathbb{Q}(\alpha)$ be an algebraic number field and let $f \in \mathbb{Q}[t]$ be the minimal polynomial of the algebraic number α. Then every element of K can be written as a linear combination of elements in $\{1, \alpha, \alpha^2, \ldots, \alpha^{d-1}\}$ where $d = \deg f$. Hence we may regard every element of K as a polynomial in α with coefficients in $\mathbb{Q}$. Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and let t be an extra variable. Consider the polynomial rings $S = \mathbb{Q}(\alpha)[X]$, $T = \mathbb{Q}[X, t]$, and $\mathbb{Q}[t]$. Fix a global monomial ordering $>_1$ on the monoid of monomials $\mathrm{Mon}(X)$ and consider the product ordering $>_K := (>_1, >)$ on $\mathrm{Mon}(X, t)$, where $>$ is the global ordering on $\mathrm{Mon}(t)$. Note that this implies $X^a >_K t^b$ for all $a \in \mathbb{N}^n \setminus \{(0, \ldots, 0)\}$ and $b \in \mathbb{N}$.

Let $\widetilde{H} = \{g_1(X, t), \ldots, g_s(X, t)\}$ be a subset of T, let $I \subseteq S$ be the ideal generated by $H := \{g_1(X, \alpha), \ldots, g_s(X, \alpha)\}$, and let $\widetilde{I} \subseteq T$ be the ideal generated by $\widetilde{H} \cup \{f\}$. Furthermore, let $\widetilde{G} \subseteq T$ be the reduced Gröbner basis of $\widetilde{I}$ w.r.t. $>_K$. Let φ be the canonical homomorphism from T to S which leaves the x_i fixed and maps t to α. We will show, in Theorem 2.4.1, that the non-zero elements of $\varphi(\widetilde{G}) \subseteq S$ form the reduced Gröbner basis of I w.r.t. $>_1$.

Let p be a prime number, and let $Z_{\langle p \rangle} \subseteq \mathbb{Q}$ be the set of all rational numbers (in lowest terms) such that none of the denominators of the elements in $Z_{\langle p \rangle}$ vanishes modulo p. That is,

$$Z_{\langle p \rangle} = \left\{ \frac{a}{b} \mid a \in \mathbb{Z},\ b \in \mathbb{Z} \setminus p\mathbb{Z} \right\} \subseteq \mathbb{Q}\,.$$

Consider the map from $Z_{\langle p \rangle}$ to $\mathbb{F}_p$ which sends $\frac{a}{b}$ to $ab^{-1} \in \mathbb{F}_p$. If p does not divide any denominator of the coefficients of f and $g_1(X, t), \ldots, g_s(X, t)$, we apply this map to the coefficients of these polynomials. We then write $f_p := (f \bmod p) \in \mathbb{F}_p[t]$ and $\widetilde{I}_p := \langle g_1(X, t)_p, \ldots, g_s(X, t)_p, f_p \rangle \subseteq \mathbb{F}_p[X, t]$.

2.2 Structure of the New Method

Noro [35] has presented a modified version of Buchberger's algorithm which computes Gröbner bases over an algebraic number field using the arithmetic in $\mathbb{Q}[t]/\langle f \rangle$. Instead of computing in the ring $(\mathbb{Q}[t]/\langle f \rangle)[X]$, one might as well add the minimal polynomial f to the ideal to be considered and work over $\mathbb{Q}[X, t]$, see Theorem 2.4.1. In this situation,

the elements of a reduced Gröbner basis are, except f itself, all monic in $(\mathbb{Q}[t])[X]$, that is, they are of the form $X^a + (\text{lower terms})$, see the proof of Theorem 2.4.1. Noro noticed that during the execution of Buchberger's algorithm, many (superfluous) intermediate basis elements of the form $t^b X^a + (\text{lower terms})$ are computed before a monic element $X^a + (\text{lower terms})$ is generated. Of course, each additional basis element produces new S-pairs which usually make the subsequent computation inefficient. Noro has resolved this problem by making each generated basis element monic in $(\mathbb{Q}[t])[X]$ before it is added to the basis. For this, the inverse of an algebraic number has to be computed which is in general computationally expensive. Instead, we use a different approach to reduce the number of basis elements which are computed before a monic element $X^a + (\text{lower terms})$ is generated.

The new method computes the reduced Gröbner basis of the input ideal in three steps: In the first step, for a suitable prime p such that $f_p \in \mathbb{F}_p[t]$ is reducible and squarefree, see Definition 2.4.2, we compute the reduced Gröbner basis $\widetilde{G}_p$ of $\widetilde{I}_p$ over $\mathbb{F}_p$ w.r.t. $>_K$, as follows: Let $f_p = \prod_{1 \le i \le r_p} f_{i,p}$ be the irreducible factorization of f_p over $\mathbb{F}_p$, with $r_p > 1$. Set $\widetilde{I}_{i,p} := \langle \widetilde{H}_p \cup \{f_{i,p}\} \rangle \subseteq \mathbb{F}_p[X,t]$. For each $i \in \{1,\ldots,r_p\}$, we compute the reduced Gröbner basis $\widetilde{G}_{i,p}$ of $\widetilde{I}_{i,p}$. Using the Chinese remainder algorithm for polynomials (see Algorithm 2.5 below), we determine a set of polynomials $\widetilde{G}_p \equiv \left(\widetilde{G}_{i,p} \setminus \{f_{i,p}\}\right) \mod f_{i,p}$ which together with f_p is the reduced Gröbner basis of $\widetilde{I}_p$ with high probability (see Remark 2.4.6). Note that, at this step of the algorithm, computing modulo the different factors of the minimal polynomial $f_{i,p}$ (by adding them to the ideal $\langle \widetilde{H}_p \rangle$) is, from the theoretical point of view, just the same as computing modulo several prime numbers, see Section 2.3.

In the second step, following [2, 27], we use the Chinese remainder algorithm for integers together with rational reconstruction to lift these results to a set $\widetilde{G}$ which is the reduced Gröbner basis of $\widetilde{I}$ with high probability. In the last step, we lift $\widetilde{G}$ to a Gröbner basis G of I over K by mapping t to α (see Theorem 2.4.1).

The idea of the algorithm is based on the concepts of modular methods and univariate polynomial factorization over finite fields. For the former we need the Chinese remainder theorem.

2.3 Factorization and the Chinese Remainder Algorithm

The well-known Chinese remainder theorem is essential for our algorithm.

Theorem 2.3.1 ([42, Corollary 5.3]). *Let R be a Euclidean domain and let $m_1, \ldots, m_r \in R$ be coprime elements so that $\gcd(m_i, m_j) = 1$ for $0 \le i < j \le r$. Let $m = m_1 \cdots m_r$ be the product of these elements. Then $R/\langle m \rangle$ is isomorphic to the product*

ring $R/\langle m_1\rangle \times \ldots \times R/\langle m_r\rangle$ *via the isomorphism*

$$\begin{aligned} R/\langle m\rangle &\to R/\langle m_1\rangle \times \ldots \times R/\langle m_r\rangle\,, \\ a \bmod m &\mapsto (a \bmod m_1, \ldots, a \bmod m_r)\,. \end{aligned}$$

For our purpose, we need this theorem in the following two incarnations.

Corollary 2.3.2. *Let* $p_1, \ldots, p_k$ *be prime numbers, and let* $N = p_1 \cdots p_k$ *be their product. Then we have the following isomorphism:*

$$\begin{aligned} \mathbb{Z}/\langle N\rangle &\cong \mathbb{F}_{p_1} \times \ldots \times \mathbb{F}_{p_k}\,, \\ a \bmod N &\longmapsto (a \bmod p_1, \ldots, a \bmod p_k)\,. \end{aligned}$$

The second application of the Chinese remainder theorem refers to univariate polynomial rings over finite fields.

Corollary 2.3.3. *Let* $f_{1,p}, \ldots, f_{r_p,p} \in \mathbb{F}_p[t]$ *be pairwise coprime polynomials, and let* $f_p = f_{1,p} \cdots f_{r_p,p}$ *be their product. Then we have the ring isomorphism*

$$\begin{aligned} \mathbb{F}_p[t]/\langle f_p\rangle &\cong \mathbb{F}_p[t]/\langle f_{1,p}\rangle \times \ldots \times \mathbb{F}_p[t]/\langle f_{r_p,p}\rangle\,, \\ g \bmod f_p &\longmapsto (g \bmod f_{1,p} \ldots, g \bmod f_{r_p,p})\,. \end{aligned}$$

The proof of Theorem 2.3.1 is constructive (see [42, Theorem 5.2, Corollary 5.3]) and yields the Chinese remainder algorithm. For reference, we state it here in the form of Corollary 2.3.3, see Algorithm 2.5.

Remark 2.3.4.

(1) Since $c_i h_i \equiv 0 \mod f_{j,p}$ for $j \neq i$ and $c_i h_i \equiv q_i s_i h_i \equiv q_i \mod f_{i,p}$, we have

$$g \equiv c_i h_i \equiv q_i \mod f_{i,p}\,.$$

Hence, the algorithm works correctly.

(2) Although stated here for $\mathbb{F}_p[t]$, Algorithm 2.5 works for polynomial rings over any ground field.

(3) Instead of $q_1, \ldots, q_{r_p} \in \mathbb{F}_p[t]$, Algorithm 2.5 can also be applied coefficient-wise to polynomials and vectors with coefficients in $\mathbb{F}_p[t]$.

Algorithm 2.5 Chinese Remainder Algorithm (CRA) for polynomials

Input: $q_1, \ldots, q_{r_p} \in \mathbb{F}_p[t]$, $f_{1,p}, \ldots, f_{r_p,p} \in \mathbb{F}_p[t]$ pairwise coprime.
Output: $g \in \mathbb{F}_p[t]$ such that $g \equiv q_i \mod f_{i,p}$ for $1 \leq i \leq r_p$.

1: $g \longleftarrow 0$
2: $f_p \longleftarrow \prod_{1 \leq i \leq r_p} f_{i,p}$
3: **for** $i = 1, \ldots, r_p$ **do**
4: $\quad h_i \longleftarrow \dfrac{f_p}{f_{i,p}}$
5: $\quad$ by the Extended Euclidean Algorithm [42, Algorithm 3.14], compute $s_i, t_i \in \mathbb{F}_p[t]$ such that
$$s_i h_i + t_i f_{i,p} = 1$$
6: $\quad c_i \longleftarrow \mathrm{NF}(q_i s_i, f_{i,p})$
$\quad$ (c_i is the remainder in $\mathbb{F}_p[t]$ on dividing $q_i s_i$ by $f_{i,p}$)
7: $\quad g \longleftarrow g + c_i h_i$
8: **return** g

2.4 Gröbner Bases using Factorization and Modular Methods

As Noro does (see [35, Theorem 1]), we rely on the following result whose proof we give for the lack of reference.

Theorem 2.4.1. *Let $\widetilde{G}$ be the reduced Gröbner basis of $\widetilde{I}$ w.r.t. $>_K$. Then $(\widetilde{G} \setminus \{f\})|_{t=\alpha}$ is the reduced Gröbner basis of I w.r.t. $>_1$.*

Consider the ring homomorphism

$$\begin{aligned} \varphi : T &\longrightarrow S\,, \\ t &\longmapsto \alpha\,, \\ x_i &\longmapsto x_i\,. \end{aligned}$$

Since φ is the identity map on $\mathbb{Q}[X]$, we get an isomorphism

$$S \cong T/\langle f \rangle\,.$$

Clearly, $\varphi(\widetilde{I}) = I$. We are now ready to prove Theorem 2.4.1.

Proof. Without loss of generality, we may assume that $\widetilde{I} \neq \langle 1 \rangle$. Let

$$\widetilde{G} = \{m_1(X,t), \ldots, m_a(X,t), m_{a+1}(X,t)\}$$

be the reduced Gröbner basis of $\widetilde{I}$. We first prove that $f \in \widetilde{G}$. Suppose $f \notin \widetilde{G}$. Then there exists a non-zero non-constant polynomial $f' \in \widetilde{G} \cap \mathbb{Q}[t]$ with $\deg f' < \deg f$. Hence

$$I = \varphi(\widetilde{I}) = \langle \varphi(f'), \varphi(\widetilde{G} \setminus \{f'\}) \rangle = \langle 1 \rangle$$

since $\varphi(f')$ is invertible in S. This implies $\widetilde{I} = \langle 1 \rangle$, a contradiction. So, $f = m_i(X,t)$ for some i, say $i = a+1$. Then we have

$$\begin{aligned} \varphi(\widetilde{G} \setminus \{f\}) &= \{m_1(X,\alpha), \ldots, m_a(X,\alpha)\} \\ &= (\widetilde{G} \setminus \{f\})|_{t=\alpha} =: G\,. \end{aligned}$$

The result follows easily once we show that the leading coefficient of $m(X,t)$, considered as an element in the polynomial ring $\mathbb{Q}[t]$, is equal to 1 for all $m(X,t) \in \widetilde{G} \setminus \{f\}$. To prove this statement, suppose there is an index $1 \leq j \leq a$ such that $\mathrm{lt}(m_j(X,t)) = c \cdot X^\delta$ with $c \in \mathbb{Q}[t]$ and $\deg c > 0$. Clearly, c is monic. Write

$$m_j(X,t) = c \cdot X^\delta + V(X,t)$$

where $V(X,t) = \mathrm{tail}(m_j(X,t))$, which implies that $V(X,t)$ does not contain any term divisible by X^δ. We have $\deg c < \deg f$ and therefore $\gcd(c,f) = 1$ since f is irreducible. Thus, by the extended Euclidean algorithm (see [42, Algorithm 3.14]), there exist $a, b \in \mathbb{Q}[t]$ such that $a \cdot c + b \cdot f = 1$. Considering the polynomial $a \cdot m_j(X,t) + b \cdot f \cdot X^\delta$, we have

$$\begin{aligned} \langle \widetilde{G} \rangle \ni\ & a \cdot m_j(X,t) + b \cdot f \cdot X^\delta \\ &= (a \cdot c + b \cdot f) \cdot X^\delta + a \cdot V(X,t) \\ &= X^\delta + a \cdot V(X,t) \\ &=: F(X,t)\,. \end{aligned}$$

But $\mathrm{lt}(F(X,t)) = X^\delta$ divides $c \cdot X^\delta = \mathrm{lt}(m_j(X,t))$ which is a contradiction to the choice of $\widetilde{G}$. □

The notion of primes which are *admissible of type* A w.r.t. a monic irreducible polynomial, which is essential for our algorithm, is defined as follows:

Definition 2.4.2. Let $f \in \mathbb{Q}[t]$ be as given above. Let p be a prime not dividing any numerator or any denominator of the coefficients occurring in f. We say that p is *admissible of type* A w.r.t. f if f_p is reducible and squarefree over $\mathbb{F}_p$. In this case, we write f_p as $f_p = \prod_{1 \leq i \leq r_p} f_{i,p}$.

For a non-zero polynomial $g \in T$ considered as a polynomial in X over $\mathbb{Q}[t]$, that is, $g \in (\mathbb{Q}[t])[X]$, let S_g be the set of all distinct coefficients (in $\mathbb{Q}[t]$) of g of degree greater than or equal to 1. That is,

$$S_g = \left\{ \mathrm{lc}_{\mathbb{Q}[t]}(u) \mid u \text{ is a term of } g \text{ with } \deg(\mathrm{lc}_{\mathbb{Q}[t]}(u)) \geq 1 \right\} .$$

With notation as above, the notion of primes which are *admissible of type* B w.r.t. a monic irreducible polynomial and a set of polynomials is defined as follows:

Definition 2.4.3 (Weak version)**.** Let $\widetilde{H} = \{g_1(X,t), \ldots, g_s(X,t)\}$ be as given above. Let p be a prime not dividing any numerator or any denominator of the coefficients occurring in $\widetilde{H}$. We say that p is *admissible of type* B w.r.t. f and $\widetilde{H}$ if p is admissible of type A w.r.t. f and if, for each g in $\widetilde{H}$, none of the elements in S_g is divisible by any of the factors of f_p over $\mathbb{F}_p$.

To see the relevance of this definition, consider the ideal

$$J = \langle x^2 + xy + t,\ x + y + t - 1 \rangle =: \langle h_1, h_2 \rangle$$

and the minimal polynomial $f = t^3 + t + 1$. If $p = 3$, then

$$f_p \equiv (t-1)(t^2 + t - 1) =: f_{1,p} \cdot f_{2,p} \mod p$$

and, using the degree reverse lexicographic ordering with $x > y$, the reduced Gröbner bases of the ideals $J + \langle f_{1,p} \rangle$ and $J + \langle f_{2,p} \rangle$ are

$$\{1\} \text{ and } \{t^2 + t - 1,\ y + 1,\ x + t + 1\},$$

respectively. In this case, Algorithm 2.5 cannot be applied since the sizes of these sets do not fit. The calculation suggests that the reason for this is that the element $t - 1 \in S_{h_2}$ vanishes when reduced w.r.t. the set $\{t - 1,\ t^2 + t - 1\}$.

Next, consider the ideal

$$J' = \langle x^2 + xy + t, t^2 x + y \rangle =: \langle g_1, g_2 \rangle.$$

Here, the reduced Gröbner bases of the ideals $J' + \langle f_{1,p} \rangle$ and $J' + \langle f_{2,p} \rangle$ are

$$\{1\} \text{ and } \{t^2 + t - 1,\ x + yt - t,\ y^2 - 1\},$$

respectively. Again the sizes of these sets do not coincide, hence, we still cannot apply Algorithm 2.5. Moreover, none of the coefficients in S_{g_1} and S_{g_2} is divisible by either $f_{1,p}$ or $f_{2,p}$ which shows that the condition in Definition 2.4.3 is not sufficient. Indeed, the element $t^2 \in S_{g_2}$ vanishes when reduced w.r.t. the set $\{t^2 + t - 1,\ t - 1\}$. Therefore, we may impose a stronger condition by saying that for all $g \in \widetilde{H}$ none of the elements in S_g vanishes when reduced w.r.t. the set $\{f_{1,p}, \ldots, f_{r_p,p}\}$ (in some order) and thus reduce the probability that the reconstruction fails. In the following example we see that this condition is still not sufficient.

Consider the ideal

$$J'' = \langle x^2 + xy + t,\ tx + y + t \rangle =: \langle k_1, k_2 \rangle.$$

The reduced Gröbner bases of the ideals $J'' + \langle f_{1,p} \rangle$ and $J'' + \langle f_{2,p} \rangle$ are

$$\{t - 1,\ x - 1,\ y - 1\} \text{ and } \{t^2 + t - 1,\ x + yt - y + t + 1,\ y^2 + yt + y + t - 1\},$$

respectively. Although none of the elements in S_{k_1} and S_{k_2} vanishes when reduced w.r.t. the set $\{t^2 + t - 1,\ t - 1\}$, and the sizes of these sets coincide, we see that applying Algorithm 2.5 yields

$$\{t^2 - t + 1,\ x - 1,\ y^2t^2 + y^2t - y^2 + yt^2 + yt + t^2 + t + 1\}$$

which is not the desired result because the reduced Gröbner basis of $J'' + \langle f_p \rangle$ is

$$\{t^2 + t - 1,\ y + 1,\ x + t + 1\}.$$

In practice, however, it is very unlikely that this case happens. It is, nevertheless, important to address this problem. A possible way to handle this difficulty is to refine Definition 2.4.3 as follows:

Definition 2.4.4 (Strong version)**.** Let f and $\widetilde{H} = \{g_1(X,t), \ldots, g_s(X,t)\}$ be as given above. Let p be a prime which is admissible of type A w.r.t. f, and write $f = f_{1,p} \cdots f_{r_p,p}$ as in Definition 2.4.2. Suppose that p does not divide any numerator or any denominator of the coefficients occurring in $\widetilde{H}$. For $i = 1, \ldots, r_p$, set $\widetilde{I}_{i,p} := \langle \widetilde{H}_p \cup \{f_{i,p}\} \rangle$, and let $\widetilde{G}_{i,p}$ be the reduced Gröbner basis of the ideal $\widetilde{I}_{i,p}$. We say that p is *admissible of type* B w.r.t. f and $\widetilde{H}$ if for all indices i, j with $i \neq j$

1. the sizes of $\widetilde{G}_{i,p}$ and $\widetilde{G}_{j,p}$ coincide, and
2. $\mathrm{Lm}(\widetilde{G}_{i,p} \setminus \{f_{i,p}\}) = \mathrm{Lm}(\widetilde{G}_{j,p} \setminus \{f_{j,p}\})$.

In the above examples, the prime number 3 is not admissible of type B w.r.t. $t^3 + t + 1$ and the generators of each of the ideals J, J' and J'' in the sense of Definition 2.4.4. This is because in the first two cases, both conditions of this definition are violated whereas in the third case, the second condition is not satisfied. For the rest of our discussion we use the strong version of this definition.

We now turn our attention to the notion of *lucky primes*:

Definition 2.4.5 ([27])**.** Let $\widetilde{I}$ be an ideal given as above and let p be a prime number. Furthermore, let $\widetilde{G}$ be the reduced Gröbner basis of $\widetilde{I}$ and let $\widetilde{G}_p$ be the reduced Gröbner basis of $\widetilde{I}_p$. Then p is called *lucky* for $\widetilde{I}$ if and only if $\mathrm{Lm}(\widetilde{G}_p) = \mathrm{Lm}(\widetilde{G})$. Otherwise p is called *unlucky* for $\widetilde{I}$.

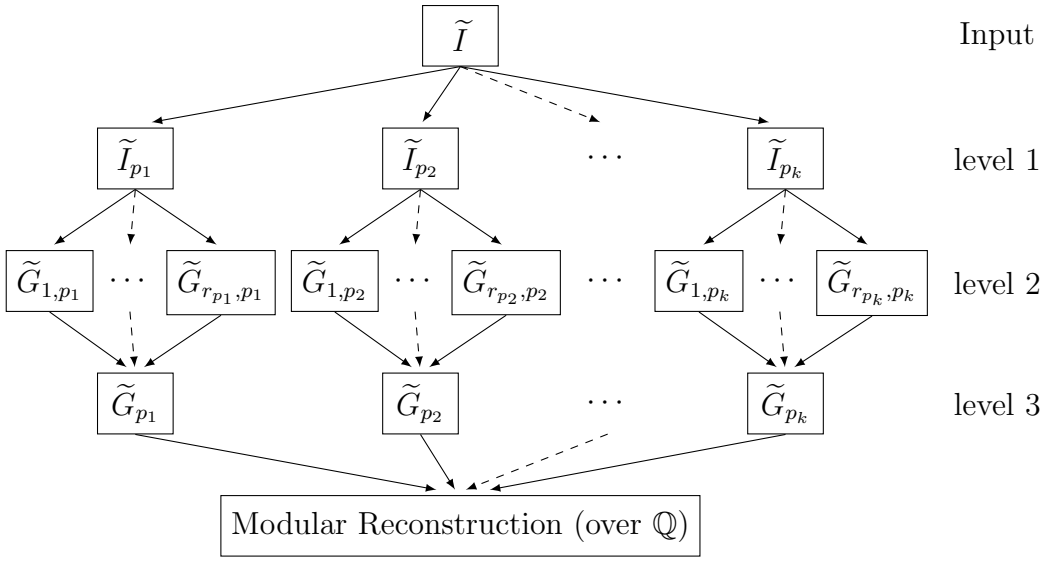

Figure 2.1: General scheme for the new algorithm

Since f is independent of X, we get, by Corollary 2.3.3, the isomorphism

$$\mathbb{F}_p[X,t]/\langle f_p\rangle \cong \mathbb{F}_p[X,t]/\langle f_{1,p}\rangle \times \ldots \times \mathbb{F}_p[X,t]/\langle f_{r_p,p}\rangle\,.$$

Remark 2.4.6. Let $\widetilde{I}$, $\widetilde{H}$, and f be as above. Let p be a prime which is both admissible of type B w.r.t. f and $\widetilde{H}$ as well as lucky for $\widetilde{I}$. We work over $\mathbb{F}_p[X,t]$ equipped with the product ordering $>_K$. Suppose a set of polynomials $\widetilde{G}_p$ is the reduced Gröbner basis of the ideal $\widetilde{I}_p$. For $i = 1,\ldots,r_p$, set $S_i := (\widetilde{G}_p \setminus \{f_p\}) \bmod f_{i,p} \subseteq \mathbb{F}_p[X,t]/\langle f_{i,p}\rangle$. Then for each i, the set $S_i \cup \{f_{i,p}\}$ is the reduced Gröbner basis of the ideal $\widetilde{I}_{i,p}$ (as in Definition 2.4.4) with high probability. Conversely, let $\widetilde{G}_{i,p}$ be the reduced Gröbner basis of $\widetilde{I}_{i,p}$. Let $\widetilde{G}'_p$ be the set of polynomials that is obtained by applying Algorithm 2.5 coefficient-wise to the input

$$\Big((\widetilde{G}_{1,p} \setminus \{f_{1,p}\},\ldots,\widetilde{G}_{r_p,p} \setminus \{f_{r_p,p}\}),\,(f_{1,p},\ldots,f_{r_p,p})\Big)\,.$$

Then the set $\widetilde{G}'_p \cup \{f_p\}$ is the reduced Gröbner basis of the ideal $\widetilde{I}_p$ with high probability. Hence, we have $\widetilde{G}'_p \cup \{f_p\} = \widetilde{G}_p$ with high probability.

The main innovation of our new algorithm, which is illustrated in Figure 2.1, is as follows: Instead of computing the reduced Gröbner bases at level 1, our algorithm computes them at level 2. For the primes satisfying the conditions in Definition 2.4.4 (and only for those), the Chinese remainder algorithm for polynomials then combines these results at level 3. The ideals $\langle \widetilde{G}_{p_i}\rangle$ at this level are expected to be the same as the ideals $\widetilde{I}_{p_i}$ at level 1 with high probability (see Remark 2.4.6). The remaining parts of the computation are carried out in the same way as in the modular algorithms described in [27].

Now we give a brief description of the new algorithm. In the beginning, randomly choose a set $\mathcal{P}$ of prime numbers which are admissible of type A w.r.t. f. At level 2, given a prime $p \in \mathcal{P}$, factorize $f \in \mathbb{Q}[t]$ over $\mathbb{F}_p$ and compute, for each i, the reduced Gröbner basis $\widetilde{G}_{i,p}$ of the ideal $\widetilde{I}_{i,p}$ corresponding to the i-th factor. If the prime p is admissible of type B w.r.t. f and $\widetilde{H}$, then lift these results via Chinese remaindering for polynomials (at level 3) to obtain the reduced Gröbner basis $\widetilde{G}_p$ of $\widetilde{I}_p$ with high probability. Repeat this process for every prime $p \in \mathcal{P}$ which is admissible of type B, in the same way as in the modular algorithms in [27].

The main reason why the method to compute Gröbner bases over algebraic number fields described above is faster than other known methods, see Section 2.7, is that factorizing the minimal polynomial f in positive characteristic allows us to compute in rings with minimal polynomials of degree much less than $\deg f$: Experiments have shown that the performance of Gröbner basis computations over simple algebraic extensions depends heavily on the degree of the minimal polynomial. Additionally, the computations are carried out over finite fields which avoids the problem known as coefficient swell, and we do not directly use the computationally expensive arithmetic in K. Finally, the new method is a priori easily parallelizable.

2.5 Modular Algorithms

To compute the reduced Gröbner basis of the ideal $\widetilde{I}$, the modular algorithm described in [27] first chooses a set of primes $\mathcal{P}$ and computes the reduced Gröbner basis $\widetilde{G}_p$ of $\widetilde{I}_p$ for each $p \in \mathcal{P}$. It then uses the Chinese remainder algorithm and rational reconstruction to obtain the reduced Gröbner basis $\widetilde{G}$ over $\mathbb{Q}$ with high probability. Finally, it verifies whether $\widetilde{I}$ is contained in $\langle \widetilde{G} \rangle$ and $\widetilde{G}$ is the reduced Gröbner bases of the ideal it generates. One of the problems after computing the set of reduced Gröbner bases $\mathcal{GP} := \{\widetilde{G}_p \mid p \in \mathcal{P}\}$ is that $\mathcal{P}$ may contain unlucky primes. To deal with such unlucky primes, the following method is used, see [4]:

DeleteUnluckyPrimesSB ([27]): *We define an equivalence relation on* $(\mathcal{GP}, \mathcal{P})$ *by*

$$(\widetilde{G}_p, p) \sim (\widetilde{G}_q, q) :\Longleftrightarrow \mathrm{Lm}(\widetilde{G}_p) = \mathrm{Lm}(\widetilde{G}_q).$$

Then the equivalence class of largest cardinality[1] *is stored in* $(\mathcal{GP}, \mathcal{P})$*, the others are deleted.*

Now, all $\widetilde{G}_p$, $p \in \mathcal{P}$, have the same set of leading monomials. Hence, we can apply the Chinese remainder algorithm for integers and the rational reconstruction algorithm to the coefficients of the Gröbner bases in $\mathcal{GP}$ to obtain a reduced Gröbner basis $\widetilde{G}$ of $\widetilde{I}$ with high probability. Since we cannot check, however, whether $\mathcal{P}$ is sufficiently large, a

[1] Here, we have to use a weighted cardinality count if Algorithm 2.6 requires more than one round of the loop, see [4, Remark 5.7].

final verification step is needed. Since this may be expensive, especially if $\widetilde{I} \neq \langle \widetilde{G} \rangle$, we first perform a test in positive characteristic:

pTestSB ([27]): *We randomly choose a prime $p \notin \mathcal{P}$ which is admissible of type* B *w.r.t. f and $\widetilde{H}$. We test if including this prime in the set $\mathcal{P}$ would improve the result. That is, explicitly test whether $\widetilde{I}$ reduces to zero w.r.t $\widetilde{G}$ mapped to $\mathbb{F}_p[X,t]$, and vice-versa, whether $\widetilde{G}$ mapped to $\mathbb{F}_p[X,t]$ reduces to zero w.r.t. $\widetilde{G}_p$.*

The advantage of this test is that it accelerates the algorithm enormously. Algorithm 2.6 is a modified version of Algorithm 1 in [27] (which is implemented in Singular [17] in the library `modstd.lib` [26]), in the sense that we do apply modular methods not only once, but twice, where the second application is with respect to the factors of the minimal polynomial f.

Now, taking Theorem 2.4.1 into account, we can compute a Gröbner basis of an ideal in $K[X] = \mathbb{Q}(\alpha)[X]$ as in Algorithm 2.7: We first map α to t and join the minimal polynomial $f \in \mathbb{Q}[t]$ to the ideal to be considered. Then, after applying Algorithm 2.6, we only need to map t back to α to get a Gröbner basis of the input ideal.

Algorithm 2.6 is probabilistic in the sense that the test in lines 17 to 19 does not guarantee that $\langle \widetilde{G} \rangle = \widetilde{I}$. If I is homogeneous, however, the result G of Algorithm 2.7 can be verified along the lines of [2, Theorem 7.1]. With this test included, Algorithm 2.7 is deterministic.

Remark 2.5.1. Some parts of Algorithm 2.6 are inherently parallelizable. In the current implementation, see Section 2.7, we could easily take advantage of this thanks to Singular's parallel framework. We have, first of all, parallelized the for-loop starting in line 4. This corresponds to the modular computations on level 1, see Figure 2.1. Besides this, we also make use of parallelization for the selection of primes in line 1, for the application of the Farey rational map in line 16, and for the final test in line 18. The for-loop starting in line 6, which corresponds to the modular computations on level 2, is inherently parallelizable as well, but experiments have shown that a parallel implementation of this step does not yield any further speedup for our test cases.

2.6 Example

The following example illustrates how the new algorithm works:

Consider the ideal

$$I = \langle x^2 + ay, axy - x + a \rangle \subset \mathbb{Q}(a)[x,y]$$

where a is a zero of the polynomial $f = t^2 + 1 \in \mathbb{Q}[t]$. A Singular computation shows that the reduced Gröbner basis of I with respect to the degree reverse lexicographical ordering (`dp` in Singular) with $x > y$ is

$$\{y^2 + ax + ay,\ xy + ax + 1,\ x^2 + ay\}\,.$$

Algorithm 2.6 Modified modular Gröbner bases algorithm over $\mathbb{Q}$

Input: An ideal $\widetilde{I} = \langle \widetilde{H}, f \rangle \subseteq T = \mathbb{Q}[X,t]$ where $\widetilde{H} = \{g_1(X,t), \ldots, g_s(X,t)\}$ and $f \in \mathbb{Q}[t]$ is irreducible.

Output: $\widetilde{G} \subseteq T$, a Gröbner basis of $\widetilde{I}$ w.r.t. $>_K$.

choose $\mathcal{P}$, a set of random primes which are admissible of type A w.r.t. f
$\mathcal{GP} \longleftarrow \{\}$
loop
  for $p \in \mathcal{P}$ **do**
    factorize $f_p \in \mathbb{F}_p[t]$ into irreducible factors $f_p = \prod_{1 \le i \le r_p} f_{i,p}$
    for $i = 1, \ldots, r_p$ **do**
      $\widetilde{I}_{i,p} \longleftarrow \langle \widetilde{H}_p \cup \{f_{i,p}\} \rangle \subseteq \mathbb{F}_p[X,t]$
      compute the reduced Gröbner basis $\widetilde{G}_{i,p}$ of $\widetilde{I}_{i,p}$ w.r.t. $>_K$
    if p is admissible of type B w.r.t. f and $\widetilde{H}$ over $\mathbb{F}_p$ **then**
      apply Algorithm 2.5 coefficient-wise to the input

$$\Big(\big(\widetilde{G}_{1,p} \setminus \{f_{1,p}\}, \ldots, \widetilde{G}_{r_p,p} \setminus \{f_{r_p,p}\} \big), \ (f_{1,p}, \ \ldots, \ f_{r_p,p}) \Big)$$

      to obtain a set of polynomials $\widetilde{G}_p \subseteq \mathbb{F}_p[X,t]$
      $\widetilde{G}_p \longleftarrow \widetilde{G}_p \cup \{f_p\}$
    else
      $\widetilde{G}_p \longleftarrow 0$
    $\mathcal{GP} \longleftarrow \mathcal{GP} \cup \{\widetilde{G}_p\}$
  $(\mathcal{GP}, \mathcal{P}) \longleftarrow$ DeleteUnluckyPrimesSB$(\mathcal{GP}, \mathcal{P})$
  lift $(\mathcal{GP}, \mathcal{P})$ to $\widetilde{G} \subseteq T$ by applying the Chinese remainder algorithm and the Farey rational map
  if pTestSB$(\widetilde{I}, \widetilde{G}, \mathcal{P})$ **then**
    if $\widetilde{G}$ is the reduced Gröbner basis of $\langle \widetilde{G} \rangle$ **then**
      if $\widetilde{I} \subseteq \langle \widetilde{G} \rangle$ **then**
        return $\widetilde{G}$
  enlarge $\mathcal{P}$

Algorithm 2.7 Modular Gröbner basis algorithm over $K = \mathbb{Q}(\alpha)$ (`nfmodStd`)

Input: $I = \langle g_1(X, \alpha), \ldots, g_s(X, \alpha)\rangle \subseteq S = K[X]$.

Output: $G \subseteq S$, a Gröbner basis of I w.r.t. $>_1$.

1: map I to $\langle \widetilde{H}\rangle$ via the map sending α to t
2: $\widetilde{I} \longleftarrow \langle \widetilde{H}\rangle + \langle f\rangle$
3: call Algorithm 2.6 to compute the reduced Gröbner basis $\widetilde{G}$ of $\widetilde{I}$ w.r.t. $>_K = (>_1, >)$
4: lift $\widetilde{G}$ to G via the map sending t to α
5: **return** G

In the following, we show how this basis is obtained using our method: At level 1, let us choose $k = 2$ with $p_1 = 5$ and $p_2 = 13$. At level 2, we have

$$\begin{aligned} f_{p_1} &\equiv (t-2)(t+2) \mod p_1 \text{ and} \\ f_{p_2} &\equiv (t-5)(t+5) \mod p_2. \end{aligned}$$

Now, corresponding to each factor, we compute, using SINGULAR, the reduced Gröbner bases of the following ideals:

$$\begin{aligned} \widetilde{I}_{1,p_1} &= \langle x^2 + ty, txy - x + t, t - 2\rangle, \\ \widetilde{I}_{2,p_1} &= \langle x^2 + ty, txy - x + t, t + 2\rangle, \\ \widetilde{I}_{1,p_2} &= \langle x^2 + ty, txy - x + t, t - 5\rangle, \\ \widetilde{I}_{2,p_2} &= \langle x^2 + ty, txy - x + t, t + 5\rangle. \end{aligned}$$

```
> ring r = 5, (x,y,t), (dp(2),dp(1));
> ideal I1p1 = x^2+y*t, x*y*t-x+t, t-2;
> ideal I2p1 = x^2+y*t, x*y*t-x+t, t+2;
> option(redSB);
> ideal S1 = std(I1p1);
> S1;
  S1[1] = t-2
  S1[2] = y^2+2*x+2*y
  S1[3] = x*y+2*x+1
  S1[4] = x^2+2*y
> ideal S2 = std(I2p1);
> S2;
  S2[1] = t+2
  S2[2] = y^2-2*x-2*y
  S2[3] = x*y-2*x+1
  S2[4] = x^2-2*y
```

The Chinese remainder algorithm for polynomials combines these results at level 3 to obtain the reduced Gröbner basis of $\widetilde{I}_{p_1}$ with high probability, as follows:

```
> list l = S1, S2;
> poly f = t^2+1;
> list m = t-2, t+2;
  // CRA for polynomials (coefficient-wise):
> ideal Gp1 = chinrempoly(l, m);
> Gp1 = simplify(Gp1,2); // erase the zero entries
> Gp1 = f, Gp1;
> Gp1;
  Gp1[1] = t^2+1
  Gp1[2] = y^2+x*t+y*t
  Gp1[3] = x*y+x*t+1
  Gp1[4] = x^2+y*t
```

Similarly, the reduced Gröbner basis of $\widetilde{I}_{p_2}$, with high probability, is

```
> Gp2;
  Gp2[1] = t^2+1
  Gp2[2] = y^2+x*t+y*t
  Gp2[3] = x*y+x*t+1
  Gp2[4] = x^2+y*t
```

It is not hard to see that the primes p_1 and p_2 are admissible of type B w.r.t. f and $\widetilde{H} = \{x^2 + ty, txy - x + t\}$. Furthermore, it is also clear that they are lucky primes for $\widetilde{I} = \langle \widetilde{H}, f \rangle$. At this point we have to change the current basering in SINGULAR to characteristic zero in order to apply the Chinese remainder algorithm for integers and to pull the modular coefficients back to the rational numbers.

```
/* Chinese remaindering for integers */
> ring s = 0, (x,y,t), (dp(2),dp(1));
> list l = imap(r, Gp1), imap(r, Gp2);
> intvec m = 5, 13;
> ideal j = chinrem(l, m);
> j;
  j[1] = t^2+1
  j[2] = y^2+x*t+y*t
  j[3] = x*y+x*t+1
  j[4] = x^2+y*t
```

```
/* rational reconstruction */
> j = farey(j, 5*13);
> j;
  j[1] = t^2+1
  j[2] = y^2+x*t+y*t
  j[3] = x*y+x*t+1
  j[4] = x^2+y*t
```

Note that the computed result already coincides with the reduced Gröbner basis stated above. To simplify the presentation, we therefore skip some of the steps in Algorithm 2.6, such as the final test. However, we have to map the result back to the ring $\mathbb{Q}(a)[x,y]$ in SINGULAR:

```
> ring sr = (0,a), (x,y,t), (dp(2), dp(1));
> minpoly = a^2+1;
> ideal G = imap(s, j);
> G = subst(G, t, a);
> G = simplify(G,2);
> G; // G is the reduced Groebner basis of I
  G[1] = y^2+a*x+a*y
  G[2] = x*y+a*x+1
  G[3] = x^2+a*y
```

Thus we get the same result as the one we mentioned at the beginning.

2.7 Implementation and Timings

Algorithm 2.7 is implemented in the SINGULAR library `nfmodstd.lib` and we compare its performance against the implementation of [27, Algorithm 1] in the library `modstd.lib` (the command is `modStd`), the SINGULAR command `std`, and the Magma [9, 38] command `GroebnerBasis`. For `modStd`, we added the minimal polynomial f to the given input ideal I (considered as an ideal in a polynomial ring over a polynomial ring) and computed the reduced Gröbner basis of the ideal $\widetilde{I} = \langle \widetilde{H} \rangle + \langle f \rangle$ w.r.t. $>_K$. For `GroebnerBasis` and `std`, we computed the reduced Gröbner basis of the ideal I over an algebraic number field with the minimal polynomial f. Note that the implementation of our algorithm is internally linked with the existing implementation of Algorithm 1 in [27].

We consider nine benchmark problems (see appendix Section A.1) to demonstrate the superiority of our new algorithm. The cyclic ideal C_n in n variables has become a benchmark problem for Gröbner basis techniques. For our algorithm, we have replaced the coefficients of this ideal by a random element in $\mathbb{Q}(a)$ where a is an algebraic number

(see, for example, the ideal `I6` in the appendix). Some of the benchmark problems are chosen from [2, 35] (the ideals `I1` and `I2` are from [2], `I6` and `I7` are from [35]) where the coefficients are replaced by a random algebraic number. The minimal polynomials, selected for our computations, are:

$$\begin{aligned}
m_1 &= a^2 + 1\,,\\
m_2 &= a^5 + a^2 + 2\,,\\
m_3 &= a^7 - 7a + 3\,,\\
m_4 &= a^6 + a^5 + a^4 + a^3 + a^2 + a + 1\,,\\
m_5 &= a^{12} - 5a^{11} + 24a^{10} - 115a^9 + 551a^8 - 2640a^7\\
&\quad + 12649a^6 - 2640a^5 + 551a^4 - 115a^3 + 24a^2 - 5a + 1\,,\\
m_6 &= a^2 + 5a + 1\,,\\
m_7 &= a^8 - 16a^7 + 19a^6 - a^5 - 5a^4 + 13a^3 - 9a^2 + 13a\\
&\quad + 17\,,\ \text{and}\\
m_8 &= a^7 + 10a^5 + 5a^3 + 10a + 1\,.
\end{aligned}$$

With respect to these minimal polynomials, timings are conducted by using Singular 4.0.2 and Magma version V2.21-2 on a Dell PowerEdge R720 machine with two Intel Xeon E5-2690 CPUs, 16 cores and 32 threads in total, 2.9-3.8 GHz, and 192 GB of RAM running the Gentoo Linux operating system.

