Homotopy techniques for multiplication modulo triangular sets

Muhammad Foizul Islam Chowdhury

Homotopy techniques for multiplication modulo triangular sets (Spine Title: Homotopy techniques for triangular multiplication)

(Thesis format: Monograph)

by

Muhammad Foizul Islam Chowdhury

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada
April 30, 2009

Abstract

Triangular representations are a versatile tool to manipulate systems of polynomial equations; however, many basic complexity questions are still open with respect to such objects. This matter of fact appears clearly with multiplication modulo triangular sets: it plays an essential part in algorithms for solving polynomial systems, but the cost of this operation remains difficult to estimate precisely.

This thesis studies the complexity of this operation. Following previous work by Li, Moreno Maza and Schost, we propose an algorithm that relies on homotopy and fast evaluation-interpolation techniques. We obtain a quasi-linear time complexity for substantial families of examples, for which no such result was known before. Applications are given notably to addition of algebraic numbers in small characteristic.

Keywords. Multivariate polynomial, Polynomial multiplication, Triangular Set, Homotopy.

Acknowledgments

I would first like to thank my thesis supervisor Assistant Professor Éric Schost in the Department of Computer Science at University of western Ontario. The door to Prof. Éric Schost office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently helped me on the way of this thesis and steered me in the right direction whenever he thought I needed it. I am grateful to him for his excellent support to me in all arenas.

Sincere thanks and appreciation are extended to all the members from our Ontario Research Centre for Computer Algebra (ORCCA) lab and the Computer Science Department for their invaluable teaching and assistance.

Contents

Certificate of Examination ii

Abstract iii

Acknowledgments iv

1 Introduction 1

1.1 Problem statement, overview of our results 3

1.2 Our approach 5

1.3 Previous work 5

1.4 Outlook of this thesis 6

2 Background and Preliminaries 7

2.1 Triangular sets: definitions and notation 7

2.2 Preliminaries: complexity of basic operations 9

3 Evaluation and Interpolation 15

3.1 Equiprojectable sets 16

3.2 Evaluation 18

3.3 Interpolation 22

3.4 Associated triangular set 25

3.5 Multiplication 27

4 Homotopy Techniques for Multiplication 29

4.1 Introduction 29

4.2 A worked example 30

4.3 Computing the roots of **S** 33

4.4 The multiplication algorithm 36

5 Precision Analysis **38**
5.1 First worked example 38
5.2 Second worked example 42
5.3 Precision analysis for the general case 44

6 Families of application **51**
6.1 Main family of examples 51
6.2 Computing with Cauchy modules 53
6.3 Polynomial multiplication 56
6.4 Exponential generating series multiplication 58

7 Experimental Results **63**
7.1 Presentation of the problem 63
7.2 Our algorithm 64
7.3 Experimental results 66

8 Conclusions and future work **69**

Curriculum Vitae **74**

List of Algorithms

1 LiftRoots$(T, a_1, \ldots, a_d, \ell)$ 14
2 Eval(A, V) 20
3 Interp(f, V) 24
4 AssociatedTriangularSet(f, V) 26
5 Mul(A, B, V) 28
6 LiftRootsMultivariate$(V, \mathbf{S}, \ell)$ 35
7 Mul$(A, B, \mathbf{T}, V)$ 37
8 ComposedSum(f, g) 66

List of Figures

2.1 Example of sub product tree 12

3.1 Equiprojectable set 17
3.2 Non equiprojectable set 17
3.3 Equiprojectable set in dimension 3 18
3.4 A single butterfly 21
3.5 A single butterfly, second level 21
3.6 The butterfly structure in 2 variables 21
3.7 Tree structure of the roots in 3 variables 22
3.8 Butterfly of multivariate evaluation 23
3.9 The butterflies of multivariate interpolation 25

4.1 Tree structure of four roots 32

5.1 Tree structure of the reduction by S_2. 40
5.2 A view of the reduction tree in the coordinates (f_1, f_2) 41
5.3 Tree structure of the reduction by S_3. 43

7.1 Detailed timings for our algorithm 67
7.2 Timings in magma 68

List of Tables

2.1 Complexities of polynomial multiplication algorithms 10

Chapter 1

Introduction

Solving systems of linear or non-linear, algebraic or differential equations, is a fundamental problem in mathematical sciences and engineering. In this thesis, we focus on one of these questions, the study of systems of polynomials (*i.e.*, non-linear) equations.

Many approaches, and many data representations, are available to deal with such systems. Here, our interest will be on so-called *triangular representations*, or *triangular sets*. The precise definition of triangular set is given in Chapter 2; for the moment, one should just think of these as a set of equations with a triangular shape, that generalize triangular matrices to non-linear situations.

It is known that the common roots of any set of multivariate polynomials can be represented by finitely many triangular sets. This decomposition is also known as triangular decomposition because of the triangular shape of the objects it involves.

The theory of triangular representations is especially complex for systems with infinitely many solutions, that is, in positive dimension. In this case, several related definitions have been introduced. Wu Wen Tsun [41] introduced the first method for solving systems of algebraic equations by means of *characteristic sets* in 1987. In this method many redundant components are produced without any function to remove them. In 1993, Kalkbrener introduced the related notion of *regular chains* [17]. At the same time Lazard [19] another, different, notion of triangular representation that actually strengthened the notion of regular chain. It was not until [27] that the relations between all such definitions were clarified. Improved and unified algorithms are in [26] and are the basis of the `RegularChains` library [21].

On the contrary, the emphasis of this thesis is for systems with finitely many solutions, that is, with dimension zero. In this case, the definitions are much more straightforward, since the underlying geometric problems are much simpler than in positive dimension.

Then, a high-level question is to provide sharp estimates on the cost of solving a polynomial system by means of triangular representations. This problem itself relies on several difficult "lower-level" questions, such as the cost of basic operations with triangular sets.

This thesis discusses one of these questions, arguably the most important one: the complexity of multiplication modulo a triangular set. We illustrate it by an example first. Let

$$\begin{aligned} T_2(X_1, X_2) &= X_2^2 + X_1X_2 \\ T_1(X_1) &= X_1^2 + 3X_1 \end{aligned}$$

be two polynomials in variables X_1, X_2; since T_1 depends only on X_1 and T_2 on X_1, X_2, they fit our definition of a triangular set. Consider now two polynomials

$$\begin{aligned} A &= X_1X_2 + X_2 + X_1 + 1 \\ B &= X_1X_2 + X_2 + X_1 + 1. \end{aligned}$$

These polynomials are *reduced* with respect to the triangular set $\mathbf{T} = (T_1, T_2)$: their degrees in X_1 (resp. X_2) are less than the degree of T_1 in X_1 (resp. of T_2 in X_2). The product of these two polynomials is

$$AB = X_1^2X_2^2 + 2X_2^2X_1 + 2X_2X_1^2 + 4X_1X_2 + X_2^2 + 2X_2 + X_1^2 + 2X_1 + 1$$

before any reduction. Since the degrees have grown, it is possible to reduce this product with respect to the triangular set $\mathbf{T}$:

- by computing the remainder C of AB in the Euclidean (long) division by T_2, using X_2 as the main variable (concretely, this amounts to replace X_2^2 by $-X_1X_2$);
- then, by computing the remainder D of C in the Euclidean (long) division by T_1, using now X_1 as the main variable (concretely, this amounts to replace X_1^2 by $-3X_1$).

In the end, we obtain

$$D = -6X_1X_2 + 2X_2 - X_1 + 1.$$

This operation is actually critical to many algorithms for polynomial system solving. Indeed, many higher-level routines are built on top of it, such as triangular decomposition of algebraic varieties [26, 24], inversion modulo a triangular set [11, 18, 23, 25], lifting techniques for modular algorithms [34, 33, 11], all of those eventually contributing to solving systems of equations [22]. For instance, the *RegularChain* Maple library [21] now partly relies on a high-performance C implementation of basic arithmetic operations modulo triangular sets, and in particular on multiplication algorithms.

1.1 Problem statement, overview of our results

This thesis gives an algorithm for doing multiplication modulo a triangular set efficiently; our focus is on the complexity aspects of this problem. The main results of this thesis can be found in the preprint [4].

Our problem is the generalization of the example seen above: we work in $\mathsf{R}[X_1, \ldots, X_n]$, where R is a ring, and we are given a set of polynomials

$$\mathbf{T} \left| \begin{array}{l} T_n(X_1, \ldots, X_n) \\ \quad \vdots \\ T_2(X_1, X_2) \\ T_1(X_1) \end{array} \right.$$

that form the triangular set $\mathbf{T} = (T_1, \ldots, T_n)$. As input, we also consider two polynomials A, B reduced modulo $\mathbf{T}$: as in the example, this means that the degrees of A and B in X_i is less than the degree of T_i in X_i, for all i.

Our question is to determine how many operations in R it takes to compute the product AB modulo $\langle \mathbf{T} \rangle$, that is, reduced by $T_1, \ldots, T_n$. The goal is to obtain an algorithm of complexity linear (up to logarithmic factors) in the "size of the problem", that is, the number of monomials that appear in the input and output.

Hereafter, for simplicity, we call *quasi-linear* algorithm an algorithm with a running-time linear in the input and output size, up to logarithmic factors. Correspondingly, we use the notation $f = O\tilde{~}(g)$ to indicate that $f = O(g \log(g)^\alpha)$, for some constant α.

As in the example, the direct approach is to perform a polynomial multiplication, followed by the reduction modulo $\langle \mathbf{T} \rangle$, by a generalization of Euclidean division.

As far as complexity is concerned, when the number of variables grows, this kind of approach cannot give quasi-linear algorithms. Consider for instance the case where all T_i have degree 2 in their main variables X_i. Then, A and B both have degree at most 1 in each variable, so they can have no more than 2^n monomials. However, their product before reduction has degree at most 2 in each variable, so it can have up to 3^n monomials: this was the case in the previous example. Finally, after reduction, the number of monomials is reduced to 2^n again. If we let $\delta = 2^n$ be a measure of the input and output size (that is, number of monomials in A, B and in the result), the cost of such an algorithm is at least $3^n = \delta^{\log_2 3} \simeq \delta^{1.59}$. This is not a quasi-linear algorithm.

In this thesis, generalizing an idea by Li, Moreno Maza and Schost [23], we show that a different approach can lead to a quasi-linear time algorithm, in cases where the monomial support of $\mathbf{T}$ is sparse, or when the polynomials in $\mathbf{T}$ have a low total degree. This will for example be the case for systems of the form

$$\left|\begin{array}{l} X_n^2 - 2X_{n-1} \\ \vdots \\ X_2^2 - 2X_1 \\ X_1^2 \end{array}\right. \quad \text{or} \quad \left|\begin{array}{l} X_n^2 - X_{n-1} \\ \vdots \\ X_2^2 - X_1 \\ X_1^2, \end{array}\right. \tag{1.1}$$

whose applications are described in Chapter 6. Our result also applies to the following construction: start from $F \in \mathsf{R}[X]$, say $F = X^3 - X^2 + X - 3$, and define the so-called "Cauchy modules" [31], which are triangular sets used in effective Galois theory [31, 1, 30], or in the study of polynomial systems with symmetries [10, 15]:

$$\left|\begin{array}{lclcl} T_3(X_1,X_2,X_3) & = & \frac{T_2(X_1,X_2)-F(X_1,X_3)}{X_2-X_3} & = & X_3+X_2+X_1-1 \\ T_2(X_1,X_2) & = & \frac{T_1(X_1)-T_1(X_2)}{X_1-X_2} & = & X_2^2+X_2X_1-X_2+X_1^2-X_1+1 \\ T_1(X_1) & = & F(X_1) & = & X_1^3-X_1^2+X_1-3. \end{array}\right. \tag{1.2}$$

For examples (1.1) and (1.2), our algorithms give the following results:

- for $\mathbf{T}$ as in (1.1), multiplication modulo $\langle \mathbf{T} \rangle$ can be performed in quasi-linear time $O\tilde{~}(\delta)$, where $\delta = 2^n$ is the input and output size, and where as indicated above, $O\tilde{~}(\delta)$ stands for $O(\delta \log(\delta)^\alpha)$, for some constant α.

- for $\mathbf{T}$ as in (1.2), with $n = \deg(F)$, the polynomials $T_1, \ldots, T_n$ have degrees in $X_1, \ldots, X_n$ of the form $n, n-1, \ldots, 1$ (in the example, this is $3, 2, 1$). Here multiplication modulo $\langle \mathbf{T} \rangle$ can be performed in quasi-linear time $O\tilde{~}(\delta)$, where the input and output size is $\delta = n \cdots 1 = n!$.

For both cases, to our knowledge, no previous algorithm was known featuring such complexity estimates.

1.2 Our approach

To obtain a quasi-linear cost, we have to avoid multiplying A and B as polynomials, since the number of monomials may prevent us from reaching our target. Our solution is to use evaluation and interpolation techniques, just as Fast Fourier Transform (FFT) multiplication of univariate polynomials is multiplication modulo $X^n - 1$.

Unfortunately, fast evaluation and interpolation may not be possible directly, if $\mathbf{T}$ does not have roots in R (this is for instance the case in the examples (1.1) and (1.2)). However, they become possible using deformation techniques described below.

We construct a new triangular set $\mathbf{U}$ with all roots in R, and multiply A and B modulo $\mathbf{S} = \eta\mathbf{T} + (1 - \eta)\mathbf{U}$, where η is a new variable. The triangular set $\mathbf{S}$ has roots which are power series in η (that is, they belong to $\mathsf{R}[[\eta]]$), so one can use evaluation-interpolation techniques over $\mathsf{R}[[\eta]]$. This approach will be called the *homotopy* method, since it introduces a whole family of triangular sets $\mathbf{S}$ that connects the "nice" triangular set $\mathbf{U}$ to the target triangular set $\mathbf{T}$.

This idea was introduced by Li, Moreno Maza and Schost in [23], but was limited to the case where all polynomials in $\mathbf{T}$ are univariate: T_i was restricted to depend on X_i only, so this did not apply to the examples above. Here, we extend this idea to cover such examples; our main technical contribution is a study of precision-related issues involved in the power series computations, and how they relate to the monomial support of $\mathbf{T}$.

1.3 Previous work

It is only recently that fast algorithms for triangular representations have been thoroughly investigated from the complexity viewpoint; thus, previous results on efficient multiplication algorithms are scarce. All natural approaches introduce in their cost estimate an overhead of the form k^n, for some constant k.

The main challenge (still open) is to get rid of this exponential factor unconditionally: we want algorithms of cost $O\tilde{}(\delta)$, where δ is the number of monomials in A, B and their product modulo $\langle\mathbf{T}\rangle$. For instance, with $T_i = X_i^{d_i}$, the first complexity result of the form $O(\delta^{1+\varepsilon})$, for any $\varepsilon > 0$, was in [35]; note that this is not quite as good as $O\tilde{}(\delta)$.

The previous work [23] gives a general algorithm of cost $O\tilde{}(4^n\delta)$, that holds for any triangular set. That algorithm uses fast Euclidean division. Previous mention of such complexity estimates (with a constant higher than 4) are in [18].

As said above, in [23], one also finds the precursor of the algorithm presented here; the algorithm of [23] applies to families of polynomials having $T_i \in \mathsf{R}[X_i]$, and achieves the cost $O(\delta^{1+\varepsilon})$ for any $\varepsilon > 0$. In that case, the analysis of the precision in power series computation was immediate. Our main contribution here is to perform this study in the general case, and to show that we can still achieve similar costs for much larger families of examples.

1.4 Outlook of this thesis

The thesis is organized as follows.

- Chapter 2 gives some background and some useful notation used in this thesis.
- Chapter 3 presents basic evaluation-interpolation algorithms for *equiprojectable* sets, which are extensions of algorithms known for univariate polynomials. We deduce a multiplication algorithm which works for triangular sets $\mathbf{T}$ whose roots are known, and are in R.
- Chapter 4 describes our multiplication algorithm using the homotopy method when the triangular set $\mathbf{T}$ does not have all its roots in R.
- Chapter 5 provides the key technical result, an estimate on the precision of some power series computations that take place in the algorithm of Chapter 4.
- Chapter 6 presents some families of examples where our algorithm gives quasi-linear running time.
- Chapter 7 is dedicated to experimental results, and an application to addition of algebraic numbers.
- Chapter 8 presents our conclusions and future work.

Chapter 2

Background and Preliminaries

The aim of this chapter is to provide the background and some notation used in this thesis: we start by recalling basic definitions, then introduce some classical algorithms for fast polynomial arithmetic in one variable.

2.1 Triangular sets: definitions and notation

In this section, we give the formal definition of a triangular set and introduce all necessary notation.

Let R be a ring. A family of non-constant polynomials $\mathbf{T} = (T_1, \dots, T_n)$ in $\mathsf{R}[X_1, \dots, X_n]$ is a *triangular set* if:

- for $i = 1, \dots, n$, T_i is in $\mathsf{R}[X_1, \dots, X_i]$;
- for $i = 1, \dots, n$, the leading coefficient of T_i in X_i is equal to 1 (we say that T_i is *monic* in X_i),
- for $i = 2, \dots, n$, T_i is *reduced* modulo $T_1, \dots, T_{i-1}$, in the sense that $\deg(T_i, X_j) < \deg(T_j, X_j)$ holds for $1 \leq j < i$.

In all that follows, we will let d_i be the degree $\deg(T_i, X_i)$.

Such families of polynomials have been the object of many previous works. As far as we are concerned, a landmark paper is Lazard's [20], which formally introduced such objects and gave basic algorithms to compute them.

Observe that (if R is a field), such a family of polynomials can have at most a finite number of solutions:

- $T_1(X_1)$ can have at most d_1 roots;

- for each possible root x_1 of T_1, $T_2(x_1, X_2)$ can have at most d_2 roots;
- etc.

Thus, the triangular set $\mathbf{T}$ has at most $d_1 \cdots d_n$ distinct roots. In other words, the solution set of $\mathbf{T}$ has *dimension zero.*

Example. The following polynomials T_1 and T_2 form a triangular set $\mathbf{T} = (T_1, T_2)$:

$$\begin{aligned} T_2(X_1, X_2) &= X_2^2 + X_2 X_1 \\ T_1(X_1) &= X_1^2 + 3X_1 \end{aligned}$$

Here T_1 and T_2 both are monic (their leading coefficients in X_1 (resp. X_2) is one. In addition, T_2 is reduced with respect to T_1, *i.e.*, $\deg(T_2, X_1) < 2 = \deg(T_1, X_1)$.

Next, we introduce useful notation used in the rest of this thesis. Let $\mathbf{d} = (d_1, \ldots, d_n)$ be a vector of positive integers: in what follows, these will represent the main degrees of the polynomials in our triangular sets. We will always suppose $d_i \geq 2$ for all i, for reasons explained at the end of this section.

Recall that R is our base ring and that $X_1, \ldots, X_n$ are indeterminates over R. We let $M_{\mathbf{d}}$ be the set of monomials

$$M_{\mathbf{d}} = \left\{ X_1^{e_1} \cdots X_n^{e_n} \mid 0 \leq e_i < d_i \text{ for all } i \right\}$$

and we denote by $\mathsf{Span}(M_{\mathbf{d}})$ the subset

$$\mathsf{Span}(M_{\mathbf{d}}) = \left\{ A = \sum_{m \in M_{\mathbf{d}}} a_m m \mid a_m \in \mathsf{R} \text{ for all } m \in M_{\mathbf{d}} \right\}$$

of $\mathsf{R}[X_1, \ldots, X_n]$. This is thus the set of polynomials A in $\mathsf{R}[X_1, \ldots, X_n]$ such that $\deg(A_i, X_i) < d_i$ holds for all i. Finally, we let $\delta_{\mathbf{d}}$ be the product $\delta_{\mathbf{d}} = d_1 \cdots d_n$; this is the cardinality of $M_{\mathbf{d}}$. Remark that since all d_i are at least 2, we have the bounds

$$2^n \leq \delta_{\mathbf{d}} \quad \text{and} \quad \sum_{i \leq n} d_1 \cdots d_i \leq 2\delta_{\mathbf{d}}.$$

The former plainly follows from the inequality $2 \leq d_i$; the latter comes from observing that $d_1 \cdots d_i 2^{n-i} \leq d_1 \cdots d_n = \delta_{\mathbf{d}}$; this gives $d_1 \cdots d_i \leq \delta_{\mathbf{d}}/2^{n-i}$. The claim follows by summation.

Let us now consider a triangular set $\mathbf{T} = (T_1, \ldots, T_n)$. The *ideal* $\langle \mathbf{T} \rangle$ is the set of all polynomials F that can be written as

$$F = \sum_{i=1}^{n} F_i T_i,$$

with all F_i in $\mathsf{R}[X_1, \ldots, X_n]$.

The *multi-degree* of the triangular set $\mathbf{T} = (T_1, \ldots, T_n)$ is the n-uple $\mathbf{d} = (d_1, \ldots, d_n)$, with $d_i = \deg(T_i, X_i)$ for $1 \le i \le n$. We say that a polynomial $A \in \mathsf{R}[X_1, \ldots, X_n]$ is *reduced* with respect to $\mathbf{T}$ if $\deg(A, X_i) < d_i$ holds for all i.

When a polynomial is not reduced with respect to $\mathbf{T}$, one can reduce it. For any $A \in \mathsf{R}[X_1, \ldots, X_n]$, we let A' be the polynomial constructed by the generalized Euclidean division process (already presented in the introduction):

- let $A_{n+1} = A$;
- for $i = n, \ldots, 1$, let A_i be the remainder of the Euclidean division of A_{i+1} by T_i, using X_i as the leading variable;
- then, let $A' = A_1$.

We claim that the difference $A' - A$ is in the ideal $\langle \mathbf{T} \rangle$: this is established by induction, by proving that $A_{i+1} - A_i$ is in $\langle \mathbf{T} \rangle$ for all i using the definition of Euclidean division. Besides, A' is reduced with respect to $\mathbf{T}$: this is proved by induction, by showing that for all i, A_i has degree less than d_j in T_j for $j = i, \ldots, n$. The polynomial A' is called the *normal form* of A.

Finally, let us explain why we only consider the case where $d_i \ge 2$ holds for all i. Suppose indeed that $\mathbf{T}$ has multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$, and that $d_i = 1$ for some i. Then, T_i has the form

$$T_i = X_i - r_i(X_1, \ldots, X_{i-1}).$$

This means that we can eliminate X_i by replacing it by $r_i(X_1, \ldots, X_{i-1})$, so that X_i and T_i are actually unnecessary.

2.2 Preliminaries: complexity of basic operations

We continue with a review of some well-known complexity results. In all that follows, the cost of our algorithms will be expressed by counting number of operations in the base ring. Precisely, we count additions, multiplications, and inversions, when they are possible.

Whenever we use a big-O inequality such as $f \in O(g)$, it is implied that there exists a universal constant λ such that $f(v_1, \dots, v_s) \leq \lambda g(v_1, \dots, v_s)$ holds for *all* possible values of the arguments. Finally, we recall that the notation $f \in O\tilde{}(g)$ means that there exists a constant α such that $f \in O(g \log(g)^\alpha)$, where the big-O is to be understood as above.

The following paragraphs review results for several operations on univariate polynomials; all of them can be found in [14].

Polynomial multiplication. The key to fast algorithms for univariate polynomials is polynomial multiplication. We denote by $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ a function such that over any ring, polynomials of degree less than d can be multiplied in $\mathsf{M}(d)$ operations. Besides, as [14], we will suppose that the inequality $\mathsf{M}(d + d') \geq \mathsf{M}(d) + \mathsf{M}(d')$ holds for all d; it will help us simplify some formulas.

Table 2.1 gives the value of $\mathsf{M}(d)$ for various polynomial multiplication algorithms. The algorithm of Cantor and Kaltofen [9] applies over any ring and gives $\mathsf{M}(d) = O(d \log(d) \log\log(d)) = O\tilde{}(d)$, that is, a quasi-linear running time. We describe here a simpler form of it, using Fast Fourier Transform (FFT), since the basic idea of this thesis will comes from FFT-based polynomial multiplication (but in several variables).

The FFT-based polynomial multiplication uses evaluation and interpolation techniques. This algorithm first evaluates both polynomials at roots of unity and does term-wise multiplication of the values thus obtained; after that, the algorithm interpolates the result. Using this technique, one can do the multiplication of two polynomials of degree less than d in $O(d \log(d))$ operations; however, it is not always applicable, since we need roots of unity in R.

Algorithm	$\mathsf{M}(d)$
Classical	$2d^2$
Karatsuba	$O(d^{1.59})$
FFT multiplication	$O(d \log(d))$
Schönhage and Strassen	$O(d \log(d) \log\log(d))$
Cantor and Kaltofen	$O(d \log(d) \log\log(d))$

Table 2.1: Complexities of polynomial multiplication algorithms

Power series. A *power series* is an infinite "formal" sum of the type

$$f = \sum_{i \geq 0} f_i X^i,$$

with coefficients f_i in the ring R; the set of all such power series is written $\mathsf{R}[[X]]$ (the variable of our power series will sometimes also be written η). Since f is an infinite object, we can only handle *truncations* of it, as the finite sums

$$f \bmod X^d = \sum_{0 \le i < d} f_i X^i.$$

Power series can be added, multiplied, etc. Since the truncations $f \bmod X^d$ are actually polynomials, we can use the fast algorithms seen before to multiply them.

A major difference between power series and polynomials is that power series can be inverted. If the constant coefficient f_0 is invertible, there exists a power series g such that $fg = 1$: for instance, with $f = 1 - X$, g is given by

$$g = 1 + X + X^2 + X^3 + X^4 + \cdots$$

In general, given an integer d, the polynomial $\widetilde{g} = g \bmod X^d$ is called the inverse of f modulo X^d, since it satisfies

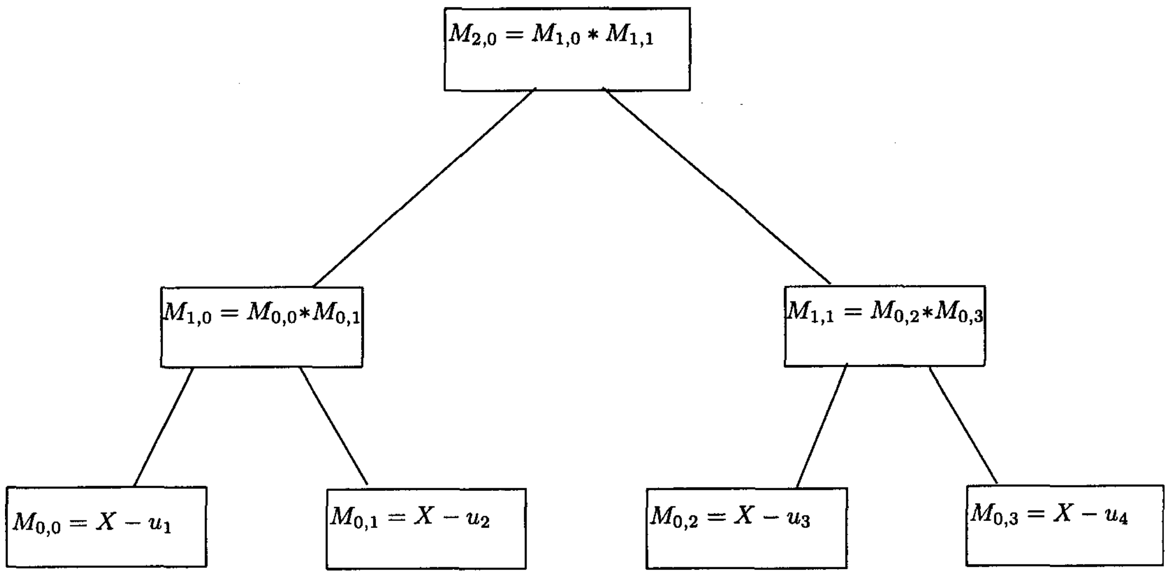
$$f\widetilde{g} = 1 \bmod X^d.$$

Newton iteration is one of the keys to fast algorithms involving power series: it enables us to compute solutions to our problems (here, the inverse of f) modulo the successive powers $X, X^2, X^4, \dots, X^{2^k}, \dots$.

In the case at hand, Newton's iteration enables us to compute $g \bmod X^d$ using $O(\mathsf{M}(d))$ operations, that is, in a cost proportional to the cost of multiplication. The details of the algorithm are given in [14]. We will give more details for another use of Newton iteration, lifting roots of polynomials, in the next paragraphs.

Sub-product tree for multi-point evaluation and interpolation. Our next question is the evaluation of a polynomial $A \in \mathsf{R}[X]$ of degree less than d at some given points $u_1, u_2, \cdots, u_d$ in R: this problem is known as *multi-point evaluation.* The inverse problem consists in recovering the coefficients of A from its values at the points $u_1, u_2, \cdots, u_d$; this is known as interpolation.

For both questions, we use the *sub-product tree.* The leaves of that tree contain the polynomials $X - u_1, X - u_2, \cdots, X - u_d$; each node (except the leaves) holds the product of its children. This is depicted Figure 2.1 for four values u_1, u_2, u_3, u_4. Remark that the root of the tree holds the polynomial $(X - u_1) \cdots (X - u_4)$.



$M_{2,0} = M_{1,0} * M_{1,1}$
$M_{1,0} = M_{0,0}*M_{0,1}$
$M_{1,1} = M_{0,2}*M_{0,3}$
$M_{0,0} = X - u_1$
$M_{0,1} = X - u_2$
$M_{0,2} = X - u_3$
$M_{0,3} = X - u_4$

Figure 2.1: Example of sub product tree

We are going to introduce a short-hand notation to describe the cost of all these operations. We let $\mathsf{C}_0 : \mathbb{N} \to \mathbb{N}$ be a function such that we have, over any ring R:

1. for any $u_1, \ldots, u_d$ in R and any polynomial $A \in \mathsf{R}[X]$ of degree less than d, one can compute all values $A(u_i)$ using $\mathsf{C}_0(d)$ additions and multiplications in R;

2. for any $u_1, \ldots, u_d$ in R, one can compute the coefficients of the polynomial $(X - u_1) \cdots (X - u_d)$ using $\mathsf{C}_0(d)$ additions and multiplications in R;

3. for any $u_1, \ldots, u_d$ in R, with $u_i - u_j$ a unit for $i \neq j$, and any values $v_1, \ldots, v_d$ in R, one can compute the unique polynomial $A \in \mathsf{R}[X]$ of degree less than d such that $A(u_i) = v_i$ holds for all i using $\mathsf{C}_0(d)$ operations in R.

By the results of [14, Chapter 10], one can take $\mathsf{C}_0(d) \in O(\mathsf{M}(d)\log(d)) = \tilde{O}(d)$; thus, all these operations take time proportional to that of multiplication, up to a logarithmic factor.

Lifting roots of polynomials. We continue with the well-known fact that the function C_0 also enables us to estimate the cost of lifting power series roots of a bivariate polynomial.

Consider a polynomial T in $\mathsf{R}[\eta, X]$, where η and X are two variables. We suppose that T is in X. If $a \in \mathsf{R}$ is a root of $T(0, X)$ with multiplicity 1 (that is, $\partial T/\partial X(0, a)$ is non-zero), we want to compute a power series A in $\mathsf{R}[[\eta]]$ such that:

- $A \bmod \eta = a$, that is, the constant coefficient of A is a (we say that A is a *lift* of a);
- $T(\eta, A) = 0$ (that is, A is a root of $T(\eta, X)$).

Since R is not a field, we must actually impose that $\partial T/\partial X(0, a)$ is invertible in R.

Actually, of course, we do not compute the infinite series A but only truncations of it of the form $A \bmod \eta^\ell$. Newton iteration provides a way to compute such truncations starting from a: it is known [14, Lemma 9.21] that the sequence defined by

$$a^{(0)} = a, \quad a^{(i+1)} = a^{(i)} - \frac{T(\eta, a^{(i)})}{\frac{\partial T}{\partial X}(\eta, a^{(i)})} \bmod \eta^{2^{i+1}} \tag{2.1}$$

is well-defined (that is, $\partial T/\partial X(\eta, a^{(i)})$ is invertible modulo $\eta^{2^{i+1}}$) and satisfies

$$T(\eta, a^{(i)}) = 0 \bmod \eta^{2^i}.$$

Using this formula, the following lemma studies the complexity of computing roots of $T(\eta, X)$ modulo η^ℓ. Actually, we are interesting not only in computing one root, but in computing all of them. This lemma shows that we can do it quasi-linear time.

Lemma 1. *Let T be as above, of degree d in X, and suppose that we are given $a_1, \dots, a_d$ in R such that for all i, $T(0, a_i) = 0$ and $\partial T/\partial X(0, a_i)$ is invertible in R. Then, for any integer $\ell \geq 0$, one can compute $A_1, \dots, A_i$ in $\mathsf{R}[\eta]$ such that for all i,*

- *$A_i \bmod \eta = a_i$ and*
- *$T(\eta, A_i) = 0 \bmod X^\ell$;*

using $O(\mathsf{C}_0(d)\mathsf{M}(\ell)) = O\tilde{}(d\ell)$ operations in R.

Proof. The algorithm is given in Algorithm 1. It consists in lifting all roots of T in parallel using the Newton iteration of Equation (2.1). We use the fast multi-point evaluation algorithm described in the previous paragraphs to compute the needed values of T and $\partial T/\partial X$; in the pseudo-code, this is done using a subroutine called EvalUnivariate.

Each pass through the loop at line 2 takes two evaluations in degree d and d inversions, with coefficients that are power series of precision ℓ'. The cost of one inversion modulo $\eta^{\ell'}$ is $O(\mathsf{M}(\ell'))$ [14, Th. 9.4]. Thus, cost of one iteration of line 2 is $2\mathsf{C}_0(d)\mathsf{M}(\ell') + \lambda d\mathsf{M}(\ell')$, for some constant λ. The first term corresponds to evaluations of polynomials at d numbers of series modulo $\eta^{\ell'}$ whereas the second term is for the inversions. The total number of iterations is $r = \lceil \log \ell \rceil$.

Using the inequality $\mathsf{M}(2\ell) \geq \mathsf{M}(\ell) + \mathsf{M}(\ell)$ mentioned before and the technique of [14, Th. 9.4], one can deduces the result. □

Algorithm 1 LIFTROOTS$(T, a_1, \ldots, a_d, \ell)$

```
1: $\ell' \leftarrow 2$
2: while $\ell' < \ell$ do
3:   $v_1, \ldots, v_d \leftarrow$ EvalUnivariate$(T, a_1, \ldots, a_d)$ mod $\eta^{\ell'}$
4:   $w_1, \ldots, w_d \leftarrow$ EvalUnivariate$(\partial T/\partial X, a_1, \ldots, a_d)$ mod $\eta^{\ell'}$
5:   for $i = 1, \ldots, d$ do
6:     $a_i \leftarrow a_i - v_i/w_i$ mod $\eta^{\ell'}$
7:     $\ell' \leftarrow 2\ell'$
8:   end for
9: end while
10: return $[a_i \bmod X^{\ell} \mid 1 \leq i \leq d]$
```

Some simplifying notation. To obtain simpler big-O-free estimates, we let $\mathsf{C}(d) = \Lambda \mathsf{C}_0(d)$, where $\Lambda \geq 1$ is the constant implied in the big-O estimate in the former lemma. Hence, the evaluation and interpolation problems (1), (2) and (3) above can be dealt with in $\mathsf{C}(d)$ operations, and the lifting problem of the previous lemma can be solved in $\mathsf{C}(d)\mathsf{M}(\ell)$ operations. Remark also that $\mathsf{C}(d)$ is $O(\mathsf{M}(d)\log(d))$.

Finally, we introduce another short-hand notation: for a multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$, we write

$$\mathsf{L}(\mathbf{d}) = \sum_{i \leq n} \frac{\mathsf{C}(d_i)}{d_i} \leq n \frac{\mathsf{C}(d)}{d}, \tag{2.2}$$

with $d = \max_{i \leq n} d_i$. Recall that we wrote $\delta_{\mathbf{d}} = d_1 \cdots d_n$. In view of the estimates on C and M, we deduce have the upper bound

$$\mathsf{L}(\mathbf{d}) \in O(\log(\delta_{\mathbf{d}})^3).$$

Chapter 3

Evaluation and Interpolation

Recall the following facts about Fast Fourier Transform: over the complex field $\mathbb{C}$, the roots of the polynomial $T = X^d - 1$ are the roots of unity $V = \{\exp(2ik\pi/d) \mid k = 0, \ldots, d-1\}$. To multiply two polynomials A and B modulo $X^d - 1$, FFT multiplication does the following:

- evaluate A and B at V;
- multiply the values pairwise;
- interpolate the result, in degree less than d.

This scheme can be generalized: if T is *any* univariate polynomial that factors as a product $T = (X - a_1) \cdots (X - a_d)$, multiplication modulo T can be done using the same evaluation-interpolation process at the set $V = \{a_1, \ldots, a_d\}$.

In this chapter, we give an extension of this idea to multivariate situations. We start by recalling from [2] the definition of *equiprojectable sets*: these will be the sets of points at which we do evaluation and interpolation. As in the one-variable case, we deduce an algorithm for multiplication modulo their *associated triangular set* (defined below), which generalizes the case of a single polynomial T we saw above.

These results extend those given in [28, 23], which dealt with the case of points on a regular grid. The extension to our more general context is not very complicated, but did not appear before, as far as we can tell.

In all this chapter, our coefficients are in a ring R, which is not necessarily a field: we will need this extra flexibility in our applications. Unfortunately, this will require us to add some invertibility assumptions to make possible all the divisions we want to do.

We study subsets of R^n and their successive projections on the subspaces R^i, for $i \leq n$. For definiteness, we let R^0 be a one-point set. Then, for $1 \leq j \leq n$, we let π_j be the projection

$$\begin{array}{cccc} \pi_j : & \mathsf{R}^n & \to & \mathsf{R}^j \\ & (x_1, \dots, x_n) & \mapsto & (x_1, \dots, x_j); \end{array}$$

if $j = 0$, we adapt this definition by letting π_0 be the constant map $\mathsf{R}^i \to \mathsf{R}^0$. Finally, since this is the projection we use most, we simply write $\pi = \pi_{n-1}$ for the projection $\mathsf{R}^n \to \mathsf{R}^{n-1}$.

If V is a subset of R^n, for β in $\pi(V)$, we let V_β be the set $V \cap \pi^{-1}(\beta)$. Hence, if β has coordinates $(\beta_1, \dots, \beta_{n-1})$, the points in V_β have the form $(\beta_1, \dots, \beta_{n-1}, a)$, for some values a in R. In all that follows, a finite set is by convention non-empty.

3.1 Equiprojectable sets

Let V be a finite set in R^n. *Equiprojectability* is a property of V that describes combinatorial properties in the successive projections of V; this notion was introduced in [2]. For $n = 0$, we say that the unique non-empty subset of R^0 is a single point which is equiprojectable. Then, for $n > 0$, $V \subset \mathsf{R}^n$ is equiprojectable if the following holds:

- the projection $\pi(V)$ is equiprojectable in R^{n-1}, and
- there exists an integer d_n such that for all β in $\pi(V)$, V_β has cardinality d_n.

The vector $\mathbf{d} = (d_1, \dots, d_n)$ is called the *multi-degree* of V.

Remark that for $n = 1$, any finite set $V \subset \mathsf{R}$ is equiprojectable. One easily sees that if V is equiprojectable, its cardinality equals $\delta_{\mathbf{d}} = d_1 \cdots d_n$; more generally, $\pi_i(V) \subset \mathsf{R}^i$ is equiprojectable of cardinality $d_1 \cdots d_i$.

Examples. Assume that $n = 2$. Figure 3.1 gives an example of an equiprojectable set of multi-degree $\mathbf{d} = (3, 3)$; here, the projection π takes a point in the plane and projects it on the X_1-axis. The cardinality of V is $3 \times 3 = 9$; for all β in $\pi(V)$ there are exactly 3 distinct points above β in V. The set V of Figure 3.2 is not equiprojectable because of unequal cardinality in X_2.

For $n = 3$, a simple example of an equiprojectable is the configuration of Figure 3.3, which forms the 8 vertices of a cube.

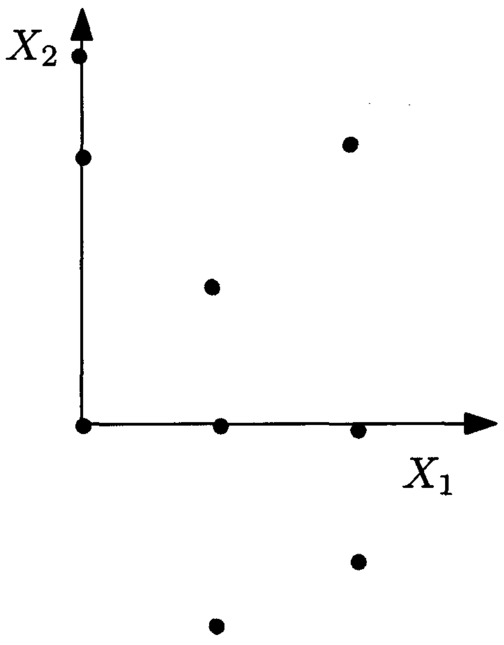


Figure 3.1: Equiprojectable set

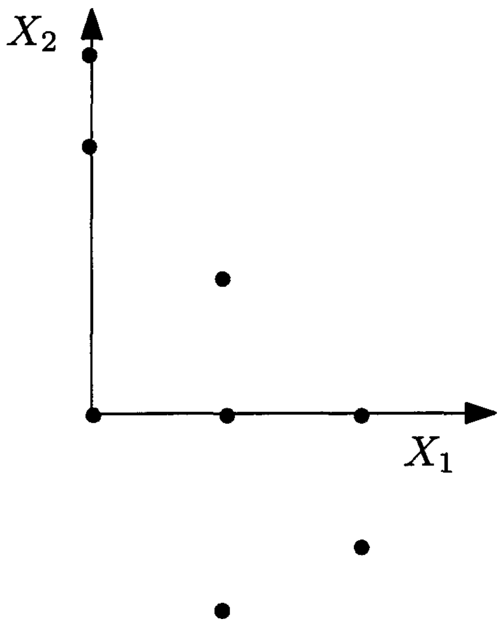


Figure 3.2: Non equiprojectable set

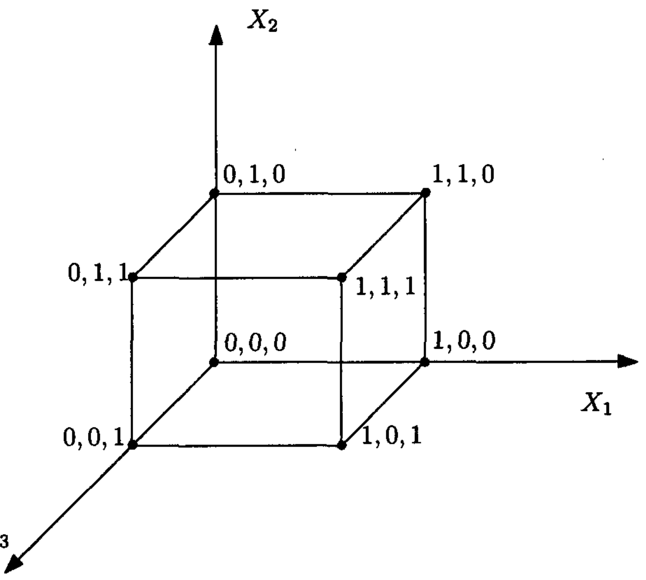


Figure 3.3: Equiprojectable set in dimension 3

We will use a slightly more precise notation for the sets V_β: if V is equiprojectable, then for all $\beta = (\beta_1, \ldots, \beta_{n-1}) \in \pi(V)$, there exist exactly d_n pairwise distinct values $v_\beta = [a_{\beta,1}, \ldots, a_{\beta,d_n}]$ in R such that

$$V_\beta = [(\beta_1, \ldots, \beta_{n-1}, a_{\beta,i}) \mid a_{\beta,i} \in v_\beta]$$

and thus

$$V = [(\beta_1, \ldots, \beta_{n-1}, a_{\beta,i}) \mid \beta = (\beta_1, \ldots, \beta_{n-1}) \in \pi(V), \ a_{\beta,i} \in v_\beta, \ 1 \le i \le d_n].$$

For instance, n-dimensional grids are special cases of equiprojectable sets, where v_β is independent of β.

3.2 Evaluation

In this section, we give an algorithm for the evaluation of a polynomial at an equiprojectable set V. We put constraints on this polynomial: we limit ourselves to polynomials whose degrees match the multi-degree of V.

Let V be an equiprojectable set of multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$, and let $M_\mathbf{d}$, $\delta_\mathbf{d}$

be as in Chapter 2. We denote by Eval_V the evaluation map

$$\begin{array}{rccc} \mathsf{Eval}_V : & \mathsf{Span}(M_{\mathbf{d}}) & \to & \mathsf{R}^{\delta_{\mathbf{d}}} \\ & F & \mapsto & [F(\alpha) \mid \alpha \in V]; \end{array}$$

this is thus the map that takes as input a polynomial F with $\deg(F, X_i) < d_i$ for all i, and computes all its values at V.