The results are summarized in Table 2.1. Some of the computations in Magma did not finish within 12 hours. This is indicated by a dash (-). Note that in all those cases, the computation also occupied an excessive amount of memory, more than 100 GB at the point when we interrupted it. All timings are in seconds. We use the degree reverse lexicographical ordering (`dp` in Singular) for all examples.

In our implementation, the number of primes which are chosen in line 1 of Algorithm 2.6 depends on the number of cores. For our timings, we started with 10 primes on one core and 25 primes on 32 cores. The runtime depends heavily on the splitting behaviour of the minimal polynomial modulo the chosen primes. Finding the optimal strategy for this is still under active research.

Remark 2.7.1. We understand that Magma has no parallel version of the Gröbner basis algorithm which works over algebraic number fields. Therefore we have conducted the timings in Magma using one core only.

From Table 2.1, we see that the Singular commands `std` and `modStd` perform well in comparison to the Magma command `GroebnerBasis`. However, one can see that our algorithm `nfmodStd` is even much faster.

Example		Magma	Singular				
Ideal	Min. Poly.	`Groebner Basis`	`std`	`modStd`		`nfmodStd`	
				one core	32 cores	one core	32 cores
I1	m_1	1241.98	1.51	1.24	0.37	0.22	0.13
I2	m_2	error	70.55	19.59	4.79	1.89	0.61
I3a	m_3	-	0.90	143.79	9.34	3.27	0.51
I3b	m_3	-	314.00	11212.00	1118.78	97.43	19.23
I4	m_4	-	265.53	9163.38	567.03	686.01	99.41
I5	m_5	-	2061.95	3321.28	256.58	430.23	71.47
I6	m_6	2.93	8931.13	197.20	47.54	24.26	8.99
I7	m_7	-	0.90	2044.08	195.41	8.54	1.87
I8	m_8	-	15477.87	15274.97	4787.49	92.99	23.89

Table 2.1: Total running times in seconds for computing a Gröbner basis of the considered ideals with the corresponding minimal polynomial via `GroebnerBasis`, `std`, `modStd` and `nfmodStd`, using 1 core and 32 cores where applicable

Chapter 3

Syzygies over Algebraic Number Fields

One of the most immediate and important applications of Gröbner bases is the computation of syzygies. Roughly speaking, a *syzygy* is a relation to zero between given elements of a given module. In many algorithms of commutative algebra, syzygies play an important role, for example, for the computation of ideal quotients or ideal intersections, see [25, 34]. Moreover, many important constructions in algebraic geometry make use of the computation of syzygies, see [18]. Syzygies contain important algebraic and geometric information.

Let A be a ring, and let M be an A-module. Let $f_1, \ldots, f_k$ be elements of M. Then the set of all syzygies on $f_1, \ldots, f_k$ is an A-module and we call this module the *syzygy module* of $f_1, \ldots, f_k$. Buchberger's algorithm for computing Gröbner bases allows us to compute syzygy modules as well, see [25, Lemma 2.5.3]. The reduction to zero of the S-vector of a pair of vectors in a Gröbner basis provides a syzygy. Since, from the theoretical point view, Gröbner bases can be computed over any field using Buchberger's algorithm, the same holds for modules of syzygies. In particular, the syzygy module can be computed over an algebraic number field. Consider a simple extension $K = \mathbb{Q}(\alpha)$ of $\mathbb{Q}$, where α is an algebraic number. The main result of this chapter is a fast algorithm for computing syzygy modules over K. This algorithm uses the results from Chapter 2. In Chapter 2, we described modular techniques over the field K that solve the coefficient swell problem which occurs due to the arithmetic in K. Let $f \in \mathbb{Q}[t]$ be the minimal polynomial of the algebraic number α. Like in Chapter 2, the new method which we present in this chapter uses the concepts of modular techniques [2, 27] and univariate polynomial factorization over finite fields. However, unlike in Chapter 2, it does not rely on Theorem 2.4.1. This is because, in contrast to Chapter 2, we do not add the factors of $(f \bmod p)$, where p is a suitable prime, to the input submodule, see Remark 3.3.4.

To state the idea of the new method more precisely, we proceed as follows: Let K be defined as above and consider the ring $K[x_1, \ldots, x_n] =: K[X]$, and let $>$ be a global monomial ordering on $\mathrm{Mon}(X)$. Let $H = \{f_1, \ldots, f_k\}$ be a subset of $K[X]^r = \bigoplus_{i=1}^{r} K[X]e_i$ where $e_1, \ldots, e_r$ form the canonical basis of the free module $K[X]^r$, and let $M \subseteq K[X]^r$ be the $K[X]$-module generated by H. Consider the embedding

$$K[X]^r \subseteq K[X]^{r+k} = \bigoplus_{i=1}^{r+k} K[X]e_i,\ e_i \mapsto e_i\,.$$

Consider the submodule $F = \langle f_1 + e_{r+1}, \ldots, f_k + e_{r+k} \rangle$ of $K[X]^{r+k}$, and let $G = \{g_1, \ldots, g_s\}$ be a Gröbner basis of F w.r.t. the ordering $>_{\mathrm{POT}}$ which is an elimination ordering w.r.t. $e_1, \ldots, e_r$, see Definition 1.2.10. Here we compute G using a method similar to the one described in Chapter 2, that is, using Algorithm 3.8, see Section 3.3 below. Note that we order the set G w.r.t. $>_{\mathrm{POT}}$ reversely. Suppose

$$G \cap \bigoplus_{i=r+1}^{r+k} K[X]e_i \neq \emptyset.$$

Then we may partition the generating set G of the submodule F as a 2×2 block matrix

$$G = \begin{bmatrix} B_0 & B_{\mathrm{GB}} \\ B_{\mathrm{syz}} & B_T \end{bmatrix} \in K[X]^{(r+k)\times s}$$

where

(a) $B_0 \in K[X]^{r\times l}$ is the zero matrix,

(b) $B_{\mathrm{GB}} \in K[X]^{r\times(s-l)}$ does not contain zero columns, and

(c) $B_{\mathrm{syz}} \in K[X]^{k\times l}$ and $B_T \in K[X]^{k\times(s-l)}$

for some uniquely determined integer l with $1 \leq l \leq s$. With this notation, the block B_{syz} is a syzygy matrix of H, that is, $HB_{\mathrm{syz}} = B_0$, see [25, Lemma 2.5.3]. Equivalently, the vector $(h_1, \ldots, h_k) \in K[X]^k$ is a syzygy on $f_1, \ldots, f_k \in M$ if and only if the extended vector $(\mathbf{0}, h_1, \ldots, h_k) \in K[X]^{r+k}$ with $\mathbf{0} \in K[X]^r$ is a $K[X]$-linear combination of the columns of the matrix

$$\begin{pmatrix} f_1 & \ldots & f_k \\ 1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 1 \end{pmatrix} \in K[X]^{(r+k)\times k}.$$

The columns of the block B_{GB} form a Gröbner basis of M whereas the block B_T is a transformation matrix such that $B_{\mathrm{GB}} = HB_T$. In this chapter, we are not interested in the matrices B_{GB} and B_T. If $G \cap \bigoplus_{i=r+1}^{r+k} K[X]e_i = \emptyset$, then this means that the module of syzygies of M is the zero module.

The Gröbner basis G of the submodule F of $K[X]^r$ as above can be computed w.r.t. $>_{\mathrm{POT}}$ as follows: First recall that the algebraic number field K can be represented as the residue class ring $\mathbb{Q}[t]/\langle f \rangle$. Starting from a polynomial ring over $\mathbb{Q}[t]/\langle f \rangle$, we apply the modular methods for computing Gröbner bases described in [2, 4, 27] to pass to positive characteristic p. As in Chapter 2, we choose a set $\mathcal{P}$ of suitable prime numbers, see Definition 2.4.2, such that the image f_p of f in $\mathbb{F}_p[t]$ is, for $p \in \mathcal{P}$, reducible and squarefree. Let $f_{1,p}, \ldots, f_{r_p,p}$ be the factors of f_p. With respect to these factors, we again apply modular methods passing to the rings $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$, by making slight modifications to the algorithms presented in Chapter 2. However, unlike in Chapter 2, we do not avoid computing in the quotient rings $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$. Once we have computed the corresponding

Gröbner basis for each of these factors, we first recombine the results to a set of vectors G_p over $\mathbb{F}_p[t]/\langle f_p\rangle$ using Chinese remaindering for polynomials, see Algorithm 2.5. In a second lifting step, the sets G_p, $p \in \mathcal{P}$, are then used to reconstruct a set of vectors G over $\mathbb{Q}[t]/\langle f\rangle$, via Chinese remaindering for integers and rational reconstruction. By making a final test as described in Chapter 2, we finally obtain the set G that generates F as desired.

Section 3.1 introduces some notation which is used throughout this chapter. An overview of the new method is outlined in Section 3.2. Section 3.3 presents the proposed new algorithm for computing syzygy modules. An illustrating example is presented in Subsection 3.3.1 whereas timings comparing the new algorithm to other approaches are presented in Subsection 3.3.2. A further optimization is presented in Section 3.4. An example illustrating the optimized algorithm is presented in Subsection 3.4.1. Subsection 3.4.2 presents timings comparing the algorithms in Section 3.3 and Section 3.4. The benchmark problems which we use for the timings are listed in the appendix, see Section A.2.

3.1 Notation

Let $K = \mathbb{Q}(\alpha)$ be an algebraic number field and let $f \in \mathbb{Q}[t]$ be the minimal polynomial of the algebraic number α. Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and let t be an extra variable. Consider the polynomial rings $A = \mathbb{Q}(\alpha)[X] = K[X]$ and $\mathbb{Q}[t]$. Fix a global ordering $>$ on $\mathrm{Mon}(X)$. Consider the free A-module A^r and fix a module ordering $\succ$ on A^r, see Definition 1.2.8. We assume that $\succ$ is of type $>_{\mathrm{POT}}$ or $>_{\mathrm{TOP}}$, see Definition 1.2.10. Let $H = \{g_1, \ldots, g_k\}$ be a subset of A^r, and let $M = \langle H\rangle \subseteq A^r$ be the A-module generated by H. Let g be a non-zero element of A^r. Note that by Theorem 1.3.13, any coefficient $\beta \in K$ of g can be written as a polynomial $\beta = \sum_{i=0}^{d} c_i\,\alpha^i$ in α with coefficients in $\mathbb{Q}$, where $d < \deg f$. In this chapter, when we say that some prime p does not divide any numerator or any denominator of the coefficients of g, we mean that none of the numerators or denominators of the coefficients $c_i \in \mathbb{Q}$ of any coefficient β of g is divisible by p.

Let p be a prime number, and let $Z_{\langle p\rangle} \subseteq \mathbb{Q}$ be the set of all rational numbers (in lowest terms) such that none of the denominators of the elements in $Z_{\langle p\rangle}$ vanishes modulo p. That is,

$$Z_{\langle p\rangle} = \left\{ \frac{a}{b} \mid a \in \mathbb{Z},\ b \in \mathbb{Z} \setminus p\mathbb{Z} \right\} \subseteq \mathbb{Q}\,.$$

Consider the map from $Z_{\langle p\rangle}$ to $\mathbb{F}_p$ which sends $\frac{a}{b}$ to $ab^{-1} \in \mathbb{F}_p$. If p does not divide any denominator of the coefficients of each element in $H \cup \{f\}$, we apply this map to the coefficients of these elements. We then write $f_p := (f \bmod p) \in \mathbb{F}_p[t]$ and $M_p := \langle H_p\rangle = \langle g_p \mid g \in H\rangle \subseteq A_p^r$ with $A_p := (\mathbb{F}_p[t]/\langle f_p\rangle)[X]$. Let F be the submodule of A^{r+k} generated by the set $\{g_1 + e_{r+1}, \ldots, g_k + e_{r+k}\} \subseteq A^{r+k}$, where $e_1, \ldots, e_{r+k}$ denote the canonical generators of A^{r+k}.

Let p be a prime which is admissible of type A w.r.t. f, see Definition 2.4.2. In this

case, we write

$$f_p = \prod_{i=1}^{r_p} f_{i,p} \in \mathbb{F}_p[t],$$

where $r_p > 1$ and each factor $f_{i,p}$ of f_p is irreducible. With respect to this prime, by Corollary 2.3.3, we have the ring isomorphism

$$\mathbb{F}_p[t]/\langle f_p \rangle \cong \prod_{i=1}^{r_p} \mathbb{F}_p[t]/\langle f_{i,p} \rangle. \tag{3.1}$$

Since f is independent of X, the above isomorphism can naturally be extended to a ring isomorphism

$$\left(\mathbb{F}_p[t]/\langle f_p \rangle\right)[X] \cong \prod_{i=1}^{r_p} \left(\mathbb{F}_p[t]/\langle f_{i,p} \rangle\right)[X]. \tag{3.2}$$

Note that since each $f_{i,p}$ is irreducible, each residue class ring $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$ is a field, see Theorem 1.1.10.

3.2 Overview of the New Method

The idea of the algorithm which we consider in this chapter is similar to the one discussed in Chapter 2. As mentioned in the introduction, the main difference is the following: Let p be a prime which is admissible of type A w.r.t. f, see Definition 2.4.2. Instead of adding the factors of f_p to the input submodule, we work over the residue class rings $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$, where the $f_{i,p}$ are factors of f_p.

The general idea for computing a syzygy module of M is as follows: First, set

$$H_{i,p} := \left\{ g_p \bmod f_{i,p} \mid g \in \{g_1 + e_{r+1}, \ldots, g_k + e_{r+k}\} \subseteq A^{r+k} \right\} \subseteq A_{i,p}^{r+k}$$

where $A_{i,p} = (\mathbb{F}_p[t]/\langle f_{i,p} \rangle)[X]$. For each $i \in \{1, \ldots, r_p\}$, we compute the reduced Gröbner basis $G_{i,p}$ of the $A_{i,p}$-module generated by $H_{i,p}$ over $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$ w.r.t. $>_{\mathrm{POT}}$. Using the Chinese remainder algorithm for polynomials (see Algorithm 2.5), we obtain a set of vectors $G_p \subseteq A_p^{r+k}$ satisfying the conditions $G_p \equiv G_{i,p} \bmod f_{i,p}$. We then use, following [2, 27], the Chinese remainder algorithm for integers together with rational reconstruction to lift these results to a set G which is the reduced Gröbner basis of F with high probability. Finally, from

$$G \cap \bigoplus_{i=r+1}^{r+k} K[X]e_i,$$

we compute a generating set for the syzygy module of M by [25, Lemma 2.5.3].

3.3 Algorithm for Computing Syzygies

We start by the following definition:

Definition 3.3.1. Let A be a ring, let M be an A-module, and let $f_1, \ldots, f_k \in M$. A *syzygy* or *relation* on k elements $f_1, \ldots, f_k$ is an element of the kernel of the homomorphism $\varphi : A^k \longrightarrow M$, $e_i \longmapsto f_i$, where $e_1, \ldots, e_k$ is the canonical basis of the free A-module A^k. In other words, it is a k-tuple $(g_1, \ldots, g_k) \in A^k$ satisfying $\sum_{i=1}^{k} g_i f_i = 0$.

In this definition, the set of all syzygies, denoted by $\mathrm{Syz}(f_1, \ldots, f_k)$, on $f_1, \ldots, f_k$ is a submodule of A^k. The map φ surjects onto the A-module $\langle f_1, \ldots, f_k \rangle$ and $\mathrm{Syz}(f_1, \ldots, f_k) = \mathrm{Ker}(\varphi)$ is called the *syzygy module* of $f_1, \ldots, f_k$.

Before presenting an algorithm for computing syzygy modules over K, we first consider a modified version of Algorithm 2.7 that computes the reduced Gröbner bases of given submodules over the field K. Note that since some of the definitions and remarks for ideals which we used in Chapter 2 carry over to modules almost without any changes, we will not repeat them here. For a more detailed description, see Chapter 2. For computing the reduced Gröbner basis of the submodule F, we can simply adapt those definitions and remarks for ideals to modules based on the description given in Section 3.2.

Algorithm 3.8 is the aforementioned modified version of Algorithm 2.7 which in turn calls Algorithm 2.6. It computes the reduced Gröbner basis of a given submodule over the field K.

Remark 3.3.2. Some parts of Algorithm 3.8 can be parallelized as described in Remark 2.5.1.

The following lemma provides a method to compute syzygies for submodules of A^r.

Lemma 3.3.3 ([25, Lemma 2.5.3])**.** *Let $>$ be a global monomial ordering on A. Let $M = \langle g_1, \ldots, g_k \rangle \subseteq A^r = \bigoplus_{i=1}^{r} Ae_i$, with $e_1, \ldots, e_r$ the canonical basis of A^r. Consider the embedding $A^r \subseteq A^{r+k} = \bigoplus_{i=1}^{r+k} Ae_i$, $e_i \mapsto e_i$ and the canonical projection $\pi : A^{r+k} \longrightarrow A^k$. Let $G = \{g'_1, \ldots, g'_s\}$ be a Gröbner basis of the submodule $F = \langle g_1 + e_{r+1}, \ldots, g_k + e_{r+k} \rangle$ of A^{r+k} w.r.t. the elimination ordering $>_{\mathrm{POT}}$ for $e_1, \ldots, e_r$. Suppose that $\{g'_1, \ldots, g'_l\} = G \cap \bigoplus_{i=r+1}^{r+k} Ae_i$, then*

$$\mathrm{Syz}(M) = \langle \pi(g'_1), \ldots, \pi(g'_l) \rangle.$$

The method for computing syzygy modules given in Lemma 3.3.3 is summarized in Algorithm 3.9. Although this algorithm works for any ring $A = \Bbbk[X]$ where $\Bbbk$ is a field (see also [25, Algorithm 2.5.4]), we restrict ourselves to the ring A with $\Bbbk = K = \mathbb{Q}(\alpha)$.

Remark 3.3.4. Recall that Algorithm 3.8 is an extension of Algorithm 2.6 to submodules of A^r that computes in quotient rings. However, one can also easily extend

Algorithm 3.8 Modified modular Gröbner bases algorithm over $K = \mathbb{Q}(\alpha)$

Input: A submodule $M = \langle H \rangle \subseteq A^r$, where $H = \{g_1, \ldots, g_k\}$ and $A = K[X]$.

Output: $G \subseteq A^r$, the reduced Gröbner basis of M w.r.t. $\succ$.

1: let f be the minimal polynomial of α
2: choose $\mathcal{P}$, a set of random primes which are admissible of type A w.r.t. f
3: $\mathcal{GP} \longleftarrow \{\}$
4: **loop**
5: **for** $p \in \mathcal{P}$ **do**
6: factorize $f_p \in \mathbb{F}_p[t]$ into irreducible factors $f_p = \prod_{1 \le i \le r_p} f_{i,p}$
7: **for** $i = 1, \ldots, r_p$ **do**
8: $M_{i,p} \longleftarrow M_p \bmod f_{i,p} \subseteq A_{i,p}^r$ with $A_{i,p} = (\mathbb{F}_p[t]/\langle f_{i,p} \rangle)[X]$
9: compute the reduced Gröbner basis $G_{i,p}$ of $M_{i,p}$ over $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$ w.r.t. $\succ$
10: **if** p is admissible of type B w.r.t. f and H over $\mathbb{F}_p$ **then**
11: apply Algorithm 2.5 coefficient-wise to the input

$$\big((G_{1,p}, \ldots, G_{r_p,p}), (f_{1,p}, \ldots, f_{r_p,p})\big)$$

to obtain a set of vectors $G_p \subseteq A_p^r$ where $A_p = (\mathbb{F}_p[t]/\langle f_p \rangle)[X]$
12: **else**
13: $G_p \longleftarrow 0$
14: $\mathcal{GP} \longleftarrow \mathcal{GP} \cup \{G_p\}$
15: $(\mathcal{GP}, \mathcal{P}) \longleftarrow$ DeleteUnluckyPrimes$(\mathcal{GP}, \mathcal{P})$
16: lift $(\mathcal{GP}, \mathcal{P})$ to $G \subseteq A^r$ by applying the Chinese remainder algorithm for integers and the Farey rational map
17: **if** pTestSB$(M, G, \mathcal{P})$ **then**
18: **if** G is the reduced Gröbner basis of $\langle G \rangle$ **then**
19: **if** $M \subseteq \langle G \rangle$ **then**
20: **return** G
21: enlarge $\mathcal{P}$

Algorithm 3.9 Syzygy modules over $K = \mathbb{Q}(\alpha)$ (`nfmodSyz`)

Input: A submodule $M = \langle g_1, \ldots, g_k \rangle$ of A^r, where $A = K[X]$.

Output: $S = \{s_1, \ldots, s_l\} \subseteq A^k$ such that $\langle S \rangle = \mathrm{Syz}(M) \subseteq A^k$.

1: let $F = \{g_1 + e_{r+1}, \ldots, g_k + e_{r+k}\}$, where $e_1, \ldots, e_{r+k}$ denote the canonical generators of $A^{r+k} = A^r \bigoplus A^k$ such that $g_1, \ldots, g_k \in A^r = \bigoplus_{i=1}^{r} Ae_i$

2: let $G \subseteq A^{r+k}$ be the output of Algorithm 3.8 applied to F w.r.t. $>_{\mathrm{POT}}$

3: let $\{g'_1, \ldots, g'_l\} = G \cap \bigoplus_{i=r+1}^{r+k} Ae_i$ with $g'_i = \sum_{j=1}^{k} a_{ij} e_{r+j},\ i = 1, \ldots, l$

4: let $s_i = (a_{i1}, \ldots, a_{ik}),\ i = 1, \ldots, l$

5: **return** $S := \{s_1, \ldots, s_l\}$

Algorithm 2.6 to an algorithm computing Gröbner bases for submodules of A^r that does not compute in quotient rings as in the case of ideals. This can be done by adding the vectors $fe_1, \ldots, fe_r \in A^r$, where f is the minimal polynomial of α, to the input submodules to be considered. Algorithm 3.9 still works if we apply this extended algorithm in line 2 of Algorithm 3.9. Nonetheless, experiments have shown that this approach is limited in terms of efficiency compared to using Algorithm 3.8 in line 2 of this algorithm for our test cases. This might be due to the additional r column vectors.

3.3.1 An Illustrative Example

Consider the polynomial ring $A = \mathbb{Q}(\alpha)[x, y, z]$, where α is algebraic over $\mathbb{Q}$ with minimal polynomial $f = t^2 + 1 \in \mathbb{Q}[t]$. Consider the A-module $M \subseteq A^2$ generated by

$$\begin{aligned} g_1 &= (\alpha\, x + y)e_1 + (\alpha\, x - z)e_2\,, \\ g_2 &= (yz - \alpha\, x)e_1\,, \text{ and} \\ g_3 &= (y + z)e_2\,. \end{aligned}$$

To compute the syzygy module of M, first, as in Algorithm 3.9, we consider the submodule F of $A^{2+3} = A^5$ generated by $g_1 + e_3$, $g_2 + e_4$, and $g_3 + e_5$. In other words, F is generated by the columns of the matrix

$$\begin{pmatrix} \alpha\, x + y & yz - \alpha\, x & 0 \\ \alpha\, x - z & 0 & y + z \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in A^{5\times 3}\,.$$

By applying Algorithm 3.8, we compute the reduced Gröbner basis of F w.r.t. $>_{\mathrm{POT}}$. We thus obtain the set $G = \{g'_1, \ldots, g'_5\} \subseteq A^5$ of vectors, where

$$\begin{aligned}
g'_1 &= (y^2z + yz^2 - \alpha\, xy - \alpha\, xz)e_3 + (-\alpha\, xy - y^2 - \alpha\, xz - yz)e_4 \\
&\quad + (-\alpha\, xyz + yz^2 - x^2 - \alpha\, xz)e_5, \\
g'_2 &= (y + z)e_2 + e_5, \\
g'_3 &= (xz^2 + \alpha\, z^3 + \alpha\, x^2 - xz)e_2 + (\alpha\, yz + x)e_3 + (x - \alpha\, y)e_4 + (xz + \alpha\, z^2)e_5, \\
g'_4 &= (x - \alpha\, y)e_1 + (x + \alpha\, z)e_2 + (-\alpha)e_3, \\
g'_5 &= (yz + y)e_1 + (\alpha\, x - z)e_2 + e_3 + e_4\,.
\end{aligned}$$

From the above result, we get, as in line 3 of Algorithm 3.9,

$$G \cap \bigoplus_{i=3}^{5} Ae_i = \{g'_1\} = \left\{ \begin{pmatrix} 0 \\ 0 \\ y^2z + yz^2 - \alpha\, xy - \alpha\, xz \\ -\alpha\, xy - y^2 - \alpha\, xz - yz \\ -\alpha\, xyz + yz^2 - x^2 - \alpha\, xz \end{pmatrix} \right\}.$$

From this set, we obtain the syzygy matrix

$$B_{\mathrm{syz}} = \begin{pmatrix} y^2z + yz^2 - \alpha\, xy - \alpha\, xz \\ -\alpha\, xy - y^2 - \alpha\, xz - yz \\ -\alpha\, xyz + yz^2 - x^2 - \alpha\, xz \end{pmatrix} \in A^{3\times 1}$$

of

$$H := (g_1, g_2, g_3) = \begin{pmatrix} \alpha\, x + y & yz - \alpha\, x & 0 \\ \alpha\, x - z & 0 & y + z \end{pmatrix} \in A^{2\times 3}$$

such that $H \cdot B_{\mathrm{syz}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, that is, $\mathrm{Syz}(M) = \langle \pi(g'_1) \rangle \subseteq A^3$, where

$$\pi(g'_1) = (y^2z + yz^2 - \alpha\, xy - \alpha\, xz)e_1 + (-\alpha\, xy - y^2 - \alpha\, xz - yz)e_2 + (-\alpha\, xyz + yz^2 - x^2 - \alpha\, xz)e_3\,.$$

3.3.2 Implementation and Timings

Algorithm 3.9 is implemented in the Singular library `nfmodsyz.lib`. We compare its performance against the Magma [9, 39] command `SyzygyModule` and the Singular command `syz`.

We consider twelve benchmark problems as described in appendix Section A.2 to demonstrate the efficiency of the new algorithm Algorithm 3.9. The first six benchmark problems are chosen from appendix Section A.1. These are I1, I2, I3a, I4, I5 and I7 and

their corresponding minimal polynomials are:

$$\begin{aligned}
m_1 &= a^2 + 1\,,\\
m_2 &= a^5 + a^2 + 2\,,\\
m_3 &= a^7 - 7a + 3\,,\\
m_4 &= a^6 + a^5 + a^4 + a^3 + a^2 + a + 1\,,\\
m_5 &= a^{12} - 5a^{11} + 24a^{10} - 115a^9 + 551a^8 - 2640a^7\\
&\quad + 12649a^6 - 2640a^5 + 551a^4 - 115a^3 + 24a^2 - 5a + 1\,, \text{ and}\\
m_7 &= a^8 - 16a^7 + 19a^6 - a^5 - 5a^4 + 13a^3 - 9a^2 + 13a + 17\,.
\end{aligned}$$

For the remaining benchmark problems, we choose the following minimal polynomials:

$$\begin{aligned}
m_8 &= a^3 + a + 1\,,\\
m_9 &= a^4 + 2a^2 + 3\,,\\
m_{10} &= a^3 + 7a - 5\,,\\
m_{11} &= a^7 + 2a^6 + 7a^4 + 3a + 17\,, \text{ and}\\
m_{12} &= a^6 + 3a^5 + a^3 + 7a^2 + 11.
\end{aligned}$$

Remark 3.3.5. The implementation of Algorithm 3.9 also works for the coefficient field $K = \mathbb{Q}$, we may formally consider a as minimal polynomial in this case, see the last example in Table 3.1. In this case, we apply the modular algorithms described in [2, 27] by making slight modifications. However, we do not discuss this case here.

With respect to the above minimal polynomials, the timings are conducted by using SINGULAR 4.0.3 and MAGMA V2.21-11 on a Dell PowerEdge R720 machine with 16 cores and 32 threads, 2.9-3.8 GHz, and 192 GB of RAM running the Gentoo Linux operating system.

The results are summarized in Table 3.1. For the example in the last row of this table, see Remark 3.3.5. All timings are in seconds. We use $>_{\mathrm{POT}}$ on A^{r+k}, where $>$ is the degree reverse lexicographical ordering on $\mathrm{Mon}(X)$ for all examples. In this case, $>_{\mathrm{POT}}$=`(c,dp)` in SINGULAR. Some of the computations did not finish within twelve hours. This is indicated by a dash (-).

In this table, there are examples where `SyzygyModule` is faster than `syz` and, vice-versa, there are examples where `syz` is faster than `SyzygyModule`. However, with respect to the average timings, we observe that `syz` is faster than `SyzygyModule`. In Table 3.1, we also see that for almost all examples none of the methods `SyzygyModule` or `syz` beats the algorithm `nfmodSyz`. Nonetheless, there are examples where `nfmodSyz` is less efficient in comparison to `SyzygyModule` and `syz`. Moreover, none of the methods mentioned above is able to compute the example I5. Using `SyzygyModule`, we got an error message whereas the computations using `syz` and `nfmodSyz` did not finish within 12 hours. Nevertheless, there exists an algorithm which beats Algorithm 3.9 for all examples in Table 3.1, and which then also beats `syz` and `SyzygyModule`. The next section describes this algorithm.

Example		Magma	Singular	
module	min. poly.	`SyzygyModule`	`syz`	Algorithm 3.9
				32 cores
I1	m_1	-	-	124.31
I2	m_2	error	-	8083.78
I3a	m_3	-	-	117.52
I4	m_4	-	6594.93	702.96
I5	m_5	error	-	-
I7	m_7	-	30.81	70.17
I8	m_8	20.00	1272.34	34.32
I9	m_9	9.56	166.91	73.81
I10	m_{10}	182.32	134.95	40.16
I11	m_{11}	-	3785.13	1116.28
I12	m_{12}	-	7793.77	1878.58
I13	a	34.47	285.65	3.21

Table 3.1: Total running times in seconds for computing a generating set of the syzygy module of the considered submodules with the corresponding minimal polynomial via `SyzygyModule`, `syz`, and Algorithm 3.9

3.4 Optimizations

In this section, we present an algorithm for computing syzygy modules which does not use Lemma 3.3.3 directly and, hence, is different from Algorithm 3.9. In fact, it is a simple modification of Algorithm 3.8. The main difference to Algorithm 3.9 is the following: Instead of applying Algorithm 3.8 to F as in line 2 of Algorithm 3.9, we directly apply it to the input submodule M and, instead of computing reduced Gröbner bases of the submodules $M_{i,p}$, we compute generating sets for their syzygy modules in line 9 of this algorithm. By doing so, we obtain a generating set for the syzygy module of M using Algorithm 3.8 only. However, we have to be more careful about the generators of the syzygy modules of the submodules $M_{i,p}$ since they may not be unique. Nevertheless, if we compute the reduced Gröbner bases of the syzygy modules of the submodules $M_{i,p}$ in line 9 of Algorithm 3.8, we obtain a unique generating set in each case since the reduced Gröbner bases are unique, see Proposition 1.1.13. To compute the reduced Gröbner bases of the syzygy modules of the $M_{i,p}$, we internally use the SINGULAR command `syz` w.r.t. some given options, see Remark 3.4.5.

Note that in both approaches, we compute the reduced Gröbner basis of the syzygy module of M w.r.t. $>_{\mathrm{POT}}$. But a modified version of Algorithm 3.8 which we present later in this section also works w.r.t. $>_{\mathrm{TOP}}$.

In what follows, we describe the idea of a modified version of Algorithm 3.8 by applying modifications to the definitions and remarks in Chapter 2. Let us start with the following refined definition, compare Definition 2.4.4:

Definition 3.4.1. Let f and $M = \langle H \rangle$ be given as in Section 3.1. Let p be a prime which is admissible of type A w.r.t. f, and write $f_p = f_{1,p} \cdots f_{r_p,p}$ as in Definition 2.4.2. Suppose that p does not divide any numerator or any denominator of the coefficients occurring in H. For $i = 1, \ldots, r_p$, set

$$H_{i,p} := \{g_p \bmod f_{i,p} \mid g \in H\} \subseteq A_{i,p}^r$$

where $A_{i,p} = (\mathbb{F}_p[t]/\langle f_{i,p} \rangle)[X]$. Let $M_{i,p}$ be the $A_{i,p}$-module generated by $H_{i,p}$, and let $G_{i,p} \subseteq A_{i,p}^k$ be the reduced Gröbner basis of the syzygy module of $M_{i,p}$. We say that p is *admissible of type* B′ w.r.t. f and M if for all indices i, j with $i \neq j$

1. the sizes of $G_{i,p}$ and $G_{j,p}$ coincide, and
2. $\mathrm{Lm}(G_{i,p}) = \mathrm{Lm}(G_{j,p})$.

The notion of *lucky primes* for syzygy modules is defined as follows:

Definition 3.4.2 ([27])**.** Let $M = \langle H \rangle$ and f be defined as above, and let p be a prime which is admissible of type B′ w.r.t. f and H. Let $G_{i,p} \subseteq A_{i,p}^k$ be given as in the above definition. Furthermore, let G be the reduced Gröbner basis of the syzygy module of M. Then p is called *lucky* for M if and only if $\mathrm{Lm}(G_{i,p}) = \mathrm{Lm}(G)$ for $1 \leq i \leq r_p$. Otherwise p is called *unlucky* for M.

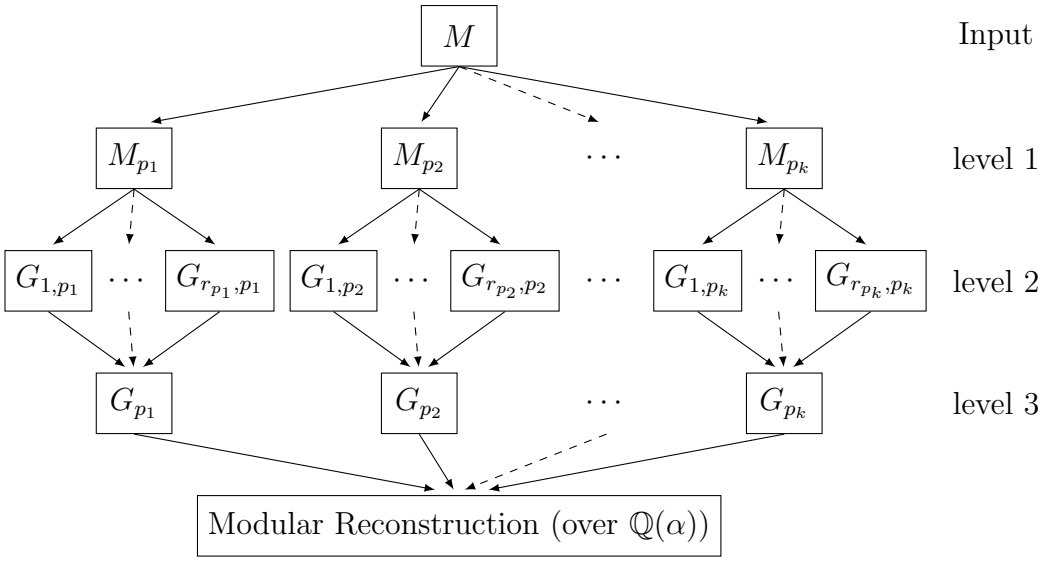

Figure 3.1: General scheme for computing the syzygy module of M over K

The general scheme of the modified new algorithm for computing syzygy modules is similar to the one described in Chapter 2 for computing Gröbner bases. Nevertheless, to simplify our presentation, we repeat it here with slight modifications.

Figure 3.1 illustrates the complete picture of the modified new algorithm for computing syzygy modules. In this figure, the only difference in comparison to Algorithm 3.8 is that, at level 2, we compute the reduced Gröbner bases of the syzygy modules of the $M_{i,p}$ instead of computing the reduced Gröbner bases of the $M_{i,p}$ themselves. The remaining steps are similar to the one described in Chapter 2: For the primes in the list $\{p_1, \ldots, p_k\}$ satisfying the conditions in Definition 3.4.1 (and only for those), the Chinese remainder algorithm for polynomials then combines these results at level 3. The remaining parts of the computation are carried out in the same way as in the modular algorithms described in [27].

To give a more detailed description of the new modified algorithm, we proceed as follows: In the first step, randomly choose a set $\mathcal{P}$ of prime numbers which are admissible of type A w.r.t. f. At level 2, given a prime $p \in \mathcal{P}$, factorize $f \in \mathbb{Q}[t]$ over $\mathbb{F}_p$ such that $f_p = \prod_{i=1}^{r_p} f_{i,p}$, $r_p > 1$, as in Section 3.1. Then compute, for each i, the reduced Gröbner basis $G_{i,p}$ of the syzygy module of the submodule $M_{i,p}$ over $\mathbb{F}_p[t]/\langle f_{i,p}\rangle$ w.r.t. $\succ$. If the prime p is admissible of type B′ w.r.t. f and H, then lift these results to a set of vectors $G_p \subseteq A_p^k$ via Chinese remaindering for polynomials (at level 3). Repeat this process for every prime $p \in \mathcal{P}$ which is admissible of type B′, in the same way as in the modular algorithms in [27]. Note that since the elements in each set $G_{i,p}$ are monic (see Definition 1.2.13), so are the elements in G_p.

Let $\mathcal{GP} := \{G_p \mid p \in \mathcal{P}\}$. From $\mathcal{P}$, the following method DeleteUnluckyPrimesSyz selects primes that are lucky with high probability:

DeleteUnluckyPrimesSyz ([27]): *We define an equivalence relation on* $(\mathcal{GP}, \mathcal{P})$

by $(G_p, p) \sim (G_q, q) :\Longleftrightarrow \mathrm{Lm}(G_p) = \mathrm{Lm}(G_q)$. *Then the equivalence class of largest cardinality*[1] *is stored in* $(\mathcal{GP}, \mathcal{P})$*, the others are deleted.*

Now, for those primes that are chosen by the above method, we apply the Chinese remainder algorithm for integers and the rational number reconstruction algorithm to the coefficients of the elements in $\mathcal{GP}$ to obtain a set $G = \{g'_1, \ldots, g'_s\} \subseteq A^k$ of vectors which generates the syzygy module of M with high probability. Note that since the elements in each set G_{p_i} are monic (see Definition 1.2.13), so are the elements in G. Once we have computed such a set G, a final verification test is needed since we cannot check whether $\mathcal{P}$ is sufficiently large. However, since this test may consume a lot of time, we first perform a test in positive characteristic as follows.

pTestSyz ([27]): *We randomly choose a prime* $p \notin \mathcal{P}$ *which is admissible of type* B′ *w.r.t.* f *and* H*. Let* $G'_p \subseteq A_p^k$ *be a set of vectors such that* $G'_p \equiv G'_{i,p} \bmod f_{i,p}$ *where* $G'_{i,p}$ *is the reduced Gröbner basis of the syzygy module of* $M_{i,p}$*. The test is positive if and only if* $G_p \subseteq \mathrm{Syz}(M_p)$ *and* G'_p *reduces to zero w.r.t.* G_p*.*

Remark 3.4.3. In pTestSyz, we use the following facts:

(1) Consider the set of vectors H_p (resp. G_p) considered as a matrix in $A^{r\times k}$ (resp. $A^{k\times s}$). If $H_p \cdot G_p = 0$, then by Definition 3.3.1 $G_p \subseteq \mathrm{Syz}(M_p)$.

(2) Note that we reduce the set G'_p w.r.t. G_p over the ring $\mathbb{F}[t]/\langle f_p\rangle$ which is not necessarily a field. Nevertheless, since the elements of G_p are monic, the reduction can still be carried out without any problem. Thus, if each element in G'_p reduces to zero w.r.t. G_p, then we have $G'_p \subseteq \langle G_p\rangle$.

Note that the reason why the method for computing the syzygy modules over algebraic number fields described above is faster than `SyzygyModule` and `syz`, see Section 3.4.2, is the same as the one given in Chapter 2, see the last paragraph of Section 2.4.

Algorithm 3.10, which is a modified version of Algorithm 3.8, computes a generating set of the syzygy module of a given submodule over the field $K = \mathbb{Q}(\alpha)$. The only differences between this algorithm and Algorithm 3.8 lie in the output, in line 9, and in lines 17-18. Algorithm 3.8 also contains one more test, see line 18, which is not part of Algorithm 3.10.

Remark 3.4.4. Suppose that the test pTestSyz$(M, G, \mathcal{P})$ in line 17 of Algorithm 3.10 succeeds. Then the next and final step in this algorithm is to verify that the set G indeed generates the syzygy module of M. That is, we have to test explicitly whether $G \subseteq \mathrm{Syz}(M)$ and $\mathrm{Syz}(M) \subseteq \langle G\rangle$. In fact, it is always possible to carry out the test $G \subseteq \mathrm{Syz}(M)$ (see line 18) by checking that $H \cdot G = 0$ (see Remark 3.4.3), see Definition 3.3.1. Unfortunately, the other inclusion, $\mathrm{Syz}(M) \subseteq \langle G\rangle$, cannot be checked since we do not

[1] Here, we have to use a weighted cardinality count if Algorithm 3.10 requires more than one round of the loop, see [4, Remark 5.7].