We let $\mathsf{C}_{\mathsf{Eval}}$ be a function such that for any V equiprojectable of multi-degree $\mathbf{d}$, the map Eval_V can be evaluated in $\mathsf{C}_{\mathsf{Eval}}(\mathbf{d})$ operations. In one variable, with $n = 1$ and $\mathbf{d} = (d_1)$, $\mathsf{C}_{\mathsf{Eval}}(\mathbf{d})$ simply describes the cost of evaluating a polynomial of degree less than d_1 at d_1 points of R, so we take $\mathsf{C}_{\mathsf{Eval}}(\mathbf{d}) = \mathsf{C}(d_1)$, using the notation of the previous chapter. More generally, we have the following time estimate; since $\mathsf{L}(\mathbf{d})$ is $O(\log(\delta_{\mathbf{d}})^3)$, this result shows that evaluation can be done in quasi-linear time $O\tilde{}(\delta_{\mathbf{d}})$.

Proposition 1. *One can take* $\mathsf{C}_{\mathsf{Eval}}(\mathbf{d}) \le \delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})$.

Proof. We will use a straightforward recursion over the variables $X_n, \dots, X_1$. Let $W = \pi(V)$, let $\mathbf{e} = (d_1, \dots, d_{n-1})$ be the multi-degree of W and let $A(X_1, \dots, X_n) \in \mathsf{Span}(M_{\mathbf{d}})$ be the polynomial to evaluate. We write

$$A = \sum_{i<d_n} A_i(X_1, \dots, X_{n-1})X_n^i,$$

with A_i in $\mathsf{Span}(M_{\mathbf{e}})$, and, for $\beta = (\beta_1, \dots, \beta_{n-1})$ in R^{n-1}, we define

$$A_\beta = \sum_{i<d_n} A_i(\beta)X_n^i \ \in \ \mathsf{R}[X_n].$$

Hence, for $\beta = (\beta_1, \dots, \beta_{n-1})$ in R^{n-1} and x in R, $A(\beta_1, \dots, \beta_{n-1}, x) = A_\beta(x)$. As a consequence, to evaluate A at V, we start by evaluating all A_i at all points $\beta \in W$. This gives all polynomials A_β, which we evaluate at the sets v_β.

The algorithm is given in Algorithm 2. From this, we deduce that we can take $\mathsf{C}_{\mathsf{Eval}}$ satisfying the recurrence

$$\mathsf{C}_{\mathsf{Eval}}(d_1, \dots, d_n) \le \mathsf{C}_{\mathsf{Eval}}(d_1, \dots, d_{n-1})\, d_n + \, d_1 \cdots d_{n-1}\mathsf{C}(d_n).$$

Unrolling the recurrence, this implies

$$\mathsf{C}_{\mathsf{Eval}}(d_1, \dots, d_n) \ \le \ \sum_{i\le n} \delta_{\mathbf{d}} \frac{\mathsf{C}(d_i)}{d_i},$$

Algorithm 2 Eval(A, V)

```
1: if $n = 0$ then
2:     return $[A]$
3: end if
4: $W \leftarrow \pi(V)$
5: for $i = 0, \ldots, d_n - 1$ do
6:     $A_i \leftarrow \mathsf{coeff}(A, X_n, i)$
7:     val$[i] \leftarrow \mathsf{Eval}(A_i, W)$        /* val$[i]$ has the form $[A_i(\beta) \mid \beta \in W]$ */
8: end for
9: for $\beta$ in $W$ do
10:    $A_\beta \leftarrow \sum_{i<d_n} A_i(\beta) X_n^i$
11: end for
12: return $[\mathsf{EvalUnivariate}(A_\beta, v_\beta) \mid \beta \in W]$
```

which proves the proposition. □

We conclude this section by showing the trace of the evaluation algorithm on a simple example. Let A be a polynomial in $\mathsf{R}[X_1, X_2]$ having degree 1 in both variables. We write this polynomial as

$$A = \xi_1 + \xi_2 X_1 + \xi_3 X_2 + \xi_4 X_1 X_2 = (\xi_1 + \xi_2 X_1) + (\xi_3 + \xi_4 X_1) X_2$$

for some $\xi_i \in \mathsf{R}$. Let V be a finite subset in R^2 (as the number of variable is 2) and let $\mathbf{d} = (2, 2)$ be the multi-degree vector. To fix notation, we assume that

$$V = \{(\alpha_1, \beta_1), (\alpha_1, \beta_2), (\alpha_2, \beta_3), (\alpha_2, \beta_4)\}.$$

The algorithm will evaluate $A_1 = \xi_1 + \xi_2 X_1$ and $A_2 = \xi_3 + \xi_4 X_1$ at (α_1, α_2) where A_1 and A_2 are the coefficients of A of X_2^0 and X_2^1 respectively. In this simple case, the evaluation of A_1 at (α_1, α_2) can be described by a *butterfly diagram*, similar to the butterfly of univariate FFT evaluation: on input ξ_1 and ξ_2, we compute $\xi_1 + \alpha_1 \xi_2$ and $\xi_1 + \alpha_2 \xi_2$.

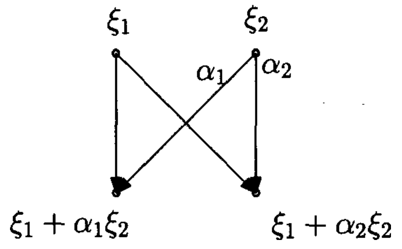


Figure 3.4: A single butterfly

The next task of the algorithm is to gather the evaluated values of A_1 and A_2 at $X_1 = \alpha_1$ to form the univariate polynomial $A_1(\alpha_1)+A_2(\alpha_1)X_2$ which will be evaluated at (β_1, β_2) for the variable X_2. The same thing will be repeated for the evaluated values of A_1, A_2 at $X_1 = \alpha_2$ and will be evaluated at (β_3, β_4) for the variable X_2. Again, each of these evaluations can be represented by a small butterfly diagram.

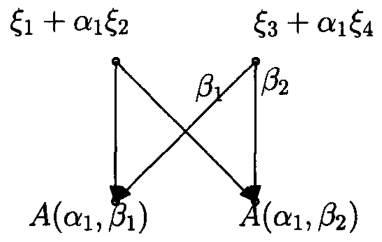


Figure 3.5: A single butterfly, second level

Putting the two levels of butterflies gives the computational flow for the 2-variable evaluation.

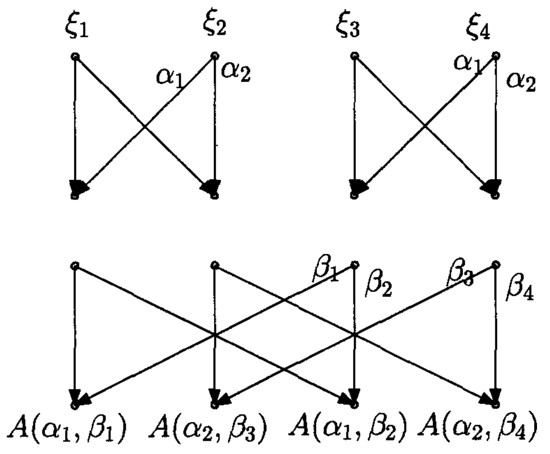


Figure 3.6: The butterfly structure in 2 variables

Let us add one variable, and consider an equiprojectable set V of multi-degree $\mathbf{d} = (2, 2, 2)$. Now, V can be described as

$$\begin{aligned} V &= \{(\alpha_1, \beta_1, \gamma_1), (\alpha_1, \beta_1, \gamma_2), (\alpha_1, \beta_2, \gamma_3), (\alpha_1, \beta_2, \gamma_4), \\ &\quad (\alpha_2, \beta_3, \gamma_5), (\alpha_2, \beta_3, \gamma_6), (\alpha_2, \beta_4, \gamma_7), (\alpha_2, \beta_4, \gamma_8)\} \end{aligned}$$

It is convenient to represent this set of points as a tree, giving first the two possible values α_1, α_2 for X_1, and so on; this is given in Figure 3.7.

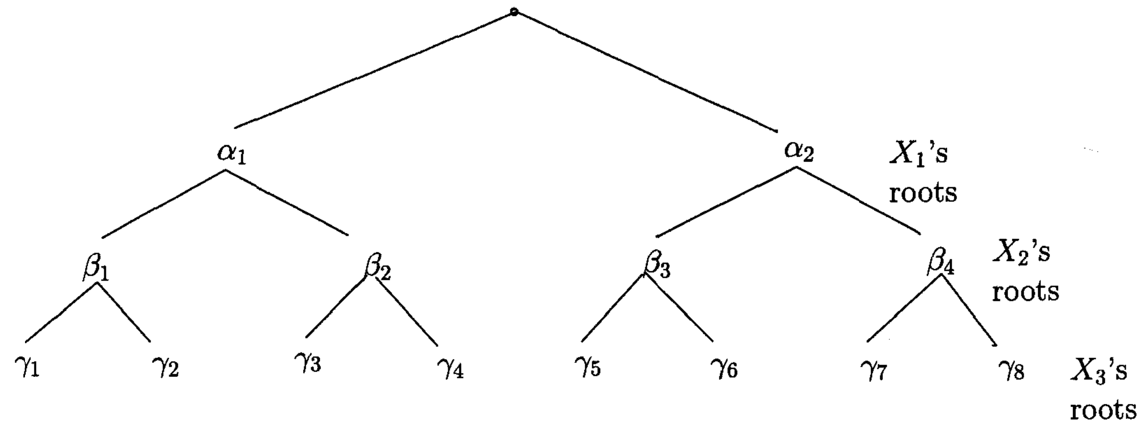


Figure 3.7: Tree structure of the roots in 3 variables

Let the equation to be evaluated be

$$(1 + X_1) + (1 + X_1)X_2 + ((1 + X_1) + (1 + X_1)X_2)X_3.$$

The monomials of this polynomial are $1, X_1, X_2, X_1X_2, X_3, X_3X_1, X_3X_2, X_3X_2X_1$. As we did in the 2-variable case, we can represent the evaluation process through a series of butterflies operations, in Figure 3.8.

3.3 Interpolation

Our next question is the inverse of the previous one: recovering the polynomial from its values. Using the same notation as above, the inverse of the evaluation map is interpolation at V:

$$\begin{array}{rccc} \mathsf{Interp}_V : & \mathsf{R}^{\delta_\mathbf{d}} & \to & \mathsf{Span}(M_\mathbf{d}) \\ & [F(\alpha) \mid \alpha \in V] & \mapsto & F. \end{array}$$

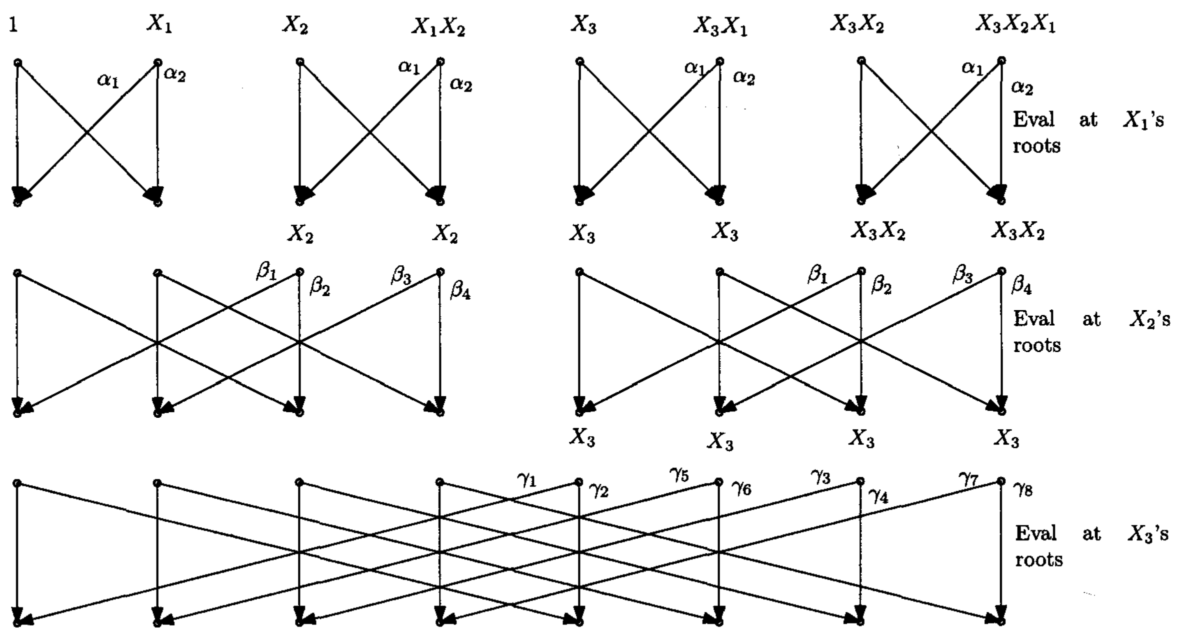


Figure 3.8: Butterfly of multivariate evaluation

For this map to be well-defined, we impose a condition on the points of V. Let $W = \pi(V) \in \mathsf{R}^{n-1}$. We say that V *supports interpolation* if the following holds:

- if $n > 1$, W supports interpolation, and
- for all β in W and all x, x' in v_β, $x - x'$ is invertible.

If the base ring is a field, any equiprojectable set of points V supports interpolation, since any difference $x - x'$ is invertible. However, in situations such as $\mathsf{R} = \mathbb{Z}/4\mathbb{Z}$, more care must be taken: the set $V = \{0, 2\} \subset \mathbb{Z}/4\mathbb{Z}$ does *not* support interpolation, since 2 is not invertible in $\mathbb{Z}/4\mathbb{Z}$.

We will see in the following proposition that if V supports interpolation, then the map Interp_V is well-defined. Moreover, we let $\mathsf{C}_{\mathsf{Interp}}$ be such that, for V equiprojectable of multi-degree $\mathbf{d}$, if V supports interpolation, then the map Interp_V can be evaluated in a cost $\mathsf{C}_{\mathsf{Interp}}(\mathbf{d})$ (including inversions) that is quasi-linear.

Proposition 2. *If V supports interpolation, the map* Interp_V *is well-defined. Besides, one can take* $\mathsf{C}_{\mathsf{Interp}}(\mathbf{d}) \le \delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})$.

Proof. If $n = 0$, we do nothing; otherwise, we let $W = \pi(V)$. The set of values to interpolate at V has the shape $[f_\alpha \mid \alpha \in V] \in \mathsf{R}^{\delta_{\mathbf{d}}}$, i.e. $\delta_{\mathbf{d}}$ numbers of values. We can thus rewrite it as $[f_\beta \mid \beta \in W]$, where each f_β is in R^{d_n}; i.e. above each $\beta \in W$, f_β is a set of d_n values that need to be interpolated.

Since V supports interpolation, for β in W, there exists a unique polynomial $A_\beta \in \mathsf{R}[X_n]$ of degree less than d_n, such that $\mathsf{Eval}(A_\beta, v_\beta) = f_\beta$. Applying the algorithm recursively on the coefficients of the polynomials A_β, we can find a polynomial A such that $A(\beta, X_n) = A_\beta(X_n)$ holds for all $\beta \in W$. Then, the polynomial A satisfies our constraints.

Algorithm 3 $\mathsf{Interp}(f, V)$

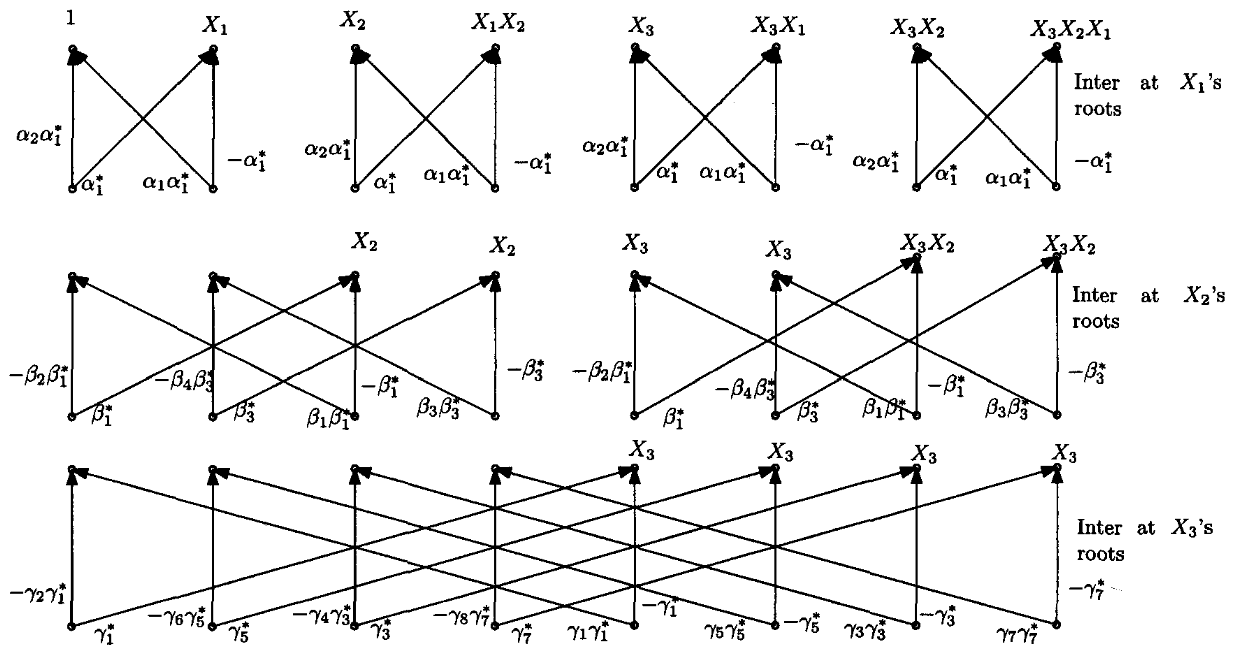
```
1: if $n = 0$ then
2:     return $[f]$
3: end if
4: $W \leftarrow \pi(V)$
5: for $\beta$ in $W$ do
6:     $A_\beta \leftarrow \mathsf{InterpUnivariate}(f_\beta, v_\beta)$
7: end for
8: for $i = 0, \ldots, d_n - 1$ do
9:     $c_i \leftarrow [\mathsf{coeff}(A_\beta, X_n, i) \mid \beta \in W]$
10:    $A_i \leftarrow \mathsf{Interp}(c_i, W)$
11: end for
12: return $\sum_{i<d_n} A_i X_n^i$
```

The algorithm is given in Algorithm 3; we use a subroutine called InterpUnivariate for univariate interpolation. As for evaluation, we deduce that we can take $\mathsf{C}_{\mathsf{Interp}}$ satisfying

$$\mathsf{C}_{\mathsf{Interp}}(d_1, \ldots, d_n) \leq \mathsf{C}_{\mathsf{Interp}}(d_1, \ldots, d_{n-1})\, d_n + d_1 \cdots d_{n-1} \mathsf{C}(d_n),$$

which gives our claim, as in the case of evaluation. □

Returning to our examples, in multi-degree $(2, 2, 2)$, interpolation can be described by means of butterflies as well, as showed in Figure 3.9. The roots are assumed to be the same as in the last section.



$$\gamma_1^* = \frac{1}{\gamma_1-\gamma_2} \qquad \beta_1^* = \frac{1}{\beta_1-\beta_2} \qquad \alpha_1^* = \frac{1}{\alpha_1-\alpha_2}$$

$$\gamma_3^* = \frac{1}{\gamma_3-\gamma_4} \qquad \beta_3^* = \frac{1}{\beta_3-\beta_4}$$

$$\gamma_5^* = \frac{1}{\gamma_5-\gamma_6}$$

$$\gamma_7^* = \frac{1}{\gamma_7-\gamma_8}$$

Figure 3.9: The butterflies of multivariate interpolation

3.4 Associated triangular set

Next, we associate to an equiprojectable set $V \subset \mathsf{R}^n$ of multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$ a triangular set $\mathbf{T} = (T_1, \ldots, T_n)$ of the same multi-degree, which vanishes on V.

If the base ring R was a field, it would suffice to take for $\mathbf{T}$ the lexicographic Gröbner basis of the vanishing ideal of V, for the order $X_n > \cdots > X_1$. However, since we do not assume that R is field, we cannot rely on the theory of Gröbner bases, so we have to redevelop the corresponding results.

As soon as V supports interpolation, the existence of $\mathbf{T}$ is guaranteed (and is established in the proof of the next proposition). Uniqueness holds as well: if $(T_1, \ldots, T_n)$ and $(T'_1, \ldots, T'_n)$ both vanish on V and have multi-degree $\mathbf{d}$, then for all i, $T_i - T'_i$ vanishes at V as well and is in $\mathsf{Span}(M_{\mathbf{d}})$; hence, it is zero. We call $\mathbf{T}$ the *associated triangular set* of V.

Proposition 3. *Given an equiprojectable set V of multi-degree $\mathbf{d}$ that supports interpolation, one can construct the associated triangular set $\mathbf{T}$ in time $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d}))$.*

Proof. We proceed inductively, and suppose that we already have computed $T_1, \dots, T_{n-1}$ as the associated triangular set of $W = \pi(V)$. We will write $\mathbf{d} = (d_1, \dots, d_n)$ and $\mathbf{e} = (d_1, \dots, d_{n-1})$.

For β in W, let T_β be the polynomial $\prod_{a \in v_\beta}(X_n - a) \in \mathsf{R}[X_n]$. For $j < d_n$, let further $T_{j,n}$ be the polynomial in $\mathsf{Span}(M_{\mathbf{e}})$ that interpolates the jth coefficient of the polynomials T_β at W; for $j = d_n$, we take $T_{d_n,n} = 1$. We then write $T_n = \sum_{j \le d_n} T_{j,n} X_n^j$: this polynomial is in $\mathsf{R}[X_1, \dots, X_n]$, monic of degree d_n in X_n, has degree less than d_i in X_i, for $i < n$, and vanishes on V. Thus, the polynomials $\mathbf{T} = (T_1, \dots, T_n)$ form the triangular set we are looking for.

The algorithm is in Algorithm 4. We use a function PolyFromRoots to compute the polynomials T_β. This is done using the sub-product tree algorithm described in Section 2.2.