Algorithm 3.10 Syzygy modules over $K = \mathbb{Q}(\alpha)$ (`nfmodSyz`)

Input: A submodule $M = \langle H \rangle \subseteq A^r$, where $H = \{g_1, \ldots, g_k\}$ and $A = K[X]$.

Output: $G \subseteq A^k$, a generating set for the syzygy module of M w.r.t. $\succ$.

1: let f be the minimal polynomial of α
2: choose $\mathcal{P}$, a set of random primes which are admissible of type A w.r.t. f
3: $\mathcal{GP} \longleftarrow \{\}$
4: **loop**
5: **for** $p \in \mathcal{P}$ **do**
6: factorize $f_p \in \mathbb{F}_p[t]$ into irreducible factors $f_p = \prod_{1 \le i \le r_p} f_{i,p}$
7: **for** $i = 1, \ldots, r_p$ **do**
8: $M_{i,p} \longleftarrow M_p \bmod f_{i,p} \subseteq A_{i,p}^r$ with $A_{i,p} = (\mathbb{F}_p[t]/\langle f_{i,p} \rangle)[X]$
9: compute the reduced Gröbner basis $G_{i,p}$ of the syzygy module of $M_{i,p}$ over $\mathbb{F}_p[t]/\langle f_{i,p} \rangle$ w.r.t. $\succ$
10: **if** p is admissible of type B′ w.r.t. f and H over $\mathbb{F}_p$ **then**
11: apply Algorithm 2.5 coefficient-wise to the input

$$\big((G_{1,p}, \ldots, G_{r_p,p}),\ (f_{1,p}, \ldots, f_{r_p,p})\big)$$

to obtain a set of vectors $G_p \subseteq A_p^k$ with $A_p = (\mathbb{F}_p[t]/\langle f_p \rangle)[X]$
12: **else**
13: $G_p \longleftarrow 0$
14: $\mathcal{GP} \longleftarrow \mathcal{GP} \cup \{G_p\}$
15: $(\mathcal{GP}, \mathcal{P}) \longleftarrow$ DeleteUnluckyPrimesSyz$(\mathcal{GP}, \mathcal{P})$
16: lift $(\mathcal{GP}, \mathcal{P})$ to $G \subseteq A^k$ by applying the Chinese remainder algorithm for integers and the Farey rational map
17: **if** pTestSyz$(M, G, \mathcal{P})$ **then**
18: **if** $G \subseteq \mathrm{Syz}(M)$ **then**
19: **return** G
20: enlarge $\mathcal{P}$

know $\mathrm{Syz}(M)$ without computing it which is the same as calling Algorithm 3.10. Hence we cannot guarantee that the set G generates $\mathrm{Syz}(M)$. Due to the missing test, our algorithm is probabilistic.

Remark 3.4.5. Let N be one of the submodules $M_{i,p}$ as in line 9 of Algorithm 3.10. To compute the reduced Gröbner basis of the syzygy module of N, in our implementation, see Section 3.4.2, we use the SINGULAR options `redSB` and `returnSB`. If these options are set, the SINGULAR command `syz` applied to the input submodule N returns the reduced Gröbner basis of the syzygy module $\mathrm{Syz}(N)$ of N.

Remark 3.4.6. Some parts of Algorithm 3.10 can be parallelized as described in Remark 2.5.1.

3.4.1 An Illustrative Example

Consider the polynomial ring $A = \mathbb{Q}(a)[x, y, z]$, where a is algebraic over $\mathbb{Q}$ with minimal polynomial $f = t^3 + t + 1 \in \mathbb{Q}[t]$. Compute the syzygy module of the A-module $M \subseteq A^2$ generated by the vectors

$$\begin{aligned} f_1 &= (yz + ay)e_1 + (z + (a+2)y)e_2 \,, \\ f_2 &= (y^2 + az)e_1 + z^2 e_2 \,, \text{ and} \\ f_3 &= (-xz)e_1 + ze_2 \,, \end{aligned}$$

where (e_1, e_2) is the canonical basis of A^2. A SINGULAR computation shows that the syzygy module of M is generated by just one vector $v = v_1e_1 + v_2e_2 + v_3e_3 \in A^3$, where

$$\begin{aligned} v_1 &= xz^3 + y^2z + az^2 \,, \\ v_2 &= (-a-2)xyz - xz^2 - yz^2 - ayz \,, \text{ and} \\ v_3 &= yz^3 + (-a-2)y^3 - y^2z + ayz^2 + (-a^2 - 2a)yz - az^2 \,. \end{aligned}$$

In the following, we show how this generating set is obtained using Algorithm 3.10: We use the module ordering $>_{\mathrm{POT}}$ on A^r, where $>$ is the degree reverse lexicographical ordering (`dp` in SINGULAR) on A. That is, $>_{\mathrm{POT}}$ = (`c,dp`) in SINGULAR. At level 1, let us choose $k = 2$ with $p_1 = 13$ and $p_2 = 17$. At level 2, we have

$$\begin{aligned} f_{p_1} &\equiv (t+6)(t^2 - 6t - 2) \mod p_1 \text{ and} \\ f_{p_2} &\equiv (t+6)(t^2 - 6t + 3) \mod p_2. \end{aligned}$$

Now, corresponding to each factor, we compute, using SINGULAR, the reduced Gröbner basis of the syzygy module of M over the fields

$$\mathbb{F}_{13}[t]/\langle t+6\rangle \,,\ \mathbb{F}_{17}[t]/\langle t+6\rangle \,,\ \mathbb{F}_{13}[t]/\langle t^2 - 6t - 2\rangle \,, \text{ and } \mathbb{F}_{17}[t]/\langle t^2 - 6t + 3\rangle$$

as follows:

```
> ring r1 = (13,t), (x,y,z), (c,dp);
> minpoly = t+6;
> module M = [y*z+(t)*y,(t+2)*y+z], [y^2+(t)*z,z^2], [-x*z,z];
> option(redSB);
> option(returnSB);
> module S1 = syz(M);
> S1;
  S1[1] = [x*z^3+y^2*z-6*z^2,4*x*y*z-x*z^2-y*z^2+6*y*z,
          y*z^3+4*y^3-y^2*z-6*y*z^2+2*y*z+6*z^2]
> ring r2 = (13,t), (x,y,z), (c,dp);
> minpoly = t^2-6*t-2;
> module M = [y*z+(t)*y,(t+2)*y+z], [y^2+(t)*z,z^2], [-x*z,z];
> module S2 = syz(M);
> S2;
  S2[1] = [x*z^3+y^2*z+(t)*z^2, (-t-2)*x*y*z-x*z^2-y*z^2
          +(-t)*y*z,y*z^3+(-t-2)*y^3-y^2*z+(t)*y*z^2
          +(5*t-2)*y*z+(-t)*z^2]
```

The Chinese remainder algorithm for polynomials combines these results at level 3 to obtain the set of vectors $G_{p_1} \subseteq A_{p_1}^2$, with $A_{p_1} = (\mathbb{F}_{p_1}[t]/\langle f_{p_1}\rangle)[x,y,z]$, as follows:

```
> ring rr = 13, (x,y,z,t), (c,dp);
> module S1, S2;
> S1 = imap(r1,S1);
> S2 = imap(r2,S2);
> list l = S1, S2;
> list m = t+6, t^2-6*t-2;
  // CRA for polynomials (coefficient-wise):
> LIB "nfmodstd.lib";
> module Gp1 = chinrempoly(l, m);
> ring S = (13,t), (x,y,z), (c,dp);
> minpoly = t^3+t+1;
> module Gp1 = imap(rr,Gp1);
> Gp1;
  Gp1[1] = [x*z^3+y^2*z+(t)*z^2,(-t-2)*x*y*z-x*z^2-y*z^2
           +(-t)*y*z,y*z^3+(-t-2)*y^3-y^2*z+(t)*y*z^2
           +(-t^2-2*t)*y*z+(-t)*z^2]
```

Similarly, we obtain the set of vectors

```
> Gp2;
  Gp2[1] = [x*z^3+y^2*z+(t)*z^2,(-t-2)*x*y*z-x*z^2-y*z^2
           +(-t)*y*z,y*z^3+(-t-2)*y^3-y^2*z+(t)*y*z^2
           +(-t^2-2*t)*y*z+(-t)*z^2].
```

in $A^2_{p_2}$ w.r.t. p_2 where $A_{p_2} = (\mathbb{F}_{p_2}[t]/\langle f_{p_2}\rangle)[x, y, z]$. With these results, it is easy to check that the primes p_1 and p_2 are admissible of type B′ w.r.t. f and M, see Definition 3.4.1. Furthermore, since $\mathrm{Lm}(G_{p_1}) = \mathrm{Lm}(G_{p_2})$, we choose these primes by the DeleteUnluckyPrimeSyz method for the remaining parts of the computation. At this point we have to change the current base ring in Singular to characteristic zero in order to apply the Chinese remainder algorithm for integers and to pull the modular coefficients back to the rational numbers.

```
/* Chinese remaindering for integers */
> ring Rng = (0,t), (x,y,z), (c,dp);
> module M1 = imap(S, Gp1);
> ring s = 0, (x,y,z,t), (c,dp);
> module M1 = imap(Rng, M1);
> module M2 = M1; // since Gp1 and Gp2 are generated by the same elements
> list l = M1, M2;
> intvec m = 13, 17;
> module J = chinrem(l, m);
> J;
/* rational reconstruction */
> J = farey(J, 13*17);
> J;
  J[1] = [x*z^3+y^2*z+z^2*t,-x*y*z*t-2*x*y*z-x*z^2
         -y*z^2-y*z*t,y*z^3-y^3*t+y*z^2*t-y*z*t^2
         -2*y^3-y^2*z-2*y*z*t-z^2*t]
```

Note that if we map the variable t back to a, then we get the same result as the one we mentioned at the beginning:

```
> ring sr = (0,a), (x,y,z,t), (c,dp);
> minpoly = a^3+a+1;
> module G = imap(s, J);
> G = subst(G, t, a);
> G = simplify(G, 2); // erase the zero entries
> G; // G is the syzygy module of M
  G[1] = [x*z^3+y^2*z+(a)*z^2,(-a-2)*x*y*z-x*z^2-y*z^2
```

Example		Magma	Singular			
module	min. poly.	`SyzygyModule`	`syz`	`nfmodSyz`		
				Algorithm 3.9	Algorithm 3.10	
				32 cores	one core	32 cores
I1	m_1	-	-	124.31	326.67	47.20
I2	m_2	error	-	8083.78	559.71	45.23
I3a	m_3	-	-	117.52	262.58	37.67
I4	m_4	-	6594.93	702.96	175.72	20.88
I5	m_5	error	-	-	1618.61	201.92
I7	m_7	-	30.81	70.17	221.89	28.02
I8	m_8	20.00	1272.34	34.32	51.94	8.31
I9	m_9	9.56	166.91	73.81	13.77	4.41
I10	m_{10}	182.32	134.95	40.16	9.34	2.40
I11	m_{11}	-	3785.13	1116.28	11681.350	1105.00
I12	m_{12}	-	7793.77	1878.58	1417.77	210.77
I13	a	34.47	285.65	3.21	11.07	3.21

Table 3.2: Total running times in seconds for computing a generating set for the syzygy module of the considered submodules with the corresponding minimal polynomial via `SyzygyModule`, `syz`, Algorithm 3.9 and Algorithm 3.10

```
+(-a)*y*z,y*z^3+(-a-2)*y^3-y^2*z+(a)*y*z^2
+(-a^2-2*a)*y*z+(-a)*z^2]
```

To simplify the presentation, we therefore skip some of the steps in Algorithm 3.10, such as the pTestSyz and the final verification test.

3.4.2 Implementation and Timings

Algorithm 3.10 is implemented in the Singular library `nfmodsyz.lib`. The performance of this algorithm is compared against Algorithm 3.9 and the results are summarized in Table 3.2. Like in Table 3.1, consider Remark 3.3.5 for the example in the last row.

Looking at example I5 in Table 3.2, we see that Algorithm 3.10, the optimized version of Algorithm 3.9, is faster than Algorithm 3.9. One can also see that for all examples, this algorithm outperforms the other methods `SyzygyModule` (Magma) and `syz` (Singular) by far.

Chapter 4

Sparse Interpolation of Multivariate Rational Functions

Interpolation is the process of converting a polynomial given by a black box back to the representation where the polynomial is given by a list of non-zero coefficients and corresponding terms. Polynomials are represented as either dense or sparse. So in designing an efficient algorithm for multivariate polynomial computations, it is often crucial to be careful about the expected sparsity of the polynomial, because an approach that is efficient for dense polynomials may not be for sparse cases. The problem of interpolating sparse polynomials has always been one of the central objects of research in the area of computer algebra. It is the key part of many algorithms such as the computation of polynomial gcds over algebraic function fields, see [41]. In this chapter, we discuss, following [16], how to interpolate multivariate rational functions (or how to recover fractions in $\mathbb{Q}(x_1, \ldots, x_n)$) from given numerical data. Consider a black box multivariate rational function

$$f(x_1, \ldots, x_n) = \frac{p(x_1, \ldots, x_n)}{q(x_1, \ldots, x_n)} \in \mathbb{Q}(x_1, \ldots, x_n) \tag{4.1}$$

where $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$ are multivariate polynomials of degree at most d. We want to reconstruct f in a way which is cost sensitive to its sparsity, namely the number of non-zero terms in p and q in the power basis, instead of its dense representation size $O(d^n)$. Let τ be the maximum number of terms in p and q. We say that f is τ-*sparse* if $\tau \ll \binom{n+d}{d}$ where $\binom{n+d}{d} \in O(d^n)$ is the maximum possible number of terms either p or q can have. Although the discussions in [16] are stated for a general coefficient field, we are, in particular, interested in rational functions over the field of rational numbers. The *black box representation* of f (see Figure. 4.1) is a routine that takes as input a value for each variable and evaluates the rational function at the given input. The reconstruction

Figure 4.1: Black box for rational function evaluation

of the rational function f which we will present here is based on the idea of A. Cuyt and W.-s. Lee [16]. The idea is as follows:

First, starting from the black box representation of the multivariate rational function, a set of auxiliary univariate rational functions is obtained and interpolated densely. The multivariate rational function is then reconstructed from the coefficients of the auxiliary functions through sparse multivariate polynomial interpolation. The sparsity of the multivariate function is preserved in the coefficients of the auxiliary functions. The number of black box probes which this interpolation process requires depends on the total degree d and the sparsity τ of f. Using the early termination Ben-Or/Tiwari algorithm, the overall performance of this rational interpolation algorithm requires $O(\tau d)$ black box evaluations to recover fractions in $\mathbb{Q}(x_1, \ldots, x_n)$.

4.1 Multivariate Sparse Rational Interpolation

If a given black box multivariate rational function is defined at $0 = (0, \ldots, 0)$, its constant term in the denominator is known to be non-zero and can be normalized to 1. However, in general, the black box multivariate rational function may not be defined at 0. In this case, one can always shift the coordinates such that the rational function has a non-zero constant in the denominator. But the sparsity of the original multivariate rational function will in general be lost in this representation. This is because the representation of a rational function usually becomes dense when the basis is shifted. As a result, the interpolation algorithm becomes much slower. To handle this situation, we now present a shifting strategy preserving the sparsity of f which is demonstrated in [16]. Note that the purpose of the non-zero constant is to guarantee an a priori normalization of f.

To begin with, let $R = \mathbb{Q}[x_1, \ldots, x_n]$ and consider

$$f(x_1, \ldots, x_n) = \frac{\overbrace{\sum_{k=1}^{s} a_k x_1^{d_{k,1}} \cdots x_n^{d_{k,n}}}^{p(x_1,\ldots,x_n)\in R}}{\underbrace{\sum_{l=1}^{t} b_l x_1^{e_{l,1}} \cdots x_n^{e_{l,n}}}_{q(x_1,\ldots,x_n)\in R}} \tag{4.2}$$

where $\mathbb{Q} \ni a_k, b_l \neq 0$ for $1 \leq k \leq s,\ 1 \leq l \leq t$, $\deg p =: \nu$, $\deg q =: \delta$, and $\gcd(p, q) = 1$.

Let us introduce a homogenizing variable for f and define a shift for the power basis, to form an auxiliary rational function, in the following definition:

Definition 4.1.1. Let z be a homogenizing variable for f. Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be any point in $\mathbb{Q}^n$. The point σ is called a *shift* if $f(\sigma)$ is defined, that is, if $q(\sigma) = q(\sigma_1, \ldots, \sigma_n) \neq 0$. Moreover, the σ-shifted homogenization of f, denoted by $\Gamma_\sigma(z, x_1, \ldots,$

x_n), is defined as

$$\Gamma_\sigma(z, x_1, \ldots, x_n) = f(x_1 z + \sigma_1, \ldots, x_n z + \sigma_n) \in \mathbb{Q}(x_1 z, \ldots, x_n z)\,. \tag{4.3}$$

The function Γ_σ is called an *auxiliary rational function.*

Remark 4.1.2. We write the function Γ_σ in the above definition as:

$$\Gamma_\sigma(z, x_1, \ldots, x_n) = \frac{\overbrace{\widetilde{\alpha}_\nu \cdot z^\nu + \ldots + \widetilde{\alpha}_1 \cdot z + \widetilde{\alpha}_0}^{\widetilde{N}(z) \in R[z]}}{\underbrace{\widetilde{\beta}_\delta \cdot z^\delta + \ldots + \widetilde{\beta}_1 \cdot z + \widetilde{\beta}_0}_{\widetilde{D}(z) \in R[z]}} \tag{4.4}$$

where $\widetilde{\alpha}_k, \widetilde{\beta}_l \in R$ are homogeneous multivariate polynomials of total degree k and l, respectively, for all k, l with $0 \leq k \leq \nu, 0 \leq l \leq \delta$.

In (4.4), terms are collected with respect to the homogenizing variable z. The numerator $\widetilde{N}(z)$ and denominator $\widetilde{D}(z)$ are regarded as univariate polynomials in z with coefficients from R. The degrees of $\widetilde{N}(z)$ and $\widetilde{D}(z)$ are ν and δ, respectively. They are also the respective total degrees of p and q in f.

Now by the definition of the σ-shifted homogenization, we have

$$\widetilde{D}(z) = c \cdot q(x_1 z + \sigma_1, \ldots, x_n z + \sigma_n)$$

for some $c \neq 0$. Then

$$\widetilde{D}(0) = \widetilde{\beta}_0 = c \cdot q(\sigma_1, \ldots, \sigma_n) \neq 0.$$

Since $\widetilde{\beta}_0$ is a non-zero value, the auxiliary univariate function Γ_σ can be normalized such that the non-zero constant in the denominator is 1,

$$\begin{aligned} \Gamma_\sigma(z, x_1, \ldots, x_n) &= f(x_1 z + \sigma_1, \ldots, x_n z + \sigma_n) \\ &= \frac{\overbrace{\alpha_\nu \cdot z^\nu + \ldots + \alpha_1 \cdot z + \alpha_0}^{N(z) \in R[z]}}{\underbrace{\beta_\delta \cdot z^\delta + \ldots + \beta_1 \cdot z + 1}_{D(z) \in R[z]}} \end{aligned} \tag{4.5}$$

where $\alpha_k, \beta_l \in R$ for $0 \leq k \leq \nu,\ 1 \leq l \leq \delta$.

Let us take a look at the following example:

Example 4.1.3. Consider the rational function

$$f = \frac{x_1^3 + x_1 x_2 + x_2 x_3 + x_3^2}{x_1 + x_2} \in \mathbb{Q}(x_1, x_2, x_3)$$

Let $\sigma = (\sigma_1, \sigma_2, \sigma_3) = (2, 3, -1) \in \mathbb{Q}^3$. Since $f(\sigma)$ is defined, σ can be chosen as a shift. Then the σ-shifted homogenization of f is

$$\begin{aligned}\Gamma_\sigma(z, x_1, x_2, x_3) &= f(x_1 z + 2, x_2 z + 3, x_3 z - 1) \\ &= \frac{x_1^3 \cdot z^3 + (6x_1^2 + x_1x_2 + x_2x_3 + x_3^2) \cdot z^2 + (15x_1 + x_2 + x_3) \cdot z + 12}{(x_1 + x_2) \cdot z + 5} \\ &= \frac{\frac{1}{5}\left(x_1^3 \cdot z^3 + (6x_1^2 + x_1x_2 + x_2x_3 + x_3^2) \cdot z^2 + (15x_1 + x_2 + x_3) \cdot z + 12\right)}{\frac{1}{5}(x_1 + x_2) \cdot z + 1}.\end{aligned}$$

In this example, one can see that each coefficient of z^i is a homogeneous polynomial of degree i in the polynomial ring $\mathbb{Q}[x_1, x_2, x_3]$. In the following, we will see why it is important to shift the power basis if a given rational function is not defined at zero.

Let $\sigma = (0, 0, 0)$. Since f is not defined at σ, σ cannot be chosen as a shift. Working with this σ, nevertheless, the σ-shifted homogenization of f with respect to z is

$$\begin{aligned}\Gamma_\sigma(z, x_1, x_2, x_3) &= f(x_1 z, x_2 z, x_3 z) \\ &= \frac{x_1^3 \cdot z^3 + (x_1x_2 + x_2x_3 + x_3^2) \cdot z^2}{(x_1 + x_2) \cdot z} \\ &= \frac{x_1^3 \cdot z^2 + (x_1x_2 + x_2x_3 + x_3^2) \cdot z}{(x_1 + x_2)}.\end{aligned}$$

Here, we see that $\gcd(x_1^3 \cdot z^3 + (x_1x_2 + x_2x_3 + x_3^2) \cdot z^2, (x_1 + x_2) \cdot z) = z \neq 1$ in $(\mathbb{Q}[x_1, x_2, x_3])[z]$. The polynomial $x_1 + x_2$ is a constant term in $(\mathbb{Q}[x_1, x_2, x_3])(z)$ which we will never recover since we would expect it to be constant in $\mathbb{Q}[x_1, x_2, x_3]$, which it is not.

We now turn our attention to Equation (4.5). Once we have obtained the unique normalized auxiliary univariate rational function Γ_σ by a σ-shifted homogenization of f, we determine evaluations of the coefficients of Γ_σ, α_k and β_l as in (4.5), as follows:

Fix a point $\omega = (\omega_1, \ldots, \omega_n) \in \mathbb{Q}^n \setminus \{(0, \ldots, 0)\}$. We then pick distinct values $\eta_0, \ldots, \eta_{\nu+\delta}$ for z and evaluate $f(\omega_1\eta_j + \sigma_1, \ldots, \omega_n\eta_j + \sigma_n)$ for $0 \leq j \leq \nu + \delta$. Finally, we recover

$$\begin{aligned}\Gamma_\sigma(z, \omega) &= f(\omega_1 z + \sigma_1, \ldots, \omega_n z + \sigma_n) \\ &= \frac{\sum_{k=0}^{\nu} \alpha_k(\omega) \cdot z^k}{\sum_{l=0}^{\delta} \beta_l(\omega) \cdot z^l}\end{aligned} \tag{4.6}$$

with $\beta_0(\omega) = 1$ from the $\nu + \delta + 1$ evaluations by the early termination version of dense univariate rational interpolation (see Section 1.4 and Subsection 1.6.1). Now, the evaluations of $\alpha_0, \ldots, \alpha_\nu, \beta_1, \ldots, \beta_\delta$ at ω are just the coefficients in $\Gamma_\sigma(z, \omega)$. Once we have obtained these evaluations, the multivariate polynomials α_k and β_l can be obtained through sparse multivariate polynomial interpolation (see Section 1.5). Note that if we

homogenize the function f w.r.t. z, we get

$$f(x_1 z, \ldots, x_n z) = \frac{A_\nu \cdot z^\nu + \ldots + A_1 \cdot z + A_0}{B_\delta \cdot z^\delta + \ldots + B_1 \cdot z + B_0} \tag{4.7}$$

where $A_k, B_l \in K$ for $0 \leq k \leq \nu,\ 0 \leq l \leq \delta$. Clearly, we have

$$p = \sum_{k=0}^{\nu} A_k \text{ and } q = \sum_{l=0}^{\delta} B_l.$$

Unlike in (4.7), the polynomial coefficients α_k and β_l in (4.4) also contain terms due to the expansions of the shift σ. In this case, the number of terms becomes large, that is, the function's representation becomes dense which forces us to make additional black box probes. To address this problem, a simple but very important strategy demonstrated in [16] is to adjust the coefficients α_k and β_l for all k and l. We will now explain this in detail:

Multivariate polynomial interpolation depends on a term bound. In our case, it depends on the maximum number of terms in A_k or B_l for all k, l. For $i = 0, 1, \ldots$, we have

$$\begin{aligned}\Gamma_\sigma(z, \omega^i) &= f(\omega_1^i z + \sigma_1, \ldots, \omega_n^i z + \sigma_n) \\ &= \frac{\alpha_\nu(\omega^i) \cdot z^\nu + \alpha_{\nu-1}(\omega^i) \cdot z^{\nu-1} + \ldots + \alpha_0(\omega^i)}{\beta_\delta(\omega^i) \cdot z^\delta + \beta_{\delta-1}(\omega^i) \cdot z^{\delta-1} + \ldots + \beta_1(\omega^i) + 1}.\end{aligned}$$

Here, all A_k and B_l, for $0 \leq k \leq \nu,\ 0 \leq l \leq \delta$, can be interpolated if

$$i \geq 2 \cdot \max\left\{\{\#A_k \mid 0 \leq k \leq \nu\} \cup \{\#B_l \mid 0 \leq l \leq \delta\}\right\}$$

where $\#g$ denotes the number of terms in a polynomial g. However, for the shifted power basis this number is not usually sufficient to recover α_k and β_l. So, in order to preserve the sparsity, we need to adjust the coefficients α_k and β_l in the same way as described in [16]. Note that a shift affects neither the total degrees ν and δ nor the coefficients α_ν and β_δ of the highest degree terms in $N(z)$ and $D(z)$ (see Example 4.1.3). Moreover, only terms from the expansion of

$$c \cdot \left(\sum_{d_{k,1}+\ldots+d_{k,n}=\nu} a_k (x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}} \right)$$

can contribute to α_ν, and similarly for β_δ. Since for a fixed shift σ we require the coefficients in p and q to be normalized such that $c = 1$, we have

$$\begin{aligned}\alpha_\nu &= \sum_{d_{k,1}+\ldots+d_{k,n}=\nu} a_k x_1^{d_{k,1}} \cdots x_n^{d_{k,n}}, \\ \beta_\delta &= \sum_{e_{l,1}+\ldots+e_{l,n}=\delta} b_l x_1^{e_{l,1}} \cdots x_n^{e_{l,n}}.\end{aligned}$$

Once we have obtained evaluations of α_k and β_l at ω^i for $i = 0, 1, \ldots$, we recover the polynomials α_k and β_l in the following way:

We start with sparse multivariate interpolation of only the highest degree coefficients α_ν and β_δ. The required evaluations $\alpha_\nu(\omega^i)$ and $\beta_\delta(\omega^i)$ are obtained through repeated dense univariate rational interpolations of (4.6). Other evaluations of $\alpha_k(\omega^i)$ and $\beta_l(\omega^i)$ for $1 \leq k < \nu$ and $1 \leq l < \delta$ are recorded for later interpolations. Once both α_ν and β_δ are interpolated, we move to the next coefficients $\alpha_{\nu-1}$ and $\beta_{\delta-1}$. Since the expansion of a shifted lower degree term can never affect the representation of higher degrees, only terms in

$$\sum_{d_{k,1}+\ldots+d_{k,n}=\nu} a_k(x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}} \text{ and} \tag{4.8}$$

$$\sum_{d_{k,1}+\ldots+d_{k,n}=\nu-1} a_k(x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}} \tag{4.9}$$

can contribute to $\alpha_{\nu-1}$, which by definition collects the coefficients of $z^{\nu-1}$ in the expansion of

$$N(z) = \sum_{k=1}^{s} a_k(x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}}. \tag{4.10}$$

Now the contribution to $\alpha_{\nu-1}$ from (4.8) can be obtained as the coefficient $u_{\nu-1}^{(\nu)} \in R$ in the expansion of

$$\begin{aligned}\alpha_\nu(\theta) &= \sum_{d_{k,1}+\ldots+d_{k,n}=\nu} a_k(x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}} \\ &= u_\nu^{(\nu)} \cdot z^\nu + u_{\nu-1}^{(\nu)} \cdot z^{\nu-1} + \ldots + u_0^{(\nu)}\end{aligned}$$

where $\theta = (x_1 z + \sigma_1, \ldots, x_n z + \sigma_n)$ and $u_k^{(\nu)} \in R$ for $0 \leq k \leq \nu$. The effect of the shift in the second highest degree term can be removed accordingly:

Denote the contribution from (4.9) to $\alpha_{\nu-1}$ by $\widetilde{A}_{\nu-1} = \alpha_{\nu-1} - u_{\nu-1}^{(\nu)}$. Hence by comparing the highest degree terms in (4.9), we conclude that

$$\widetilde{A}_{\nu-1} = \sum_{d_{k,1}+\ldots+d_{k,n}=\nu-1} a_k x_1^{d_{k,1}} \cdots x_n^{d_{k,n}},$$

which now has a structure identical to that of $A_{\nu-1}$ in (4.7). Note that at this point, since we have no evaluations of $\widetilde{A}_{\nu-1}$, this is the step where we need the adjustment of the coefficient $\widetilde{A}_{\nu-1}$. Since we already have the stored evaluations $\alpha_{\nu-1}(\omega^i)$, the evaluations $\widetilde{A}_{\nu-1}(\omega^i)$ are computed as follows:

$$\widetilde{A}_{\nu-1}(\omega^i) = \alpha_{\nu-1}(\omega^i) - u_{\nu-1}^{(\nu)}(\omega^i).$$

To determine evaluations of $\widetilde{A}_{\nu-1}$ at ω^i, we do not make additional black box probes except adjusting $\widetilde{A}_{\nu-1}$ by evaluating the polynomial $u_{\nu-1}^{(\nu)}$ at ω^i outside the black box. But we only know that the number of the stored evaluations $\alpha_{\nu-1}(\omega^i)$ is sufficient for the earlier interpolations of the higher degree terms. If the number of evaluations $\alpha_{\nu-1}(\omega^i)$ does not produce enough evaluations $\widetilde{A}_{\nu-1}(\omega^i)$ for the current interpolation of $\widetilde{A}_{\nu-1}$, more evaluations of $\alpha_{\nu-1}$ can be added through new univariate rational interpolation of $\Gamma_\sigma(z, \omega^i)$ at additional i. In this way, the shifting strategy interpolates the adjusted $\widetilde{A}_\nu, \widetilde{A}_{\nu-1}, \ldots, \widetilde{A}_0$ sequentially. More precisely, we start from the highest degree term $\widetilde{A}_\nu = \alpha_\nu$ and interpolate $\widetilde{A}_\nu$. For $r = 0, \ldots, \nu-1$, after interpolating

$$\widetilde{A}_{\nu-r} = \sum_{d_{k,1}+\ldots+d_{k,n}=\nu-r} a_k x_1^{d_{k,1}} \cdots x_n^{d_{k,n}},$$

the polynomials $u_0^{(\nu-r)}, u_1^{(\nu-r)}, \ldots, u_{\nu-r-1}^{(\nu-r)} \in R$ are computed from the expansion

$$\begin{aligned} \widetilde{A}_{\nu-r}(\theta) &= \sum_{d_{k,1}+\ldots+d_{k,n}=\nu-r} a_k (x_1 z + \sigma_1)^{d_{k,1}} \cdots (x_n z + \sigma_n)^{d_{k,n}} \\ &= u_{\nu-r}^{(\nu-r)} \cdot z^{\nu-r} + u_{\nu-r-1}^{(\nu-r)} \cdot z^{\nu-r-1} + \ldots + u_0^{(\nu-r)} \end{aligned} \tag{4.11}$$

where $\theta = (x_1 z + \sigma_1, \ldots, x_n z + \sigma_n)$. The polynomials $u_0^{(\nu-r)}, u_1^{(\nu-r)}, \ldots, u_{\nu-r-1}^{(\nu-r)}$, for $0 \le r \le \nu - 1$, can be represented as an upper triangular matrix

$$A = \begin{bmatrix} u_{\nu-1}^{(\nu)} & u_{\nu-2}^{(\nu)} & u_{\nu-3}^{(\nu)} & \ldots & u_0^{(\nu)} \\ & u_{\nu-2}^{(\nu-1)} & u_{\nu-3}^{(\nu-1)} & \ldots & u_0^{(\nu-1)} \\ & & u_{\nu-3}^{(\nu-2)} & \ldots & u_0^{(\nu-2)} \\ & & & \ddots & \vdots \\ & & & & u_0^{(1)} \end{bmatrix}.$$

The sum of the polynomials (of degree $\nu - r - 1$) in the $(r+1)$-th column of this matrix is a contribution to $\alpha_{\nu-r}$ from the polynomials $\widetilde{A}_{\nu-j}$ in (4.11) for $j = 0, \ldots, r$. In other words, for $r = 1, \ldots, \nu$, the adjusted $\widetilde{A}_{\nu-r}$ is obtained by removing from $\alpha_{\nu-r}$ the contribution denoted by $U_{\nu-r}$,

$$\widetilde{A}_{\nu-r} = \alpha_{\nu-r} - U_{\nu-r}, \tag{4.12}$$

where $U_{\nu-r}$ is sum of the polynomials in the r-th column of the matrix A, that is,

$$U_{\nu-r} = \sum_{j=0}^{r-1} u_{\nu-r}^{(\nu-j)}.$$

From the evaluations of (4.12), $\widetilde{A}_{\nu-r}$ can be interpolated and then included in the adjustment for the next $\widetilde{A}_{\nu-r-1}$. We continue interpolating every newly adjusted $\widetilde{A}_{\nu-r}$ until all polynomials $\widetilde{A}_k$ are interpolated. Finally, we obtain

$$p = \sum_{r=0}^{\nu} \widetilde{A}_r.$$

The interpolation of q can be carried out in a similar way (see [16]) and, hence, we have $\frac{p}{q} = f$.

Algorithm 4.11 reconstructs a multivariate rational function given in a black box, see the sparse interpolation algorithm given in [16, Section 2.2]. In this algorithm, for any positive integer λ we denote the vector $(\omega_1^\lambda, \ldots, \omega_n^\lambda) \in \mathbb{Q}^n$ by ω^λ.

In line 9 of Algorithm 4.11, since we do not know the number of interpolation points required to recover the black box rational function f, we apply the early termination strategy discussed in Subsection 1.6.1. By applying this strategy, we obtain a rational function $\Gamma_\sigma(z, \omega^i) \in \mathbb{Q}(z)$ (see line 10) and then, from the degrees of the numerator and the denominator of this function, a bound d on the number of interpolation points, see line 12.

Recall from Section 1.5 that a bound on the number of terms is not known in advance. To handle this situation, the early termination strategy discussed in Subsection 1.6.2 (see Algorithm 1.4) can be applied. The while loop in line 14 does this step. In this loop, the rational functions $\Gamma_\sigma(z, \omega^i)$ are first generated for $i = 0, 1, 2, \ldots$ and the values of $\alpha_k(\omega^i)$ and $\beta_l(\omega^i)$ for $0 \le k \le \nu-1$ and $0 \le l \le \delta-1$ are stored for the later interpolations. If the stored values $\alpha_{\nu-r}(\omega^i)$ or $\beta_{\delta-r}(\omega^i)$ are not sufficient for interpolating either $\widetilde{A}_{\nu-r}$ or $\widetilde{B}_{\delta-r}$ (see line 18), we increase i (see line 15) and continue the interpolation of $\widetilde{A}_{\nu-r}$ and/or $\widetilde{B}_{\delta-r}$ by adding more interpolation points. Once we interpolate all $\widetilde{A}_{\nu-r}$ and $\widetilde{B}_{\delta-r}$ (see line 27), we therefore obtain the desired polynomial given in a black box. Algorithm 4.11 returns an error message only if some unlucky (or bad) evaluation points are used in the computation otherwise it returns the correct result with high probability.

4.2 Illustration by Example

This section presents an example illustrating how the reconstruction method discussed above works.

Example 4.2.1. Consider the rational function

$$f = \frac{x_1^3 + x_1x_2 + x_2x_3 + x_3^2}{x_1 + x_2} \in \mathbb{Q}(x_1, x_2, x_3).$$

Algorithm 4.11 Sparse Rational Interpolation

Input: A multivariate black box rational function $f(x_1, \ldots, x_n)$ and a positive integer ζ (1 by default), the threshold required by the early termination strategy.