Algorithm 4 AssociatedTriangularSet(f, V)

```
1: if $n = 0$ then
2:     return []
3: end if
4: $W \leftarrow \pi(V, n)$
5: $(T_1, \dots, T_{n-1}) \leftarrow$ AssociatedTriangularSet$(W, n-1)$
6: for $\beta$ in $W$ do
7:     $T_\beta \leftarrow$ PolyFromRoots$(v_\beta)$
8: end for
9: for $i = 0, \dots, d_n - 1$ do
10:     $T_{i,n} \leftarrow$ Interp([coeff$(T_\beta, X_n, i)$ | $\beta \in W$], $W$)
11: end for
12: return $\sum_{j<d_n} T_{j,n} X_n^j + X_n^{d_n}$
```

For a given β in W, the function PolyFromRoots computes T_β in $\mathsf{C}(d_n)$ operations, by definition of C; this implies that given $T_1, \dots, T_{n-1}$, one can construct T_n using $d_1 \cdots d_{n-1}\mathsf{C}(d_n) + \mathsf{C}_{\mathsf{Interp}}(d_1, \cdots, d_{n-1})d_n$ operations. The total cost for constructing all T_i is thus at most

$$\sum_{i \le n} d_1 \cdots d_{i-1}\mathsf{C}(d_i) + \sum_{i \le n} \mathsf{C}_{\mathsf{Interp}}(d_1, \cdots, d_{i-1})d_i.$$

Using the trivial bound $d_1 \cdots d_i \le \delta_{\mathbf{d}}$ for the left-hand term, and the bound given in

Proposition 2 for the right-hand one, we get the upper bounds

$$\delta_{\mathbf{d}} \sum_{i \le n} \frac{\mathsf{C}(d_i)}{d_i} + \sum_{i \le n} d_1 \cdots d_i \sum_{j \le i-1} \frac{\mathsf{C}(d_j)}{d_j} \le \delta_{\mathbf{d}} \sum_{i \le n} \frac{\mathsf{C}(d_i)}{d_i} + \sum_{i \le n} d_1 \cdots d_i \sum_{j \le n} \frac{\mathsf{C}(d_j)}{d_j}.$$

Using the upper bound $\sum_{i \le n} d_1 \cdots d_i \le 2\delta_{\mathbf{d}}$, we finally obtain the estimate $3\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})$. □

As an example, let us consider again the subset V in R^2 of multi-degree $\mathbf{d} = (2,2)$ given by

$$V = \{(\alpha_1, \beta_1), (\alpha_1, \beta_2), (\alpha_2, \beta_3), (\alpha_2, \beta_4)\}.$$

We construct the associated triangular set (T_1, T_2) in the two variables X_1 and X_2. The constraints that need to be satisfied by the triangular set are

- $\deg(T_1, X_1) = 2$
- $\deg(T_2, X_2) = 2$ and $\deg(T_2, X_1) < 2$.

The projection of $\pi(V)$ from R^2 to R is (α_1, α_2); this gives us the two roots of T_1. So, we write

$$T_1 = (X_1 - \alpha_1)(X_1 - \alpha_2) = X_1^2 - (\alpha_1 + \alpha_2)X_1 + \alpha_1\alpha_2.$$

Above $\alpha_1 \in \pi(V)$, we have two distinct values in R, which are (β_1, β_2); above for $\alpha_2 \in \pi(V)$, we have another two distinct values in R which are (β_3, β_4). So, we have

$$T_2(\alpha_1, X_2) = (X_2 - \beta_1)(X_2 - \beta_2) = X_2^2 - (\beta_1 + \beta_2)X_2 + \beta_1\beta_2$$
$$T_2(\alpha_2, X_2) = (X_2 - \beta_3)(X_2 - \beta_4) = X_2^2 - (\beta_3 + \beta_4)X_2 + \beta_3\beta_4$$

We are now able to construct $T_2(X_1, X_2)$ by extracting the coefficients of X_2^0 and X_2^1 and performing interpolation at (α_1, α_2) in both of them.

Remark that we could also choose V as $\{(0,0),(0,1),(1,0),(1,1)\}$. In this case, one can directly obtain $T_1 = X_1^2 - X_1$ and $T_2 = X_2^2 - X_2$.

3.5 Multiplication

Using our evaluation and interpolation algorithms, it becomes immediate to perform multiplication modulo a triangular set $\mathbf{T}$ associated to an equiprojectable set. In terms of complexity, the following result is close to optimal: it shows that in this case, the multiplication can be done in quasi-linear time.

Proposition 4. *Let $V \subset \mathsf{R}^n$ be an equiprojectable set of multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$ that supports interpolation, and let $\mathbf{T}$ be the associated triangular set. Then one can perform multiplication modulo $\langle \mathbf{T} \rangle$ in time $O(\delta_{\mathbf{d}} \mathsf{L}(\mathbf{d}))$.*

Proof. The algorithm is the same as in [23, Section 2.2], except that we now use the more general evaluation and interpolation algorithms presented here. Let A and B be reduced modulo $\langle \mathbf{T} \rangle$, and let $C = AB \bmod \langle \mathbf{T} \rangle$. Then for all α in V, $C(\alpha) = A(\alpha)B(\alpha)$. Since C is reduced modulo $\langle \mathbf{T} \rangle$, it suffices to interpolate the values $A(\alpha)B(\alpha)$ to obtain C. The cost is thus that of two evaluations, one interpolation, and of all pairwise pairwise products; the bounds of Propositions 1 and 2 conclude the proof. □

Algorithm 5 $\textsc{Mul}(A, B, V)$

```
1: $\mathsf{Val}_A \leftarrow \mathsf{Eval}(A, V)$
2: $\mathsf{Val}_B \leftarrow \mathsf{Eval}(B, V)$
3: $\mathsf{Val}_C \leftarrow [\ \mathsf{Val}_A(\alpha)\mathsf{Val}_B(\alpha) \mid \alpha \in V\ ]$
4: return $\mathsf{Interp}(\mathsf{Val}_C, V)$
```

We conclude by an example of multiplication by evaluation and interpolation. Let V be the equiprojectable set $V = \{(0,0), (0,1), (1,0), (1,1)\}$, and let the polynomials to multiply be $A = (1 + X_1) + (1 + X_1)X_2$ and $B = A = (1 + X_1) + (1 + X_1)X_2$.

The target is to multiply these two polynomials by using Algorithm 5. As both polynomials are same, their values at V will be the same, in this case $\{1, 2, 2, 4\}$. The term-wise multiplication gives us $\{1, 4, 4, 16\}$. After interpolation at V, we obtain the polynomial $(1 + 3X_1) + (3 + 9X_1)X_2$.

We saw in the previous section that the associated triangular set of V is given by $T_1 = X_1^2 - X_1$ and $T_2 = X_2^2 - X_2$.. The product of A and B before any reduction is

$$1 + 2X_1 + 2X_2 + 4X_2X_1 + X_1^2 + 2X_2X_1^1 + X_2^2 + 2X_2^2X_1 + X_2^2X_1^2.$$

Reduction with respect to $\langle T_1, T_2 \rangle$ gives us $1 + 3X_1 + 3X_2 + 9X_2X_1$, which is the same as above.

Chapter 4

Homotopy Techniques for Multiplication

4.1 Introduction

In this chapter, we present our main multiplication algorithm. This extends the approach of Li, Moreno Maza and Schost in [23, Section 2.2].

Let $\mathbf{T}$ be a triangular set in $\mathsf{R}[X_1, \ldots, X_n]$. We saw in the previous chapter that if $\mathbf{T}$ has all its roots in R, and if the set of roots V of $\mathbf{T}$ supports interpolation, then multiplication modulo $\langle \mathbf{T} \rangle$ can be done in quasi-linear time. In this chapter, we extend this approach to an arbitrary $\mathbf{T}$ by setting up an homotopy between $\mathbf{T}$ and a new, more convenient, triangular set $\mathbf{U}$. This idea was introduced in [23, Section 2.2], which dealt with the case where T_i is in $\mathsf{R}[X_i]$ for all i.

In general, the idea of homotopy consists in connecting the situation one wants to deal with, here multiplying modulo $\langle \mathbf{T} \rangle$, to a situation that is easier by design.

Here is how we put this idea to practice for the multiplication problem. Let $\mathbf{d}$ be the multi-degree of $\mathbf{T}$ and assume that there exists an equiprojectable set V in R^n which supports interpolation and has multi-degree $\mathbf{d}$. Let $\mathbf{U}$ be the triangular set associated to V and let η be a new variable. We then define the set $\mathbf{S}$ in $\mathsf{R}[[\eta]][X_1, \ldots, X_n]$ by

$$S_i = \eta T_i + (1 - \eta) U_i, \quad 1 \leq i \leq n.$$

Since $\mathbf{U}$ and $\mathbf{T}$ have the same multi-degree $\mathbf{d}$, this set $\mathbf{S}$ is triangular, with multi-degree $\mathbf{d}$.

We will prove (in Section 4.3) that $\mathbf{S}$ has all its roots in $\mathsf{R}[[\eta]]$. Thus, we can use evaluation-interpolation techniques to do multiplication modulo $\langle \mathbf{S} \rangle$; this will in turn be used to perform multiplication modulo $\langle \mathbf{T} \rangle$.

The algorithm involves computing with power series; the quantity that will determine the cost of the algorithm will be the required precision in η. For $\mathbf{e} = (e_1, \dots, e_n)$ in $\mathbb{N}^n$, we define

$$H_0(e_1, \dots, e_n) = \deg(X_1^{e_1} \cdots X_n^{e_n} \bmod \langle \mathbf{S} \rangle, \eta)$$

,i.e. the function $H_0(e_1, \dots, e_n)$ gives the degree in η after reducing the monomial $X_1^{e_1} \cdots X_n^{e_n}$ w.r.t. $\langle \mathbf{S} \rangle$, and

$$H(e_1, \dots, e_n) = \max_{e'_1 \le e_1, \ \dots, \ e'_n \le e_n} H_0(e'_1, \dots, e'_n).$$

Let us then define $r = H(2d_1 - 2, \dots, 2d_n - 2)$. Section 4.4 shows that multiplication modulo $\langle \mathbf{T} \rangle$ can be performed in time $O\tilde{~}(\delta_{\mathbf{d}} r)$, so the lower r the better. We postpone to Chapter 5 the study of r, which is the technical core of this thesis.

First of all, though, we describe the homotopy techniques for multiplication by a simple example.

4.2 A worked example

Suppose that the base ring is $\mathsf{R} = \mathbb{Q}$, and let the triangular set $\mathbf{T} = (T_1, T_2)$ be composed of

$$\begin{aligned} T_1 &= X_1^2 + 3X_1 + 3 \\ T_2 &= X_2^2 + X_1X_2 + 10. \end{aligned}$$

We consider the polynomials

$$\begin{aligned} A &= 1 + X_1 + X_2 + X_1X_2 \\ B &= 2 + X_1 - X_2 - X_1X_2; \end{aligned}$$

these two polynomials reduced with respect to $\mathbf{T}$. The roots of $\mathbf{T}$ are not in $\mathbb{Q}$, so we cannot apply the algorithm of the previous chapter to multiply A and B modulo $\langle \mathbf{T} \rangle$. To solve this problem, let

$$V = \{(0,0), \ (0,1), \ (1,0), \ (1,1)\}$$

be the equiprojectable set described in Section 3.1. We saw in the previous chapter that the associated triangular set $\mathbf{U}$ of V is

$$\begin{aligned} U_1 &= X_1^2 - X_1 \\ U_2 &= X_2^2 - X_2. \end{aligned}$$

Then we can define the polynomials $\mathbf{S}$ in $\mathbb{Q}[[\eta]][X_1 X_2]$ by

$$\begin{aligned} S_1 &= \eta T_1 + (1-\eta)U_1 = X_1^2 + (4\eta - 1)X_1 + 3\eta \\ S_2 &= \eta T_2 + (1-\eta)U_2 = X_2^2 + \eta X_1 X_2 + (\eta - 1)X_2 + 10\eta. \end{aligned}$$

This set $\mathbf{S}$ is triangular of multi-degree $\mathbf{d} = (2,2)$. Because $\mathbf{S}(\eta = 0, X_1, X_2)$ equals $\mathbf{U}$, and because the roots of $\mathbf{U}$ are known (they form the set V), the roots of $\mathbf{S}$ are power series in η. For simplicity, we will consider truncations of the roots of S_1 and S_2 modulo η^4; we will see in the next chapter how to figure out that this would be sufficient. The two roots of S_1 up to that precision are

$$\begin{aligned} x_{1,1} &= 3\eta + 21\eta^2 + 210\eta^3 \bmod \eta^4; \\ x_{1,2} &= 1 - 7\eta - 21\eta^2 - 210\eta^3 \bmod \eta^4. \end{aligned}$$

We can obtain these roots by applying Algorithm 1 of Chapter 2. Evaluation of S_2 at the roots $x_{1,1}$ and $x_{1,2}$ produces

$$S_{2,1} = S_2(x_{1,1}, X_2) = X_2^2 + (-1 + \eta + 3\eta^2 + 21\eta^3)X_2 + 10\eta \mod \eta^4$$

and

$$S_{2,2} = S_2(x_{1,2}, X_2) = X_2^2 + (-1 + 2\eta - 7\eta^2 - 21\eta^3)X_2 + 10\eta \mod \eta^4.$$

Applying Algorithm 1 to both $S_{2,1}$ gives the roots

$$\begin{aligned} x_{2,1,1} &= 10\eta + 110\eta^2 + 2340\eta^3 \bmod \eta^4 \\ x_{2,1,2} &= 1 - 11\eta - 113\eta^2 - 2361\eta^3 \bmod \eta^4, \end{aligned}$$

and doing the same with $S_{2,2}$ gives the roots

$$\begin{aligned} x_{2,2,1} &= 10\eta + 120\eta^2 + 2570\eta^3 \bmod \eta^4 \\ x_{2,2,2} &= 1 - 12\eta - 113\eta^2 - 2549\eta^3 \bmod \eta^4 \end{aligned}$$

The tree structure of these roots is given in Figure 4.1.

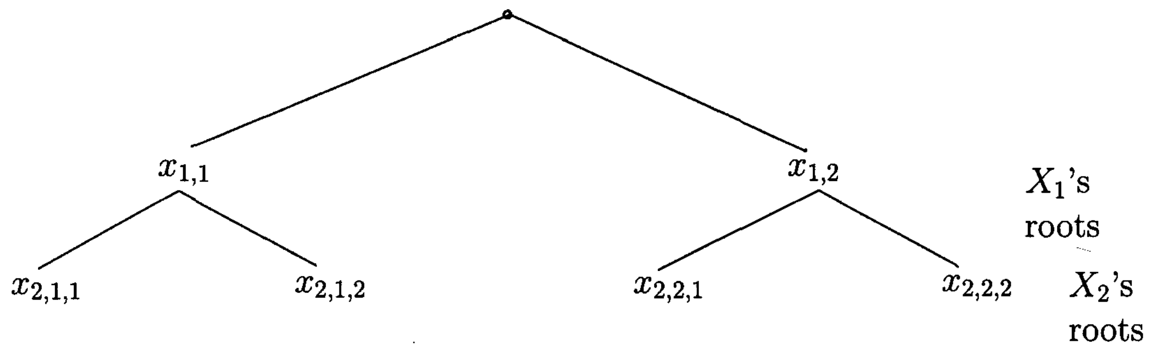


Figure 4.1: Tree structure of four roots

Now, we can evaluate both polynomials A and B at the above roots, still modulo η^4. For A, we obtain

$$\begin{aligned} a_{1,1} &= 1 + 13\eta + 161\eta^2 + 3090\eta^3 \quad \bmod \eta^4; \\ a_{1,2} &= 2 - 5\eta - 104\eta^2 - 2511\eta^3 \quad \bmod \eta^4; \\ a_{2,1} &= 2 + 13\eta + 149\eta^2 + 3880\eta^3 \quad \bmod \eta^4; \\ a_{2,2} &= 4 - 38\eta - 184\eta^2 - 4475\eta^3 \quad \bmod \eta^4 \end{aligned}$$

and for B, we have

$$\begin{aligned} b_{1,1} &= 2 - 7\eta - 119\eta^2 - 2670\eta^3 \quad \bmod \eta^4; \\ b_{1,2} &= 1 + 11\eta + 146\eta^2 + 2931\eta^3 \quad \bmod \eta^4; \\ b_{2,1} &= 3 - 27\eta - 191\eta^2 - 4300\eta^3 \quad \bmod \eta^4; \\ b_{2,2} &= 1 + 24\eta + 142\eta^2 + 4055\eta^3 \quad \bmod \eta^4. \end{aligned}$$

The term-wise multiplication gives us the values of $C = AB$ at the roots:

$$\begin{aligned}
c_{1,1} &= 2 + 19\eta + 112\eta^2 + 836\eta^3 \mod \eta^4; \\
c_{1,2} &= 2 + 17\eta + 133\eta^2 + 1477\eta^3 \mod \eta^4; \\
c_{2,1} &= 6 - 15\eta - 286\eta^2 - 3466\eta^3 \mod \eta^4; \\
c_{2,2} &= 4 + 58\eta - 528\eta^2 + 1933\eta^3 \mod \eta^4.
\end{aligned}$$

Interpolation of these values at the roots produces

$$\begin{aligned}
C &= 2 + 7\eta - 30\eta^2 + (4 + 26\eta - 40\eta^2)X_1 \\
&\quad +(4\eta - 12\eta^2 + 12\eta^3)X_2 + (-2 + 11\eta - 23\eta^2 + 16\eta^3)X_1X_2 \mod \eta^4.
\end{aligned}$$

The substitution of $\eta = 1$ in C gives us the final result

$$-21 - 10X_1 + 4X_2 + 2X_1X_2.$$

To verify this result in the naive way, the product of A and B is

$$2 + 3X_1 + X_2 + X_1^2 + X_2X_1 - X_2^2 - 2X_2^2X_1 - X_2^2X_1^2.$$

After reducing this with respect to $\langle \mathbf{T} \rangle$, we obtain $-21 - 10X_1 + 4X_2 + 2X_1X_2$, which is same as before. The remainder of this chapter is devoted to study the algorithmic aspects of this approach, except for the estimate on the precision in η, which is left to the next chapter.

4.3 Computing the roots of S

From now on, we continue with the notation given in the introduction of this chapter. We show here that $\mathbf{S}$ has all its roots in $\mathsf{R}[[\eta]]$.

First, we need some notation. Given positive integers k, ℓ and a subset $A \subset \mathsf{R}[[\eta]]^k$, $A \bmod \eta^\ell$ denotes the set $[a \bmod \eta^\ell \mid a \in A]$. Besides, we usually denote objects over $\mathsf{R}[[\eta]]$ with a * superscript, to distinguish them from their counterparts over R. Finally, we extend the notation π to denote the following projection

$$\begin{array}{cccc}
\pi : & \mathsf{R}[[\eta]]^n & \to & \mathsf{R}[[\eta]]^{n-1} \\
& (\alpha_1^\star, \ldots, \alpha_n^\star) & \mapsto & (\alpha_1^\star, \ldots, \alpha_{n-1}^\star).
\end{array}$$

Recall in what follows that $V \subset \mathsf{R}^n$ is equiprojectable of multi-degree $\mathbf{d}$, that its associated triangular set is $\mathbf{U}$, and that $\mathbf{S} = \eta\mathbf{T} + (1-\eta)\mathbf{U}$.

Proposition 5. *There exists a set $V^\star$ in $\mathsf{R}[[\eta]]^n$ such that the following holds:*

- $V = V^\star \bmod \eta$;
- *$V^\star$ is equiprojectable of multi-degree* $\mathbf{d}$;
- *$V^\star$ supports interpolation;*
- $\mathbf{S}$ *is the triangular set associated to $V^\star$.*

Proof. We prove by induction that for $i = 1, \ldots, n$, there exists a unique equiprojectable set $V_i^\star$ such that $\pi_i(V) = V_i^\star \bmod \eta$, $V_i^\star$ is equiprojectable of multi-degree $(d_1, \ldots, d_i)$, supports interpolation, and has $(S_1, \ldots, S_i)$ for associated triangular set.

For $i = 1$, the constraints imply that the points of $V_1^\star$ are the roots of S_1. Let $a_1, \ldots, a_{d_1}$ be the points in $\pi_1(V)$. Since V supports interpolation by assumption on V, all differences $a_i - a_j$ are units in R, for $i \neq j$. Remark now that

$$\frac{\partial S_1}{\partial X_1}(0, X_1) = \frac{\partial U_1}{\partial X_1}(X_1)$$

is given by

$$\sum_{i \le d_1} \prod_{j \le d_1, j \neq i} (X_1 - a_j),$$

so its value at a_i is

$$\prod_{j \le d_1, j \neq i} (a_i - a_j);$$

by what we said before, this is a product of invertible elements, so it is invertible. We can thus apply by the Newton iteration presented in Chapter 2, which shows the existence of $A_1, \ldots, A_{d_1}$ that are lifts of $a_1, \ldots, a_{d_1}$ and roots of S_1. Besides, for $i \neq j$, $A_i - A_j \bmod \eta = a_i - a_j$ is a unit, so using Newton iteration for inverses, we deduce that $A_i - A_j$ is a unit: this shows that $V_1^\star$ supports interpolation.

Supposing that the result was established at index i, we prove it at index $i+1$ in exactly the same manner as for $i = 1$. For any $\beta^\star = (\beta_1^\star, \ldots, \beta_{i-1}^\star)$ in $V_{i-1}^\star$, we compute the set $V_{\beta^\star}^\star$ by lifting the roots of $S_1(\eta, \beta_1^\star, \ldots, \beta_{i-1}^\star, X_i)$. All the arguments given above for $i = 1$ apply in the same manner.

This establishes our claim by induction. Taking $i = n$ gives the result of the proposition. $\square$

We continue with complexity estimates: we prove that the roots of $\mathbf{S}$ can be computed in quasi-linear time. This is done by using the fast algorithms seen before to implement the construction of the previous proposition.

Proposition 6. *Given* $\mathbf{T}$, V *and* $\ell > 0$, *one can compute* $V^\star \bmod \eta^\ell$ *in time* $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})\mathsf{M}(\ell))$.

Proof. As before, we proceed inductively: we suppose that the projection $W^\star = \pi(V^\star)$ is known modulo η^ℓ, and show how to deduce $V^\star \bmod \eta^\ell$. To do so, we evaluate all coefficients of S_n at all points of $W^\star$ modulo η^ℓ. Then, for each $\beta^\star$ in $W^\star$, it suffices to use Algorithm 1 (**LiftRoots**) to lift the roots of $S_n(\beta^\star, X_n)$ at precision ℓ. The pseudo-code is in Algorithm 6; for simplicity, we write there $W^\star$ instead of $W^\star \bmod \eta^\ell$.