Output: $a_k, b_l \in \mathbb{Q}$ and $(d_{k,1}, \ldots, d_{k,n}), (e_{l,1}, \ldots, e_{l,n}) \in \mathbb{N}^n$ such that

$$f(x_1, \ldots, x_n) = \frac{\sum_{k=1}^{s} a_k x_1^{d_{k,1}} \cdots x_n^{d_{k,n}}}{\sum_{l=1}^{t} b_l x_1^{e_{l,1}} \cdots x_n^{e_{l,n}}} \in \mathbb{Q}(x_1, \ldots, x_n) \text{ with high probability,}$$

or an error message if the procedure fails to complete.

choose a random non-zero point $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{Q}^n$ such that $f(\sigma)$ is defined
$i \longleftarrow 0, (\omega_1, \ldots, \omega_n) \longleftarrow (p_1, \ldots, p_n)$ where $p_1, \ldots, p_n$ are distinct primes
choose a random non-zero element $\eta_0 \in \mathbb{Q}$ and set $\mathcal{B} := \{\eta_0\}$
$j \longleftarrow 0,\ \Gamma_{\sigma,j}(z, \omega^i) \longleftarrow \frac{\eta_j}{1}$
while (TRUE) **do**
  $j \longleftarrow j + 1$
  choose a random non-zero element $\eta_j \in \mathbb{Q} \setminus \mathcal{B}$ such that $f(\omega_1^i \eta_j + \sigma_1, \ldots, \omega_n^i \eta_j + \sigma_n) =: y_j$ is defined
  $\mathcal{B} \longleftarrow \mathcal{B} \cup \{\eta_j\}$
  call the dense univariate rational interpolation algorithm (see Section 1.4) with inputs $((\eta_0, \ldots, \eta_j), (y_0, \ldots, y_j))$ to obtain a rational function $\Gamma_{\sigma,j}(z, \omega^i) = f(\omega_1^i z + \sigma_1, \ldots, \omega_n^i z + \sigma_n) = \frac{\sum_{k=0}^{\nu_j} \alpha_{k,j}(\omega^i) \cdot z^k}{\sum_{l=0}^{\delta_j} \beta_{l,j}(\omega^i) \cdot z^l}$
  if $\Gamma_{\sigma,j-1}(z, \omega^i) = \Gamma_{\sigma,j}(z, \omega^i)$ **then**
    break
$d \longleftarrow \nu_j + \delta_j + 1 + \zeta,\ \nu \longleftarrow \nu_j, \delta \longleftarrow \delta_j,\ p \longleftarrow 0,\ q \longleftarrow 0$
$U_\nu \longleftarrow U_{\nu-1} \longleftarrow \cdots \longleftarrow U_0 \longleftarrow 0,\ V_\delta \longleftarrow V_{\delta-1} \longleftarrow \cdots \longleftarrow V_1 \longleftarrow 0$
while (TRUE) **do**
  $i \longleftarrow i + 1$
  choose $d + 1$ distinct random points $\eta_0, \ldots, \eta_d \in \mathbb{Q}$ such that, for $0 \le j \le d$, $f(\omega_1^i \eta_j + \sigma_1, \ldots, \omega_n^i \eta_j + \sigma_n) = y_j$ is defined. From these evaluations interpolate

$$\Gamma_\sigma(z, \omega^i) = \frac{\sum_{k=0}^{\nu} \alpha_k(\omega^i) \cdot z^k}{\sum_{l=0}^{\delta} \beta_l(\omega^i) \cdot z^l}$$

  for $r = 0, \ldots, \max(\nu, \delta)$ **do**
    call Algorithm 1.4 to interpolate $\widetilde{A}_{\nu-r}$ and $\widetilde{B}_{\delta-r}$ from the evaluations $\widetilde{A}_{\nu-r}(\omega^i) = \alpha_{\nu-r}(\omega^i) - U_{\nu-r}(\omega^i),\ \widetilde{B}_{\delta-r}(\omega^i) = \beta_{\delta-r}(\omega^i) - V_{\delta-r}(\omega^i)$
    if both $\widetilde{A}_{\nu-r}$ and $\widetilde{B}_{\delta-r}$ interpolated without error message **then**
      $p \longleftarrow p + \widetilde{A}_{\nu-r},\ q \longleftarrow q + \widetilde{B}_{\delta-r}$
      $\widetilde{A}_{\nu-r}(x_1 z + \sigma_1, \ldots, x_n z + \sigma_n) \longleftarrow \sum_{j=0}^{\nu-r} u_j^{\nu-r}(x_1, \ldots, x_n) \cdot z^j$
      $\widetilde{B}_{\delta-r}(x_1 z + \sigma_1, \ldots, x_n z + \sigma_n) \longleftarrow \sum_{j=0}^{\delta-r} v_j^{\delta-r}(x_1, \ldots, x_n) \cdot z^j$
      for $j = r, \ldots, \max(\nu, \delta - 1)$ **do**
        $U_{\nu-j} \longleftarrow U_{\nu-j} + u_j^{(\nu-r)},\ V_{\delta-j} \longleftarrow V_{\delta-j} + v_j^{(\nu-r)}$
    else
      FAIL
  if $r = \max(\nu, \delta)$ **then**
    break
return $\frac{p}{q}$

Recall from Example 4.1.3 that, for $\sigma = (2, 3, -1)$,

$$\begin{aligned}\Gamma_\sigma(z, x_1, x_2, x_3) &= f(x_1 z + 2, x_2 z + 3, x_3 z - 1) \\ &= \frac{x_1^3 \cdot z^3 + (6x_1^2 + x_1x_2 + x_2x_3 + x_3^2) \cdot z^2 + (15x_1 + x_2 + x_3) \cdot z + 12}{(x_1 + x_2) \cdot z + 5} \\ &= \frac{\frac{1}{5}(x_1^3 \cdot z^3 + (6x_1^2 + x_1x_2 + x_2x_3 + x_3^2) \cdot z^2 + (15x_1 + x_2 + x_3) \cdot z + 12)}{\frac{1}{5}(x_1 + x_2) \cdot z + 1}.\end{aligned}$$

Since $\nu = 3$ and $\delta = 1$, we choose $3 + 1 + 1 + 1 = 6$ distinct values for z. We first fix the point $(x_1, x_2, x_3) = (2^0, 3^0, 5^0) = (1, 1, 1) \in \mathbb{Q}^3$. Then by dense univariate rational interpolation (see Section 1.4), we obtain the auxiliary rational function

$$\Gamma_\sigma(z, 2^0, 3^0, 5^0) = \frac{1/5z^3 + 9/5z^2 + 17/5z + 12/5}{2/5z + 1} \tag{4.13}$$

from its evaluation at the distinct values $1, 2, 3, 4, 5, 6$ for z. Note that since we do not know the bounds ν and δ, we use the early termination methods discussed in Subsection 1.6.1 to find the rational function $\Gamma_\sigma(z, 1, 1, 1) \in \mathbb{Q}(z)$. The degree $\nu = 3$ (resp. $\delta = 1$) of the polynomial in the numerator (resp. denominator) of this function gives an information on the number of interpolation points which are required to generate such rational functions in the later interpolations for $i = 1, 2, \ldots$. In fact, the number of interpolation points that we need is at least $\nu + \delta + 1 + \zeta$, $\zeta \geq 1$. Note that the coefficients of the auxiliary rational function $\Gamma_\sigma(z, 2^0, 3^0, 5^0)$ are the same as in the evaluation of

$$\Gamma_\sigma(z, x_1, x_2, x_3) = \frac{\frac{1}{5}(x_1^3 \cdot z^3 + (6x_1^2 + x_1x_2 + x_2x_3 + x_3^2) \cdot z^2 + (15x_1 + x_2 + x_3) \cdot z + 12)}{\frac{1}{5}(x_1 + x_2) \cdot z + 1} \tag{4.14}$$

at the point (1,1,1) fixed above. Since we have already obtained a bound on the number of interpolation points with high probability, we do not need to apply the early termination strategy for the remaining tasks regarding rational interpolations. Instead, we apply the dense rational interpolation algorithm discussed in Section 1.4. Now by generating, for each $i = 1, 2, \ldots$, a rational function of degree 3 (resp. 1) in the numerator (resp. denominator) as above, we apply the early termination strategy to recover the coefficients in $\mathbb{Q}[x_1, x_2, x_3]$ of z^k for $0 \leq k \leq \max\{\nu, \delta\} = 3$. But for simplicity, we generate, for

$i = 1, 2, \ldots, 6$, a total of 6 such auxiliary rational functions as in the following:

$$\begin{aligned}
\Gamma_\sigma(z, 2, 3, 5) &= \frac{8/5z^3 + 14z^2 + 38/5z + 12/5}{z + 1}, \\
\Gamma_\sigma(z, 2^2, 3^2, 5^2) &= \frac{64/5z^3 + 982/5z^2 + 94/5z + 12/5}{13/5z + 1}, \\
\Gamma_\sigma(z, 2^3, 3^3, 5^3) &= \frac{512/5z^3 + 3920z^2 + 272/5z + 12/5}{7z + 1}, \\
\Gamma_\sigma(z, 2^4, 3^4, 5^4) &= \frac{4096/5z^3 + 444082/5z^2 + 946/5z + 12/5}{97/5z + 1}, \\
\Gamma_\sigma(z, 2^5, 3^5, 5^5) &= \frac{32768/5z^3 + 2107784z^2 + 3848/5z + 12/5}{55z + 1}, \text{ and} \\
\Gamma_\sigma(z, 2^6, 3^6, 5^6) &= \frac{262144/5z^3 + 255602482/5z^2 + 17314/5z + 12/5}{793/5z + 1}.
\end{aligned}$$

To reconstruct f from the coefficients of powers of z, we follow the steps discussed above. As it is explained in this chapter, we start by recovering the coefficients of the highest degree term in the numerator of $\Gamma_\sigma(z, 2^i, 3^i, 5^i)$ for $i = 0, 1, \ldots, 6$, that is, the coefficients of z^3. These coefficients, respectively, are:

$$(a_0, \ldots, a_6) = (1/5, 8/5, 64/5, 512/5, 4096/5, 32768/5, 262144/5).$$

By applying Algorithm 1.4 to these evaluations (as input), we get the polynomial $\widetilde{A}_3 = \alpha_3 = 1/5x_1^3 \in \mathbb{Q}[x_1, x_2, x_3]$. From this result we see that only two evaluation points, $1/5$ and $8/5$, would be enough to lift the coefficients of z^3 to $\mathbb{Q}[x_1, x_2, x_3]$.

The next step is now to recover the coefficient $\widetilde{A}_2 \in \mathbb{Q}[x_1, x_2, x_3]$ of z^2. Recall from Section 1.5 that the number of terms required to recover the polynomial $\alpha_2 = 6x_1^2 + x_1x_2 + x_2x_3 + x_3^2$ (the coefficient of z^2 in the numerator of $\Gamma_\sigma(z, x_1, x_2, x_3)$ as in (4.14)) is at least 8. Although we have only 7 evaluation points which is not sufficient, we know that the term $6x_1^2$ in α_2 is an additional term due to the expansion of the shift σ. Thus in order to determine the coefficient $\widetilde{A}_2$ of z^2, first we have to evaluate $\widetilde{A}_3$ at $\theta = (x_1z + 2, zx_2 + 3, x_3z - 1)$ to obtain

$$\widetilde{A}_3(\theta) = 1/5x_1^3 \cdot z^3 + 6/5x_1^2 \cdot z^2 + 12/5x_1 \cdot z + 8/5.$$

Then for $i = 0, 1, \ldots, 6$, by evaluating the polynomial $U_2 = 6/5x_1^2$ at $(x_1, x_2, x_3) = (2^i, 3^i, 5^i)$ outside the black box, we obtain the evaluation of the unknown polynomial $\widetilde{A}_2$ at these points as follows:

$$\widetilde{A}_2(2^i, 3^i, 5^i) = \alpha_2(2^i, 3^i, 5^i) - U_2(2^i, 3^i, 5^i).$$

Since the polynomial $\alpha_2 - U_2$ needs only 6 evaluation points, the number of the stored evaluations is sufficient to recover the polynomial $\widetilde{A}_2$. By applying Algorithm 1.4 to these evaluations (as input), we obtain the polynomial $\widetilde{A}_2 = x_1x_2 + x_2x_3 + x_3^2 \in \mathbb{Q}[x_1, x_2, x_3]$.

Continuing in this way, the adjusted $\widetilde{A}_{\nu-r}$ (resp. $\widetilde{B}_{\delta-r}$) is obtained by removing from $\alpha_{\nu-r}$ (resp. $\beta_{\delta-r}$) the contribution

$$U_{\nu-r} = \sum_{j=0}^{r-1} u_{\nu-r}^{(\nu-j)} \left(\text{resp. } V_{\delta-r} = \sum_{j=0}^{r-1} v_{\delta-r}^{(\delta-j)} \right),$$

given as upper triangular matrices

$$\begin{bmatrix} u_2^{(3)} & u_1^{(3)} & u_0^{(3)} \\ & u_1^{(2)} & u_0^{(2)} \\ & & u_0^{(1)} \end{bmatrix} = \begin{bmatrix} \frac{6}{5}x_1^2 & \frac{12}{5}x_1 & \frac{8}{5} \\ & \frac{1}{5}(3x_1 + x_2 + x_3) & \frac{4}{5} \\ & & 0 \end{bmatrix}, \quad \left[v_0^{(1)} \right] = \left[1 \right],$$

for $r = 1, \ldots, \max\{\nu, \delta\} = 3$. The contributions are obtained from the expansion of $\widetilde{A}_{\nu-r-1}(x_1 z + 2, z x_2 + 3, x_3 z - 1)$ $\Big(\text{resp. } \widetilde{B}_{\delta-r-1}(x_1 z + 2, z x_2 + 3, x_3 z - 1)\Big)$, $0 \leq r \leq 2$, at each interpolation step. Finally, we obtain the rational function

$$f = \frac{1/5 x_1^3 + 1/5(x_1 x_2 + x_2 x_3 + x_3^2)}{1/5(x_1 + x_2)},$$

as desired.

Chapter 5

Gröbner Bases over Algebraic Function Fields

Computing Gröbner bases is a technique that provides algorithmic solutions to a variety of problems in commutative algebra and algebraic geometry. From the theoretical point of view, such a basis can be computed over any field using Buchberger's algorithm. However, the computational efficiency depends on the coefficient field. Consider an algebraic function field K over $\mathbb{Q}$. Like for algebraic number fields, the arithmetic operations in K usually make the computation of Gröbner bases inefficient if they are used directly. During the computation of Gröbner bases using Buchberger's algorithm, in contrast to computations over algebraic number fields (see Chapter 2), there is no minimal polynomial with coefficients in $\mathbb{Q}$ that bounds the degrees of elements of K. The problems are now twofold: We have to tackle the coefficient swell in $\mathbb{Q}$ and the problems that arise from the arithmetic of K. These two problems usually make the computation very slow. For the former problem, various methods to avoid this have been investigated; the trace algorithm [40] and modular algorithms [2, 27] are successful in this direction. Unfortunately, the problems that arise from the arithmetic of K cannot be solved by these methods, see, for example, Example 5.2.1. Brickenstein [10] has introduced a variation of Buchberger's algorithm [11] to compute Gröbner bases over algebraic function fields using so-called slim polynomials. The algorithm is designed to keep coefficients small and polynomials short in the intermediate computations. This method, however, is limited in terms of efficiency, especially when the reduced Gröbner basis of the input ideal has sparse coefficients.

In this chapter we present a new efficient method to compute Gröbner bases over an algebraic function field. The new method uses the concept of sparse multivariate rational interpolation. To state the ideas more precisely, let $K = \mathbb{Q}(t_1, \ldots, t_m)$ and consider the ring $K[x_1, \ldots, x_n]$ of multivariate polynomials in n variables $x_1, \ldots, x_n$ whose coefficients are rational functions of the symbolic parameters $t_1, \ldots, t_m$ with coefficients in $\mathbb{Q}$. Let $H = \{f_1, \ldots, f_w\} \subseteq K[x_1, \ldots, x_n]$, and let

$$I = \left\langle f_i = \sum_{\alpha=(\alpha_1,\ldots,\alpha_n)\in\mathbb{N}^n} c_{\alpha,i} x_1^{\alpha_{1,i}} \cdots x_n^{\alpha_{n,i}} \;\middle|\; c_{\alpha,i} \in K,\ 1 \leq i \leq w \right\rangle \subseteq K[x_1, \ldots, x_n]$$

be the ideal generated by H. By a *specialization of parameters* we mean a choice

of substituting each symbolic parameter t_i by a particular element b_i of $\mathbb{Q}$. Once a convenient choice of specialization has been made, we obtain an ideal

$$I_b = \left\langle f_{i,b} = \sum_{\alpha \in \mathbb{N}^n} c_{\alpha,i}(b) x_1^{\alpha_{1,i}} \cdots x_n^{\alpha_{n,i}} \;\middle|\; c_{\alpha,i}(b) \in \mathbb{Q},\ 1 \leq i \leq w \right\rangle$$

in $\mathbb{Q}[x_1, \ldots, x_n]$ where $b = (b_1, \ldots, b_m)$. When we choose the b_i from $\mathbb{Q}$, however, we have to be careful since the leading term of $f_{i,b}$ w.r.t. a given monomial ordering on $\mathrm{Mon}(x_1, \ldots, x_n)$ may change or the $f_{i,b}$ may be zero. In the polynomial ring over $\mathbb{Q}$, we compute the reduced Gröbner bases of I_b for different b's as many times as necessary, see Section 5.3. Once we have computed a sufficiently large set of Gröbner bases in the polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$, we recover the coefficients in K of the reduced Gröbner basis of I using the sparse rational interpolation algorithm discussed in Chapter 4.

This chapter is organized as follows: In Section 5.1, we introduce some notation which is used throughout this chapter. An overview of the new method is outlined in Section 5.2. Here we also give an example motivating our new method. The core part of the proposed algorithm is discussed in Section 5.3. Since the new method relies on the sparse rational interpolation algorithm from Chapter 4, we also give a more detailed explanation on how to apply this algorithm coefficient-wise to the sets of polynomials in a polynomial ring with coefficients in $\mathbb{Q}$. In Subsection 5.3.1, we explain how we choose the evaluation points. An illustrating example is given in Subsection 5.3.2 whereas timings comparing the new algorithm to other approaches are presented in Subsection 5.3.3. In Section 5.4, we introduce a special case algorithm for computing Gröbner bases of ideals over fields of rational functions with one variable. An illustrating example for this special case is presented in Subsection 5.4.1. Timings comparing the algorithms in Section 5.3 and Section 5.4 are presented in Subsection 5.4.2. Further optimizations are discussed in Section 5.5. In Subsection 5.5.1, we present timings comparing the algorithm discussed in Section 5.3 to the one presented in Section 5.5. Here we also give remarks on the implementation of the new algorithm in SINGULAR [17] and overall timings comparing it to other approaches. The benchmark problems which we use for the timings are listed in the appendix, see Section A.3.

5.1 Notation

Let $T = \{t_1, \ldots, t_m\}$ be a set of symbolic parameters. Throughout this chapter we work over the algebraic function field $K = \mathbb{Q}(T)$ (see Definition 5.3.1), that is, over the field of rational functions in $t_1, \ldots, t_m$ with coefficients in $\mathbb{Q}$. Furthermore, consider the m-dimensional affine space

$$\mathbb{A}^m(\mathbb{Q}) = \{(a_1, \ldots, a_m) \mid a_1, \ldots, a_m \in \mathbb{Q}\}$$

over $\mathbb{Q}$. Polynomials are related to the affine space in the following sense. The idea is that a polynomial $f \in \mathbb{Q}[T]$ yields a function

$$\begin{aligned} f : \mathbb{A}^m(\mathbb{Q}) &\longrightarrow \mathbb{Q}, \\ (a_1, \ldots, a_m) &\longmapsto f(a_1, \ldots, a_m). \end{aligned} \tag{5.1}$$

A *specialization of parameters* is a well-defined choice of an element b_i of $\mathbb{Q}$ for each symbolic parameter t_i. Let us denote such a selection by a mapping σ from T to the affine space $\mathbb{A}^m(\mathbb{Q})$, that is,

$$\begin{aligned} \sigma : T &\longrightarrow \mathbb{Q}, \\ t_i &\longmapsto b_i. \end{aligned}$$

Consider the ring $R_{\langle t_1 - b_1, \ldots, t_m - b_m \rangle} \subseteq \mathbb{Q}(T)$ of rational functions whose denominators do not vanish at $(b_1, \ldots, b_m)$, that is,

$$R_{\langle t_1 - b_1, \ldots, t_m - b_m \rangle} = \left\{ \frac{p}{q} \in \mathbb{Q}(T) \;\middle|\; p \in \mathbb{Q}[T],\ q \in \mathbb{Q}[T] \setminus \{0\} \text{ with } q(b_1, \ldots, b_m) \neq 0 \right\}.$$

The map σ can naturally be extended to ring homomorphisms, denoted also by σ,

$$\sigma : \mathbb{Q}[T] \longrightarrow \mathbb{Q} \ \text{ and } \sigma : R_{\langle t_1 - b_1, \ldots, t_m - b_m \rangle} \longrightarrow \mathbb{Q} \tag{5.2}$$

defined as follows: For $f = \sum_\alpha c_\alpha T^\alpha \in \mathbb{Q}[T]$ with $\alpha \in \mathbb{N}^m$, we have $\sigma(f) := \sum_\alpha c_\alpha \sigma(T)^\alpha$ where $T^\alpha = t_1^{\alpha_1} \cdots t_m^{\alpha_m}$ (a monomial of symbolic parameters) and $\sigma(T^\alpha) = \sigma(T)^\alpha$, and for $f = p/q \in \mathbb{Q}(T)$ with $\sigma(q) \neq 0$, we have $\sigma(f) = \sigma(p)/\sigma(q)$. Thus σ is just the evaluation map.

Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. Consider the polynomial ring $S = K[X] = \mathbb{Q}(T)[X]$. Fix a global monomial ordering $>$ on the monoid of monomials $\mathrm{Mon}(X)$. Let $H = \{f_1, \ldots, f_w\}$ be a subset of S, and let $I \subseteq S$ be the ideal generated by H. We assume that the elements of H are monic w.r.t. $>$. Let G be the reduced Gröbner basis of I w.r.t. $>$, and for any coefficient c (in lowest terms) occurring in G, let c_N and c_D be the numerator and the denominator of c, respectively, considered as polynomials in $\mathbb{Q}[T]$. With this notation, set

$$\begin{aligned} d_N &:= \max\{d \in \mathbb{N} \mid d = \deg(c_N),\ c \text{ a coefficient occurring in } G\}, \\ d_D &:= \max\{d \in \mathbb{N} \mid d = \deg(c_D),\ c \text{ a coefficient occurring in } G\}, \\ d &:= d_N + d_D + 2\,, \end{aligned} \tag{5.3}$$

and

$$\tau := \max\left\{ \tau' \in \mathbb{N} \;\middle|\; \begin{array}{c} \tau' \text{ is the number of terms either in } c_N \\ \text{or } c_D, c \text{ a coefficient occurring in G} \end{array} \right\}. \tag{5.4}$$

Thus the integer d_N (resp. d_D) is a bound on the degrees of the numerator (resp. denominator) of the coefficients occurring in G, and the integer d is a bound on the number of interpolation points required for recovering the coefficients occurring in G. Furthermore, the integer τ is a bound on the number of terms in the denominator and in the numerator of any coefficient occurring in G.

5.2 Overview of the New Method

In this section we present an overview of our new approach. Let us start by taking a quick look at the following example:

Example 5.2.1. Consider the ideal $I \subseteq \mathbb{Q}(t_1, t_2)[x_1, x_2, x_3]$ generated by the following polynomials:

$$\begin{aligned}
f_1 &= x_1^2x_2^3x_3 + 2t_1x_1x_2x_3^2 + 7x_2^3,\\
f_2 &= x_1^2x_2^4x_3 + (t_1 - 7t_2)x_1^2x_2x_3^2 - x_1x_2^2x_3^2 + 2x_1^2x_2x_3 - 12x_1 + t_2x_2,\\
f_3 &= (t_1^2 + t_2 - 2)x_2^5x_3 + (t_1 + 5t_2)x_1^2x_2^2x_3 - t_2x_1x_2^3x_3 - x_1x_2^3 + x_2^4 + 2t_1^2x_2^2x_3,\\
f_4 &= t_1x_1^2x_2^2x_3 - x_1x_2^3x_3 + (-t_1 + 4)x_2^3x_3^2 + 3t_1x_1x_2x_3^3 + 4x_3^2 - t_2x_1.
\end{aligned}$$

The reduced Gröbner basis G of I w.r.t. the degree reverse lexicographical ordering (`dp` in SINGULAR) with $x_1 > x_2 > x_3$ is

$$G = \{g_1, g_2, g_3, g_4\},$$

where

$$g_1 = x_1 - \frac{1}{12}t_2x_2,\ g_2 = x_3^2 - \frac{1}{48}t_2^2x_2,\ g_3 = x_2^2x_3,\ \text{ and } g_4 = x_2^3.$$

In this example, the set G has sparse coefficients. Although it is hard to compute the basis G of I using Buchberger's algorithm directly over $\mathbb{Q}(t_1, t_2)$, we will see later in Section 5.3.2 that it can easily be computed using our new method. In fact, during the computation of the reduced Gröbner basis of I using Buchberger's algorithm, we observe that the coefficients in $\mathbb{Q}(t_1, t_2)$ of the intermediate polynomials grow to an enormous size both with regard to the number of terms and the total degrees even though the coefficients of the polynomials of the input ideal and the Gröbner basis are relatively sparse. For example, when computing the reduced Gröbner basis of I modulo $p = 499$, in the intermediate computations we observe a polynomial in $\mathbb{F}_p(t_1, t_2)[x_1, x_2, x_3]$ whose number of terms is 88. The number of terms of the numerator (resp. denominator) of one of the coefficients $c = \frac{f}{g} \in \mathbb{F}_p(t_1, t_2)$ of this polynomial is 105 (resp. 46). Furthermore, the degree of f (resp. g) is 20 (resp. 15). As mentioned in the introduction, this shows that the degree of the numerator or denominator of this function cannot be controlled modulo p. Moreover, the reduced Gröbner basis computation of this ideal modulo p using Buchberger's algorithm took more than 24 hours. This means that modular algorithms alone cannot solve the problems that arise from the arithmetic of K. One can also easily see that the representation of the function $\frac{f}{g}$ as compared to the coefficients in the output is very dense. Note that there is no element of G whose number of terms is more than 2. Thus this polynomial must be one of the superfluous (or redundant) elements in the intermediate computations. The reduced Gröbner basis of the above ideal can be

computed within less than 3 seconds using our new method. With this motivation, we now give an overview of our new approach.

Our new method computes the reduced Gröbner basis of a given ideal $I = \langle H \rangle \subseteq K[X]$ as follows: First, we choose a suitable non-zero evaluation point $s = (s_1, \ldots, s_m)$ from $\mathbb{Q}^m$. Note that the bounds d and τ defined in Notation 5.1 are not known in advance. Nevertheless, we will discuss in the next section how to find the bound d with high probability. In fact, this bound can be determined by fixing one point. The idea is as follows: We first fix a point $b = (p_1, \ldots, p_m) \in \mathbb{Q}^m$ where the p_i's are distinct primes (recall that the Ben-Or/Tiwari algorithm uses distinct primes for $t_1, \ldots, t_m$ to recover sparse multivariate polynomial given in a black box, see Section 1.5). Let z be an extra variable, and consider the subring $\mathbb{Q}(t_1 z, \ldots, t_m z)$ of $\mathbb{Q}(t_1, \ldots, t_m, z)$ obtained as image of the map

$$\phi : \mathbb{Q}(t_1, \ldots, t_m) \longrightarrow \mathbb{Q}(t_1, \ldots, t_m, z),\ t_i \longmapsto t_i z\,.$$

Set $Tz := (t_1 z, \ldots, t_m z)$ and $\mathbb{Q}(Tz) := \mathbb{Q}(t_1 z, \ldots, t_m z)$. For any polynomial $g \in S$, define $\Gamma_{g,s}$ by

$$\Gamma_{g,s}(T, z, X) := g(t_1 z + s_1, \ldots, t_m z + s_m, X) \in \mathbb{Q}(Tz)[X].$$

We choose suitable random elements $z_0, z_1, \ldots$ from $\mathbb{Q}$ such that the points $(p_1 z_i + s_1, \ldots, p_m z_i + s_m) \in \mathbb{Q}^m$ satisfy the conditions in Definition 5.3.3. With these evaluation points, we obtain the ideals

$$I_i = \langle \Gamma_{g,s}(b, z_i, X) \mid g \in H \rangle$$

in the polynomial ring $\mathbb{Q}[X]$. Let G_i be the reduced Gröbner basis of I_i. By applying the early termination version of the dense univariate rational interpolation algorithm (see Subsection 1.6.1) coefficient-wise to the sets of polynomials G_i (ordered in such a way that corresponding polynomials in each set have the same leading monomials), we obtain a set G_b of polynomials with coefficients in $\mathbb{Q}(z)$. From the set G_b, we determine the bounds d, d_N and d_D defined in (5.3) with high probability. With respect to these bounds, after computing such G_b's as many times as necessary and applying the sparse rational interpolation algorithm (see Algorithm 1.3) coefficient-wise to the sets of polynomials in G_b (ordered in such a way that corresponding polynomials in each set have the same leading monomials), we obtain a set G' of polynomials with coefficients in $\mathbb{Q}(T)$. Finally, we verify that the set G' of polynomials is indeed a reduced Gröbner basis of the input ideal with high probability. A detailed description will be given in the next section.

5.3 Gröbner Bases Using Sparse Rational Interpolation

We start with the following definition:

Definition 5.3.1. An *algebraic function field* K in m variables $t_1, \ldots, t_m$ over a field k is a field extension K over k such that K is a finite extension of $k(t_1, \ldots, t_m)$. That is, $[K : k(t_1, \ldots, t_m)] < \infty$.

In this chapter, we only consider the field of rational functions $K = \mathbb{Q}(t_1, \ldots, t_m)$ in m variables which is an algebraic function field over $\mathbb{Q}$.

Definition 5.3.2. Let k be a field. For $f \in k[t_1, \ldots, t_m]$, a point $a = (a_1, \ldots, a_m) \in \mathbb{A}^m(k)$ with $f(a) = f(a_1, \ldots, a_m) = 0$ is called a *zero* of f and

$$Z(f) = \{a \in \mathbb{A}^m(k) \mid f(a) = 0\}$$

is called the *zero set* or *zero locus* of f.

Recall from Section 5.2 that we have introduced the new ring $\mathbb{Q}(Tz)$. Denote the polynomial ring $\mathbb{Q}[Tz]$ by R. Now for any polynomial g in $\mathbb{Q}(Tz)[X]$, there exists a polynomial h_g in R such that $g \cdot h_g \in R[X]$. Consider the map

$$\begin{aligned} \varphi : \mathbb{Q}(Tz)[X] &\longrightarrow R[X], \\ g &\longmapsto g \cdot h_g. \end{aligned} \tag{5.5}$$

In this map, we may take h_g to be the least common multiple of the denominators in the coefficients of g. Let $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ be a point such that none of the denominators in the coefficients of H vanishes when evaluated at $T = s$. For a polynomial $g = \sum_{\alpha \in \mathbb{N}^n} c_\alpha(T) X^\alpha \in H$ with $c_\alpha \in \mathbb{Q}(T)$, let $g_{z,s}$ be the polynomial (considered as an auxiliary polynomial) defined by

$$g_{z,s} := \sum_{\alpha \in \mathbb{N}^n} c_\alpha(t_1 z + s_1, \ldots, t_m z + s_m) X^\alpha \in \mathbb{Q}(Tz)[X],$$

with coefficients (considered as auxiliary rational functions, see Definition 4.1.1) in $\mathbb{Q}(Tz)$. Let $\widetilde{H} = \{g_{z,s} \mid g \in H\}$. Let $\widetilde{I}$ be the ideal generated by $\widetilde{H}$, that is,

$$\widetilde{I} = \langle \widetilde{H} \rangle = \langle g_{z,s} \mid g \in H \rangle \subseteq \mathbb{Q}(Tz)[X], \tag{5.6}$$

and call this ideal an auxiliary ideal. Let J be the set of coefficients c of

$$\varphi(\widetilde{H}) = \{h' \in R[X] \mid h' = g \cdot h_g \text{ with } g \in \widetilde{H} \text{ and } h_g \in R\}$$

with $\deg(c) > 0$, that is,

$$J = \{c \in R \mid c \text{ is a coefficient in } \varphi(\widetilde{H}) \text{ with } \deg(c) > 0\}. \tag{5.7}$$

Let $\widetilde{b} = (b, z_0)$ be a non-zero point in $\mathbb{Q}^{m+1}$ with $z_0 \in \mathbb{Q}$ such that none of the denominators in the coefficients of $\widetilde{H}$ vanishes when evaluated at $(T, z) = (b, z_0)$. Consider the map

$$\begin{aligned} \sigma_{\widetilde{b}} : \mathbb{Q}(Tz)[X] \supseteq \widetilde{H} &\longrightarrow \mathbb{Q}[X], \\ \widetilde{g}(T, z, X) &\longmapsto \widetilde{g}(b, z_0, X). \end{aligned} \tag{5.8}$$

We denote the set $\{\sigma_{\widetilde{b}}(\widetilde{g}) \mid \widetilde{g} \in \widetilde{H}\}$ of polynomials in $\mathbb{Q}[X]$ by $\sigma_{\widetilde{b}}(\widetilde{H})$ and define $I_{\widetilde{b}} := \sigma_{\widetilde{b}}(\widetilde{I})$ to be the ideal generated by $\sigma_{\widetilde{b}}(\widetilde{H})$. For our algorithm, although the evaluation points are random, they need to be chosen carefully. Let V be the union of the zero set of the elements in J, that is,

$$V = \bigcup_{f \in J} Z(f).$$

The notion of *admissible evaluation points* w.r.t. $\widetilde{H}$ is defined as follows:

Definition 5.3.3. Let z_0 be a non-zero element in $\mathbb{Q}$ and let $p_1, \ldots, p_m \in \mathbb{Q}$ be distinct primes. Moreover, let $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ be a point such that none of the denominators in the coefficients of H vanishes when evaluated at $T = s$. The point $\widetilde{b} = (p_1, \ldots, p_m, z_0) \in \mathbb{Q}^{m+1}$ is said to be an *admissible evaluation point* w.r.t. $\widetilde{H}$ if $\widetilde{b} \notin V$ (or, equivalently, if none of the elements in J is contained in the ideal $\langle t_1 - p_1, \ldots, t_m - p_m, z - z_0 \rangle$).

The conditions in this definition enable us to choose well-defined evaluation points from $\mathbb{Q}^{m+1}$. For the choice of evaluation points, see Subsection 5.3.1 where we explain to which extend the algorithms discussed in this chapter are probabilistic.

The notion of *lucky evaluation points* w.r.t. $\widetilde{H}$ is defined as follows:

Definition 5.3.4. Let $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ be a point such that none of the denominators in the coefficients of H vanishes when evaluated at $T = s$. Let z_0 be a non-zero element in $\mathbb{Q}$ and let $p_1, \ldots, p_m \in \mathbb{Q}$ be distinct primes. Suppose that the point $\widetilde{b} = (b, z_0) \in \mathbb{Q}^{m+1}$ with $b = (p_1, \ldots, p_m)$ is an admissible evaluation point w.r.t. $\widetilde{H}$, see Definition 5.3.3. Let G be the reduced Gröbner basis of I, and let $G_{\widetilde{b}}$ be the reduced Gröbner basis of the ideal $I_{\widetilde{b}} = \langle \sigma_{\widetilde{b}}(\widetilde{g}) \mid \widetilde{g} \in \widetilde{H} \rangle$ where $\sigma_{\widetilde{b}}$ is defined as in (5.8). Then the point $\widetilde{b}$ is called a *lucky evaluation point* for $\widetilde{I}$ if and only if $\mathrm{Lm}(G) = \mathrm{Lm}(G_{\widetilde{b}})$. Otherwise $\widetilde{b}$ is called *unlucky* for $\widetilde{I}$.

By this definition, we cannot decide whether the point $\widetilde{b}$ is lucky for $\widetilde{I}$ without computing G. This situation is the same as the situation in the modular algorithms (see Definition 2.4.5). Despite that, we will see later that we can choose lucky evaluation points with high probability by a similar test as in the modular algorithms [27].
Before turning our attention to the general scheme of our new method, let us consider the following remark:

Remark 5.3.5. Let $\widetilde{I} = \langle \widetilde{H} \rangle$ be defined as in (5.6) and let $p_1, \ldots, p_m$ be distinct primes. Set $b := (p_1, \ldots, p_m)$ and let $z_0, \ldots, z_{d-1} \in \mathbb{Q} \setminus \{0\}$ be distinct such that for $i = 0, \ldots, d-1$, the points $\widetilde{b}_i = (b, z_i) \in \mathbb{Q}^{m+1}$ satisfy the conditions in Definition 5.3.3. Let $I_b = \langle h \mid h = \widetilde{g}(b, z, X) \in \mathbb{Q}(z)[X], \widetilde{g} \in \widetilde{H} \rangle \subseteq \mathbb{Q}(z)[X]$ and let G_b be the reduced Gröbner basis of I_b. Suppose that each $\widetilde{b}_i$ is lucky for $\widetilde{I}$ and that the number d of these points is sufficiently

large to recover the coefficients in $\mathbb{Q}(z)$ which occur in G_b from the coefficients in $\mathbb{Q}$. For $i = 0, \ldots, d-1$, let $I_{\widetilde{b}_i} = \langle \sigma_{\widetilde{b}_i}(\widetilde{g}) \mid \widetilde{g} \in \widetilde{H} \rangle$ be the ideal in $\mathbb{Q}[X]$ where the map $\sigma_{\widetilde{b}_i}$ is defined as in (5.8). Let $G_{\widetilde{b}_i}$ be the reduced Gröbner basis of the ideal $I_{\widetilde{b}_i}$. Let G'_b be the set of polynomials obtained by applying the dense univariate rational interpolation algorithm (see Section 1.4) coefficient-wise to the set of polynomials, from $G_{\widetilde{b}_i}$, whose leading monomials are the same. Then the set G'_b is the reduced Gröbner basis of I_b with high probability, that is, $G'_b = G_b$ with high probability.

This remark says that, in particular, the set G'_b of polynomials obtained by the dense rational interpolation algorithm generates the ideal I_b with high probability.

Coming back to the general scheme of our new approach which is illustrated in Figure 5.1, we describe the complete picture of the new method as follows: Instead of computing the reduced Gröbner bases over the field K using Buchberger's algorithm, our algorithm computes them in six levels. The levels 1-3 have already been described above. At level 4, for the points $\widetilde{b}_i = (p_1, \ldots, p_m, z_i) \in \mathbb{Q}^{m+1}$ satisfying the conditions in Definition 5.3.3, we compute the reduced Gröbner bases of $I_{\widetilde{b}_i} \subseteq \mathbb{Q}[X]$. For the points in the list $\{\widetilde{b}_0, \ldots, \widetilde{b}_\nu\}$ satisfying the condition in Definition 5.3.4 (and only for those), the dense univariate rational interpolation algorithm then lifts the results to the ring $\mathbb{Q}(z)[X]$ at level 5. The results at this level are expected to be the reduced Gröbner bases of the ideals at level 2 (see Remark 5.3.5). Finally, after sufficiently large sets of polynomials in $\mathbb{Q}(z)[X]$ have been computed, the results are lifted to the ring $\mathbb{Q}(T)[X]$ via sparse multivariate interpolation algorithm at level 6.

Note that Algorithm 4.11 is a combination of the early termination version of the dense univariate rational interpolation algorithm, which is a combination of Algorithm 1.1 and Algorithm 1.2, and the sparse multivariate polynomial interpolation algorithm (see Algorithm 1.3) together with the strategy described in [16] which is also described in Chapter 4. Thus in what follows, whenever we call the algorithms Algorithm 1.1, Algorithm 1.2 and Algorithm 1.3, we refer to Algorithm 4.11.

In the following we give a brief description of the new method: In the beginning, randomly choose a point $s \in \mathbb{Q}^m$ such that none of the denominators of the coefficients in H vanishes when evaluated at $T = s$. With respect to z and s, we obtain an ideal $\widetilde{I}$ as in (5.6) in the polynomial ring $\mathbb{Q}(Tz)[X]$ at level 1. We then fix a point $b = (p_1, \ldots, p_m) \in \mathbb{Q}^m$ by taking random distinct primes from a set of prime numbers. At level 2, we randomly choose rational numbers $z_0, z_1, \ldots, z_v$ for some v such that the points $\widetilde{b}_i = (b, z_i)$ satisfy the conditions in Definition 5.3.3. Let $\mathcal{B} = \left\{ \widetilde{b}_i \mid 0 \leq i \leq v \right\} \subseteq \mathbb{Q}^{m+1}$. At level 3, after evaluating the ideal $\widetilde{I}$ at these points, we obtain ideals, denoted by $I_{\widetilde{b}_i}$ for $0 \leq i \leq v$, in the polynomial ring $\mathbb{Q}[X]$. At level 4, we compute the reduced Gröbner basis $G_{\widetilde{b}_i}$ of the ideal $I_{\widetilde{b}_i}$. Like in modular algorithms [2, 27], one of the problems after computing the set of reduced Gröbner bases $\mathcal{GB} := \{G_{\widetilde{b}_i} \mid \widetilde{b}_i \in \mathcal{B}\}$ is that $\mathcal{B}$ may contain unlucky evaluation points. In this case, as in the modular algorithms, we use the following method to delete the unlucky evaluation points:

DeleteUnluckyEvaluationPoints([27]): *We define an equivalence relation on* ($\mathcal{GB}$,

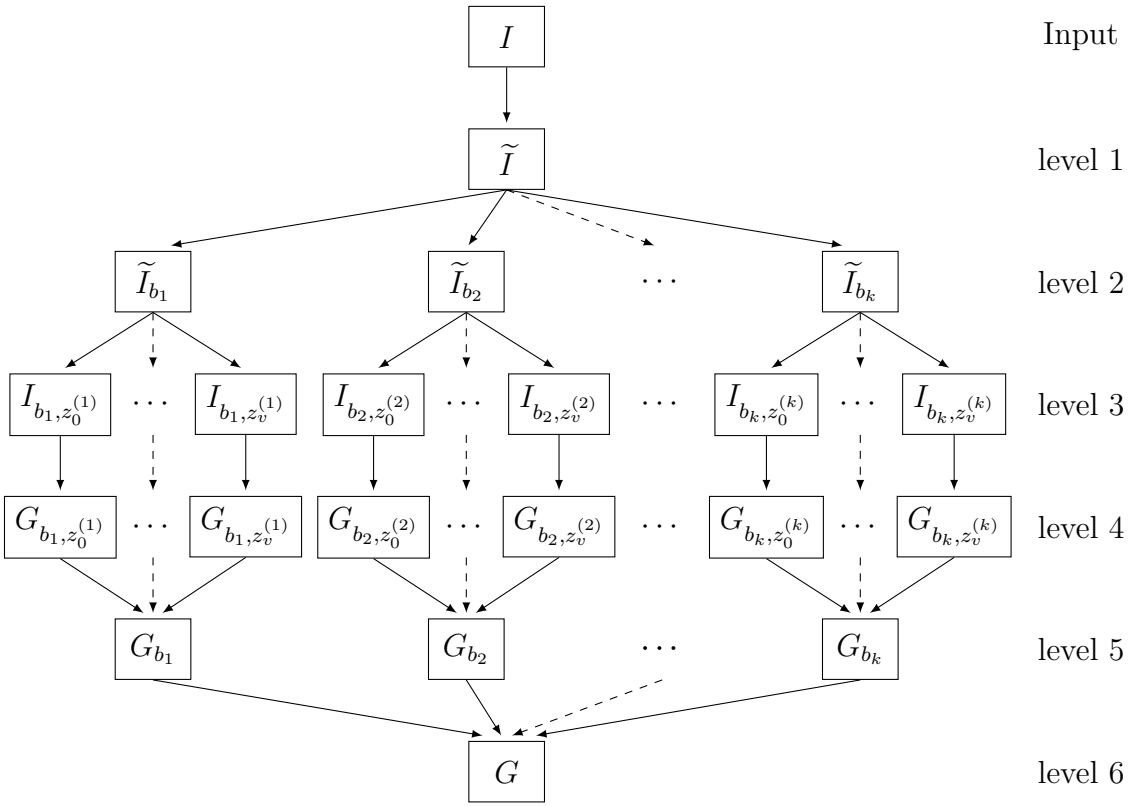

Figure 5.1: General scheme for the new algorithm (general case)

$\mathcal{B}$) *by* $(G_{\widetilde{b}_i}, \widetilde{b}_i) \sim (G_{\widetilde{b}_j}, \widetilde{b}_j) :\Longleftrightarrow \mathrm{Lm}(G_{\widetilde{b}_i}) = \mathrm{Lm}(G_{\widetilde{b}_j})$. *Then the equivalence class of largest cardinality*[1] *is stored in* $(\mathcal{GB}, \mathcal{B})$, *the others are deleted.*

With this method, we assume that all $G_{\widetilde{b}_i}$, $\widetilde{b}_i \in \mathcal{B}$, have the same set of leading monomials. Once we have such sets of polynomials, we can then apply the dense univariate rational interpolation algorithm coefficient-wise to a set of polynomials whose leading monomials are the same. Let $G_{\widetilde{b}_0} = \{h_{\widetilde{b}_0,1}, \ldots, h_{\widetilde{b}_0,r}\}$. Then for $j = 1, \ldots, r$, consider the set

$$F_j = \left\{ g_{i,j} \in G_{\widetilde{b}_i} \mid \mathrm{lm}\,(g_{i,j}) = \mathrm{lm}\left(h_{\widetilde{b}_0,j}\right) \text{ for } i = 0, \ldots, v \right\}$$

of polynomials, in the polynomial ring $\mathbb{Q}[X]$, whose leading monomials are the same. Since we do not know whether $\mathcal{B}$ is sufficiently large (or, equivalently, whether $v \geq d-1$ where d is as in (5.3)), we apply the early termination version of the dense univariate rational interpolation algorithm (see Section 1.4 and Subsection 1.6.1) so that we can add more interpolation points if necessary. This algorithm can be applied coefficient-wise to each set of polynomials F_j as follows: First we lift the coefficients in $\mathbb{Q}$ to $\mathbb{Q}[z]$. To do this, we apply Algorithm 1.1 coefficient-wise to each set F_j to obtain, for each coefficient given as $c_0, \ldots, c_v$ w.r.t. the $v+1$ distinct evaluation points $z_0, \ldots, z_v \in \mathbb{Q}$, a polynomial $g \in \mathbb{Q}[z]$ of degree at most v. Once we lift all the coefficients in $\mathbb{Q}$ to $\mathbb{Q}[z]$, we obtain a set $\widetilde{G}_b$ of polynomials in the polynomial ring $(\mathbb{Q}[z])[X]$. In the second step, by taking Theorem 1.6.1 into account, for each coefficient $g \in \mathbb{Q}[z]$ of $\widetilde{G}_b$, we apply Algorithm 1.2 w.r.t. the polynomial $f = \prod_{i=0}^{v}(z - z_i) \in \mathbb{Q}[z]$ to lift the coefficients in $\mathbb{Q}[z]$ to $\mathbb{Q}(z)$. We thus obtain a set G_b of polynomials in the polynomial ring $\mathbb{Q}(z)[X]$ at level 4. Moreover, from the set G_b we obtain the following:

(a) A bound d on the number of interpolation points required to recover the coefficients in G_b as well as in G.

(b) A bound on the degree of each numerator (resp. denominator) of the rational functions which occur as coefficients of G.

At this level, we compute the sets $G_{b_1}, \ldots, G_{b_k}$ of polynomials with coefficients in $\mathbb{Q}(z)$ w.r.t. the bounds in (a) and (b) and with $b_j = (p_1^j, \ldots, p_m^j)$ for $j = 1, \ldots, k$ for some k. The next step is now to lift the coefficients in $\mathbb{Q}(z)$ to $\mathbb{Q}(T)$ w.r.t. b and s. Note that since we do not know the bound τ in (5.4) in advance, we do not know how many G_{b_j}'s we need to lift these coefficients. We can nevertheless handle this situation by applying the early termination version of the sparse interpolation algorithm (see Algorithm 1.6.2). We do this as follows: Let $G_{b_1} = \{h_{b_1,1}, \ldots, h_{b_1,r}\}$. For $i = 1, \ldots, r$, consider the set

$$G_i = \left\{ g \in G_{b_j} \mid \mathrm{lm}\,(g) = \mathrm{lm}\,(h_{b_1,i}) \text{ for } j = 1, \ldots, k \right\}$$

of polynomials, in the polynomial ring $\mathbb{Q}(z)[X]$, whose leading monomials are the same. At level 5, we apply Algorithm 1.4 coefficient-wise to each set G_i to obtain, for each

[1] Here, we have to use a weighted cardinality count if Algorithm 5.12 requires more than one round of the loop, see [4, Remark 5.7].

coefficient of z to some power, given as $a_1, \ldots, a_k \in \mathbb{Q}$ w.r.t. b and s, a rational function $f \in \mathbb{Q}(T)$, see Algorithm 4.11.

Once we lift all the coefficients in $\mathbb{Q}(z)$ to $\mathbb{Q}(T)$, we obtain a set G of polynomials in the polynomial ring with coefficients in $\mathbb{Q}(T)$. The following remark gives some conditions under which the set G is the reduced Gröbner basis of I with high probability.

Remark 5.3.6. If I reduces to zero w.r.t. G and G is the reduced Gröbner basis of $\langle G \rangle$, then $I = \langle G \rangle$ with high probability.

The method discussed above for computing the reduced Gröbner basis of I w.r.t. $>$ is summarized in Algorithm 5.14. This algorithm calls Algorithms 5.12 and 5.13 to compute the reduced Gröbner bases of the ideals at level 2 with high probability (see Remark 5.3.5). These two algorithms are almost the same. The only difference between them is that Algorithm 5.12 computes the first basis G_b without bounds on the number of interpolation points and on the degrees of the numerators and denominators of the coefficients of G_b, while Algorithm 5.13 computes the remaining Gröbner bases with some given bounds, where these bounds are obtained from the output of Algorithm 5.12.

Remark 5.3.7. In line 6 of Algorithm 5.12 and in line 5 of Algorithm 5.13, we compute d reduced Gröbner bases. However, for the ideals in these lines, we do not know whether coefficient swell occurs during the computation of Gröbner bases. If so, the computations may be expensive. We can, nevertheless, decide which method to use by running two algorithms for computing Gröbner bases in parallel since the generating sets of these ideals only differ by their coefficients. To do this, first, we run the SINGULAR commands `std` and `modStd` in parallel (see remarks in the last page of Subsection 5.5.1) to compute the reduced Gröbner basis of $I_{\tilde{b}_0}$ as in line 5 of Algorithm 5.13. Once we run this, we take the output from whatever method finishes first and continue the remaining computations w.r.t. this method.

In line 6 of Algorithm 5.12 and in line 5 of Algorithm 5.13, if coefficient swell occurs in the intermediate computations, we use the modular algorithms described in [2, 27], see Remark 5.3.7. In line 8 of Algorithm 5.13, since we know that d is a bound on the number of interpolation points, we have to choose at least d distinct lucky evaluation points w.r.t. the DELETEUNLUCKYEVALUATIONPOINTS method.

A more detailed description how to lift the coefficients in $\mathbb{Q}(z)$ to $\mathbb{Q}(T)$, see line 16 of Algorithm 5.14, is given in Chapter 4. If in line 23 of Algorithm 5.14 either $I \not\subseteq \langle G \rangle$ or G is not the reduced Gröbner basis of $\langle G \rangle$, then the algorithm restarts the whole process with new random evaluation points until it succeeds. Since the set of evaluation points for which the coefficients in $\mathbb{Q}$ are correctly lifted to $\mathbb{Q}(T)$ is a non-empty open subset in the Zariski topology on $\mathbb{A}^m(\mathbb{Q})$, see Subsection 5.3.1, Algorithm 5.14 returns the correct result with high probability, see Remark 5.3.11. For $g \in G$, let s'_g be the number of terms in g. In this algorithm, Algorithm 4.11 is applied coefficient-wise s' times where

$$s' = \sum_{g \in G} (s'_g - 1).$$

Algorithm 5.12 Gröbner bases over $\mathbb{Q}(z)$ without known bounds

Input: An ideal $\widetilde{I} = \langle \widetilde{H} \rangle \subseteq \mathbb{Q}(Tz)[X]$, an evaluation point $b \in \mathbb{Q}^m$.

Output: G_b, the reduced Gröbner basis of $I_b := \langle h \mid h = g(b, z, X), g \in \widetilde{H} \rangle \subseteq \mathbb{Q}(z)[X]$ w.r.t. $>$ with high probability.

1: let $\mathcal{B}$ be the set of pairs (b, a) for a finite number of random points $a \in \mathbb{Q}$ such that (b, a) is admissible w.r.t. $\widetilde{H}$
2: $\mathcal{GB} \longleftarrow \{\}$
3: $e \longleftarrow 0$, $G_b \longleftarrow \{\}$
4: **loop**
5: **for** $\widetilde{b} = (b, a) \in \mathcal{B}$ **do**
6: compute the reduced Gröbner basis $G_{\widetilde{b}}$ of $I_{\widetilde{b}} = \left\langle \widetilde{H}|_{(T,z)=\widetilde{b}} \right\rangle \subseteq \mathbb{Q}[X]$ w.r.t. $>$
7: $\mathcal{GB} \longleftarrow \mathcal{GB} \cup \{G_{\widetilde{b}}\}$
8: $(\mathcal{GB}, \mathcal{B}) \longleftarrow$ DeleteUnluckyEvaluationPoints$(\mathcal{GB}, \mathcal{B})$
9: let $G_{\widetilde{b}_0}, \ldots, G_{\widetilde{b}_l}$ be the elements in $\mathcal{GB}$
10: let $g_{0,1}, \ldots, g_{0,r}$ be the elements in $G_{\widetilde{b}_0}$
11: $f \longleftarrow \prod_{(b,a)\in\mathcal{B}}(z - a) \subseteq \mathbb{Q}[z]$
12: **for** $j = e + 1, \ldots, r$ **do**
13: consider the set

$$G_j = \{g \in G_{\widetilde{b}_i} \mid \mathrm{lm}(g) = \mathrm{lm}(g_{0,j}),\ 0 \leq i \leq l\} \subseteq \mathbb{Q}[X]$$

of polynomials whose leading monomials are the same
14: lift the coefficients of G_j to $\mathbb{Q}(z)$ by applying the early termination version of the dense rational interpolation algorithm (see Subsection 1.6.1) coefficient-wise w.r.t. f
15: **if** all coefficients of G_j are successfully lifted to $\mathbb{Q}(z)$ **then**
16: let g_j be the polynomial obtained by lifting G_j to $\mathbb{Q}(z)[X]$
17: $G_b \longleftarrow G_b \cup \{g_j\}$
18: $e \longleftarrow e + 1$
19: **else**
20: break out of the for-loop
21: **if** $e = r$ **then**
22: break out of the loop
23: enlarge $\mathcal{B}$
24: **return** G_b

Algorithm 5.13 Gröbner bases over $\mathbb{Q}(z)$ with given bounds

Input: An ideal $\widetilde{I} = \langle \widetilde{H} \rangle \subseteq \mathbb{Q}(Tz)[X]$, $b \in \mathbb{Q}^m$, $d \in \mathbb{N}$, and $N, D \subseteq \mathbb{N}$, where d is a bound on the number of interpolation points and N and D are degree bounds for the numerators and denominators of the coefficients in the result as explained in the text.

Output: G_b, the reduced Gröbner basis of $I_b := \langle h \mid h = g(b, z, X), g \in \widetilde{H} \rangle \subseteq \mathbb{Q}(z)[X]$ w.r.t. $>$ with high probability.

1: choose d distinct random points $z_0, \ldots, z_{d-1} \in \mathbb{Q}$ such that each $\widetilde{b}_i := (b, z_i)$ is admissible w.r.t. $\widetilde{H}$
2: $\mathcal{B} \longleftarrow \{\widetilde{b}_0, \ldots, \widetilde{b}_{d-1}\}$
3: $\mathcal{GB} \longleftarrow \{\}$
4: **for** $i = 0, \ldots, d-1$ **do**
5: compute the reduced Gröbner basis $G_{\widetilde{b}_i}$ of $I_{\widetilde{b}_i} = \left\langle \widetilde{H}|_{(T,z)=\widetilde{b}} \right\rangle \subseteq \mathbb{Q}[X]$ w.r.t. $>$
6: $\mathcal{GB} \longleftarrow \mathcal{GB} \cup \{G_{\widetilde{b}_i}\}$
7: $(\mathcal{GB}, \mathcal{B}) \longleftarrow$ DeleteUnluckyEvaluationPoints $(\mathcal{GB}, \mathcal{B})$
8: **if** $\#\mathcal{B} =: d' < d$ **then**
9: go to line 1 to replace the unlucky evaluation points with lucky ones
10: $f \longleftarrow \prod_{\substack{(b,z_\lambda)\in\mathcal{B} \\ 1\le\lambda\le d'}} (z - z_\lambda) \subseteq \mathbb{Q}[z]$
11: let $g_{0,1}, \ldots, g_{0,r}$ be the elements in $G_{\widetilde{b}_0}$
12: **for** $j = 1, \ldots, r$ **do**
13: consider the set

$$G_j = \{g \in G_{\widetilde{b}_i} \mid \mathrm{lm}(g) = \mathrm{lm}(g_{0,j}),\ 0 \le i \le d' - 1\} \subseteq \mathbb{Q}[X]$$

of polynomials whose leading monomials are the same
14: apply the dense rational interpolation algorithm w.r.t. f and the degree bounds in N and D (first Algorithm 1.1 and then Algorithm 1.2, see Section 1.4) coefficient-wise to each set G_j to obtain a set G_b of polynomials in $\mathbb{Q}(z)[X]$
15: **return** G_b

Algorithm 5.14 Gröbner bases over $K = \mathbb{Q}(T)$ (`ffmodStd`)

Input: An ideal $I = \langle H \rangle \subseteq S = K[X]$, where $K = \mathbb{Q}(T)$ and $H = \{f_1, \ldots, f_w\}$.

Output: $G \subseteq S$, the reduced Gröbner basis of I w.r.t. $>$ with high probability.

1: choose a random non-zero point $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ such that none of the denominators in the coefficients of H vanishes when evaluated at $T = s$
2: let $\widetilde{I} = \langle \widetilde{H} \rangle \subseteq \mathbb{Q}(Tz)[X]$ be as in (5.6) w.r.t. z and s
3: choose m distinct random prime numbers $p_1, \ldots, p_m$
4: $j \longleftarrow 1,\ b \longleftarrow b_j \longleftarrow (p_1, \ldots, p_m)$
5: let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.12 applied to $\widetilde{I}$ and b
6: let (N, D) be the pair of ordered sets containing the degrees of the polynomials in the numerators (resp. denominators) of the coefficients of G_{b_j}
7: $d_N \longleftarrow \max N,\ d_D \longleftarrow \max D,\ d \longleftarrow d_N + d_D + 2$
8: $G \longleftarrow \{\},\ e \longleftarrow 0$
9: let $h_{b_1,1}, \ldots, h_{b_1,r}$ be the elements in G_{b_1}
10: **while** $e < r$ **do**
11: $\quad j \longleftarrow j + 1$
12: $\quad b_j \longleftarrow (p_1^j, \ldots, p_m^j)$
13: $\quad$ let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.13 applied to $\widetilde{I},\ b_j$, and (d, N, D)
14: $\quad$ **for** $i = e + 1, \ldots, r$ **do**
15: $\quad\quad$ consider the set

$$G_i = \{g \in G_{b_k} \mid \mathrm{lm}(g) = \mathrm{lm}(h_{b_1,i}),\ 1 \leq k \leq j\} \subseteq \mathbb{Q}(z)[X]$$

$\quad\quad$ of polynomials whose leading monomials are the same
16: $\quad\quad$ lift the coefficients of G_i to $\mathbb{Q}(T)$ by applying the early termination version of the sparse multivariate interpolation algorithm, Algorithm 1.4, coefficient-wise w.r.t. b and s
17: $\quad\quad$ **if** all coefficients of G_i are successfully lifted to $\mathbb{Q}(T)$ **then**
18: $\quad\quad\quad$ let g_i be the polynomial obtained by lifting G_i to $\mathbb{Q}(T)[X]$
19: $\quad\quad\quad$ $G \longleftarrow G \cup \{g_i\}$
20: $\quad\quad\quad$ $e \longleftarrow e + 1$
21: $\quad\quad$ **else**
22: $\quad\quad\quad$ break out of the for-loop
23: **if** $I \not\subseteq \langle G \rangle$ or G is not the reduced Gröbner basis of $\langle G \rangle$ **then**
24: $\quad$ reapply Algorithm 5.14 by choosing different evaluation points in lines 1 and 3
25: **return** G

In line 5 of Algorithm 5.14, let $G_b := G_{b_1} = \{g_1, \ldots, g_r\}$ and let q_i be the number of terms in g_i. Let $n_{\lambda,i}$ (resp. $d_{\lambda,i}$) be the degree of the numerator (resp. denominator) of the coefficient of the λ-th term of g_i. In line 6 of this algorithm, we consider a pair of ordered sets

$$(N, D) = (\{n_{\lambda,i}\}_{1\leq\lambda\leq q_i,\, 1\leq i\leq r}, \{d_{\lambda,i}\}_{1\leq\lambda\leq q_i,\, 1\leq i\leq r})\,.$$

Remark 5.3.8. Some parts of Algorithms 5.12 and 5.13 are inherently parallelizable. The SINGULAR framework enables us to do some of the tasks in these algorithms in parallel. In the current implementation, see Section 5.3.3, the for-loops starting in line 5 of Algorithm 5.12 and in line 4 of Algorithm 5.13 are parallelized. Recall from above that we use the modular techniques [2, 27] in line 6 of Algorithm 5.12 and in line 5 of Algorithm 5.13 if coefficient swell occurs in the intermediate computations, see Remark 5.3.7. In this case, some of the tasks in the modular algorithms are parallelized as described in Remark 2.5.1. Moreover, in line 14 of Algorithm 5.13, we lift the coefficients from $\mathbb{Q}$ to $\mathbb{Q}(z)$ w.r.t. some given bounds in parallel.

5.3.1 Choice of Evaluation Points

In this subsection, we briefly explain to which extend the algorithms discussed in this chapter are probabilistic.

Proposition 5.3.9. *Let $W \subseteq \mathbb{Q}(T) = \mathbb{Q}(t_1, \ldots, t_m)$ be a finite set of rational functions. Then there exists a non-empty Zariski open subset U of $\mathbb{A}^m(\mathbb{Q})$ such that for each $f \in W$ and each point $s \in U$, the function f can be evaluated at s, that is, $f(s) \in \mathbb{Q}$.*

Proof. For any element $c \neq 0$ (in lowest terms) of $\mathbb{Q}(T)$, let c_d be the denominator of c, considered as a polynomial in $\mathbb{Q}[T]$. With this notation, set $C_W := \{c_d \in \mathbb{Q}[T] \mid c \in W\}$ and define a set V by

$$V := \bigcup_{f\in C_W} Z(f)\,. \tag{5.9}$$

Each algebraic set $Z(f)$, $f \in C_W$, is a closed set in the Zariski topology on $\mathbb{A}^m(\mathbb{Q})$. Since finite unions of closed sets are closed, the set V is closed, too. Consider the subset $U := \mathbb{A}^m(\mathbb{Q}) \setminus V$ of $\mathbb{A}^m(\mathbb{Q})$. Then U is an open subset in the Zariski topology and by definition, each function $f \in W$ can be evaluated at each point $s \in U$. Moreover, since none of the polynomials in C_W is zero, the dimension of V is at most $m - 1$, see [13, Chapter 9]. Thus we have $V \neq \mathbb{A}^m(\mathbb{Q})$ which implies that the set U is non-empty. □

Consider Algorithm 5.15 which is derived from Algorithm 5.14 by replacing line 5 with (a copy of) line 13 and by skipping lines 23 to 24.

The evaluation points which we use for Algorithm 5.15 are taken from Zariski open subsets of $\mathbb{A}^m(\mathbb{Q})$.

Algorithm 5.15 Gröbner bases over $K = \mathbb{Q}(T)$

Input: An ideal $I = \langle H \rangle \subseteq S = K[X]$, and $N, D \subseteq \mathbb{N}$, where $K = \mathbb{Q}(T)$, $H = \{f_1, \ldots, f_w\}$, and N and D are degree bounds for the numerators and denominators of the coefficients of the reduced Gröbner basis of I as explained in the paragraph above Remark 5.3.8.

Output: $G \subseteq S$, the reduced Gröbner basis of I w.r.t. $>$.

choose a random non-zero point $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ such that none of the denominators in the coefficients of H vanishes when evaluated at $T = s$
let $\widetilde{I} = \langle \widetilde{H} \rangle \subseteq \mathbb{Q}(Tz)[X]$ be as in (5.6) w.r.t. z and s
choose m distinct random prime numbers $p_1, \ldots, p_m$
$j \longleftarrow 1,\ b \longleftarrow b_j \longleftarrow (p_1, \ldots, p_m)$
$d_N \longleftarrow \max N,\ d_D \longleftarrow \max D,\ d \longleftarrow d_N + d_D + 2$
let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.13 applied to $\widetilde{I}$, b_j and (d, N, D)
$G \longleftarrow \{\},\ e \longleftarrow 0$
let $h_{b_1,1}, \ldots, h_{b_1,r}$ be the elements in G_{b_1}
while $e < r$ **do**
  $j \longleftarrow j + 1$
  $b_j \longleftarrow (p_1^j, \ldots, p_m^j)$
  let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.13 applied to $\widetilde{I}$, b_j, and (d, N, D)
  for $i = e + 1, \ldots, r$ **do**
    consider the set

$$G_i = \{g \in G_{b_k} \mid \operatorname{lm}(g) = \operatorname{lm}(h_{b_1,i}),\ 1 \leq k \leq j\} \subseteq \mathbb{Q}(z)[X]$$

    of polynomials whose leading monomials are the same
    lift the coefficients of G_i to $\mathbb{Q}(T)$ by applying the early termination version of the sparse multivariate interpolation algorithm, Algorithm 1.4, coefficient-wise w.r.t. b and s
    if all coefficients of G_i are successfully lifted to $\mathbb{Q}(T)$ **then**
      let g_i be the polynomial obtained by lifting G_i to $\mathbb{Q}(T)[X]$
      $G \longleftarrow G \cup \{g_i\}$
      $e \longleftarrow e + 1$
    else
      break out of the for-loop
return G

Proposition 5.3.10. *Let $I = \langle H \rangle = \langle f_1, \ldots, f_w \rangle \subseteq K[X]$ be an ideal where $K = \mathbb{Q}(t_1, \ldots, t_m)$ is an algebraic function field, and let $N, D \subseteq \mathbb{N}$ be degree bounds for the numerators and denominators of the coefficients of the reduced Gröbner basis of I, see the paragraph above Remark 5.3.8. Let G be the output of Algorithm 5.15 applied to I and (N, D). Then there exists a non-empty Zariski open subset U of $\mathbb{A}^m(\mathbb{Q})$ such that for every $s \in U$, the following holds if s is chosen in line 1 of Algorithm 5.15:*

(1) *Each time when Algorithm 5.13 is called within Algorithm 5.15 (see lines 6 and 12), then there exists a non-empty Zariski open subset U' of $\mathbb{A}(\mathbb{Q})$ such that if $z_0, \ldots, z_{d-1}$ are chosen from U' in line 1 of Algorithm 5.13, the result of Algorithm 5.13 is correct.*

(2) *If all the results of Algorithm 5.13 in the intermediate steps are correct, then the result of Algorithm 5.15 is correct, that is, G is the reduced Gröbner basis of I.*

Proof. Let ALG be an algorithm, such as Buchberger's algorithm, computing a reduced Gröbner basis of an ideal over any field. We assume that each polynomial in the intermediate steps of this algorithm is monic, that is, the leading coefficient of this polynomial is equal to 1. Moreover, we assume without loss of generality that each element in the input is monic.

Let $W \subseteq \mathbb{Q}(T)$ be the set of intermediate rational functions (including the input and the output) which occur during the computation of the reduced Gröbner basis of I using the algorithm ALG. Since the integer j in line 10 of Algorithm 5.15 is bounded by two times the maximum number of terms in the numerators and denominators of the coefficients of the reduced Gröbner basis of I as given in (5.4), the set W is finite. Thus by Proposition 5.3.9 there exists a non-empty Zariski open subset U of $\mathbb{A}^m(\mathbb{Q})$ such that for each $f \in W$ and for each $s \in U$, f can be evaluated at s. Now, for any point $s = (s_1, \ldots, s_m) \in U$, the image of the input ideal

$$\widetilde{I} = \langle \widetilde{H} \rangle \subseteq \mathbb{Q}(Tz)[X] = \mathbb{Q}(t_1 z, \ldots, t_m z)[X] \subseteq \mathbb{Q}(t_1, \ldots, t_m, z)[X]$$

in Algorithm 5.13 under the map

$$\begin{aligned} \varphi_s : \ \mathbb{Q}(Tz)[X] &\longrightarrow \mathbb{Q}(z)[X], \\ t_i z &\longmapsto s_i z, \end{aligned} \tag{5.10}$$

which leaves X fixed is the ideal

$$I' := \varphi_s\left(\widetilde{I}\right) = \varphi_s\left(\langle \widetilde{H} \rangle\right) = \left\langle \varphi_s\left(\widetilde{H}\right) \right\rangle \subseteq \mathbb{Q}(z)[X].$$

Let $p_1, \ldots, p_m$ be distinct primes. Set $b_j := (p_1^j, \ldots, p_m^j)$ for some j. Let G' be the output of Algorithm 5.13 applied to $\widetilde{I}$, b_j and (d, N, D) as in lines 6 and 12 of Algorithm 5.15. In fact, instead of applying Algorithm 5.13, one could call any algorithm to compute the reduced Gröbner basis of I'. Let $W' \subseteq \mathbb{Q}(z)$ be the set of intermediate rational functions (including the input and the output) which occur during the computation of the reduced Gröbner basis of I' using the algorithm ALG.

Since the number of interpolation points required to recover the coefficients of the reduced Gröbner basis of I' is bounded by d, where d is defined as in line 5 of Algorithm 5.15 w.r.t. N and D, the set W' is finite. Thus by Proposition 5.3.9 there exists a non-empty Zariski open subset U' of $\mathbb{A}(\mathbb{Q})$ such that for each $f \in W'$ and for each $s \in U'$, f can be evaluated at s. Let $\psi : \mathbb{Q}(z)[X] \longrightarrow \mathbb{Q}(z)[X]$ and $\phi : \mathbb{Q}[X] \longrightarrow \mathbb{Q}[X]$ be the maps that send a given ideal to its reduced Gröbner basis. Furthermore, for any point $z_0 \in U'$, consider the set $R_{\langle z-z_0\rangle} \subseteq \mathbb{Q}(z)$ of all rational functions whose denominators do not vanish at $z = z_0$, that is,

$$R_{\langle z-z_0\rangle} = \left\{ \frac{f}{g} \in \mathbb{Q}(z) \;\middle|\; f \in \mathbb{Q}[z],\ g \in \mathbb{Q}[z] \setminus \{0\} \text{ with } g(z_0) \neq 0 \right\} \supseteq W' .$$

In what follows, we consider the map

$$\begin{aligned} \varphi_{z_0} : \mathbb{Q}(z)[X] &\longrightarrow \mathbb{Q}[X], \\ z &\longmapsto z_0, \end{aligned} \tag{5.11}$$

which leaves X fixed where this map is only applied to polynomials in the polynomial ring $R_{\langle z-z_0\rangle}[X] \subseteq \mathbb{Q}(z)[X]$. Consider the following diagram:

$$\begin{array}{ccc} \mathbb{Q}(z)[X] & \xrightarrow{\ \psi\ } & \mathbb{Q}(z)[X] \\ \Big\downarrow{\scriptstyle \varphi_{z_0}} & & \Big\downarrow{\scriptstyle \varphi_{z_0}} \\ \mathbb{Q}[X] & \xrightarrow{\ \phi\ } & \mathbb{Q}[X] \end{array} \tag{5.12}$$

The above diagram commutes for I' in the following sense: The maps ψ and ϕ can be realized by applying the algorithm ALG. If the input ideals are I' and $\varphi_{z_0}(I')$, respectively, then for any $z_0 \in U'$, the algorithm ALG performs the same steps in the same order. In particular, the diagram commutes. The same holds for any elements $z_0, \ldots, z_{d-1}$ of U'. Therefore, the result of Algorithm 5.13 is correct if $z_0, \ldots, z_{d-1}$ are chosen from U' in line 1 of Algorithm 5.13.

It remains to show that the statement in (2) holds for any $s \in U$. Consider the map

$$\begin{aligned} \varphi' : \mathbb{Q}(T)[X] &\longrightarrow \mathbb{Q}(Tz)[X], \\ t_i &\longmapsto t_i z . \end{aligned} \tag{5.13}$$

As above, consider the diagram

$$\begin{array}{ccc} \mathbb{Q}(T)[X] & \xrightarrow{\psi'} & \mathbb{Q}(T)[X] \\ \downarrow{\scriptstyle \varphi'_s} & & \downarrow{\scriptstyle \varphi'_s} \\ \mathbb{Q}(z)[X] & \xrightarrow{\psi} & \mathbb{Q}(z)[X] \end{array} \tag{5.14}$$

where

(a) s is any element chosen from U as in line 1 of Algorithm 5.15,

(b) ψ is as in diagram (5.12),

(c) $\varphi'_s = \varphi_s \circ \varphi'$ where φ_s is as defined in (5.10), and

(d) ψ' is the map sending a given ideal to its reduced Gröbner basis.

Then it is easy to see that the above diagram commutes in the same sense as explained for diagram (5.12). Thus the result of Algorithm 5.15 is correct. □

Let the notation be as in the proof of the above proposition. For any element $c \neq 0$ (in lowest terms) of $\mathbb{Q}(T)$ (or $\mathbb{Q}(z)$), let c_d be the denominator of c, considered as a polynomial in $\mathbb{Q}[T]$ (or $\mathbb{Q}[z]$). Set

$$\begin{aligned} C_H &:= \{c_d \in \mathbb{Q}[T] \mid c \text{ a coefficient occurring in } H\} \subseteq C_W := \{c_d \in \mathbb{Q}[T] \mid c \in W\}, \\ C_{W'} &:= \{c_d \in \mathbb{Q}[z] \mid c \in W'\}, \text{ and} \\ C_{\varphi_s(\widetilde{H})} &:= \left\{c_d \in \mathbb{Q}[z] \;\middle|\; c \text{ a coefficient occurring in } \varphi_s\left(\widetilde{H}\right)\right\} \subseteq C_{W'}. \end{aligned} \tag{5.15}$$

Remark 5.3.11. Let C_H, C_W, $C_{W'}$ and $C_{\varphi_s(\widetilde{H})}$ be as defined in (5.15). From the theoretical point of view, we recover the coefficients of G and G', respectively, using Algorithm 4.11 and the algorithms described in Section 1.4, by choosing evaluation points from U and U'. From the practical point of view, however, we choose these evaluation points from the larger subsets

$$V' = \mathbb{A}^m(\mathbb{Q}) \setminus \left(\bigcup_{f \in C_H} Z(f)\right) \supseteq U \text{ and}$$

$$V'' = \mathbb{A}(\mathbb{Q}) \setminus \left(\bigcup_{f \in C_{\varphi_s(\widetilde{H})}} Z(f)\right) \supseteq U',$$

respectively, which are open sets in the Zariski topology. In this chapter, although we cannot guarantee that none of the polynomials in $C_W \setminus C_H$ or $C_{W'} \setminus C_{\varphi_s(\widetilde{H})}$ vanishes at any of the chosen evaluation points, in practice, it is very unlikely that this case happens. Thus, the coefficients of G and G' can be reconstructed with high probability.

Remark 5.3.12. In line 1 of Algorithm 5.15, in practice, we choose the evaluation points from a *large finite* subset of $\mathbb{Z}^m \cap V'$. In our implementation, we choose these evaluation points from the set

$$\{100, \ldots, 150 + 7 \cdot (m-1)\}^m \cap V' .$$

The same is true in line 1 of Algorithm 5.13. The primes we use for modular computations can be represented in SINGULAR by at most 29 bits. However, for sparse multivariate interpolations, see line 3 of Algorithm 5.15, we choose small primes.

5.3.2 An Illustrative Example

In this subsection, we consider an example which illustrates how to use the new method discussed above to compute the reduced Gröbner basis of a given ideal. We compute, following the steps in Algorithm 5.14, the reduced Gröbner basis of the ideal $I \subseteq \mathbb{Q}(t_1, t_2)[x_1, x_2, x_3]$ generated by the polynomials

$$\begin{aligned}
f_1 &= x_1^2x_2^3x_3 + 2t_1x_1x_2x_3^2 + 7x_2^3,\\
f_2 &= x_1^2x_2^4x_3 + (t_1 - 7t_2)x_1^2x_2x_3^2 - x_1x_2^2x_3^2 + 2x_1^2x_2x_3 - 12x_1 + t_2x_2,\\
f_3 &= (t_1^2 + t_2 - 2)x_2^5x_3 + (t_1 + 5t_2)x_1^2x_2^2x_3 - t_2x_1x_2^3x_3 - x_1x_2^3 + x_2^4 + 2t_1^2x_2^2x_3, \text{ and}\\
f_4 &= t_1x_1^2x_2^2x_3 - x_1x_2^3x_3 + (-t_1 + 4)x_2^3x_3^2 + 3t_1x_1x_2x_3^3 + 4x_3^2 - t_2x_1
\end{aligned}$$

w.r.t. the degree reverse lexicographical ordering (`dp` in SINGULAR) with $x_1 > x_2 > x_3$ as follows:

By making each f_i monic, we obtain $t_1^2 + t_2 - 2, t_1 \in \mathbb{Q}[t_1, t_2]$ as the denominators of the coefficients in $f \in \{f_1, \ldots, f_4\}$ whose degrees are greater than zero. Since these polynomials do not vanish when evaluated at $(t_1, t_2) = (2, 3)$, we may choose the point $s = (2, 3)$ as a shift. Note that in this example, we have $X = \{x_1, x_2, x_3\}$ and $T = \{t_1, t_2\}$. Let

$$\widetilde{I} = \langle \widetilde{H} \rangle = \langle h \mid h = f(t_1z + s_1, t_2z + s_2, X),\ f \in H = \{f_1, \ldots, f_4\} \rangle \subseteq \mathbb{Q}(Tz)[X]$$

be the auxiliary ideal w.r.t. z as in (5.6). Consider the set

$$\begin{aligned}
J = \{&2t_1z + 4,\ (t_1 - 7t_2)z - 19,\ t_2z + 3,\ t_1^2z^2 + (4t_1 + t_2)z + 5,\ (t_1 + 5t_2)z + 17,\\
&-t_2z - 3,\ 2t_1^2z^2 + 8t_1z + 8,\ t_1z + 2,\ -t_1z + 2,\ 3t_1z + 6,\ -t_2z - 3\} \subseteq \mathbb{Q}[Tz]
\end{aligned}$$

of polynomials, as in (5.7), containing all distinct coefficients of $\widetilde{H}$ which are of degrees greater than 0. Set $b := b_1 := (p_1, p_2) = (3, 5)$. For simplicity, we start the computation by taking three evaluation points. As in Algorithm 5.12, let us choose random distinct points $3, 5$ and 9 from $\mathbb{Q}$. Now consider the points $\widetilde{b_1} = (b_1, 3)$, $\widetilde{b_2} = (b_1, 5)$ and $\widetilde{b_3} = (b_1, 9) \in \mathbb{Q}^3$. Since none of the polynomials in J vanishes when evaluated at these points,

the evaluation points are admissible w.r.t. $\widetilde{H}$, see Definition 5.3.3. The reduced Gröbner bases of the ideals $\widetilde{I}|_{(T,z)=\widetilde{b}_i}$, $i = 1, 2, 3$, are

$$\begin{aligned} G_{\widetilde{b}_1} &= \{x_1 - 3/2x_2,\ x_3^2 - 27/4x_2,\ x_2^2x_3,\ x_2^3\}, \\ G_{\widetilde{b}_2} &= \{x_1 - 7/3x_2,\ x_3^2 - 49/3x_2,\ x_2^2x_3,\ x_2^3\}, \\ G_{\widetilde{b}_3} &= \{x_1 - 4x_2,\ x_3^2 - 48x_2,\ x_2^2x_3,\ x_2^3\}. \end{aligned} \tag{5.16}$$

Note that according to the test in line 8 of Algorithm 5.12, all of the evaluation points are lucky for $\widetilde{I}$ with high probability. As in line 13 of Algorithm 5.12, we have the sets

$$\begin{aligned} G_1 &= \{x_1 - 3/2x_2, x_1 - 7/3x_2, x_1 - 4x_2\}, \\ G_2 &= \{x_3^2 - 27/4x_2, x_3^2 - 49/3x_2, x_3^2 - 48x_2\}, \\ G_3 &= \{x_2^2x_3, x_2^2x_3, x_2^2x_3\}, \text{ and} \\ G_4 &= \{x_2^3, x_2^3, x_2^3\} \end{aligned} \tag{5.17}$$

of polynomials whose leading monomials are the same. By applying the dense rational interpolation algorithm, discussed in Section 1.4, coefficient-wise to each set G_i w.r.t. the polynomial $f = (z-3)\cdot(z-5)\cdot(z-9)$, we obtain the set

$$\begin{aligned} G_{b_1} = \{&x_1 + (-5/12z - 1/4)x_2,\ x_3^2 + (-25/48z^2 - 5/8z - 3/16)x_2,\ x_2^2x_3,\ x_2^3\} \\ &\subseteq \mathbb{Q}(z)[X]. \end{aligned} \tag{5.18}$$

To see whether the coefficients in each set G_i are correctly lifted to $\mathbb{Q}(z)$, we have to check this by taking one more admissible point. The point $\widetilde{b}_4 = (b_1, 10)$ is an admissible evaluation point w.r.t. $\widetilde{H}$ because it satisfies the conditions in Definition 5.3.3. The reduced Gröbner basis of the ideal $\widetilde{I}|_{(T,z)=\widetilde{b}_4}$ is

$$G_{\widetilde{b}_4} = \{x_1 - 53/12x_2,\ x_3^2 - 2809/48x_2,\ x_2^2x_3,\ x_2^3\}\,.$$

It is not hard to see that $\widetilde{b}_4$ is lucky for $\widetilde{I}$ with high probability. Having added $G_{\widetilde{b}_4}$ to the sets in (5.16), we continue by applying the dense rational interpolation algorithm w.r.t. the polynomial $f = (z-3)\cdot(z-5)\cdot(z-9)\cdot(z-10)$ coefficient-wise to the set of polynomials whose leading monomials are the same. Having applied this algorithm, we obtain the set

$$\begin{aligned} G_{b_1} = \{&x_1 + (-5/12z - 1/4)x_2,\ x_3^2 + (-25/48z^2 - 5/8z - 3/16)x_2,\ x_2^2x_3,\ x_2^3\} \\ &\subseteq \mathbb{Q}(z)[X]. \end{aligned} \tag{5.19}$$

Since the set G_{b_1} of polynomials in (5.18) coincides with the one in (5.19), the early termination algorithm says that the coefficients in $\mathbb{Q}$ are correctly lifted to $\mathbb{Q}(z)$ with high probability, see Remark 5.3.11. In G_{b_1}, the maximum degree of z in one of the numerators (resp. denominators) is 2 (resp. 0). Hence $d = 3$ is a bound on the number of interpolation points required to recover the coefficients in $\mathbb{Q}(z)$. The remaining task is now to lift the coefficients in $\mathbb{Q}(z)$ to $\mathbb{Q}(T)$. To do this, we continue iterating Algorithm 5.14 until we

reach the bound required by the early termination. For $j = 2, 3$, it is easy to check that the evaluation points

$$(b_2, 3),\ (b_2, 5) \text{ and } (b_2, 9),\ \text{and}$$
$$(b_3, 3),\ (b_3, 5) \text{ and } (b_3, 9)$$

with $b_j = (3^j, 5^j)$ are admissible w.r.t. $\widetilde{H}$. With respect to these evaluation points, we obtain, in a similar way, the sets

$$G_{b_2} = \{x_1 + (-25/12z - 1/4)x_2,\ x_3^2 + (-625/48z^2 - 25/8z - 3/16)x_2,\ x_2^2x_3,\ x_2^3\} \text{ and}$$

$$G_{b_3} = \{x_1 + (-125/12z - 1/4)x_2,\ x_3^2 + (-15625/48z^2 - 125/8z - 3/16)x_2,\ x_2^2x_3,\ x_2^3\}$$

of polynomials in $\mathbb{Q}(z)[X]$. We now apply Algorithm 1.4, following the steps in Algorithm 4.11, coefficient-wise to the set of rational functions in $\mathbb{Q}(z)$ which are the coefficients in G_{b_j} for $j = 1, 2, 3$. Having applied this algorithm, we obtain the set

$$G = \{g_1 = x_1 - \frac{1}{12}t_2x_2,\ g_2 = x_3^2 - \frac{1}{48}t_2^2x_2,\ g_3 = x_2^2x_3,\ g_4 = x_2^3\} \subseteq \mathbb{Q}(T)[X].$$

To see whether G is a Gröbner basis of I with high probability, we verify the conditions in line 23 of Algorithm 5.14 (see Remark 5.3.6) as follows:

```
> ring S = (0,t1,t2), (x1,x2,x3), dp;
> poly f1 = x1^2*x2^3*x3+2*t1*x1*x2*x3^2+7*x2^3;
> poly f2 = x1^2*x2^4*x3+(t1-7*t2)*x1^2*x2*x3^2-x1*x2^2*x3^2
            +2*x1^2*x2*x3-12*x1+t2*x2;
> poly f3 = (t1^2+t2-2)*x2^5*x3+(t1+5*t2)*x1^2*x2^2*x3
            -t2*x1*x2^3*x3-x1*x2^3+x2^4+2*t1^2*x2^2*x3;
> poly f4 = t1*x1^2*x2^2*x3-x1*x2^3*x3+(-t1+4)*x2^3*x3^2
            +3*t1*x1*x2*x3^3+4*x3^2-t2*x1;
> ideal I = f1,f2,f3,f4;
> G = x1+(-t2/12)*x2, x3^2+(-t2^2/48)*x2, x2^2*x3, x2^3;
> attrib(G,"isSB",1);
> reduce(I,G); // I contained in <G>
 _[1] = 0
 _[2] = 0
 _[3] = 0
 _[4] = 0
> option(redSB);
> ideal K = std(G);
> reduce(K,G); // G is a Groebner basis of <G>
```

```
_[1] = 0
_[2] = 0
_[3] = 0
_[4] = 0
```

Since both conditions in line 23 of Algorithm 5.14 are not satisfied (or, equivalently, both conditions in Remark 5.3.6 are satisfied), we thus have correctly lifted the coefficients from $\mathbb{Q}(z)$ to $\mathbb{Q}(T)$ with high probability. Hence it follows from this remark that the set G is the reduced Gröbner basis of I with high probability.

Algorithm 5.14 is implemented in the SINGULAR library `ffmodstd.lib` [6]. Using the command `ffmodStd` from this library, the above steps can be executed as follows:

```
> LIB "ffmodstd.lib";
> ideal G = ffmodStd(I);
> G;
  G[1] = x1+(-t2/12)*x2
  G[2] = x3^2+(-t2^2/48)*x2
  G[3] = x2^2*x3
  G[4] = x2^3
```

5.3.3 Implementation and Timings

As mentioned above, Algorithm 5.14 is implemented in SINGULAR in the library `ffmodstd.lib`. We compare its performance against the MAGMA [9, 39] command `GroebnerBasis` and the SINGULAR commands `std` and `slimgb`.