Algorithm 6 LIFTROOTSMULTIVARIATE($V, \mathbf{S}, \ell$)

```
1: $n = |\mathbf{S}|$
2: if $n = 0$ then
3:     return []
4: end if
5: $W^\star \leftarrow$ LIFTROOTSMULTIVARIATE($\pi(V), (S_1, \dots, S_{n-1}), \ell$)
6: for $i = 0, \dots, d_n - 1$ do
7:     $\mathrm{val}_i \leftarrow \mathsf{Eval}(\mathsf{coeff}(S_n, X_n, i), W^\star)$   /* all computations are done modulo $\eta^\ell$ */
8: end for
9: for $\beta^\star$ in $W^\star$ do
10:    $S_{\beta^\star} \leftarrow \sum_{i<d_n} \mathrm{val}_{i,\beta^\star} X_n^i + X_n^{d_n}$
11:    $v^\star_{\beta^\star} \leftarrow$ LIFTROOTS($S_{\beta^\star}, v_\beta, \ell$)
12: end for
13: return $[v^\star_{\beta^\star} \mid \beta^\star \in W^\star]$
```

Lemma 1 shows that we can lift the power series roots of a bivariate polynomial of degree d at precision ℓ in time $\mathsf{C}(d)\mathsf{M}(\ell)$. As a consequence, the overall cost $\mathsf{C}_{\mathsf{Lift}}$ of the lifting process satisfies

$$\begin{aligned}\mathsf{C}_{\mathsf{Lift}}(d_1, \dots, d_n, \ell) \quad &\leq \quad \mathsf{C}_{\mathsf{Lift}}(d_1, \dots, d_{n-1}, \ell) + \mathsf{C}_{\mathsf{Eval}}(d_1, \dots, d_{n-1}) d_n \mathsf{M}(\ell) \\ &\quad\quad + d_1 \cdots d_{n-1} \mathsf{C}(d_n)\mathsf{M}(\ell);\end{aligned}$$

the middle term gives the cost of evaluating the coefficients of S_n at $W^\star \bmod \eta^\ell$ (so we apply our evaluation algorithm with power series coefficients); the right-hand term gives the cost of lifting the roots of S_n. This gives

$$\mathsf{C}_{\mathsf{Lift}}(d_1, \dots, d_n, \ell) \leq \sum_{i \leq n} \mathsf{C}_{\mathsf{Eval}}(d_1, \dots, d_{i-1}) d_i \mathsf{M}(\ell) + \sum_{i \leq n} d_1 \cdots d_{i-1} \mathsf{C}(d_i)\mathsf{M}(\ell).$$

As in the proof of Proposition 3, one deduces that the overall sum is bounded by

$$3\delta_{\mathbf{d}} \sum_{i \le n} \frac{\mathsf{C}(d_i)}{d_i} \mathsf{M}(\ell) = 3\delta_{\mathbf{d}} \mathsf{L}(\mathbf{d}) \mathsf{M}(\ell),$$

which concludes the proof. □

4.4 The multiplication algorithm

We continue with the same notation as before. To multiply two polynomials $A, B \in \mathsf{Span}(M_{\mathbf{d}})$ modulo $\langle \mathbf{T} \rangle$, we may multiply them modulo $\langle \mathbf{S} \rangle$ over $\mathsf{R}[\eta]$ and let $\eta = 1$ in the result. Now, the results of the multiplication modulo $\langle \mathbf{S} \rangle$ over $\mathsf{R}[\eta]$ and over $\mathsf{R}[[\eta]]$ are the same. When working over $\mathsf{R}[[\eta]]$, we may use the evaluation-interpolation techniques from Chapter 3. Indeed, by Proposition 5, $\mathbf{S}$ is associated to a subset $V^\star$ of $\mathsf{R}[[\eta]]^n$ that supports interpolation.

Of course, when multiplying A and B modulo $\langle \mathbf{S} \rangle$ over $\mathsf{R}[[\eta]]$, we cannot compute with (infinite) power series, but rather with their truncations at a suitable order. On the one hand, this order should be larger than the largest degree of a coefficient of the multiplication of A and B modulo $\langle \mathbf{S} \rangle$ over $\mathsf{R}[\eta]$. On the other hand, this order will determine the cost of the multiplication algorithm, so it should be kept to a minimum. For $\mathbf{e} = (e_1, \dots, e_n)$ in $\mathbb{N}^n$, recall that we have defined in the introduction

$$H_0(e_1, \dots, e_n) = \deg(X_1^{e_1} \cdots X_n^{e_n} \bmod \langle \mathbf{S} \rangle, \eta)$$

and

$$H(e_1, \dots, e_n) = \max_{e'_1 \le e_1, \ \dots, \ e'_n \le e_n} H_0(e'_1, \dots, e'_n).$$

For the moment, we do not give any more details on these functions; we leave this to the next chapter. Taking this for granted, the following proposition relates the cost of our algorithm to the function H.

Proposition 7. *Given A, B, $\mathbf{T}$ and V, one can compute $AB \bmod \langle \mathbf{T} \rangle$ in time $O(\delta_{\mathbf{d}} \mathsf{L}(\mathbf{d}) \mathsf{M}(r)) \subset O\tilde{\ }(\delta_{\mathbf{d}} r)$, with $r = H(2d_1 - 2, \dots, 2d_n - 2)$.*

Proof. The algorithm is simple: we compute $\mathbf{U}$ and use it to obtain $V^\star$ at a high enough precision. In $\mathsf{R}[X_1, \dots, X_n]$, the product AB satisfies $\deg(AB, X_i) \le 2d_i - 2$ for all $i \le n$; since the multiplication algorithm does not perform any division by η, it suffices to apply it with coefficients modulo η^{r+1}, with $r = H(2d_1 - 2, \dots, 2d_n - 2)$.

The resulting algorithm is given in Algorithm 7; as before, we write $V^\star$ for simplicity, whereas we should write $V^\star \bmod \eta^{r+1}$.

Algorithm 7 $\textsc{Mul}(A, B, \mathbf{T}, V)$

```
1: $\mathbf{U} \leftarrow \textsc{AssociatedTriangularSet}(V, n)$
2: $\mathbf{S} \leftarrow \eta\mathbf{T} + (1-\eta)\mathbf{U}$
3: $V^\star \leftarrow \textsc{LiftRootsMultivariate}(V, \mathbf{S}, r+1)$
4: $C_\eta \leftarrow \textsc{Mul}(A, B, V^\star)$
5: return $C_\eta(1, X_1, \ldots, X_n)$
```

The computation of $\mathbf{U}$ takes time $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d}))$ by Proposition 3; that of $\mathbf{S}$ takes time $O(\delta_{\mathbf{d}})$. Computing $V^\star \bmod \eta^{r+1}$ takes time $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})\mathsf{M}(r))$ by Proposition 6. Finally, the modular multiplication takes time $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})\mathsf{M}(r))$ by Proposition 4; remark that this algorithm is run with coefficients modulo η^{r+1}, where all arithmetic operations take time $O(\mathsf{M}(r))$. Finally, specializing η at 1 takes time $O(\delta_{\mathbf{d}}r)$. Summing all these costs gives our result. $\square$

Chapter 5

Precision Analysis

In this chapter, we study the functions H_0 and H introduced in the previous chapter, and show how they are related to the monomial support of the polynomials in the triangular set $\mathbf{T}$. Before giving a general bound for these functions, we have a look at examples of two special cases, to motivate our general result.

5.1 First worked example

For this example, we assume that $n = 3$ and that the triangular set $\mathbf{S} = (S_1, S_2, S_3)$ is as follows:

$$\begin{aligned} S_1 &= X_1^3 + \eta X_1^2 + \eta X_1 + \eta \\ S_2 &= X_2^3 + \eta X_2^2 + \eta X_1^2 \\ S_3 &= X_3^3 + \eta X_3^2 + \eta X_2^2. \end{aligned}$$

Here, the multi-degree of $\mathbf{S}$ is $\mathbf{d} = (3, 3, 3)$. Remark that both S_2 and S_3 are sparse: several monomials are missing, such as X_1X_2. Since in our main result the actual coefficients are irrelevant, we take the simplest possible form: all are equal to η.

The degree of any polynomial reduced modulo $\langle \mathbf{S} \rangle$ is at most $(2, 2, 2)$ in respectively X_1, X_2 and X_3. Thus, if A and B are two polynomials reduced modulo $\langle \mathbf{S} \rangle$, the degree of $C = AB$ in X_1, X_2 and X_3 is at most $(4, 4, 4)$ before any reduction. Thus, we will consider here the reduction of the largest monomial in C, $m = X_1^4X_2^4X_3^4$.

Our question of estimating the functions H_0 and H essentially boils down to the following: what will be the degree in η of the normal form of $m = X_1^4X_2^4X_3^4$ modulo

$\langle S \rangle$? We are going to follow the steps of this reduction to show the growth of the degrees.

Step 1: reduction with respect to S_3. The first reduction step of m with respect to S_3 amounts to replacing X_3^4 by

$$-\eta X_3^3 - \eta X_2^2 X_3$$

and thus m by

$$-\eta X_1^4 X_2^4 X_3^3 - \eta X_1^4 X_2^6 X_3.$$

Since we are only interested in degree bounds in η, we can forget about the signs. The remaining monomials are $m_1 = \eta X_1^4 X_2^4 X_3^3$ and $m_2 = \eta X_1^4 X_2^6 X_3$. Thus, we will focus on the reductions of these two monomials. Reducing m_1 with respect to S_3 amounts to replacing X_3^3 by (up to sign)

$$\eta X_3^2 + \eta X_2^2;$$

this gives us the normal form of m_1 as (up to sign)

$$\eta^2 X_1^4 X_2^4 X_3^2 + \eta^2 X_1^4 X_2^6.$$

The monomial m_2 is already reduced with respect to S_3.

All remaining monomials are now reduced with respect to S_3. Among all the monomials, $m' = \eta^2 X_1^4 X_2^6$ is the "largest" one (it has the largest degree in X_2), so we focus on the reduction of this monomial.

Step 2: reduction with respect to S_2 and S_1. If we reduce $\eta^2 X_1^4 X_2^6$ with respect to S_2 once, we will get (up to sign)

$$\eta^3 X_1^4 X_2^5 + \eta^3 X_1^6 X_2^3.$$

The monomials we are interested in are $m_1' = \eta^3 X_1^4 X_2^5$ and $m_2' = \eta^3 X_1^6 X_2^3$. We can either choose m_1' then do the reduction, or take m_2' and then do the reduction.

- Let us take the monomial $m_1' = \eta^3 X_1^4 X_2^5$ and reduce it with respect to S_2. If we follow the same process we will reach $\eta^6 X_1^4 X_2^2 + \cdots + \eta^6 X_1^6$. Next we can reduce the monomial $\eta^6 X_1^6$, with respect to S_1; it will increases the degree in η by 4 and the total degree in η will be 10. This is the largest degree in η that can be obtained from m_1'.

- If we choose the monomial $m'_2 = \eta^3 X_1^6 X_2^3$ and then do the reduction with respect to S_2, we will obtain $\eta^4 X_1^6 X_2^2 + \eta^4 X_1^8$. Here, the last monomial has the highest degree in X_1. So, the reduction of $\eta^4 X_1^8$ with respect to S_1 will increase the degree in η by 6 and the total degree will be 10 as well.

The reductions with respect to S_2 amount to replace X_2^3 by (up to sign) $\eta X_2^2 + \eta X_1^2$. These reductions can be described by a tree structure: the initial monomial is the root, and the left and right children of a node correspond to the two monomials that appear after one reduction step. This enables us to dispense with writing the η factors: the degree in η of any monomial is the length of the path from the root to the corresponding node. Figure 5.1 gives such an example for the initial monomial $X_1^4 X_2^6$.

$$\begin{tikzcd}
& & & & X_1^4X_2^6 \arrow[dl] \arrow[dr] & & \\
& & & X_1^4X_2^5 \arrow[dl] \arrow[dr] & & X_1^6X_2^3 \arrow[dl] \arrow[dr] & \\
& & X_1^4X_2^4 \arrow[dl] \arrow[dr] & & X_1^6X_2^2 & & X_1^8 \\
& X_1^4X_2^3 \arrow[dl] \arrow[dr] & & X_1^6X_2 & & & \\
X_1^4X_2^2 & & X_1^6 & & & &
\end{tikzcd}$$

Figure 5.1: Tree structure of the reduction by S_2.

Thus, starting from the root, we can follow paths obtained by applying the following two moves:

- one step to the left, which reduces the degree in X_2 by 1;
- one step to the right, which reduces the degree in X_2 by 3 and increases the degree in X_1 by 2;

We perform reductions as long as the degree in X_2 remains at least 3; after the last reduction, this degree remains ≥ 0.

Each step in the path increases the degree in η by 1. Besides, when we reach the final node of the path, with degree r_1 in X_1, we can increase our degree by $r_1 - 2$, which corresponds to the reduction by S_1. With these rules, our goal is to find an upper-bound on the degree in η.

Let us introduce two variables f_1 and f_2, that respectively represent the number of steps to the left and to the right. After (f_1, f_2) steps, the degree in X_1 has become $4 + 2f_2$, and the degree in X_2 has become $6 - f_1 - 3f_2$. Hence, the constraint that this degree is at least 3 gives the inequality $6 - f_1 - 3f_2 \geq 3$, so that the end-points of our walk satisfy $0 \leq 6 - f_1 - 3f_2 \leq 2$.

Figure 5.2 summarizes these considerations, by depicting the plane of coordinates (f_1, f_2). The lower oblique line has equation $6 - f_1 - 3f_2 = 2$ and the higher one has equation $6 - f_1 - 3f_2 = 0$, so all end-points are between these lines. Also, at each point with integer coordinates (f_1, f_2), we give the value of the degrees in (X_1, X_2), that is, the values $4 + 2f_2$ and $6 - f_1 - 3f_2$; the origin $(f_1, f_2) = (0, 0)$ is in the lower corner on the left. The points written in **bold face** are those which can be reached by our reduction process; the two points with degrees $(4, 1)$ and $(4, 0)$ are unreachable, but will still matter for our purposes.

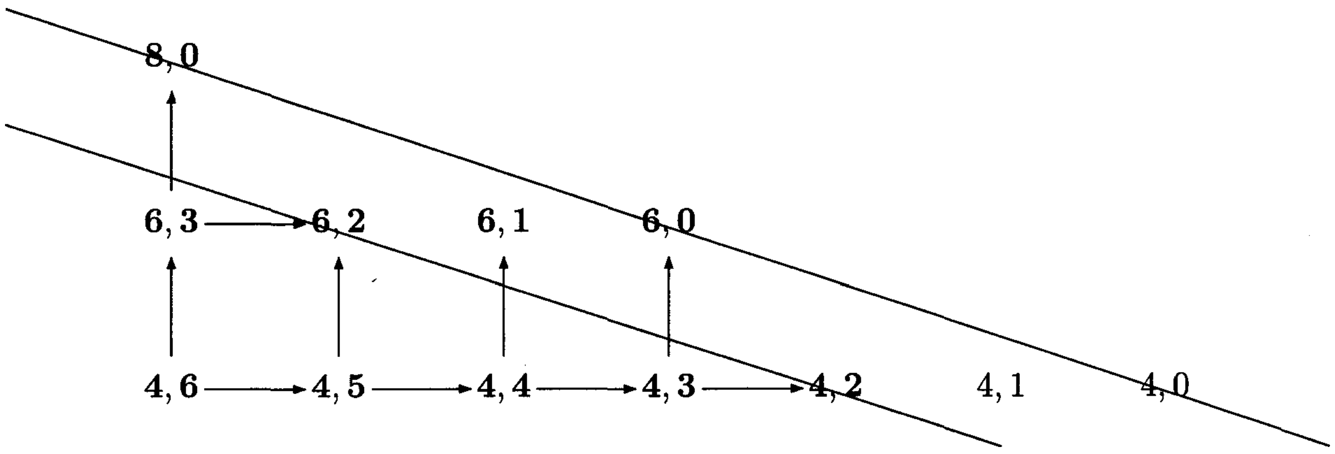


Figure 5.2: A view of the reduction tree in the coordinates (f_1, f_2)

Remember that this walk only describes the reduction process with respect to S_2; after that, we still need to reduce with respect to S_1. This is done by remarking, as we did before, that the reduction of a monomial $\eta^{f_0} X_1^{f_1} \cdots X_n^{f_n}$ with respect to S_1 increases the degree in η by $f_1 - 2$.

Thus, starting from $X_1^4 X_2^6$, the degree in η we obtain after reduction is bounded by the maximal value of the sum $(f_1 + f_2) + (2 + 2f_2)$ over all (f_1, f_2) that lie in between the two oblique lines: the term $(f_1 + f_2)$ gives the degree increase due to the reduction with respect to S_2, and the term $2 + 2f_2 = 4 + 2f_2 - 2$ gives the degree increase due to the reduction with respect to S_1 (remember that the degree in X_1 is $4+2f_2$).

In other words, to find a bound on the degree in η, it is enough to maximize $2+f_1+3f_2$ in the area contained between the oblique lines. To simplify the maximization, we do the following simplifications:

- we can forget the lower constraint $6-f_1-3f_2 \leq 2$, and maximize in the larger area defined by $f_1 \geq 0,\ f_2 \geq 0,\ 6-f_1-3f_2 \geq 0$;
- we can remove the condition that f_1 and f_2 are integers.

Indeed, both simplifications increase the size of the search space, so any upper-bound obtained after simplifications will be an upper bound for the initial problem as well.

After these simplifications, we are left to maximize a linear function over D; it is know that the maximum will be obtained at one of the end-vertices of D. The vertex $(f_1, f_2) = (0,0)$ is not a maximum (it is the minimum); the other vertices are $(f_1, f_2) = (0,2)$ and $(f_1, f_2) = (6,0)$, and the maximum of $2+f_1+3f_2$ is 8, obtained at both vertices.

This gives us an upper bound on the degree in η of the normal form of $X_1^4X_2^6$; since we actually started from $\eta^2X_1^4X_2^6$, the bound becomes $8+2=10$. The final value we obtain is indeed the *exact* maximum for our original problem; however, in most cases, we will only get upper bounds, but that will be sufficient for our purposes.

5.2 Second worked example

Our second example uses the following triangular set:

$$\begin{aligned} S_1 &= X_1^2+\eta X_1+\eta \\ S_2 &= X_2^2+\eta X_2+\eta X_1+\eta \\ S_3 &= X_3^2+\eta X_3+\eta X_1X_2. \end{aligned}$$

Observe that as in the previous case, not all monomials are present; for instance, S_2 does not have the monomial X_1X_2. The reduction of a monomial having degree ≥ 2 in X_3 by S_3 replaces the monomial X_3^2 by (up to sign) $\eta X_3+\eta X_1X_2$, so we create two monomials:

- in one direction, we reduce the degree in X_3 by 1;

- in the other direction, we reduce the degree in X_3 by 2, but increase the degrees in X_1 and X_2 by 1.

Again, we can display the reduction process using a tree structure. We give in Figure 5.3 an example of reduction, of the monomial $X_1^9X_2^9X_3^9$ by S_3; instead of writing the monomials, we just write the exponents that appear.

$$\begin{tikzcd}
 & & & & & & & & 9,9,9 \arrow[dl] \arrow[dr] & & & & \\
 & & & & & & & 9,9,8 \arrow[dl] \arrow[dr] & & 10,10,7 \arrow[dl] \arrow[dr] & & & \\
 & & & & & & 9,9,7 \arrow[dl] \arrow[dr] & & 10,10,6 \arrow[dl] \arrow[dr] & & 11,11,5 \arrow[dl] \arrow[dr] & & \\
 & & & & & 9,9,6 \arrow[dl] \arrow[dr] & & 10,10,5 \arrow[dl] \arrow[dr] & & 11,11,4 \arrow[dl] \arrow[dr] & & 12,12,3 \arrow[dl] \arrow[dr] & \\
 & & & & 9,9,5 \arrow[dl] \arrow[dr] & & 10,10,4 \arrow[dl] \arrow[dr] & & 11,11,3 \arrow[dl] \arrow[dr] & & 12,12,2 \arrow[dl] \arrow[dr] & & 13,13,1 \\
 & & & 9,9,4 \arrow[dl] \arrow[dr] & & 10,10,3 \arrow[dl] \arrow[dr] & & 11,11,2 \arrow[dl] \arrow[dr] & & 12,12,1 & & 13,13,0 & \\
 & & 9,9,3 \arrow[dl] \arrow[dr] & & 10,10,2 \arrow[dl] \arrow[dr] & & 11,11,1 & & 12,12,0 & & & & \\
 & 9,9,2 \arrow[dl] \arrow[dr] & & 10,10,1 & & 11,11,0 & & & & & & & \\
9,9,1 & & 10,10,0 & & & & & & & & & &
\end{tikzcd}$$

Figure 5.3: Tree structure of the reduction by S_3.

When we reach the leaves of the tree, we should proceed with the reduction with respect to S_1 and S_2.

As in the previous example, we introduce two variables f_1 and f_2 that give the number of steps to the left, resp. to the right, that we do. After (f_1, f_2) steps, the degree in X_1 becomes $9 + f_2$, that in X_2 is $9 + f_2$ and that in X_3 is $9 - f_1 - 2f_2$.

The area we can reach is thus entirely contained in the set D defined by the constraints $f_1 \geq 0$, $f_2 \geq 0$ and $0 \leq 9 - f_1 - 2f_2$; the end-points of our walk are contained in the set D' defined by $f_1 \geq 0$, $f_2 \geq 0$ and $0 \leq 9 - f_1 - 2f_2 \leq 1$. Here, f_1 and f_2 are supposed to be integers.

Suppose inductively that we know the function $H_0(e_1, e_2)$, that gives us the degree in η of $X_1^{e_1}X_2^{e_2}$ mod $\langle S_1, S_2\rangle$. Then, the degree in η of $X_1^9X_2^9X_3^9$ mod $\langle S_1, S_2, S_3\rangle$ is the maximum for (f_1, f_2) in D' of $f_1 + f_2 + H_0(9 + f_1, 9 + f_2)$.