We consider twenty benchmark problems as described in appendix Section A.3 to demonstrate the efficiency of Algorithm 5.14. The timings for all benchmark examples are conducted by using SINGULAR 4.0.3 and MAGMA V2.21-11 on a Dell PowerEdge R720 machine with 16 cores and 32 threads, 2.9-3.8 GHz, and 192 GB of RAM running the Gentoo Linux operating system.

The results are summarized in Table 5.1 for Gröbner bases with sparse coefficients and in Table 5.2 for Gröbner bases with dense coefficients. Some of the computations did not finish within 12 hours. This is indicated by a dash (-). All timings are in seconds. The ideals I8 and J2 to J6 are computed using the lexicographic ordering (`lp` in SINGULAR), the remaining ideals are computed using the degree reverse lexicographical ordering (`dp` in SINGULAR).

In Table 5.1, there are examples where `GroebnerBasis` is faster than `slimgb` and, vice-versa, there are examples where `slimgb` is faster than `GroebnerBasis`. However, with respect to the average timings, one can see that `slimgb` is faster than `GroebnerBasis`. On the other hand, although `slimgb` is faster than `std` for some examples, we see that our new algorithm `ffmodStd` is even much faster than `slimgb` for most examples. However, it is less efficient for the examples in Table 5.2. In other words, Algorithm 5.14

ideal	`GroebnerBasis`	`std`	`slimgb`	`ffmodStd`
I1	17702.41	-	-	0.25
I2	15858.92	-	-	13.89
I3	1.87	65.65	27.96	1.58
I5	-	-	18671.40	26.30
I6	4642.14	-	-	1.07
I7	-	-	-	12.92
I8	0.01	5669.89	0.38	0.67
I9	-	-	-	8.51
I10	-	-	-	21.81
I11	-	-	-	5.32
J2	-	31654.76	1.18	1.25
J3	-	-	-	155.59
J6	-	33973.89	1.18	1.23
J9	-	-	-	0.14

Table 5.1: Total running times in seconds for computing a Gröbner basis (with sparse coefficients) of the considered ideals via `GroebnerBasis`, `std`, `slimgb`, and `ffmodStd`

ideal	`GroebnerBasis`	`std`	`slimgb`	`ffmodStd`
I4	80.65	2564.01	643.32	18053.34
J1	477.47	193.70	0.31	7.45
J4	11.92	251.09	3.26	151.71
J5	0.01	0.02	0.01	55.31
J7	0.01	0.02	0.02	104.46
J8	0.01	0.01	0.01	8.98

Table 5.2: Total running times in seconds for computing a Gröbner basis (with dense coefficients) of the considered ideals via `GroebnerBasis`, `std`, `slimgb`, and `ffmodStd`

performs well for Gröbner bases with sparse coefficients, while it is less efficient compared to for Gröbner bases with dense coefficients. Nonetheless, if the coefficient field K is the function field of one variable, then there exists an algorithm which beats the above methods even for Gröbner bases with dense coefficients. As a special case, the next section describes this algorithm.

5.4 Special Case for Function Fields of One Variable

The main result of this section is a fast algorithm for computing Gröbner bases of ideals in the polynomial ring $S = K[X]$ over the function field K of one variable, that is, over $K := \mathbb{Q}(t)$. The case of one variable is special because unlike in the case of multivariate variables $t_1, \ldots, t_m$, we can directly apply modular methods over $\mathbb{Q}(t)$. Moreover, instead of applying Algorithm 4.11, it suffices to apply the dense univariate rational interpolation algorithm discussed in Section 1.4. Thus the fast algorithm which we discuss later in this section uses the concepts of modular methods [2, 27] and the dense univariate rational interpolation algorithm discussed in Section 1.4.

To begin with, as in (5.5), consider the map

$$\begin{aligned} \varphi : \mathbb{Q}(t)[X] &\longrightarrow (\mathbb{Q}[t])[X], \\ f &\longmapsto f \cdot h_f, \end{aligned} \tag{5.20}$$

for suitable polynomials $h_f \in \mathbb{Q}[t]$ (such polynomials always exist). Let

$$H = \{f_1, \ldots, f_w\} \subseteq S,$$

and let $I \subseteq S$ be the ideal generated by H. Let G be the reduced Gröbner basis of I w.r.t. $>$, and let d be a bound on the number of interpolation points required for recovering the coefficients occurring in G, as defined in (5.3). Let I_{coef} be the set of coefficients, $c \in \mathbb{Q}[t]$, of $\varphi(H) = \{\varphi(h) \mid h \in H\}$ of degree greater than 0, that is,

$$I_{\text{coef}} = \{c \in \mathbb{Q}[t] \mid c \text{ is a coefficient in } \varphi(H) \text{ with } \deg(c) > 0\} \subseteq \mathbb{Q}[t]. \tag{5.21}$$

With this notation as above, the refined version of Definition 5.3.3 for the special case $K = \mathbb{Q}(t)$ is as follows:

Definition 5.4.1. Let $H \subseteq S$ be given as above. Let p be a prime not dividing any numerator or any denominator of the coefficients occurring in I_{coef}. An element $z_0 \in \mathbb{F}_p \setminus \{0\}$ is said to be *admissible* w.r.t. p and H_p if none of the elements in $I_{\text{coef},p} := I_{\text{coef}} \bmod p$ is contained in the ideal $\langle t - z_0 \rangle \subseteq \mathbb{F}_p[t]$ (or, equivalently, if none of the elements in $I_{\text{coef},p}$ vanishes when evaluated at $t = z_0$).

The notion of *lucky evaluation points* for the special case $K = \mathbb{Q}(t)$ is defined as follows:

Definition 5.4.2. Let $I = \langle H \rangle$ be given as above and let I_{coef} be defined as in (5.21). Let p be a prime not dividing any numerator or any denominator of the coefficients occurring in I_{coef}. Let $z_0 \in \mathbb{F}_p \setminus \{0\}$ be a random point satisfying the conditions in Definition 5.4.1 and let G' be the reduced Gröbner basis of the ideal $I' = \langle h \mid h = g(z_0, X), g \in I_p \rangle \subseteq \mathbb{F}_p[X]$. Furthermore, let G_p be the reduced Gröbner basis of I_p. Then z_0 is called a *lucky evaluation point* for I_p if and only if $\mathrm{Lm}(G') = \mathrm{Lm}(G_p)$. Otherwise z_0 is called *unlucky* for I_p.

Remark 5.4.3. Let $I = \langle H \rangle$ be given as above and let I_{coef} be defined as in (5.21). Let p be a prime not dividing any numerator or any denominator of the coefficients occurring in I_{coef}. For $p \gg d$, let $z_0, \ldots, z_{d-1} \in \mathbb{F}_p$ be non-zero pairwise distinct elements such that each z_i satisfies the conditions in Definition 5.4.1 and is lucky for I_p. Let G_{p,z_i} be the reduced Gröbner basis of the ideal $I_{p,z_i} = \langle h \mid h = g(z_i, X), g \in I_p \rangle \subseteq \mathbb{F}_p[X]$. Let $G_p \subseteq \mathbb{F}_p(t)[X]$ be the set of polynomials obtained by applying the dense univariate rational interpolation algorithm (see Section 1.4) coefficient-wise to those sets of polynomials, from G_{p,z_i}, whose leading monomials are the same. Then the set G_p is the reduced Gröbner basis of I_p with high probability, see Remark 5.3.11.

In this remark, the assumption $p \gg d$ is important for the following reason: Recall from Subsection 1.6.1 that for successful interpolation, the size of $\mathbb{F}_p$ needs to cover enough distinct points for the early termination, that is, $p - 1 \geq d$ (assuming that each element $z_0 \in \{0, \ldots, p-1\}$ is lucky for I_p). For our application, if the coefficient field $\mathbb{F}_p$ does not cover enough distinct points for the early termination, then the probability that the set G_p fails to be the reduced Gröbner basis of I_p, as in the above remark, is high.

Figure 5.2 illustrates the general scheme for computing the reduced Gröbner basis of I over $\mathbb{Q}(t)$. In this figure, instead of computing the reduced Gröbner bases at level 1, our algorithm computes them at level 3. For the evaluation points that are lucky with high probability (and only for those), the dense rational interpolation algorithm (see Section 1.4) then combines these results at level 4. The ideals $\langle G_{p_i} \rangle$ at this level are expected to be the same as the ideals I_{p_i} with high probability (see Remark 5.4.3). The remaining parts of the computation are carried out as in the modular algorithms described in [27].

We now turn our attention to a more detailed description of the special case algorithm. First, randomly choose a set $\mathcal{P}$ of prime numbers. At level 2, given a prime $p \in \mathcal{P}$, choose $v + 1$ distinct random evaluation points $z_0, \ldots, z_v \in \mathbb{F}_p$ for some v such that each z_i satisfies the conditions in Definition 5.4.1 and set $\mathcal{B} := \{z_0, \ldots, z_v\}$. At this level, after evaluating the ideal I_p at these points, we obtain ideals in the polynomial ring $\mathbb{F}_p[X]$ denoted by I_{p,z_i} for $0 \leq i \leq v$. At level 3, we compute the set $\mathcal{GB} = \{G_{p,z_i} \mid z_i \in \mathcal{B}\}$ of the reduced Gröbner bases of the ideals I_{p,z_i}. Given the inputs $\mathcal{GB}$ and $\mathcal{B}$, we can choose the lucky evaluation points from $\mathcal{B}$ with high probability by the DeleteUnluckyEvaluationPoints method described in Section 5.3. We may thus

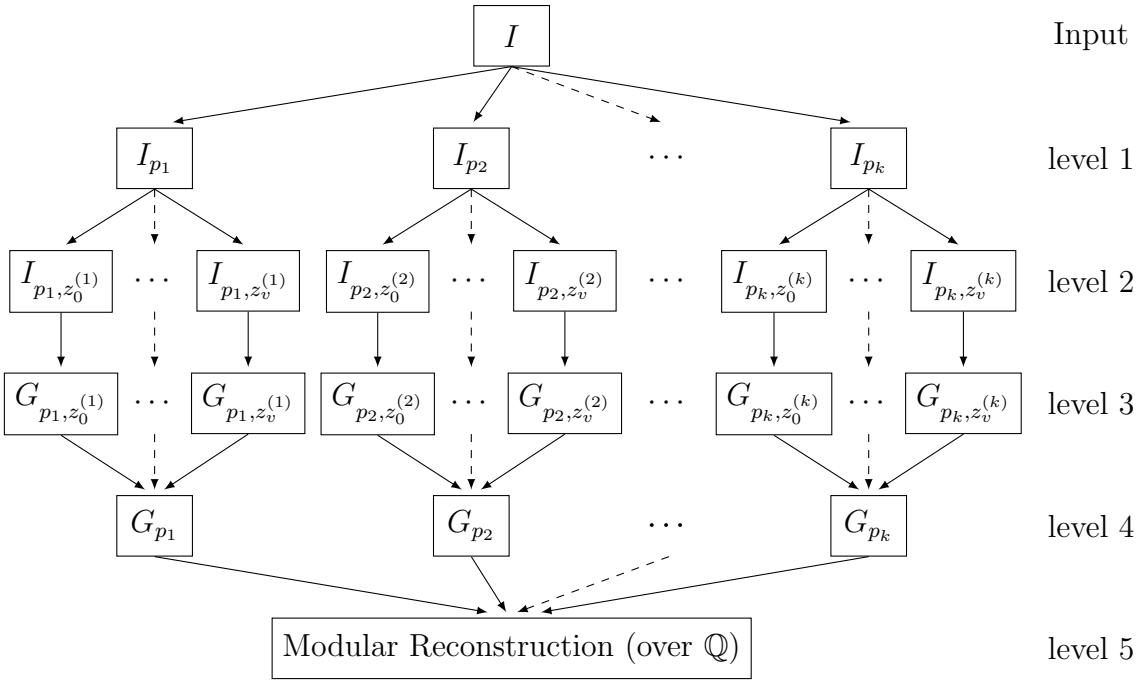

Figure 5.2: General scheme for the new algorithm (special case)

assume that all G_{p,z_i}, $z_i \in \mathcal{B}$, have the same set of leading monomials. Once we have such sets of polynomials, we can then apply the dense univariate rational interpolation algorithm coefficient-wise to a set of polynomials whose leading monomials are the same. Let $G_{p,z_0} = \{g_{0,1}, \ldots, g_{0,r}\}$. Then for $j = 1, \ldots, r$, consider the set

$$F_j = \{g_{i,j} \in G_{p,z_i} \mid \operatorname{lm}(g_{i,j}) = \operatorname{lm}(g_{0,j}),\ i = 0, \ldots, v\} \subseteq \mathbb{F}_p[X]$$

of polynomials whose leading monomials are the same. As described in Section 5.3, we do not know whether the $v+1$ distinct evaluation points are sufficient (or, equivalently, whether $v \geq d-1$ where d is as in (5.3)). Thus we apply the early termination version of the dense univariate rational interpolation algorithm (see Section 1.4 and Subsection 1.6.1) so that we can add more interpolation points if necessary. This algorithm can be applied coefficient-wise to the set of polynomials F_j over $\mathbb{F}_p$, in a similar way as described in Section 5.3, to lift the coefficients from $\mathbb{F}_p$ to $\mathbb{F}_p(t)$. Having applied this algorithm, we obtain a set G_p of polynomials in the polynomial ring $\mathbb{F}_p(t)[X]$ at level 4. Moreover, as in Section 5.3, we obtain from the set G_p:

(a) A bound d on the number of interpolation points required to recover the coefficients in G_p as well as in G.

(b) A bound on the degree of each numerator (resp. denominator) of the rational functions which occur as coefficients in G.

With respect to these bounds, we compute such G_p's for several primes $p \in \mathcal{P}$ to obtain the set of reduced Gröbner bases $\mathcal{GP} = \{G_p \mid p \in \mathcal{P}\}$ by choosing at least d distinct evaluation points which are lucky for I_p. From $\mathcal{P}$, the method DeleteUnluckyPrimesSB

[27] (see also Section 2.5) selects primes that are lucky with high probability. For those primes in $\mathcal{P}$, we apply the Chinese remainder algorithm for integers and the rational number reconstruction algorithm to the coefficients of the Gröbner bases in $\mathcal{GP}$ to obtain a set $G \subseteq \mathbb{Q}(t)[X]$ of polynomials which is the reduced Gröbner basis of I with high probability. In modular algorithms, once we have computed such a set G, a final verification test is needed since we cannot check whether $\mathcal{P}$ is sufficiently large. However, the test may be expensive, especially if $I \neq \langle G \rangle$. We thus first perform a similar test as in Section 2.5 in positive characteristic as follows:

pTestSB ([27]): *We randomly choose a prime $p \notin \mathcal{P}$ which does not divide any numerator or any denominator of the coefficients occurring in I_{coef} as in (5.21). The test is positive if and only if G_p is the reduced Gröbner basis of I_p. That is, we explicitly test whether $G_p \subseteq I_p$ and $(f_i \bmod p) \in \langle G_p \rangle$ for $i = 1, \ldots, w$.*

Based on the above description, Algorithm 5.18, which is a modified version of Algorithm 1 in [27], computes the reduced Gröbner basis of the input ideal. This algorithm calls Algorithms 5.16 and 5.17, which are almost the same, to compute the reduced Gröbner bases of the ideals at level 1 with high probability. The former algorithm only computes the initial Gröbner basis G_p without bounds on the number of interpolation points and on the degrees of the numerators and denominators of the coefficients in G_p, while the latter algorithm computes the remaining Gröbner bases w.r.t. some given bounds, where these bounds are obtained from the output of Algorithm 5.16.

Remark 5.4.4. In Algorithm 5.16 (resp. 5.17), we have parallelized the for-loop starting in line 4 (resp. 9). Moreover, some parts of Algorithm 5.18 are parallelized as described in Remark 2.5.1.

5.4.1 An Illustrative Example

In this subsection, we consider an example illustrating the method discussed above. Consider the ideal

$$I = \langle H \rangle = \langle (t+3) \cdot y^2 + (-t) \cdot x,\ (t-1) \cdot xy + y \rangle \subseteq \mathbb{Q}(t)[x, y].$$

A Singular computation shows that the reduced Gröbner basis of I with respect to the degree reverse lexicographical ordering (`dp` in Singular) with $x > y$ is

$$G = \{(t+3) \cdot y^2 + (-t) \cdot x\,,\ (t-1) \cdot xy + y\,,\ (t-1) \cdot x^2 + x\}\,.$$

In the following, we show how this basis is obtained using Algorithm 5.18. Since none of the monomials in I vanishes modulo the prime $p = 97$, we may choose this prime for the modular computation. For simplicity, we start by taking the ideals

$$P_{p,0} = \langle t - 5 \rangle,\ P_{p,1} = \langle t - 9 \rangle,\ P_{p,2} = \langle t - 3 \rangle,\ \text{and}\ P_{p,3} = \langle t - 13 \rangle$$

Algorithm 5.16 Gröbner bases over $\mathbb{F}_p(t)[X]$ without known bounds

Input: An ideal $I_p = \langle H_p \rangle \subseteq \mathbb{F}_p(t)[X]$ and a prime number p, where $H = \{f_1, \ldots, f_w\}$.

Output: G_p, the reduced Gröbner basis of I_p w.r.t. $>$.

let $\mathcal{B}$ be a set of distinct random points $a \in \mathbb{F}_p$ which are admissible w.r.t. H_p
$\mathcal{GB} \longleftarrow \{\}$
$e \longleftarrow 0,\ G \longleftarrow \{\}$
loop
  for $a \in \mathcal{B}$ **do**
    $I_{p,a} \longleftarrow I_p|_{t=a}$
    compute the reduced Gröbner basis $G_{p,a}$ of $I_{p,a} \subseteq \mathbb{F}_p[X]$ w.r.t. $>$
    $\mathcal{GB} \longleftarrow \mathcal{GB} \cup \{G_{p,a}\}$
  $(\mathcal{GB}, \mathcal{B}) \longleftarrow$ DeleteUnluckyEvaluationPoints$(\mathcal{GB}, \mathcal{B})$
  let $G_{0,a_0}, \ldots, G_{k,a_k}$ be the elements in $\mathcal{GB}$
  $f_p \longleftarrow \prod_{a \in \mathcal{B}} (t - a) \subseteq \mathbb{F}_p[t]$
  let $h_{0,1}, \ldots, h_{0,r}$ be the elements in G_{0,a_0}
  for $j = e + 1, \ldots, r$ **do**
    consider the set

$$F_j = \{g \in G_{i,a_i} \mid \mathrm{lm}(g) = \mathrm{lm}(h_{0,j}),\ 0 \leq i \leq k\} \subseteq \mathbb{F}_p[X]$$

    of polynomials whose leading monomials are the same
    lift the coefficients of F_j to $\mathbb{F}_p(t)$ by applying the early termination version of the dense rational interpolation algorithm (see Subsection 1.6.1) coefficient-wise w.r.t. f_p
    if all coefficients of F_j are successfully lifted to $\mathbb{F}_p(t)$ **then**
      let g_j be the polynomial obtained by lifting F_j to $\mathbb{F}_p(t)[X]$
      $G \longleftarrow G \cup \{g_j\}$
      $e \longleftarrow e + 1$
    else
      break out of the for-loop
  if $e = r$ **then**
    break out of the loop
  enlarge $\mathcal{B}$
$G_p \longleftarrow G$
return G_p

Algorithm 5.17 Gröbner bases over $\mathbb{F}_p(t)[X]$ with given bounds

Input: An ideal $I_p = \langle H_p \rangle \subseteq \mathbb{F}_p(t)[X]$, a prime number p, $d \in \mathbb{N}$, and $N, D \subseteq \mathbb{N}$, where $H = \{f_1, \ldots, f_w\}$, d is a bound on the number of interpolation points and N and D are degree bounds for the numerators and denominators of the coefficients in the result as explained in the text.

Output: G_p, the reduced Gröbner basis of I_p w.r.t. $>$.

1: choose d distinct random points $a_0, \ldots, a_{d-1} \in \mathbb{F}_p$ such that each a_i is admissible w.r.t. p and H_p
2: $\mathcal{B} \longleftarrow \{a_0, \ldots, a_{d-1}\}$
3: $\mathcal{GB} \longleftarrow \{\}$
4: **for** $i = 0, \ldots, d-1$ **do**
5: compute the reduced Gröbner basis G_{p,a_i} of $I_{p,a_i} := I_p|_{t=a_i} \subseteq \mathbb{F}_p[X]$ w.r.t. $>$
6: $\mathcal{GB} \longleftarrow \mathcal{GB} \cup \{G_{p,a_i}\}$
7: $(\mathcal{GB}, \mathcal{B}) \longleftarrow$ DeleteUnluckyEvaluationPoints$(\mathcal{GB}, \mathcal{B})$
8: **if** $\#\mathcal{B} =: d' < d$ **then**
9: go to line 1 to replace the unlucky evaluation points with lucky ones
10: $f_p \longleftarrow \prod_{a \in \mathcal{B}} (t - a) \subseteq \mathbb{F}_p[t]$
11: let $h_{0,1}, \ldots, h_{0,r}$ be the elements in G_{p,a_0}
12: **for** $j = 1, \ldots, r$ **do**
13: consider the set

$$F_j = \{g \in G_{p,a_i} \mid \mathrm{lm}(g) = \mathrm{lm}(h_{0,j}),\ 0 \leq i < d'\} \subseteq \mathbb{F}_p[X]$$

of polynomials whose leading monomials are the same
14: apply the dense rational interpolation algorithm (first Algorithm 1.1 and then Algorithm 1.2, see Section 1.4) w.r.t. f_p and the degree bounds N and D coefficient-wise to each set F_j to obtain a set G_p of polynomials in $\mathbb{F}_p(t)[X]$
15: **return** G_p

Algorithm 5.18 Gröbner bases algorithm over $K = \mathbb{Q}(t)$ (`ffmodStd`)

Input: An ideal $I = \langle H \rangle \subseteq S = K[X]$, where $K = \mathbb{Q}(t)$ and $H = \{f_1, \ldots, f_w\}$.

Output: $G \subseteq S$, the reduced Gröbner basis of I w.r.t. $>$.

1: let I_{coef} be defined as in (5.21)
2: choose $\mathcal{P}$, a set of random primes such that none of the primes in $\mathcal{P}$ divides any numerator or any denominator of the coefficients occurring in I_{coef}
3: pick an element p from $\mathcal{P}$
4: let $G_p \subseteq \mathbb{F}_p(t)[X]$ be the output of Algorithm 5.16 applied to I_p
5: $\mathcal{GP} \longleftarrow \{G_p\}$
6: let (N, D) be the pair of ordered sets containing the degrees of the polynomials in the numerators (resp. denominators) of the coefficients of G_p
7: $d_N \longleftarrow \max N$, $d_D \longleftarrow \max D$, $d \longleftarrow d_N + d_D + 2$
8: **loop**
9: **for** $q \in \mathcal{P} \setminus \{p\}$ **do**
10: let $G_q \subseteq \mathbb{F}_q(t)[X]$ be the output of Algorithm 5.17 applied to I_q and (d, N, D)
11: $\mathcal{GP} \longleftarrow \mathcal{GP} \cup \{G_q\}$
12: $(\mathcal{GP}, \mathcal{P}) \longleftarrow$ DeleteUnluckyPrimesSB$(\mathcal{GP}, \mathcal{P})$
13: lift $(\mathcal{GP}, \mathcal{P})$ to $G \subseteq S$ by applying the Chinese remainder algorithm and the Farey rational map
14: **if** pTestSB$(I, G, \mathcal{P})$ **then**
15: **if** $I \subseteq \langle G \rangle$ **then**
16: **if** G is the reduced Gröbner basis of $\langle G \rangle$ **then**
17: **return** G
18: enlarge $\mathcal{P}$

in the polynomial ring $\mathbb{F}_{97}[t]$, that is, $\mathcal{B} = \{5, 9, 3, 13\} \subseteq \mathbb{F}_{97}$. Let φ (resp. I_{coef}) be defined as in (5.20) (resp. (5.21)). Then we have

$$I_{\mathrm{coef}} = \{t + 3,\ t,\ t - 1\} \subseteq \mathbb{Q}[t].$$

Since none of the elements in $I_{\mathrm{coef},p} = I_{\mathrm{coef}} \bmod p \subseteq \mathbb{F}_p[t]$ is contained in the ideal $P_{p,i}$ for $i = 0, \ldots, 3$, by Definition 5.4.1 the points $5, 9, 3$ and 13 are admissible w.r.t. p and H_p. Set $(a_0, \ldots, a_3) := (5, 9, 3, 13)$. With respect to $P_{p,i}$, the reduced Gröbner bases of $I_{p,a_i} = \varphi(I_p) \bmod P_{p,i}$ for $i = 0, 1, 2, 3$, respectively are

$$\begin{aligned}
G_{p,a_0} &= \{y^2 - 37x\,,\ xy - 24y\,,\ x^2 - 24x\}\,,\\
G_{p,a_1} &= \{y^2 - 25x\,,\ xy - 12y\,,\ x^2 - 12x\}\,,\\
G_{p,a_2} &= \{y^2 + 48x\,,\ xy - 48y\,,\ x^2 - 48x\}\,,\ \text{and}\\
G_{p,a_3} &= \{y^2 - 19x\,,\ xy - 8y\,,\ x^2 - 8x\}\,.
\end{aligned}$$

By the deleteUnluckyEvaluationPoints method, we see that none of the evaluation points used in the computation is unlucky. Now by applying Algorithm 1.1 and then Algorithm 1.2, as explained in the text, coefficient-wise to the set of polynomials

$$\begin{aligned}
F_{1,p} &= \{y^2 - 37x\,,\ y^2 - 25x\,,\ y^2 + 48x\,,\ y^2 - 19x\}\,,\\
F_{2,p} &= \{xy - 24y\,,\ xy - 12y\,,\ xy - 48y\,,\ xy - 8y\}\,,\ \text{and}\\
F_{3,p} &= \{x^2 - 24x\,,\ x^2 - 12x\,,\ x^2 - 48x\,,\ x^2 - 8x\}
\end{aligned}$$

w.r.t. the polynomial $f_p = \prod_{a \in \mathcal{B}}(t - a) \bmod p \in \mathbb{F}_p[t]$, we obtain the set

$$G_p = \left\{ y^2 - \frac{t}{t+3}x\,,\ xy + \frac{1}{t-1}y\,,\ x^2 + \frac{1}{t-1}x \right\} \subseteq \mathbb{F}_p(t)[x, y]. \tag{5.22}$$

Let us now take one more ideal to see whether the evaluation points used in the computations are sufficient. Since none of the elements in $I_{\mathrm{coef},p}$ is contained in $P_{p,4} = \langle t - 17 \rangle$, we may take this ideal. The reduced Gröbner basis of $I_{p,a_4} = \varphi(I_p) \bmod P_{p,4}$ with $a_4 = 17 \in \mathcal{B} = \{5, 9, 3, 13, 17\}$ is

$$G_{p,a_4} = \{y^2 + 4x\,,\ xy - 6y\,,\ x^2 - 6x\}\,.$$

By applying, once more, Algorithm 1.1 and then Algorithm 1.2 coefficient-wise to the set of polynomials

$$\begin{aligned}
F'_{1,p} &= F_{1,p} \cup \{y^2 + 4x\}\,,\ F'_{2,p} = F_{2,p} \cup \{xy - 6y\}\,,\ \text{and}\\
F'_{3,p} &= F_{3,p} \cup \{x^2 - 6x\}
\end{aligned}$$

w.r.t. the polynomial $f_p = \prod_{a \in \mathcal{B}}(t - a) \bmod p \in \mathbb{F}_p[t]$, we obtain, again, the set

$$G_p = \left\{ y^2 - \frac{t}{t+3}x\,,\ xy + \frac{1}{t-1}y\,,\ x^2 + \frac{1}{t-1}x \right\} \subseteq \mathbb{F}_p(t)[x, y]. \tag{5.23}$$

Since the sets G_p in (5.22) and (5.23) coincide, the early termination version of the dense rational interpolation algorithm (see Section 1.4 and Subsection 1.6.1) indicates that we have correctly lifted the coefficients in $\mathbb{F}_p$ to $\mathbb{F}_p(t)$ with high probability. From the coefficients of G_p, we obtain a bound on the number of interpolation points and on the degree of each numerator (resp. denominator) of the rational functions which occur as coefficients of G required to recover the coefficients occurring in G_p as well as in G. The next step is now to repeat this process in the same way as in the modular algorithms described in [27] (see also Section 2.5) w.r.t. these bounds. However, to simplify the presentation, we first show that the prime p is already sufficiently large to lift the modular coefficients back to $\mathbb{Q}$. To see this, note that it is clear that the prime p is lucky. At this point, we have to change the current base ring in SINGULAR to characteristic zero in order to pull the modular coefficients back to the rational numbers. For this, we use the SINGULAR command `farey` as follows:

```
/* rational reconstruction */
> ring S = (0,t), (x,y), dp;
> ideal Gp = y^2-t/(t+3)*x, xy+1/(t-1)*y, x^2+1/(t-1)*x;
> ideal G = farey(Gp, 97);
> G;
  G[1] = y^2-t/(t+3)*x
  G[2] = xy+1/(t-1)*y
  G[3] = x^2+1/(t-1)*x
```

We see that the computed result already coincides with the reduced Gröbner basis stated above. This shows that p is sufficiently large. We therefore skip some of the steps in Algorithm 5.18, such as the Chinese remainder algorithm, the pTESTSB, and the final verification test.

Algorithm 5.18 is implemented in SINGULAR in the library `ffmodstd.lib` [6]. Using the command `ffmodStd` from this library, the above steps can be conveniently executed as follows:

```
> LIB "ffmodstd.lib";
> ring S = (0,t), (x,y), dp;
> ideal I = (t-1)*x*y+y, (t+3)*y^2+(-t)*x;
> ideal G = ffmodStd(I);
> G;
  G[1] = y^2-t/(t+3)*x
  G[2] = xy+1/(t-1)*y
  G[3] = x^2+1/(t-1)*x
```

Ideal	ffmodStd	
	Algorithm 5.14	Algorithm 5.18
I1	0.25	0.01
I2	13.89	0.25
I3	1.58	0.34
I4	18053.34	33.67
I5	26.30	1.09

Table 5.3: Total running times in seconds for computing a Gröbner basis of the considered ideals via `ffmodStd`

5.4.2 Implementation and Timings

As mentioned above, Algorithm 5.18 is implemented in SINGULAR in the library `ffmodstd.lib`. We compare its performance against Algorithm 5.14. The results are summarized in Table 5.3. The examples in this table are the first four examples from Table 5.1 together with I4 from Table 5.2.

Looking at example I4 in Table 5.3, we see that Algorithm 5.18 is much faster. Moreover, the results from this subsection can be used for further improvement of Algorithm 5.14, see the next section.

5.5 Further Optimizations

In Figure 5.1, if we use Algorithm 5.18 or a variant of Buchberger's algorithm over the field $K = \mathbb{Q}(z)$ instead of using levels 2 to 5 to compute the reduced Gröbner basis G_{b_j} of $\widetilde{I}_{b_j}$, where $b_j = (p_1^j, \ldots, p_m^j)$, Algorithm 5.14 can be further improved as in Algorithm 5.19 which computes the reduced Gröbner basis of I with high probability. In lines 8 and 13 of this algorithm, we use Algorithm 5.18 only if there is coefficient swell, otherwise we use the variant of Buchberger's algorithm from [10]. In the implementation of this algorithm, we use the bounds d, N and D defined above internally.

5.5.1 Implementation and Timings

Algorithm 5.19 is implemented in SINGULAR in the library `ffmodstd.lib`. We compare its performance against Algorithm 5.14. The results are summarized in Table 5.4 for Gröbner bases with sparse coefficients and in Table 5.5 for Gröbner bases with dense coefficients. In these tables, we see that for most examples, Algorithm 5.19 is faster than Algorithm 5.14.

The overall timings are given in Table 5.6 and Table 5.7. In these tables, we use the same command `ffmodStd` as before for Algorithm 5.19.

Algorithm 5.19 Gröbner bases over K (`ffmodStd`)

Input: An ideal $I = \langle H \rangle \subseteq S = K[X]$, where $K = \mathbb{Q}(T)$ and $H = \{f_1, \ldots, f_w\}$.

Output: $G \subseteq S$, the reduced Gröbner basis of I w.r.t. $>$.

1: choose a random non-zero point $s = (s_1, \ldots, s_m) \in \mathbb{Q}^m$ such that none of the denominators of the coefficients in H vanishes when evaluated at $T = s$

2: let $\widetilde{I} \subseteq \mathbb{Q}(Tz)[X]$ be defined as in (5.6) w.r.t. z and s

3: choose m distinct random prime numbers $p_1, \ldots, p_m$

4: $j \longleftarrow 1,\ b \longleftarrow (p_1, \ldots, p_m)$

5: $b_j \longleftarrow b$

6: $I_{b_j} \longleftarrow \widetilde{I}|_{(T,z,X)=(b_j,z,X)} \subseteq \mathbb{Q}(z)[X]$

7: $G \longleftarrow \{\},\ e \longleftarrow 0$

8: let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.18 applied to I_{b_j}

9: **while** $e < r$ **do**

10: $\quad j \longleftarrow j + 1$

11: $\quad b_j \longleftarrow (p_1^j, \ldots, p_m^j)$

12: $\quad I_{b_j} \longleftarrow \widetilde{I}|_{(T,z,X)=(b_j,z,X)} \subseteq \mathbb{Q}(z)[X]$

13: $\quad$ let $G_{b_j} \subseteq \mathbb{Q}(z)[X]$ be the output of Algorithm 5.18 applied to I_{b_j}

14: $\quad$ let $g_{1,1}, \ldots, g_{1,r}$ be the elements in G_{b_1}

15: $\quad$ **for** $i = e + 1, \ldots, r$ **do**

16: $\quad\quad$ consider the set

$$F_i = \{g_{k,i} \in G_{b_k} \mid \mathrm{lm}(g_{k,i}) = \mathrm{lm}(g_{1,i}),\ 1 \leq k \leq j\} \subseteq \mathbb{Q}(z)[X]$$

$\quad\quad$ of polynomials whose leading monomials are the same

17: $\quad\quad$ lift the coefficients of F_i to $\mathbb{Q}(T)$ by applying the early termination version of the sparse multivariate interpolation algorithm, Algorithm 1.4, coefficient-wise to the set of polynomials F_i

18: $\quad\quad$ **if** all coefficients of F_i are successfully lifted to $\mathbb{Q}(T)$ **then**

19: $\quad\quad\quad$ let g_i be the polynomial obtained by lifting the F_i to $\mathbb{Q}(T)[X]$

20: $\quad\quad\quad$ $G \longleftarrow G \cup \{g_i\}$

21: $\quad\quad\quad$ $e \longleftarrow e + 1$

22: $\quad\quad$ **else**

23: $\quad\quad\quad$ break out of the for-loop

24: **if** $I \not\subseteq \langle G \rangle$ or G is not the reduced Gröbner basis of $\langle G \rangle$ **then**

25: $\quad$ reapply Algorithm 5.19 by choosing different evaluation points in lines 1 and 3

26: **return** G

Ideal	`ffmodStd`	
	Algorithm 5.14	Algorithm 5.19
I6	1.07	3.82
I7	12.92	2.18
I8	0.67	0.62
I9	8.51	25.70
I10	21.81	3.44
I11	5.32	0.92
J2	1.25	0.36
J3	155.59	23.35
J6	1.23	0.36
J9	0.14	0.13

Table 5.4: Total running times in seconds for computing a Gröbner basis (with sparse coefficients) of the considered ideals via `ffmodStd`

Ideal	`ffmodStd`	
	Algorithm 5.14	Algorithm 5.19
J1	7.45	1.13
J4	151.71	4.35
J5	55.31	0.90
J7	104.46	1.96
J8	8.98	0.97

Table 5.5: Total running times in seconds for computing a Gröbner basis (with dense coefficients) of the considered ideals via `ffmodStd`

Example	Magma	Singular			
Ideal	`Groebner Basis`	`std`	`slimgb`	`ffmodStd`	
				one core	32 cores
I1	17702.41	-	-	0.01	0.01
I2	15858.92	-	-	0.74	0.25
I3	1.87	65.65	27.96	1.04	0.34
I5	-	-	18671.40	2.81	1.09
I6	4642.14	-	-	16.51	3.82
I7	-	-	-	12.73	2.18
I8	0.01	5669.89	0.38	13.38	0.62
I9	-	-	-	252.43	25.70
I10	-	-	-	22.52	3.44
I11	-	-	-	3.52	0.92
J2	-	31654.76	1.18	15.90	0.36
J3	-	-	-	3494.64	23.35
J6	-	33973.89	1.18	13.45	0.36
J9	-	-	-	0.20	0.13

Table 5.6: Total running times in seconds for computing a Gröbner basis (with sparse coefficients) of the considered ideals via `GroebnerBasis`, `std`, `slimgb`, and `ffmodStd`

Example	Magma	Singular			
Ideal	`Groebner Basis`	`std`	`slimgb`	`ffmodStd`	
				one core	32 cores
I4	80.65	2564.01	643.32	88.32	33.67
J1	477.47	193.70	0.31	75.52	1.13
J4	11.92	251.09	3.26	4205.75	4.35
J5	0.01	0.02	0.01	292.40	0.90
J7	0.01	0.02	0.02	984.92	1.96
J8	0.01	0.01	0.01	211.21	0.97

Table 5.7: Total running times in seconds for computing a Gröbner basis (with dense coefficients) of the considered ideals via `GroebnerBasis`, `std`, `slimgb`, and `ffmodStd`

Some Remarks on the Timings

Modular algorithms over $\mathbb{Q}$ perform well at time consuming examples where huge coefficients occur. However, we have seen in Example 5.2.1 that over the field $K = \mathbb{Q}(T)$ modular algorithms over $\mathbb{Q}$ alone cannot solve the problems that arise from the arithmetic of K. The probabilistic algorithm `ffmodStd` described in this chapter solves these problems. However, its efficiency depends on the type of the input. To see this, we distinguish two classes of benchmark problems:

(1) *type S-examples*: These are examples whose Gröbner bases have *sparse coefficients*.

(2) *type D-examples*: These are examples whose Gröbner bases have *dense coefficients*.

In type S-examples, if there is coefficient swell in the intermediate computations, our algorithm `ffmodStd` is faster than other known methods especially for time consuming examples. More than half of the benchmark problems are type S-examples, see Table 5.6. However, if no coefficient swell occurs, then in comparison to `slimgb`, our algorithm, Algorithm 5.14, is less efficient, check, for example, ideal I8 from Table 5.1 w.r.t. the degree reverse lexicographic ordering (`dp` in SINGULAR). Since we are applying modular algorithms, Algorithm 5.14 is even slower for small examples.

On the other hand, suppose there is coefficient swell in the intermediate computations for examples of type D. In this chapter, examples of this type are not much investigated. This is because, as mentioned in the introduction of Chapter 4, an algorithm that is efficient for sparse polynomials may not be for dense cases. Nevertheless, in the case where the input ideals are in a polynomial ring with coefficients in $K = \mathbb{Q}(t)$, Algorithm 5.18 described in Section 5.4 solves the coefficient swell problem. An example of type D is I4, see Table 5.7. In this table, we see that Algorithm 5.19 is faster than `GroebnerBasis`. It is also much faster than `slimgb` and `std` even if we do not run it in parallel, that is, even if we run it on a single core.

If no coefficient swell occurs for examples of type D, our algorithm is slower than other known methods. In order to minimize the time difference between them as in Table 5.2, we use `slimgb` which is a variant of Buchberger's algorithm [10] to compute the reduced Gröbner bases of ideals in a polynomial ring with coefficients in $K = \mathbb{Q}(t)$, see Algorithm 5.19. However, since we cannot decide which command to use, either `slimgb` or Algorithm 5.18, without knowing the output, our implementation tries both Algorithm 5.18 (the command internally used for this algorithm is `ffmodStdOne`) and `slimgb` in parallel. By using the SINGULAR library `parallel.lib` [37], this can be done in SINGULAR as follows:

```
> LIB "parallel.lib";
> ring r = (0,t), (x,y), dp;
> ideal I = tx+y, y2-txy;
> list commands = list("ffmodStdOne", "slimgb");
> list args = list(list(I), list(I));
```

```
> list L = parallelWaitFirst(commands, args); // perform both tasks in
         // parallel and wait for the first of them to finish
> if(typeof(L[2]) != "none")
{
   list M = "slimgb", L[2]; // if slimgb finishes first
}
else
{
   list M = "ffmodStdOne", L[1]; // if ffmodStdOne finishes first
}
> M;
```

Once we run this, we take the output M from whatever method finishes first and continue the remaining computations w.r.t. this method. The examples in Table 5.7 are of type D, see also the timings in Table 5.5. Note that one can also run the above tasks in a machine with a single core. In this case, we use Algorithm 5.18 by default in our implementation.

Chapter 6

Conclusion and Future Work

In this thesis, we have investigated modular algorithms over extension fields of $\mathbb{Q}$ with a focus on Gröbner bases and syzygy modules.

To compute a Gröbner basis of an ideal I in a polynomial ring with coefficients in K where $K = \mathbb{Q}(\alpha)$ is a number field, we have applied modular methods w.r.t. different prime numbers. However, with this approach there might still be some coefficient swell in the modular computations. Furthermore, we have observed that the efficiency of computing such a basis over K strongly depends on the degree of the field extension. To tackle these problems, the main innovation in our new algorithm `nfmodStd` is to factorize the minimal polynomial of α modulo suitable primes such that these factors are squarefree and irreducible, thereby reducing the degree of the modular field extensions. One advantage of this algorithm is that it is twofold parallel which increases the speed of the computation. The algorithm `nfmodStd` is implemented in the SINGULAR library `nfmodstd.lib` [5]. To investigate the performance of `nfmodStd`, we have compared the computation time of this algorithm against other known methods such as the MAGMA command `GroebnerBasis` and the SINGULAR commands `std` and `modStd`. The results show that our algorithm `nfmodStd` performs extremely well in comparison to the above mentioned methods.