Generalizing, if now we start from a monomial $m = X_1^{e_1}X_2^{e_2}X_3^{e_3}$, after (f_1, f_2) steps, the degree in X_1 becomes $e_1 + f_2$, that in X_2 is $e_2 + f_2$ and that in X_3 is $e_3 - f_1 - 2f_2$. The set D is given by the constraints $f_1 \geq 0,\ f_2 \geq 0$ and $0 \leq e_3 - f_1 - 2f_2$, and D' by the constraints $f_1 \geq 0,\ f_2 \geq 0$ and $0 \leq e_3 - f_1 - 2f_2 \leq 1$.

Besides, $H_0(e_1, e_2, e_3)$ is the degree in η of m mod $\langle S_1, S_2, S_3\rangle$; it is thus, as before, the maximum for (f_1, f_2) in D' of $f_1 + f_2 + H_0(e_1 + f_2, e_2 + f_2)$.

We are going to show that if we know an upper bound for $H_0(e_1, e_2)$ in two variables, we can deduce a bound for $H_0(e_1, e_2, e_3)$ in three variables. Suppose indeed that we know integers $h_1 \geq 1$ and $h_2 \geq 1$ such that $H_0(e_1, e_2) \leq h_1e_1 + h_2e_2$ holds for all e_1, e_2. Then, we deduce that $H_0(e_1, e_2, e_3)$ is bounded by the maximum for all f_1, f_2 in D' of

$$L(f_1, f_2) = f_1 + f_2 + h_1(e_1 + f_2) + h_2(e_2 + f_2).$$

Since D' is contained in D, we can take the maximum over D instead, and as before, we can allow f_1, f_2 to take real values: the upper bound obtained this way will still be valid (it may not be sharp, but obtaining sharp upper bounds is probably out of our reach).

Then, the maximum is obtained at one of the possible end-points of D, which have coordinates $(f_1, f_2) = (0, e_3/2)$ and $(f_1, f_2) = (e_3, 0)$. The values of the function $L(f_1, f_2)$ at these points are

$$L(0, e_3/2) = h_1e_1 + h_2e_2 + \frac{1 + h_1 + h_2}{2}e_3, \quad L(e_3, 0) = h_1e_1 + h_2e_2 + e_3;$$

the first one is the largest. This shows that for all (e_1, e_2, e_3), we have

$$H_0(e_1, e_2, e_3) \leq h_1e_1 + h_2e_2 + h_3e_3, \quad \text{with} \quad h_3 = \frac{1 + h_1 + h_2}{2}.$$

This mechanism enables us to establish degree bounds one variable after the other, by defining coefficients $h_1, h_2, \ldots$.

5.3 Precision analysis for the general case

We finally study the general case. As in the examples, we are going to see that the monomial structure of the polynomials in $\mathbf{S}$ affects the cost of the algorithm. For $i \leq n$ and $\nu = (\nu_1, \ldots, \nu_i)$ in $\mathbb{N}^i$, we will use the notation $\mathbf{X}_i^\nu = X_1^{\nu_1} \cdots X_i^{\nu_i}$ and we write the monomial expansion of S_i as

$$S_i = X_i^{d_i} + \sum_{\nu \in E_i} s_\nu \mathbf{X}_i^\nu, \tag{5.1}$$

where E_i is the set of exponents that appear in S_i, the exponents ν are in $\mathbb{N}^i$, and s_ν is linear in η. Let us further introduce the coefficients h_i defined by $h_0 = 0$, $h_1 = 1$ and for $i > 1$,

$$h_i = \max_{\nu \in E_i} \frac{h_1\nu_1 + \cdots + h_{i-1}\nu_{i-1} + 1}{d_i - \nu_i}. \tag{5.2}$$

One easily checks that all h_i are positive. The following proposition shows that through the coefficients h_i, E_i determines the cost of our algorithm.

Proposition 8. *The inequality*

$$H(e_1, \ldots, e_n) \le h_1 e_1 + \cdots + h_n e_n$$

holds for all $(e_1, \ldots, e_n) \in \mathbb{N}^n$.

Using Proposition 7, this proposition gives as an easy corollary the following statement, where we take $e_i = 2d_i - 2 \le 2d_i$; we continue using the previous notation $\mathbf{T}$ and V.

Corollary 1. *Given* A, B, $\mathbf{T}$ *and* V, *one can compute* $AB \bmod \langle \mathbf{T} \rangle$ *in time* $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})\mathsf{M}(r)) \subset O\tilde{~}(\delta_{\mathbf{d}} r)$, *with* $r \le 2(h_1 d_1 + \cdots + h_n d_n)$.

Hence, the lower the h_i the better. However, without putting extra assumptions on E_i, Corollary 1 only yields estimates of little interest. Even in sparse cases, it remains difficult to simplify the recurrence giving the coefficients h_i. Still, several examples in the next chapter will show that for some useful families, significantly sharper bounds can be derived.

The rest of this section is devoted to prove Proposition 8. In all that follows, the multi-degree $\mathbf{d} = (d_1, \ldots, d_n)$ and E_i are fixed. We also let E_i' be the set of modified exponents

$$E_i' = \{\nu - (0, \ldots, 0, d_i) \in \mathbb{Z}^i \mid \nu \in E_i\},$$

so that for all $\nu = (\nu_1, \ldots, \nu_i)$ in E_i', $\nu_j \ge 0$ for $j < i$ and $\nu_i < 0$. Hence, Equation (5.2) takes the (slightly more handy) form

$$h_i = \max_{\nu \in E_i'} \frac{h_1\nu_1 + \cdots + h_{i-1}\nu_{i-1} + 1}{-\nu_i}. \tag{5.3}$$

Recall that the function H_0 was defined in the previous subsection with domain $\mathbb{N}^n$; in what follows, we also consider H_0 as a function over $\mathbb{N}^i$, for $1 \leq i \leq n$, by defining $H_0(e_1, \dots, e_i) = H_0(e_1, \dots, e_i, 0, \dots, 0)$, where the right-hand expression contains $n - i$ zeros; for completeness, we write $H_0() = 0$ for $i = 0$. The following recurrence relation enables us to control the growth of H_0.

Lemma 2. *For $i \geq 1$, let $\mathbf{e} = (e_1, \dots, e_i)$ be in $\mathbb{N}^i$ and let $\mathbf{e}' = (e_1, \dots, e_{i-1})$ in $\mathbb{N}^{i-1}$. Then the following (in)equalities hold:*

$$H_0(\mathbf{e}) = H_0(\mathbf{e}') \quad \textit{if} \quad e_i < d_i, \qquad H_0(\mathbf{e}) \leq 1 + \max_{\nu \in E'_i} H_0(\mathbf{e} + \nu) \quad \textit{otherwise}.$$

Proof. Let us first suppose $e_i < d_i$; then,

$$X_1^{e_1} \cdots X_i^{e_i} \bmod \langle \mathbf{S} \rangle = (X_1^{e_1} \cdots X_{i-1}^{e_{i-1}} \bmod \langle \mathbf{S} \rangle) X_i^{e_i},$$

since the latter product is reduced modulo $\langle \mathbf{S} \rangle$. Both sides have thus the same degree in η, and our first claim follows.

We can now focus on the case $e_i \geq d_i$, for which we write $f_i = e_i - d_i$, so that $f_i \geq 0$. From Equation (5.1), we deduce

$$X_i^{f_i} S_i = X_i^{f_i + d_i} + \sum_{\nu \in E_i} s_\nu X_i^{f_i} \mathbf{X}_i^\nu,$$

and thus we get

$$X_i^{f_i} S_i = X_i^{e_i} + \sum_{\nu \in E'_i} s_\nu X_i^{e_i} \mathbf{X}_i^\nu,$$

by the definition of E'_i. In our notation, we have $X_1^{e_1} \cdots X_i^{e_i} = \mathbf{X}_i^{\mathbf{e}}$. Thus, after multiplication by $X_1^{e_1} \cdots X_{i-1}^{e_{i-1}}$ and term reorganization, the former equality implies that

$$\mathbf{X}_i^{\mathbf{e}} - X_1^{e_1} \cdots X_{i-1}^{e_{i-1}} X_i^{f_i} S_i = - \sum_{\nu \in E'_i} s_\nu \mathbf{X}_i^{\mathbf{e}+\nu}.$$

As a consequence, we deduce that

$$\deg(\mathbf{X}_i^{\mathbf{e}} \bmod \langle \mathbf{S} \rangle, \eta) \ \leq \ \max_{\nu \in E'_i} \deg(s_\nu \mathbf{X}_i^{\mathbf{e}+\nu} \bmod \langle \mathbf{S} \rangle, \eta).$$

Since for ν in E'_i, we have

$$\deg(s_\nu \mathbf{X}_i^{\mathbf{e}+\nu} \bmod \langle \mathbf{S} \rangle, \eta) = \deg(s_\nu, \eta) + \deg(\mathbf{X}_i^{\mathbf{e}+\nu} \bmod \langle \mathbf{S} \rangle, \eta) = 1 + H_0(\mathbf{e} + \nu),$$

the conclusion follows. □

Iterating the process of the previous lemma, we obtain the following bound. In the next lemma, $(f_\nu)_{\nu \in E'_i}$ are a family of integer valued variables.

Lemma 3. *Let* $\mathbf{e} = (e_1, \ldots, e_i)$ *be in* $\mathbb{N}^i$. *Then the following inequality holds:*

$$H_0(\mathbf{e}) \leq \max_{\substack{(f_\nu)_{\nu \in E'_i} \text{ non-negative integers} \\ \text{such that } 0 \leq e_i + \sum_{\nu \in E'_i} f_\nu \nu_i \leq d_i - 1}} H_0(\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu) + \sum_{\nu \in E'_i} f_\nu.$$

Proof. We prove the claim by induction on e_i. For $e_i \leq d_i - 1$, the family $(f_\nu = 0)_{\nu \in E'_i}$ satisfies the constraint $0 \leq e_i + \sum_{\nu \in E'_i} f_\nu \nu_i \leq d_i - 1$; for this choice, the value of the function we maximize is precisely $H_0(\mathbf{e})$, so our claim holds. Suppose now that $e_i \geq d_i$. Then, the previous lemma gives

$$H_0(\mathbf{e}) \leq 1 + \max_{\nu \in E'_i} H_0(\mathbf{e} + \nu). \tag{5.4}$$

Let us fix ν in E'_i; then $\mathbf{e} + \nu$ has non-negative integer coordinates, and its ith coordinate is less than e_i. Thus, we can apply the induction assumption, obtaining

$$H_0(\mathbf{e}+\nu) \leq \max_{\substack{(f_{\nu'})_{\nu' \in E'_i} \text{ non-negative integers} \\ \text{such that } 0 \leq e_i + \nu_i + \sum_{\nu' \in E'_i} f_{\nu'} \nu'_i \leq d_i - 1}} H_0(\mathbf{e}+\nu+\sum_{\nu' \in E'_i} f_{\nu'} \nu')+\sum_{\nu' \in E'_i} f_{\nu'}.$$

To any set of non-negative integers $(f_{\nu'})_{\nu' \in E'_i}$ with

$$0 \leq e_i + \nu_i + \sum_{\nu' \in E'_i} f_{\nu'} \nu'_i \leq d_i - 1$$

appearing in the previous maximum, we associate the non-negative integers $(f'_{\nu'})_{\nu' \in E'_i}$, with $f'_\nu = f_\nu + 1$ and $f'_{\nu'} = f_{\nu'}$ otherwise. These new integers satisfy

$$0 \leq e_i + \sum_{\nu' \in E'_i} f'_{\nu'} \nu'_i \leq d_i - 1$$

and

$$H_0(\mathbf{e} + \nu + \sum_{\nu' \in E'_i} f_{\nu'} \nu') + \sum_{\nu' \in E'_i} f_{\nu'} = H_0(\mathbf{e} + \sum_{\nu' \in E'_i} f'_{\nu'} \nu') + \sum_{\nu' \in E'_i} f'_{\nu'} - 1.$$

Taking maxima, we deduce from the previous inequality

$$H_0(\mathbf{e}+\nu) \leq \max_{\substack{(f'_{\nu'})_{\nu' \in E'_i} \text{ non-negative integers} \\ \text{such that } 0 \leq e_i + \sum_{\nu' \in E'_i} f'_{\nu'} \nu'_i \leq d_i - 1}} H_0(\mathbf{e} + \sum_{\nu' \in E'_i} f'_{\nu'} \nu') + \sum_{\nu' \in E'_i} f'_{\nu'} - 1.$$

Substituting in Equation (5.4) and taking the maximum over ν in E'_i concludes the proof. □

For $i \leq n$, let L_i be the linear form $(e_1, \ldots, e_i) \mapsto h_1 e_1 + \cdots + h_i e_i$, where the h_i are as in Equation (5.2). The following lemma concludes the proof of Proposition 8; as we did for H_0, for $i \leq n$, we extend H to $\mathbb{N}^i$, by writing $H(e_1, \ldots, e_i) = H(e_1, \ldots, e_i, 0, \ldots, 0)$.

Lemma 4. *For $i \leq n$ and $\mathbf{e} = (e_1, \ldots, e_i)$ in $\mathbb{N}^i$, the inequality $H(\mathbf{e}) \leq L_i(\mathbf{e})$ holds.*

Proof. It is sufficient to prove that $H_0(\mathbf{e}) \leq L_i(\mathbf{e})$ holds; since all coefficients of L_i are non-negative, L_i is non-decreasing with respect to all of its variables, which implies the thesis.

We prove our inequalities by induction on $i \geq 0$. For $i = 0$, we have $H_0() = L_0() = 0$; hence, our claim vacuously holds at this index. For $i \geq 1$, we now prove that if our inequality holds at index $i-1$, it will also hold at index i. Lemma 3 shows that for any $\mathbf{e} \in \mathbb{N}^i$, we have the inequality

$$H_0(\mathbf{e}) \leq \max_{\substack{(f_\nu)_{\nu \in E'_i} \text{ non-negative integers} \\ \text{such that } 0 \leq e_i + \sum_{\nu \in E'_i} f_\nu \nu_i \leq d_i - 1}} H_0(\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu) + \sum_{\nu \in E'_i} f_\nu.$$

Let φ be the natural projection $\mathbb{N}^i \to \mathbb{N}^{i-1}$, let $(f_\nu)_{\nu \in E'_i}$ be non-negative integers that satisfy the conditions in the previous inequality. Since $\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu$ has degree in X_i less than d_i, the first point of Lemma 2 shows that

$$H_0(\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu) = H_0(\varphi(\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu));$$

the induction assumption implies that this quantity is bounded from above by

$$L_{i-1}(\varphi(\mathbf{e} + \sum_{\nu \in E'_i} f_\nu \nu)).$$

As a consequence, $H_0(\mathbf{e})$ admits the upper bound

$$\max_{\substack{(f_\nu)_{\nu\in E_i'} \text{ non-negative integers} \\ \text{such that } 0 \le e_i + \sum_{\nu\in E_i'} f_\nu \nu_i \le d_i - 1}} L_{i-1}(\varphi(\mathbf{e} + \sum_{\nu\in E_i'} f_\nu \nu)) + \sum_{\nu\in E_i'} f_\nu.$$

This quantity itself is upper-bounded by a similar expression, where we allow the f_ν to be non-negative reals numbers; this gives

$$H_0(\mathbf{e}) \le \max_{\substack{(f_\nu)_{\nu\in E_i'} \text{ non-negative real numbers} \\ \text{such that } 0 \le e_i + \sum_{\nu\in E_i'} f_\nu \nu_i \le d_i - 1}} L_{i-1}(\varphi(\mathbf{e} + \sum_{\nu\in E_i'} f_\nu \nu)) + \sum_{\nu\in E_i'} f_\nu.$$

Since all h_i and all $\nu_1, \ldots, \nu_{i-1}$ are non-negative, the function of $(f_\nu)_{\nu\in E_i'}$ we want to maximize is affine with non-negative coefficients. The domain where we maximize it is defined by the conditions

$$f_\nu \ge 0 \text{ for all } \nu \in E_i', \quad 0 \le e_i + \sum_{\nu\in E_i'} f_\nu \nu_i \le d_i - 1,$$

and it is contained in the domain D defined by the conditions

$$f_\nu \ge 0 \text{ for all } \nu \in E_i', \quad 0 \le e_i + \sum_{\nu\in E_i'} f_\nu \nu_i.$$

Since all unknowns f_ν are non-negative, while the coefficients ν_i are negative, the domain D is convex and bounded. Hence, the maximal value we look for is upper-bounded by the maximal value at the end-vertices of D, distinct from the origin; these vertices are

$$E_\nu = \{f_{\nu'} = 0 \ \text{ for } \ \nu' \ne \nu, \qquad f_\nu = -\frac{e_i}{\nu_i}\}, \quad \text{for} \quad \nu \in E_i'.$$

At the point E_ν, the objective function takes the value

$$L_{i-1}(\varphi(\mathbf{e} - \frac{e_i}{\nu_i}\nu)) - \frac{e_i}{\nu_i}.$$

By the linearity of L_{i-1} and φ, this can be rewritten as

$$L_{i-1}(\varphi(\mathbf{e})) - L_{i-1}(\varphi(\frac{e_i}{\nu_i}\nu)) - \frac{e_i}{\nu_i} = L_{i-1}(\varphi(\mathbf{e})) - \frac{L_{i-1}(\varphi(\nu)) + 1}{\nu_i} e_i.$$

As a consequence, we obtain the upper bound

$$H_0(\mathbf{e}) \;\leq L_{i-1}(\varphi(\mathbf{e})) + \max_{\nu \in E_i'} \frac{L_{i-1}(\varphi(\nu)) + 1}{-\nu_i} e_i.$$

To simplify this further, note that the term $L_{i-1}(\varphi(\mathbf{e}))$ rewrites as $h_1 e_1 + \cdots + h_{i-1} e_{i-1}$. Similarly, $L_{i-1}(\varphi(\nu)) + 1$ equals $h_1 \nu_1 + \cdots + h_{i-1} \nu_{i-1}$. We deduce the inequality

$$H_0(\mathbf{e}) \;\leq\; h_1 e_1 + \cdots + h_{i-1} e_{i-1} \;+\; \max_{\nu \in E_i'} \frac{h_1 \nu_1 + \cdots + h_{i-1} \nu_{i-1} + 1}{-\nu_i} e_i,$$

which we can finally rewrite as

$$H_0(\mathbf{e}) \;\leq\; h_1 e_1 + \cdots + h_{i-1} e_{i-1} + h_i e_i,$$

as requested. □

Chapter 6

Families of application

In this chapter, we apply the general bounds obtained in the previous chapter to large families of examples; for several of them, we reach our goal of quasi-linear cost, whereas the previous results in the literature did not obtain so good estimates. The first section presents our families of examples; the latter sections give concrete illustrations.

6.1 Main family of examples

We consider triangular sets $\mathbf{T} = (T_1, \ldots, T_n)$ such that T_i has the form

$$T_i = X_i^{d_i} + \sum_{\nu \in D_i} t_\nu \mathbf{X}_i^\nu, \quad t_\nu \in \mathsf{R}, \tag{6.1}$$

where all $\mathbf{X}_i^\nu$ are monomials in $X_1, \ldots, X_i$ of total degree at most λ_i, for some $\lambda_i \in \mathbb{N}$. We let $d = \max_{i \leq n} d_i$, and we suppose that R contains at least d pairwise distinct values $x_1, \ldots, x_d$, with $x_i - x_j$ a unit for $i \neq j$.

Under these assumptions, we can estimate the cost of multiplication modulo the triangular set $\mathbf{T}$ by giving explicit estimates for the coefficients h_i of the previous chapter.

The following proposition illustrates three different situations. The first two cases display a cost quasi-linear in $d\delta_{\mathbf{d}}$, which is satisfying, especially for small d; the last one shows that small changes in the assumptions can induce large overheads. We will see in the next section cases where d_i is constant equal to d, or $d_i = n + 1 - i$; in such cases, d is logarithmic in $\delta_{\mathbf{d}}$ and the cost $O\tilde{}(d\delta_{\mathbf{d}})$ is thus $O\tilde{}(\delta_{\mathbf{d}})$, which is what we were aiming at.

Proposition 9. *With assumptions as above, multiplication modulo $\langle \mathbf{T} \rangle$ can be performed with the following complexities:*

$$\begin{array}{rcll} O\big(n\,\delta_{\mathsf{d}}\,\frac{\mathsf{C}(d)}{d}\,\mathsf{M}(nd)\big) & \subset & O\tilde{}(d\delta_{\mathsf{d}}) & \text{if } \lambda_i = d_i - 1 \text{ for all } i, \\ O\big(n\,\delta_{\mathsf{d}}\,\frac{\mathsf{C}(d)}{d}\,\mathsf{M}(n^2 d)\big) & \subset & O\tilde{}(d\delta_{\mathsf{d}}) & \text{if } \lambda_i = d_i \text{ for all } i, \\ O\big(n\,\delta_{\mathsf{d}}\,\frac{\mathsf{C}(d)}{d}\,\mathsf{M}(2^n d)\big) & \subset & O\tilde{}(2^n d\delta_{\mathsf{d}}) & \text{if } \lambda_i = d_i + 1 \text{ for all } i. \end{array}$$

Proof. First, we construct V: we simply choose the grid

$$V = [x_1, \dots, x_{d_1}] \times \cdots \times [x_1, \dots, x_{d_n}]. \tag{6.2}$$

Thus, we have $U_i = (X_i - x_1) \cdots (X_i - x_{d_i})$; as before we let $\mathbf{S} = \eta \mathbf{T} + (1-\eta)\mathbf{U}$. Thus, the monomial support E_i associated with S_i is contained in

$$D_i' = D_i \cup \{(0, \dots, 0, \nu_i) \mid 0 \le \nu_i < d_i\}.$$

Since each monomial in D_i has an exponent of the form $(\nu_1, \dots, \nu_i)$, with $\nu_1 + \cdots + \nu_i \le \lambda_i$ and $\nu_i < d_i$, we deduce from Equation (5.2) that

$$h_i \le \max_{\nu \in D_i'} \frac{h_1\nu_1 + \cdots + h_{i-1}\nu_{i-1} + 1}{d_i - \nu_i} \le \max\left(\max_{\nu \in D_i} \frac{h_1\nu_1 + \cdots + h_{i-1}\nu_{i-1} + 1}{d_i - \nu_i}, 1\right).$$

Let $h_i' = \max(h_1, \dots, h_i)$, so that

$$h_i \le \max\left(\max_{\nu \in D_i} \frac{h_{i-1}'(\nu_1 + \cdots + \nu_{i-1}) + 1}{d_i - \nu_i}, 1\right) \le \max\left(\max_{\nu \in D_i} \frac{h_{i-1}'(\lambda_i - \nu_i) + 1}{d_i - \nu_i}, 1\right). \tag{6.3}$$

Knowing the distribution of the d_i and λ_i, the former relation makes it possible to analyze the growth of the coefficients h_i, and thus of $2(d_1h_1 + \cdots + d_nh_n)$. In what follows, remember that d is the maximum of $d_1, \dots, d_n$.

Case 1. Suppose first that $\lambda_i = d_i - 1$. Then, the former inequality implies $h_i \le 1$ for all i, so that $2(d_1h_1 + \cdots + d_nh_n) \le 2nd$

Case 2. If $\lambda_i = d_i$, then (6.3) becomes $h_i \le h_{i-1}' + 1$, so that $h_i \le i$ for all i, and thus $2(d_1h_1 + \cdots + d_nh_n) \le 2(d_1 + 2d_2 + 3d_3 + \cdots + nd_n) \le n(n+1)d$.

Case 3. If finally $\lambda_i = d_i + 1$, then (6.3) becomes $h_i \le 2h_{i-1}' + 1$. We claim that in this case, $h_i \le 2^i - 1$. This is proved by induction

- *Basis:* For $i = 1$, we have $h_1 = 2^1 - 1 = 1$ which is true.

- *Induction:* Assume that $h_i \leq 2^i - 1$ is true for i; we have to prove that this is also true for $i+1$. Then, from $h_{i+1} \leq 2h'_i + 1$ we get $h_{i+1} \leq 2(2^i - 1) + 1 = 2^{i+1} - 1$, which is what we wanted.

Thus, in this case, we get $2(d_1h_1 + \cdots + d_nh_n) \leq 2(2^{n+1})d = 2^{n+2}d$.

To conclude the proof, we simply plug the previous estimates in the cost estimate $O(\delta_{\mathbf{d}}\mathsf{L}(\mathbf{d})\mathsf{M}(r))$ of Corollary 1, with $r \leq 2(d_1h_1 + \cdots + d_nh_n)$, and we use the upper bound $\mathsf{L}(\mathbf{d}) \leq n\mathsf{C}(d)/d$ of Equation (2.2). □

The bounds given in the proof of Proposition 9 are sharp and the complexities are better then the complexity of [23] which is $O\tilde{}(4^n\delta_{\mathbf{d}})$. Examples for first two cases of Proposition 9 are given in next sections where we will see that the bounds are actually sharp. For third case, we are not sure whether our bound is sharp or not.

6.2 Computing with Cauchy modules

In this section, we give an application coming from the study of polynomial systems with symmetries. The content of this section consists mainly of a review of known results; we will show how to improve one of those using the previous proposition. Here, the base ring R is actually $\mathbb{Q}$.

Definition. We consider polynomials in d variables $X_1, \ldots, X_d$. The elementary symmetric polynomials in the d variables $X_1, \ldots, X_d$ are written $E_k(X_1, \ldots, X_d)$, for $k = 0, 1, \ldots, d$. They are defined as

$$\begin{aligned} E_0(X_1,\ldots,X_d) &= 1 \\ E_1(X_1,\ldots,X_d) &= \sum_{1\leq j\leq d} X_j \\ E_2(X_1,\ldots,X_d) &= \sum_{1\leq j_1<j_2\leq d} X_{j_1}X_{j_2} \\ E_3(X_1,\ldots,X_d) &= \sum_{1\leq j_1<j_2<j_3\leq d} X_{j_1}X_{j_2}X_{j_3}; \end{aligned}$$

and so forth, down to

$$E_d(X_1,\ldots,X_d) = X_1X_2\cdots X_d.$$

Thus, there is one elementary symmetric polynomial of degree k in d variables for any $k \leq d$, and it is formed by adding together all distinct products of k distinct variables.

The following lists the elementary symmetric polynomials for the first three positive values of d. In every case, $E_0 = 1$ is also one of the polynomials.
For $d = 1$:

$$E_1(X_1) = X_1.$$

For $d = 2$:

$$\begin{aligned} E_1(X_1, X_2) &= X_1 + X_2 \\ E_2(X_1, X_2) &= X_1X_2 \end{aligned}$$

For $d = 3$:

$$\begin{aligned} E_1(X_1, X_2, X_3) &= X_1 + X_2 + X_3 \\ E_2(X_1, X_2, X_3) &= X_1X_2 + X_1X_3 + X_2X_3 \\ E_3(X_1, X_2, X_3) &= X_1X_2X_3 \end{aligned}$$

The elementary symmetric polynomials are the basic building block for symmetric polynomials: any symmetric polynomial P in $\mathbb{Q}[X_1, \ldots, X_d]$ can be expressed as a polynomial in the elementary symmetric polynomials, that is, there exists another polynomial Q such that $P(X_1, \ldots, X_d) = Q(E_1, \ldots, E_d)$.

Example. Consider the following polynomials, quoted from [10]:

$$\left| \begin{array}{l} P_1 = 1 - X_1(\alpha + X_2^2 + X_3^2) = 0 \\ P_2 = 1 - X_2(\alpha + X_3^2 + X_1^2) = 0 \\ P_3 = 1 - X_3(\alpha + X_1^2 + X_2^2) = 0 \end{array} \right. \tag{6.4}$$

where α is an independent parameter. In [10], it is shown how to use the symmetries of this system to solve it. The first step is to replace it by the following:

$$\left| \begin{array}{l} Q_1 = P_1 + P_2 + P_3 = 0 \\ Q_2 = P_1P_2 + P_2P_3 + P_3P_1 = 0 \\ Q_3 = P_1P_2P_3 = 0 \end{array} \right. \tag{6.5}$$

Each equation of this new system is symmetric: it remains unchanged through all permutations of X_1, X_2, X_3. Thus, every Q_i can be expressed in terms of

$$E_3 = X_1X_2X_3, \quad E_2 = X_1X_2 + X_2X_3 + X_1X_3, \quad E_1 = X_1 + X_2 + X_3.$$

For instance, we get

$$Q_1 = 3E_3 - (E_2 + \alpha)E_1 + 3 = 0.$$

This remark is the key used by [10] to solve the system: express all Q_i in terms of E_1, E_2, E_3 and solve in these new variables.

Computing Q. Given a symmetric polynomial P, our question here is how to compute the polynomial Q such that $P(X_1, \ldots, X_d) = Q(E_1, \ldots, E_d)$. We will do this using a basic construction in Galois theory and invariant theory [38, 31, 1, 30], Cauchy modules [31]. Let T_1 be the monic polynomial

$$T_1 = X_1^d - E_1X_1^{d-1} + E_2X_1^{d-2} + \cdots + (-1)^dE_d.$$

The next polynomials $T_2, \ldots, T_d$ are obtained by taking divided differences:

$$T_{i+1}(X_1, \ldots, X_{i+1}) = \frac{T_i(X_1, \ldots, X_{i-1}, X_i) - T_i(X_1, \ldots, X_{i-1}, X_{i+1})}{X_i - X_{i+1}} \qquad 1 \le i < d.$$

For instance, starting from $F = X^3 - E_1X^2 + E_2X - E_3$, we have

$$\left|\begin{array}{lcl} T_3(X_1, X_2, X_3) & = & \frac{T_2(X_1,X_2)-T_2(X_1,X_3)}{X_2-X_3} = X_3 + X_2 + X_1 - E_1 \\ T_2(X_1, X_2) & = & \frac{T_1(X_1)-T_1(X_2)}{X_1-X_2} = X_2^2 + X_2X_1 - E_1X_2 + X_1^2 - E_1X_1 + E_2 \\ T_1(X_1) & = & X_1^3 - E_1X_1^2 + E_2X_1 - E_3. \end{array}\right.$$

The polynomials $T_1, \ldots, T_d$ form a triangular set with coefficients in $\mathbb{Q}(E_1, \ldots, E_d)$, which has multi-degree $\mathbf{d} = (d, d-1, \ldots, 1)$, so that $\delta_{\mathbf{d}} = d!$.

It is proved in [38] that T_i has total degree at most $d+1-i$. Hence, we are under the assumptions of Subsection 6.1, with $\lambda_i = d_i = d+1-i$ for all i and $(x_1, \ldots, x_d) = (0, \ldots, d-1)$. As a consequence, Proposition 9 shows that multiplication modulo $\langle T_1, \ldots, T_d\rangle$ can be done using $O\big(d!\,\mathsf{C}(d)\,\mathsf{M}(d^3)\big)$ operations, that is, in quasi-linear time $O\tilde{}(d!)$. The previous known results of multiplication modulo $\langle T_1, \ldots, T_d\rangle$ were $O(4^d(d!)^2)$ in [15] and $O\tilde{}(4^dd!)$ in [23].

Let us now describe the applications of these polynomials. It is known (see e.g. [38, 15]) that if P is symmetric, its normal form with respect to $T_1, \ldots, T_d$ is the polynomial $Q(E_1, \ldots, E_d)$ we want to compute.

To measure the cost of computing Q, instead of considering that P is given to us in its expanded form, we should think that it is given by a *straight-line program*, that is, by a sequence of operations $(+, -, \times)$. Then, it is possible to obtain a similar sequence of operations that computes Q, by performing all operations for P modulo $\langle T_1, \ldots, T_d \rangle$. The following proposition gives the complexity of this process; it is a restatement of Theorem 1 in [15], but using the improved bounds $O\tilde{}(d!)$ obtained above.

Proposition 10. *Let P be in $\mathbb{Q}[X_1, \ldots, X_d]$ be a symmetric polynomial that can be computed using L operations of the form $(+, -, \times)$, and let Q be such that $P = Q(E_1, \ldots, E_d)$. Then, Q can be computed using $O\tilde{}(d!L)$ operations $(+, -, \times)$.*

6.3 Polynomial multiplication

Our next application is to give quasi-linear time algorithms for *univariate* multiplication in $\mathsf{R}[X]$ from our previous multivariate construction. Unfortunately, our algorithm does not improve on the complexity of Cantor-Kaltofen's algorithm [9]; however, we believe it is worth mentioning as the complexity is quasi linear.

Precisely, given $n \geq 1$, we give here an algorithm to perform truncated multiplication in $\mathsf{R}[X]/\langle X^{2^n} \rangle$. We introduce variables $X_1, \ldots, X_n$; computing in $A = \mathsf{R}[X]/\langle X^{2^n} \rangle$ is equivalent to computing in $B = \mathsf{R}[X_1, \ldots, X_n]/\langle V_1, \ldots, V_n \rangle$, with $\mathbf{V} = (V_1, \ldots, V_n)$ given by

$$\left| \begin{array}{l} X_1 - X_n^{2^{n-1}} \\ \vdots \\ X_{n-1} - X_n^2 \\ X_n^{2^n}, \end{array} \right.$$

since the dummy variables $X_1, \ldots, X_{n-1}$ play no role in this representation. However, changing the order of the variables, we see that the ideal $\langle V_1, \ldots, V_n \rangle$ is also equal to the ideal $\langle T_1, \ldots, T_n \rangle$ given by

$$\left| \begin{array}{l} X_n^2 - X_{n-1} \\ \vdots \\ X_2^2 - X_1 \\ X_1^2. \end{array} \right.$$

The R-basis of B corresponding to $\mathbf{V}$ is $(X_n^i)_{i<2^n}$; the basis corresponding to $\mathbf{T}$ is $M_\mathbf{d}$

(notation defined in the introduction), with $\mathbf{d} = (2, \ldots, 2)$. Besides, the change of basis does not use any arithmetic operation, since it amounts to rewrite the exponents i in base 2, and conversely.

Hence, we can apply our multivariate multiplication algorithm modulo $\langle \mathbf{T} \rangle$. Remark that the triangular set $\mathbf{T}$ satisfies the assumptions of Subsection 6.1 (for any R), with $d_1 = \cdots = d_n = 2$, $\delta_{\mathbf{d}} = 2^n$, $\lambda_1 = \cdots = \lambda_n = 1$ and $(x_1, x_2) = (0, 1)$. By Proposition 9, we deduce that the cost of a multiplication in B, and thus in A, is $O(2^n n \mathsf{M}(n))$. One can multiply univariate polynomials of degree 2^n using two multiplications in A. The first multiplication gives the lower part of the product whereas the second one gives the upper part (which can be achieved by reversing both polynomials and then performing the multiplication).This gives the recurrence

$$\mathsf{M}(2^n) \;\leq\; \mathsf{k}2^n n\mathsf{M}(n) \quad \text{and thus} \quad \mathsf{M}(d) \;\leq\; \mathsf{k}' d \log(d) \mathsf{M}(\log(d))$$

for some constants k, k' (for d not a power of 2, we let 2^n be first power of 2 greater than or equal to d).

Unrolling the recursion once, and taking $\mathsf{M}(n) \in O(n^2)$ to end the recursion, we obtain the quasi-linear estimate of the form

$$\mathsf{M}(d) \leq \mathsf{k}' d \log d (\log d)^2 \in O(d \log(d)^3)$$

Unrolling 2 times, we will have

$$\begin{aligned} \mathsf{M}(d) &\leq \mathsf{k}' d \log \mathsf{M}(\log d) \\ &\leq \mathsf{k}'^2 d \log(d) \log(d) \log\log(d) \mathsf{M}(\log\log(d)) \\ &\leq \mathsf{k}'^2 d \log(d) \log(d) \log\log(d) (\log\log(d))^2 \\ &\in O(d \log(d)^2 \log\log(d)^3) \end{aligned}$$

and so on.

The main noteworthy feature of this multiplication algorithm is that no root of unity is present, though our multivariate evaluation-interpolation routine is somewhat similar to a multivariate Fourier Transform. In particular, the case when 2 is not invertible in R requires no special treatment, contrary to [9].

6.4 Exponential generating series multiplication

We continue with a question somehow similar to the one in the previous subsection. Given two sequences $a_0, \dots, a_d$ and $b_0, \dots, b_d$ in R, we want to compute the sequence $c_0, \dots, c_d$ such that

$$c_k = \sum_{i+j=k} \binom{k}{i} a_i b_j, \tag{6.6}$$

where the binomial coefficients are the coefficients of the expansion of $(1+X)^k$ in $\mathsf{R}[X]$. We discuss an application of this question in the next chapter; note that the naive algorithm has cost $O(d^2)$.

Recall that if $1, \dots, d$ are invertible in R, the *exponential generating series* associated with a sequence $(a_0, \dots, a_d)$ is

$$\sum_{i=0}^{d} \frac{a_i}{i!} X^i;$$

note that exponential generating series are usually defined as infinite sums, but the finite sum we have here will be enough for us. If $1, \dots, d$ are invertible, Equations (6.6) is equivalent to the following:

$$\sum_{i \le d} \frac{c_i}{i!} X^i = \sum_{i \le d} \frac{a_i}{i!} X^i \sum_{i \le d} \frac{b_i}{i!} X^i \mod X^{d+1}. \tag{6.7}$$

In this case, we can achieve a cost $O(\mathsf{M}(d))$ to compute $c_0, \dots, c_d$. However, if the invertibility assumption does not hold, for instance over $\mathsf{R} = \mathbb{Z}/2\mathbb{Z}$ or more generally $\mathbb{Z}/2^\alpha\mathbb{Z}$, this approach fails, and it was unknown up to now how to compute the coefficients c_i efficiently.

Under some mild assumptions on R, we are going to see how to achieve a similar cost through multivariate computations. We will suppose that there exists a prime p such that for $a \in \mathbb{N}$, if $\gcd(a, p) = 1$, then a is invertible in R: this is the case e.g. for $\mathsf{R} = \mathbb{Z}/p^k\mathbb{Z}$. Let n be such that $d + 1 \le p^n$, and let us introduce the triangular set $\mathbf{T} = (T_1, \dots, T_n)$ defined by

$$\left|\begin{array}{l} X_n^p - pX_{n-1} \\ \vdots \\ X_2^p - pX_1 \\ X_1^p. \end{array}\right.$$

In what follows, for $i \geq 0$, $(i_0, i_1, \dots)$ denotes the sequence of its coefficients in base p, i.e. $i = i_0 + i_1 p + i_2 p^2 + \dots$; thus, for $i \leq d$, only $i_0, \dots, i_{n-1}$ can be non-zero.

Next, let v be the p-adic valuation: $v(a)$ is the largest integer e such that p^e divides a. We can then define the function $f : \mathbb{N} \to \mathbb{N}$ by

$$f(i) = \frac{i!}{p^{v(i!)}}.$$

This definition means that $f(i)$ is the factorial of i, where we removed all powers of p; thus, $f(i)$ is invertible in R.

Our main proposition shows that instead of using univariate exponential generating series, one can compute the coefficients $c_0, \dots, c_d$ using multivariate multiplication.

Proposition 11. *With a, b, c as above, let*

$$A = \sum_{i \leq d} \frac{a_i}{f(i)} X_n^{i_0} \cdots X_1^{i_{n-1}}, \quad B = \sum_{i \leq d} \frac{b_i}{f(i)} X_n^{i_0} \cdots X_1^{i_{n-1}}, \quad C = \sum_{i \leq d} \frac{c_i}{f(i)} X_n^{i_0} \cdots X_1^{i_{n-1}}.$$

Then $C = AB \bmod \langle \mathbf{T} \rangle$.

Before proving this proposition, let us explain its consequences. As in the previous subsection, we can apply our multivariate multiplication algorithm modulo $\langle \mathbf{T} \rangle$. Note that the triangular set $\mathbf{T}$ satisfies the assumptions of Subsection 6.1, with $d_1 = \cdots = d_n = p$, $\delta_{\mathbf{d}} = p^n$, $\lambda_1 = \cdots = \lambda_n = 1$ and $(x_1, \dots, x_p) = (0, \dots, p-1)$. Note as well that we can take $n \in O(\log_p(d))$, and that we have $\delta_{\mathbf{d}} \leq pd$.

By Proposition 9, the cost of computing C is $O\big(n \, \delta_{\mathbf{d}} \frac{\mathsf{C}(p)}{p} \mathsf{M}(np)\big)$; after a few simplifications, we obtain the bound

$$O(d \log(d) \, \mathsf{M}(p) \, \mathsf{M}(p \log_p(d))).$$

If p is fixed (e.g., $p = 2$) this can be simplified into $O(d \log(d) \mathsf{M}(\log(d)))$. This is not as good as the estimate $O(\mathsf{M}(d)) = O(d \log(d) \log\log(d))$ we obtained when $1, \dots, d$ are units in R, but quite close.

The rest of this section is devoted to prove this proposition. First, we need a few technical lemmas on the function f.

Lemma 5. *For i, j, k such that $i + j = k$, we have $\binom{k}{i} = \frac{f(k)}{f(i)f(j)} p^{v(\binom{k}{i})}$.*

Proof. The starting point is simply the definition of f, by $f(i) = i!/p^{v(i!)}$. Then, we can write

$$\begin{aligned}\binom{k}{i} &= \frac{k!}{i!j!}\\ &= \frac{\frac{k!}{p^{v(k!)}}}{\frac{i!}{p^{v(i!)}}\frac{j!}{p^{v(j!)}}}\frac{p^{v(k!)}}{p^{v(i!)}p^{v(j!)}}\\ &= \frac{f(k)}{f(i)f(j)}p^{v(k!)-v(i!)-v(j!)}.\end{aligned}$$

Now, the definition of the valuation v implies that $v(a/b) = v(a) - v(b)$ and $v(ab) = v(a) + v(b)$ for all a, b. This shows that the last quantity is equal to

$$\frac{f(k)}{f(i)f(j)}p^{v(\frac{k!}{i!j!})} = \frac{f(k)}{f(i)f(j)}p^{v(\binom{k}{i})},$$

which is the conclusion we wanted. $\square$

Lemma 6. *Let i, j, k be integers with $i + j = k$ and $k < p^n$, and write the expansions in base p*

$$i = i_0 + i_1p + \cdots + i_{n-1}p^{n-1}, \quad j = j_0 + j_1p + \cdots + j_{n-1}p^{n-1}$$

and

$$k = k_0 + k_1p + \cdots + k_{n-1}p^{n-1}.$$

Then we have

$$X_n^{i_0} \cdots X_1^{i_{n-1}} \times X_n^{j_0} \cdots X_1^{j_{n-1}} = p^{v(\binom{k}{i})}X_n^{k_0} \dots X_1^{k_{n-1}} \mod \langle \mathbf{T} \rangle.$$

Proof. Before reduction, the product is $m = X_n^{i_0+j_0} \dots X_1^{i_{n-1}+j_{n-1}}$. Reduction with respect to the triangular set $\mathbf{T}$ will raise the power of p; we need to follow the reduction process step-by-step to estimate by how much.

Observe that the equality $i + j = k$ implies the relation $i_0 + j_0 = q_0p + k_0$, where q_0 is the carry. Then, the first reduction of m with respect to the first polynomial of the triangular set, $X_n^p - pX_{n-1}$, will increase the exponents of p and X_{n-1} by q_0, and the exponent of X_n will be reduced to k_0.

As before, $i + j = k$ now implies that $i_1 + j_1 + q_0 = q_1p + k_1$, where q_1 is the new carry. At this stage, the exponent of X_{n-1} is $i_1 + j_1 + q_0$, and the reduction by the second polynomial of the triangular set, $X_{n-1}^p - pX_{n-2}$, increases the exponents of p and X_{n-2} by q_1, and leaves X_{n-1} with exponent k_1.

Continuing, we obtain the series of equalities that relate the coefficients i_ℓ, j_ℓ, k_ℓ:

$$\begin{aligned} i_0 + j_0 &= pq_0 + k_0 \\ i_1 + j_1 + q_0 &= pq_1 + k_1 \\ i_2 + j_2 + q_1 &= pq_2 + k_2 \\ &\vdots \\ i_{n-1} + j_{n-1} + q_{n-2} &= pq_{n-1} + k_{n-1}, \end{aligned}$$

with actually $q_{n-1} = 0$ since $i + j < p^n$. In parallel, as we have seen above, the exponent of p after reduction of m modulo $\langle \mathbf{T} \rangle$ is $q_0 + \cdots + q_{n-2}$; the exponent of X_n is k_0, that of X_{n-1} is k_1, $\ldots$, and that of X_1 is k_{n-1}.