Furthermore, the implementation of this algorithm is not restricted to ideals, it also works for modules. More so, by slightly modifying this algorithm, we computed generating sets for syzygy modules. In fact, two slightly different modular algorithms are presented. To examine their performance, several examples have been considered, and timing tables comparing them to the MAGMA command `SyzygyModule` and the SINGULAR command `syz` are presented. For both approaches, the tables show that our algorithms perform better than the above methods for most of the examples considered here. However, our second approach, which is implemented in the SINGULAR library `nfmodsyz.lib` [7], outperforms the above methods by far, including our former approach, for all examples.

Another task investigated in this thesis is an efficient method for computing a Gröbner basis of an ideal I in a polynomial ring with coefficients in K where $K = \mathbb{Q}(t_1, \ldots, t_m)$ is an algebraic function field. Starting from the idea highlighted in the introduction, we have studied the modular algorithm `ffmodStd` for computing such a basis which combines modular methods w.r.t. different prime numbers and sparse interpolation of multivariate

rational functions. In this study, we have noticed that modular algorithms over $\mathbb{Q}$ alone cannot address the coefficient swell problem in K and, vice-versa, the methods which we used to avoid the extreme growth of the intermediate coefficients in K alone cannot improve the efficiency of the total computation. Thus in order to attack this problem, we used a combination of the two modular algorithms which we call modular algorithms over K. This combined approach is implemented in the SINGULAR library `ffmodstd.lib` [6]. To recover the coefficients of the reduced Gröbner basis of I, we use the sparse rational function interpolation algorithm from [16]. The sparse rational function interpolation algorithm uses an algorithm which interpolates dense univariate rational functions (resp. sparse multivariate polynomials) in a black box from given numerical data. Since the dense (resp. sparse) algorithm requires a bound on the degrees (resp. number of terms) of the numerators and denominators of the black box rational functions, we use the early termination version of these algorithms. Moreover, for further optimizations we have introduced special case modular algorithms over algebraic function fields of one variable using the early termination version of the dense univariate rational interpolation algorithm. The efficiency of the algorithm `ffmodStd` combined with this algorithm is demonstrated by several examples. The presented timings show that `ffmodStd` is faster than `GroebnerBasis` and the SINGULAR commands `std` and `slimgb`, especially for time consuming examples whose Gröbner bases have sparse coefficients.

On the other hand, from the timing tables, we have also observed that `ffmodStd` is limited in terms of efficiency compared to other known methods for input ideals whose Gröbner bases have dense coefficients. Howbeit, our special case algorithm included in `ffmodStd` outperforms the other methods mentioned above for most examples, even if the reduced Gröbner bases of the input ideals have dense coefficients. In fact, we have considered two classes of examples: type S-examples and type D-examples. These are examples whose Gröbner bases have sparse (resp. dense) coefficients. It is clear that modular algorithms perform well for time consuming examples. In other words, if there is no coefficient swell in the intermediate computations, then there is no point to use modular algorithms. From the presented timing tables, we have observed that those examples where `ffmodStd` is slower than `GroebnerBasis`, `std`, or `slimgb` are examples of type D and that their Gröbner bases have no intermediate coefficient swell whereas for examples of type S, `ffmodStd` is much faster. Thus, for examples of type S, provided that there is intermediate coefficient swell, it is preferable to use `ffmodStd` for computing Gröbner bases over K.

To improve the efficiency of the algorithm `ffmodStd`, especially for examples of type D, we internally run the algorithms `slimgb` and the special case algorithm in parallel to compute the reduced Gröbner basis of an ideal with coefficients in $K = \mathbb{Q}(t)$, where t is a single parameter. This allows us to choose the method which finishes the computation first and to use this algorithm for the remaining computations. Note that this method is only applied for the case $m > 1$, that is, if K is a function field of at least two parameters. With this method included in `ffmodStd`, we have noticed that the running time is indeed improved for type D examples as desired. However, we believe that this requires further investigation since an efficient algorithm recovering sparse rational functions may not be optimal for dense cases. This is a possible direction for future research.

Appendices

Appendix A

Benchmark Problems

A.1 Benchmark Problems for `nfmodStd`

The ideals given below are the benchmark problems used to demonstrate the efficiency of Algorithm 2.7. Some of these are chosen from [2, 35] (the ideals `I1` and `I2` are from [2], `I6` and `I7` are from [35]) where the coefficients are replaced by a random algebraic number. The generators of the remaining ideals except I3b (which is a cyclic ideal where the coefficients are replaced by random algebraic numbers) are constructed from randomly chosen monomials and coefficients.

1. ```
   ring R = (0,a), (x,y,z), dp;
   minpoly = (a^2+1);
   poly f1 = (a+8)*x^2*y^2+5*x*y^3+(-a+3)*x^3*z+x^2*y*z;
   poly f2 = x^5+2*y^3*z^2+13*y^2*z^3+5*y*z^4;
   poly f3 = 8*x^3+(a+12)*y^3+x*z^2+3;
   poly f4 = (-a+7)*x^2*y^4+y^3*z^3+18*y^3*z^2;
   ideal I1 = f1, f2, f3, f4;
   ```

2. ```
   ring R = (0,a), (x,y,z), dp;
   minpoly = (a^5+a^2+2);
   poly f1 = 2*x*y^4*z^2+(a-1)*x^2*y^3*z+(2*a)*x*y*z^2+7*y^3+(7*a+1);
   poly f2 = 2*x^2*y^4*z+(a)*x^2*y*z^2-x*y^2*z^2+(2*a+3)*x^2*y*z-12*x+(12*a)*y;
   poly f3 = (2*a)*y^5*z+x^2*y^2*z-x*y^3*z+(-a)*x*y^3+y^4+2*y^2*z;
   poly f4 = (3*a)*x*y^4*z^3+(a+1)*x^2*y^2*z-x*y^3*z+4*y^3*z^2+(3*a)*x*y*z^3+4*z^2-x+(a)*y;
   ideal I2 = f1, f2, f3, f4;
   ```

3. ```
   ring R = (0,a), (v,w,x,y,z), dp;
   minpoly = (a^7-7*a+3);
   poly f1 = (a)*v+(a-1)*w+x+(a+2)*y+z;
   poly f2 = v*w+(a-1)*w*x+(a+2)*v*y+x*y+(a)*y*z;
   poly f3 = (a)*v*w*x+(a+5)*w*x*y+(a)*v*w*z+(a+2)*v*y*z+(a)*x*y*z;
   poly f4 = (a-11)*v*w*x*y+(a+5)*v*w*x*z+(a)*v*w*y*z+(a)*v*x*y*z+(a)*w*x*y*z;
   poly f5 = (a+3)*v*w*x*y*z+(a+23);
   ideal I3a = f1, f2, f3, f4, f5;
   ```

4. ```
   ring R = (0,a), (u,v,w,x,y,z), dp;
   minpoly = (a^7-7*a+3);
   poly f1 = (a)*u+(a+2)*v+w+x+y+z;
   poly f2 = u*v+v*w+w*x+x*y+(a+3)*u*z+y*z;
   poly f3 = u*v*w+v*w*x+(a+1)*w*x*y+u*v*z+u*y*z+x*y*z;
   ```

```
poly f4 = (a-1)*u*v*w*x+v*w*x*y+u*v*w*z+u*v*y*z+u*x*y*z+w*x*y*z;
poly f5 = u*v*w*x*y+(a+1)*u*v*w*x*z+u*v*w*y*z+u*v*x*y*z+u*w*x*y*z+v*w*x*y*z;
poly f6 = u*v*w*x*y*z+(-a+2);
ideal I3b = f1, f2, f3, f4, f5, f6;
```

5.
```
ring R = (0,a), (w,x,y,z), dp;
minpoly = (a^6+a^5+a^4+a^3+a^2+a+1);
poly f1 = (a+5)*w^3*x^2*y+(a-3)*w^2*x^3*y+(a+7)*w*x^2*y^2;
poly f2 = (a)*w^5+(a+3)*w*x^2*y^2+(a^2+11)*x^2*y^2*z;
poly f3 = (a+7)*w^3+12*x^3+4*w*x*y+(a)*z^3;
poly f4 = 3*w^3+(a-4)*x^3+x*y^2;
ideal I4 = f1, f2, f3, f4;
```

6.
```
ring R = (0,a), (w,x,y,z), dp;
minpoly = (a^12-5*a^11+24*a^10-115*a^9+551*a^8-2640*a^7+12649*a^6-2640*a^5+551*a^4
        -115*a^3+24*a^2-5*a+1);
poly f1 = (2*a+3)*w*x^4*y^2+(a+1)*w^2*x^3*y*z+2*w*x*y^2*z^3+(7*a-1)*x^3*z^4;
poly f2 = 2*w^2*x^4*y+w^2*x*y^2*z^2+(-a)*w*x^2*y^2*z^2+(a+11)*w^2*x*y*z^3
          -12*w*z^6+12*x*z^6;
poly f3 = 2*x^5*y+w^2*x^2*y*z-w*x^3*y*z-w*x^3*z^2+(a)*x^4*z^2+2*x^2*y*z^3;
poly f4 = 3*w*x^4*y^3+w^2*x^2*y*z^3-w*x^3*y*z^3+(a+4)*x^3*y^2*z^3+3*w*x*y^3*z^3
         +(4*a)*y^2*z^6-w*z^7+x*z^7;
ideal I5 = f1, f2, f3, f4;
```

7.
```
ring R = (0,a), (u,v,w,x,y,z), dp;
minpoly = (a^2+5*a+1);
poly f1 = u+v+w+x+y+z+(a);
poly f2 = u*v+v*w+w*x+x*y+y*z+(a)*u+(a)*z;
poly f3 = u*v*w+v*w*x+w*x*y+x*y*z+(a)*u*v+(a)*u*z+(a)*y*z;
poly f4 = u*v*w*x+v*w*x*y+w*x*y*z+(a)*u*v*w+(a)*u*v*z+(a)*u*y*z+(a)*x*y*z;
poly f5 = u*v*w*x*y+v*w*x*y*z+(a)*u*v*w*x+(a)*u*v*w*z+(a)*u*v*y*z+(a)*u*x*y*z
          +(a)*w*x*y*z;
poly f6 = u*v*w*x*y*z+(a)*u*v*w*x*y+(a)*u*v*w*x*z+(a)*u*v*w*y*z+(a)*u*v*x*y*z
          +(a)*u*w*x*y*z+(a)*v*w*x*y*z;
poly f7 = (a)*u*v*w*x*y*z-1;
ideal I6 = f1, f2, f3, f4, f5, f6, f7;
```

8.
```
ring R = (0,a), (w,x,y,z), dp;
minpoly = (a^8-16*a^7+19*a^6-a^5-5*a^4+13*a^3-9*a^2+13*a+17);
poly f1 = (-a^2-1)*x^2*y+2*w*x*z-2*w+(a^2+1)*y;
poly f2 = (a^3-a-3)*w^3*y+4*w*x^2*y+4*w^2*x*z+2*x^3*z+(a)*w^2-10*x^2+4*w*y-10*x*z
          +(2*a^2+a);
poly f3 = (a^2+a+11)*x*y*z+w*z^2-w-2*y;
poly f4 = -w*y^3+4*x*y^2*z+4*w*y*z^2+2*x*z^3+(2*a^3+a^2)*w*y+4*y^2-10*x*z-10*z^2
          +(3*a^2+5);
ideal I7 = f1, f2, f3, f4;
```

9.
```
ring R = (0,a), (t,u,v,w,x,y,z), dp;
minpoly = (a^7+10*a^5+5*a^3+10*a+1);
poly f1 = v*x+w*y-x*z-w-y;
poly f2 = v*w-u*x+x*y-w*z+v+x+z;
poly f3 = t*w-w^2+x^2-t;
poly f4 = (-a)*v^2-u*y+y^2-v*z-z^2+u;
poly f5 = t*v+v*w+(-a^2-a-5)*x*y-t*z+w*z+v+x+z+(a+1);
poly f6 = t*u+u*w+(-a-11)*v*x-t*y+w*y-x*z-t-u+w+y;
poly f7 = w^2*y^3-w*x*y^3+x^2*y^3+w^2*y^2*z-w*x*y^2*z+x^2*y^2*z+w^2*y*z^2
```

```
          -w*x*y*z^2+x^2*y*z^2+w^2*z^3-w*x*z^3+x^2*z^3;
poly f8 = t^2*u^3+t^2*u^2*v+t^2*u*v^2+t^2*v^3-t*u^3*x-t*u^2*v*x-t*u*v^2*x
          -t*v^3*x+u^3*x^2+u^2*v*x^2+u*v^2*x^2+v^3*x^2;
ideal I8 = f1, f2, f3, f4, f5, f6, f7, f8;
```

A.2 Benchmark Problems for nfmodSyz

The ideals I1, I2, I3a, I4, I5 and I7 from Section A.1 and the submodules given below are the benchmark problems used to demonstrate the efficiency of Algorithms 3.9 and 3.10. The generating set of each submodule is constructed from randomly chosen monomials and coefficients. Note that for all examples we use the same module ordering which is given below.

1.
```
ring R8 = (0,a), (x,y,z), (c,dp);
minpoly = (a^3+a+1);
vector f1 = [x^2*z+x+(-a)*y, z^2+(a+2)*x];
vector f2 = [y^2+(a)*z+(a), (a+3)*z^3+(-a)*x^2];
vector f3 = [-x*z+(a^2+3)*y*z, x*y+(a^2)*z];
vector f4 = [y*z+(a^2+3)*x*y*z, a*x*y+z+y];
vector f5 = [z^2+(a+3)*x*y, x*z+a*z+x];
module I8 = f1, f2, f3, f4, f5;
```

2.
```
ring R9 = (0,a), (x,y,z), (c,dp);
minpoly = (a^4+2*a^2+3);
vector f1 = [y*z + (a+7)*x + z, (-a)*x-y-z];
vector f2 = [(a)*x*z^2+y*z^2+(a^2+1), x+y+(a)*z];
vector f3 =  [z^2 + a*x*y+a, y^2+y*z+z^2+(a)];
vector f4 = [(a)*x*z^2+y*z^2+(a^2+1), y^2*z^2+(a)*y+z] ;
vector f5 = [x*z+y+(a+1), y*z+(a)*y+x];
module I9 = f1, f2, f3, f4, f5;
```

3.
```
ring R10 = (0,a), (x,y,z), (c,dp);
minpoly = (a^3+7*a-5);
vector f1 = [(a+8)*x^2*y^2+5*x*y^3, (-a^2+3)*x^3*z+x^2*y*z];
vector f2 = [x^5+2*y^3*z^2, 13*a*y^2*z^3+5*y*z^4];
vector f3 = [8*x^3+(a^2+12)*y^3, x*z^2+(3*a)];
vector f4 = [(-a+7)*x^2*y^4+y^3*z^3, 18*y^3*z^2];
module I10 = f1, f2, f3, f4;
```

4.
```
 ring R11 = (0,a), (x,y,z,w), (c,dp);
minpoly = (a^7+2*a^6+7*a^4+3*a+17);
vector f1 = [(a^3)*w*y^2+(a^3+3*a^2)*x^2*z+(a^2)*w*x+(a)*y*z];
vector f2 = [(a^2)*x^2+(a^2)*w*y+(11*a^2)*x*z+(a)*z^2];
vector f3 = [(-a^4+5*a^2)*w*x+(a)*y*z+(7*a)*w];
vector f4 = [(a^6+2*a^3)*w*x*y+(2*a)*w+(a^2)*y];
vector f5 = [(a^4)*w*x^2*y*z+(a^3)*w^2];
vector f6 = [(a^3)*w^3+(a^3)*x^2+(a^6)*y^2];
module I11 = f1, f2, f3, f4, f5, f6;
```

5.
```
ring R12 = (0,a), (x,y,z,w), (c,dp);
minpoly = (a^6+3*a^5+a^3+7*a^2+11);
vector f1 = [(a)*w*y^2+(a^4+3*a^3)*x^2*z, (a^2)*w*x+(a)*y*z];
vector f2 = [(a^2)*x^2+(11*a^3)*x*z, (a)*w*y+(a^3)*z^2];
vector f3 = [(-a^4+5*a^2)*w*x+(a)*y*z, (7*a)*w+(2*a)];
vector f4 = [(a^5+2*a^2)*w*x*y+(a)*y, (-a^3)*w*x+(2*a)*w];
vector f5 = [(a^4)*w*x^2*y*z+(a^3)*w^2, (a^3)*w^3
            +(a^3)*x^2+(a^4)*y^2];
module I12 = f1, f2, f3, f4, f5;
```

6.
```
ring R13 = 0, (x,y,z), (c,dp);
vector f1 = [2*x*y*z^2+7*y^3, 2*x*y^4*z^2-x^2*y^3*z];
vector f2 = [2*x^2*y^4*z+x^2*y*z^2, -x*y^2*z^2+3*x^2*y*z-12*x+12*y ];
vector f3 = [-x*y^3+y^4+2*y^2*z, 2*y^5*z+x^2*y^2*z-x*y^3*z];
vector f4 = [3*x*y^4*z^3+x^2*y^2*z-x*y^3*z, 4*y^3*z^2+3*x*y*z^3
            +4*z^2-x+y];
module I13 = f1, f2, f3, f4, f5;
```

A.3 Benchmark Problems for `ffmodStd`

The ideals given below are the benchmark problems used to demonstrate the efficiency of Algorithms 5.14, 5.18, and 5.19. The ideal I2 is taken from Section A.1 where the coefficients are replaced by random transcendental elements whereas the ideal I8 is taken from Section A.2.3 in the manual of `Singular` 4.0.2 where we changed the characteristic to zero and the ordering to the lexicographic ordering (`lp` in SINGULAR). The generators of the remaining ideals are constructed from randomly chosen monomials and coefficients.

1.
```
ring R1 = (0,a), (x,y,z), dp;
poly f1 = (a^10-5*a^9-8*a^8+74*a^7-27*a^6-381*a^5+402*a^4 +760*a^3-1120*a^2-400*a+800)*x*y^4*z^2
          +(a^7-a^6-10*a^5+6*a^4+33*a^3-a^2-40*a-20)*x^3*y^2*z +(-a^7+4*a^6+4*a^5-30*a^4+9*a^3
          +58*a^2-20*a-40)*x^2*y^3*z+(2*a^7-2*a^6-24*a^5+20*a^4+90*a^3-50*a^2-100*a)*x*y*z^2
          +(7*a^3-42*a^2+84*a-56)*y^3+(7*a+1);
poly f2 = (a^9-8*a^8+16*a^7+30*a^6-145*a^5+98*a^4+216*a^3-272*a^2-80*a+160)*x^2*y^4*z+(a^7-13*a^5
          -2*a^4+55*a^3+20*a^2-75*a-50)*x^2*y*z^2+(-a^7+3*a^6+10*a^5-34*a^4-25*a^3+115*a^2
          -100)*x*y^2*z^2+(2*a^5-16*a^3-4*a^2+30*a+20)*x^2*y*z+(-12*a-12)*x+(a^2+10*a-24)*y;
poly f3 = (2*a^7-20*a^6+70*a^5-60*a^4-240*a^3+736*a^2-800*a+320)*y^5*z+(a^7+3*a^6-18*a^5-26*a^4
          +89*a^3+75*a^2-120*a-100)*x^2*y^2*z+(-a^7+5*a^6-a^5-29*a^4+38*a^3+20*a^2-40*a)*x*y^3*z
          +(-a^4+5*a^3-6*a^2-4*a+8)*x*y^3+(a^4-8*a^3+24*a^2-32*a+16)*y^4+(2*a^5-8*a^4
          -2*a^3+40*a^2-40*a)*y^2*z;
poly f4 = (3*a^11-21*a^10+3*a^9+291*a^8-543*a^7-1167*a^6+3945*a^5+105*a^4-9600*a^3+6600*a^2
          +6000*a-6000)*x*y^4*z^3+(a^7-2*a^6-8*a^5+14*a^4+19*a^3-20*a^2-20*a)*x^2*y^2*z+(-a^6
          +5*a^5-a^4-29*a^3+38*a^2+20*a-40)*x*y^3*z+(-a^8+10*a^7-26*a^6-44*a^5+303*a^4-310*a^3
          -580*a^2+1400*a-800)*y^3*z^2+(3*a^8-3*a^7-51*a^6+45*a^5+315*a^4-225*a^3-825*a^2
          +375*a+750)*x*y*z^3+(4*a^4-40*a^2+100)*z^2+(-a-1)*x+(a^2-2*a)*y;
ideal I1 = f1,f2,f3,f4;
```

2.
```
ring R2 = (0,a), (x,y,z), dp;
poly f1 = (a^10-a^9-10*a^8+7*a^7+32*a^6-11*a^5-34*a^4+13*a^3-a^2-40*a-20)*x^3*y^2*z+(-a^7+4*a^6
          +4*a^5-30*a^4+9*a^3+58*a^2-20*a-40)*x^2*y^3*z+(2*a^7-2*a^6-24*a^5
          +20*a^4+90*a^3-50*a^2-100*a)*x*y*z^2+(7*a^3-42*a^2+84*a-56)*y^3;
```

```
poly f2 = (a^8-6*a^7+4*a^6+38*a^5-69*a^4-40*a^3+136*a^2-80)*x^2*y^4*z+(a^8-7*a^7-13*a^6+89*a^5
          +69*a^4-365*a^3-215*a^2+475*a+350)*x^2*y*z^2+(-a^7+3*a^6+10*a^5-34*a^4-25*a^3+115*a^2
          -100)*x*y^2*z^2+(2*a^5-16*a^3-4*a^2+30*a+20)*x^2*y*z+(-12*a-12)*x+(a^2-a-2)*y;
poly f3 = (a^8-12*a^7+55*a^6-100*a^5-60*a^4+608*a^3-1136*a^2+960*a-320)*y^5*z+(a^7+3*a^6-18*a^5
          -26*a^4+89*a^3+75*a^2-120*a-100)*x^2*y^2*z+(-a^7+5*a^6-a^5-29*a^4+38*a^3+20*a^2
          -40*a)*x*y^3*z+(-a^4+5*a^3-6*a^2-4*a+8)*x*y^3+(a^4-8*a^3+24*a^2-32*a+16)*y^4
          +(2*a^5-8*a^4-2*a^3+40*a^2-40*a)*y^2*z;
poly f4 = (a^12-7*a^11+a^10+97*a^9-181*a^8-389*a^7+1315*a^6
          +35*a^5-3200*a^4+2200*a^3+2000*a^2
          -2000*a)*x*y^4*z^3+(a^7-2*a^6-8*a^5+14*a^4+19*a^3-20*a^2-20*a)*x^2*y^2*z
          +(-a^6+5*a^5-a^4-29*a^3+38*a^2+20*a-40)*x*y^3*z+(-a^8+10*a^7-26*a^6-44*a^5
          +303*a^4-310*a^3-580*a^2+1400*a-800)*y^3*z^2+(3*a^8-3*a^7-51*a^6+45*a^5+315*a^4
          -225*a^3-825*a^2+375*a+750)*x*y*z^3+(4*a^4-40*a^2+100)*z^2+(-a-1)*x;
ideal I2 = f1,f2,f3,f4;
```

3.
```
ring R3 = (0,a), (x,y,z,w), dp;
poly f1 = (a^4+2*a^3-4*a^2-10*a-5)*x^2*z+(a^2-4*a+4)/(a^2+3*a)*y^2*w+(-a^2-2*a-1)*x^2
          +(-7*a^2+28*a-28)/(a^3+3*a^2)*y^2+(-6*a^3-6*a^2+30*a+30)*x*z+(a^3-2*a^2
          -5*a+10)/(a^3+3*a^2)*y*z+(a+1)/(a+3)*x*w+(2*a-4)/(a+3)*y*w+(6*a^3+24*a^2
          +11*a-7)/(a^2+3*a)*x+(-14*a^2+27*a+2)/(a^3+3*a^2)*y+(9*a^4+27*a^3-44*a^2
          -135*a-5)/(a^2+3*a)*z+(a-3)/(a+3)*w+(-9*a^2-34*a+20)/(a^2+3*a);
poly f2 = (a^2+2*a+1)*x^2+(11*a^3+11*a^2-55*a-55)*x*z+(a^5-10*a^3+25*a)*z^2+(a-2)*y*w
          +(-17*a-17)*x+(-7*a+14)/(a)*y+(-2*a^3-33*a^2+10*a+165)*z+(a)*w+(a+35);
poly f3 = (a^3-2*a^2-5*a+10)*y*z+(-a^4-a^3)*x*w+(7*a^3+7*a^2)*x+(-a+2)*y
          +(a^3-5*a)*z+(3*a^3+7*a)*w+(-21*a^2-a-49);
poly f4 = (a^3-a^2-2*a)*x*y*w+(-7*a^2+7*a+14)*x*y+(a^3+a^2)*x*w+(-3*a^2+6*a)*y*w
          +(-7*a^2-7*a)*x+(21*a^2-41*a-2)/(a)*y+(-3*a^2+2)*w+(21*a^2+a-14)/(a);
poly f5 = (a^6-8*a^4-2*a^3+15*a^2+10*a)*x^2*y*z*w+(-7*a^5+56*a^3+14*a^2-105*a
          -70)*x^2*y*z+(-a^4+3*a^2+2*a)*x^2*y*w+(a^6+2*a^5-4*a^4-10*a^3
          -5*a^2)*x^2*z*w+(-6*a^5+6*a^4+42*a^3-30*a^2-60*a)*x*y*z*w
          +(7*a^3-21*a-14)*x^2*y+(-7*a^5-14*a^4+28*a^3+70*a^2+35*a)*x^2*z
          +(42*a^4-42*a^3-294*a^2+210*a+420)*x*y*z+(-a^4-2*a^3-a^2)*x^2*w
          +(6*a^3-6*a^2-12*a)*x*y*w+(-6*a^5-6*a^4+30*a^3+30*a^2)*x*z*w
          +(9*a^4-18*a^3-45*a^2+90*a)*y*z*w+(7*a^3+14*a^2+7*a)*x^2+(-42*a^2+42*a
          +84)*x*y+(42*a^4+42*a^3-210*a^2-210*a)*x*z+(-63*a^3+126*a^2+315*a
          -630)*y*z+(6*a^3+6*a^2)*x*w+(-9*a^2+18*a)*y*w+(9*a^4-45*a^2)*z*w+(a)*w^2
          +(-42*a^2-42*a)*x+(63*a-126)*y+(-63*a^3+315*a)*z+(-9*a^2-14)*w
          +(63*a^2+49)/(a);
poly f6 = (a^3)*w^3+(a^2+2*a+1)/(a)*x^2+(a^2-4*a+4)/(a^3)*y^2+(-21*a^2)*w^2
          +(-6*a-6)/(a)*x+(2*a-4)/(a^2)*y+(147*a)*w+(-343*a+10)/(a);
ideal I3 = f1,f2,f3,f4,f5,f6;
```

4.
```
ring R4 = (0,a), (v,w,x,y,z), dp;
poly f1 = (4*a^3+24*a^2+45*a+27)*v^2+(a^4+2*a^3+3*a^2+4*a+2)*x*z+(-a^2-2*a-1)*x
          +(5*a^3+5*a^2+10*a+10)*z+(-5*a-5);
poly f2 = (a^3)*w^3+(-a^7-6*a^5-12*a^3-8*a)*z^3+(-18*a^2)*w^2+(3*a^5+12*a^3
          +12*a)*z^2+(108*a)*w+(-3*a^3-6*a)*z+(a-216);
poly f3 = (8*a^4+36*a^3+54*a^2+27*a)*v^3+(-a^3-3*a^2-3*a-1)*x^3+(a^4+8*a^3
          +9*a^2+16*a+14)*x*y*z+(-15*a^2-30*a-15)*x^2+(-a^2-8*a-7)*x*y
          +(5*a^3+35*a^2+10*a+70)*y*z+(-75*a-75)*x+(-5*a-35)*y-125;
poly f4 = (a^2+8*a+7)*x*y+(a^5+a^4+2*a^3+2*a^2)*w*z+(-a^3-a^2)*w+(5*a+35)*y
          +(-6*a^4-6*a^3-12*a^2-12*a)*z+(6*a^2+6*a);
poly f5 = (a^3+21*a^2+147*a+343)*y^3+(-a^3-2*a^2-a)*x^2+(-10*a^2-10*a)*x+(-25*a);
poly f6 = (a^3+a^2)*w*x+(2*a^2+17*a+21)*v*y+(2*a^3+3*a^2+4*a+6)*v*z
```

```
          +(-2*a-3)*v+(5*a^2)*w+(-6*a^2-6*a)*x+(-30*a);
poly f7 = (4*a^3+28*a^2+57*a+36)*v^2+(a^2+7*a)*w*y+(a^3+12*a^2+21*a-98)*y^2
          +(2*a^4+3*a^3+5*a^2+6*a+2)*x*z+(-14*a-21)*v
          +(a^2-5*a)*w+(-2*a^2-3*a-1)*x+(-6*a-42)*y+(11*a^3+12*a^2+22*a+24)*z
          +(-17*a+18);
poly f8 = (4*a^4+44*a^3+133*a^2+156*a+63)*v^2*x*y+(a^6+8*a^5+4*a^4-26*a^3-37*a^2
          -14*a)*x^3*y+(4*a^5+68*a^4+373*a^3+714*a^2+441*a)*v^2*y^2
          +(a^7+14*a^6+46*a^5-44*a^4-175*a^3-98*a^2)*x^2*y^2+(4*a^7+16*a^6
          +29*a^5+41*a^4+42*a^3+18*a^2)*v^2*w*z+(a^9+a^8-a^7-3*a^6
          -8*a^5-10*a^4-4*a^3)*w*x^2*z+(-4*a^5-16*a^4-21*a^3-9*a^2)*v^2*w
          +(-a^7-a^6+3*a^5+5*a^4+2*a^3)*w*x^2+(20*a^3+200*a^2+465*a
          +315)*v^2*y+(15*a^5+105*a^4-45*a^3-345*a^2-210*a)*x^2*y+(10*a^6
          +130*a^5+330*a^4-770*a^3-980*a^2)*x*y^2+(-24*a^6-96*a^5-174*a^4-246*a^3
          -252*a^2-108*a)*v^2*z+(10*a^8-10*a^6-20*a^5-60*a^4-40*a^3)*w*x*z
          +(-6*a^8-6*a^7+6*a^6+18*a^5+48*a^4+60*a^3+24*a^2)*x^2*z
          +(24*a^4+96*a^3+126*a^2+54*a)*v^2+(-10*a^6+30*a^4+20*a^3)*w*x
          +(6*a^6+6*a^5-18*a^4-30*a^3-12*a^2)*x^2+(75*a^4+450*a^3
          -675*a^2-1050*a)*x*y+(25*a^5+300*a^4+525*a^3-2450*a^2)*y^2
          +(25*a^7-25*a^6-50*a^4-100*a^3)*w*z+(-60*a^7+60*a^5
          +120*a^4+360*a^3+240*a^2)*x*z+(-25*a^5+25*a^4+50*a^3)*w
          +(60*a^5-180*a^3-120*a^2)*x+(125*a^3+625*a^2-1750*a)*y+(-150*a^6+150*a^5
          +300*a^3+600*a^2)*z+(150*a^4-150*a^3-300*a^2);
ideal I4 = f1,f2,f3,f4,f5,f6,f7,f8;
```

5.
```
ring R5 = (0,a), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10), dp;
poly f1 = (a^3+2*a)*x2*x5+(3*a^3-20*a^2-69*a+54)*x1*x6+(a^4+7*a^3)*x4*x9
          +(-a)*x2+(-6*a^2-12)*x5+6;
poly f2 = (a^5+4*a^3+4*a)*x5^2+(a^4+7*a^3-a^2-7*a)*x3*x8+(a^5
          +8*a^4+7*a^3)*x4*x9+(-2*a^3-4*a)*x5+(5*a^3+30*a^2-35*a)*x8+(a);
poly f3 = (a^2+14*a+49)*x4^2+(a^3+a^2+a+1)*x3+(a^2-a)*x10+(5*a^2+5);
poly f4 = (2*a^5+33*a^4+168*a^3+245*a^2)*x8^2+(-2*a^4+17*a^3+9*a^2)*x6*x9
          +(5*a^2-5*a)*x7*x10;
poly f5 = (a^4+2*a^3-35*a^2)*x2*x4*x7+(a^3-3*a^2+3*a-1)*x10^3+(-6*a^3-12*a^2
          +210*a)*x4*x7+(a^3-3*a^2+2*a-6)*x5+(-a+3);
poly f6 = (a^2-8*a-9)*x3*x6+(a^4)*x9^2+(5*a-45)*x6;
poly f7 = (a^3-9*a^2)*x6*x9+(a^2+7*a)*x8+(a^3-a^2)*x10;
poly f8 = (a^3)*x7^2+(-8*a^2-56*a)*x8+(a^3)*x9+(a-1)*x10;
poly f9 = (a^2-18*a+81)*x6^2+(-14*a^4)*x7*x9;
poly f10 = (3*a-2)*x1+(a^2)*x2+(-6*a);
poly f11 = (a+7)*x4+(a^3+7*a^2+2*a+14)*x5+(a)*x7+(-a-7);
poly f12 = (a^2-10*a+9)*x6+(a^2)*x9+(a^2-a)*x10;
ideal I5 = f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12;
```

6.
```
ring R6 = (0,a,b), (x,y,z,w), dp;
poly f1 = (a^2)*x^2+(11*a^2*b)*x*z+(a^3)*z^2+(b)*y*w+(-22*a*b^2+4*a)*x+(a*b^2)*y
          +(-4*a^2*b+22*a*b)*z+(a)*w+(a^2*b+4*a*b^2-44*b^2+4);
poly f2 = (a^2*b)*x*z+(-a^2)*x*w+(-a^3*b-2*a*b^2)*x+(2*a*b)*z+(-2*a+7)*w
          +(-2*a^2*b+7*a*b-4*b^2);
poly f3 = (a*b^2)*y^2*z+(-2*b^3)*y^2+(2*a^2*b)*y*z+(-a^2)*z*w+w^2+(-4*a*b^2)*y
          +(-a^3*b+a^3)*z+(4*a*b)*w+(3*a^2*b^2-2*a^2*b);
poly f4 = w^2+(b^2)*y+(2*a*b+1)*w+(a^2*b^2+2*a*b);
poly f5 = (a^6)*x^2*z^4+(-8*a^5*b)*x^2*z^3+(4*a^5)*x*z^4+(2*a*b)*y*w^4
          +(24*a^4*b^2)*x^2*z^2+(-32*a^4*b)*x*z^3+(4*a^4)*z^4+(-a^2*b)*x*y*z*w
          +(8*a^2*b^2)*y*w^3+(2*a^2)*w^4+(-32*a^3*b^3)*x^2*z+(-a^3*b^2)*x*y*z
```

```
            +(96*a^3*b^2)*x*z^2+(-32*a^3*b)*z^3+(2*a*b^2)*x*y*w+(-a^3)*x*z*w+(-2*a*b)*y*z*w
            +(12*a^3*b^3)*y*w^2+(8*a^3*b)*w^3+(16*a^2*b^4)*x^2+(2*a^2*b^3)*x*y
            +(-a^4*b-128*a^2*b^3)*x*z+(-2*a^2*b^2)*y*z+(96*a^2*b^2)*z^2+(2*a^2*b)*x*w
            +(8*a^4*b^4+4*b^2)*y*w+(-2*a^2)*z*w+(12*a^4*b^2)*w^2+(2*a^3*b^2+64*a*b^4)*x
            +(2*a^5*b^5+4*a*b^3)*y+(-2*a^3*b-128*a*b^3)*z+(8*a^5*b^3+4*a*b)*w
            +(2*a^6*b^4+4*a^2*b^2+64*b^4);
   ideal I6 = f1,f2,f3,f4,f5;


7. ring R7 = (0,a,b,c), (x,y,z), dp;
   poly f1 = (a^5*b)*x^2*y^3*z+(-2*a^6)*x^2*y^3+(3*a^4*b*c)*x^2*y^2*z
            +(4*a^4*b*c)*x*y^3*z+(-6*a^5*c)*x^2*y^2+(-8*a^5*c)*x*y^3
            +(3*a^3*b*c^2)*x^2*y*z+(12*a^3*b*c^2)*x*y^2*z+(4*a^3*b*c^2)*y^3*z
            +(2*a^3*b^2+a^2*b^4)*x*y*z^2+(-6*a^4*c^2)*x^2*y+(-24*a^4*c^2)*x*y^2
            +(-8*a^4*c^2+7*a^3)*y^3+(a^2*b*c^3)*x^2*z+(-8*a^4*b-4*a^3*b^3
            +12*a^2*b*c^3)*x*y*z+(12*a^2*b*c^3)*y^2*z+(2*a^2*b^2*c+a*b^4*c)*x*z^2
            +(4*a^2*b^2*c+2*a*b^4*c)*y*z^2+(-2*a^3*c^3)*x^2+(8*a^5+4*a^4*b^2
            -24*a^3*c^3)*x*y+(-24*a^3*c^3+21*a^2*c)*y^2+(-8*a^3*b*c-4*a^2*b^3*c
            +4*a*b*c^4)*x*z+(-16*a^3*b*c-8*a^2*b^3*c+12*a*b*c^4)*y*z
            +(4*a*b^2*c^2+2*b^4*c^2)*z^2+(8*a^4*c+4*a^3*b^2*c-8*a^2*c^4)*x+(16*a^4*c
            +8*a^3*b^2*c-24*a^2*c^4+21*a*c^2)*y+(-16*a^2*b*c^2-8*a*b^3*c^2
            +4*b*c^5)*z+(16*a^3*c^2+8*a^2*b^2*c^2-8*a*c^5+7*c^3);
   poly f2 = (a^7*b-a^6*b*c)*x^2*y^4*z+(-2*a^8+2*a^7*c)*x^2*y^4+(4*a^6*b*c
            -4*a^5*b*c^2)*x^2*y^3*z+(4*a^6*b*c-4*a^5*b*c^2)*x*y^4*z+(-8*a^7*c
            +8*a^6*c^2)*x^2*y^3+(-8*a^7*c+8*a^6*c^2)*x*y^4+(6*a^5*b*c^2
            -6*a^4*b*c^3)*x^2*y^2*z+(16*a^5*b*c^2-16*a^4*b*c^3)*x*y^3*z+(4*a^5*b*c^2
            -4*a^4*b*c^3)*y^4*z+(a^5*b^2+a^3*b^3*c-7*a^3*b^2)*x^2*y*z^2
            +(-a^3*b^2)*x*y^2*z^2+(-12*a^6*c^2+12*a^5*c^3)*x^2*y^2
            +(-32*a^6*c^2+32*a^5*c^3)*x*y^3+(-8*a^6*c^2+8*a^5*c^3)*y^4+(-4*a^6*b
            -4*a^4*b^2*c+4*a^4*b*c^3+28*a^4*b-4*a^3*b*c^4+2*a^3*b)*x^2*y*z
            +(24*a^4*b*c^3+4*a^4*b-24*a^3*b*c^4)*x*y^2*z+(16*a^4*b*c^3
            -16*a^3*b*c^4)*y^3*z+(a^4*b^2*c+a^2*b^3*c^2-7*a^2*b^2*c)*x^2*z^2
            +(4*a^4*b^2*c+4*a^2*b^3*c^2-30*a^2*b^2*c)*x*y*z^2+(-2*a^2*b^2*c)*y^2*z^2
            +(4*a^7+4*a^5*b*c-8*a^5*c^3-28*a^5+8*a^4*c^4-4*a^4)*x^2*y
            +(-48*a^5*c^3-4*a^5+48*a^4*c^4)*x*y^2+(-32*a^5*c^3
            +32*a^4*c^4)*y^3+(-4*a^5*b*c-4*a^3*b^2*c^2+a^3*b*c^4+28*a^3*b*c
            -a^2*b*c^5+2*a^2*b*c)*x^2*z+(-16*a^5*b*c-16*a^3*b^2*c^2+16*a^3*b*c^4
            +120*a^3*b*c-16*a^2*b*c^5+8*a^2*b*c)*x*y*z+(24*a^3*b*c^4
            +8*a^3*b*c-24*a^2*b*c^5)*y^2*z+(4*a^3*b^2*c^2+4*a*b^3*c^3
            -29*a*b^2*c^2)*x*z^2+(4*a^3*b^2*c^2+4*a*b^3*c^3
            -32*a*b^2*c^2)*y*z^2+(4*a^6*c+4*a^4*b*c^2-2*a^4*c^4-28*a^4*c
            +2*a^3*c^5-4*a^3*c)*x^2+(16*a^6*c+16*a^4*b*c^2-32*a^4*c^4-120*a^4*c
            +32*a^3*c^5-16*a^3*c)*x*y+(-48*a^4*c^4-8*a^4*c+48*a^3*c^5)*y^2
            +(-16*a^4*b*c^2-16*a^2*b^2*c^3+4*a^2*b*c^5+116*a^2*b*c^2-4*a*b*c^6
            +8*a*b*c^2)*x*z+(-16*a^4*b*c^2-16*a^2*b^2*c^3+16*a^2*b*c^5+128*a^2*b*c^2
            -16*a*b*c^6+8*a*b*c^2)*y*z+(4*a^2*b^2*c^3+4*b^3*c^4-30*b^2*c^3)*z^2
            +(16*a^5*c^2+16*a^3*b*c^3-8*a^3*c^5-116*a^3*c^2+8*a^2*c^6
            -16*a^2*c^2-12*a)*x+(16*a^5*c^2+16*a^3*b*c^3-32*a^3*c^5-128*a^3*c^2
            +32*a^2*c^6-16*a^2*c^2+a*b)*y+(-16*a^3*b*c^3-16*a*b^2*c^4+4*a*b*c^6
            +120*a*b*c^3-4*b*c^7+8*b*c^3)*z+(16*a^4*c^3+16*a^2*b*c^4
            -8*a^2*c^6-120*a^2*c^3+8*a*c^7-16*a*c^3+b*c-22*c);
   poly f3 = (a^6*b^2+a^5*b*c-2*a^5*b)*y^5*z+(-2*a^7*b-2*a^6*c+4*a^6)*y^5
            +(a^5*b+5*a^4*b)*x^2*y^2*z+(-a^5*b-a^4*b^2)*x*y^3*z+(5*a^5*b^2*c
            +5*a^4*b*c^2-10*a^4*b*c)*y^4*z+(-2*a^6-10*a^5)*x^2*y^2+(2*a^6
            +2*a^5*b-a^4*c)*x*y^3+(-10*a^6*b*c-10*a^5*c^2+20*a^5*c+a^4)*y^4
            +(2*a^4*b*c+10*a^3*b*c)*x^2*y*z+(a^4*b*c-3*a^3*b^2*c
```

```
            +20*a^3*b*c)*x*y^2*z+(10*a^4*b^2*c^2-2*a^4*b*c-2*a^3*b^2*c+10*a^3*b*c^3
            -20*a^3*b*c^2)*y^3*z+(-4*a^5*c-20*a^4*c)*x^2*y+(-2*a^5*c+6*a^4*b*c
            -40*a^4*c-3*a^3*c^2)*x*y^2+(-20*a^5*b*c^2+4*a^5*c+4*a^4*b*c
            -20*a^4*c^3+40*a^4*c^2-2*a^3*c^2+4*a^3*c)*y^3+(a^3*b*c^2
            +5*a^2*b*c^2)*x^2*z+(5*a^3*b*c^2-3*a^2*b^2*c^2+40*a^2*b*c^2)*x*y*z
            +(10*a^3*b^2*c^3+2*a^3*b^2*c-2*a^3*b*c^2-6*a^2*b^2*c^2+10*a^2*b*c^4
            -20*a^2*b*c^3+20*a^2*b*c^2)*y^2*z +(-2*a^4*c^2-10*a^3*c^2)*x^2
            +(-10*a^4*c^2+6*a^3*b*c^2-80*a^3*c^2-3*a^2*c^3)*x*y+(-20*a^4*b*c^3
            -4*a^4*b*c+4*a^4*c^2+12*a^3*b*c^2-20*a^3*c^4+40*a^3*c^3-40*a^3*c^2
            -6*a^2*c^3+6*a^2*c^2)*y^2+(3*a^2*b*c^3-a*b^2*c^3+20*a*b*c^3)*x*z
            +(5*a^2*b^2*c^4+4*a^2*b^2*c^2+2*a^2*b*c^3-6*a*b^2*c^3
            +5*a*b*c^5-10*a*b*c^4+40*a*b*c^3)*y*z+(-6*a^3*c^3+2*a^2*b*c^3-40*a^2*c^3
            -a*c^4)*x+(-10*a^3*b*c^4-8*a^3*b*c^2-4*a^3*c^3+12*a^2*b*c^3-10*a^2*c^5
            +20*a^2*c^4-80*a^2*c^3-6*a*c^4+4*a*c^3)*y+(a*b^2*c^5+2*a*b^2*c^3+2*a*b*c^4
            -2*b^2*c^4+b*c^6-2*b*c^5+20*b*c^4)*z+(-2*a^2*b*c^5-4*a^2*b*c^3-4*a^2*c^4
            +4*a*b*c^4-2*a*c^6+4*a*c^5-40*a*c^4-2*c^5+c^4);
   poly f4 = (a^5*b*c)*x^2*y^2*z+(-a^4*b)*x*y^3*z+(-a^4*b^3+4*a^3*b^2)*y^3*z^2
            +(3*a^2*b^3)*x*y*z^3+(-2*a^6*c)*x^2*y^2+(2*a^5)*x*y^3+(2*a^4*b*c^2)*x^2*y*z
            +(4*a^4*b*c^2-3*a^3*b*c)*x*y^2*z +(4*a^5*b^2-16*a^4*b-2*a^3*b*c)*y^3*z
            +(-18*a^3*b^2)*x*y*z^2+(-3*a^3*b^3*c+12*a^2*b^2*c)*y^2*z^2+(3*a*b^3*c)*x*z^3
            +(6*a*b^3*c)*y*z^3+(-4*a^5*c^2)*x^2*y+(-8*a^5*c^2+6*a^4*c)*x*y^2+(-4*a^6*b
            +16*a^5+4*a^4*c)*y^3+(a^3*b*c^3)*x^2*z +(36*a^4*b+8*a^3*b*c^3-3*a^2*b*c^2)*x*y*z
            +(12*a^4*b^2*c+4*a^3*b*c^3-48*a^3*b*c-6*a^2*b*c^2)*y^2*z+(-18*a^2*b^2*c)*x*z^2
            +(-3*a^2*b^3*c^2-36*a^2*b^2*c+12*a*b^2*c^2)*y*z^2+(6*b^3*c^2)*z^3
            +(-2*a^4*c^3)*x^2+(-24*a^5-16*a^4*c^3 +6*a^3*c^2)*x*y+(-12*a^5*b*c-8*a^4*c^3
            +48*a^4*c+12*a^3*c^2)*y^2+(36*a^3*b*c+4*a^2*b*c^4-a*b*c^3)*x*z+(12*a^3*b^2*c^2
            +72*a^3*b*c+8*a^2*b*c^4-48*a^2*b*c^2-6*a*b*c^3)*y*z+(-a*b^3*c^3-36*a*b^2*c^2
            +4*b^2*c^3+4*b^2*c)*z^2+(-24*a^4*c-8*a^3*c^4+2*a^2*c^3-a)*x+(-12*a^4*b*c^2-48*a^4*c
            -16*a^3*c^4+48*a^3*c^2+12*a^2*c^3)*y+(4*a^2*b^2*c^3+72*a^2*b*c^2+4*a*b*c^5
            -16*a*b*c^3-16*a*b*c-2*b*c^4)*z+(-4*a^3*b*c^3-48*a^3*c^2-8*a^2*c^5
            +16*a^2*c^3+16*a^2*c+4*a*c^4+b-2*c);
   ideal I7 = f1,f2,f3,f4;
```