We are thus almost done. To finish, we need to rewrite $q_0 + \cdots + q_{n-2}$ as in the statement of the lemma. From the above relations, we can deduce

$$\begin{aligned} & i_0 + \cdots + i_{n-1} + j_0 + \cdots + j_{n-1} = (q_0 + q_1 + \cdots + q_{n-2})(p-1) + k_0 + \cdots + k_{n-1} \\ \Rightarrow\ & i_0 + \cdots + i_{n-1} + j_0 + \cdots + j_{n-1} - (k_0 + \cdots + k_{n-1}) = (q_0 + q_1 + \cdots + q_{n-2})(p-1) \\ \Rightarrow\ & (q_0 + q_1 + \cdots + q_{n-2}) = \frac{(i_0 + \cdots + i_{n-1}) + (j_0 + \cdots + j_{n-1}) - (k_0 + \cdots + k_{n-1})}{(p-1)} \\ \Rightarrow\ & (q_0 + q_1 + \cdots + q_{n-2}) = v\Big(\binom{k}{i}\Big), \end{aligned}$$

where the last equation is [37, Eq. (1.6)]. The total increment of the power of p, in this reduction process, is thus $(q_0 + q_1 + \cdots + q_{n-1}) = v\big(\binom{k}{i}\big)$, as requested. □

Finally, we can prove our proposition. Let $i, j \leq d$, with $k = i + j \leq d$. From the previous lemma, the normal form of the product $\frac{a_i}{f(i)} X_n^{i_0} \cdots X_1^{i_{n-1}}$ by $\frac{b_j}{f(j)} X_n^{j_0} \cdots X_1^{j_{n-1}}$ modulo $\langle \mathbf{T} \rangle$ is

$$\frac{a_i b_j}{f(i) f(j)} p^{v\left(\binom{k}{i}\right)} X_n^{k_0} \cdots X_1^{k_{n-1}},$$

Thus, by Lemma 5, the former product equals

$$\binom{k}{i} \frac{a_i b_j}{f(k)} X_n^{k_0} \cdots X_1^{k_{n-1}}.$$

Summing over all i, j such that $i + j = k$ gives

$$\sum_{i+j=k} \frac{a_i}{f(i)} X_n^{i_0} \cdots X_1^{i_{n-1}} \frac{b_j}{f(j)} X_n^{j_0} \cdots X_1^{j_{n-1}} = \frac{c_k}{f(k)} X_n^{k_0} \cdots X_1^{k_{n-1}} \bmod \langle \mathbf{T} \rangle,$$

and summing over all k finishes the proof.

Chapter 7

Experimental Results

We finally present an application of the previous constructions to computation with algebraic numbers, and give timings of our implementation.

7.1 Presentation of the problem

Let k be a field and let f and g be monic polynomials in $k[T]$, of degrees m and n respectively. We are interested in computing their *composed sum* $h = f \oplus g$. This is the polynomial of degree $d = mn$ defined by

$$h = f \oplus g = \prod_{\alpha,\beta} (T - \alpha - \beta),$$

the product running over all the roots α of f and β of g, counted with multiplicities, in an algebraic closure $\overline{k}$ of k. For example, if $f = T^2 - 2$ and $g = T^2 - 3$, with coefficients in $\mathbb{Q}$, we have

$$h = (T - \sqrt{2} - \sqrt{3})(T - \sqrt{2} + \sqrt{3})(T + \sqrt{2} - \sqrt{3})(T + \sqrt{2} + \sqrt{3}),$$

which becomes after expansion

$$h = T^4 - 10T^2 + 1.$$

As suggested by this example, computing composed sums is a basic operation when one wants to do operations with algebraic numbers such as $\sqrt{2}$ or $\sqrt{3}$.

There is already a long series of previous work on this question; as we will see, our contribution is to complete these works by a study over fields k of the form $k = \mathbb{Z}/p\mathbb{Z}$, with p small (typically, $p = 2$).

A natural approach consists of computing $h(T)$ as the resultant of $f(T-U)$ and $g(U)$ in U. However, the fastest algorithm for resultants [29] has a complexity of order $O\tilde{}(d^{1.5})$ for $m = n$. To do better, Dvornicich and Traverso [12] suggested to compute the power sums

$$a_i = \sum_{f(\alpha)=0} \alpha^i, \quad b_i = \sum_{g(\beta)=0} \beta^i$$

of respectively f and g, and deduce the power sums c_i of h, which are given by the relation (already seen in the last chapter)

$$c_k = \sum_{i+j=k} \binom{k}{i} a_i b_j. \tag{7.1}$$

Finally, assuming that $1, \ldots, d$ are invertible in k, we can recover h.

In [5], this approach is shown to take time $O(\mathsf{M}(d))$, assuming that $1, \ldots, d$ are invertible in k. Indeed, computing $(a_i)_{i \le d}$ and $(b_i)_{i \le d}$ can be done in $O(\mathsf{M}(d))$ operations, over any field, using Newton iteration [32]. Then, by our assumption on the characteristic, one can compute $(c_i)_{i \le d}$ in quasi-linear time using the equation (that was already given in the last chapter)

$$\sum_{i \le d} \frac{c_i}{i!} X^i = \sum_{i \le d} \frac{a_i}{i!} X^i \sum_{i \le d} \frac{b_i}{i!} X^i \mod X^{d+1}. \tag{7.2}$$

for another $\mathsf{M}(d) + O(d)$ operations. Finally, knowing $(c_i)_{i \le d}$, one can then recover h in time $O(\mathsf{M}(d))$ as well, using fast exponential computation [8, 32, 40, 7]; this step relies as well on the assumption that $1, \ldots, d$ are invertible in k.

7.2 Our algorithm

In this section, we deal with the situation where our assumption on k fails, that is, $1, \ldots, d$ are not all invertible in k. Then, two issues arise: Equation (7.2) makes no sense anymore and $(c_i)_{i \le d}$ are actually not enough to recover h. To our knowledge, no general solution better than the resultant method was known up to now (partial answers are in [5, 35] under restrictive conditions). We propose here a solution that works over finite fields, following an idea introduced in [16].

For simplicity, we consider $k = \mathbb{Z}/p\mathbb{Z}$. Since our algorithm actually does computations over rings of the form $\mathbb{Z}/p^\alpha\mathbb{Z}$, measuring its complexity in operations in k as we did up to now is not appropriate: instead, we count bit operations. Thus, we let $\mathsf{M}_\mathbb{Z}$ be such that integers of bit-length ℓ can be multiplied using $\mathsf{M}_\mathbb{Z}(\ell)$ bit operations; quasi-linear estimates are known as well for $\mathsf{M}_\mathbb{Z}$, the best to date being Fürer's $\ell\log(\ell)2^{O(\log^*(\ell))}$ [13].

Proposition 12. *Given f and g, one can compute h using*

$$O\big((\mathsf{M}(d) + d\log(d)\,\mathsf{M}(p)\,\mathsf{M}(p\log_d(p)))\;\mathsf{N}(p,d)\big)$$

bit operations, with $\mathsf{N}(p,d) = O(\mathsf{M}_\mathbb{Z}(\log(p))\log(\log(p)) + \mathsf{M}_\mathbb{Z}(\log(d)))$.

This cost estimate is hard to read. After simplification, this cost is seen to be $O\tilde{}(dp^2)$ bit operations. Also, if we consider p fixed, such as $p = 2$, the cost becomes

$$O\big(\,(\mathsf{M}(d) + d\log(d)\mathsf{M}(\log(d)))\,\mathsf{M}_\mathbb{Z}(\log(d))\,\big),$$

which is indeed quasi-linear.

Proof. Let F and G be monic polynomials in $\mathbb{Z}[T]$ such that $f = F \bmod p$ and $g = G \bmod p$.

Let further $(A_i)_{i\geq 0}$, $(B_i)_{i\geq 0}$ and $(C_i)_{i\geq 0}$ be the power sums of respectively F, G and H. For *any* $\alpha \geq 0$, the reductions $A_i \bmod p^\alpha$, $B_i \bmod p^\alpha$, and $C_i \bmod p^\alpha$ satisfy Equation (7.1). In particular, given any α, we can apply the results of Subsection 6.4 to deduce $(C_i \bmod p^\alpha)_{i\leq d}$ from $(A_i \bmod p^\alpha)_{i\leq d}$ and $(B_i \bmod p^\alpha)_{i\leq d}$, by following the algorithm given in that section with coefficients in $\mathbb{Z}/p^\alpha\mathbb{Z}$.

It remains to choose a correct α. Taking $\alpha = \lfloor\log_p(d)\rfloor + 1$, it is proved in [6] that given $(C_i \bmod p^\alpha)_{i\leq d}$, one can compute h in quasi-linear time $O(\mathsf{M}(d)\mathsf{M}_\mathbb{Z}(\log_p(d)))$ bit operations. Remark that this step is non trivial: recovering a polynomial of degree d from its Newton sums requires divisions by $1, \dots, d$, and not all these numbers are invertible modulo p if p is too small.

These ingredients are sufficient to design our algorithm. We use the following subroutines:

- the function Lift simply lifts its argument from $\mathbb{Z}/p\mathbb{Z}[T]$ to $\mathbb{Z}/p^\alpha\mathbb{Z}[T]$;
- the function $\mathsf{PowerSums}$ computes the first d power sums of its argument: it is detailed in [5] and originates from [32];

- the function ExponentialGeneratingSeriesMultiplication applies the algorithm of Subsection 6.4;
- the function PowerSumsToPolynomial recovers h in $\mathbb{Z}/p\mathbb{Z}$ starting from the power sums $(C_i)_{i\leq d}$, that are in $\mathbb{Z}/p^\alpha\mathbb{Z}$: it is taken from [6].

Our choice of α implies that $\log(p^\alpha) = O(\log(d))$. Thus, operations $(+, \times)$ modulo p^α take $O(\mathsf{M}_\mathbb{Z}(\log(d)))$ bit operations [14, Chapter 9]. The cost of inversions modulo p^α is slightly higher. We denote it by $\mathsf{N}(p,d)$; an upper bound on this quantity is given in Lemma 7 below.

The cost of computing $(A_i)_{i\leq d}$ and $(B_i)_{i\leq d}$ is $O(\mathsf{M}(d))$ operations modulo p^α. The next cost is that of computing $(C_i)_{i\leq d}$, which is reported in Subsection 6.4 in terms of numbers of operations modulo p^α. The final part is the cost of recovering h; it is given in [6] and is negligible compared to the previous costs. Summing these costs, and using the estimate on $\mathsf{N}(p,d)$ in Lemma 7, we conclude the proof. □

Lemma 7. *Inversion modulo p^α takes $\mathsf{N}(p,d) = O(\mathsf{M}_\mathbb{Z}(\log(p))\log(\log(p)) + \mathsf{M}_\mathbb{Z}(\log(d)))$ bit operations.*

Proof. We use Newton iteration to do the inversion: to invert a modulo p^α, we first invert it modulo p, then modulo the powers of p, to finally reach the inverse modulo p^α. In the complexity estimate, the first term stands for the cost computing the inverse modulo p: from [14, Corollary 11.10], the cost of inversion of an integer in $\mathbb{Z}_p$ is $O(\mathsf{M}_\mathbb{Z}(\log(p))\log(\log(p))$. Lifting it to p^α costs $O(\mathsf{M}_\mathbb{Z}(\log(d))$ [14, Chapter 9]. □

Algorithm 8 ComposedSum(f, g)

```
1: $d \leftarrow \deg(f)\deg(g)$
2: $\alpha \leftarrow \lfloor \log_p(d) \rfloor + 1$
3: $F \leftarrow \mathsf{Lift}(f, \alpha)$
4: $(A_i)_{i\leq d} \leftarrow \mathsf{PowerSums}(F, d)$
5: $G \leftarrow \mathsf{Lift}(g, \alpha)$
6: $(B_i)_{i\leq d} \leftarrow \mathsf{PowerSums}(G, d)$
7: $(C_i)_{i\leq d} \leftarrow \mathsf{ExponentialGeneratingSeriesMultiplication}((A_i)_{i\leq d}, (B_i)_{i\leq d})$
8: return $\mathsf{PowerSumsToPolynomial}((C_i)_{i\leq d})$
```

7.3 Experimental results

We implemented the composed sum algorithm over $\mathbb{Z}/2\mathbb{Z}$ (*i.e.*, $p = 2$ here). We used the NTL C++ package as a basis [36]. Since NTL does not implement bivariate resultants, we also used Magma [3] for comparison with the resultant method. All timings are obtained on an AMD Athlon 64 with 5GB of RAM.

Figure 7.1 gives detailed timings for our algorithm; each colored area gives the time of one of the main tasks. The less costly step is the first, the conversion from the original polynomials to their Newton sums. Then, we give the time needed to compute all the power series roots needed for our multiplication algorithm, followed by the evaluation-interpolation process itself; finally, we give the time necessary to recover h from its power sums. Altogether, the practical behavior of our algorithm matches the quasi-linear complexity estimates. The steps we observe correspond to the increase in the number of variables in our multivariate polynomials, and are the analogues of the steps observed in classical FFT.

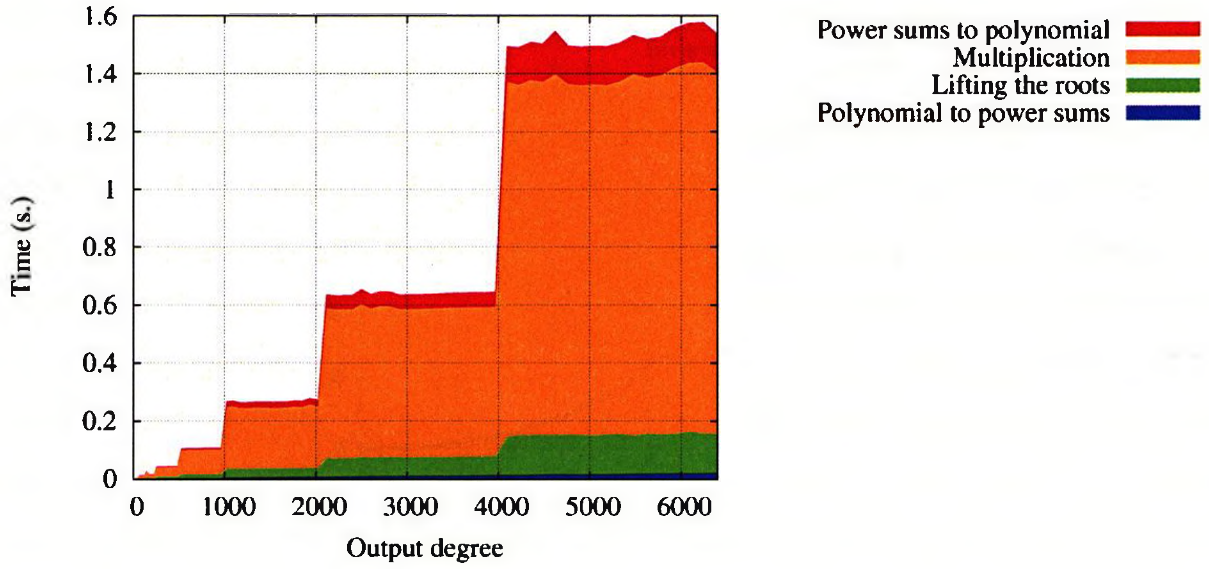


Figure 7.1: Detailed timings for our algorithm

Figure 7.2 gives timings obtained in Magma, using the built-in resultant function, on the same set of problems as above. As predicted by the complexity analysis, the results are significantly slower (about two orders of magnitude for the larger problems).

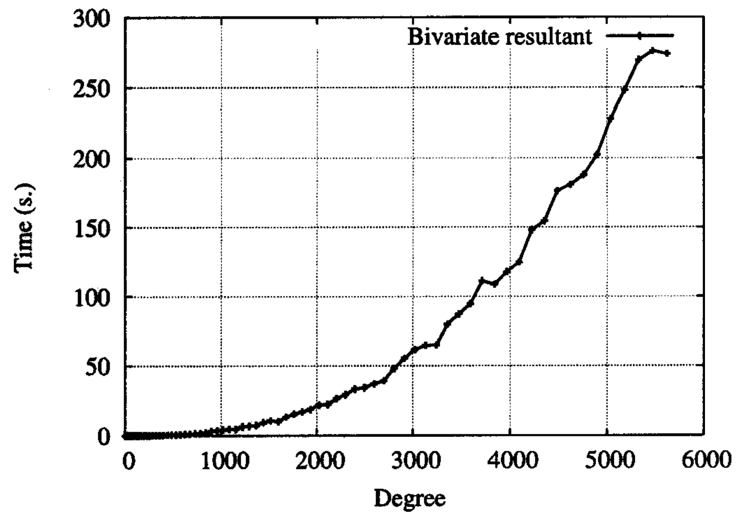


Figure 7.2: Timings in magma

Chapter 8

Conclusions and future work

The homotopy techniques for triangular multiplication give us quasi linear time complexity for some cases such as multivariate polynomial multiplication, exponential generating series multiplication, computation with Cauchy modules. The comparison of this complexity with other algorithms [15, 23] of multiplication modulo triangular sets tells us that this is the best complexity. Besides, the polynomials structure of the triangular set determines the complexity of the algorithm.

Several questions remain open after this work. Of course, the most challenging one remains how to unconditionally get rid of all exponential factors in multiplication algorithms for triangular sets. More immediate questions may be the following: at the fine tuning level, adapting the idea of the Truncated Fourier Transform [39] should enable us to reduce the step effect in the timings of the previous chapter. Besides, it will be worthwhile to investigate what other applications can be dealt with using the "homotopy multiplication" model, such as the product of matrices with entries defined modulo a triangular set, or further tasks such as modular inversion or modular composition.

Bibliography

[1] I. Abdeljaouad, S. Orange, G. Renault, and A. Valibouze. Computation of the decomposition group of a triangular ideal. *Applicable Algebra in Engineering Communication and Computing*, 15(3-4):279–294, 2004.

[2] P. Aubry and A. Valibouze. Using Galois ideals for computing relative resolvents. *J. Symb. Comp.*, 30(6):635–651, 2000.

[3] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symb. Comp.*, 24(3-4):235–265, 1997.

[4] A. Bostan, M.F.I. Chowdhury, J. van der Hoeven, and É. Schost. Homotopy methods for multiplication modulo triangular sets. Technical Report `http://arxiv.org/abs/0901.3657v1`, Arxiv, 2009.

[5] A. Bostan, P. Flajolet, B. Salvy, and É. Schost. Fast computation of special resultants. *J. Symb. Comp.*, 41(1):1–29, 2006.

[6] A. Bostan, L. González-Vega, H. Perdry, and É. Schost. From Newton sums to coefficients: complexity issues in characteristic p. In *MEGA'05*, 2005.

[7] A. Bostan and É. Schost. A simple and fast algorithm for computing exponentials of power series. Available at `http://algo.inria.fr/bostan/`, 2008.

[8] R. P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity*, pages 151–176. Academic Press, 1976.

[9] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[10] A. Colin. Solving a system of algebraic equations with symmetries. *Journal of Pure and Applied Algebra*, 117-118:195–215, 1997.

[11] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM, 2005.

[12] R. Dvornicich and C. Traverso. Newton symmetric functions and the arithmetic of algebraically closed fields. In *AAECC-5*, volume 356 of *LNCS*, pages 216–224. Springer, 1989.

[13] M. Fürer. Faster integer multiplication. In *39th Annual ACM Symp. Theory Comp.*, pages 57–66. ACM, 2007.

[14] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 1999.

[15] P. Gaudry, É. Schost, and N. Thiéry. Evaluation properties of symmetric polynomials. *International Journal of Algebra and Computation*, 16(3):505–523, 2006.

[16] L. González-Vega and H. Perdry. Computing with Newton sums in small characteristic. In *EACA'04*, 2004.

[17] M. Kalkbrener. A generalized euclidean algorithm for computing triangular representation of algebraic varieties. *J. Symb. Comp.*, 15(2):143–167, 1993.

[18] L. Langemyr. Algorithms for a multiple algebraic extension. In *Effective methods in algebraic geometry)*, volume 94 of *Progr. Math.*, pages 235–248. Birkhäuser, 1991.

[19] D. Lazard. A new method for solvong algebraic systems of positive dimension. *Disc. Appl. Math.*, 33:147–160, 1991.

[20] D. Lazard. Solving zero-dimensional algebraic systems. *J. Symb. Comp.*, 15:117–133, 1992.

[21] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[22] X. Li, M. Moreno Maza, R. Rasheed, and É Schost. High-performance symbolic computation in a hybrid compiled-interpreted programming environment. In *ICCSA'08*, pages 331–341. IEEE, 2008.

[23] X. Li, M. Moreno Maza, and É. Schost. Fast arithmetic for triangular sets: from theory to practice. In *ISSAC'07*, pages 269–276. ACM, 2007.

[24] Marc Moreno Maza Li Xin and Wei Pan. Computations modulo regular chains. In *ISSAC'09*, pages 151–160. ACM, 2009.

[25] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *ISSAC'02*, pages 109–116. ACM, 2002.

[26] M. Moreno Maza. On triangular decompositions of algebraic varieties. In MEGA-2000, number TR 4/99, Oxford, UK, 2000. `http://www.csd.uwo.ca/~moreno/`.

[27] D. Lazard P. Aubry and M. Moreno Maza. On the theories of triangular sets. *J. of symbolic computation*, 28(1,2):45–124, 1999.

[28] V. Y. Pan. Simple multivariate polynomial multiplication. *J. Symb. Comp.*, 18(3):183–186, 1994.

[29] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC'97*, pages 233–240. ACM, 1997.

[30] G. Renault and K. Yokoyama. A modular algorithm for computing the splitting field of a polynomial. In *Algorithmic Number Theory, ANTS VII*, number 4076 in LNCS, pages 124–140. Springer, 2006.

[31] N. Rennert and A. Valibouze. Calcul de résolvantes avec les modules de Cauchy. *Experimental Mathematics*, 8(4):351–366, 1999.

[32] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Univ. Tübingen, 1982.

[33] É. Schost. Complexity results for triangular sets. *Journal of Symbolic Computation*, 36(3–4):555–594, 2003.

[34] É. Schost. Computing parametric geometric resolutions. *Applicable Algebra in Engineering, Communication and Computing*, 13(5):349–393, 2003.

[35] É. Schost. Multivariate power series multiplication. In *ISSAC'05*, pages 293–300. ACM, 2005.

[36] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net`.

[37] A. Straub, T. Amdeberhan, and V. H. Moll. The p-adic valuation of k-central binomial coefficient, 2008.

[38] B. Sturmfels. *Algorithms in invariant theory.* Texts and Monographs in Symbolic Computation. Springer-Verlag, 1993.

[39] J. van der Hoeven. The Truncated Fourier Transform and applications. In *ISSAC'04*, pages 290–296. ACM, 2004.

[40] J. van der Hoeven. Newton's method and FFT trading. Technical Report 2006-17, Univ. Paris-Sud, 2006. Submitted to J. Symb. Comp.

[41] W. T. Wu. A zero structure theorem for polynomial equations solving. *MM Research Preprints*, 1:2–12, 1987.