8.
```
   ring R8 = (0,u1,u2,u3,u4), (x1,x2,x3,x4,x5,x6,x7), lp;
   poly f1 = -x4*u3+x5*u2;
   poly f2 =  x1*u3+2*x2*u1-2*x2*u2-2*x3*u3-u1*u4+u2*u4;
   poly f3 = -2*x1*x5+4*x4*x6+4*x5*x7+x1*u3-2*x4*u1-2*x4*u4-2*x6*u2-2*x7*u3+u1*u2+u2*u4;
   poly f4 = -x1*x5+x1*x7-x4*u1+x4*u2-x4*u4+x5*u3+x6*u1-x6*u2+x6*u4-x7*u3;
   poly f5 = -x1*x4+x1*u1-x5*u1+x5*u4;
   poly f6 = -2*x1*x3+x1*u3-2*x2*u4+u1*u4+u2*u4;
   poly f7 = x1^2*u3+x1*u1*u2-x1*u2^2-x1*u3^2-u1*u3*u4+u3*u4^2;
   ideal I8 = f1, f2, f3, f4, f5, f6, f7;
```

9.
```
   ring R9 = (0,a,b,c,d), (x,y,z,w,v), dp;
   poly f1 = (d)*x*z+(c)*y*z+(a^2*b^2+a*b^2*c)*w*v+(2*a+d)*z+(-7*a*b*c+a*b*d-7*b*c^2
            +b*c*d)*w+(-2*a^3*b-2*a^2*b*c)*v+(14*a^2*c-2*a^2*d+14*a*c^2-2*a*c*d);
   poly f2 = (b^2)*w^2+(b*d)*x+(c)*y+(-a)*z+(-4*a*b)*w+(4*a^2+2*a*b+d);
   poly f3 = z^2+(a*b^2*d)*w*v+(-7*b*c*d+b*d^2)*w +(-2*a^2*b*d)*v+(14*a*c*d-2*a*d^2);
   poly f4 = (c*d^2)*x^2*y+(d^3)*x^2+(4*a*c*d)*x*y+(a*b^3)*w^2+(a*b*d)*x*v+(4*a*d^2
            -7*c*d+d^2)*x+(4*a^2*c)*y +(-4*a^2*b^2)*w+(2*a^2*b)*v
            +(4*a^3*b+4*a^2*d-14*a*c+2*a*d);
   poly f5 = (c^2)*y^2+(b^2*c)*w^2+(a^2*b^2)*v^2+(2*c*d)*y+(-4*a*b*c)*w+(-14*a*b*c
            +2*a*b*d)*v+(4*a^2*c+49*c^2-14*c*d+2*d^2);
   poly f6 = (d^2)*x^2+(d)*x*z+(b*d)*x*w +(2*a*d)*x+(2*a)*z+(2*a*b)*w;
```

```
ideal I9 = f1, f2, f3, f4, f5, f6;
```

10.
```
ring R10 = (0,a,b,c,d,e,f,g), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10), dp;
poly f1 = x1*x3+(c+a)*x1*x7+b*x10;
poly f2 = x2*x5+c*x4*x9+g*x1*x6;
poly f3 = g*x3*x8+(a+g)*x4*x9+f*x5^2;
poly f4 = x4^2+d*x3+e*x10;
poly f5 = (e+f)*x8^2-(2b)*x6*x9+5*x7*x10;
poly f6 = x10^3+a*x2*x7*x4+d*x5;
poly f7 = x9^2+x3*x6;
poly f8 = x6*x9+bc*x10;
poly f9 = b*x7^2-8*x8+d*x9+x10;
poly f10 = x6^2-14*c*x7*x9;
poly f11 = x1+a*x2;
poly f12 = x4+d*x5;
ideal I10 = f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12;
```

11.
```
ring R11 = (0, a,b,u1,u2,u3,u4), (x,y,z,w,v), dp;
poly f1 = w2-az+bx+y;
poly f2 = x1*u3+2*x2*u1-2*x2*u2-2*x3*u3;
poly f3 = u1*x2-u4*yz+b*u3*wv;
poly f4 = x^2+11b*x*z+(a)*z^2+y*w;
poly f5 = (a)*xz+(a-2)*y2 +v2+(a+7)*z;
poly f6 = (a-5)*w-7v+wy+(a+1)*xz+(a+3)*v2;
ideal I11 = f1, f2, f3, f4, f5, f6;
```

12.
```
ring S1 = (0,t1,t2,t3,t4), (x1,x2,x3,x4,x5,x6), dp;
poly f1 = (96*t1+59)/(13*t1^2)*x3*x4^2*x6 +10/(23*t1^3+4*t1)*x3*x5^2*x6;
poly f2 = (1165*t1^2)/(252*t2^2)*x1*x2^3+(89*t2^2)/(82*t1^3)*x1*x2;
poly f3 = (10*t2^3)/(33*t1^3)*x1^3+(5*t2)/(41*t3)*x1;
poly f4 = (3977*t1^3+533*t1^2*t2+2173*t1*t2^2+806*t1*t2+3286*t2^2)/(3977*t1^2
          +6014*t1)*x1^3+(22*t1*t2^2)/(5*t1^2 +61*t2^2)*x2^3
          +(42*t1^3+65*t2^3)/(65*t1^2*t2 +47*t1)*x1^2;
poly f5 = (3741*t1^4+94)/43*x2*x3^2+(8*t1^2)/21*x3*x6^2+(54*t1^2)*x3^2;
poly f6 = (96*t1^2*t2^2+41*t1*t2^2*t4)/(95*t1^2*t3^2+32*t2^3*t4)*x1*x2*x3*x5
          +(31*t1*t2*t3^2+11*t2^2*t3*t4)/(39*t1^4+51*t1^3*t2)*x3^3*x5+(21*t1*t4
          +90*t3^2)/(37*t1*t3+62*t2^2)*x3*x4*x5^2+(63*t1^2*t2*t3+27*t2^4)/(16*t1^3*t3
          +67*t2^4)*x2*x3^2;
poly f7 = (726*t1^4+205*t1*t2^4)/(110*t2^3)*x1^2+(61*t2^2)/(36*t1^2)*x1*x3;
ideal J1 = f1, f2, f3, f4, f5, f6, f7;
```

13.
```
ring S2 = (0,t1,t2,t3,t4,t5,t6,t7,t8), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14), lp;
poly f1 = (10*t1^2*t2+t2^2)*x1^3+(4*t1^2+9*t2)/(2*t2^3)*x2^2;
poly f2 = (17*t1*t6+12*t4*t5)/(8*t3*t4+15*t7)*x2*x4*x9
          +(7*t1*t2+6*t2*t4)/(3*t2^2+7*t8^2)*x11^3;
poly f3 = (16*t2^2+17*t4^2)/(7*t1*t3+17*t1*t4)*x4*x12^2
          +(19*t2+15*t3*t4)/(13*t3)*x10*x13^2;
poly f4 = (19*t1^2+16*t1*t2*t3)/(7*t1*t2*t5+4*t2*t4*t5)*x2*x3*x4
          +(11*t3+6*t5)/(19*t1^2+7*t2*t5)*x4^2;
poly f5 = (3*t1*t3+10*t2*t4)/(15*t1^2*t6+10*t1*t3^2)*x1^2*x2
          +(6*t1*t2*t5+7*t1*t3*t4)/(10*t1^2*t5+3*t2^3)*x3^2;
poly f6 = (15*t1*t2+20*t2*t3^2)/(t1^3)*x1+(t2)/(19*t1*t2+6*t1*t3)*x3^2;
poly f7 = (19*t5)/(12*t1)*x1*x2+(7*t3*t5)/(11*t1)*x5*x11;
```

```
poly f8 = (13*t3)/(17*t4)*x1*x4*x7+1/(8*t1)*x3*x4*x9;
poly f9 = (8*t1*t4+20*t2*t8+12*t3*t7)/(11*t1^2+17*t2*t3+9*t2)*x2
          +(16*t1*t4+8*t2*t7+9*t3*t8)/(11*t1^2+t1+13*t2*t3)*x3;
poly f10 = (18*t1*t3+8*t1*t8)/(18*t5*t7+9*t5*t8)*x1^2*x3 +(2*t1^2
           +2*t1*t8)/(10*t4*t6+11)*x3;
poly f11 = 15/(2*t2)*x1*x2*x7;
poly f12 = (10*t1^3+13*t1*t2*t3)/(13*t3*t4+7*t4^2)*x2*x3*x12+(11*t1*t5
           +12*t3*t4)/(17*t1^2+6*t1*t4)*x7*x9;
poly f13 = (3*t2^3+5*t2^2)/(14*t1)*x4*x5+(8*t1)/5*x7;
poly f14 = (20*t3^2+13*t4^2)/(11*t1*t2)*x1*x2*x3
           +(2*t3+19*t4^2)/(17*t2^2+17*t3*t4)*x2*x6*x7;
poly f15 = (11*t1^2*t4+t1*t3^2+20*t1*t3*t5)/(13*t1^2*t3+8*t1*t3*t5+16*t5)*x3*x5*x10
           +(2*t1+11*t2+2*t5)/(20*t1*t2+8*t2*t3+12*t2)*x3*x6*x7;
poly f16 = (14*t2*t4+15*t3^2+11*t3*t4)/12*x1^3+(6*t1^2*t2+8*t1^2*t3
           +9*t1*t2^2)/(13*t1*t4^2+8*t1+t2^2)*x1*x2^2;
poly f17 = (9*t1)/17*x2*x7*x9+(19*t1)/(3*t2)*x4*x5*x9;
poly f18 = 1/(20*t1)*x2*x6+(8*t2^2)/(13*t1*t3)*x9;
poly f19 = (5*t1*t3)/(7*t2^2)*x4*x6*x7 +(16*t2)/(15*t1^2)*x7*x8^2;
poly f20 = (2*t1*t4+19*t2*t4)/(4*t1*t2)*x3*x7*x9+(3*t1*t2+4*t1*t4)/(4*t1^2+t3*t4)*x5^2;
ideal J2 = f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,
           f15,f16,f17,f18,f19,f20;
```

14.
```
ring S3 = (0,t1,t2,t3,t4,t5,t6,t7,t8), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14), lp;
poly f1 = (19*t2*t3)/(10*t1^2)*x1*x3+(5*t1)/(t2)*x5*x6*x7;
poly f2 = (3*t2)/(19*t3)*x1^2+(t4^2)/(t2*t3)*x1*x3^2;
poly f3 = (11*t1^2)*x1+(13*t1^3+17*t1)/(8*t2^2)*x2^2;
poly f4 = (2*t1*t6^2+t1*t7+20*t2^2)/(9*t1*t2*t6 +3*t1*t3*t6+3*t1*t7)*x2*x4^2
          +(11*t1*t4+18*t1+5*t3*t4)/(15*t1*t4+17*t2*t6+8*t4*t7)*x5^3;
poly f5 = (11*t2*t4^2)/(19*t1*t2*t3+11*t2*t5^2+8*t5^3)*x3^2*x6+(2*t1^2*t2
          +16*t1^2*t3+4*t1^2*t5)/(19*t1^3+19*t1*t3*t4+t2^2)*x7*x11*x12;
poly f6 = 1/18*x3*x5^2+8/9*x5;
poly f7 = (14*t1^2+17*t1*t5^2)/(5*t1^2*t3+t2*t5)*x1*x3*x7+(19*t1^2*t4
          +12*t1*t5^2)/(12*t2^2*t4 +13*t2*t3)*x4;
poly f8 = (19*t2*t3)/(10*t1^2)*x4^3+(8*t1)/(9*t2*t5)*x4*x10*x13;
poly f9 = (5*t1*t2*t4+8*t1*t2+8*t1*t4*t5)/(10*t3^3+15*t3*t4^2
          +20*t3*t5^2)*x3^2*x4 +(8*t1^2*t4+6*t4^3+t4^2*t5)/(19*t1*t3*t5
          +13*t1*t5+3*t2*t3*t5)*x4*x6^2;
poly f10 = 1/3*x1*x10^2+(16*t1^2)/(9*t2*t8)*x3*x6*x12;
poly f11 = (19*t2)/(3*t1*t3)*x1^2*x4 +(16*t2^2)/(5*t1*t6)*x2*x4;
poly f12 = (18*t1*t3*t5+8*t2^2*t3+6*t2*t3)/(3*t1*t2*t4+3*t1*t3^2
           +6*t1*t3*t4)*x2^2*x11+(6*t1*t2*t5+8*t2^2+9*t2*t3^2)/(16*t4*t5)*x7*x8^2;
poly f13 = (7*t1*t7+6*t2*t6)/(4*t4*t6+2*t5^2)*x1*x8^2
           +(9*t1*t2+19*t2*t3)/(13*t2*t6+18*t4^2)*x3*x4;
poly f14 = (11*t3+14*t4)/(14*t3+10*t6)*x2*x6^2
           +(6*t3*t6+17*t4)/(13*t1^2+20*t3)*x8*x9*x12;
poly f15 = (20*t1*t2+5*t2*t5^2+19*t2*t5*t6)/(t1^2*t3+19*t1*t2*t6
           +4*t1*t3^2)*x1*x7+(4*t1*t4 +19*t5*t6
           +19*t5)/(15*t1*t5+8*t3^2+20*t4*t5)*x4*x5*x7;
poly f16 = (t1*t2)/(5*t3^2)*x1^2*x2+(14*t2*t3)/(t1^2)*x1*x4^2;
poly f17 = 9/8*x2*x3*x4+(13*t1^2*t2)*x3*x4^2;
poly f18 = (6*t1*t3*t6+5*t1*t7*t8+19*t2^2*t8)/(19*t1^2*t6+18*t1*t2*t4
           +13*t1*t5)*x4*x6^2+(t1^2*t5+3*t1^2*t8+t1*t3*t4)/(13*t1*t4
           +19*t1*t6*t8+3*t2*t3)*x4*x12^2;
ideal J3 = f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14,
           f15, f16, f17, f18;
```

15.
```
ring S4 = (0,t1,t2,t3,t4,t5,t6,t7,t8), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14), lp;
poly f1 = 8/(17*t2)*x3^3+(2*t1^3)*x3*x4*x7;
poly f2 = (18*t1^3+t2*t3+8*t2*t4)/(6*t1*t3*t4+6*t1*t3+14*t4^3)*x1*x2*x3
          +(14*t1^2*t4+4*t4^3)/(15*t1^2*t4+19*t1*t2^2+4*t1*t2)*x2^2*x3;
poly f3 = (4*t1*t2^2+16*t1*t2*t6+14*t1*t2)/(12*t1*t2*t5+12*t1*t4*t5
          +13*t2^2*t3)*x4^2*x8+(20*t1*t3+5*t1*t4+10*t2*t3)/(15*t1^2+20*t1*t6
          +13*t8)*x10^2*x13;
poly f4 = (11*t1*t3)/(4*t2^2)*x6^2+(t1*t7)/(t2*t4)*x6*x9^2;
poly f5 = (16*t2*t4+12*t5^2)/(10*t2^2+t4*t5)*x1*x2^2+(10*t1*t3*t5
          +20*t2^2*t3)/(17*t1^2*t3+15*t1*t2*t4)*x4;
poly f6 = (4*t1*t4+8*t2^2*t4+13*t4^3)/(11*t1^2+15*t1*t4^2+3*t4)*x2*x4*x6
          +(5*t2^3+9*t2^2*t4+2*t2^2)/(16*t1^2*t3+5*t1^2+3*t2*t3*t4)*x4*x9^2;
poly f7 = (18*t1^2+18*t2*t3)/(13*t1*t7+4*t2^2)*x4^2*x7 +(11*t1*t2*t3
          +10*t1*t2*t4)/(11*t1*t5+14*t2*t4*t7)*x5*x6*x7;
poly f8 = (14*t1*t2^2+11*t3^3)/(14*t1*t2^2+11*t1+14*t2*t3)*x1*x3*x5;
poly f9 = (2*t2*t5)/(t1*t3)*x2*x4*x8+11/7*x3;
poly f10 = (9*t1+19*t2*t3^2+17*t3^2*t4)/(13*t1*t2^2+10*t1*t3*t4)*x1*x2*x7
           +(8*t1*t2*t4+5*t2*t4+3*t3^3)/(7*t1*t4+2*t2*t3*t4)*x3*x6;
poly f11 = (t5)/(12*t2)*x3*x6*x8+(t1*t4^2)/(5*t2^3)*x3;
poly f12 = (10*t2*t4*t5+14*t3^2*t5+5*t4^3)/(14*t2*t3+6*t4*t5^2
           +4*t5^2)*x1*x4*x7+(19*t1*t2^2 +10*t1*t3*t5+2*t1)/(10*t1*t2*t4+4*t2^3
           +t2^2*t3)*x4*x5*x7;
poly f13 = (9*t1^2*t4+14*t1*t5+6*t2*t3^2)/(13*t1^2*t2+17*t1*t4*t5
           +12*t1*t5)*x2*x9^2+(16*t1^2*t4+19*t2^2*t5+12*t2*t3*t5)/(10*t2*t3*t4
           +7*t2*t4+18*t3^2*t4)*x3*x10^2;
poly f14 = 13/(16*t1)*x1*x2*x4;
poly f15 = (t1*t3+12*t3*t4+t4)/(2*t1*t2+4*t1*t4+11*t2)*x1*x4
           +(3*t1^2+15*t1*t3*t4+17*t2)/(15*t2^2*t3)*x2*x3*x5;
poly f16 = (16*t1^3+2*t1)*x5*x10^2+(6*t1^2)*x6*x7*x9;
poly f17 = (4*t1*t2+12*t3^3)/(11*t2^2)*x5*x11^2+(2*t1*t2^2+18*t1
           +13*t3^2)/(10*t1*t3^2+6*t2*t3^2)*x6^2;
poly f18 = (9*t4)/(8*t1^2*t5+13*t1*t2*t3
           +12*t1*t3*t5)*x1*x2*x5+(14*t1*t2*t5+14*t1*t3*t4
           +7*t2^2*t3)/(4*t1^2*t3+t1*t2^2+19*t2*t3*t5)*x4;
poly f19 = (9*t1*t3+6*t2*t8)/(9*t1*t5+t3*t8)*x3^2*x9
           +(11*t1^2+18*t2)/(16*t1^2+8*t2*t4)*x3*x5*x7;
ideal J4 = f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14,
           f15, f16, f17, f18, f19;
```

16.
```
ring S5 = (0,t1,t2,t3), (x1,x2,x3,x4,x5,x6,x7), lp;
poly f1 = (5*t2^2)/(2*t3^2)*x2^3+(8*t1+16*t3^2)/(5*t1*t2*t3+18*t1*t2)*x2*x3^2;
poly f2 = (12*t1^2)/(19*t2^2)*x1*x2+(16*t3^2)/(5*t1^3)*x4^2*x5;
poly f3 = (5*t1^3+4*t1)*x2^2*x5;
poly f4 = 1/(3*t1)*x1^2+(10*t2^2)*x1*x2*x3;
poly f5 = (2*t1*t2+9*t2*t3)/(12*t1^2*t3+9*t2*t3^2)*x3*x4*x7+(8*t1^2+19*t1*t3^2)*x4^3;
poly f6 = (17*t1*t2)*x1*x3^2;
poly f7 = 2/(4*t1^2+7*t1+3*t2)*x1*x2+(4*t1^3)*x2^2*x4;
ideal J5 = f1, f2, f3, f4, f5, f6, f7;
```

17.
```
ring S6 = (0,t1,t2,t3,t4,t5,t6,t7,t8), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14),  lp;
poly f1 = (10*t1^2*t2+t2^2)*x1^3+(4*t1^2+9*t2)/(2*t2^3)*x2^2;
poly f2 = (10*t1^2*t2+t2^2)*x1^3+(4*t1^2+9*t2)/(2*t2^3)*x2^2;
poly f3 = (17*t1*t6+12*t4*t5)/(8*t3*t4+15*t7)*x2*x4*x9
          +(7*t1*t2+6*t2*t4)/(3*t2^2+7*t8^2)*x11^3;
poly f4 = (16*t2^2+17*t4^2)/(7*t1*t3+17*t1*t4)*x4*x12^2
          +(19*t2+15*t3*t4)/(13*t3)*x10*x13^2;
```

```
poly f5 = (19*t1^2+16*t1*t2*t3)/(7*t1*t2*t5+4*t2*t4*t5)*x2*x3*x4
          +(11*t3+6*t5)/(19*t1^2+7*t2*t5)*x4^2;
poly f6 = (3*t1*t3+10*t2*t4)/(15*t1^2*t6+10*t1*t3^2)*x1^2*x2
          +(6*t1*t2*t5+7*t1*t3*t4)/(10*t1^2*t5+3*t2^3)*x3^2;
poly f7 = (15*t1*t2+20*t2*t3^2)/(t1^3)*x1+(t2)/(19*t1*t2+6*t1*t3)*x3^2;
poly f8 = (19*t5)/(12*t1)*x1*x2+(7*t3*t5)/(11*t1)*x5*x11;
poly f9 = (13*t3)/(17*t4)*x1*x4*x7+1/(8*t1)*x3*x4*x9;
poly f10 = (8*t1*t4+20*t2*t8+12*t3*t7)/(11*t1^2+17*t2*t3+9*t2)*x2+(16*t1*t4
           +8*t2*t7+9*t3*t8)/(11*t1^2+t1+13*t2*t3)*x3;
poly f11 = (18*t1*t3+8*t1*t8)/(18*t5*t7+9*t5*t8)*x1^2*x3
           +(2*t1^2+2*t1*t8)/(10*t4*t6+11)*x3;
poly f12 = 15/(2*t2)*x1*x2*x7;
poly f13 = (10*t1^3+13*t1*t2*t3)/(13*t3*t4+7*t4^2)*x2*x3*x12
           +(11*t1*t5+12*t3*t4)/(17*t1^2+6*t1*t4)*x7*x9;
poly f14 = (3*t2^3+5*t2^2)/(14*t1)*x4*x5+(8*t1)/5*x7;
poly f15 = (20*t3^2+13*t4^2)/(11*t1*t2)*x1*x2*x3+(2*t3
           +19*t4^2)/(17*t2^2+17*t3*t4)*x2*x6*x7;
poly f16 = (11*t1^2*t4+t1*t3^2+20*t1*t3*t5)/(13*t1^2*t3
           +8*t1*t3*t5+16*t5)*x3*x5*x10+(2*t1+11*t2
           +2*t5)/(20*t1*t2+8*t2*t3+12*t2)*x3*x6*x7;
poly f17 = (14*t2*t4+15*t3^2+11*t3*t4)/12*x1^3+(6*t1^2*t2
           +8*t1^2*t3+9*t1*t2^2)/(13*t1*t4^2+8*t1+t2^2)*x1*x2^2;
poly f18 = (9*t1)/17*x2*x7*x9+(19*t1)/(3*t2)*x4*x5*x9;
poly f19 = 1/(20*t1)*x2*x6+(8*t2^2)/(13*t1*t3)*x9;
poly f20 = (5*t1*t3)/(7*t2^2)*x4*x6*x7+(16*t2)/(15*t1^2)*x7*x8^2;
poly f21 = (2*t1*t4+19*t2*t4)/(4*t1*t2)*x3*x7*x9
           +(3*t1*t2+4*t1*t4)/(4*t1^2+t3*t4)*x5^2;
ideal J6 = f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14,
           f15, f16, f17, f18, f19, f20, f21;
```

18.
```
ring S7 = (0, t1,t2,t3,t4,t5), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10), dp;
poly f1 = (28*t2)/(41*t1)*x1^2*x2^2+95/18*x1^3;
poly f2 = (84*t1^3+63*t1*t2^2)/(26*t1^3+55*t2*t3^2)*x1*x5^3+(82*t1^3
          +85*t1*t2*t3^2)/(65*t1^3*t2+69*t2^4)*x1^2*x3*x6;
poly f3 = (27*t1^4+34*t1^3*t2)/(69*t1^3*t2+82*t2^3)*x4*x7^3
          +(83*t1^4+3*t2^3)/(61*t1^2*t2)*x3*x7*x9;
poly f4 = (99*t2)/(97*t1)*x1*x3*x7^2+(35*t1^3)/(71*t3^2*t4)*x5*x6*x8;
poly f5 = (7*t1)/(44*t2)*x2*x3^2 +(18*t1^3)/(53*t2)*x2^2;
ideal J7 = f1, f2, f3, f4, f5;
```

19.
```
ring S8 = (0, t1,t2,t3,t4,t5), (x1,x2,x3,x4,x5,x6,x7,x8,x9,x10), dp;
poly f1 = (9*t1^2+9*t1+35*t2^2)/(26*t1*t2+48*t2)*x3^2*x7^2;
poly f2 = (19*t1*t2^2*t3+27*t1*t2*t3^2+80*t2^2*t3^2)/(67*t1^3*t3
          +26*t1^2*t2*t3+7*t1^2*t2*t4)*x1*x2*x3*x4+(79*t1^3*t3+11*t1^2*t2^2
          +16*t2^4)/(66*t1^3*t2+90*t1^3*t3+41*t1*t2*t3^2)*x3^3*x7;
poly f3 = (19*t1)/49*x1*x3^2*x7+(62*t1)*x6*x8;
poly f4 = (31*t1)/(45*t2)*x3^2*x6+(71*t2)/(44*t1)*x1^2*x8;
poly f5 = (37*t1^2)/(8*t2^2)*x1*x2*x3^2+83/(49*t1)*x1^2*x3*x4;
ideal J8 = f1, f2, f3, f4, f5;
```

20.
```
ring S9 = (0,u1,u2,u3,u4,u5), (a,b,x,y,z,u,v,w), dp;
poly f1 = (36*u1)*z-136;
poly f2 = (66*u1^2)*a*z+(78*u1^2*u2)*z*v+(-1056*u1)*a
          +(90*u5)*x+336*y+(-90*u1)*u;
poly f3 = (-162*u1^2)*a^2+(50*u1)*a*y+(180*u1^2)*a*z+(55*u1^2)*z*u
```

```
          +(-284*u1^2*u2)*a*v+(60*u1*u2)*y*v+(-112*u4)*b
          +(260*u5)*x+(70*u3*u5)*w;
poly f4 = (28*u1^3*u2)*a*z*v+(-648*u1^2)*a^2+(36*u4*u5)*b*x
          +(128*u1)*a*y+(36*u1*u4)*b*z+(-300*u1^2)*a*u
          +(40*u1)*y*u+(44*u1*u2*u5)*x*v+(192*u3*u5)*w;
poly f5 = (-162*u1^2*u4)*a^2*b+(2*u1*u4)*a*b*y+(3*u1^2*u5)*a*x*u
          +(4*u1^2*u2*u3*u5)*a*v*w+(6*u3*u4*u5)*b*w;
poly f6 = (u2*u4)*b*y+(u1^3)*z^2+(-55*u1*u3*u5)*z*w;
poly f7 = (-162*u4*u5)*b*x+(u1^2*u2^2)*v^2;
poly f8 = (28*u1^2*u2*u5)*u*v+(u3^2*u5^2)*w^2;
poly f9 = (u2*u4)*b*y+(u1^3)*z^2+(-55*u1*u3*u5)*z*w;
poly f10 = (-240*u1)*a+112*y+(420*u1)*z+(-64*u1*u2)*v;
ideal J9 = f1, f2, f3, f4, f5, f6, f7, f8, f9, f10;
```

Bibliography

[1] W. W. Adams and P. Loustaunau. *An introduction to Gröbner bases.* American Mathematical Society, 1994.

[2] E. A. Arnold. Modular algorithms for computing Gröbner bases. *J. Symb. Comput.*, 35(4):403–419, 2003.

[3] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309. ACM, 1988.

[4] J. Böhm, W. Decker, C. Fieker, and G. Pfister. The use of bad primes in rational reconstruction. *Math. Comp.*, 84(296):3013–3027, 2015.

[5] D. K. Boku, W. Decker, and C. Fieker. `nfmodstd.lib`. A SINGULAR 4-0-2 library for computing Gröbner bases of ideals in polynomial rings over algebraic number fields, 2015. Included in SINGULAR 4-0-2 as `algemodstd.lib` and renamed to `nfmodstd.lib` for subsequent releases.

[6] D. K. Boku, W. Decker, and C. Fieker. `ffmodstd.lib`. A SINGULAR 4-0-3 library for computing Gröbner bases of ideals in polynomial rings over algebraic function fields, 2016.

[7] D. K. Boku, W. Decker, and C. Fieker. `nfmodsyz.lib`. A SINGULAR 4-0-3 library for computing syzygy modules of given submodules of free modules over algebraic number fields, 2016.

[8] D. K. Boku, W. Decker, C. Fieker, and A. Steenpass. Gröbner bases over algebraic number fields. In *Proceedings of the 2015 International Workshop on Parallel Symbolic Computation*, PASCO '15, pages 16–24, New York, NY, USA, 2015. ACM.

[9] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symb. Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[10] M. Brickenstein. Slimgb: Gröbner bases with slim polynomials. *Rev. Mat. Complut.*, 23(2):453–466, 2010.

[11] B. Buchberger. An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Translation from the German. *J. Symb. Comput.*, 41(3-4):475–511, 2006.

[12] R. L. Burden and J. Faires. *Numerical analysis. 9th ed.* Boston, MA: PWS Publishing Company; London: ITP International Thomson Publishing, 5th ed. edition, 1993.

[13] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms. An introduction to computational algebraic geometry and commutative algebra.* Springer, New York, third edition, 2007.

[14] D. A. Cox, J. B. Little, and D. O'Shea. *Using algebraic geometry.* Graduate texts in mathematics. Springer, New York, 1998.

[15] Şaban Alaca and K. S. Williams. *Introductory algebraic number theory.* Cambridge: Cambridge University Press, 2004.

[16] A. Cuyt and W.-S. Lee. Sparse interpolation of multivariate rational functions. *Theor. Comput. Sci.*, 412(16):1445–1456, 2011.

[17] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 4-0-2 – A computer algebra system for polynomial computations, 2015. http://www.singular.uni-kl.de.

[18] W. Decker and C. Lossen. *Computing in algebraic geometry : A quick start using* SINGULAR. Algorithms and computation in mathematics. Springer New Delhi, Berlin, Heidelberg, New-York, 2006. Information sur SINGULAR l'adresse http://www.singular.uni-kl.de.

[19] H. Derksen and G. Kemper. *Computational invariant theory.* Encyclopaedia of mathematical sciences. Springer, Berlin, New York, 2002.

[20] C. Eder and J. Perry. F5C: A variant of Faugère's {F5} algorithm with reduced Gröbner bases. *Journal of Symbolic Computation*, 45(12):1442 – 1458, 2010. MEGA2009.

[21] D. Eisenbud. *Commutative algebra. With a view toward algebraic geometry.* Berlin: Springer-Verlag, 1995.

[22] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra*, 139(1-3):61–88, 1999.

[23] S. Gao, Y. Guan, and F. Volny, IV. A new incremental algorithm for computing Gröbner bases. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 13–19, New York, NY, USA, 2010. ACM.

[24] S. Gao, F. V. IV, and M. Wang. A new algorithm for computing Gröbner bases, 2010.

[25] G.-M. Greuel and G. Pfister. *A Singular introduction to commutative algebra. With contributions by Olaf Bachmann, Christoph Lossen and Hans Schönemann.* Springer, Berlin, second extended edition, 2007.

[26] A. Hashemi, G. Pfister, H. Schönemann, A. Steenpass, and S. Steidel. `modstd.lib`. A Singular 4-0-2 library for computing Gröbner bases of ideals using modular methods, 2014.

[27] N. Idrees, G. Pfister, and S. Steidel. Parallelization of modular algorithms. *J. Symb. Comput.*, 46(6):672–684, 2011.

[28] F. Jarvis. *Algebraic number theory.* Cham: Springer, 2014.

[29] S. M. M. Javadi and M. Monagan. Parallel sparse polynomial interpolation over finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pages 160–168, New York, NY, USA, 2010. ACM.

[30] E. Kaltofen, Y. N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '90, pages 135–139, New York, NY, USA, 1990. ACM.

[31] E. Kaltofen, W.-s. Lee, and A. A. Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. In *Proc. 2000 (ISSAC'00)*, pages 192–201.

[32] E. Kaltofen and W. shin Lee. Early termination in sparse interpolation algorithms. *J. Symb. Comput.*, 36(3-4):365–400, 2003.

[33] S. Khodadad and M. Monagan. Fast rational function reconstruction. In *Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation*, ISSAC '06, pages 184–190, New York, NY, USA, 2006. ACM.

[34] M. Kreuzer and L. Robbiano. *Computational commutative algebra. I.* Berlin: Springer, 2000.

[35] M. Noro. An efficient implementation for computing Gröbner bases over algebraic number fields. In *Mathematical software – ICMS 2006. Second international congress on mathematical software, Castro Urdiales, Spain, September 1–3, 2006. Proceedings*, pages 99–109. Springer, 2006.

[36] G. Pfister. On modular computation of standard basis. *Analele tiinifice ale Universitii Ovidius" Constana. Seria: Matematic*, 15(1):129–138, 2007.

[37] A. Steenpass. parallel.lib. A Singular library providing an abstraction layer for parallel skeletons , 2013.

[38] The Magma Group. The Magma Computational Algebra System V2.21-2, 2015. http://magma.maths.usyd.edu.au.

[39] The Magma Group. The Magma Computational Algebra System V2.21-11, 2016. http://magma.maths.usyd.edu.au.

[40] C. Traverso. Gröbner trace algorithms. In P. Gianni, editor, *Symbolic and Algebraic Computation*, volume 358 of *Lecture Notes in Computer Science*, pages 125–138. Springer, 1989.

[41] M. van Hoeij and M. Monagan. Algorithms for polynomial gcd computation over algebraic function fields. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, ISSAC '04, pages 297–304, New York, NY, USA, 2004. ACM.

[42] J. von zur Gathen and J. Gerhard. *Modern computer algebra.* Cambridge University Press, Cambridge, third edition, 2013.

[43] P. S. Wang. A p-adic algorithm for univariate partial fractions. Symbolic and algebraic computation, Proc. ACM Symp., Snowbird/Utah 1981, 212-217 (1981)., 1981.

[44] K. Yokoyama. Usage of modular techniques for efficient computation of ideal operations - (invited talk). In *Computer Algebra in Scientific Computing - 14th International Workshop, CASC 2012, Maribor, Slovenia, September 3-6, 2012. Proceedings*, pages 361–362, 2012.

[45] R. Zippel. Probabilistic algorithms for sparse polynomials. Symbolic and algebraic computation, EUROSAM '79, int. Symp., Marseille 1979, Lect. Notes Comput. Sci. 72, 216-226 (1979)., 1979.

Wissenschaftlicher Werdegang

Dereje Kifle Boku

- 2008: Bachelor in Mathematik, Haramaya Universität, Äthiopien
- 2011: Master in Mathematik, Addis Ababa Universität, Äthiopien
- 2016: PhD in Mathematik, TU Kaiserslautern, Deutschland

Scientific Career

Dereje Kifle Boku

- 2008: Bachelor in Mathematics, Haramaya University, Ethiopia
- 2011: Master in Mathematics, Addis Ababa University, Ethiopia
- 2016: PhD in Mathematics, University of Kaiserslautern, Germany